Linear Algebra 2023 HW1
Cycle Detection

TA : b08201054 鄭承櫸、b08901172 莊鳴鐸
2023.09.22

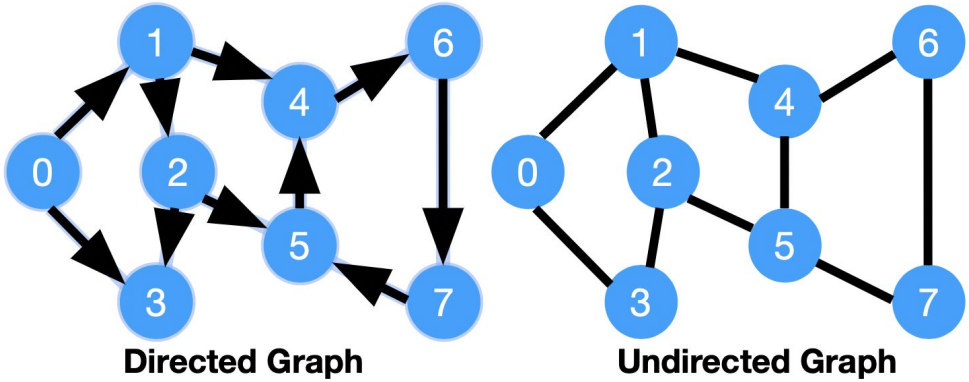
Outline

1. Task Introduction
    a. Graph
    b. Cycle
    c. Cycle detection application
2. Promblem1
3. Promblem2
4. Python Tips
5. Submission

Task Introduction

Graph

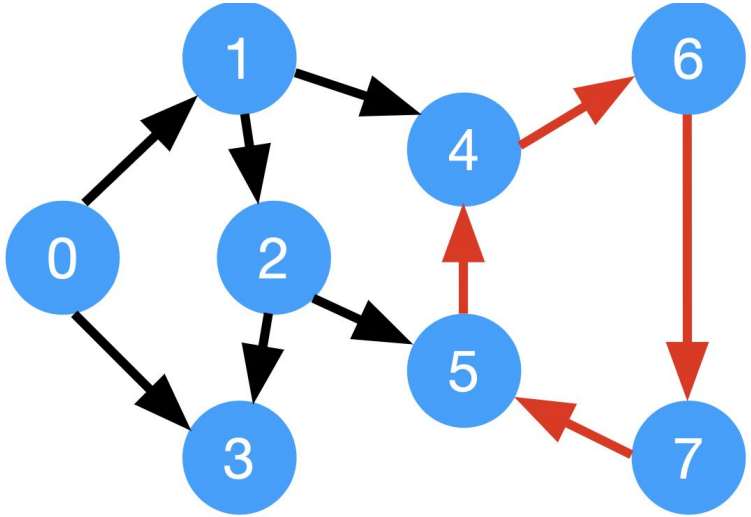
- A graph contains some nodes and some edges.
  - Often represented as G = (V, E) with V nodes and E edges.
- The edges can be directed or undirected.
  - (a, b) is an edge from node a to node b in a directed graph.
- This task is given the **directed graph**, we need to find out whether **there is a cycle** in the graph.



Cycle

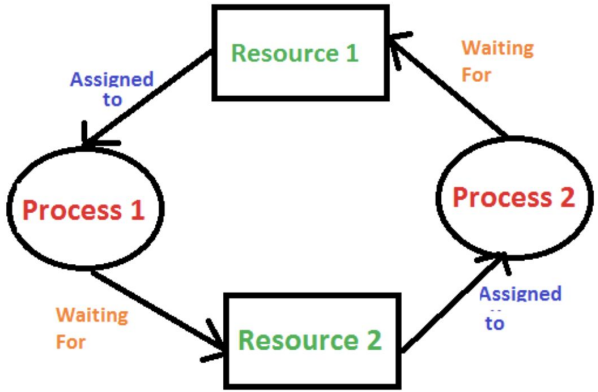
- A (simple) cycle is a path <v0, v1, ..., vk> in which v0 = vk.
  - <4, 6, 7, 5, 4> is a cycle in this graph.

0 → 1
0 → 3
1 → 2
1 → 4
2 → 3
2 → 5
4 → 6
5 → 4
6 → 7
7 → 5



Application of Cycle Detection

- Detect whether there is a **deadlock**.
  - **Deadlock** is a situation where two or more processes (running program) are unable to proceed because each is **waiting for the other to release a resource** they need in order to continue.
  - In a deadlock, the processes are stuck and **the system cannot make any progress**.



https://www.geeksforgeeks.org/deadlock-detection-recovery/

Problem 1

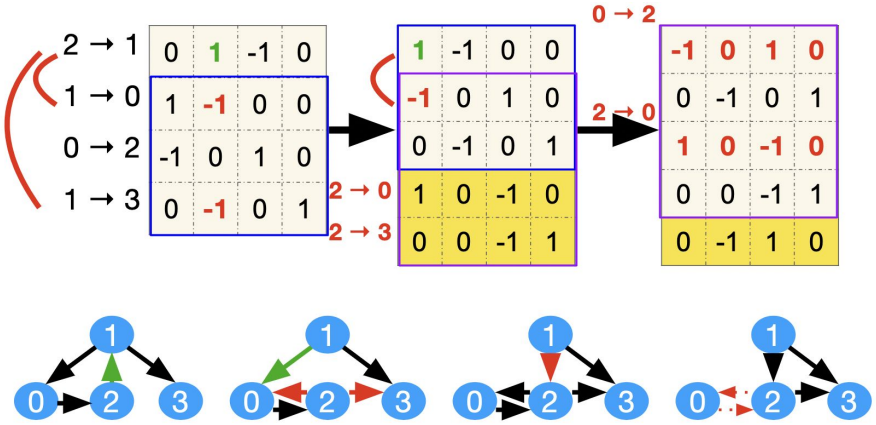
Graph Representation-Incidence Matrix

- Each row corresponds to an edge and each column corresponds to a node.
- Each row is an edge.
- If an edge is from **node 1** to **node 2**, the value of **column 1** will be **-1** and the value of **column 2** will be **1**.
- 0 otherwise.

		0	1	2	3	4	5	6	7
0 → 1	0	-1	1						
0 → 3	1	-1			1				
1 → 2	2		-1	1					
1 → 4	3		-1			1			
2 → 3	4			-1	1				
2 → 5	5			-1			1		
4 → 6	6					-1		1	
5 → 4	7					1	-1		
6 → 7	8							-1	1
7 → 5	9						1		-1

Linear Dependent to Detect Cycle

- If the rows of the incidence matrix of graph G are linearly dependent, then there is a cycle in G.



https://math.stackexchange.com/questions/3906422/submatrix-of-signed-incidence-matrix-of-a-graph-containing-a-cycle

Termination

- If we get **ALL 0** row after addition, then the graph has a cycle.
- If we do addition on **all the edges** and we don’t get ALL 0, then the graph does not have a cycle.
- Hint : You cannot terminate until there remains only one row in the sets.

P1 in colab

Please finish the function **p1_has_cycle()**.

```python
import scipy.sparse
import numpy as np

def p1_has_cycle(sets):
    # TODO
    # return True if the graph has cycle; return False if not
    '''
      HINT: You can `print(sets)` to show what the matrix looks like
        If we have a directed graph with 2->3 4->1 3->5 5->2 0->1
              0  1  2  3  4  5
            0  0  0 -1  1  0  0
            1  0  1  0  0 -1  0
            2  0  0  0 -1  0  1
            3  0  0  1  0  0 -1
            4 -1  1  0  0  0  0
        The size of the matrix is (5,6)
    '''

    return False
```

Problem 2

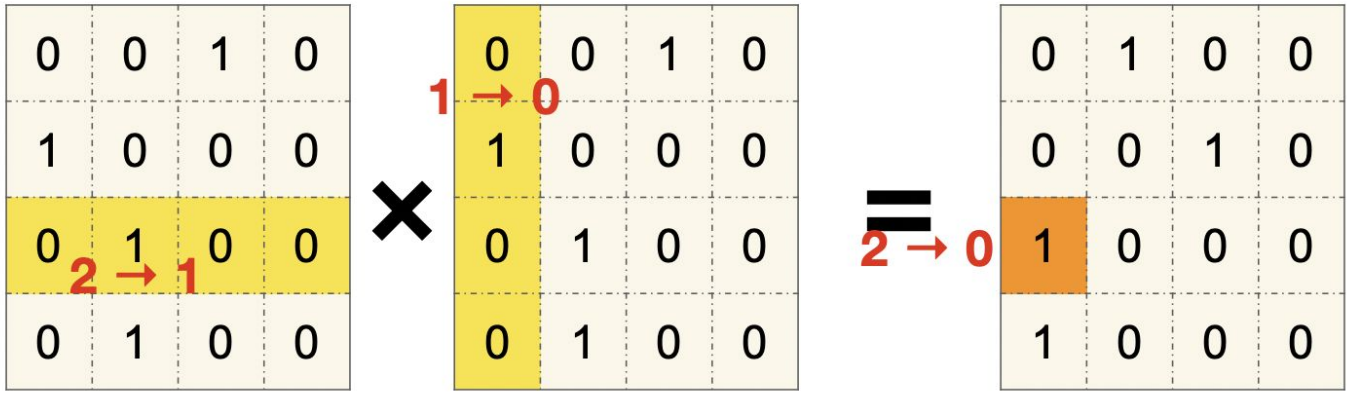
Graph Representation-Adjacency Matrix

- Each row and each column corresponde to a node.
- A cell (x, y) = 1 if there is an edge from node x to node y.
- 0 otherwise.

0 → 1
0 → 3
1 → 2
1 → 4
2 → 3
2 → 5
4 → 6
5 → 4
6 → 7
7 → 5

	0	1	2	3	4	5	6	7
0		1		1				
1			1		1			
2				1		1		
3								
4							1	
5					1			
6								1
7						1		

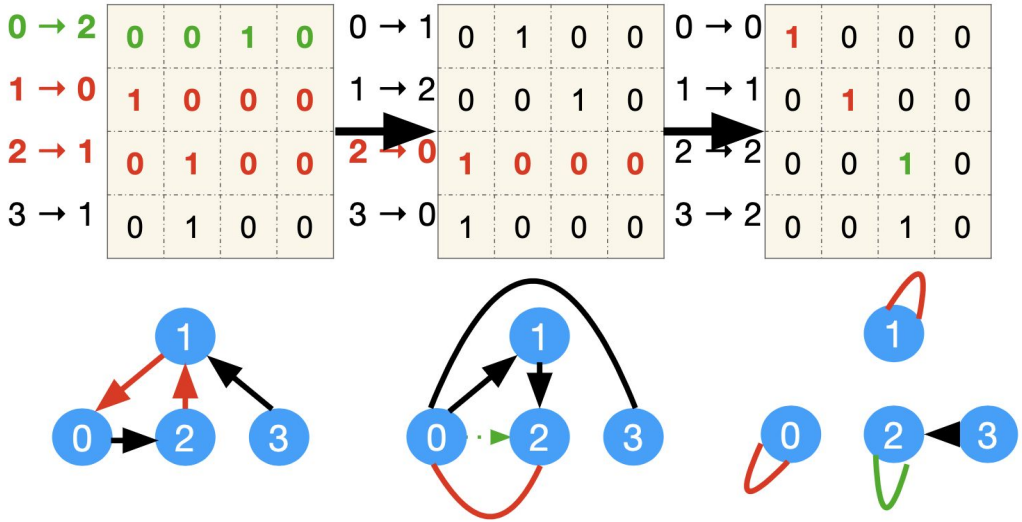
Matrix Multiplication



https://www.ics.uci.edu/~irani/w15-6B/BoardNotes/MatrixMultiplication.pdf

Matrix Multiplication

- If there are non-zero numbers in the diagonal, there are cycles in the graph.



P2 in colab

Please finish the function **p2_has_cycle()**.

```python
import scipy.sparse
import numpy as np

def p2_has_cycle(sets):
    # TODO
    # return True if the graph has cycle; return False if not
    '''
      HINT: You can `print(sets)` to show what the matrix looks like
        If we have a directed graph with 2->3 4->1 3->5 5->2 0->1
              0  1  2  3  4  5
          0   0  1  0  0  0  0
          1   0  0  0  0  0  0
          2   0  0  0  1  0  0
          3   0  0  0  0  0  1
          4   0  1  0  0  0  0
          5   0  0  1  0  0  0
        The size of the matrix is (6,6)
    '''


    return False
```

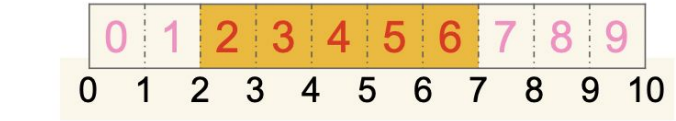
Code implementation

- You should only complete functions **p1_has_cycle()** and **p2_has_cycle()**.
  - True for the graph has cycle.
  - False for the graph does not has cycle.
- To reduce the execution time, you may use scipy.sparse.csr_matrix to implement your code rather than list or numpy array.
  - Faster for computing sparse matix.
  - You can also use numpy to solve P1 and P2.

Python Tips

List

- Generate an empty list.
  - L = List() or L = []
- Get a sublist of a list L.
  - From A to B : L[A : B]
  - From begin to B : L[:B]
  - From A to end : L[A :]



This can be used in Numpy and Scipy too.

```
>>> a = [1,2]
>>> b = a
>>> b.append(3)
>>> a
[1,2,3]
```

- a = b, if b is a class object (e.g. numpy array), this operation is sampe as ”given b an alias”, that is, a and b point to the same memory address.

```
>>> A = [1, 2, 3]
>>> B = A
>>> C = A[:]
>>> C.append(123)
>>> B.append(-17)
>>> print(f"A : {A}\nB : {B}\nC : {C}")
A : [1, 2, 3, -17]
B : [1, 2, 3, -17]
C : [1, 2, 3, 123]
```

Useful functions

	Numpy	Scipy (s_ here means its type is csr_matrix)
Create an object	L = numpy.array([[3, 0],[0, 1]])	s_L = scipy.sparse.csr_matrix([[3, 0],[0, 1]])
Get the shape	L.shape	s_L.shape
Return indices of maximum elements.	numpy.argmax(L)	s_L.argmax()
Return the indices whose values are -1	np.where(L==-1)[0]	np.where(s_L.toarray()==-1)[0]
Return the minimum of the matrix	L.min()	s_L .min()

	Numpy	Scipy (s_ here means its type is csr_matrix)
Get all values at diagonal line	L.diagonal()	s_L.diagonal()
Add (With matrix M)	L = L+M	s_L = s_L + s_M
Delete the first row	L = L[1:]	s_L = s_L[1:]
Stack	numpy.vstack((L,newrow))	scipy.sparse.vstack((s_L,s_newrow))
Multiplication (With matrix M)	L = np.dot(L,M)	s_L = s_L.dot(s_M)

What you should do in HW1

- Open this colab link, and copy the this file to your drive.
- Finish p1_has_cycle(), and p2_has_cycle().
- You can modify seed, #edges, #nodes for debug/test.

```
1 p1_main(student_id)
```

```
The 0th graph AC.
Bug in the 1th graph. P1.
The 2th graph AC.
The 3th graph AC.
The 4th graph AC.
Bug in the 5th graph. P1.
The 6th graph AC.
Bug in the 7th graph. P1.
Bug in the 8th graph. P1.
Bug in the 9th graph. P1.
--- Execution time for p1: 0.132039 seconds ---
```

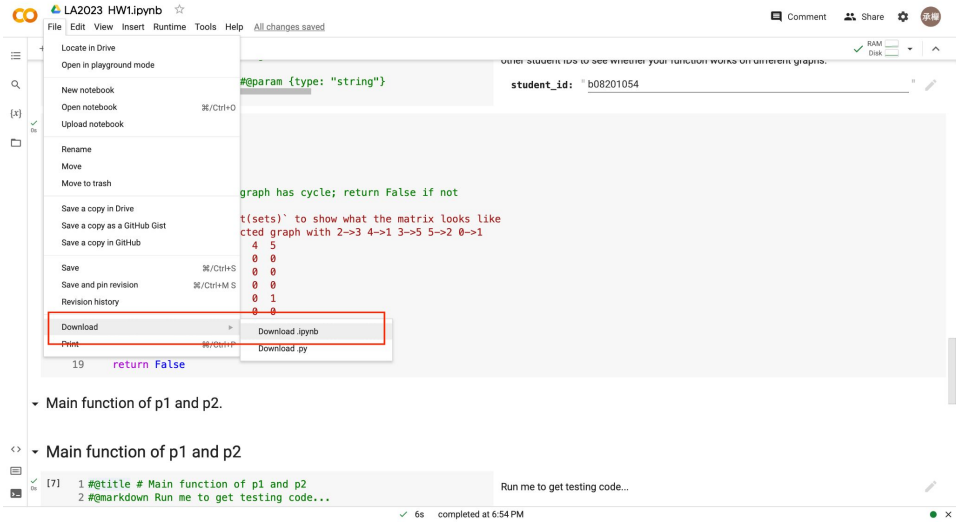
Grading

- **Problem 1** (50%)
  - Group 0 (8% x 5 = 40%)
    - 1000 <= #edges <= 6000, 1000 <= #nodes <= 6000
  - Group 1 (10%)
    - 10000 <= #edges <=12000, 10000 <= #nodes <= 12000
- **Problem 2** (50%)
  - Group 0 (8% x 5 = 40%)
    - 1000 <= #edges <= 6000, 1000 <= #nodes <= 6000
  - Group 1 (10%)
    - 10000 <= #edges <=12000, 10000 <= #nodes <= 12000
- **Time limit : 5mins for each problem.**

Note : There are no self-loops. We will execute Group 0 and Group 1 consecutively for each problem.

Submission Rule

- Download the ipynb file, named it {學號}_HW1.ipynb, and submit it to NTU Cool.
  - e.g. b08201054_HW1.ipynb



Policy

- **No plagiarism.** The first time, you will receive a score of 0 on your homework, and the second time, you will fail this course.
- **Deadline : 2023/10/4 23:59**, score * 0.8 per day, 0 for more than 3 days.
- Incorrect format : score * 0.9

Q & A