# The 2025 Google Code Golf Championship

**Michael D. Moffitt** MOFFITT@GOOGLE.COM
**Divy Thakkar** DTHAKKAR@GOOGLE.COM
**Ryan Burnell** RBURNELL@GOOGLE.COM
**Orhan Firat** ORHANF@GOOGLE.COM
**Walter Reade** INVERSION@GOOGLE.COM
**Sohier Dane** SOHIER@GOOGLE.COM
**Addison Howard** ADDISONHOWARD@GOOGLE.COM
*Competition Organizers*

**Lucas Deutschmann** LUCAS.DEUTSCHMANN@RPTU.DE
**Luke Gustafson** LUKEG3@GMAIL.COM
**Adam Kues** AK@AKORLITH.COM
**Mukundan Senthilkumar** MUKUNDAN314@GMAIL.COM
*Team: Code Golf International (1st Place)*

**Nicolas Posada** POSADANICO@OUTLOOK.COM
**Lyndon Ho** HLYNDON20@GMAIL.COM
**Larry Wu** WUPIR2@GMAIL.COM
**Lav Patel** SUSPICIOUS.PMLR.EMAIL@GMAIL.COM
*Team: jailctf merger (2nd Place)*

**Hunter Richardson** HUNTERTR2002@GMAIL.COM
**Wei Heng Kuan** KUANWH1996@GMAIL.COM
*Team: ox jam! (3rd Place)*

**Editors:** Tao Qin, Kun Zhang, and Jes Frellsen

## Abstract

For over half a decade, the Abstraction and Reasoning Corpus has played a critical role in evaluating the fluid intelligence and skill acquisition efficiency of machine learning systems. While the dataset has proven especially useful in advancing program synthesis, few investments have been made in distilling the transformational logic of these tasks into their most irreducible computational forms. In conjunction with the NeurIPS 2025 Competition Track, we organized the *2025 Google Code Golf Championship*, a three-month online challenge designed to elicit minimal Python solutions for all training puzzles in the seminal ARC-AGI-1 benchmark suite. Sponsored by Google DeepMind and funded with $100,000 in prizes, the contest attracted *over 1,000 teams* and yielded tens of thousands of verified, open-source programs that collectively establish tight upper bounds on the Kolmogorov complexity of each task. In this paper, we offer a comprehensive retrospective of the event, including an overview of the contest format, a survey of approaches taken by top teams, and a discussion of potential implications for the future design of intelligent agents.

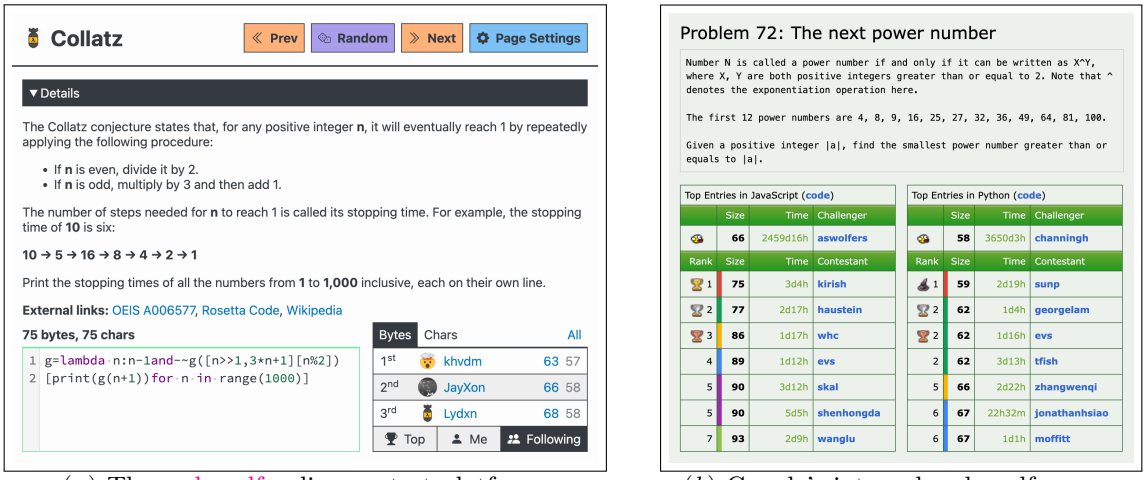
© 2026 M.D. Moffitt *et al.*

## 1. Introduction

The Abstraction and Reasoning Corpus (ARC-AGI) has established itself as a cornerstone benchmark for measuring fluid intelligence and generalization in machine learning systems (Chollet, 2019). Unlike traditional datasets that reward pattern matching via manifold interpolation, ARC-AGI is a few-shot problem-solving suite designed to test an agent's ability to efficiently acquire new skills when presented with novel, previously unseen tasks. By focusing on core knowledge priors — e.g., translation, symmetry, counting, etc. — rather than semantic knowledge, the benchmark provides a unique litmus test for evaluating progress toward achieving general intelligence. Its rigor lies in its refusal to allow memorization, as a successful system cannot merely recall data, but must instead actively reason about the underlying generative process demonstrated in each puzzle.

Among the most promising approaches to solving ARC is the application of deep learning-guided program synthesis, where neural networks direct the search for symbolic programs that satisfy the task examples (Alford et al., 2022; Ainooson et al., 2023; Neumann and Pintér, 2023; Xu et al., 2023; Bober-Irizar and Banerjee, 2024; Lei et al., 2024; Ouellette, 2024; Akyürek et al., 2025; Hocquette and Cropper, 2025; Li et al., 2025; Liao and Gu, 2025; Macfarlane and Bonnet, 2025; Pourcel et al., 2025; Rocha et al., 2025; Witt et al., 2025). Yet, compared to conventional software engineering patterns (such as those found in massive pre-training corpora like GitHub), the puzzles embodied by ARC represent a significant distributional shift, with the visual-spatial logic required to manipulate grids having relatively little overlap with standard coding benchmarks. Compounding this challenge is the fact that each transformation in ARC is defined only implicitly as input-output image pairs: given that no canonical set of source codes is bundled with the benchmark suite to serve as ground truth, this absence represents a critical gap for researchers. Without access to verified, human-authored logic adapted to this domain, it remains difficult to effectively fine-tune agents or to evaluate the efficiency of the programs they generate (Chen et al., 2021; Du et al., 2024; Luo et al., 2024; Wei et al., 2024; Zhu et al., 2024; Qiu et al., 2025).

To bridge this gap, we adopted a competitive incentive structure similar in spirit to the ARC Prize (Chollet et al., 2024, 2026) by organizing the *2025 Google Code Golf Championship* in conjunction with the NeurIPS 2025 Competition Track. Specifically, we framed the creation of reference implementations as an instance of *code golf* — a decades-old recreational programming pursuit in which participants strive to solve a particular problem using the absolute minimum number of bytes or characters (see Figure 1 for two other examples). This format naturally aligns with the theoretical principle of Minimum Description Length (MDL), effectively leveraging the optimization pressure of the game to establish tight upper bounds on the Kolmogorov complexity of each task (Solomonoff, 1964; Kolmogorov, 1965; Chaitin, 1966; Rissanen, 1978; Barron et al., 1998; Grünwald, 2004; Li and Vitányi, 2019). Crucially, our competition remained agnostic regarding the nature of the participating intelligence: we invited solutions derived from purely human intuition, automated program synthesis, or hybrid human-AI workflows. These open parameters elicited exceptional algorithmic creativity, requiring teams to identify and exploit structural regularities to shave bytes from their submissions. By channeling this collective ingenuity, the championship succeeded in crowdsourcing highly compressed Python solutions for the entire ARC-AGI-1 training set.




(*a*) The code.golf online contest platform. (*b*) Google's internal code golf server.

Figure 1: Screenshots illustrating the long-standing culture of competitive code minimization, spanning (*a*) the well-established code.golf public platform active since 2017, and (*b*) an internal server used by Google engineers since 2014.

The result is a threefold contribution to the scientific community:

- A novel set of minimal reference programs from winning teams that faithfully approximate the true complexity of the underlying task mechanics.
- A massive corpus of verified solutions produced by dozens of other participating teams that, while not necessarily minimal, nevertheless represent an earnest attempt at optimizing lexical density, capturing a wide variety of algorithmic approaches.
- A rich repository of community-generated resources, including open-source validation tools and qualitative "golfer's guides" that explicitly codify the strategies used to compress the abstract priors of ARC-AGI into minimal Python logic.

## 2. Contest Format

Our contest was administered by Kaggle, an online community and competition platform for machine learning practitioners and data scientists. The site allows users to collaborate, compete, and access a vast library of public datasets, code, and tutorials. Over the past decade, it has hosted hundreds of high-profile competitions (including the original Abstraction and Reasoning Challenge, along with both the 2024 and 2025 editions of the ARC Prize)[1,2,3] and has become a crucial hub for developing skills, networking, and advancing the field of machine learning.

1. https://kaggle.com/competitions/abstraction-and-reasoning-challenge
2. https://kaggle.com/competitions/arc-prize-2024
3. https://kaggle.com/competitions/arc-prize-2025

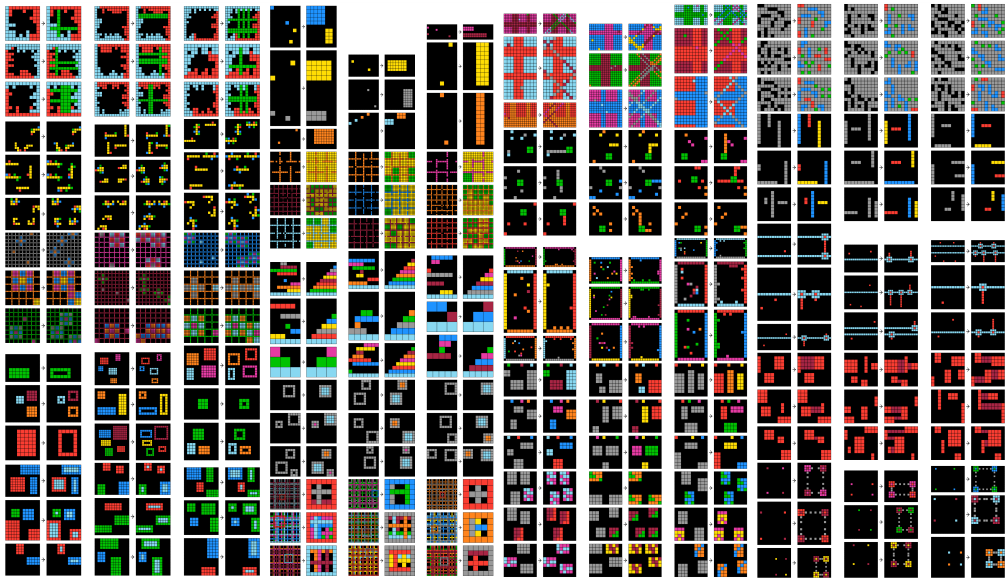


Figure 2: A subset of input/output pairs taken from the ARC-GEN-100K dataset.

### 2.1. Evaluation

For each task in the ARC-AGI-1 training benchmark suite, each team earned a score of $\max(1, 2500 - \textit{length})$ points for a functionally correct Python program using *length* characters (or more specifically, bytes) in its definition. Any functionally incorrect program — i.e., that produced an inconsistent outcome for some $\langle \textit{input}, \textit{output} \rangle$ pair — earned 0.001 points, allowing teams to deduce the overall number of failures from their total score alone. We believe this scoring system offered several key advantages:

- It was extremely *easy* for both participants and contest organizers to measure.
- It was entirely *objective*, leaving very little room for error or misinterpretation.
- It encouraged *short* programs likely to be most useful in establishing a canonical reference suite.

### 2.2. Datasets

To establish the functional correctness of a submission, we employed two separate validation datasets. The first is the set of demonstration exemplars included in the official ARC-AGI-1 release, which contains between three and five image pairs for the majority of tasks.[4] To disincentivize the creation of hardcoded solutions that overfit to these specific input/output grids, we incorporated elements from a second suite containing 100,000 additional pairs produced by ARC-GEN, a mimetic procedural benchmark generator designed to mirror the original distribution of examples as faithfully as possible (Moffitt, 2025).[5,6] Refer to Figure 2 for a visual gallery of several such instances.

4. https://github.com/fchollet/ARC-AGI
5. https://github.com/google/ARC-GEN
6. https://kaggle.com/datasets/arcgen100k/the-arc-gen-100k-dataset

This starter notebook for the 2025 Google Code Golf Championship is designed to help contestants verify the functional correctness of their programs. It allows one to load example pairs for a task, visualize them, and test whether a candidate code snippet produces expected results across all competition benchmarks.

Enter a task number (between 1 and 400):

```
task_num = 0  # Task 0 is just an illustrative example (and not eligible for points)
```

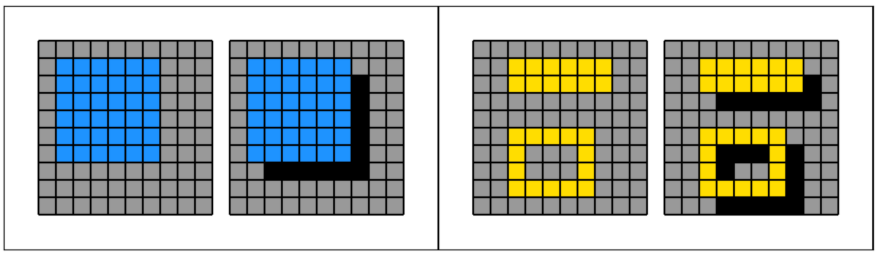
Color legend:

```
import sys
sys.path.append("/kaggle/input/google-code-golf-2025/code_golf_utils")
from code_golf_utils import *
show_legend()
```




Example <input, output> pairs:

```
examples = load_examples(task_num)
show_examples(examples['train'] + examples['test'])
```



Enter your code to solve the task:

```
%%writefile task.py
def p(g):
 for r, row in enumerate(g):
  for c, color in enumerate(row):
   if r and c and color==5 and g[r-1][c-1] not in [0,5]: g[r][c]=0
 return g
```

```
Writing task.py
```

Verify your code:

```
verify_program(task_num, examples)
```

```
Results on ARC-AGI examples: 2 pass, 0 fail
Results on ARC-GEN examples: 1 pass, 0 fail

Your code IS READY for submission!
Its length appears to be 150 bytes.
Next steps:
 * Copy it into a file named task000.py on your local machine.
 * Create a zip file containing that program along with all others.
 * Submit that zip file to the Kaggle competition so that it can be officially scored.
```

Figure 3: The official competition starter notebook.

| Notebook Title | Author | Score | Link |
|---|---|---|---|
| A bit more of Code Golf: 255/400 + visualization | jacekwl | 578,523.147 | [link] |
| Qwen2.5 32B + ARC + Local Score | bibanh | 640,225.123 | [link] |
| Code Golf - Ensemble - Local Score - 391/400 DSL | seshurajup | 677,725.013 | [link] |
| Oh Barnacles | jazivxt | 737,616.018 | [link] |
| Road to 400 collaboration | mbilichenko | 819,231.002 | [link] |
| python-minifier applied | vladislavlassa | 851,678.000 | [link] |
| Big Zippa | cheeseexports | 877,446.000 | [link] |
| Solutions for all 400 Tasks + Ensemble | kerta27 | 899,523.000 | [link] |
| GCGC playground | jonathanchan | 912,484.000 | [link] |
| Google Code Golf Championship 101 | adilshamim8 | 913,211.000 | [link] |
| Google Code Golf Community Best | tonylica | 913,449.000 | [link] |
| *Compilation of winning solutions** | garrymoss | 963,258.000 | [link] |

**post-competition notebook*

Table 1: Notable notebooks.

### 2.3. Timeline

- June 27th, 2025 — Announcement of accepted NeurIPS 2025 Competitions.[7]
- June 27th, 2025 — Contest pre-registration begins.[8]
- July 31st, 2025 — Start date and launch of the Kaggle site.[9]
- October 23rd, 2025 — Entry deadline & team merger deadline.
- October 30th, 2025 — Final submission deadline.

## 3. Notable Notebooks

Contestants were supplied with a starter notebook (see Figure 3) designed to assist with the construction and verification of programs.[10] This interactive script allowed participants to load examples for a task, visualize them, and develop the corresponding Python solution. Teams would then create a zip file containing all such programs and submit this for official scoring and placement on a live leaderboard (see Appendix A). Throughout the competition, users also had an opportunity to create and share their own notebooks, resulting in **over 140 public notebooks** of which twelve are highlighted in Table 1. Several tricks and techniques revealed in these codes made their way into the submissions of top-scoring teams.

7. https://blog.neurips.cc/2025/06/27/neurips-2025-competitions-announced/
8. https://sites.google.com/corp/view/neurips-2025-code-golf
9. https://kaggle.com/competitions/google-code-golf-2025
10. https://kaggle.com/code/mmoffitt/neurips-2025-google-code-golf-championship

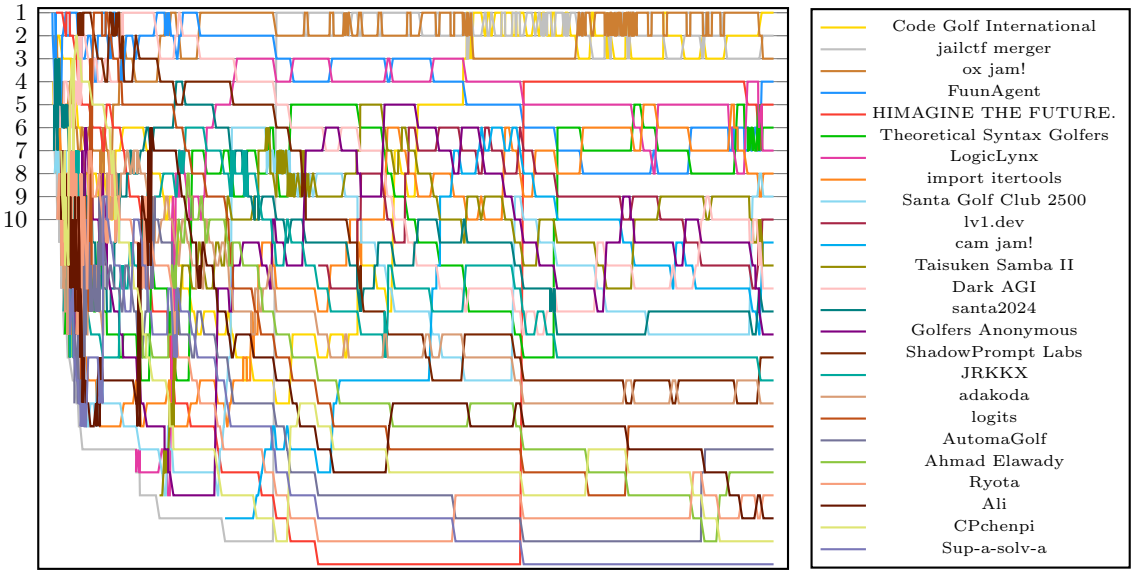



Figure 4: A time series of team ranks.

## 4. Results

Our competition attracted a total of **6,306** entrants, yielding **41,314** submissions from **1,142 teams**. In Figure 4, we present a time series of ranks across all teams who placed within the top ten at any point following the first week of the contest. At the end of the competition, the highest-ranking team had earned a total of 962,070 points, while the aggregate "super-score" — calculated by taking the best solutions discovered among all participating teams — reached an astonishing **963,226** points, translating to an average solution length of approximately **92 bytes per task** (less than half that of any public notebook). Complete statistics and codes from the shortest programs are provided in Appendix D.

## 5. Team Strategies

Full descriptions of the strategies employed by the 1st, 2nd, and 3rd place teams are included in Appendix B. Given that ARC-AGI was explicitly designed to resist conventionally-trained AI systems, it may come as little surprise that no fully-automated end-to-end pipeline emerged from the contest. Yet — perhaps remarkably — none of these top three teams employed *any* form of Artificial Intelligence in the production of their codes, instead relying exclusively upon (a) their personal expertise in manual golfing, and (b) a combination of bespoke utilities for code minimization. As shown in Figure 5, this helped place considerable distance between their total byte counts and those of other winning teams (whose approaches are summarized in the sections below). In Table 2, we provide a link to each team's post-competition solution write-up, along with their final score and a distribution of their "golf medals".[11]

11. In code golf vernacular, a {*gold*, *silver*, *bronze*} medal is earned for every {*shortest*, *second-shortest*, *third-shortest*} solution (respectively), and a *diamond* if no other team matched the shortest solution.

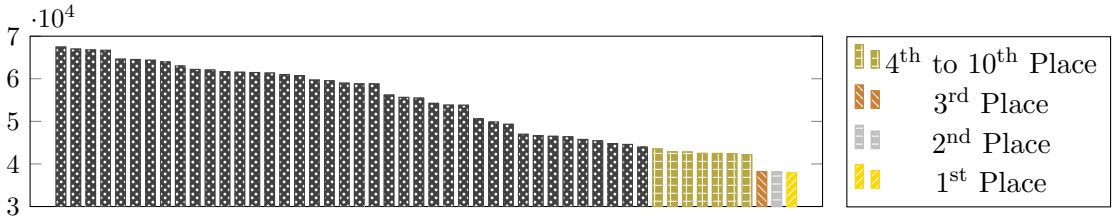



Figure 5: Distribution of total byte counts.

### 5.1. The "Human-in-the-Loop" vs. Manual Implementation Spectrum

Teams varied significantly in their reliance on Artificial Intelligence, ranging from nearly full automation to purely manual golfing.

- **LLM-Driven Automation:** The 4th place team used a "human-in-the-loop" algorithm where 98% of their solutions were generated by an LLM (specifically gpt5-codex). They ran parallel sampling sessions ($N = 4$) and used rule-based prompts generated from the code's Abstract Syntax Tree (AST) to suggest specific improvements, such as converting loops to recursion.
- **Semi-Automation:** The 7th place team used a semi-automated workflow with the Codex web interface, selecting the shortest of 4 generated outputs. They provided XML-structured prompts and specific "work history" contexts to prevent the AI from repeating failed experiments.
- **DSL Experimentation:** The 5th place team initially experimented with a domain-specific language previously developed for the original ARC challenge (Hodel, 2024), achieving compressed sizes around 2000 bytes.[12] However, they ultimately abandoned this approach in favor of handwritten solutions to achieve better results.
- **Manual Golfing:** Similarly, the 8th and 9th place teams found LLMs insufficient for top-tier scores and pivoted to manual implementation. The 9th place team spent over 1,000 hours manually optimizing code, using LLMs only for learning concepts rather than generating solutions.

### 5.2. Advanced Compression Pipelines (Zopfli & Custom Tools)

Standard libraries were often insufficient; teams built sophisticated compression pipelines to shave off bytes.

- **Zopfli over Zlib:** Multiple teams (5th, 7th, 8th) identified `zopfli` as superior to the standard Python `zlib` library for compression ratios.
- **Variable Name Optimization:** Teams discovered that variable naming impacts Huffman coding efficiency in compression. The 5th and 6th place teams used static analysis and local search (e.g., genetic algorithms or simulated annealing) to rename

12. See https://github.com/michaelhodel/arc-dsl and https://github.com/michaelhodel/re-arc.

| Team Rank | Team Name | Final Score | # Golf Medals | | | | Solution Write-Up |
|---|---|---|---|---|---|---|---|
| | | | Diamond | Gold | Silver | Bronze | |
| #1 | Code Golf International | 962,070 | 80 | 229 | 80 | 43 | [link] |
| #2 | jailctf merger | 961,805 | 55 | 192 | 94 | 56 | [link] |
| #3 | ox jam! | 961,784 | 70 | 194 | 80 | 50 | [link] |
| #4 | FuunAgent | 957,810 | 3 | 74 | 23 | 13 | [link] |
| #5 | HIMAGINE THE FUTURE. | 957,568 | 0 | 58 | 14 | 10 | [link] |
| #6 | Theoretical Syntax Golfers | 957,527 | 5 | 63 | 22 | 18 | [link] |
| #7 | LogicLynx | 957,513 | 8 | 89 | 29 | 21 | [link] |
| #8 | import itertools | 957,177 | 1 | 74 | 11 | 6 | [link] |
| #9 | Santa Golf Club 2500 | 957,151 | 0 | 70 | 13 | 9 | [link] |
| #10 | lv1.dev | 956,458 | 3 | 95 | 22 | 15 | [link] |

Table 2: The winning teams.

variables to be "compression-friendly." Similarly, the 7th and 8th place teams ran variable name searches to optimize compressibility.

- **Latin-1 Encoding:** To avoid the overhead of multibyte Unicode characters in Python strings, teams encoded compressed payloads using Latin-1 (`#coding: L1`), which represents all 256 of its characters with one byte.
- **Custom Decompression Stubs:** Teams optimized the decompression code itself, often using `exec(zlib.decompress(..., -9))` to handle raw deflate streams and minimize header overhead.

### 5.3. Exploiting Domain Specifics: Seed Cracking

Some teams bypassed the logic puzzles entirely by analyzing the test case generation engine.

- **Recovering Random Seeds:** The 5th place team analyzed the ARC-GEN generator and discovered that `random.seed` was often predictable based on task parameters like grid size and color count.
- **Regeneration:** By recovering the seed, they could regenerate the exact solution grid using a short random generator script rather than solving the puzzle logic, significantly compressing the solution size.

### 5.4. Common Python Golfing Idioms

Several specific syntactic patterns appeared repeatedly across different teams' solutions.

- **Recursive Templates:** A dominant pattern for grid updates was using lambda recursion (e.g., `p=lambda g, i=count: -i*g or p(..., i-1)`) to repeat logic without explicit loops.

- **Rotation and Transposition:** Teams frequently used `[*zip(*g)]` for transposition and `[*zip(*g[::-1])]` for rotation. This allowed them to write logic for one direction and rotate the grid to apply it in all four directions.
- **Regular Expressions:** Regex was heavily utilized for pattern recognition and replacement via `re.sub` on the string representation of the grid. The 4$^{\text{th}}$ place team used AST analysis to prompt LLMs to try regex, while the 5$^{\text{th}}$ place team used advanced conditional group syntax.
- `exec` **Loops:** For complex iterations, using `exec` with string repetition was a common byte-saving alternative to standard loops (e.g., `exec("..."*count)`).

### 5.5. Infrastructure and Tooling

Success was supported by robust tooling built around the golfing process.

- **Dashboards:** Teams built custom web dashboards to track progress, visualize score trajectories, and identify tasks where they lagged behind best-known results.
- **CI/CD Integration:** The 4$^{\text{th}}$ place team implemented CI/CD pipelines to auto-generate README stats and notify the team via Discord upon every push, facilitating rapid iteration.
- **Search Tools:** Teams utilized tools like `pysearch` to brute-force short Python expressions for specific logic.

## 6. Conclusion

While our code golf championship was framed as a gamified test of software engineering ingenuity, the collective output of its participants has delivered far-reaching implications for the future of machine learning and agentic systems. What began as a recreational challenge for computer programming enthusiasts has culminated in a foundational contribution to the field of automated reasoning. The resulting repository — comprising tens of thousands of verified, highly compressed Python solutions — transforms what was once a collection of abstract visual puzzles into a rigorously defined corpus of logic. These open-source artifacts can now serve the critical scientific function of establishing precise complexity bounds for the ARC-AGI-1 benchmark suite and providing the ground truth necessary to train the next generation of coding agents. Ultimately, this initiative demonstrates that the path to solving general intelligence may not lie solely in massive scale, but also in the creative coordination of human intellect and expertise.

## Acknowledgments

We are grateful to the NeurIPS 2025 Competition Track Chairs (specifically Jes Frellsen, Kun Zhang, and Tao Qin) for reviewing and accepting our proposal, and to the members of the ARC Prize Foundation (including François Chollet, Greg Kamradt, Mike Knoop, and Bryan Landers) for allowing us to use their benchmark suite as the basis for our challenge. Finally, we wish to thank Google leadership (particularly Samrat Ghosh, Ya Xu, and Zongwei Zhou) for lending their support to this event.

## References

James Ainooson, Deepayan Sanyal, Joel P. Michelson, Yuan Yang, and Maithilee Kunda. A Neurodiversity-Inspired Solver for the Abstraction & Reasoning Corpus (ARC) Using Visual Imagery and Program Synthesis, 2023. URL https://arxiv.org/abs/2302.09425.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The Surprising Effectiveness of Test-Time Training for Few-Shot Learning. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML 2025, 2025. URL https://openreview.net/forum?id=asgBo3FNdg.

Simon Alford, Anshula Gandhi, Akshay Rangamani, Andrzej Banburski, Tony Wang, Sylee Dandekar, John Chin, Tomaso Poggio, and Peter Chin. Neural-Guided, Bidirectional Program Search for Abstraction and Reasoning. In *Proceedings of the 10th International Conference on Complex Networks and Their Applications*, volume 1, pages 657–668, 2022.

Andrew Barron, Jorma Rissanen, and Bin Yu. The Minimum Description Length Principle in Coding and Modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

Mikel Bober-Irizar and Soumya Banerjee. Neural networks for abstraction and reasoning. *Scientific Reports*, 14, 2024. URL https://doi.org/10.1038/s41598-024-73582-7.

Gregory J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *J. ACM*, 13(4):547–569, October 1966. URL https://doi.org/10.1145/321356.321363.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code, 2021. URL https://arxiv.org/abs/2107.03374.

François Chollet. On the Measure of Intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC Prize 2024: Technical Report, 2024. URL https://arxiv.org/abs/2412.04604.

François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC Prize 2025: Technical Report, 2026. URL https://arxiv.org/abs/2601.10904.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Evaluating Large Language Models in Class-Level Code Generation. In *Proceedings of the 46th International Conference on Software Engineering*, ICSE 2024, 2024. URL https://doi.org/10.1145/3597503.3639219.

Peter Grünwald. A Tutorial Introduction to the Minimum Description Length Principle, 2004. URL https://arxiv.org/abs/math/0406077.

Céline Hocquette and Andrew Cropper. Relational Decomposition for Program Synthesis. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence*, IJCAI 2025, pages 4526–4534, 2025. URL https://doi.org/10.24963/ijcai.2025/504.

Michael Hodel. Addressing the Abstraction and Reasoning Corpus via Procedural Example Generation, 2024. URL https://arxiv.org/abs/2404.07353.

Andrey Kolmogorov. Three Approaches to the Quantitative Definition of Information. *Problems of Information Transmission*, 1(1), 1965.

Chao Lei, Nir Lipovetzky, and Krista A. Ehinger. Generalized Planning for the Abstraction and Reasoning Corpus. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, AAAI 2024, pages 20168–20175, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29996.

Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 4th edition, 2019. ISBN 978-3-030-11297-4. URL https://doi.org/10.1007/978-3-030-11298-1.

Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M. Dunn, Hao Tang, Wei-Long Zheng, Yewen Pu, and Kevin Ellis. Combining Induction and Transduction for Abstract Reasoning. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL https://openreview.net/forum?id=UmdotAAVDe.

Isaac Liao and Albert Gu. ARC-AGI Without Pretraining, 2025. URL https://arxiv.org/abs/2512.06104.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. WizardCoder: Empowering Code Large Language Models with Evol-Instruct. In *Proceedings of the 12th International Conference on Learning Representations*, ICLR 2024, 2024. URL https://openreview.net/forum?id=UnUwSIgK5W.

Matthew Macfarlane and Clément Bonnet. Searching Latent Program Spaces. In *Proceedings of the 39th Conference on Neural Information Processing Systems*, NeurIPS 2025, 2025. URL https://openreview.net/forum?id=CsXKGIqZtr.

Michael D. Moffitt. ARC-GEN: A Mimetic Procedural Benchmark Generator for the Abstraction and Reasoning Corpus, 2025. URL https://arxiv.org/abs/2511.00162.

Norbert Neumann and Ádám Pintér. Solving ARC with non-procedural program induction. In *Proceedings of the 23rd International Symposium on Computational Intelligence and Informatics*, CINTI 2023, pages 271–276, 2023. URL https://doi.org/10.1109/CINTI59972.2023.10382009.

Simon Ouellette. Towards Efficient Neurally-Guided Program Induction for ARC-AGI, 2024. URL https://arxiv.org/abs/2411.17708.

Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-Improving Language Models for Evolutionary Program Synthesis: A Case Study on ARC-AGI. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML 2025, 2025. URL https://openreview.net/forum?id=z4IG090qt2.

Ruizhong Qiu, Weiliang Will Zeng, James Ezick, Christopher Lott, and Hanghang Tong. How Efficient is LLM-Generated Code? A Rigorous & High-Standard Benchmark. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL https://openreview.net/forum?id=suz4utPr9Y.

J. Rissanen. Modeling By Shortest Data Description. *Automatica*, 14(5):465–471, 1978. URL https://www.sciencedirect.com/science/article/pii/0005109878900055.

Filipe Marinho Rocha, Inês Dutra, Vítor Santos Costa, and Luís Paulo Reis. Program Synthesis Using Inductive Logic Programming for the Abstraction and Reasoning Corpus. *Intelligenza Artificiale*, 19(2):85–101, 2025. URL https://journals.sagepub.com/doi/10.1177/17248035251363178.

R. J. Solomonoff. A Formal Theory of Inductive Inference. Part I. *Information and Control*, 7(1):1–22, 1964. URL https://www.sciencedirect.com/science/article/pii/S0019995864902232.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering Code Generation with OSS-Instruct. In *Proceedings of the 41st International Conference on Machine Learning*, ICML 2024, 2024. URL https://openreview.net/forum?id=XUeoOBid3x.

Jonas Witt, Sebastijan Dumančić, Tias Guns, and Claus-Christian Carbon. A Divide, Align, and Conquer Strategy for Program Synthesis. *Journal of Artificial Intelligence Research*, 82:1961–1997, March 2025. URL https://doi.org/10.1613/jair.1.16847.

Yudong Xu, Elias B. Khalil, and Scott Sanner. Graphs, Constraints, and Search for the Abstraction and Reasoning Corpus. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, AAAI 2023, pages 4115–4122, 2023. URL https://doi.org/10.1609/aaai.v37i4.25527.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, et al. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence, 2024. URL https://arxiv.org/abs/2406.11931.

## Appendix A. The Official Competition Leaderboard

| # | Team | Members | Score | Entries |
|---|---|---|---|---|
| 1 | Code Golf International | | 962070.000 | 384 |
| 2 | jailctf merger | | 961805.000 | 169 |
| 3 | ox jam! 🎤🐂🎸🐂🎷🐂🥁🐂🎹🐂 | | 961784.000 | 330 |
| 4 | FuunAgent | | 957810.000 | 573 |
| 5 | HIMAGINE THE FUTURE. | | 957568.000 | 51 |
| 6 | Theoretical Syntax Golfers | | 957527.000 | 38 |
| 7 | LogicLynx | | 957513.000 | 88 |
| 8 | import itertools | | 957177.000 | 569 |
| 9 | Santa Golf Club 2500 | | 957151.000 | 290 |
| 10 | lv1.dev | | 956458.000 | 39 |
| 11 | cam jam! 💥📸🧀🎹💥🧀🎹📸 | | 956008.000 | 181 |
| 12 | ﾀｲｽｹﾝｻﾝﾊﾞ II | | 955424.000 | 112 |
| 13 | 暗黒AGI | | 955227.000 | 2278 |
| 14 | ALE-Agent🐟 | | 954571.000 | 55 |
| 15 | santa2024 | | 954246.000 | 3216 |
| 16 | mystery+bono+Lucien | | 953632.000 | 314 |
| 17 | Golfers Anonymous | | 953512.000 | 48 |
| 18 | Team JYCDT | | 953336.000 | 1219 |
| 19 | Team | | 953039.000 | 887 |
| 20 | ShadowPrompt Labs | | 950693.000 | 22 |
| 21 | abdikmnrsttvwxyz | | 950146.000 | 44 |
| 22 | JRKKX | | 949384.000 | 531 |
| 23 | needAJob | | 946216.000 | 64 |
| 24 | R.U.M. | | 946190.000 | 16 |
| 25 | Ravi Annaswamy | | 945774.000 | 154 |

## Appendix B. Detailed Team Strategies

In the sections that follow, we include detailed strategies for each of the top three teams. The first two are taken nearly verbatim from their online Kaggle solution write-ups, while the third summarizes a presentation delivered at our in-person Special Session held during NeurIPS on December 7th, 2025 in San Diego, California.[13,14,15]

### B.1. Code Golf International (1st Place)

First and foremost, we are very grateful to the organizers of the competition for such a unique opportunity to engage in competitive code golf with the broader Kaggle community. The last three months have been a thrilling experience for all of us, and finishing in first is a tremendous honor.

#### B.1.1. Overview

*Solutions*
Our code repository is available online.[16] The final contest submission was from this commit. To reproduce our entry, you can download the repo and simply run `python auto_zip.py`, which will create the `submission.zip` file.

*Approach*
Our team consisted of 5 experienced code golfers. Our approach to this competition was very manual, with all solutions authored by hand. The only important automation was zlib-based compression, which applied to 21 of the 400 tasks (more on that below).

*Results*
Our strong results in the competition can be mainly attributed to expertise in golfing and a large investment of time. To help understand the time investment, the average task might take a few hours to hand-optimize to a high level (including time of both the original author and people who reviewed), plus many tasks needed to be revisited as we discovered new or improved techniques. Multiplying by 400 tasks and it becomes quite an undertaking, even for a team of 5 splitting the work.

The competition was extremely close among the top 3 teams, and there is no particular technique or strategy we possessed that gave a decisive advantage over 2nd or 3rd. Indeed, reviewing others' solutions post-competition, we missed some useful techniques that the other teams had.

#### B.1.2. Techniques

Because all the problems are grid-based, there were many recurring techniques in our solutions. In this section, we outline some of the highlights, focusing on techniques that were unique to this contest rather than standard Python golf knowledge. However, this is in no way a complete list: every problem was hand-optimized and so there are many unique tricks.

13. https://kaggle.com/competitions/google-code-golf-2025/writeups/cgi
14. https://kaggle.com/competitions/google-code-golf-2025/writeups/solution
15. https://neurips.cc/virtual/2025/competition/127722
16. https://github.com/Seek64/NeurIPS-Code-Golf-2025

*Short grid operations*
Because these operations are quite short, they were essential building blocks of solutions.

### Grid operations

- `zip(*g)` transposes the grid
- `g[::-1]` vertically mirrors the grid
- `zip(*g[::-1])` rotates 90 degrees
- `sum(g,[])` flattens the 2-d grid to a 1-d list. Note that in many cases `sum(g,g)` could be used to save a byte.
- `filter(any,g)` removes empty rows

### Row operations

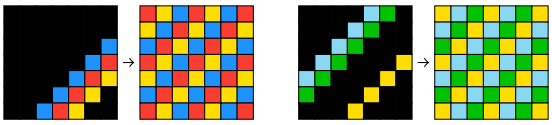
- `min`, `max`, `any`, `all`, and `sorted` built-in functions were used ubiquitously. For `min`, `max`, and `sorted`, the `key` argument was often utilized, e.g., to get the least/most common color.
- `{}.fromkeys(r)` will get the unique colors, sorted in order of first appearance. Alternatively, a dict comprehension could be used: `{v:0for v in r}`
- `filter(int,r)` removes black cells

*Recursion and loops*
Another fundamental building block was using recursion or loops to repeat operations. Many solutions are of the form "do operation X, rotate 90 degrees, repeat 3 more times".

The baseline recursion template is `p=lambda g,n=3:-n*g or F(p(g,n-1))`, taking advantage that `-n*g` is falsy for `n=0...3`. This is more efficient than writing out a loop such as `for n in range(4)`. In special cases, there are shorter methods or useful alternatives:

- If the recursion is just depth 2, then one can write `p=lambda g,h=0:F(h or p(g,g))`, saving 4 bytes in most cases. The variation `p=lambda g,*h:F(*h or p(*g))` saves another 2 bytes but removes the first row.
- Depth 2 recursion with transposition could sometimes be done as `p=lambda g:[*map(F:=lambda*r:...,*map(F,*g))]`, saving 1 more byte.
- If the grid is fixed height, then there's a "slice recursion" technique, where you double the grid with each iteration and use a slice to stop the recursion when the desired depth is reached: `p=lambda g:F(g[70:]or p(g*2))` will iterate 4 times on a $10 \times 10$ grid. This saves bytes because it eliminates extra arguments to `p`. For example, Task #093 uses `p=lambda g:g[57330:]or ...` to iterate 12 times over a $14 \times 14$ grid, because $57330 = 14 \times (2^{12} - 1)$.
- As an alternative to recursion, there's a trick to make short loops: `p=lambda g:[g:=F(g)for _ in g][3]`. This is useful when `g` needs to be used multiple times inside `F`.

- `exec/eval` can be used with string multiplication to repeat code, e.g., `exec("F(x);"*9)`. It's very situational whether it's useful, but it can lead to small byte saves in a number of cases. An example is Task #007:



```
p=lambda g:eval(f'[{"max(sum(g:=g[1:3]+g,[])[2::3]),"*7}],'*7)
```

In this task, there is a pattern that repeats with length 3, and we must fill in the missing cells. To do so, we take the maximum element of the flattened grid after slicing by steps of 3, and we mutate the grid so that we can always use the same slice, avoiding any need for loop variables. Because the grid is always $7 \times 7$, we can use string multiplication with `eval` to do each cell operation 7 times and each row operation 7 times.

*List comprehensions*

The most common structure of our solutions was to use a list comprehension to transform the grid. The most basic form of the code will look like this:

```
p=lambda g:[[F(x)for x in r]for r in g]
```

This operates on single cells only, which is quite limiting. To do more complex operations that require looking beyond a single cell, state variables can be introduced with the walrus `:=` operator. For example, to include the cumulative sum of the row, one can write

```
p=lambda g:[(s:=0)or[F(x,s:=s+x)for x in r]for r in g]
```

Initializing the state variable could sometimes be done more efficiently using `map` and `lambda`:

```
[(a:=0)or[F(x,a:=...)for x in r)for*r,in zip(*g)]
#shortens to
[*map(lambda*r,a=0:[F(x,a:=...)for x in r],*g)]
```

Another technique is mutating the list while operating on it. In particular, `list.pop()` was a source of multiple tricks. For example, it could facilitate "lookahead" in a list comprehension, such as testing whether a certain color appears later in the row:

```
#Original version, check if there is a gray cell in the row to the
#right of the current cell
p=lambda g:[[F(x,5in r[i:])for i,x in enumerate(r)]for r in g]
#Optimized with pop()
p=lambda g:[[F(r.pop(0),5in r)for _ in r*1]for r in g]
```

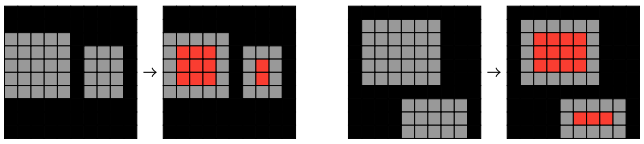
`list.pop()` can also implicitly reverse a list when popping from the back instead of front:

```
p=lambda g:[[F(r.pop(0))for _ in r*1][::-1]for r in g]
#shortens to
p=lambda g:[[F(r.pop())for _ in r*1]for r in g]
```

*Regular expressions*

Regular expressions, in particular `re.sub`, are extremely powerful for concisely editing strings. Therefore, it is often shorter to convert the given input matrix into a string, perform the substitution, and convert back to a matrix using `eval`. Our final submission used regex in a total of 46 solutions.

A simple example of regex can be seen for Task #294:



```
import re
p=lambda g:eval(re.sub("(?<=5.{34})5(?=.{34}5)","2",str(g)))
```

The above substitution replaces all gray pixels with red if they are in the middle of a diagonal of three gray pixels. Since the grid size is fixed ($10 \times 10$), we know that diagonally connected pixels are 34 characters apart.

For more complex drawing tasks, a *rotate-and-replace* strategy was often used:

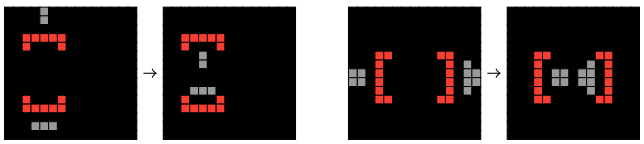
```
p=lambda g:[g:=eval(re.sub("...","...",f"{*zip(*g[::-1]),}"))↩
for _ in g][3]
```

We employed different variations of this approach (list comprehension, recursion, or loops) depending on the task. An interesting trick we found to save one byte is that f-strings allow unpacking, i.e., `f"{*zip(*g[::-1]),}"` vs `str([*zip(*g[::-1])])`.

In some of the most complex tasks, we utilized multiple substitutions with different expressions. One way of doing so is:

```
p=lambda g:[g:=eval(re.sub(*s,f"{*zip(*g[::-1]),}"))for s in[↩
("...","...")]*16+[("...","...")]*4][-1]
```

Because the result after `re.sub` is `eval`ed, the replacement could be any Python expression. This was a source for some interesting tricks. For example, in Task #154, we use the replacement `*[\g<0>][::-1]` to reverse the matched substring.



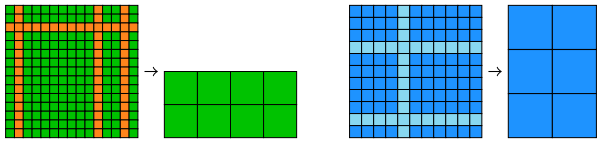
```
import re
p=lambda g,k=0:eval(re.sub("[^(2]{9}2"*2+"?","*[\g<0>][::-1]",↩
f"{*zip(*k or p(g,g)),}"))
```

*"Dimension" recursion*

A conceptually difficult but very powerful technique is to re-use the function `p` for different shapes of input: a 2-d grid (`list[list[int]]`), 1-d row (`list[int]`), or 0-d cell (`int`). This can lead to significant savings in cases where the 2-d operation and 1-d operation can use overlapping code. There are several variations, but one template is

```
p=lambda g:g*0!=0and[p(r)for r in g]or F(g)
```

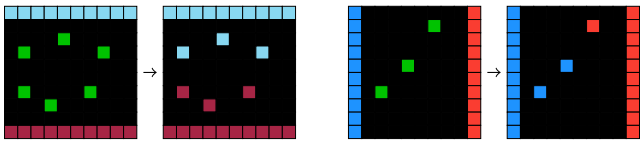
where `F` is the `int -> int` operation. The condition `g*0!=0` checks if the input is a list, and separates the 2-d/1-d cases from the `int` case. For example, Task #021:



```
p=lambda g:g*-1*-1or[p(g:=r)for r in g if g!=r][::2]
```

In this task, the row and column operations can use the exact same code: skip identical consecutive elements and downsample by 2. Note that the formula `g*-1*-1` can be used here instead of `g*0!=0` because cells never have value 0 (black) in this task.

This technique becomes extremely potent when utilizing additional arguments to `p`. For example, Task #040:



```
p=lambda g,h=[]:g*0!=0and[*map(p,g[:1]*5+g[9:]*5,h+g)]or h%~h&g
```

The high-level idea behind this solution is to color each cell according to the nearest corner of the grid. The trick is that recursively applying `g[:1]*5+g[9:]*5` will yield the colors of the nearest corners (taking advantage that the grid is always $10 \times 10$). At the end of the recursion, `h` will contain the current cell and `g` will contain the nearest corner. Note that `h%~h&g` is a math trick to efficiently express `h and g`.

*Guess-and-check*

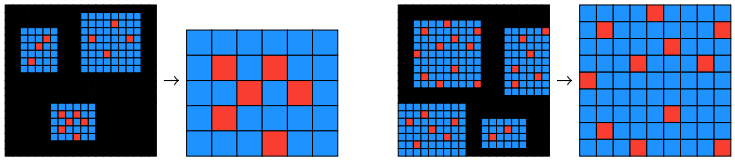
In some problems, it is difficult to search for an object, but easy to verify whether an object is correct. As such, it is more efficient to simply keep guessing the object until the correct one is found. One example is Task #124:



```
p=lambda g,x=0,y=2:(G:=[(i//y*x*[0]+g[i%y])[:10]↩
for i in range(10)])*(G[:4]<g<G)or p(g,~-x%3,y^1)
```

The goal of this task is to extend a pattern of height 2–3 and a horizontal offset of 0–2. In this solution, we guess the height `y` and offset `x` and use `G[:4]<g<G` to check if it agrees with the input. If the condition is true, the function returns, otherwise it will proceed to the next guess `p(g,~-x%3,y^1)`.

Built-ins such as `max` and `min` could also be used to choose the correct guess. For example, Task #216:



```
p=lambda g:max((-(c:=sum(l:=[r[x%17:x%21]for r in g[x%19:x%22]]↩
,g).count)(0),c(2),c(1),l)for x in range(8**6))[3]
```

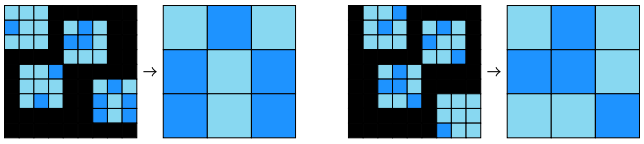
It's difficult to enumerate *only* the blue rectangles; it's much easier to enumerate every rectangular region and check that it's a valid blue rectangle. To enumerate every rectangular region, we use the math trick that `x%17`, `x%21`, `x%19`, `x%22` go through every combination of values (by the Chinese Remainder Theorem). To pick out the answer, we use `max` to simultaneously find the most red cells (via `c(2)`) and verify the region is, in fact, a complete blue rectangle, i.e., has no black cells `-c(0)` and cannot be extended with more blue cells `c(1)`.

*3x3 neighborhood iterator*

A compact method to iterate over all $3 \times 3$ neighborhoods in the grid is

```
[F(i)for*h,in map(zip,g,g[1:],g[2:])for*i,in map(zip,h,h[1:],h[2:])]
```

Using `eval`/`exec`, the repetition in the two `for` clauses can be reduced, as seen in our solution to Task #271:



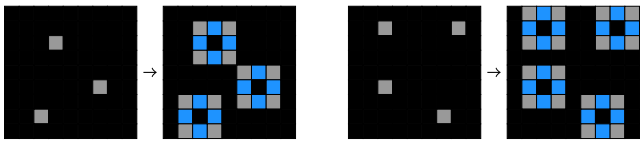
```
exec(f"p=lambda g:max([str(g).count('1'),g]{'↩
for*g,in map(zip,g,g[1:],g[2:])'*2})[1]")
```

*pysearch*

Many problems required small lookup tables, e.g., mapping of one color to another. We used the utility `pysearch` extensively to find the shortest expressions that reproduced a desired input–output relationship.[17]

17. https://github.com/lynn/pysearch

A more sophisticated usage of pysearch was to find formulas that could be iterated to produce the desired transformation. One example is Task #282:

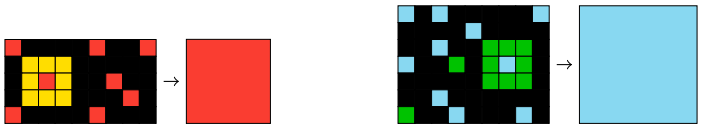


Using pysearch with custom code to simulate four rotate-and-transform operations, we found that the desired pattern could be produced with the formula `p//5|p^c`, where `p` is the previous color (when scanning left-to-right in each row) and `c` is the current color. With some clever usage of `eval` and slice recursion, our solution is:

```
p=lambda g:g[99:]or[eval(8*"+(x:=r.pop()),x//5|x^0")↩
for*r,in zip(*p(g*2))]
```

*Cheese and hardcoding*

In code golf parlance, "cheese" refers to solutions that exploit weaknesses in the judge, rather than adhering to the intended solution (similar to the concept of overfitting in machine learning). In this competition, most tasks had hundreds of tests, so opportunities for cheese were limited. However, an example of severe cheese is Task #346:



For human eyes, the intention is obviously to pick the color which has a point surrounded by the other color. But that usually is the least frequent color in the grid, which is much easier to express in Python. By optimizing the slice of the grid to look at, we can force the least frequent color to always match the solution:

```
p=lambda g:[[min(f:=sum(g[:6]+g,[])[11:],key=f.count)]]
```

In addition, we solved two problems by hard coding a lookup table: Task #048 and Task #319. Both of these tasks need only 1 bit to resolve, so hard coding a table of bits was reasonably efficient. (In fact, Task #048 required less than 1 bit per test case, because the answer was more likely to be 8 than 0 by a factor of about 2 : 1, which has an entropy of 0.92 bits.) Our hard-coding technique is quite interesting by itself, using the hash of the grid combined with data stored in a byte string:

```
(o:=hash((*b"%a"%g,)))//b'data_goes_here'[o%14]%2
```

Extra bytes can be added to the hashed data `b"%a"%g` to optimize the efficiency of the look up table, which can be searched by brute force.

*zlib compression*
Using a combination of Python's `zlib` and `exec`, it is possible to run compressed code. This technique was useful for solutions longer than about 200 bytes. To get the best possible compression, we had two pieces of infrastructure:

1. We used `zopfli` or `libdeflate` for an initial pass of compression. However, these tools do not account for the extra cost of embedding data into a Python script, i.e., certain bytes must be escaped. We wrote a re-encoder that used the same Huffman tree from `zopfli` or `libdeflate`, and then used dynamic programming to find the optimal encoding.

2. We used ad-hoc scripts to try different combinations of variable names to find what compressed best. Golfing with compression completely changed the techniques for optimal code size. Compression would make code repetition almost free, so the goal with zlib golfing is to use the same constructs over and over, rather than trying to find the shortest construct for each step of the algorithm.

*No shell*
It is worth mentioning that although shell commands were allowed in the competition (e.g. `os.popen`), and likely would have been useful in certain tasks (e.g., perl was available and is an excellent golfing language), we did not use this technique as per mutual agreement among the teams in the top 3. The reason being that the legality of this technique was clarified somewhat late in the competition, and fully exploring this technique would add considerably more work to an already large workload.

#### B.1.3. Acknowledgements

We are grateful to @garrymoss's *Golfed solutions and explanations to hard problems*,[18] @kenkrige's *Chipping skills 3: Regex*,[19] and the commenters there for sharing solutions that improved on our best at the time. We would also like to thank everyone who contributed to the public scores spreadsheet, which was an invaluable source of score benchmarks.

### B.2. jailctf merger (2nd Place)

#### B.2.1. Summary

Our team was formed from 5 CTFers involved in the organization of jailCTF,[20] one of whom is a seasoned code golfer. We focused on getting tooling for compression and easy collaboration very early on the competition, hoping that a team effort could make up for the lack of golfing experience the other four of us had. In the last month, we wrote a fuzzer to get some easy bytes on some tasks, which ended up finding a cool trick and helped save 100+ bytes in the final stretch.

The work was approximately split as such:

- **lydxn** did most of the heavy code golfing, finding the majority of the tricks in our solutions

18. https://kaggle.com/code/garrymoss/golfed-solutions-and-explanations-to-hard-problems
19. https://kaggle.com/code/kenkrige/chipping-skills-3-regex
20. https://ctf.pyjail.club/

- **quasar098** worked on much of the tooling
- **oh_word** focused on compression and compression based tasks
- **hellopir2** did a lot of optimization all around
- **Quasar-0147** was mostly inactive

*Submission and Tooling*
All of the code and tooling that we used can be found in our online repositories,[21] including `charbrute` and `var_genetic`.[22] Our final `submission.zip` file is also available,[23] along with the spreadsheet we used as a central hub for information.

### B.2.2. Charbrute

We named the fuzzer charbrute, because what it does is randomly mutate the code at the character level (deletions, substitutions, insertions). A so-called "astbrute" was proposed but never implemented due to high complexity and low perceived gains. The fuzzer was originally pretty bad, and only handled deletions and substitutions. Over time, we added a few improvements, the major ones being:

- Handling insertions as a replacement for substitutions
- Adding a chunk mover which moved around chunks of code (this one didn't really give good results, maybe moving it to before the other mutations would've helped)
- Forcing the spot mutations to be centralized around a hotspot with a probability distribution
- Using pypy, since pypy3.11 came out recently. This creates a few minor differences such as set ordering and float caching but these can be manually filtered or blacklisted.

It mostly found minor saves and optimizations in our code, but the occasional large save was found. It very quickly got saturated for our code dataset, and so we were forced to keep adding new features or slight modifications to keep its productivity. Examples of some saves that it found are below:

```
# task 131, -1b
p=lambda g:exec("c=2;g[:]=zip(*([j for*j,in g if(c:=c-(s:=max(j))↩
*(c>0))|s]+[[8]*9]+[[0]*9]*99)[len(g)-1::-1]);"*4)or g
p=lambda g:exec("c=2;g[:]=zip(*([j for*j,in g if(c:=c-(s:=max(j))↩
*(c>0))|s]+[[8]*9]+[g[0]]*99)[len(g)-1::-1]);"*4)or g
# human optimization, -1b
p=lambda g:exec("c=2;g[:]=zip(*([j for*j,in g if(c:=c-(s:=max(j))↩
*(c>0))|s]+[[8]*9]+g[:1]*99)[len(g)-1::-1]);"*4)or g

# task 125, -3b
```

21. https://github.com/orgs/jailctf-merger-goog-golf/repositories
22. https://github.com/jailctf-merger-goog-golf/compression
23. https://github.com/jailctf-merger-goog-golf/golf/blob/master/submission.zip

```
p=lambda g,i=87,q=8:g*-i or[[[12%c,~q%7//c*8,-(q&(q:=c)%3)%5]↩
[i//42]or c for c in g]for g[::-1]in zip(*p(g,i-1))]
p=lambda g,i=87,q=8:g*-i or[[[12%c,q//c*c,-(q&(q:=c)%3)%5]↩
[i//42]or c for c in g]for g[::-1]in zip(*p(g,i-1))]

# task 196, -3b
p=lambda g,i=23:g*-i or[[q:=[c|4-c&6%~q,c|-c&~q%3,c%4][i//8]↩
for c in g]for g[::-1]in zip(*p(g,i-1))if(q:=8)]
p=lambda g,i=23:g*-i or[[q:=[c|8-c&q,c|-c&~q%3,c%4][i//8]↩
for c in g]for g[::-1]in zip(*p(g,i-1))if(q:=8)]
```

The most notable trick we fuzzed is what I call "pop rotation". For some context, shortly after we matched the public best on Task #015, which was 93 bytes at the time, it looked like this:

```
p=lambda g,i=-7:g*i or p([[r.pop()%9|36%(6^-[0,*r][i//7])%13↩
for _ in g]for*r,in zip(*g)],i+1)
```

Note the unique inside-out recursion structure with the `p` call on the outside (which was rarely done because putting the recursion on the inside saved 1 whitespace character), and the squareness of all the input grids.

The fuzzer found this:

```
p=lambda g,i=-7:g*i or p([[r.pop()%9|36%(6^-[0,*r][i//7])%13↩
for r in g]for*r,in zip(g)],i+1)
```

And it turns out you can just completely remove the `zip()` for the current known best of **87 bytes**:

```
p=lambda g,i=-7:g*i or p([[r.pop()%9|36%(6^-[0,*r][i//7])%13↩
for r in g]for*r,in g],i+1)
```

This is pop rotation, where you perform a 90 degree counterclockwise rotation of a grid with just `.pop()`! For comparison, let's see the other "rotate a grid 90 degrees in a list comprehension" formats:

```
# square grid:
p=lambda g:[[r.pop()for r in g]for r in g]
p=lambda g:[[r.pop()for _ in g]for*r,in zip(*g)]
p=lambda g:[[c for c in g]for g[::-1]in zip(*g)]

# rectangular grid:
p=lambda g:[[r.pop()for r in g]for r in g[0]*1]
p=lambda g:[[r.pop()for _ in r*1]for*r,in zip(*g)]
p=lambda g:[[c for c in r]for r in zip(*g)][::-1]
```

Of course, it has its own advantages and disadvantages compared to the other typical rotation formats, which is why it is not able to be used everywhere.

In the end, we applied this on these tasks:

| | |
|---|---|
| Task #002 | 86 bytes |
| Task #015 | 87 bytes |
| Task #086 | 154 bytes |
| Task #095 | 70 bytes |
| Task #220 | 82 bytes |
| Task #230 | 95 bytes |
| Task #251 | 84 bytes |
| Task #272 | 87 bytes |
| Task #283 | 79 bytes |
| Task #336 | 86 bytes |
| Task #352 | 82 bytes |

### B.2.3. Compression

For compression, we utilized a variable rename optimization script, similar to the public ones. There are two key differences though:

1. Stripping the header and checksum and using `wbits=-9` in the decompression saves a few bytes.

2. Using a genetic algorithm (implemented by Gemini) to optimize for raw compressed length, compared to naive brute force.

Overall, these optimizations saved a few bytes on each compressed task that we had, which was down to just 24 by the end of the competition.

Our initial approach to the longer tasks was to get an LLM to generate an initial working solution by prompting it with the generation code, general code golfing tips, and some human ideas for the specific task at hand, and then afterwards optimizing the code manually for both compressibility and length. However, most of the llm solutions we golfed got obsoleted by either public notebooks or with a solution rewrite.

Other than that, we had a few main strategies, either to rewrite the code for compressibility/brevity or to repeat/reuse code to do different subtasks. We abused the second strategy quite a lot more than the first. In fact, out of every compressed diamond, we own the two highest compression ratios of ~2.91 on our Task #044 diamond and ~2.46 on our Task #173 diamond. Some examples of compression optimizations we found are below:

```
# "for r in " substring repeating during rotation
[r for*r,in zip(*g)][::-1]
[[*r]for r in zip(*g)][::-1]

# "g[y]" substring repeating during len checks
len(g[0])
len(g[y])

# "_ in range(len(_))" substring repeating during bounds checks
len(g)>y>-1<x<len(g[y])
y in range(len(g))!=x in range(len(g[y]))

# optimizing the huffman tree
```

```
x>y
x<y

# renaming variables to make your code more readable
[[c for c in r]for r in g]
[[g for g in g]for g in g]

# adding whitespace in the correct places
}for i in range(r)
} for i in range(r)
```

Note that these tips are not always positive changes! Mileage varies depending on the contents of the rest of the code. As such, optimizing for compressed length is a lot of trial and error (and pattern recognition), and when each compression attempt takes a couple minutes to brute force an optimized variable renamed version to check if it is shorter, it becomes very annoying.

However, the one benefit of compression golfing is that since no one started out particularly experienced at this, our team wasn't at a disadvantage with respect to this golfing skill. But it is quite annoying and very easy to waste a lot of time on if you don't approach the task correctly. Our team had many attempted rewrites that turned out to be a lot longer when compressed compared to the existing solutions, even if they were potentially shorter in terms of uncompressed length.

#### B.2.4. Web UI

We created a centralized web server(s) for golfing and storing solutions, which the other top two teams did not do. This involved a flask server and a websocket server. We did not have a git repository storing all of our solutions because that is too annoying to handle.

Features of the web UI + websocket server that promoted collaboration and efficiency:

- Store solutions and annotations (i.e. per-task notes)
- Real time updates to both solutions and annotations (using websockets)
- Basic authentication to prevent intruders
- Copy test cases button for easy regex manufacturing in regex101 or other
- View test cases button to see how the manually written test cases are wrong compared to the generated ones
- Gen code button to view test case generation code per task
- 20s timeout changeable to 90s timeout to prevent server strain by default
- Two-way integration with our central spreadsheet (google apps script) for easier analysis.
- Random buttons for golfable task finding
- Other tools integration

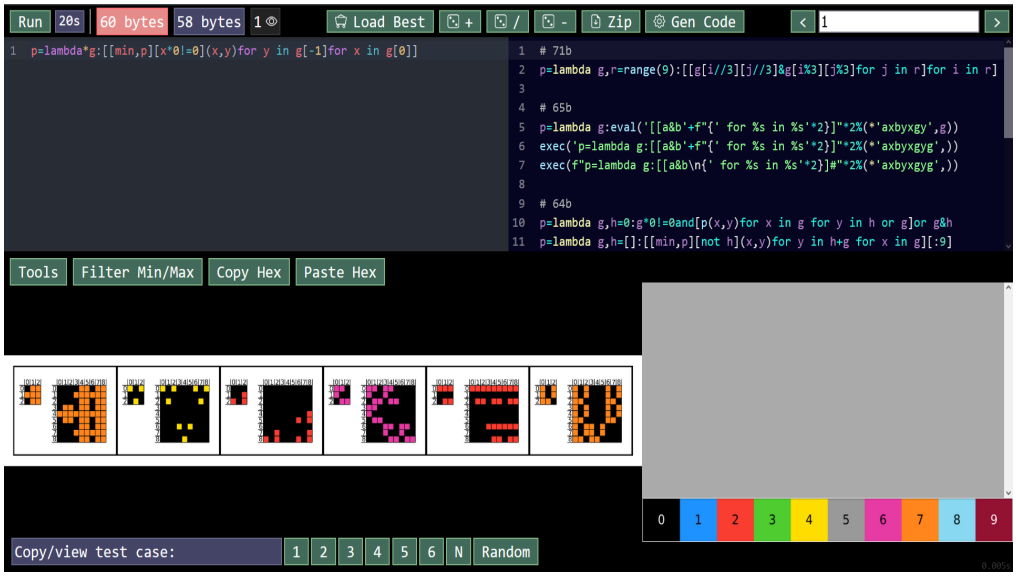



Figure 6: A screenshot of the web UI employed by *jailctf merger*.

### B.2.5. PYSEARCH

Like several other teams, we used the `pysearch` tool to brute force arithmetic expressions. A good example of this is Task #006, where we found an optimal 13-byte expression which isn't possible to find by hand:

```
# 43b (gold)
p=lambda g:[[109//c&12%c+7for c in g[0]]]*3
```

pysearch tends to work well up to length 13/14 until it starts to become too slow. We tried to utilize pysearch as much as possible on any expressions below or around that length. In some cases when the expression is too long to brute force, we can also perform a "partial" pysearch on parts of the expression.

After running pysearch on several tasks, we realized that specifying `INPUTS` and `GOAL` fields manually became extremely annoying and started generating the parameters programmatically. Our gen script for Task #293 looked like this:

```
from utils import pysearch
import json
import re

task_num = 293

with open(f'../data/training/task{task_num:03d}.json', 'r') as f:
    data = json.load(f)

testcases = data['train'] + data['test'] + data['arc-gen']

search = set()
```

```
for _, data in enumerate(testcases):
    g = data['input']

    for r in g:
        for c in g[0]:
            b = +(c in r)
            x = r[0]
            ans = c-c*b or x or c
            search.add((c, b, x, ans))

pysearch(search)
```

Based on the testcases, we fill a set called `search` containing tuples in the format `(input_1, input_2, ..., input_n, goal)` and pass that into a "utils" script that outputs Rust code to plug into `params.rs`:

```
def pysearch(search, min_uses=1, max_uses=255, varnames=()):
    values = list(map(list, zip(*search)))

    big_template = 'pub const INPUTS: &[Input] = &[%s\n];'

    template = '''
    Input {
        name: "%s",
        vec: &%r,
        min_uses: %d,
        max_uses: %d,
    },'''

    output = ''
    for i, value in enumerate(values[:-1]):
        if not varnames:
            c = chr(i + 97)
        else:
            c = varnames.pop(0)
        output += template % (c, value, min_uses, max_uses)
    print(big_template % output)
    print('pub const GOAL: &[Num] = &%r;' % values[-1])
```

`c-c*b or x or c` was the formula we found by hand, while the shortest one is actually `b**c*x or c`. We basically reused this template at every opportunity for easy pysearch gains (which was a lot of tasks!). This was especially helpful for the "grid rotation" tasks, e.g. Task #002 / Task #243.

### B.2.6. Other Tooling

One useful tool we found online was called `infgen` and it allowed us to debug raw deflate streams, seeing both how many bytes each zlib mini-segment used and also how the substitutions were taking place in ways that we might not be able to intuitively see.[24] Another

24. https://github.com/madler/infgen

utility we made was `autogolf`, which made sure we didn't do any silly things with excess spacing or parenthesis and such. This tool was a heavy modification of `ast.unparse`, but with golfing in mind. We integrated both of these to the golf server as buttons to allow us to easily use them.

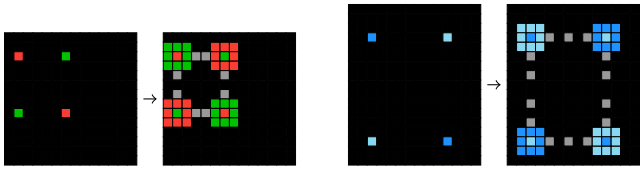
### B.2.7. Interesting Tasks

*Task #048*

This was the only task where we hardcoded every test case using `hash()`, as described elsewhere in a detailed blog post.[25] This trick could also be applied to Task #319 to find the right color.

*Task #387*

This task has you surround the nonzero colors with the other nonzero color, as well as connecting the pixels with an alternating strip of gray and black pixels.



Looking at the other teams' solutions, we were the only team to use regex to solve the task, with it beating out the non-regex solutions of all teams but Code Golf International. So, we thought it would be interesting to explain the regex approach we came up with.

Firstly, notice that this task, like 99% of the tasks, can be greatly simplified if you only deal with one orientation at a time and then rotate. But for this task, the question of how you handle each orientation is not so easily answered. The first thought might be to do something as follows, where you complete one $3 \times 3$ and one edge at a time:

```
___     ___       BBB       ___
_A______B_  -->  BAB5_5_5_B_
___     ___       BBB       ___
```

However, what's interesting is that this proposal actually fills 2 $3 \times 3$s and 2 edges on the initial grid! This gives a new idea: since each $3 \times 3$ and edge is going to be matched twice in the full rotation, we should try to fill in only half of an edge and half of a corner on each passthrough.

The edges are more apparent in how to approach than the $3 \times 3$s. Rather than filling the whole space between the two dots, you only need to fill it halfway. This turns our first transformation into something similar to the below:

```
___     ___       BBB       ___
_A______B_  -->  BAB5_5___B_
___     ___       BBB       ___
```

25. https://lydxn.github.io/posts/fun-with-magic-hashes-in-python/

The 3 × 3s are a little trickier in the approach. If we stare at it a little, we can see that we have a few options in which pair of opposite corners and which pair of opposite edges we want to fill in on the first pass.

```
proposal 1                              proposal 2
___    ___     B__     ___              ___    ___     BB_     ___
_A______B_ --> BAB5_5___B_              _A______B_ --> _A_5_5___B_
___    ___     __B     ___              ___    ___     _BB     ___

proposal 3                              proposal 4
___    ___     __B     ___              ___    ___     _BB     ___
_A______B_ --> BAB5_5___B_              _A______B_ --> _A_5_5___B_
___    ___     B__     ___              ___    ___     BB_     ___
```

Hold on, aren't we doing this in regex? Before we decide which proposal is the best, we need to figure out how to implement this in regex.

So the basic idea we had to implement this was like this:

```
[(stuff)___    ___
[(stuff)_A______B_
[(stuff)___    ___
```

Notice how since the rectangular frame is the only thing on the grid, all of the `(stuff)` groups should be equal. This fixes the issue that we don't know the distance from `A` to `B` which would otherwise make drawing the half dotted line from `A` to `B` quite annoying when trying to access the row below. Instead, if we capture `(stuff)`, we can just go to the next line by doing something like `[^[]+` and then matching `[(stuff)` to find the cells below `A`.

From here, there's not that large of a difference between the methods, so I picked the first one because it was the easiest to implement and probably slightly shorter. After a bit of golfing, we arrive at this 201 byte regex solution:

```
import re
p=lambda g:[g:=eval(re.sub(r'([^)]*)0([^(]*\1)0(, ([^0]), )0,([↩
^)]*)(?!0|\4)((\d).*?\1.{6})0',r'\1\7\2\7\3\7,*[5,0,5][:(c:=1|l↩
en([\5])//3)]+[\5][c:],\6\7',f'{*zip(*g),}'))[::-1]for _ in g][3]
```

Let's break down how this works:

- Firstly, the `([^)]*)0` captures all characters between a `(` and a `0` on a single row. This means `\1` is the `(stuff)` that we wanted to match from earlier. The `0` is consumed because it will be replaced with color `B` in the replacement.

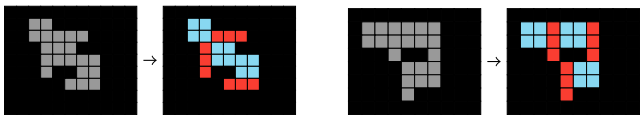
- Then, the `([^(]\1)` captures all characters until the `0` below the previously consumed `0`. This `0` is again consumed because it will be replaced with the color `B`. Following this `0`, we expect `, A,` in the row, which is matched with `(, ([^0]), )`, making `A` be group `\4`. The following `0,` is consumed, again to be replaced with `B`, later.

- After this `0`, we expect the `B` cell to be somewhere in the row. All the cells between the `0` and the `B` are captured with `([^)]*)`. The regex ensures the color `B` exists by searching for the last non-zero, non-A color by using `(?!0|\4)(\d)`. Note that this never captures a `5` instead of `B` because `*` is greedy.
- Then it simply captures up to the next `(stuff)` group using a lazy `.*?\1`, and then consumes the third `0` to be replaced with `B`, matching with the `B` on the third row of proposal 1.
- The replacement is mostly just putting back everything that was consumed and replacing `0`s with `B`s, but there is a small pysearch slice for replacing the first half of the cells between the `0` and `B` on the middle row with the alternating `5, 0` pattern, which will get evaled into the correct replacement.

There are slight variations to this regex that are the same length, but we never found anything shorter than 201 bytes.

*Task #023*

In this task, light blue $2 \times 2$ squares and red $1 \times 3$ lines were placed to fill gray slots on the grid.



This task was tricky because there are many possible positions for the squares and lines but only one would work. We found that the easiest way to solve this task was to use a recursive depth-first search. Essentially, each call of the recursive function would find a suitable gray square, and check all shapes ($1 \times 3$, $2 \times 2$, $3 \times 1$) at that location to see if they fit. Then the algorithm would call itself with the shape retained, and eventually this would fill up the whole grid.

This is one example of a common theme in golfed solutions: **the most efficient example is rarely the shortest**. There were no backtracking checks for $1 \times 1$ gray squares which would obviously mean no red or light blue shape could fit there, since that check would take up extra bytes. We also had to optimize for compression, since our solution was so long. To get the gray square, we found that simply getting the topmost leftmost gray square using two loops would work since all previous shapes would be filled in going top-down left-right as well, so the optimal solution would always be found.

The shapes are encoded as bytestrings, and unpacked using *x,y where y is the color of the shape and x is the list of positive offsets, encoded using modulo. We used byte strings and modulo to get the position offsets to be as short as possible. Also, try except was a short way to cancel the recursion if the shape did not fit into the gray slots, so we used that, along with a non-existent variable, to cancel the loop. Lastly, bitwise operators allowed us to reuse the number `4` a few times which would align nicely with the huffman tree during compression.

```
def p(i):
```

```
for u,m in enumerate(i):
 for r,f in enumerate(m):
  for*e,m in b"lur☒",b"lpt☒",b"lpum☒":
   try:
    for t in e:i[u+t%4][r+t%3]&4or n
    for t in e:i[u+t%4][r+t%3]=m
    if p(i):return i
    for t in e:i[u+t%4][r+t%3]=5
   except:i
  if f&4:return
 return i
```

We originally had the above which got compressed with our genetic varname compressor to be 198b. However, one more byte could be saved by using bytestring bytes that aligned more nicely with the existing code (in huffman tree). The exact bytes and offsets (see `%4` and `%5` below) were brute forced for ones that seemed good.

```
def p(e):
 for n,t in enumerate(e):
  for s,r in enumerate(t):
   for*b,t in b"(in☒",b"(tp☒",b"(ite☒":
    try:
     for i in b:e[n+i%4][s+i%5]&4or y
     for i in b:e[n+i%4][s+i%5]=t
     if p(e):return e
     for i in b:e[n+i%4][s+i%5]=5
    except:e
   if r&4:return
 return e
```

Other teams were able to get their solution shorter using regex, which we did not end up using for this task.

### B.3. ox jam! (3rd Place)

Our team managed to finish 3rd place in this competition. The five of us are a pretty international team, coming from four different continents.

#### B.3.1. Competition Overview

As has been discussed extensively in prior sections, our goal was to create Python functions to solve all four-hundred problems in the ARC-AGI-1 training dataset, and (of course) to do so using the shortest possible solutions, minimizing total byte size with no sharing of code between tasks. Our overall approach — which employed no Artificial Intelligence whatsoever — can be summarized as follows:

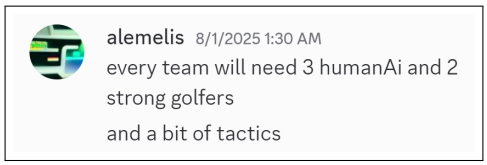
- Manual solving and optimization by human experts, using various code golf tricks that are described in depth on the Kaggle discussion forum and in many of the write-ups produced by different teams.

- A collaboration pipeline we set up through automated Git workflows, allowing us to quickly collaborate and make improvements under the pretty significant time constraints that we faced.

- Similar to other teams, some automated search for improved size over zlib compression.

### B.3.2. Team Expertise

Collectively, the five of us have just over three decades of experience code golfing.[26] Although the hobby has been around for a long time, this was probably the single biggest competition there's ever been, and so we've spent many years solving similar problems for mere recreation without looking for the big sort of pay in tournaments like this one. It's our impression that between the top teams, there was really not that much variance in the tricks that we used; nearly all of us know how to do the rotations, the pops, splits, zlib compressions, etc. However, perhaps one major distinction (in addition to raw effort and hard work) was the sense of a "golfer's instinct." We'll touch upon this topic later when discussing code golf as a global optimization problem, but it basically boils down to how one decides (when there are multiple possible paths to optimize) which one is the *right* one. Regarding team structure, we highlight below one particularly relevant message posted within hours of the competition's launch, speculating that any team wanting to seriously compete would need three (as they referred to it) "human AI members" — basically, people whose goal was to just *grind, grind, grind*, because 400 code golf problems within three months is no light work.

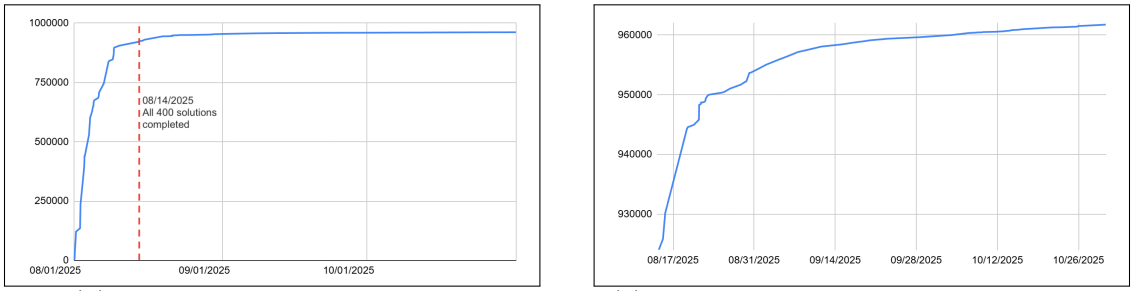



While many other teams were able to supplement that with Artificial intelligence, *none* of us in the top three used AI to produce any of our solutions.

### B.3.3. Score Timeline

In Figure 7(*a*) we display a timeline of our team's score throughout all three months of the contest. One can clearly observe a huge jump toward the beginning, when we went from zero problems solved up to four-hundred. Within the first two weeks of this twelve-week competition, we had solutions posted to all 400 problems. From then on (over the next two-and-a-half months), our time was spent hunting down any small optimizations we could find. At least on this scale, it may appear that basically no real improvement was made during the last two months of the competition — and yet, this is where all the serious back-and-forth occurred between the top teams. To zoom in further on that section, Figure 7(*b*) shows our score on a smaller range after we had produced our initial four-hundred solutions. Once again, almost all of our improvements occurred very quickly, with seemingly little progress

26. Links to our code.golf user profiles: {`ovs-code`, `MeWhenI`, `GrayJoKing`, `xsot`, `zacjl`}.




(*a*) Overall *ox jam!* score timeline (*b*) Timeline after completing 400 solutions

Figure 7: Score timelines for *ox jam!* spanning different segments of the competition.

made over the last few weeks. It was the activity during these final days that ultimately determined who came in first, second, and third.

### B.3.4. Workflow Pipeline for Collaborative Global Optimization

Team code golfing isn't something that happens too often. One major challenge we encountered early on was not just in applying several small golfs, but also how to consider entirely different methodologies when approaching a given problem. In response, we developed a series of GitHub workflows that allowed us all to work individually, and then compile our solutions into a single document for each problem where we could explore our own improvements along with the ideas of everyone else simultaneously. In Figure 8(*a*), we illustrate one such document for Task #390, where every line shows a different solution and/or different approach. Some of these are small improvements on each other, but others reveal completely different approaches. Hence, one of the interesting questions that arises when treating this as an optimization problem is considering not just "*how can I pick the top solution and improve it?*" but also "*is this even the solution that I should be dedicating my time to improving?*" As for how best to answer that question, we believe it invariably comes down to instinct and expertise. This is something our teammates have gotten rather good at figuring out over time: even though one solution may currently be the shortest, there may actually be more opportunity for improvement in another solution altogether. What that means for AI — and whether it can eventually match human expertise — is something we'll cover in subsequent sections, but we believe it significantly helped set us apart. Figure 8(*b*) shows a screenshot of our GitHub commits over time, all of which were pushed using an automated system that would track our golf medals (e.g., diamonds, golds, etc.) along with our total score and what we thought the best possible score might be.

### B.3.5. Community-Driven Optimization

Another important aspect of this competition was the emphasis on community-driven optimization, and especially the unofficial score spreadsheet (see Figure 9) that emerged early on. Coming from the code golf community, we recognized that an incredibly valuable piece of information is knowing how well other people are performing. Very often you'll see a problem where it seems like an optimal score has been reached (e.g., 70 bytes), but then



```
merged > task390.py > ...
# joking (100 vs 98 bytes for gold)
import re
p=lambda g,*G:eval(re.sub("([^2(]{9}2[^2]{9})",r"*[\1][::-1]","%s"%[*zip(*G or p(g,*g))]))

## this task is (almost) identical to task 154
p=lambda g,k=-1:k*g or p([*zip(*[g[a+sum(k*2for k in(2,3,-2,-3)if(g*2)[a+k].count(2)>4)]for a in range(15))])],k+1)

## ovs' solution
p=lambda g,k=-1:k*g or p([*zip(*[g[~sum(~k*2for k in(1,2,-3,-4)if g[k].count(2)>4)]for r in g if(g:=g[1:]+[r]))])],k+1)

### mwi (104 bytes)
import re
p=lambda g:[g:=eval(re.sub("([^2(]{9}2[^2]{9})",r"*[\1][::-1]","%s"%[*zip(*g)]))for _ in g][1]

##
p=lambda g:[g:=[*zip(*[[*g[k~~k]for k in(1,2,-3,-4)if g[k].count(2)>4],r][0]for r in g if(g:=g[1:]+[r]))])for _ in g][1]
p=lambda g:[g:=[*zip("*2, "*3in"%s"%g and g[:(i:=g.index(max(g)))-3]+g[i+3:i+1:-1]+g[i-1:i+2]+g[:2]+g[i+4:]or g)][::-1]for _ in g][3]

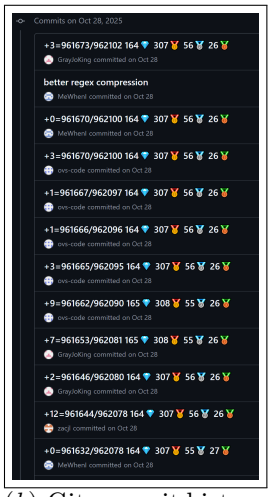
### ovs (109 bytes)
p=lambda g,k=59:-k*g or p([*zip(*g[:-(h:=g[v:=k//4].count(2)>4and k%4|1)+v]+g[v-h:v~~h][::-1]+g[v~~h:])],k-1)
##

p=lambda g:[g:=[*zip(*[g[a-sum(k for k in b'[illegible]if g[a-k*8%15].count(2)>4)]for a in range(15))])for _ in g][1]
p=lambda g,k=-1:k*g or p([*zip(*[[*g[k~~k]for k in(1,2,-3,-4)if g[k].count(2)>4],r][0]for r in g if(g:=g[1:]+[r]))])],k+1)

### combined (126 bytes)
p=lambda m,k=3,R=range(-14,1):-k*m or p([*zip(*[m[(len([l:=i-j for j in R if 2in m[-j]])==4>abs(l)>1)*2*l-i]for i in R])],k-1)

### xsot (136 bytes)
# for each row close enough (but not too close) to the first red row, replace it with the row reflected around the red
# 4==len(d) makes sure we only apply the transform when reds are horizontal
p=lambda m,k=3,R=range(15):-k*m or p([*zip(*[m[(1<abs((d:=[j-i for j in R if 2in m[j]])[0])<4==len(d))*2*d[0]+i]for i in R][::-1])],k-1)
```

(*a*) Evolution of solutions to Task #390 by *ox jam!*




(*b*) Git commit history

Figure 8: Artifacts of the workflow pipeline employed by *ox jam!*

one person breaks through and finds a 69-byte solution, and within an hour somebody else has submitted a 67 or a 66. Just knowing the *potential* to improve is often what it takes to push golfers in the right direction, i.e. to let them know where they should dedicate their time. And so, we are really thankful to the broader code golf community for coming together and forming this resource. However, a lot of us on the top ultimately realized that (unfortunately) as much as we wanted to continue sharing this information, if this information wasn't *obligated* to be shared by everyone, we each would be putting ourselves at a competitive disadvantage. So, toward the end, there were fewer updates to this doc, but it was still great for giving us all a sense of which puzzles to approach and where potential optimizations might lie.

We did feel there were a couple pain points in this competition that made it somewhat difficult to participate and commit ourselves fully. Many other top golfers probably could have come in and contributed even better solutions than we'd seen, but instead elected to opt-out due to various troubles (e.g., the lack of a single-hole leaderboard). If a similar contest happens again in the future, the code golf community can likely offer great ideas about how to run such events — perhaps resulting in even wider participation beyond the thousands of teams that signed up this year.

### B.3.6. Looking Forward: Tractability of Code Golf for AI

As noted earlier, all top three teams used no AI in their solutions, making this a bit of an oddball competition in the context of NeurIPS. In Figure 10, we chart the total program lengths of the top ten teams; while teams $10^{\text{th}}$ through $4^{\text{th}}$ all scored in roughly the same range, there is a noticeable drop in byte count between this group and the top three. A couple factors may explain this gap: one reason (we theorize) is the prize pool being equivalent for $4^{\text{th}}$ through $10^{\text{th}}$ place, resulting in less competitive pressure for teams who felt they couldn't break into the top three. It also suggests there may have been only about

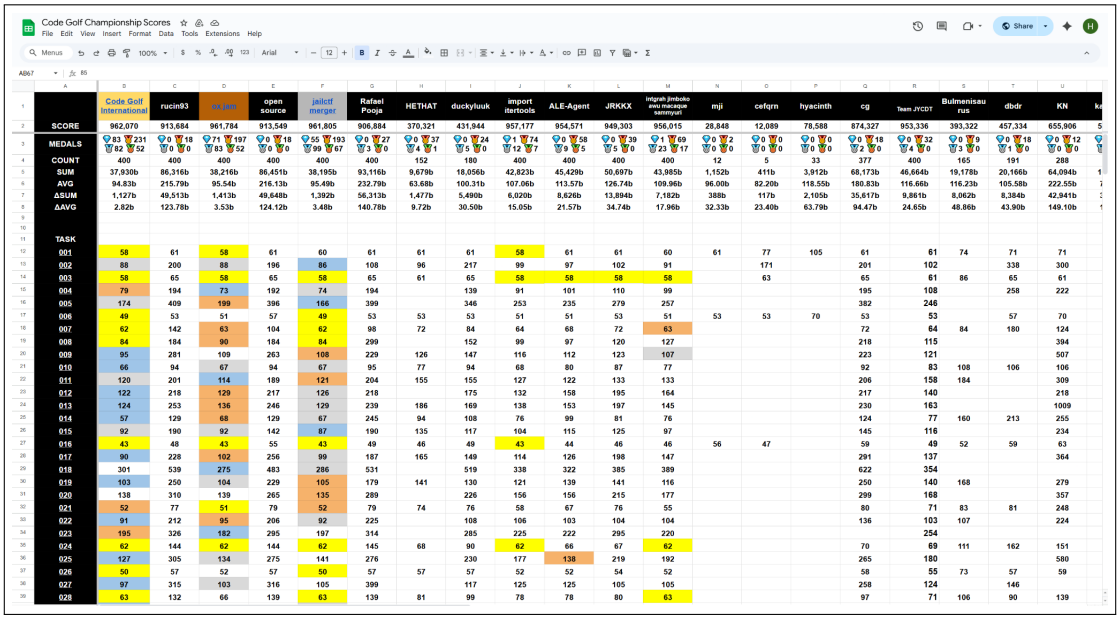



Figure 9: The (unofficial) community score spreadsheet initiated by 4atj.

three teams worth of people who had both the expertise and the time to really commit and see this contest through.

As for how Artificial Intelligence might eventually be able to match this performance (given that such a huge gap exists between human experts and other teams), it's worth noting that at conclusion of this competition, we now have probably one of the highest-quality code golf datasets that has ever existed. However, for anyone aiming to use these datasets in training AI, we would stress that (beyond the final solutions themselves) they should also consider the Git histories and their ability to track the improvement to solutions over time. This can help to shed light on which approaches turned out to be dead ends, aiding in the development of the aforementioned "golfer's sense" of not just good tricks, but also what approaches to avoid.

At the time of this writing, the ARC Prize 2025 also recently concluded, which is likely of interest to those who are curious about this golf competition. Among the resulting write-ups, the first place solution was extremely illuminating,[27] discussing issues such as (a) the creation of synthetic data to expand the available set of training inputs, (b) additional ideas on how such problems might be approached, and (c) the value of not just code solutions, but also natural language (i.e., English language) descriptions, and the huge amount of value that was able to provide. So, while AI largely struggles to perform well in golfing, it nevertheless exhibits a surprising ability to take these very ugly looking code snippets — which, to someone without a lot of experience in the field, might just look like gobbledygook — and provide an excellent explanation of what they mean, along with an exposition of how they work.

27. https://kaggle.com/competitions/arc-prize-2025/writeups/nvarc

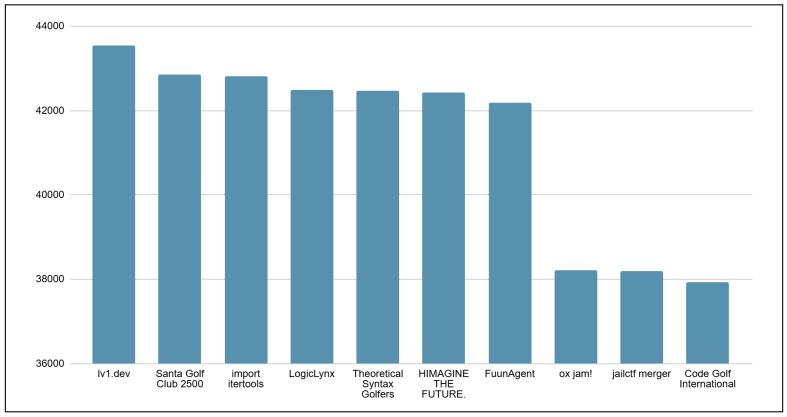
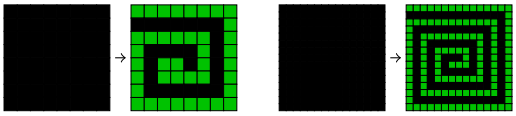



Figure 10: Total program lengths of the top ten teams (in bytes).

### B.3.7. Approaching Expert Intuition: Contextual Recall

Finally, coming back to the idea of expert intuition and a "golfer's sense," there is incredible importance to the idea of *contextual recall*. One such example that demonstrates how our team members were able to achieve top scores for these problems is Task #58, a relatively straightforward puzzle that (given a blank grid) requires the generation of a green spiral:



Even if someone knew all the golf tricks and could conceivably tackle this problem from scratch, what ultimately allowed one of our team members to find the shortest program length for this puzzle was his memory of solving a somewhat similar problem in a completely different context where he had golfed the following number spiral:[28]

```
 0  1  2  3  4  5  6  7  8  9
35 36 37 38 39 40 41 42 43 10
34 63 64 65 66 67 68 69 44 11
33 62 83 84 85 86 87 70 45 12
32 61 82 95 96 97 88 71 46 13
31 60 81 94 99 98 89 72 47 14
30 59 80 93 92 91 90 73 48 15
29 58 79 78 77 76 75 74 49 16
28 57 56 55 54 53 52 51 50 17
27 26 25 24 23 22 21 20 19 18
```

So, the ability to rely on contextual recall — i.e., to not just have a huge database of tricks, but to be able to answer the questions of *what is this similar to?*, *what is the general structure of this?*, and *what other problems does it look like?* — is probably huge in allowing both Artificial Intelligence *and* humans to escape local minima.

28. https://code.golf/number-spiral

## Appendix C. Official Competition Logo



## Appendix D. Task-Specific Statistics & Shortest Programs

In the pages that follow, we report final byte counts across all winning teams for each of the 400 puzzles in the ARC-AGI-1 training set, along with the shortest program submitted for every task. All solutions shown were produced by one of the top ten teams, with the following exceptions:

- cam jam! (11$^{\text{th}}$ place), who contributed a diamond solution to Task #278.
- Dark AGI (13$^{\text{th}}$ place), who contributed diamond solutions to Task #332, Task #349, and Task #354.
- Golfers Anonymous (17$^{\text{th}}$ place), who contributed a diamond solution to Task #277.
- Ravi Annaswamy (25$^{\text{th}}$ place), who contributed a diamond solution to Task #154.

Cell background colors represent golf medals earned — *diamond*, *gold*, *silver*, *bronze* — or program format (*uncompressed* vs. *compressed*).

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #001 | 007bbfb7 | 58 | 60 | 58 | 61 | 61 | 61 | 61 | 58 | 61 | 61 | p=lambda*g:[l*g.min,p][2](r,s)for r in g[0]for s in g[-1]]<br>"<<HH",),r"\1\4",f,{*zip(*g[147:]or p(g*2)),}:'))[::-1] |
| #002 | 00d62c1b | 88 | 86 | 88 | 98 | 107 | 98 | 94 | 99 | 94 | 95 | p=lambda g,i=67:g*-i or p([[r.pop()%sum(r[-i:],4)or i//67*4for r in g]for r in g],i-1)<br>,real*2)*1j**i+n].real*2)*1j**i+n:e[u]}.get(m*1j+f,a)for f↩<br>,a in enumerate(a)}for m,a in enumerate(a)}))for r in e];return a |
| #003 | 017c7c7b | 58 | 58 | 58 | 58 | 61 | 58 | 61 | 58 | 58 | 58 | p=lambda g:[[x*2for x in r]for r in g*g[g==2*g[:3:]|2:5]] |
| #004 | 025d127b | 79 | 74 | 73 | 86 | 84 | 85 | 80 | 91 | 92 | 104 | import re;p=lambda i:eval(re.sub(r'(.),(?=.*\1.*\1)',r"0,\1|",str(i))) |
| #005 | 045e512c | 174 | 166 | 199 | 234 | 234 | 232 | 246 | 253 | 241 | 221 | import re;p=lambda g:eval(re.sub(f"0(?=(.)+{'(.'*(.f2%%d)+(?<=%s))"%max(str(29**14),key=f'{g}'.count)*2)|"*2%(*b↩<br>"<<HH",),r"\1\4",f,{*zip(*g[147:]or p(g*2)),}:'))[::-1] |
| #006 | 0520fde7 | 49 | 49 | 51 | 51 | 49 | 53 | 51 | 51 | 50 | 49 | p=lambda g:[eval('r.pop(0)*r[3]*2','*3)for r in g] |
| #007 | 05269061 | 62 | 62 | 63 | 64 | 64 | 68 | 64 | 64 | 68 | 64 | p=lambda g:eval(f'[{"max(sum(g:=g[1:3]+g,[])[2::3]),"*7},'*7} |
| #008 | 05f2a901 | 84 | 84 | 90 | 125 | 99 | 97 | 101 | 99 | 111 | 114 | p=lambda g,*x:sorted(zip(*m or p([],*g)),key=lambda r:g.extend(r)or(8in g)*3^any(r)) |
| #009 | 06df4c85 | 95 | 108 | 109 | 113 | 116 | 108 | 108 | 116 | 113 | 112 | p=lambda g,*r,i=0:[x|max({*g[i::3]}&{*g[:(i:=i+1)]})for x in r]or[*map(p,g,*map(p,zip(*g),*g))] |
| #010 | 08ed6ac7 | 66 | 67 | 67 | 71 | 71 | 68 | 70 | 68 | 72 | 71 | p=lambda g:[g:=[x*-5or y*sum(x))%6for x,y in zip(g,r)]for r in g] |
| #011 | 09629e4f | 120 | 121 | 114 | 141 | 140 | 124 | 137 | 127 | 122 | 141 | p=eval("lambda a:max(a*(not'8'in'%s'%a)"+f"for*a,in[*map(zip,a,a,a{',[a[3]]*9,*[a[%d:]]*3'*2%(1,2)}][::4]"*2+')↩<br>')<br>%'*2][-1]) |
| #012 | 0962bcdd | 122 | 126 | 129 | 142 | 133 | 135 | 143 | 132 | 144 | 135 | import re;p=lambda g:eval([g:=re.sub(r"0(?=.{%d}(['0]).*(.).*\1)"%x,f"\{895%x%5}",str(g)[::-1])for x in b'HH⊠NB⊠+↩ |
| #013 | 0a93d479 | 124 | 129 | 136 | 139 | 150 | 156 | 136 | 138 | 160 | 163 | p=lambda g,h=0,l=[]:[(l:=[max(c+[j*(i>0)for i,j in zip(l,l[1::2])])+1)for c in g if or p(g,g)] |
| #014 | 0b148d64 | 57 | 67 | 68 | 81 | 76 | 74 | 92 | 76 | 74 | 74 | p=lambda g:[p(zip(r,*g))or r[0]for r in[*g]if{*r}[2:]] |
| #015 | 0ca9ddb6 | 92 | 87 | 92 | 105 | 116 | 106 | 112 | 104 | 102 | 119 | p=lambda g,i=-7:g*i or p([[r.pop()%9]36%(6-[0,*r][i//7])%13for r in g]for*r,in g],i+1) |
| #016 | 0d3d703e | 43 | 43 | 43 | 46 | 46 | 47 | 46 | 43 | 44 | 46 | p=lambda g:[[x**10%95%18*4for x in g[0]]*3 |
| #017 | 0dfd9992 | 90 | 99 | 102 | 118 | 109 | 94 | 107 | 114 | 104 | 144 | p=lambda g:[[*map(max,*[r*any(-i^-j>0for i,j in zip(r,s)>s for s in g])]for*r,in zip(*g)] |
| #018 | 0e206a2e | 301 | 286 | 275 | 299 | 311 | 342 | 346 | 338 | 347 | 332 | def p(a):e=[m*1j+f:a for m,a in enumerate(a)for f,a in enumerate(a)if a};[(i:=[r],[i:={r}|i for r in[*e]*5for u ↩<br>in l if abs(r-u)<2],[*l][3:]and[5for i in[1,3,6,7]for n in e if all(sum(e[u]==e.get(f)for f in[*l,(u-r-i//4*(u-r↩ |
| #019 | 10fcaaa3 | 103 | 105 | 104 | 121 | 119 | 119 | 121 | 121 | 120 | 134 | p=lambda g,n=7:-n*g or p(-~(n>5)*[g:=[r.pop()or(x*-1or 0)%-688for x in[0]+g[:-1]]for*r,in zip(*g)],n-1) |
| #020 | 11852cab | 138 | 135 | 139 | 138 | 152 | 126 | 126 | 156 | 144 | 133 | f=lambda g:g.index(max(g,key=any));p=lambda g,k=0:exec("n=f([*zip(*g)]);g[k%6+f(g)[k%5+n]=g[f(g)+~k%5][k%6+n];↩<br>k+=1;"*73)or g |
| #021 | 1190e5a7 | 52 | 52 | 51 | 54 | 52 | 66 | 51 | 58 | 55 | 54 | p=lambda i:i*-1*-1or-min(map(i.count,i))*[p(i[0])] |
| #022 | 137eaa0f | 91 | 92 | 95 | 97 | 106 | 102 | 101 | 106 | 105 | 99 | r=-1,0,1;p=lambda g:[dict(sorted(zip(o:=sum(g,[]),o[x*11+y:]+o)))[5]for y in r]for x in r] |
| #023 | 150deff6 | 195 | 197 | 182 | 206 | 250 | 227 | 191 | 225 | 227 | 208 | import re;p=lambda i,w=2:s!=(r:=re.sub((w%2*"5, "+"5(.%s)??")%(w*3%-7%len(i[0]*3)+2)*(3-w%2),r" 82,\81\ 12 \82, ↩<br>82"[w::2],s,i))and p(eval(r))or w and p(i,w-1)if"5"in(s:=str(i))else i |
| #024 | 178fcbfb | 62 | 62 | 62 | 62 | 62 | 65 | 62 | 62 | 62 | 62 | p=lambda g:[[3%-~max(x)or(21in x)*2for x in zip(*g)]for r in g] |
| #025 | 1a07d186 | 127 | 141 | 134 | 144 | 156 | 143 | 138 | 177 | 153 | 192 | p=lambda g,N=0:[g:=[[any(min(g)+[I:=(x:=r.pop())in r,A:=all(c))*x|N+(N:=A*4*x)for c in g[::-1]]for*r,in zip(*g)↩<br>]for _ in g][3] |
| #026 | 1b2d62fb | 50 | 50 | 52 | 52 | 50 | 54 | 52 | 52 | 52 | 50 | p=lambda g:[eval('8>>r.pop(0)*r[3],'*3)for r in g] |
| #027 | 1b60fb0c | 97 | 105 | 103 | 104 | 113 | 113 | 103 | 125 | 107 | 105 | R=range(10);p=lambda g:[[g[i][j]or 2*g["j+(I*["*zip(*g)][5]]<g[5][::-1]][i]for j in R]for i in R] |
| #028 | 1bfc4729 | 63 | 63 | 66 | 70 | 70 | 71 | 64 | 78 | 70 | 70 | p=lambda g:[[w:=max(g['%15)),*{y%w}*8,w]for y in b'/ / pp⊠p⊠']] |
| #029 | 1c786137 | 103 | 108 | 110 | 113 | 117 | 121 | 141 | 121 | 117 | 142 | p=lambda g,w=9:g*w and p(g,w-1)+[g:=[r[~x::-1]for*r,in zip(*g)]if w in r]for x in[0]*4+[1]*5][7]*([]==g) |
| #030 | 1caeab9d | 93 | 93 | 94 | 116 | 93 | 109 | 116 | 94 | 103 | 116 | p=lambda g:[*zip(*map(lambda*r:r[(k:=(f:=sum(g,[]).index)(max(r))//10-f(i))//10]:i+r[:k],*g))] |
| #031 | 1cf80156 | 45 | 45 | 45 | 48 | 50 | 48 | 50 | 50 | 48 | 48 | p=lambda g,*e:[*filter(any,zip(*e or p(*g)))] |
| #032 | 1e0a9b12 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | p=lambda g:[*zip(*map(sorted,zip(*g)))] |
| #033 | 1e32b0e9 | 71 | 73 | 73 | 77 | 79 | 79 | 77 | 79 | 79 | 77 | p=lambda g,h=[],i=0:g*0!=0and[*map(p,g[:6]*3,h*g,g[5:6]*17)]or(g>h)*i|h |
| #034 | 1f0c79e5 | 116 | 126 | 125 | 182 | 143 | 150 | 128 | 127 | 152 | 175 | import re;p=lambda g,n=-35:n*g or eval(re.sub("(['02]), 2(.{28})0",r"*[\1]*3\2\1,2",f"{*zip(*p(g,n+1)),*g}"))[8:↩<br>:-1] |
| #035 | 1f642eb9 | 82 | 83 | 82 | 91 | 91 | 94 | 94 | 92 | 91 | 89 | p=lambda g:[[50:]or[g:=i!=8or g*0]or e for e in l]for*l,in zip(*p(g*2))][::-1] |
| #036 | 1f65a75f | 75 | 75 | 82 | 109 | 95 | 115 | 112 | 95 | 90 | 81 | p=lambda g,*G:[f for*f,in zip(*G or p(g,*g))if{*f}-{*sum(g[:5]+g[-4:],[])}] |
| #037 | 1f876c06 | 101 | 105 | 104 | 115 | 112 | 109 | 109 | 111 | 115 | 117 | import re;p=lambda g:[:-99]or p(eval(re.sub(r"(?<=(.).{34})(?=(.{35})*\1)0",r"\1",str(g[9::-1]))+g) |

| TASK # | TASK ID | | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #038 | 1fad071e | | 51 | 51 | 51 | 51 | 51 | 51 | 51 | 51 | 51 | 51 | p=lambda g:[(str(g).count("1, 1")*[1]+9*[0])[:9:2]] |
| #039 | 2013d3e2 | | 60 | 60 | 60 | 61 | 64 | 65 | 61 | 61 | 60 | 60 | p=lambda g:[*eval("zip(*[*filter(any,"*2+")))][:3])")][:3] |
| #040 | 2204b7a8 | | 63 | 69 | 69 | 75 | 70 | 75 | 75 | 72 | 75 | 75 | p=lambda g,h=[]:g*0!=0and[*map(p,g[:1]*5+g[9:]*5,h+g)]or h%"h&g |
| #041 | 22168020 | | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | p=lambda g,a=0:[[v|(a:=a^v)]for v in r]for r in g] |
| #042 | 22233c11 | 136 | 132 | 134 | 158 | 138 | 132 | 162 | 139 | 140 | 235 | | import re;p=lambda g:exec("s=str(g);g[::-1]=zip(*eval(re.sub('0(?=.{%r}3)','(*b'%(q:=9-r),s.count('3')//8::3]↩<br>,),'8',s))),"*4)or g |
| #043 | 2281f1f4 | | 56 | 56 | 56 | 57 | 59 | 57 | 57 | 57 | 57 | 57 | p=lambda g:[[x+r[-1]&r.pop(0)+2for x in g[0]]for*r,in g] |
| #044 | 228f6490 | 201 | 195 | 212 | 262 | 223 | 269 | 286 | 239 | 257 | 290 | | def p(l):<br>for f in range(100):<br>r=[n for n in range(100)if i[n//10][n%10]<5in i[n//10][n%10]and 5in i[n//10][n%10:]and n<50]<br>t=[n for n in range(100)if i[n//10][n%10]==f]<br>if[r[0]-f for f in r]==[t[0]-f for f in t]:<br>for n in r:i[n//10][n%10]=f<br>for n in t:i[n//10][n%10]=0<br>for f in range(100):<br>r=[n for n in range(100)if i[n//10][n%10]<5in i[n//10][n%10]and 5in i[n//10][n%10:]and n>50]<br>t=[n for n in range(100)if i[n//10][n%10]==f]<br>if[r[0]-f for f in r]==[t[0]-f for f in t]:<br>for n in r:i[n//10][n%10]=f<br>for n in t:i[n//10][n%10]=0<br>return l |
| #045 | 22eb0ac0 | | 45 | 45 | 45 | 45 | 46 | 46 | 45 | 45 | 45 | 45 | p=lambda g:[(r*10)[::10]len({*r})]for r in g] |
| #046 | 234bbc79 | | 166 | 168 | 168 | 191 | 181 | 171 | 175 | 192 | 202 | 192 | def p(g,d=3):g=(5,),*zip(*g);return*zip(*[sum((*x*(q+r+s)-(5))for x in 3*r][a:3+d]for q,r,s in zip(g,g[1:],g[2:]+g)if any(r)or(d:=d-[*q,5].index(5)+s.index(5)*0)], |
| #047 | 23581191 | | 55 | 55 | 55 | 56 | 56 | 55 | 56 | 55 | 55 | 55 | p=lambda g:[[sum((*r+c)%13for*c,in zip(*g))]for r in g] |
| #048 | 239be575 | | 95 | 92 | 118 | 129 | 125 | 134 | 129 | 128 | 123 | 118 | p=lambda g:[[hash(('b'+]□ d□B□Px <la□AacF#□p3e7"□□kz□□O□□□W}&&N□□%r'[sum(b'□□%r'%g)%39:]%g,))&8]] |
| #049 | 23b5c85d | | 81 | 81 | 81 | 81 | 81 | 87 | 81 | 87 | 82 | 81 | p=lambda g:[r*[v]for l in g if(r:=l.count(v:=min(r:=sum(g,[0]*99),key=r.count)))] |
| #050 | 253bf280 | | 84 | 83 | 84 | 90 | 92 | 97 | 90 | 100 | 97 | 91 | p=lambda g,x=0:[[c|((x:=~sum(r)&c~x)>6>c)*3for c in r]for*r,in zip(*x*g or p(g,1))] |
| #051 | 25d487eb | | 114 | 113 | 111 | 139 | 121 | 124 | 115 | 115 | 123 | 119 | p=lambda i:['*eval("map(lambda*x,l=0,b=1,a=1:[[l:=l|(b!=y>a<1)*(a:=b),b:=y][y>0]for y in x)[::-1],*"*4+"i)))"] |
| #052 | 25d8a9c8 | | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | p=lambda g:[[len({*r})%2*5]*3for r in g] |
| #053 | 25ff71a9 | | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | p=lambda g:(2*g)[2:5] |
| #054 | 2643631d | | 266 | 279 | 270 | 302 | 298 | 286 | 277 | 309 | 337 | 335 | def p(r):<br>f=lp*1for p in r]<br>for p in range(28):<br>for t in range(28):<br>if r[p][t+1]==r[p][t-1]!=r[p][t]==r[p-1][t]!=f[p][t+1],r[p+1][t+1]!=f[p+1][t]:<br>for e in range(28):<br>for n in range(28):<br>if r[p][t]==r[e][n]:<br>for o in range(-1,2):<br>for h in range(-1,2):<br>i=1<br>if r[p+o*i][t+h*i]!=r[p+1][t+2]:<br>if[e+o*i][n+h*i]=r[p+o][t+h];i+=1<br>if r[p][t]!=r[p+o*i][t+h*i]:r[p+1][t+2]:<br>while r[e+o*i][n+h*i]!=r[p+1][t+2]:f[e+o*i][n+h*i]=r[p+o][t+h];i+=1<br>for o in range(-2,3):<br>for h in range(-2,3):f[p+o][t+h]=r[p+1][t+2]<br>return f |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #055 | 272f95fa | 79 | 79 | 77 | 83 | 89 | 94 | 82 | 85 | 93 | 91 | p=lambda i,z=0:i*0!=0and[p(y,3*(z:=z+(y[y>i]))[y>i])for y in i]or i or 2222096>>z&7 |
| #056 | 27a28665 | 40 | 40 | 40 | 44 | 40 | 41 | 39 | 42 | 40 | 40 | p=lambda g:[[2-(g[2]<[1])*3+(g<[[1]])]] |
| #057 | 28bf18c6 | 48 | 48 | 49 | 51 | 54 | 51 | 54 | 53 | 51 | 51 | p=lambda g:[*filter(any,zip(*g[8:]or p(g*2)*2))] |
| #058 | 28e73c20 | 86 | 88 | 85 | 89 | 90 | 103 | 99 | 103 | 99 | 102 | p=lambda a:a and[len(a)*[3],[*a[0],3>>2%len(a),3][2:],*zip(*p(zip(*a[2:]))[::-1])][::-1] |
| #059 | 29623171 | 151 | 155 | 153 | 163 | 169 | 174 | 163 | 178 | 166 | 173 | R=range(11);p=lambda g,n=36,o=0:[[g[i][j]==5)*6or(sum(S:=g[i&12|k%3][j&12|k//3]for k in R[:9])>n!=[o:=1])*max(S)for j in R]for i in R]*o or p(g,n-1) |
| #060 | 29c11459 | 47 | 47 | 47 | 48 | 48 | 48 | 47 | 47 | 47 | 48 | p=lambda g:[r[:1]*5+[3%~i%6]+[1]*5for*r,i in g] |
| #061 | 29ecfd0e | 63 | 62 | 63 | 62 | 63 | 63 | 62 | 63 | 63 | 63 | p=lambda g,q=range(18):[[i*3%max(g)[l]+1if j in q[:l]else 0for j in q]for i in q] |
| #062 | 2bcee788 | 123 | 127 | 121 | 162 | 145 | 146 | 185 | 154 | 144 | 140 | p=lambda g:exec("q=[]\nfor r in zip(*g):q+=q[:`0`*(*r)^{2}==3}<{*q[~1:]}or[[v or 3for v in r]]\ng[:]=q[9::`0`]+';'*8)or g |
| #063 | 2bee17df | 74 | 76 | 75 | 77 | 79 | 78 | 77 | 78 | 79 | 79 | p=lambda g:[[x!=3>>x+sum(r[1:]-any(c))]for-,*c,-,x in zip(*g,r)]for r in g] |
| #064 | 2c608aff | 135 | 146 | 152 | 174 | 164 | 151 | 169 | 155 | 170 | 154 | p=lambda g,n=-7:n*g or p([[P:=[x:=r.pop(),P][r.count(max((x*x*P+(g[0].count(x)>3),n)-(x,P)))>2]for _ in g]for*r,i+b n zip(*g)if[P:=0][],n*2) |
| #065 | 2dc579da | 82 | 84 | 84 | 102 | 107 | 105 | 102 | 109 | 103 | 106 | p=lambda*g:min(g,key=g.count)if g[3:]else[*map(p,*g,*[h[len(h)//2+1:]for h in g])] |
| #066 | 2dd70a9a | 240 | 237 | 244 | 266 | 282 | 257 | 250 | 250 | 301 | 327 | def f(r,n,e,l,d):<br> if len(r)>n+1>1>r[e][n]%3:r[e][n]=3;return f(r,n+d,e,l,d)<br> if r[e][n]%3:return f([[*n]for n in zip(*r)],e,n-d,l-1,1)or f([[*n]for n in zip(*r)],e,n-d,l-1,-1)if l else(3>r+b[e][n])*r<br>def p(r):(n,e),(l,d)=[(n,e)for n in range(len(r))for e in range(len(r))if r[e][n]%2];return f([[*n]for n in r],n+b,e,2,1)or f(r,n,e,2,-1)if l>n else[[*n]for n in zip(*p([[*n]for n in zip(*r)]))] |
| #067 | 2dee498d | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | p=lambda g:[r[:len(g)]for r in g] |
| #068 | 31aa019c | 99 | 112 | 103 | 122 | 129 | 117 | 116 | 128 | 119 | 122 | p=lambda g,x=0:[[(x:=x*212>>sum(g,g).count(y))%2*y|(x>>898734515&9)*2for y in r]for*r,in g*2][10:] |
| #069 | 321b1fc6 | 148 | 143 | 150 | 158 | 168 | 147 | 155 | 181 | 161 | 175 | def p(g):<br> *b,=a=sum(g,[]);i=j=100<br> while i:i-=1;a[i]==8==exec("if b[j:=j-1]%8:g+=j,;a[i-g[10]+j]=b[j];a[j]=0\n"*j)<br> return*zip(*[iter(a)]*10), |
| #070 | 32597951 | 80 | 78 | 82 | 100 | 92 | 89 | 112 | 87 | 100 | 89 | p=lambda g:[[max((*i,hash(i)%1070)&(*r))%5**i[0]for i in zip(r,*g)]for r in g] |
| #071 | 3345333e | 115 | 118 | 104 | 118 | 133 | 123 | 100 | 119 | 121 | 115 | p=lambda g,a=0:[[a*(v>0<r[~i+t])for i,v in enumerate(r)if a or{t:=i-r[::-1].index(a:=v)}]for r in g] |
| #072 | 3428a4f5 | 54 | 54 | 54 | 60 | 62 | 62 | 60 | 60 | 54 | 54 | p=lambda g,u=[]:g*0!=0and[*map(p.g,u+g[7:])]or(g!=u)*3 |
| #073 | 3618c87e | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | 46 | p=lambda g:3*g[:1]+g[3],[5^n*4for n in g[2]] |
| #074 | 3631a71a | 77 | 77 | 77 | 102 | 81 | 80 | 89 | 99 | 82 | 112 | p=lambda g:g[90:]or p(g+[*zip(*map(map,[min]*30,p(g[:30]+g),g[:21+g[::-1]]))]) |
| #075 | 36342ee | 86 | 86 | 86 | 86 | 88 | 86 | 86 | 86 | 86 | 86 | R=range(9);p=lambda g:[g[i][:4]+g[i-1-x%3+1][x-x%3+5]+g[i%3][x%3]for x in R]for i in R] |
| #076 | 36d67576 | 258 | 264 | 269 | 279 | 286 | 286 | 328 | 320 | 329 | 270 | def p(r):<br> a=r<br> for n in range(len(r)):<br> for f in range(len(r[0])):<br> if a[n][f]==1:z={(f+l,e+n)for l,n in z for f in range(-1,2)for e in range(-1,2)if e+n in range(len(r))!=f+l in range(len(r[0]))!=0<r[e+n][f+l]}<br> for n in r:z={(f+l,e+n)for l,n in z for f in range(-1,2)if e+n in range(len(r))!=f+l in range(len(r[0]))!=0<r[e+n][f+l]}<br> for n in(1,1,-1)*4:<br> r=[z for*z,in zip(*r[::n])]<br> for e in range(-13,13):<br> for f in range(-13,13):<br> if all(e+n in range(len(r))!=f+l in range(len(r[0]))!=a[n][l]in(1,3,r[e+n][f+l])for l,n in z):<br> for l,n in z:r[e+n][f+l]=a[n][l]<br> return r |
| #077 | 361dfd69 | 124 | 124 | 110 | 135 | 128 | 151 | 125 | 126 | 130 | 130 | p=lambda i,k=7,*w:k and p(*map(p,i,[k>1]*99,[i*2]+i,i[1:]+[i*2],*w)],k-1)or((c:=w.count(2)*c(4)>=2!=1)*4or i |
| #078 | 3906da3d | 60 | 60 | 59 | 60 | 62 | 61 | 60 | 61 | 61 | 60 | p=lambda i,*n:sorted(n,key=0 .__eq__)or[*zip(*map(p,i,*i))] |
| #079 | 39a8645d | 107 | 121 | 105 | 123 | 123 | 121 | 123 | 123 | 123 | 123 | p=eval(f"lambda a:max(b:=[a`for*a,in map(zip,a,a[1:],a[2:])*2]if any(min(*a,*zip(*a))),key=b.count)") |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #080 | 39e1d7f9 | 232 | 243 | 231 | 240 | 255 | 239 | 253 | 256 | 235 | 256 | `def p(f):n=f.index(min(f,key=set))+1;e={u*1j+o:f for u,f in enumerate(f[::n])for o,f in enumerate(f[::n])};retur↩ n[[[f or[e[t:=u/n*1j+o//n],*e[a+t-r]for r in e if e[r]==e[a]*2>abs(t-r)][-1]for o,f in enumerate(f)]for u,f in ↩ in enumerate(f)]for a in e if all(e.get(a+1j**u)for u,f in enumerate(f))][-1]` |
| #081 | 3aa6fb7a | 90 | 81 | 83 | 96 | 95 | 100 | 95 | 96 | 95 | 91 | `p=lambda g:exec(f"g[:]=t(q:=i)*[q:=r.pop()or[1]<r[-1:q]for r in g],'*7};"*4)or g` |
| #082 | 3ac3eb23 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | `p=lambda g:[r:=g[0],[*map(max,[0]+r,r[1:]+[0])]]*3` |
| #083 | 3af2c5a8 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | `p=lambda g:[r+r[::-1]for r in g+g[::-1]]` |
| #084 | 3bd67248 | 62 | 62 | 62 | 64 | 63 | 62 | 62 | 63 | 62 | 62 | `def p(g,i=0):`<br>`  *h,l=g`<br>`  for r in h:i-=1;r[i]=2;l[i]=4`<br>`  return g` |
| #085 | 3bdb4ada | 49 | 49 | 49 | 62 | 56 | 57 | 59 | 81 | 58 | 49 | `p=lambda g:g*0!=0and[g:=[p(r),r][r!=g]for r in g]` |
| #086 | 3befdf3e | 166 | 154 | 151 | 206 | 211 | 193 | 221 | 224 | 194 | 226 | `p=lambda i,k=7,s=0:-k*i or[[[-(s:=[abs(s)or 1,s&s/4][y>0]-1)>1)(y,*[x for x in sum(i,x)if 0<x!=y!=0],0][k>6]fo↩ r y in x]for*x,in zip(*p(i,k-1)[::-1])]` |
| #087 | 3c9b0459 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | `p=lambda g:[r[::-1]for r in g[::-1]]` |
| #088 | 3de23699 | 98 | 95 | 100 | 110 | 106 | 120 | 106 | 110 | 101 | 105 | `exec(f"def p(g):{'g[:]=zip(*g[any(h:=g[-1])-2::-1]);'*48}return[h[g<1]*'+'for g in g][1:-1]'*2)` |
| #089 | 3e980e27 | 233 | 233 | 232 | 251 | 243 | 254 | 260 | 279 | 279 | 277 | `def p(i):`<br>`  for n in(e:=[n*1j+a:i for n,i in enumerate(i)for a,i in enumerate(i)if i]):`<br>`  for r in e:`<br>`  t=[r]`<br>`  for a in[*e]*3:`<br>`  if e[n:=e[r]!=any(0<abs(a-r)<2for r in e)<any(0<abs(a-r)<2for r in t):t|={a};i[int((a-r+n).imag)][int((-(-1+↩ )**e[r]*(a-r)+n).real)]=e[a]`<br>`  return i` |
| #090 | 3eda0437 | 147 | 158 | 153 | 184 | 182 | 155 | 190 | 190 | 192 | 175 | `import re;p=lambda g:eval('6'.join(max([re.split(f'(..f{{len(g[0])*3-i%8*3}})0{i%8*"( )0"}'*(i%5+2))[8:],str(↩ g),1)for i in range(40)],key=len))` |
| #091 | 3f7978a0 | 62 | 62 | 62 | 69 | 63 | 90 | 64 | 63 | 64 | 63 | `p=lambda g,k=46:'k*g or p(['zip(*g[(5in g[k|-2])-2::-1])],k-1)` |
| #092 | 4085293 | 86 | 87 | 86 | 94 | 100 | 92 | 95 | 101 | 99 | 101 | `p=lambda g:['map(F:=lambda*l,c=0:[l.count(e)>1and(c:=c^e)or e for e in l],*map(F,*g))]` |
| #093 | 4093f84a | 98 | 98 | 98 | 108 | 118 | 117 | 102 | 100 | 113 | 100 | `import re;p=lambda g:[5730:]or eval(re.sub("[^05],([0,]*5)","r"\1,5","f"{*zip(*p(2*g)[::-1]),}")` |
| #094 | 41e4d17e | 102 | 96 | 97 | 124 | 102 | 131 | 111 | 115 | 101 | 130 | `import re;p=lambda g,x=0:eval(re.sub("8(?=[^(]**[^)]*1.f46)1, 1)","6",f'{*zip(*x or p(g,g))},')` |
| #095 | 4258a5f9 | 70 | 70 | 74 | 79 | 77 | 73 | 97 | 79 | 83 | 85 | `p=lambda g:g[99:]or[eval("r.pop()!r"[-1]%4,"*9)for*r,in zip(*p(g*2)*2)]` |
| #096 | 4290ef0e | 284 | 282 | 282 | 298 | 330 | 299 | 340 | 309 | 301 | 300 | `def p(a):u={0:(0,i:=max(n:=re.sub(' ','',f'\n,*zip(*n))+'),key=n.count)}[len(max(re.findall(f'{1}+',n))*(((r:↩ =len(re.findall(f'{1}{1}({"|".join(f"{1}"*i for i in range(2,r))}){1}|$',n+n[::-1])[0]))>0)+1)+r>1:(1+r>1,1)for 1 in[*n]-{*'[]',1}];return↩ [[int(lu[max(abs(l),abs(r))][1],1]lu[max(abs(l),abs(r))][0]>min(abs(l),abs(r))]for l in range(-max(u),max(u)+1)↩ ]for r in range(-max(u),max(u)+1)]` |
| #097 | 42a50994 | 96 | 98 | 93 | 110 | 113 | 110 | 108 | 108 | 110 | 108 | `p=lambda i,*w:i*0!=0and[*map(p,*sum([[x,x[1:]+[i*4],[i*4]+x]for x in[i,*w]],[]))]or(i in w)*i` |
| #098 | 4347f46a | 74 | 72 | 64 | 88 | 96 | 98 | 96 | 99 | 97 | 106 | `p=lambda i,*w:i*0!=0and[*map(p,i,i[:1]+i,i[1:]+i,*w)]or i^min(w)` |
| #099 | 44801d8 | 93 | 112 | 103 | 142 | 122 | 122 | 126 | 134 | 126 | 134 | `p=lambda g,n=0:[g:=[r.pop()or 1in g]*x>0and~sum(i*g)for x in g]for r in(n or p(g,g))[::-1]` |
| #100 | 445eab21 | 85 | 88 | 88 | 87 | 88 | 94 | 86 | 95 | 94 | 94 | `p=lambda g:[max(y*[r.count(y)*c.count(y)for r in g for*c,y in zip(*g,r)][-1:]*2]*2` |
| #101 | 447fd412 | 271 | 265 | 258 | 275 | 285 | 297 | 290 | 326 | 340 | 307 | `def p(e):`<br>`  i,r,o,u,n=[{a+m*2for m,e in enumerate(e)for a,e in enumerate(e)if e==f}for f in range(5)]`<br>`  for f in o:u|=f for f in o for m in r|u if abs(f-m)in(1,20)}`<br>`  for f in o:b={f}-n and{f-min(len({f+m*(s-2)for s in range(5)}&o)for m in(1,20))*(min(u)-max(u))}&o;n|=b;e=[[e o↩ r a+m*20in(b and{f-min(len({f+m*(s-2)for s in range(5)}&o)for m in(1,20))*(min(u)-m)for m in r}for a,e in enume↩ rate(e)]for m,e in enumerate(e)]`<br>`  return e` |
| #102 | 4d48ac46 | 126 | 124 | 135 | 161 | 183 | 186 | 150 | 148 | 174 | 194 | `p=lambda g,k=-11,j=1,q=0:k*g or[[q:=[i:=i<<9,2&132132>>c%511,5,c|q,0][k*6%64%6+c%~c]for c in g]for g[:-1]in zip↩ (*p(g,k+1))]` |
| #103 | 44f52bb0 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | `p=lambda g:[[g[0]=g[2]or 7]]` |
| #104 | 45220011 | 84 | 84 | 84 | 89 | 96 | 88 | 89 | 94 | 94 | 89 | `p=lambda g:[[*lu%4]*4+[u%5]*4,0][::11l-g[1][2]]for u in b"###0000(")[::11l-g[2][1]]` |
| #105 | 4612d853 | 134 | 124 | 139 | 161 | 175 | 153 | 155 | 126 | 152 | 214 | `p=lambda g,i=11,k=0:-i*g or[[c*(k:=b|(1in r)or(sum(map(bool,r))-b>2>i%6)*2for c in r]for r in zip(*p(g,i-1)[::-↩ 1])if[b:=k]]` |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #106 | 46442a0e | 67 | 59 | 55 | 66 | 67 | 74 | 66 | 67 | 66 | 66 | p=lambda i,s=[],k=3:-k*i or p([*zip(*i+s)],i[::-1],k-1)<br>def p(l):z=len({*str(i)})-5;r=range(5*z);return[l[x//z][y//z]or(x-z*0**i[0][1]in[u:=y-z*0**i[0][z*2+~u]*2for↩ |
| #107 | 46949fad | 161 | 142 | 132 | 149 | 205 | 154 | 137 | 144 | 136 | 137 | r y in r]for x in r] |
| #108 | 46f33fce | 52 | 52 | 46 | 46 | 59 | 62 | 46 | 64 | 57 | 63 | p=lambda a:a>a*0!=0and[p(a[1])]*4+p(a[2:])or a |
| #109 | 47c1f68c | 80 | 80 | 77 | 91 | 81 | 91 | 91 | 90 | 82 | 90 | p=lambda a,s=0:a*0!=0and(b:=[*map(p,a,[a[1:=len(a)//2]]*1)])+b[::~0]or a%~a&s |
| #110 | 484b58aa | 84 | 85 | 84 | 85 | 99 | 88 | 83 | 114 | 86 | 102 | p=lambda g:[[*map(max,*(r*max(-a~-b for a,b in zip(r,t))+t for t in g))]for r in g] |
| #111 | 4848fb45 | 60 | 60 | 60 | 61 | 62 | 70 | 61 | 60 | 61 | 60 | p=lambda g:[(l:=sum(g,g))[l.index(5)+n:][:3]for n in b" ⊠⊠"] |
| #112 | 4938fdc2 | 91 | 91 | 106 | 101 | 103 | 101 | 134 | 106 | 123 | 126 | p=lambda g:exec('g[::]=zip(*map(M:=max,g,(g*3)[2*g.index(M(g,key=M))-~len(g)::-1]));'*2)or g |
| #113 | 496994bd | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | p=lambda g:g[:5]+g[4:::-1] |
| #114 | 4941d64f | 64 | 64 | 64 | 65 | 65 | 65 | 65 | 64 | 65 | 65 | *z,p=0,lambda g:[2+g[0]+2,*[r[:1]+r+r[-1:]for r in g],2+g[-1]+2]<br>+1) |
| #115 | 4be741c5 | 51 | 52 | 52 | 51 | 71 | 63 | 66 | 70 | 53 | 66 | p=lambda g,F={}.fromkeys:[*F(zip(*zip(*map(F,g)))))] |
| #116 | 4c4377d9 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | p=lambda g:g[::-1]+g |
| #117 | 4c5c2cf0 | 115 | 147 | 130 | 179 | 184 | 170 | 175 | 172 | 148 | 177 | p=lambda g,n=-79:n*g or p(["".join(map(max,g,i:=(g*3)[n%-21::-1]))][40in map(str(i+6*h).count,str(h))]],n↩ |
| #118 | 50846271 | 245 | 244 | 225 | 270 | 267 | 277 | 257 | 280 | 275 | 276 | def p(u):<br>for f in(2,3):<br>a,e,n,*r=[(s-j*1jfor j,u in enumerate(u)if u>=f)for f in(2,0,5,6)]<br>for j in e:i={s for s in e if abs(s-j)in(2,0,1,f)};r+=[{i for f in r if a-f>i}<br>for f in r:<br>if a-n<f:return[[u+3*(s-j*1jin f&n)for s,u in enumerate(u)]for j,u in enumerate(u)] |
| #119 | 508bd3b6 | 102 | 102 | 102 | 107 | 111 | 114 | 112 | 107 | 107 | 102 | import re;p=lambda g:exec('g[::-1]=zip(*eval(re.sub("0(?=.{40}[38].{40}[238])","3",str(g))));'*40)or g |
| #120 | 50cb2852 | 89 | 85 | 78 | 106 | 99 | 108 | 105 | 99 | 100 | 106 | p=lambda i,*w:i*0!=0and[*map(p,i,[i*2]+i,i[1:]+[i*2],*w)]or[8,i]["0"in str(w)] |
| #121 | 5117e062 | 89 | 89 | 88 | 89 | 102 | 90 | 85 | 90 | 87 | 89 | p=lambda g:(a:=(g:=sum(g,[]))[g.index(8)-14:][:3],(g[13],max(a),g[15]),g[26:29]) |
| #122 | 5168d44c | 70 | 75 | 71 | 89 | 90 | 81 | 80 | 78 | 89 | 89 | p=lambda g:'3, 0'in'%s'%max(g)and[*map(p,g]or min(g,g[4:])[:2]+g[:-2] |
| #123 | 539a4f51 | 72 | 75 | 75 | 75 | 75 | 75 | 71 | 75 | 75 | 71 | p=lambda g:[[a:=g[0][:4+any(g[4])*3][i]*i+a[:10]for i in range(10)] |
| #124 | 53b68214 | 97 | 96 | 95 | 103 | 108 | 109 | 98 | 110 | 125 | 109 | p=lambda i:i[9:]and i or p(i+[(w:=i[4])*[0for k in(1,2)if[0]*k+i[0]>i[2]]+i[w-3]:][:10]) |
| #125 | 543a7ed5 | 118 | 110 | 116 | 128 | 134 | 130 | 142 | 119 | 128 | 172 | p=lambda g,i=87,q=8:g*-i or[[[i%c,q//c*c,-(q&(q:=c)%3%5)[i//42]or c for c in g]for g[::-1]in zip(*p(g,i-1))] |
| #126 | 54d82841 | 54 | 54 | 54 | 54 | 58 | 54 | 54 | 58 | 57 | 54 | p=lambda g:g[::-1]+[[4*(0<sum(c)in c)for c in zip(*g)]] |
| #127 | 54d9e175 | 57 | 63 | 63 | 59 | 65 | 72 | 79 | 79 | 75 | 79 | p=lambda g:g*-1and g+5or g and[p(g[1])]*3+g[3:4]+p(g[4:]) |
| #128 | 5521c0d9 | 57 | 59 | 59 | 59 | 61 | 61 | 59 | 59 | 61 | 64 | p=lambda g:[*zip(*map(lambda*c:c[c.count(c[-1]):]+c,*g))] |
| #129 | 5582e5ca | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 48 | 47 | p=lambda g:[max(h:=sum(g,g),key=h.count)]*3]*3 |
| #130 | 5614dbcf | 61 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | p=lambda g:g*g!=5and(g*-1*-1or[max(map(p,g[:3]))]+p(g[3:])) |
| #131 | 56dc2b01 | 126 | 117 | 125 | 160 | 127 | 128 | 147 | 132 | 131 | 128 | p=lambda g:exec("c=2;g[:]=zip(*([j for*j,in g if(c:=c-max(j)*(c>0))|max(j)]+[[8]*9]+g[:]*99)[len(g)-1::-1]);"*4↩<br>)or g |
| #132 | 56ff96f3 | 86 | 86 | 86 | 104 | 89 | 104 | 106 | 126 | 117 | 101 | p=lambda g,k=0:[g:=[[k|(k:=k^max({*h}&{*r}))for h in g]for r in zip(*g)]for _ in g][1]<br>def p(a):<br>*r,e={f*66+l:(a)for f,a in enumerate(a)for l,a in enumerate(a)if a},<br>for f in e:i={f};r=[n for n in r if n==n-{f-66,f-1}or i.update(n)]*[i]<br>for n in r:<br>for f in n:<br>for i in n:<br>o=n<br>for o in 1//len([u for u in o if e[f]==e[u]])*r:<br>for t in n-{i}:<br>for u in[u for u in o if e[f]==e[u]==e[t]]:u+=(len([u for u in o if e[f]==e[u]==e[t]])<6)%6*(t-i);a[u//66]↩<br>[u%66],=e[u]for u in o}-{e[f]}<br>return a |
| #133 | 57aa92db | 307 | 276 | 273 | 319 | 275 | 318 | 318 | 321 | 321 | 302 | |
| #134 | 5ad4f10b | 146 | 142 | 123 | 195 | 176 | 155 | 158 | 172 | 162 | 212 | p=lambda g,D=10:[[D*(D!=v>0)for v in r]for r in('0, %g, 0'%D in"%s"%g)*g]or(h:=[*filter(any,zip(*p(g,D-5)))][:↩<br>:len(h)//3] |
| #135 | 5bd6f4ac | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | p=lambda g:[r[6:]for r in g[:3]] |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #136 | 5c0a986e | 94 | 104 | 97 | 110 | 110 | 112 | 110 | 107 | 110 | 110 | import re;p=lambda g:eval([g:=re.sub('0(?=(.{34})*$)(2)','%s',x.str(g)[::-1])for x in"21**8][-1])<br>def p(g):*_,(a,A),b=sorted(zip(g,R:=range(len(g))));c=max(a);return[[C*(max(y-A,A-y,abs(x-a.index(C)))%(A-b[1])=<br>=0)for x in R]for y in R] |
| #137 | 5c2c9af4 | 139 | 137 | 141 | 146 | 157 | 142 | 146 | 143 | 146 | 146 | def p(g):*_,(a,A),b=sorted(zip(g,R:=range(len(g))));c=max(a);return[[C*(max(y-A,A-y,abs(x-a.index(C)))%(A-b[1])=<br>=0)for x in R]for y in R] |
| #138 | 5daaa586 | 103 | 103 | 104 | 119 | 120 | 124 | 119 | 116 | 115 | 123 | p=lambda g,k=35:-k*g or p([[g:=le,g][1][0]=g>e<1<99>k]for e in l[::-1]]for*l,in zip(*g[0in g[0]:])],k-1) |
| #139 | 60b61512 | 92 | 89 | 88 | 97 | 102 | 82 | 101 | 100 | 102 | 108 | p=lambda g:[[a or(v>g[0]<(w:=l.w[:4])[v>v[5:]])*7for*w,a in zip(*g,v)]for v in g] |
| #140 | 6150a2bd | 36 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | 36 | p=lambda g:[r[::-1]for r in g[::-1]] |
| #141 | 623ea044 | 92 | 92 | 93 | 106 | 100 | 98 | 97 | 100 | 106 | 111 | p=lambda g,i=1:[[*map(max,r,[a:=abs(g[:index(m:=max(g))+(i:=i-1)])]*m,[a:]+r,r]]for r in g] |
| #142 | 62c24649 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | p=lambda g:[r+r[::-1]for r in g+g[::-1]] |
| | | | | | | | | | | | | def p(g,i=1):<br>*R,=G=b'%r'%g<br>for k in b'⊠⊠⊠'%(BEH':R[k+1]=(48%G[k]or-5)%G[k+1]+5<br>return{*G}-{*R}and eval(bytes(R)or p(g,i+1) |
| #143 | 63613498 | 127 | 135 | 130 | 142 | 155 | 145 | 151 | 212 | 139 | 169 | |
| #144 | 6430c8c4 | 53 | 53 | 53 | 59 | 59 | 60 | 59 | 59 | 53 | 53 | p=lambda g,u=[]:g*0!=0and[*map(p,g,u*g[5:])]or 3>>g+u |
| #145 | 6455b5f6 | 177 | 167 | 157 | 197 | 227 | 196 | 226 | 187 | 220 | 225 | p=lambda x,k=7,v=1:-k**x or p([[a:=b&2or[b//max(f:=sum(x,[]))+(min({*f}-{2})==b)*8,b%511*4,*[a&~2|b]*5,v:=v*512][<br>k]for b in[2]+r][:0:-1]for*r,in zip(*x)],k-1) |
| #146 | 662c240a | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | p=lambda g:(m:=g[:3])*(m!=[*map(list,zip(*m))])or p(g[3:]) |
| #147 | 67385a82 | 81 | 81 | 72 | 89 | 86 | 92 | 89 | 87 | 87 | 113 | p=lambda l,*w:i*01+0and[*map(p,i,[i*2]+i,i[1:]+[i*2],*w)]or(3in w)+7&i*9 |
| #148 | 673ef223 | 118 | 130 | 124 | 151 | 200 | 159 | 154 | 153 | 142 | 173 | p=lambda g,X=0:[[s:=0,X:=-any(r)&1+X%6,[332%(2+sum(r,-s)*(s:=s+v))or(X in g)*8for v in r],g:=g+s//8*[X]][2]for r<br>in g] |
| #149 | 6773b310 | 75 | 74 | 75 | 78 | 82 | 80 | 78 | 80 | 80 | 80 | p=lambda g:g[3:]and[p(*zip(*g[i:i+3]))]for i in[0,4,8]]or sum(b"%r"%g)%5 |
| #150 | 67a3c6ac | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | p=lambda g:[r[::-1]for r in g] |
| | | | | | | | | | | | | def p(g):<br>e,k=[*-s.index(max(s))for s in(g,g[0])]<br>for t in b'-(#$)*% '.g[e+t%5][k+t%3]=4<br>return g |
| #151 | 67a423a3 | 103 | 103 | 103 | 105 | 114 | 118 | 100 | 113 | 111 | 102 | |
| #152 | 67e8384a | 40 | 39 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | p=lambda g:g*-1*-1or[*map(p,g+g[::-1])] |
| #153 | 681b3aeb | 118 | 133 | 131 | 173 | 150 | 137 | 128 | 162 | 139 | 119 | T=0,1,2;p=lambda g:max(all(sum(G:=[[g[x+i%7][y+i%8]`g[x-i%9][y-i%1]for y in T]for x in T],G))*G for i in range(<br>5544)) |
| #154 | 6855a6e4 | 98 | 98 | 100 | 119 | 112 | 122 | 103 | 107 | 109 | 116 | p=lambda g,n=4:n and p([[r,r[8-r[3]::-1]+r[9-r[3]:]][5 in r[:3]]for r in zip(*g[::-1])],n-1)or g |
| #155 | 68b16354 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | 18 | p=lambda g:g[::-1] |
| #156 | 694f12f3 | 118 | 142 | 128 | 161 | 177 | 160 | 159 | 181 | 172 | 180 | import re;p=lambda g:eval(g:=re.sub("(?<=4.{34})4(?=.{34}4(.*0.{31}(4))?)",r"*(X:=g.count('X\2X')//X('+')+1",str(<br>r(g)))<br>def p(m):i,t,*r=15,sum(m,e:=[]),e;[e==(e:=e+[n for n in range(150)[f::i]if(i>f)&t[n]]>[]>[r:=[x+[(e,n)]for n in<br>range(15*3)for x in r*0**t[n]],e:=[]]for f in range(150)];return max([*zip(*[(any(min(a)+n-s in a*(5>%i-n%i)for<br>or a,s in x)+t[n]%5)%3for n in range(150)])*i]for x in r) |
| #157 | 6a1e5592 | 240 | 238 | 242 | 247 | 239 | 288 | 249 | 247 | 247 | 281 | |
| | | | | | | | | | | | | def p(r):<br>f,l=max((len({*str(e:=[f[o:o+3]for f in r[f:f+3]])},e)for f in range(len(r))for o in range(len(r[-1])))<br>for e in range(len(r[-1])):<br>for f in range(len(r)-e*3):<br>for o in range(len(r[-1])-e*3):<br>for a in range(len(r[-1])):<br>for a in range(e*3*all(r[f+a][o+n]==l[a//e][n//e]or r[f+a][o+n]==r[-1][-1]!=l[a//e][n//e]==max({*l[1]}-{r[-<br>1][-1]})for a in range(e*3)for n in range(e*3))):<br>for n in range(e*3):r[f+a][o+n]=l[a//e][n//e]<br>l=*zip(*l[::-1]),<br>return r |
| #158 | 6aa204c0 | 261 | 264 | 260 | 298 | 284 | 321 | 285 | 310 | 308 | 294 | |
| #159 | 6b9890af | 103 | 107 | 105 | 115 | 107 | 114 | 141 | 141 | 112 | 187 | p=lambda g,*G:[*zip(a:=[2]*99,*[r for*r,in G or p(g,*g)for c in g[::3]if c.count(2)==2if*r}-{0,2}],a)] |
| #160 | 6c434453 | 104 | 105 | 100 | 118 | 106 | 138 | 116 | 119 | 109 | 136 | import re;p=lambda i,*m:eval(re.sub("1.{5}1(.{25})?"*3,r"0,2,0\1 2,2,2\2 0,2,0",str(n or p(i,*m)))) |
| #161 | 6cdd2623 | 82 | 82 | 79 | 83 | 89 | 82 | 83 | 87 | 81 | 80 | p=lambda m:[[4//(C:=sum(m,m).count)(x)*x|4//C(i)*i for i in m[0]]for x,* in m] |
| #162 | 6cf79266 | 96 | 97 | 97 | 122 | 104 | 124 | 113 | 109 | 110 | 110 | import re;p=lambda g,k=0:eval('1,1,1'.join(re.split("(.{55})0, 0, 0"*3)[7:],str(k or p(g,g)))) |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #163 | 6d0160f0 | 130 | 131 | 130 | 139 | 137 | 133 | 139 | 147 | 146 | 153 | R=range(11);p=lambda g:[max(5*(g[y][x]==5)or(g[p%3*4+y//4][p%-4|x//4]==4)*g[p%3*4+y%4][p%-4|x%4]for p in R]for y in R] |
| #164 | 6d0aefbc | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | p=lambda g:[r+r[::-1]for r in g] |
| #165 | 6d58a25d | 112 | 130 | 130 | 122 | 127 | 123 | 126 | 127 | 142 | 149 | p=lambda g:[*zip(*map(lambda*r:r[::-(k:=r[::-1].index(max(r,key=sum(g[::-1],g).index)))]+k*(max(r[-k:]),)+r,*g))] |
| #166 | 6d75e8bb | 61 | 63 | 63 | 64 | 65 | 65 | 65 | 61 | 65 | 63 | p=lambda g:[l:=len(c:=99<l)*2for c,e in zip(*g,l)]for l in g] |
| #167 | 6e02fde3 | 72 | 71 | 70 | 76 | 83 | 82 | 77 | 72 | 75 | 76 | p=lambda i:[[5*(y==x%len(4*str(i))%3)for x in b'⊠⊠⊠']for y in(0,1,2)] |
| #168 | 6e19193c | 110 | 110 | 112 | 127 | 116 | 135 | 116 | 116 | 119 | 111 | import re;p=lambda g:eval(re.sub(r'0(?=(.{35})+.(.[^0]).{27}\2,\2)',r'\2',f"f*zip(*g[70:]or p(g*2)),}"))[::-1] |
| #169 | 6e82a1ae | 108 | 110 | 111 | 130 | 127 | 137 | 126 | 133 | 132 | 158 | p=lambda g,k=11,l=6:-k*g or p([a:=l)*[a:=[c%7,a|c|(l:=1*8)][k>0<c]for c in r][::-1]for*r,in zip(*g)],k-1,0) |
| #170 | 6ecd11f4 | 166 | 163 | 193 | 170 | 215 | 174 | 191 | 180 | 200 | 193 | f=lambda t:t[exec("t[:]=zip(*t[sum(t[0])in t[0][:][::-1]),'*82)][:]or p=lambda g:eval(f"[r&n%`7-1\nfor r,n in zip(r,n[<br>::{-`len(n:=f(f(g)[4:]))//len(r:=f(g[:4]))]]#"*2) |
| #171 | 6f8cd79b | 51 | 51 | 51 | 51 | 53 | 56 | 61 | 51 | 53 | 51 | p=lambda g:g*all(g[0])or p(*zip(*g[::0:-1],[8]*9)]) |
| #172 | 6fa7a44f | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | p=lambda g:g+g[::-1] |
| #173 | 7232fef7 | 207 | 201 | 204 | 220 | 208 | 219 | 222 | 222 | 222 | 215 | def p(i):r=[[i[n+f//3][u+f%3]for f in range(9)]for n in range(len(i)-2)for u in range(len(i[n])-2)if{1}i[n+f//3][u+f%3]<br>u+f%3]for f in range(9)]=[i[n+f//3][u+f%3]for f in range(9)][::-1)*len((i[n+f//3][u+f%3]for f in range(9))>2]+<br>;[[i[n+f//3][u+f%3]for f in range(9)for i[n+f//3][u+f%3]in{e[f]]for e in r for n in range(len(i)-2)for u in ran+<br>ge(len(i[n])-2)if{i[n+f//3][u+f%3]for f in range(9)}=={(i[n+f//3][u+f%3]>0)*e[f]for f in range(9)][::-1]*len({+<br>i[n+f//3][u+f%3]for f in range(9))>1];return i |
| #174 | 72ca375d | 91 | 89 | 97 | 98 | 102 | 110 | 96 | 105 | 99 | 105 | p=lambda g,i=1:eval(f'(g:=g[::-1])*(g:=[x for x in zip(*g)if{i}in x]),'*4)[3]or p(g,i+1) |
| #175 | 7325la56 | 74 | 75 | 75 | 75 | 87 | 75 | 75 | 75 | 75 | 75 | p=lambda g:[g:=[y!=z or x for x,y,z in zip([0]+g,r,c)]for*c,r in zip(*g,g)] |
| #176 | 7447852a | 64 | 64 | 61 | 72 | 67 | 70 | 76 | 68 | 75 | 78 | p=lambda g,r=5:[[x!=r*(r:=r-x)%6for x in s]*(r:=1)for s in g] |
| #177 | 7468f01a | 51 | 51 | 51 | 54 | 54 | 55 | 54 | 54 | 54 | 54 | p=lambda g,*G:[*filter(any,zip(*G or p(*g)[::-1]))] |
| #178 | 746b3537 | 47 | 47 | 47 | 50 | 47 | 61 | 47 | 52 | 50 | 50 | p=lambda g:g*-1*-1or[g:=r)for r in g if g!=r] |
| #179 | 74dd1130 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | p=lambda g:[*zip(*g)] |
| #180 | 75b810ce | 79 | 79 | 74 | 82 | 83 | 84 | 82 | 84 | 83 | 82 | p=lambda a:[p(b)for*b,in map(zip,a,a[4:])]or max(sum(a+a,())[1:],key=bool) |
| #181 | 760b3cac | 67 | 67 | 67 | 67 | 69 | 74 | 67 | 67 | 67 | 67 | def p(g):<br>for l in g[:3]:x=6>>g[3][3];l[x:x+3]=l[5:2:-1]<br>return g |
| #182 | 776ffc46 | 160 | 167 | 186 | 182 | 195 | 191 | 199 | 203 | 218 | 224 | from re import*;p=lambda g,h=0:eval(sub(sub("\2|3","1","(.{47})",join(a:=findall(".*5, "*"5.{46}(.{15})"*5,s:=str<br>(h or p(g,g)))[0]),"\%d "*(1,2,3,4),s)) |
| #183 | 77fdfe62 | 79 | 93 | 94 | 93 | 94 | 94 | 93 | 98 | 93 | 93 | p=lambda g,h=0:g*0!=0and[*map(p,len(g)//2*g[:1]+g[-1:]*9,h or g)][2:-2]or h%7*g |
| #184 | 780d0b14 | 91 | 91 | 94 | 104 | 113 | 109 | 102 | 122 | 99 | 104 | p=lambda g,*h,q=[]:[q+0*(q:=r)for*r,in zip(*h or p(*g))if[0,*r,q:=[*map(max,q+r,r)]]>r]+[q] |
| #185 | 7837ac64 | 107 | 110 | 121 | 143 | 126 | 112 | 149 | 120 | 128 | 158 | p=lambda g,*P:[g:=[[x*(P==(P:=x))!=max(Z))for c,x in zip(g,r)if{*c}-{*Z}][:]for r in zip(*g)]for Z in g][1] |
| #186 | 794b24be | 60 | 60 | 61 | 61 | 61 | 63 | 61 | 60 | 61 | 60 | p=lambda g:[[2,1%(n:=sum(sum(g,[]))*2,2%n],[0,6%n,0],3*[0]] |
| #187 | 7b6016b9 | 91 | 83 | 88 | 90 | 106 | 98 | 88 | 96 | 97 | 91 | p=lambda g,i=7:g*-1 or[[i:=c|2>>i*c%7for c in[3]*r][:0:-1]for*r,in zip(*p(g,i-1))] |
| #188 | 7b7f7511 | 53 | 59 | 63 | 57 | 53 | 72 | 55 | 73 | 71 | 68 | p=lambda g:(X:=g[:53%~-len(g)])*(g==X+X)or[*map(p,g)] |
| #189 | 7c008303 | 95 | 111 | 111 | 111 | 116 | 113 | 111 | 113 | 116 | 112 | p=lambda g,h=[]:g*0!=0and[*map(p,3*[g[i:=('0'in'%r'%g[2])*7]]+3*[g[i+1]],(h+g)[3>>i:]]or h%2+g |
| #190 | 7ddcd7ec | 105 | 104 | 103 | 111 | 112 | 115 | 108 | 109 | 110 | 106 | import re;p=lambda i:exec(r"i[::-1]=zip(*eval(re.sub('.{31}0, ([^0])'*2,r'\1\g<0>',str(i))));"*20)or i |
| #191 | 7df24a62 | 221 | 228 | 227 | 240 | 243 | 264 | 275 | 237 | 248 | 287 | def p(l):<br>X=[0]*U;r=*l,=X,*zip(X,*l,X),X<br>for e in[1,1,-1]*8:l[::e]=zip(*eval('1'.join(split(sub('1',')0(',sub('[^1]+'*8,lambda l:' '*len(l[0]),str(r))).<br>strip('(,)'),',%s'%l)))[1:e]<br>return[*zip(*l[1:e])][1:e] |
| #192 | 7e0986d6 | 109 | 99 | 114 | 122 | 122 | 129 | 111 | 115 | 135 | 136 | p=lambda g,k=3:g*-k or[[k:=[c:=r.pop(),k][[0,*r][-1]in[k,0]*c]for _ in r*1]for*r,in zip(*p(g,k-1))] |
| #193 | 7f4411dc | 77 | 81 | 71 | 86 | 94 | 88 | 84 | 91 | 87 | 93 | p=lambda i,*w:i*0!=0and[*map(p,i,[1]+i,i[1:]+[1],*w)]or(w.count(1)>1)*i |
| #194 | 7fe24cda | 67 | 59 | 55 | 66 | 69 | 74 | 66 | 67 | 66 | 66 | p=lambda i,s=[],k=3:-k*i or p([*zip(*i,*s)],i[::-1],k-1) |
| #195 | 80af3007 | 103 | 106 | 94 | 105 | 105 | 103 | 108 | 106 | 110 | 102 | p=lambda a,n=1:[[y&sum(a*3,())]n:=n+3]for y in x]for x in-n*a]or p([*filter(any,zip(*a))],n-1) |
| #196 | 810b9b61 | 110 | 105 | 106 | 121 | 129 | 107 | 123 | 123 | 123 | 112 | p=lambda g,i=23:g*-i or[[q:=[c|8-c&q,c|-c&~q%3,c%4][i//8]for c in g]for g[::-1]in zip(*p(g,i-1))if{q:=8}] |
| #197 | 82819916 | 54 | 54 | 54 | 54 | 54 | 54 | 51 | 54 | 54 | 54 | p=lambda g:[[*map({}.setdefault,g[i],r)]for r in g] |
| #198 | 83302e8f | 107 | 110 | 118 | 142 | 119 | 120 | 110 | 120 | 118 | 141 | import re;p=lambda g:exec("g[::-1]=zip(*eval(re.sub('(?=0|3, 4|3[^)]*[^)34]{6})','\6-5-',str(g)));"*24)or g |
| #199 | 834ec97d | 79 | 84 | 84 | 77 | 81 | 79 | 77 | 89 | 84 | 77 | p=lambda g:~(x:=g.index(k:=max(g)))*((4,0)*8)[k[1::2]>k:][:len(k)]+g[x::-1] |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #200 | 8403a5d5 | 84 | 84 | 84 | 88 | 89 | 91 | 111 | 91 | 93 | 115 | p=lambda g,c=5:[[g[9].index(m:=max(g[9]))*[0]+[m,c,m,c:=any(r)*5]*9)[:10]for r in g] |
| #201 | 846bdb03 | 187 | 198 | 188 | 230 | 215 | 248 | 246 | 229 | 231 | 224 | def p(g):f=1;m=[R for r in zip(*g)if any(R:=[x*(f*(f:=f^(x==4)))or(g:=g+[x])>g)for x in r];return[z:=[4,*[0]*le<br>n(m),4],*[a:=g[15],*r[::a in m[0]or-1],g[-2]]for*r,in zip(*m)if any(r)],z] |
| #202 | 855e0971 | 103 | 98 | 104 | 106 | 106 | 107 | 104 | 105 | 107 | 107 | p=lambda g:exec("v=p,g[:]=zip(*[map(min,[i,v][len(*v),*i,0])<3],v:=i)for*i,in g[::-1]];"*36)or g |
| #203 | 85c4e7cd | 64 | 64 | 67 | 64 | 64 | 64 | 64 | 67 | 67 | 64 | p=lambda g:[[g[n:=len(g)//2]]r.index(c)+n]for c in r]for r in g] |
| #204 | 869dc0fa | 93 | 93 | 93 | 99 | 106 | 119 | 113 | 128 | 102 | 105 | import re;p=lambda g:eval(re.sub('([^1]..)([^1]+)',r'\1*(k:=len(\1))*[2+k%2*5],',str(g))) |
| #205 | 8731374e | 144 | 134 | 136 | 180 | 153 | 190 | 218 | 179 | 181 | 208 | p=lambda g,k=87:-k*[min(e+u,key=e.count)for*u,in zip(*g)]for*e,in g]or p(["%zip(*g)]sum(u[:6],.count(u[0])>4for <br>u in g)<6][::-1],k-1) |
| #206 | 88a10436 | 140 | 140 | 140 | 139 | 161 | 148 | 151 | 149 | 149 | 151 | def p(a):<br>  i,j=divmod(sum(a,[]).index(5),len(a[0]));a[i][j]=0<br>  for s in eval('filter(any,zip(*"*2+"a"))')):a[i-1][j-1:j+2]=s;i+=1<br>  return a |
| #207 | 88a62173 | 80 | 74 | 74 | 78 | 86 | 81 | 78 | 86 | 81 | 86 | p=lambda g:[p(i)for*i,in map(zip,g,g[3:])]or min(q:=g[0]+g[1],key=q.count) |
| #208 | 89003de9 | 163 | 189 | 211 | 227 | 222 | 229 | 238 | 219 | 224 | 223 | from re import*;p=lambda g:eval((i:=min(k:=str(g)+'#[]'*X,key=k.count)).join(split(sub(i,f')[-[i]](',sub("[^%s]+<br>"%i*18,lambda x:'.'*len(x[0]),k)).strip(".()"),k)) |
| #209 | 8a004b2b | 278 | 268 | 247 | 250 | 273 | 287 | 314 | 304 | 296 | 277 | def p(i):<br>  e=i<br>  for o in e*4:e=[[*o]for o in zip(*e[~4:])if{*o}-{0,4}];i=[[*o]for o in zip(*i[(4in i[-1])-2::-1]]<br>  for o,f in enumerate(i):<br>  if[for t,f in enumerate(o*f)for r,f in enumerate(o*all(i in[0,(e+20*[[]])[(z-t)//o]+2<br>0*[4])[(n-a)//o]]for r,i in enumerate(i)for n,i in enumerate(i))*e]for n,f in enumerate(o*f)for i[r+r][n+a]in[e[<br>r//o][n/o%1]]:return i |
| #210 | 8be77c9e | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | p=lambda g:g+g[::-1] |
| #211 | 8d5021e8 | 48 | 48 | 48 | 48 | 49 | 48 | 48 | 48 | 48 | 48 | p=lambda g:[r[::-1]+r for r in[g[::-1]+g)*2][::9] |
| #212 | 8d510a79 | 99 | 101 | 93 | 115 | 113 | 99 | 120 | 117 | 121 | 121 | p=lambda i,k=18:"k*i or[[x.pop()or[0,*x][k%-2]%5&6-(5in x)for _ in i]for*x,in zip(*p(i,k-1))] |
| #213 | 8e1813be | 86 | 87 | 90 | 88 | 89 | 89 | 94 | 89 | 87 | 106 | p=lambda g:[l:=[lo:]for r in g if(l:=[e for e in r if e%5)]["(o:=6-len(t*"%s"%g))::][o:] |
| #214 | 8e6a5113 | 62 | 62 | 62 | 63 | 66 | 65 | 62 | 61 | 65 | 63 | p=lambda g:[r+g.pop()[3::-1]for r,*r[4:]in zip(g*1,*g[::-1])] |
| #215 | 8eb1be9a | 42 | 42 | 45 | 42 | 46 | 42 | 45 | 48 | 48 | 44 | p=lambda g:[*map(max,g,g[3:6]*9,g[6:9]*9)] |
| #216 | 8efcae92 | 113 | 111 | 113 | 131 | 121 | 130 | 137 | 133 | 124 | 148 | p=lambda g:max([sorted(sum(v:=[k[i%19:i%23]for k in g[i%21:i%22]],[]),key=1 .__eq__),v]for i in range(4**9)][1] |
| #217 | 8f2ea7aa | 95 | 95 | 95 | 101 | 99 | 98 | 97 | 99 | 104 | 97 | p=lambda g:[*filter(any,zip(*g[81:]or p([c&d for c in a for d in b]for a in g*2for b in g]))] |
| #218 | 90c28cc7 | 56 | 56 | 56 | 63 | 67 | 69 | 63 | 64 | 63 | 63 | p=lambda g,*h:[*{r:0for r in zip(*h or p(*g))if any(r)}] |
| #219 | 90f3ed37 | 242 | 241 | 242 | 248 | 266 | 303 | 302 | 241 | 280 | 320 | def p(r):<br>  a=r[0],;w=3-any(r[2])>>any(r[1])<br>  while any(r[w]):a+=r[w],;w+=1<br>  for l in 0,1,2,-1:<br>  for w in range(16-len(a)):<br>  for y in range(2+(l<0),10):<br>  for f in range(len(a)):<br>  for g in range(10):<br>  if l+g in range(10)!=r[w+f][l+g]!=a[f][g]:y*=g<y<br>  for f in range(len(a)):<br>  for g in range(10):<br>  if l+g in range(10)!=r[w+f][l+g]!=0:y*=g<y<br>  return r |
| #220 | 913f03ed | 89 | 82 | 88 | 109 | 97 | 93 | 110 | 93 | 94 | 94 | p=lambda g,i=3:g*-i or p([[r.pop()or[0,*r][-1]**4%8%15for r in g]for r in g],i-1) |
| #221 | 91413438 | 86 | 87 | 86 | 89 | 98 | 98 | 87 | 100 | 90 | 89 | def p(g):w=sum(g,g).count(i:=0);return[[r*(9+(i:=i-1)//3*w)+[0]*99)[:3*w]for r in g*w] |
| #222 | 91714a58 | 102 | 107 | 93 | 117 | 109 | 110 | 137 | 119 | 108 | 141 | p=lambda g:[g:=[[c*(str(r*7+g).count(2*e"f<6, ")>9)for c in r]for*r,in zip(*g)]for _ in g][5] |
| #223 | 9172f3a0 | 46 | 51 | 46 | 46 | 53 | 54 | 50 | 52 | 50 | 50 | p=lambda g:g*-1*-1or g and[p(g[0])*3+p(g[1:])] |
| #224 | 928ad970 | 145 | 114 | 131 | 185 | 145 | 149 | 181 | 138 | 171 | 187 | p=lambda g,i=21:-1*g or[g[:(b:=any(g[0])*i<8)],[max(max(g))]*99][i<4]+[*zip(*p([*zip(*g[b:][::-1])],i-1))][::-1] |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #225 | 93b581b8 | 125 | 127 | 126 | 123 | 134 | 122 | 123 | 129 | 134 | 123 | V=1,0,1,5;p=eval('lambda g:[[g[y][x]|g[V-V[y-Y>>1]][X-V[x-X>>1]]'+'for %s in range(6)%s'*4%(*'xjyY X','if g[V]<↵[X][0],'))<br>p=lambda g,k=7,r=range(10):g*-k or p([(q:=0)or[q:=g[i][j]or~j-i<2&~j,q%5][k<7]for i in r]for j in r]<↵,k-1)<br>p=lambda g,u=[]:g*0!=0and[*map(p.g,u+g[4:])]or~g+u&2<br>import re;p=lambda g:exec(r'g[:-1]=zip(*eval(re.sub(r"0(.{34}([^0]).*)(.)(.{33}\2)",r"\3\1 0\4",str(g))));'*4)<↵<br>or g<br>p=lambda g:[[(5,v)[v==max(1:=sum(g,g),key=1.count)]for v in r]for r in g]<br>p=lambda g,i=6:g*(i<3)or p([[(d:=r.pop()or[0,*r][-1]*i)>>5*(d>6>4>i)]for r in g]for _ in g],i-1)<br>p=lambda i,e=0:i*0!=0and[p(y)or[e:=y-e,5][e<0]for y in i]<br>def p(s):<br>for e,n in sorted([[-i.count(2),i]for l in range(len(s)-2)for r in range(len(s[0])-2)if all(i:=[s[l+e][r+f]for<br>e in range(3)for f in range(3)]and{*i}!={2}:l+=[2]for e in range(3)for f in range(3)for s[l+e][r+f]in[n[e*3+f]]]for l in range(len(s)-2)for<↵<br>r in range(len(s[0])-2)if all(s[l+e][r+f]==2!=n[e*3+f]-s[l+e][r+f]==2in s[l+e]for e in range(3)for f<↵<br>in range(3))];s=[s[:-1]for*s,in zip(*s)if 2in s]<br>return s |
| #226 | 941d9a10 | 120 | 117 | 119 | 137 | 139 | 120 | 134 | 143 | 149 | 132 | |
| #227 | 94f9d214 | 52 | 52 | 52 | 60 | 55 | 58 | 55 | 58 | 52 | 52 | |
| #228 | 952a094c | 116 | 116 | 116 | 136 | 130 | 168 | 121 | 119 | 122 | 153 | |
| #229 | 9565186b | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 74 | 73 | |
| #230 | 9590924e | 102 | 95 | 110 | 125 | 113 | 115 | 120 | 111 | 117 | 113 | |
| #231 | 963e52fc | 43 | 43 | 43 | 43 | 43 | 43 | 42 | 43 | 43 | 43 | |
| #232 | 97999447 | 59 | 58 | 57 | 58 | 58 | 64 | 68 | 69 | 66 | 58 | |
| #233 | 97a05b5b | 278 | 261 | 271 | 302 | 273 | 324 | 311 | 291 | 293 | 306 | |
| #234 | 98cf29f8 | 107 | 106 | 107 | 107 | 113 | 109 | 108 | 109 | 112 | 121 | |
| #235 | 995c5fa3 | 62 | 62 | 61 | 65 | 63 | 68 | 65 | 65 | 65 | 66 | |
| #236 | 995b1bc43 | 54 | 54 | 54 | 60 | 61 | 64 | 60 | 60 | 54 | 54 | |
| #237 | 99fa7670 | 66 | 68 | 67 | 66 | 68 | 68 | 66 | 79 | 67 | 71 | |
| #238 | 9aec4887 | 209 | 195 | 201 | 232 | 228 | 227 | 237 | 223 | 237 | 241 | |
| #239 | 9af7a82c | 99 | 104 | 99 | 107 | 103 | 110 | 111 | 104 | 111 | 111 | |
| #240 | 9d9215db | 99 | 102 | 104 | 107 | 113 | 94 | 89 | 135 | 102 | 103 | |
| #241 | 9dfd6313 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | |
| #242 | 9ecd008a | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | |
| #243 | 9edfc990 | 75 | 75 | 75 | 80 | 79 | 75 | 83 | 80 | 80 | 75 | |
| #244 | 9f236235 | 61 | 63 | 63 | 65 | 63 | 64 | 63 | 63 | 65 | 63 | |
| #245 | a1570a43 | 99 | 98 | 98 | 122 | 117 | 94 | 124 | 116 | 104 | 159 | |
| #246 | a2fd1cf0 | 104 | 105 | 105 | 120 | 108 | 123 | 120 | 114 | 122 | 120 | |
| #247 | a3325580 | 95 | 95 | 92 | 96 | 108 | 104 | 95 | 103 | 98 | 98 | |
| #248 | a3df8b1e | 72 | 72 | 74 | 67 | 78 | 70 | 68 | 77 | 73 | 78 | |
| #249 | a416b8f3 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | |
| #250 | a48eeaf7 | 101 | 97 | 95 | 104 | 111 | 101 | 130 | 116 | 108 | 144 | |
| #251 | a513dff | 88 | 84 | 89 | 97 | 107 | 95 | 91 | 95 | 92 | 91 | |
| #252 | a5f85a15 | 55 | 57 | 53 | 60 | 64 | 60 | 60 | 59 | 62 | 59 | |
| #253 | a61ba2ce | 120 | 121 | 114 | 132 | 162 | 142 | 158 | 140 | 158 | 141 | |
| #254 | a61f2674 | 81 | 84 | 91 | 93 | 99 | 84 | 94 | 87 | 98 | 97 | |
| #255 | a64e4611 | 222 | 233 | 212 | 246 | 278 | 308 | 267 | 281 | 288 | 257 | |
| #256 | a65b410d | 97 | 95 | 96 | 98 | 107 | 109 | 93 | 105 | 108 | 97 | |
| #257 | a68b268e | 71 | 71 | 68 | 82 | 81 | 80 | 81 | 82 | 81 | 80 | |

```
p=lambda g:[r*2for r in g]
p=lambda g:[r[::-1]:r[j]=1;b^=r[b];j-=b|1
return g
p=lambda a:[a:=sorted(b[:(i:=str(a).find('2'))>>5])+b[i:]for*b,in zip(*a)][::-0][0]for _ in a][3]
p=lambda a,i=31:g*-i or p([[r.pop()&-4**0,*r][-1]or i>30for r in g]for _ in g],i-1)
p=lambda g,v=0:g*0!=0and[*map(p.g,[-1,4]*9)]or~g//v&v
p=lambda g:[[(i:=a&b)*0+max(v*(sum(g,g)[i-1:(i:=i+1)]==[v,v])for v in sum(g,[])]for a in b'⊠[Z⊠']for b in b'⊠A[⊠<↵
']]
p=lambda g:[[(sum(g[-sum(c)//5])(5)%sum(g[8])*x%3for*c,x in zip(*g,r)]for r in g]
import re;p=lambda g,k=15:-k*g or[*zip(*eval(re.sub(*r"\((%s|(0, )+3.{5}3),?"%re.search(r"([ ,03]{61,}){.*\1}3}<↵
|$",g:=str(p(g,k-1)))[1],"(?=0((,[^,]*){31}|, ){1-9})",r"(*3]*len((\1)),","1<"))[k<4::2],g))][::-1]
def p(g):
b=sum(m:=max(g))//2+g.index(m)
for r in g:r[:b]=[2+(r<m)-m[b]*b;b-=b>0
return g
p=lambda a:[p(b)for*b,in map(zip,a,a[5:])]or max(sum(a,()),key=bool)
```

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #258 | a699f800 | 61 | 61 | 61 | 61 | 61 | 63 | 61 | 61 | 61 | 61 | import re;p=lambda g:eval(re.sub("1, 0(?=, 1)","1,2","%s"%g)) |
| #259 | a740d043 | 83 | 83 | 85 | 86 | 86 | 96 | 85 | 86 | 86 | 85 | p=lambda g:exec('g[:]=zip(*eval(str(g).replace(*"10"))[any(g[-1])-2::-1]);'*24)or g |
| #260 | a78176bb | 132 | 134 | 130 | 142 | 168 | 165 | 151 | 140 | 141 | 172 | exec("p=lambda a:[max(f*max(a)-{5}*any(a[i])[j]%5or 2==sum(m-n-i+j+k%5==2<a[m][n]"+'for %s in range(10)%s'*6%(*m,n,k),)_j]i],)) |
| #261 | a7931da0 | 47 | 47 | 47 | 47 | 47 | 50 | 47 | 47 | 47 | 47 | p=lambda g:[[x%6for x in r]for r in[g.pop()]+g] |
| #262 | a85d4709 | 39 | 39 | 39 | 40 | 39 | 39 | 39 | 39 | 39 | 39 | p=lambda g:[3*[c*68b+~a]for a,b,c in g] |
| #263 | a87ff484 | 106 | 112 | 117 | 119 | 115 | 112 | 124 | 138 | 129 | 119 | p=lambda g,h=0:-(M:=bytes(map(bool,sum(g:=["zip(*h or p(g,g))],()))))).find(M[:9],9)*g[:3]or p(g[3:]+g[:3]) |
| #264 | a8c38be5 | 178 | 189 | 190 | 200 | 200 | 201 | 182 | 201 | 248 | 234 | T=3,2,1;p=lambda g:[sum([eval(f"sorted([0in(S:=sum(g,T)),*[i!=5for i in S],*g]{'for*g,in map(zip,g,g[i:],g[2:])'*2})#{g!r}")[42>>Y&7*~X-Y][~x]for Y in T],())for X in T]for x in T] |
| #265 | a847556c | 104 | 107 | 118 | 115 | 137 | 127 | 132 | 132 | 127 | 157 | import re;p=lambda g:eval(re.sub("0(?=.{949,952}(.{56})?0(?!.{37}0.{485}|), 0.{52}0, 0)","2","%s#"%g*2)) |
| #266 | a9f96cdd | 88 | 98 | 99 | 102 | 98 | 96 | 99 | 109 | 112 | 114 | p=lambda g:[9:]or[[8|9>c|a*6)%9for a,c in zip(*0)+r,r[:i:]+[0]]for*r,in zip(*g*g*2))] |
| #267 | aabf363d | 46 | 46 | 46 | 46 | 54 | 50 | 46 | 56 | 46 | 46 | p=lambda g:[[g[6]<r>[x]]for x in r]for r in g] |
| #268 | aba27056 | 215 | 190 | 222 | 222 | 219 | 222 | 216 | 236 | 269 | 217 | import re;p=lambda g,i=7:-i*g or p(eval(re.sub("0(?=%s)"%["(.%z0)*, ["0].%%z["0], 4"%(o:=len(g)*3+4)%(o-6),x"[0,\+(.)+(.).{3}\).*#.*\1, \1, [0, ]+\1"][i>3],"4",f'{*zip(*g),}#{g!r}')[::-1],i-1) |
| #269 | ac0a08a4 | 63 | 63 | 63 | 74 | 74 | 73 | 74 | 72 | 74 | 64 | p=lambda g:eval("[g\nfor g in g for _ in[*{*x"]%z']{5:]]#"%g*2) |
| #270 | ae3edfdc | 115 | 115 | 117 | 124 | 117 | 137 | 116 | 119 | 135 | 122 | import re;p=lambda g:[g:=eval(re.sub(f'{n}([^()]*)0(?=, {8%n}}',r'0\1n',f"{*zip(*g[::-1]),}")for n in(7,3,3*8)[\+-1] |
| #271 | ae4f1146 | 86 | 90 | 86 | 94 | 101 | 93 | 94 | 94 | 94 | 94 | exec(f"p=lambda g:max((str(g).count('1'),g)[{'for*g,in map(zip,g,g[1:],g[2:])'*2}][1]") |
| #272 | aedd82e4 | 88 | 87 | 71 | 100 | 99 | 99 | 100 | 102 | 98 | 100 | p=lambda i,*w:i*0!=0and[*map(p,i,[i*2]+i,i[1:]+[i*2],*w)]or~(2in w)*i%3 |
| #273 | af902bf9 | 75 | 88 | 81 | 99 | 105 | 107 | 99 | 100 | 87 | 99 | p=lambda g,*r,k=0:[c+(k:=k-c)%6and c~2for c in r]or[*map(p,g,*map(p.g,*g))] |
| #274 | b0c4d837 | 70 | 70 | 71 | 70 | 71 | 71 | 67 | 71 | 71 | 65 | p=lambda g:[[8,(s:=sum(map(max,g))*6)+4&8,-s&8],[0,0,~s&8],[0]*3] |
| #275 | b190ff6 | 135 | 134 | 130 | 136 | 150 | 143 | 142 | 146 | 146 | 126 | def p(g):R=len(g*g[0])//3;r=range(-R*R,0);return[[sum(g[u%R-t][v%R-t]//8*g[u//R+t][v//R+t]for t in(0,R))for v in\+ r]for u in r] |
| #276 | b1948b0a | 37 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | p=lambda g:g*-1and-g%6|2or[*map(p,g)] |
| #277 | b230c067 | 159 | 157 | 155 | 168 | 178 | 166 | 188 | 184 | 192 | 172 | p=lambda g:[g:=[[v:=c and[~-c+v*v%97,i+(sum(g,[]).count(c)!=2),v==1or c][i]for c in r]for r in zip(*g)if[v:=0]\+][::-1]for i in[0]*8+[1]+[2]*7][-1] |
| #278 | b27ca6d3 | 110 | 112 | 116 | 146 | 133 | 111 | 118 | 118 | 118 | 120 | p=lambda m,k=11:-k*m or p([[[v%9,v|3%~u,v<<u[k>>2]for u,v in zip([0]+r,r)]for*r,in zip(*m[::-1])],k-1) |
| #279 | b2862040 | 109 | 106 | 107 | 118 | 128 | 121 | 128 | 120 | 125 | 131 | p=lambda g,i=94:g*~i or p([[9&r.pop()%[q+9,9|3-q][i<9]or(i<0)*9for q in[0]+r[:0:-1]]for*r,in zip(*g)],i-1) |
| #280 | b527c5c6 | 145 | 163 | 168 | 176 | 175 | 185 | 170 | 199 | 193 | 196 | p=lambda g,n=11:-n*g or[[P:=[((a:=(8%~P<x)*a+x%2*8|2)>2<x)*a,-6%(P-2|1-x|x),x&3][n//4]or x for x in r]for r in zip(*g,n:=n-1))if[a:=0,P:=0][::-1] |
| #281 | b548a754 | 123 | 124 | 103 | 149 | 152 | 125 | 167 | 138 | 143 | 127 | p=lambda a,n=47,*P:-n*a or p([*zip(*[max(P*({0,8}in map(set,a)),P:=a.pop(),key=set)for _ in a*1])],n-1) |
| #282 | b60334d2 | 74 | 83 | 76 | 101 | 97 | 112 | 104 | 99 | 94 | 100 | p=lambda g:g[99:]or[eval(8*"+(x:=r.pop())."x//5|x"0")for*r,in zip(*p(g*2))] |
| #283 | b6afbbda | 82 | 79 | 72 | 96 | 107 | 92 | 104 | 105 | 103 | 104 | p=lambda i,*w:i*0!=0and[*map(p,i,[i]+i,i[1:]+[i],*w)]or-i%8*w.count(5)%5 |
| #284 | b7249182 | 205 | 195 | 204 | 224 | 218 | 230 | 232 | 218 | 222 | 235 | p=lambda g:exec("m=max(g)\nfor x in~-2%len(*m)*b'osqmnr':(1,c),*_,r=filter(min,enumerate(m,1));s=1-2+r[0]>>1;m[\+1:s]=~1%s*[c];g[g.index(m)+x%5-2]=c\ng[:]=[y[::-1]for*y,in zip(*g)];"*4)or g |
| #285 | b775ac94 | 226 | 248 | 272 | 234 | 259 | 293 | 264 | 296 | 317 | 271 | def p(h):<br>  for e in range(8):<br>  h[::-1]=zip(*h)<br>  for e in range(len(h)):<br>  for i in range(len(h)):<br>  for q,g in((f:=[(e,i)]):=h[q]+[h[q][i+i-g-1]=h[e][i-1];f+=[(p+q,g+n)for p in range(-1,2)for n in range(-\+1,2)if 0<h[e][i-1]!=(2*(2*h)[p+q][g+n]==h[e][i]>0==h[p+q][i+i-g-n-1]] |
| #286 | b782dc8a | 105 | 106 | 104 | 110 | 111 | 110 | 112 | 113 | 114 | 125 | return h<br>p=lambda i,k=39:-k*i or[[t:=y or sum((*t%8*sum(i,x)-{t,8})for y in[8]+x][:0:-1]for*x,in zip(*p(i,k-1))] |
| #287 | b8825c91 | 53 | 56 | 55 | 66 | 56 | 72 | 66 | 69 | 58 | 58 | p=lambda*g:g[g[0]==4]*-1*-1or[*map(p,g[-1][::-1],*g)] |
| #288 | b8cdaf2b | 86 | 87 | 91 | 86 | 109 | 85 | 86 | 88 | 95 | 88 | def p(g):*w,a,b=g;i=j=a..index(b[0]);exec(';i-=1;v=w[j-1];v[j]=v[~j-1];v[j]=v[~j]=b[i];'*i);return g |
| #289 | b91ae062 | 63 | 63 | 63 | 74 | 74 | 73 | 74 | 74 | 74 | 67 | p=lambda g:eval("[[g\nfor g in g for _ in[*{*x"]%z']{5:]]#"%g*2) |
| #290 | b94a9452 | 65 | 67 | 68 | 70 | 69 | 67 | 70 | 69 | 69 | 70 | p=lambda g,*u:[sum((*u),r*-1)or p(r,*sum(g,r))for r in g if r[~g] |
| #291 | b9b7f026 | 61 | 61 | 59 | 62 | 62 | 72 | 62 | 70 | 70 | 63 | p=lambda m,k=1:[*{r.count(k)for r in m},[k]][3:]or p(m,k+1) |
| #292 | ba26e723 | 52 | 52 | 51 | 56 | 56 | 56 | 56 | 56 | 56 | 56 | p=lambda g,v=0:g*0!=0and[*map(p,g,b'\n□□'*7)]or~g%v |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #293 | ba97ae07 | 60 | 59 | 60 | 60 | 60 | 66 | 60 | 69 | 60 | 60 | p=lambda g:[[(c in r)**c*r[0]or c for c in g[0]]for r in g] |
| #294 | bb43febb | 70 | 70 | 70 | 70 | 70 | 97 | 82 | 70 | 70 | 70 | import re;p=lambda g:eval(re.sub("(?<=5.{34})5(?=.{34}5)","2",str(g))) |
| #295 | bbc9ae5d | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 59 | 54 | 54 | p=lambda g:[P:=g[0]]+[P:=P[:1]+P[:-1]for x in P[2::2]] |
| #296 | bc1d5164 | 55 | 60 | 62 | 62 | 63 | 61 | 66 | 63 | 63 | 63 | p=lambda*g:[*map([*g.max,p],[2],*[r[-3:]for r in g],*g)] |
| #297 | bd447eb8 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | p=lambda g:g[:2]+[*zip(*g[:1]*len(g[0]))]*2 |
| #298 | bda2d7a6 | 55 | 55 | 55 | 55 | 55 | 55 | 54 | 58 | 58 | 55 | p=lambda g:[[g[2][-r.index(v)]2]for v in r]for r in g] |
| #299 | bdad9b1f | 54 | 54 | 54 | 54 | 54 | 57 | 54 | 54 | 54 | 54 | p=lambda g:[[92%(4-(2in r)+x)for x in g[0]]for r in g] |
| #300 | be94b721 | 87 | 87 | 87 | 94 | 97 | 91 | 94 | 91 | 92 | 93 | p=lambda g,*G:[r for*r,in zip(*G or g,*g)]if max(range(1,10),key=sum(g.g).count)in r] |
| #301 | beb8660c | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | s=sorted;p=lambda g:s(map(s,g)) |
| #302 | c0f76784 | 89 | 89 | 91 | 106 | 101 | 119 | 115 | 107 | 103 | 109 | import re;p=lambda g:eval(re.sub('([^5]..5,)([^5]+)',r'\1*(k:=len(\2))+k+5],',str(g))) |
| #303 | c1d99e64 | 62 | 63 | 64 | 64 | 64 | 65 | 64 | 65 | 65 | 63 | p=lambda g:[[[2,e][c>0]*99<1]for*c,e in zip(*g,1)]for l in g] |
| #304 | c3e719e8 | 92 | 92 | 92 | 93 | 92 | 92 | 92 | 92 | 93 | 92 | p=lambda g:[[c*(w==max(p:=sum(g,g).key=p.count))for w in q for c in r]for q in g for r in g] |
| #305 | c3f564a4 | 57 | 60 | 57 | 57 | 63 | 67 | 57 | 64 | 57 | 57 | p=lambda g:[[*{*g[0]}-{0}]*9][i:i+6]for i in range(16)] |
| #306 | c44b776 | 63 | 64 | 63 | 75 | 74 | 73 | 66 | 63 | 66 | 88 | p=lambda g:[g:=[*zip(*map(max,g,g[:10]+g))][::-1]for _ in g][7] |
| #307 | c59eb873 | 46 | 50 | 46 | 50 | 53 | 52 | 50 | 51 | 50 | 50 | p=lambda g:g*-1+-1or g and[p(g[0])*2+p(g[1:])] |
| #308 | c8cbb738 | 227 | 202 | 205 | 195 | 244 | 208 | 215 | 248 | 256 | 192 | def p(e):i,u=len(e[0]),enumerate;e=sum(e,[]);m=max(e:=[e.count(r)<9 and f-sum(e*(u==r)for e,u in u(e))//4:r for <br>f,r in u(e))//s;s:p=range(-m,m+1);return[[e.get(u*s+r,e[0])for r in p]for u in p] |
| #309 | c8f0f002 | 36 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | p=lambda g:g*-1and g&-3or[*map(p.g)] |
| #310 | c909285e | 78 | 81 | 70 | 88 | 95 | 87 | 95 | 87 | 88 | 85 | p=lambda a,*n:[b for b in zip(*n or p(a,*a))if{*b}-{*a[1]}&{*a[12]}] |
| #311 | c9e6f938 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | p=lambda g:[r+r[::-1]for r in g] |
| #312 | c9f8e694 | 44 | 44 | 44 | 45 | 45 | 44 | 44 | 44 | 44 | 44 | p=lambda g:[[x%~x&r[0]for x in r]for r in g] |
| #313 | caa06a1f | 61 | 63 | 62 | 64 | 71 | 71 | 64 | 64 | 65 | 71 | p=lambda g,u=[]:g*-1+-1or[*map(p.g[u>[]:2--len(u)//11]*10,g)] |
| #314 | cbded52d | 84 | 96 | 82 | 106 | 111 | 98 | 97 | 96 | 102 | 93 | p=lambda i,*w:i*0!=0and[*map(p,i,i[:3]+i,i[3:]+i,*w)]or max(w[0]&w[1],w[2]&w[3],i) |
| #315 | cce03e0d | 63 | 63 | 63 | 64 | 64 | 64 | 64 | 63 | 63 | 63 | p=lambda g:[[y&-x%5for x in r for y in s]for r in g for s in g] |
| #316 | cdecee7f | 71 | 71 | 71 | 71 | 71 | 71 | 71 | 71 | 71 | 71 | p=lambda g:[[S:=[*filter(int,map(max,*g))]+[0]*9][:3],S[5:2:-1],S[6:9]] |
| #317 | ce22a75a | 43 | 53 | 43 | 59 | 61 | 59 | 59 | 59 | 61 | 59 | p=lambda g:g==5or g and[p(g[1])]*3+p(g[3:]) |
| #318 | ce4f8723 | 54 | 54 | 54 | 58 | 59 | 60 | 58 | 60 | 54 | 54 | p=lambda g,u=[]:g*0!=0and[*map(p.g,u*g[5:])]or(g!=u)*3 |
| #319 | ce602527 | 203 | 194 | 235 | 273 | 257 | 268 | 282 | 274 | 279 | 211 | p=lambda g,x=0:[[[v[3],c][a==c]for c in r]for r in zip(*x or p(g,g))if(a:=(v:=sorted({*q:=sum(g,[])},key=q.cou<br>nt))[hash((*v,q&9&hn7&$h&]{6yr&&gHB&q[~&H"m"&r'[sum(q)%37:]%q,)%2]in r] |
| #320 | ce6e57f2 | 65 | 66 | 66 | 67 | 66 | 66 | 66 | 68 | 68 | 84 | p=lambda g:[*zip(*map(lambda*r:r[::-(k:=sum(r)//4)]+k*(8,)+r,*g))] |
| #321 | cf98881b | 54 | 54 | 55 | 55 | 56 | 61 | 55 | 57 | 55 | 54 | p=lambda g:[eval("r.pop(0)or r[4]|r[9],"*4)for r in g] |
| #322 | d037b0a7 | 48 | 48 | 48 | 45 | 48 | 48 | 48 | 48 | 49 | 48 | p=lambda g:g and[*p(g[:-1]),[*map(max,*g*2)]] |
| #323 | d06dbe63 | 106 | 100 | 104 | 131 | 124 | 122 | 112 | 119 | 120 | 131 | import re;p=lambda g:[p,eval]["in g](re.sub('0(?=(.{76})*(.{40}|(...){25,27})8)','5',str(g)[::-1])) |
| #324 | d07ae81c | 225 | 222 | 227 | 216 | 240 | 245 | 220 | 242 | 262 | 235 | def p(a):<br>n=sum(a,[]);c=n.count;o=n[c(n[0])<4]if s:={},{()}<br>for i,e in enumerate(e):<br>for j,t in enumerate(e):<br>if c(t)<4:f[o in e and o in n[j::len(e)]]=t;s|={i+j,(i-j,)}<br>for i,e in enumerate(e):<br>for j,t in enumerate(e):<br>if{i+j,(i-j,)}&s:e[j]=f[o in e and o in n[j::len(e)]]<br>return a |
| #325 | d0f5fe59 | 152 | 154 | 140 | 169 | 179 | 158 | 151 | 180 | 158 | 162 | p=lambda i,z=8,*w:i[22:]and[[8*(a==b)for b in w]for a in w]or p([[s:=h and(z:=z*2)|s|h for h in[0]+x]for*x,in zi<br>p(*i[::-0])],*{*sum(i,[0])}) |
| #326 | d10ecb37 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | p=lambda g:[g[0][:2],g[1][:2]] |
| #327 | d13f3404 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | exec("p=lambda g,P=[0]*3:[P:=[*map(max,r+[0]*3,[0]+P*2)]for r in g*[P]*3] |
| #328 | d2227ba0 | 163 | 160 | 158 | 166 | 170 | 160 | 166 | 169 | 171 | 167 | exec("p=lambda g:[[D:=sorted((sum(T:=[abs(x-r),abs(y-c)]),~max(T)%2*f[y])"+'for %s,f in enumerate(g)%s'%4%(*'y <br>x','if f[y]))[0][1]*(D[0]<D[1][:1]),*'c1r'}) |
| #329 | d23f8c26 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | p=lambda g:[(j:=len(g)//2)*[0]+[r[j]]+j*[0]for r in g] |
| #330 | d2abd087 | 116 | 114 | 110 | 135 | 132 | 127 | 133 | 136 | 138 | 135 | p=lambda i,k=-19,z=1:k%i or p([[e:=y and[1+y%7//6,z:=z*8,e|y[k>>4]for y in[0]+i][:0:-1]for*i,in zip(*i)],k+1) |
| #331 | d364b489 | 82 | 83 | 83 | 83 | 85 | 83 | 82 | 83 | 87 | 83 | p=lambda g:[g:=eval(f"{*zip(*g[::-1]),}".replace(1, 0','1,'+k))for k in'2786'][3] |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #332 | d406998b | 58 | 58 | 58 | 62 | 59 | 61 | 64 | 64 | 64 | 64 | p=lambda g:[l*map(min,A[::-1],(3,5)*10)[::-1]for A in g] |
| #333 | d43fd935 | 83 | 89 | 89 | 94 | 101 | 91 | 103 | 101 | 103 | 101 | p=lambda g:eval("n"+9*",(P:=r.pop()or(31n r)*P)"for*r,P in zip(*g[70:]or p(g*2))] |
| #334 | d446954b | 65 | 66 | 69 | 70 | 73 | 73 | 70 | 73 | 73 | 73 | p=lambda g:[i%7,i%6,i%11for i in b" M□·□M·MM□"[max(max(g))::3]] |
| #335 | d4a91cb9 | 106 | 105 | 110 | 120 | 110 | 123 | 120 | 115 | 124 | 120 | S=sum;p=lambda g,s=6:[[j~(9<9)(s:=s+S(i)*s(a))%3|S(i)>3<S(a)|~s%9)*4>>j*9for*a,j in zip(*g,i)]for i in g] |
| #336 | d4f3cd78 | 88 | 86 | 90 | 111 | 112 | 95 | 120 | 106 | 108 | 110 | p=lambda g:exec(f"g[:]=[{[r.pop()or(sum(r[:5])&sum(r)&5)*8for r in g],'*10};"*4)or g |
| #337 | d511f180 | 43 | 43 | 43 | 48 | 48 | 48 | 47 | 46 | 46 | 47 | p=lambda g:g*~-(and g"84%g%3*13or"[*map(p,g)] |
| #338 | d5a6de2d | 64 | 62 | 65 | 67 | 75 | 79 | 65 | 66 | 69 | 77 | p=lambda g:[(c:=-j~-c%7%3>1)*3for j in i]for i in g] |
| #339 | d631b094 | 37 | 37 | 37 | 37 | 37 | 37 | 37 | 37 | 37 | 37 | p=lambda g:[i*filter(int,sum(g,[]))] |
| #340 | d687bc17 | 113 | 111 | 112 | 138 | 114 | 117 | 138 | 132 | 126 | 149 | p=lambda g:[g:=[a]+-~(C:=a in r*e)+[c*(c in g[-1]+g[1])for c in r[C:]]for a,*r in zip(*g)][::-1]for _ in g][3] |
| #341 | d6ad076f | 108 | 114 | 115 | 128 | 129 | 129 | 138 | 121 | 123 | 135 | p=lambda g:[g:=[g[i][j]or(9>j>=2<len(t*min(g[i-1:i:3]))*8for i in R]for j in R]for R in[range(10)]*2][1] |
| #342 | d89b689b | 107 | 106 | 108 | 128 | 111 | 111 | 124 | 117 | 128 | 125 | import re;p=lambda g:eval(re.sub("([1-9])((.{32})+?[^8])1+8",r"0\2\1",f'{*zip(*g[70:]or p(g*2)),}'))[::-1] |
| #343 | d8c310e9 | 64 | 63 | 64 | 65 | 65 | 64 | 65 | 65 | 65 | 65 | p=lambda g:[r[r:8~-(r[8:12]+=r[:4]!=r[4:8])*3][::15]for r in g] |
| #344 | d9079ee8 | 77 | 77 | 74 | 79 | 78 | 77 | 77 | 79 | 78 | 77 | p=lambda i,a=0,*h:i[1:]and[[c[1][2&6%~sum(h)+max((h:=c)[1:])]for c in zip(*i)]]+p(i[a:],1) |
| #345 | d9f24cd1 | 90 | 90 | 89 | 95 | 108 | 100 | 95 | 93 | 104 | 95 | p=lambda a:[min(b:=sum(a[1:-1],a[3]),key=b.count)] |
| #346 | d9fac9be | 55 | 58 | 52 | 88 | 56 | 73 | 67 | 55 | 55 | 69 | p=lambda g:[[6~6>>x+r.pop(3)for x in r]for r in g] |
| #347 | dae9d2b5 | 50 | 50 | 50 | 54 | 53 | 54 | 50 | 50 | 50 | 50 | p=lambda g:[[6~6>>x+r.pop(~-l*map(max,g[:-2]),4].index(4))for r in g] |
| #348 | db3e9e38 | 88 | 92 | 94 | 101 | 101 | 104 | 101 | 105 | 103 | 103 | p=lambda g,G=0,*s:[s:=[r.pop()|~y%15for y in s[1:]]+r for r in(G or p(g,g))[::-1]][::-1] |
| #349 | db93a21d | 172 | 192 | 203 | 223 | 205 | 163 | 214 | 228 | 213 | 192 | p=lambda g,E=enumerate:[[max(B,*((A:=(*G,0).index(0,B)-B>>1)>=C-F>~A<D-B<A*3)*3|(C-F>A>D-B-A>~A)for(F,G)in E(g)]+for(B,H)in E(G)]for(D,B)in E(B)]for(C,B)in E(g)] |
| #350 | dbc1a6ce | 91 | 92 | 89 | 97 | 102 | 97 | 99 | 100 | 98 | 101 | p=lambda i:[*map(f:=lambda*x,s=0:[y|(x.count(1)>(s:=s+y%8)>y<1)*8for y in x],*map(f,*i))] |
| #351 | dc0a314f | 67 | 67 | 66 | 67 | 70 | 67 | 67 | 67 | 67 | 67 | p=lambda i:[r[:5]for x in[*i]if(r:=i.pop()[~[*x,3].index(3)::-1])] |
| #352 | dc1df850 | 82 | 82 | 82 | 90 | 90 | 88 | 105 | 88 | 88 | 91 | p=lambda g:[g:=[[r.pop()or[0]<r[-1:]<[3]for _ in g]for*r,in zip(*g)]for _ in g][3] |
| #353 | dc433765 | 87 | 83 | 83 | 123 | 107 | 105 | 124 | 111 | 91 | 124 | p=lambda g:exec("g[:]=zip(g.pop(~-l*map(max,g[:-2]),4].index(4)),*g[::-1]);"*4)or g |
| #354 | ddf7fa4f | 96 | 96 | 96 | 92 | 101 | 97 | 96 | 106 | 100 | 92 | p=lambda g:[(c:=0)or[c:=r[i]and max(c,*g[0][i:],key=bool)for i in range(10)]for r in g] |
| #355 | de1cd16c | 98 | 95 | 101 | 102 | 103 | 105 | 116 | 105 | 117 | 110 | p=lambda g:[sorted({i*x}|{i+a}^{b},)for j in g for*a,b in zip(*g,j)]],key=A.count)[-3]] |
| #356 | ded97339 | 92 | 105 | 100 | 108 | 105 | 105 | 108 | 110 | 112 | 108 | p=lambda g,*x,i=0:[x|max(g[i:])&max(g[:(i:=i+1)])for x in r]or[*map(p,g,*map(p,zip(*g),*g))] |
| #357 | e179c5f4 | 84 | 86 | 86 | 85 | 94 | 84 | 81 | 86 | 86 | 85 | def p(g,i=9):<br>  for r in g:w=~len(r);r[:]=[8]*~w+r[i%w-1//w%-2]=1;i-=1<br>  return g |
| #358 | e21d9049 | 90 | 91 | 91 | 110 | 116 | 101 | 110 | 111 | 111 | 106 | p=lambda g,*r:[max(r[::6-83>>len({*(r:=(0,)*35+r)})])]for _ in r]or[*map(p,g,*map(p,g,*g))] |
| #359 | e26a3af2 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | p=lambda g:[max(T:=r+c,key=T.count)for*c,in zip(*g)]for r in g] |
| #360 | e3497940 | 45 | 45 | 45 | 45 | 45 | 45 | 45 | 45 | 45 | 45 | p=lambda g:[[*map(max,r,r[4:-1])]for r in g] |
| #361 | e40b9e2f | 203 | 186 | 185 | 208 | 211 | 217 | 204 | 194 | 221 | 205 | def p(r):n=[r+r for r in r+r];return[[n[l][i]|n[r-a+i][r+a+e*l]|n[r+r+e*l][a+a+e*i]|n[r+a+e*i][a-r+l]for i in range(10)]for l in range(10)]for e in range(10)]for a in range(10)]for r in range(10)if all(all(r[a:a+e])for r in n[r:r+e])][-1] |
| #362 | e48d4e1a | 69 | 69 | 69 | 72 | 75 | 69 | 67 | 69 | 75 | 67 | p=lambda g:[r[(d:=g.count(g[0]))-9]+~d*r[:-1]for r in g*2][~d-9:-d] |
| #363 | e5062a87 | 175 | 203 | 204 | 210 | 205 | 187 | 214 | 217 | 237 | 216 | from re import*;def p(g):h=hash((*g[3],));g["h%7][3]=n%149<1;return eval(sub('10',eval('"2'.join(split(sub('2'+',')0(',sub('[^2]',',',k:=str(g)))).strip('.()'),"*3+"*(k)))))")) |
| #364 | e509e548 | 133 | 149 | 134 | 219 | 156 | 171 | 149 | 171 | 166 | 242 | p=lambda g,k=95:-k*g or p([g:=[7&88>>c%7,c|a|all([a,b*-1,k>83)<<k*3][k>0<c]for c,a,b in zip(r,[0]+r,g)]for*r,i+n zip(*g[::-1])],k-1) |
| #365 | e50d258f | 111 | 112 | 111 | 126 | 121 | 130 | 119 | 123 | 123 | 126 | p=lambda g:max((-(y:=sum(s:=[r%9:x%:13for r in g[x%8:x%:11],g).count)(0),y(2),y(1),s)for x in range(5*6)][3] |
| #366 | e6721834 | 331 | 337 | 301 | 367 | 328 | 371 | 372 | 374 | 374 | 370 | def p(e):<br>  *u,l,n,i=sorted({*sum(e,[])},key=sum(e,[]).count)<br>  *u,=e,<br>  for f in u*6:<br>  for f in u+(u:=[]):<br>  u+=[],<br>  for f in zip(*f):u+=u.pop()+[f]if{n,i}-{*f}>{n}or{*f}>{n}else[],<br>  [6for m,u in sorted((-sum(l^u for u in u),u)for u in u)for m,f in enumerate(e)for s,f in enumerate(f+<br>)for o,f in zip(e[m:]+e,u*all((r==f)==(i==f)for o,f in zip(e[m:]+e,u)for r,f in zip(o[s:s+len(f)]+e,f)))+<br>for o[s:s+len(f)]in[f]];return[f for f in zip(*[f for f in zip(*e)if i in f])if i in f] |

| TASK # | TASK ID | TEAM #1 | TEAM #2 | TEAM #3 | TEAM #4 | TEAM #5 | TEAM #6 | TEAM #7 | TEAM #8 | TEAM #9 | TEAM #10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #367 | e73095fd | 135 | 128 | 126 | 130 | 129 | 146 | 126 | 129 | 127 | 223 | import re;p=lambda g,k=-19:k*g or p(eval(re.sub(f'0(?=, 4|.{{%d}}0)',"4",str([*zip(*g)]<br>[::-1]))),k+1) |
| #368 | e76a88a6 | 135 | 136 | 129 | 140 | 149 | 140 | 142 | 172 | 146 | 152 | import re;p=lambda i:eval(re.sub("5(, 5)+",lambda m:re.findall("[^50](?:, [1-9])+",s*2)[-s[m.end()-i::32].count(<br>'5')],s:=str(i))) |
| #369 | e8593010 | 109 | 109 | 109 | 141 | 124 | 121 | 131 | 129 | 118 | 124 | p=lambda g,n=7:g*-n or[*map(lambda*r,b=5:[b:=a*(a>4)or[a+b//4-.25,min(a,b)][n>3]for a in r],*p(g,n-1)[::-1]]] |
| #370 | e8dc4411 | 159 | 164 | 227 | 247 | 249 | 267 | 208 | 266 | 262 | 208 | import re;p=lambda g:exec('s=f"{*zip(*g[::-1]),}";i=s.rfind;d=i("0")-i(j:=min(s,key=i));g[:]=eval(re.sub("(?=(.{<br>%d})+0)\d"%d,j,s,d%8*d%-(len(g)*3+5));'*4)or g |
| #371 | e9614598 | 102 | 109 | 109 | 109 | 124 | 125 | 109 | 125 | 118 | 109 | p=lambda g,u=0:eval((G:=f"{*zip(*u or p(g,g),)}")[:(w:=G.rfind("1")>>1)-3]+"3,"*3+G[w+5:]) |
| #372 | e98196ab | 47 | 47 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | p=lambda g:eval("[*map(max,g.pop(0),g[5])],"*5) |
| #373 | e9afcf9a | 38 | 38 | 38 | 38 | 39 | 38 | 38 | 39 | 38 | 38 | p=lambda g:[T:=max(zip(*g),T[::-1]] |
| #374 | ea32f347 | 103 | 106 | 107 | 148 | 106 | 133 | 124 | 116 | 122 | 111 | p=lambda g,i=51:-i*g or p(eval(x:=f"{*zip(*g),}".replace(i//5*",",5",i//5*",7*len({*x})%5"))[::-1],i-1) |
| #375 | ea786f4a | 53 | 53 | 53 | 53 | 53 | 58 | 53 | 53 | 53 | 53 | def p(g,i=0):<br>  for r in g:r[i]=r[~i]=0;i+=1<br>  return g |
| #376 | eb281b96 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | p=lambda g:2*(g*g[1:-1])+g[:1] |
| #377 | eb5a1d5d | 55 | 55 | 55 | 60 | 59 | 61 | 60 | 59 | 59 | 60 | p=lambda g,h=0:[h:=r for*r,in zip(*h or p(g,g))if r!=h] |
| #378 | ec883f72 | 140 | 131 | 138 | 154 | 159 | 179 | 155 | 145 | 145 | 142 | import re;p=lambda g:exec("x='...'*len(g)+'.0';g[::-1]=zip(*eval(re.sub(f'0(?=({x}, .)*, [^0]{x*2}, (.))',r'\\2'+b<br>,str(g)));"*4)or g |
| #379 | ecdecbb3 | 137 | 135 | 141 | 163 | 190 | 153 | 185 | 170 | 177 | 147 | p=lambda g,i=47:g*-i or[[3&q%~(c:=r.pop())%512&q*-(8in r),~c*q%2*9,c&1<0][8>>13-i//4]or c for q in[0]+r[:0:-1]]f<br>or*r,in zip(*p(g,i-1))] |
| #380 | ed36ccf7 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | p=lambda g:[*zip(*g)][::-1] |
| #381 | ef135b50 | 79 | 79 | 79 | 86 | 89 | 82 | 88 | 90 | 86 | 90 | p=lambda g:[r*(P:=r in g[::9])or[P:=r.pop(0)or any(r*P)*9for _ in g]for r in g] |
| #382 | f15e1fac | 133 | 129 | 124 | 138 | 146 | 123 | 167 | 179 | 131 | 152 | p=lambda g,d=8,s=0:d and p(*zip(*(g,[[c,*[0]*(s:=s+(c==2))],*g[0][1:]]for c,*v in g))[g[1>s]>g[1]])[::1-(d%4>0<b<br>)],d-1)or g |
| #383 | f1cefba8 | 123 | 120 | 121 | 130 | 128 | 136 | 121 | 129 | 138 | 124 | p=lambda g:[[(a:=[v,*{}.fromkeys(sum(g,r)))[any(0<i.count(a[2])<4for i in[r,c])*2+0**v]for*c,v in zip(*g,r)]for<br>r in g] |
| #384 | f25fbde4 | 62 | 61 | 62 | 67 | 69 | 65 | 63 | 68 | 66 | 65 | p=lambda g,*a:sum([any(x)*2*[x]for*x,in zip(*a or p(*g))],[]) |
| #385 | f25ffba3 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | p=lambda g:g[:4:-1]+g[5:] |
| #386 | f2829549 | 50 | 50 | 52 | 52 | 53 | 54 | 52 | 52 | 52 | 50 | p=lambda g:[eval('3>>r.pop(0)+r[3],'*3)for r in g] |
| #387 | f35d900a | 192 | 201 | 205 | 230 | 222 | 235 | 231 | 215 | 252 | 235 | p=lambda g,n=11:-n*g or[[[(i:=(x:=r.pop())+1>0)*-~i)%2*sum(r[:2-i])**4%5*5or x,x or-P**4%5*P*4,sum({*sum(g*(x>9<b<br>),[])}-{5,x%15})or x]for P in[0]+r[:0:-1]]for*r,in zip(*p(g,n-1))if[i:=0]] |
| #388 | f5b8619d | 61 | 61 | 62 | 62 | 62 | 62 | 62 | 61 | 61 | 61 | p=lambda g:[[x!8&x-any(c)for*c,x in zip(*g,r)]*2for r in g]*2 |
| #389 | f76d97a5 | 57 | 57 | 57 | 58 | 61 | 57 | 57 | 57 | 62 | 57 | p=lambda g:[[sum({*sum(g,r)}-{5,x})for x in r]for r in g] |
| #390 | f8a8fe49 | 98 | 98 | 100 | 121 | 113 | 122 | 103 | 107 | 109 | 116 | import re;p=lambda g,k=0:eval(re.sub("[^2]{9}2"*2+"?","*[\g<0>][::-1]",f"{*zip(*k or p(g,g),)}")) |
| #391 | f8b3ba0a | 63 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | p=lambda g:[*zip(sorted(range(10),key=sum(g,g).count)[8:5:-1])] |
| #392 | f8c80d96 | 135 | 149 | 152 | 152 | 158 | 154 | 174 | 156 | 160 | 156 | p=lambda g,n=23:-n*g or[eval(f"r.pop()or[5,w:=max(max(g))][fn}//4%((`0, %d, '%w*2in'{g}')-3)]*any(r[-1:]),"*10)f<br>or*r,in zip(*p(g,n-1))] |
| #393 | f8ff0b80 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | 63 | p=lambda g:[*zip(sorted(range(10),key=sum(g,g).count)[8:5:-1])] |
| #394 | f9012d9b | 91 | 85 | 85 | 90 | 94 | 96 | 113 | 95 | 92 | 85 | p=lambda g:[(i+i[1:])[~(8|18i%len(g)-i.index(0))%6:]for i in g if(x:=i.count(0))] |
| #395 | faffa47 | 53 | 53 | 53 | 58 | 56 | 58 | 56 | 59 | 53 | 53 | p=lambda g,u=[]:g*0!=0and[*map(p,g,u*[3:])]or g+u&2 |
| #396 | fcb5c309 | 177 | 167 | 139 | 201 | 171 | 202 | 203 | 218 | 205 | 194 | p=lambda m,n=266,f=0:[[sum((*e*sum(m,[-f]))for e in s]for r in m if(f:=((X:=n>5)*a:=(r*3)[x:=n%32])in(s:=r[x:+b<br>x+X],[f*X))*a)]or p(m,n-1) |
| #397 | fcc82909 | 117 | 110 | 113 | 126 | 145 | 130 | 124 | 132 | 135 | 124 | p=lambda g,i=81:exec("i-=1;c=1//9-1\nfor _ in*all(b:=%s+%s)*b:%s=3,3\n"%(('g[c:=c+1][%9:i%9+2]',)*3)*i)or g |
| #398 | feca6190 | 78 | 77 | 73 | 73 | 81 | 78 | 72 | 79 | 81 | 72 | def p(g):g,*g,r=*g.count(0)%6*5*[0];return[r:=[1:]+[c]for c in g+r[5:]] |
| #399 | ff28f65a | 63 | 62 | 63 | 64 | 66 | 66 | 64 | 64 | 64 | 64 | p=lambda g:[*zip(*[iter(sum(sum(g,[]))%7*[1,0]+[0]*9)]*3)][:3] |
| #400 | ff605c23 | 67 | 67 | 66 | 67 | 70 | 67 | 67 | 67 | 67 | 67 | p=lambda g:[h[:5]for r in[*g]if(h:=g.pop()[~r:1].index(1)::-0)] |