Einführung in die diskrete Mathematik Zusammenfassung

Ayushi Tsydendorzhiev

October 11, 2024

Das ist meine Zusammefassung für die Vorlesung Einführung in die diskrete Mathematik, gehalten vom Prof. Dr. Jens Vygen im Wintersemester 23/24. Wie immer, ohne Garantie auf Fehlerfreiheit.

Contents

Algorithmus	Idee	Laufzeit
EULERS ALGORITHMUS	zerlege G in Kreise	$O(m)$
BFS, DFS	Graphendurchmusterung	$O(n+m)$
STRONGLY CONNECTED COMPONENT ALGORITHMUS	zweifaches DFS mit Knotenmarkierungen	$O(n+m)$
KRUSKAL	sortiere die Kanten und füge die minimalen Kanten hinzu	$O(m \log n)$
PRIM	sukzessive wähle die kleinste Kante aus $\delta(V(T))$	$O(m + n \log n)$
EDMOND'S BRANCHING ALGORITHMUS	sukzessive kontrahiere Kreise	$O(mn)$
DIJKSTRAS ALGORITHMUS	Bellmans Optimalitätsprinzip	$O(m + n \log n)$
MOORE-BELLMAN-FORD	finde den günstigsten Vorgänger	$O(nm)$
SHALLOW-LIGHT-BÄUME	finde MST und Kürzeste-Wege-Baum, veroppele alle Kanten in MST, finde eulerschen Spaziergang. Ist Distanz größer als in dem KWB, so füge den kürzesten Weg hinzu	$O(m + n \log n)$
MINIMUM-MEAN-CYCLE	optimiere Differenzen	$O(nm)$
FORD-FULKERSON	finde beliebige augmentierende Flüsse	quasi-polynomiell
EDMONDS-KARP	finde den kürzesten augmentierenden Fluss	$O(nm^2)$
DINICS ALGORITHMUS	finde blockierende Flüsse	$O(n^2)$
PUSH-RELABEL	Levels	$O(n^2\sqrt{m})$
Nagamochi Ibaraki Karger Stein Minimum-Mean-Cycle-Cancelling-Algorithmus		

1 *Verwirrte Definitionen*

Etwas von mir: Unterscheidung zwischen

- Kantenfolge/Kantenzug
  - eine nicht leere Folge $v_1, e_1, v_2 \dots v_k, v_{k+1}$,
  - minimum eine Kante,
- Spaziergang
  - ohne wiederholte Kanten,
- Weg

  - ohne wiederholte Knoten,
- Kreis
  - geschlossener Weg,
  - minimum zwei Knoten,
- vollständiger Graph K_n
  - je zwei Knoten verbunden,
  - einfach, definiert ab $n \geq 1$.
- vollständiger bipartiter Graph $K_{n,m}$
  - je zwei Knoten aus $\{1, .., n\}$ und $\{n+1, .., n+m\}$ sind verbunden,
  - definiert für $n \geq 1$ oder $m \geq 1$.
- Wald
  - ungerichteter Graph ohne Kreise
- Baum
  - zusammenhängender Wald
- Branching
  - directed forest + each vertex has max 1 incoming edge
- Arboreszenz
  - directed tree

2 *Witzige Eigenschaften*

- bipartite Graphen
  - G bipartit $\iff$ enthält keinen ungeraden Kreis
- Baum
  - ungerichtet, zusammenhängend und azyklisch
  - mindestens zwei Knoten mit Ausgangsgrad 0
- Präprozessing für Graphen
  - für viele Graphenalgorithmen kann man annehmen, dass G einfach und zusammenhängend ist, somit $n - 1 \leq m < n^2$

2.1 *Laufzeiten für Basisoperationen auf Graphen*

Eine Adjazenzmatrix $(a_{ij})_{1\leq i,j\leq n}$ beschreibt den Graphen G mit $a_{ij} = 1$ falls $\{i,j\} \in E(G)$ und sonst 0.

Eine Adjazenzliste ist eine Liste von Listen in der n-te Liste den n-ten Knoten repräsentiert und Informationen über seine Nachbarn speichert.

	Adjazenzmatrix	Adjazenzliste
Speicherplatz	$O(n^2)$	$O(n+m)$
Kanten einfügen	$O(1)$	$O(1)$
Knoten einfügen	$O(n)$	$O(1)$
Kanten löschen	$O(1)$	$O(1)^*$
Knoten löschen	$O(n)$	$O(n)$

* – geht nur mit doppelt verketteten Listen und ist eher aufwendig + nervig.

3 *Vorlesungsstoff*

Vorlesung 1, 10.10.23

3.1 *Euler und Hamilton*

- Eulerscher Spaziergang
  - geschlossen, enthält jede Kante genau einmal ohne Wiederholung,
  - notwendige und hinreichende Bedingung (Euler, 1736):
    G hat eulerschen Spaziergang $\iff$ G eulersch.
    * G eulersch, wenn jeder Knoten geraden Grad hat.
  - Eulers Algorithmus, Laufzeit $O(m)$,
    * starte mit Knoten v_0, finde einen Kreis $v_0, v_1, \ldots, v_n$ und entferne die Kanten aus G
    * Knoten $v_1, ..., v_n$ haben immer noch geraden Grad, rekursiv für jeden Knoten auf dem Kreis finde geschlossenen Kreis/Spaziergang, entferne ihn (somit bleibt der Graph immer noch eulersch) und wiederhole solange es noch Kanten vorhanden sind. Am Ende füge alle Kreise/Spaziergänge in der richtigen Reihenfolge zusammen
    * findet Zerlegung von G in Kreise

Vorlesung 2, 12.10.23

- Hamiltonkreis
  - enthält jeden Knoten genau einmal ohne Wiederholung,
  - hinreichende Bedingung (Dirac, 1952):
    G einfacher ungerichteter Graph mit $\min\{\delta(v) \mid v \in V(G)\} \geq \frac{n}{2}$.
    Gegenbeispiel: ein Kreis der Länge 100.

3.2 *Zusammenhang*

- Schwacher Zusammenhang in Graphen
  - BFS, DFS, lineare Laufzeit
- Starker Zusammenhang in Graphen
  - Strongly Connected Component Algorithm, Laufzeit $O(n+m)$.
    - zweimaliges DFS über alle Zshgskomponenten mit Labels
    - post-order,
    - bestimmt topologische Ordnung bei Kontraktion von den starken Zusammenhangskomponenten,
    - entscheidet ob topologische Ordnung existiert $\iff$ starke Zusammenhangskomponenten bestehen aus einelementigen Mengen (sonst existiert ein Kreis)

3.3 *Aufspannende Bäume und Arboreszenzen*

Vorlesung 3, 17.10.23

- Satz von Cayley
  - Anzahl der aufspannenden Bäume in $K_n = n^{n-2}$
  - häufig interessieren wir uns für **einen** optimalen aufspannenden Baum aus n^{n-2}
  - Anzahl Branchings in $K_n = (n+1)^{n-1}$
- Äquivalente Aussagen zu T minimaler aufspannender Baum
  - für jede Kante $\{x, y\}$ nicht in T, gilt: jede Kante auf dem $x-y$-Weg in T hat niedrigere Kosten
  - für jede Kante e in T gilt: e ist die kostenminimale Brücke zwischen den zwei Zshgskomponenten in $T - e$
  - alle Kanten $\{e_1 \dots e_n\}$ in T können so angeordnet werden, dass jedes e_i jeweils in einem Schnitt X von G minimal ist und **nicht** mit e_j konkurriert
- Algorithmen für Minimum Weight Spanning Tree
  - Kruskals Algorithmus, Laufzeit $O(m \log n)$
    - sortiere die Kanten und füge die minimalen Kanten hinzu
    - $O(m \log n)$ geht wenn wir Branchings und die Länge des längsten Weges in den Arboreszenzen merken. Sollten zwei Arboreszenzen zusammgefügt werden, so wird A zu A' hinzugefügt, falls $l(A) < l(A')$.
  - Prims Algorithmus, Laufzeit $O(m + n^2)$

    * wähle einen Knoten v und füge immer minimale erreichbare Kanten hinzu
    * mit Fibonacci-Heap in $O(m + n \log n)$ Vorlesung 4, 19.10.23

- Reduzierbarkeit
  - Maximum Weight Forest $\iff$ Minimum Spanning Tree
    * Delete edges with negative costs, add minimal amount of edges such that G connected, multiply all costs with -1. Find MST, delete all previously added edges, get MWF.
    * Invert costs $c' = c + K > 0$ and find MWF. Since any solution of MWF is a tree, get MST.
  - Maximum Weight Branching $\iff$ Minimum Weight Arborescence $\iff$ Minimum Weight Rooted Arborescence
    * MWA-MWB: invert costs $c' = c + K > 0$ and find MWB, get MWA.
    * MWB-MWRA: add vertex r connected to every vertex. Invert costs. Arborescence in r is solution for MWB.
    * MWRA-MWA: add vertex s connected only to r. Any solution to MWA starts from s, delete edge (s, r) and get MWRA.

Vorlesung 5, 24.10.23

- Fibonacci-Heap (U, E)
  - Fibonacci, weil Wurzel in der Arboreszenz hat k Kindern $\implies$ Arboreszenz enthält mindestens Fibonacci$(k + 1)$ viele Knoten $(1, 1, 2, 3, 5, 8, \ldots)$
  - ein Branching (U, E) mit Markierung $\varphi : U \to \{0, 1\}$
  - hat vier Eigenschaften
    * je höher die Kante im Baum, desto kleiner der Key
      · $(u, v) \in E \iff \text{key}(u) \leq \text{key}(v)$
      · garantiert gute Laufzeit für DeleteMin, da man nur über Wurzeln der Arboreszenzen sucht
    * Nachkommenschaft-Garantie
      · $|\delta^+(v)| + \varphi(v) \geq i - 1$
      · verhindert zu tiefe Bäume
      · φ kompensiert ein verlorenes Kind
      · Wurzeln haben exponentiell viele Nachkommen in der Anzahl von Kindern
    * Wurzeln dürfen nicht dieselbe Kinderanzahl haben
      · verhindert zu viele Bäume
    * Jede Wurzel hat mindestens $\sqrt{2}^k$ Nachkommen

        - folgt aus 2.ter Eigenschaft
        - impliziert Existenz von höchstens $2\log_2 p + 1$ Wurzeln gibt
    - technische Umsetzung
        * Speichere Wurzeln in $b := \{0, 1, 2, \ldots, \lfloor 2\log_2 p \rfloor\} \to U$ mit $b(k) = u$ falls u Wurzel und k Kinder hat
            - Umkehrung gilt nicht, weil zwei Knoten können durchaus denselben Grad haben, müssen nur entscheiden, wer Wurzel wird
            - wegen φ kann die Anzahl an Kindern $k - 1$ sein
        * doppelt verkettete Liste der Kinder jedes Elements
            - damit delete in $O(1)$ läuft
        * Pointer zum Eltern
        * den Ausgangsgrad jedes Elementes
    - Insert$(r, \text{key}(r))$
        * $\varphi(r) = 0$
        * insert den Knoten r in die Liste b an geeigneter Stelle
        * falls besetzt von v dann verbinde Arboreszenzen je nach $\text{key}(v), \text{key}(r)$ und füge die neue Arboreszenz in b wieder ein.
        * Laufzeit $O(1)$
        * erfolgt rekursiv
    - DeleteMin
        * durchlaufe b und finde Wurzel mit $\min \text{key}(r)$
        * entferne r und reinsert alle Waisen
        * Laufzeit $O(\log n)$
    - DecreaseKey$(v, \text{key}(v)$
        * problematisch, wenn die 1.te Bedingung verletzt wird
        * finde den längsten Weg P zu v so dass jede Kante u auf dem Weg $\varphi(u) = 1$ hat
        * invertiere Werte von φ für Knoten in P
        * lösche alle Kanten in P
        * replante alle Waisen
        * $\varphi(x)$ ist die Kompensation für verlorenes Kind damit 2.te Bedingung nicht verletzt wird
        * gleichzeitig auch Markierung dass ein Kind nur einmal weggenommen werden kann
    - Laufzeiten $O(m + p + n \log p)$

Vorlesung 6, 26.10.23

- Algorithmen für Maximum Weight Branching

- KARP'S ALGORITHMUS
  - Idee: für jeden Knoten nehme die Kante mit maximalem Gewicht
  - Problem: es können Kreise entstehen, also es muss mindestens eine Kante gelöscht werden.
  - Lösung: nach Karp's Lemma reicht es, bereits eine Kante zu löschen

Das ist natürlich kein Algorithmus, sonder beschreibt einen möglichen Lösungsansatz für dieses Problem.

- Karp's Lemma
  - Idee 1: Branching B = underlying graph has no circles + max 1 incoming edge. Relax the condition to allow circles. Then MWB is containted in a maximum weight graph B_0 such that for any circuit in B_0, all but one edge is contained in B
  - Idee 2: es ist naheliegend, einfach die schwerste Kante zu wählen. Der resultierende Graph kann enthalten Kreise. Das Lemma sagt, es reicht, eine Kante aus den Kreisen zu entfernen, um ein optimales Branching zu bekommen.
- EDMOND'S BRANCHING-ALGORITHMUS
  - Idee: Finde B_0 und kontrahiere alle Kreise zu einem Graphen B_1. Wähle Gewichte in B_1 so, dass optimaler Branching in B_1 optimal in B_0 ist.

3.4 *Kürzeste Wege von einem Knoten aus*

- Preliminaries:
  - Bellmans Optimalitätsprinzip $\iff$ optimale Entscheidungsfolgen sind im jeden Schritt optimal
  - konservative Gewichte $c' \iff$ es existiert kein Kreis negatives Gewichts

Unsere Algorithmen basieren sich im Wesentlichen auf diesem Prinzip

Vorlesung 7, 31.10.23

- Algorithmen für SHORTEST PATH PROBLEM
  - DIJKSTRA'S ALGORITHMUS
    - Gewichte ≥ 0
    - Laufzeit $O(m + n^2)$ naiv, $O(m + n \log n)$ mit Fibonacci-Heaps
  - MOORE-BELLMAN-FORD ALGORITHMUS
    - beliebige Gewichte
    - finde den günstigsten Vorgänger + überprüfe final ob negative Zyklen existieren
    - Laufzeit $O(mn)$

* – aus sdfsdfeiner negativen Kante entstehen zwei Kanten in beide Richtungen, MBF returns negativen Kreis

- Die Algorithmen finden auch kürzeste Wege Bäume bzw. Arboreszenzen ausgehend vom Anfangsknoten s

8. Vorlesung, 02.11.23

	gerichtet	ungerichtet
Gewichte $= 1$	BFS	BFS
Gewichte ≥ 0	Dijkstra	Dijkstra mit Kanten in beide Richtungen
konservative Gewichte	MBF	Masterstudium*
beliebige Gewichte	NP-schwer	NP-schwer

Table 1: Übersicht Algorirthme für kürzeste-Wege-Problem

3.5 Shallow-Light Bäume

Diese Bäume sind ein Trade-off zwischen einem minimalen aufspannenden Baum und kürzeste-Wege-Baum von r aus. Das Ziel ist quasi ein MST, dessen längster Weg um maximal $1 + \epsilon$ von dem optimalen abweicht.

- Algorithm für Shallow-Light Baum
  - Annahmen: ungerichtet, einfach
  - Berechne den MST und den kürzeste-Wege-Baum
  - Verdoppele alle Kanten im MST und finde eulerschen Spaziergang
  - Gehe entlang des eulerschen Spaziergangs
    - Falls zu einem Zeitpunkt t die Kosten des aktuellen Weges die $1 + \epsilon$ Kosten des kürzesten Weges übersteigen, dann füge diesen kürzesten Weg zum Graphen hinzu.

3.6 Kürzeste Wege zwischen allen Knotenpaaren

- c konservativ $\iff$ existiert kein negativer Kreis C $\iff$ existiert zulässiges Potential π
  - zulässiges Potential $\pi : V(G) \to \mathbb{R}$ $\iff$ c_π nicht negativ für alle Kanten
  - reduzierte Kosten $c_\pi(e) = c(e) + c_\pi(x) - c_\pi(y)$
    - Kosten eines Kreises $c_\pi(C) = c(C)$
    - Kosten eines Weges $c_\pi(P) = c(P) + c(v_0) - c(v_n)$
  - π mit MBF in $O(nm)$ berechenbar
    - dafür setze $\pi(v) = l(v)$
    - in MBF, $l(y) \geq l(x) + c(e) \implies c(e) + l(x) - l(y) \geq 0$

Diese Kosten sind nicht wirklich "reduziert".

- All Pairs Shortest Paths Problem

  - Laufzeit $O(mn + n(m + n \log n)$
    * zuerst mit MBF zulässiges Potential π berechnen
    * auf dem Graphen mit reduzierten Kosten n-mal Dijkstra anwenden $\implies$ wegen π sind die Kantengewicht nicht negativ.
  - Anwendungsbeispiel: metrischer Abschluss von (G, c)
    * einfacher Graph $(\overline{G}, \overline{c})$, dessen Kanten (x, y) kürzeste-Wege-Gewichte sind (und existieren)

3.7 Minimum Mean Cycle Problem

- Directed Minimum Mean Cycle Problem
  - finde Kreis mit minimalen Durchschnittsgewicht der Kanten $\mu(G, c)$ oder entscheide, dass G azyklisch ist.
- Satz (Karp, 1978)
  - alle Kanten sind von s aus erreichbar,
  - Kantenfolge $F_k(x)$ der Länge k mit kleinsten Kosten von x
  - für jedes k betrachte $\{\frac{F_n(x)-F_1(x)}{n-1}, \frac{F_n(x)-F_2(x)}{n-2}, \ldots, \frac{F_n(x)-F_{n-1}(x)}{n-1}\}$
  - sei $M(x)$ das Maximum dieser gewichteten Differenzen, dann gilt $\mu(G, c) = \max_{x \in V(G)} M(x)$

9. Vorlesung, 07.11.23

- Minimum-Mean-Cycle-Algorithmus
  - Laufzeit $O(mn)$
    * füge einen Knoten s mit Nullkanten zu jedem Knoten in G
    * setze $F_0(s) = 0, F_0(x) = \infty$
    * für $1 \ldots i \ldots n$ nehme Kanten mit $F_{i-1}(x) < \infty$, finde minimale Kante von w nach x
    * speichere die Kante und den Vorgänger
    * berechne max Differenz zwischen $F_n(x) - F_k(x)/(n - k)$ für jeden Knoten
    * C ist irgendein Kreis aus $p^n(x), p^{n-1}(x), \ldots$
    * falls azyklisch, so hat der längste Weg die Länge $n - 1 \implies F_n(x) = \infty$ für alle Knoten

Wir gehen den Graphen in Stufen Schritt für Schritt durch und nehmen jedes Mal die kleinste Kante $\implies$ Minimieren über x. Wie cascading waterfall.

3.8 Maximale Flüsse I

Gegeben ist immer (G, u, s, t), wobei G gerichteter Graph, u die Kapazität von Kanten, s Quelle und t Target ist.

u steht für upper bound

- Maximum Flow Problem

  - Gegeben (G, u, s, t) finde maximalen Fluss von s nach t

- Max-Flow-Min-Cut-Theorem
  - Der Wert von maximalem Fluss von s nach t ist gleich dem Wert des minimalen s-t-Schnittes

10. Vorlesung, 09.11.23

- Ford-Fulkerson-Algorithmus
  - nur für positive rationale Kapazitäten
  - Laufzeit $O(??)$
    * finde einen Fluss von s nach t
    * finde eine Bottleneck-Kante e
    * augmentiere f um die Residualkapazität von e und starte neu

Ein Struktursatz für maximale Flüsse:

- Flussdekompositionssatz
  - jeder Fluss lässt sich als Summe von Elementarwegen und Kreisen darstellen
    * Elementarweg = konstanter s-t-Weg
    * Elementarkreis = konstanter Kreis in G

3.9 Sätze von Menger

Hat ein Graph alle Kanten mit Kapazität 1, so hat ein ganzzahliger Fluss eine Zerlegung in kantendisjunkte Wege und Kreise. Das motiviert die folgenden Sätze.

- Satz von Menger 1:
  - k kantendisjunkte Wege $\iff$ nach $k-1$ entfernten Kanten bleibt t erreichbar
- Satz von Menger 2
  - s und t nicht benachbart
  - k intern kantendisjunkte Wege $\iff$ nach $k-1$ entfernten Knoten bleibt t erreichbar

11. Vorlesung, 14.11.23

Ein wichtiges Korollar aus den Sätzen, für **ungerichtete** Graphen + neue Vokabeln:

- G hat mehr als zwei Knoten:
  - k-kantenzusammenhängend := zusammenhängend ohne $k-1$ belieibige Kanten

  - *k-kanten*zusammenhängend $\iff$ existieren *k-kanten*disjunkte Wege

- G hat mehr als k Knoten:
  - k-zusammenhängend := zusammenhängend ohne $k-1$ beliebige Knoten
  - k-zusammenhängend $\iff$ existieren k intern disjunkte Wege
    - intern disjunkt = kantendisjunkt + innere Knoten disjunkt
- Knotenzusammenhang, Kantenzusammenhang := $\max\{k \in \mathbb{N} \cup \{0\} \mid G \text{ ist } k\text{-(kanten-)zusammenhängend}\}$
- es gilt immer Knotenzusammenhang $\leq$ Kantenzusammenhang
- Block
  - maximal zusammenhängende Graphen ohne *Artikukationsknoten*
  - Klassifikation von Blöcken
    - maximaler 2-zusamenhängender Teilgraph
    - Brücke
    - isolierter Knoten
    - $\implies$ zwei Blöcke haben höchstens einen gemeinsamen Knoten

Artikulationsknoten = Brücke in Knotenversion

Ein Struktursatz für 2-zusammenhängende Graphen

- Ungerichteter Graph 2-zusammenhängend $\iff$ hat echte Ohrenzerlegung
  - Ohrenzerlegung = eine Folge $(r, \emptyset) + P_1 + P_2 + \ldots + P_k = G$
    - induktive Zerlegung von G in Ohren P_k, r ist der Anfangsknoten
    - jedes Ohr P_k ist entweder
      - ein Weg mit Anfangs- und Endknoten in $P_{i<k}$
      - ein Kreis mit genau einem Knoten in $P_{i<k}$
      - $P_0 := (r, \emptyset)$
    - alternativ ist P_k ein (maximaler) Weg oder ein Kreis P, in dem interne Knoten Grad 2 haben und Anfangs- und Endknoten Grad ≥ 3 haben
  - echte Ohrenzerlegung
    - P_1 ist ein Kreis und $P_{i>1}$ sind Wege
- Aus der Übung:

3.10 Maximale Flüsse II

Mit FORD-FULKERSON haben wir beliebige augmentierende s-t-Wege betrachtet. Wir sollten aber die kantenminimale augmentierenden Wege betrachten. Erzielen Verbesserung weil polynomielle Laufzeit.

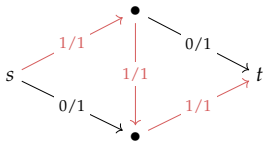
- EDMONDS-KARP-ALGORITHMUS
  - nur für positive reelle Kapazitäten
  - wie FORD-FULKERSON, nur mit BFS
  - Laufzeit $O(nm^2)$
    - Algorithmus terminiert nach höchstens $\frac{nm}{2}$ Augmentierungen
    - Jede Augmentierung hat Laufzeit $O(m)$

12. Vorlesung, 16.11.23

Die Idee von DINIC'S ALGORITHMUS ist, dass wir nicht entlang eines Weges, sondern entlang eines Flusses augmentieren. Dafür müssen wir aber gewisse neue Begriffe einführen:

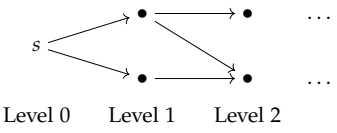
- s-t-Präfluss := Fluss, aber $ex(v)$ darf größer o sein
  - *aktiver* Knoten $\iff ex(v) > 0$ und $v \neq s, t$
- s-t-Fluss blockierend := der Graph G mit nicht saturierten Kanten enthält keinen s-t-Weg
  - Abschwächung von augmentierendem Fluss (ohne Rückwartskanten)
  - maximaler Fluss ist blockierend, aber Umkehrung gilt nicht
  - hier noch etwas falsch, siehe Shimon p. 95

Beispiel zu einem blockierenden, nicht maximalen s-t-Fluss:



- Level-Graph G_f^L := der Graph der kürzesten Wege in G_f, ausgehend von s
  - in linearer Zeit mit BFS berechenbar
  - azyklisch, keine Kante kann Levels überspringen
  - definiert topologische Ordnung auf G, wobei s ganz links und t ganz rechts stehen
  - **wichtig:** für max. Fluss reicht, einen blockierenden Fluss in G_f^L zu finden

Beispiel zu einem Level-Graph G_f^L:



- DINIC'S ALGORITHMUS
  - finde G_f^L mit BFS in $O(m)$
  - finde blockierenden Fluss f' in G_f^L in $O(n^2)$
    - PUSH: scanne die Knoten nach topologischer Ordnung (in G_f^L) und pushe so viel wie möglich von links nach rechts

    * BALANCING: nehme rechtesten aktiven Knoten v, schicke $ex(v)$ zurück, markiere als geschlossen, wiederhole.
    * augmentiere f entlang f'
    * terminiert nach $O(n)$ Iterationen, da mindestens ein Knoten als geschlossen markiert wird.
  - ist $f' = 0$, so sind wir fertig, sonst starte neu
  - Laufzeit $O(n^3) \leq O(nm^2)$
    * Algorithmus terminiert nach höchstens n Augmentierungen (Gewinn Faktor $O(m)$ im Vergleich zu EDMONDS-KARP)
    * es gibt mehrere Ansätze, wie man den blockierenden Fluss findet. Die Laufzeit ergibt sich dann im Wesentlichen aus $O(n) \cdot O(\text{blockierenden Fluss finden})$.
    * geht in $O(n^2)$ mit Ansatz von Karzanov; siehe zwei Beweise in der Mitschrift oder S. 196-198 im Buch

13. Vorlesung, 21.11.23

Wir lernen PUSH-RELABEL-ALGORITHMUS kennen. Wie immer, werden zuerst einige neue Definitionen und Lemmas eingeführt.

- s-t-Präfluss f ist ein maximaler s-t-Fluss, wenn
  - es gibt keinen aktiven Knoten und
  - es gibt keinen augmentierenden s-t-Fluss in G_f

Alle bisherigen Algorithmen haben Bedingung 1 stets erfüllt und terminierten wenn Bedingung 2 erfüllt wurde. PUSH-RELABEL erfüllt stets Bedingung 2 und terminiert bei Bedingung 1

- Distanzmarkierung ψ bezüglich s-t-Präflusses f
  - $\psi(s) = n$, $\psi(t) = 0$
  - für Kante $e = (v, w)$ gilt: $\psi(v) \leq \psi(w) + 1$
    * jede Kante kann maximal um 1 absteigen
    * Kanten die um 1 absteigen sind *erlaubte* Kanten
  - impliziert, dass es nie einen s-t-Weg in G_f gibt, da der längste Weg nur um $n - 1$ absteigen kann und nicht um n. Die Bedingung 2 ist also stets erfüllt und wir arbeiten nur daran, den Überschuss zu verteilen.
- PUSH-RELABEL-ALGORITHMUS
  - Laufzeit $O(n^2\sqrt{m}) \leq O(n^3)$
  - Initialisierung:
    * $\psi(s) = n, \psi(v) = 0$ sonst
    * saturiere alle aus s ausgehende Kanten
    * G_f hat keine (s, v) Kanten, nur Rückwartskanten $(v, s) \implies$ Distanzinvariante ist nicht verletzt
  - Main Loop:

    * solange es einen aktiven Knoten v gibt, wähle v mit maximalem $\psi(v)$
    * falls existiert aus v ausgehende und erlaubte Kante e PUSH(e)
        · setze $\Delta = \min\{$Restkapazität von e, Überschuss $v\}$
        · augmentiere f entlang e um Δ
    * sonst RELABEL(v)
        · erhöhe $\psi(v)$ um 1

	Idee	Laufzeit
Ford-Fulkerson	augmentiere entlang eines beliebigen Flusses	quasi-polynomiell
Edmonds-Karp	augmentiere entlang des kürzesten Flusses	$O(nm^2)$
Dinic	augmentiere blockierende Flüsse	$O(n^3)$
Push-Relabel		$O(n^2\sqrt{m})$

Table 2: Übersicht Algorithme für Maximum-Flow-Probleme

3.11 Minimale Schnitte (ungerichtet)

14. Vorlesung, 23.11.23

Eine leichte MAX-FLOW-Variation. In der Vorlesung wird vor allem der ungerichtete Fall behandelt, da die Ungerichtheit (?) sich gut "ausnutzen" lässt.

- MIN s-t-CUT PROBLEM
  - finde $X \subset V(G)$ mit minimaler Kapazität $u(\delta(X))$
  - Laufzeit wie MAX-FLOW (insb. $O(n^2\sqrt{m})$)
    * im gerichteten Fall finde Max-Flow f und wähle $X =$ von s aus in G_f erreichbare Knoten
      · funktioniert, weil nach Augmentierung G_f nicht zusammenhängend ist
    * im ungerichteten Fall füge zwei gerichtete Kanten (v, w), (w, v) für jede ungerichtete Kante $\{v, w\}$, dann wie oben
- lokaler Kantenzusammenhang $\lambda_{s,t}$
  - minimale Kapazizät eines s und t trennenden Schnittes
  - im Fall $u = 1$ ist gleich Anzahl der disjunkten s-t-Wege
  - es gilt "Dreiecksungleichung" $\lambda_{ik} \geq \min(\lambda_{ij}, \lambda_{jk})$
- minimale Kapazität $\lambda(G)$ eines Graphen

  - Minimum über alle $\lambda_{s,t}$
  - im Fall $u = 1$ ist gleich Kantenzusammenhang
- naive Lösungsansätze für Berechnung von $\lambda(G)$
  - berechne $\lambda_{s,t}$ für alle $\binom{n}{2}$ Paare $\implies$ $\binom{n}{2}$ Max-FLow
  - berechne $\lambda_{s,t}$ für s fix $\implies$ $(n-1)$ Max-Flow
- Algorithmus von Nagamochi und Ibaraki
  - schneller als ein einziger Max-Flow $O(mn + n^2 \log n) \leq O(n^2\sqrt{m})$,
  - MA-Reihenfolge (maximum adjacency)
    - skipped
- Algorithmus von Karger und Stein (Übung)
  - kontrahiere zufällige Kanten gewichtet nach Kapazität solange nur zwei Knoten geblieben sind
  - diese zwei Knoten repräsentieren einen Schnitt in G
  - wiederhole k-mal und wähle das Minimum

3.12 Kostenminimale Flüsse

15. Vorlesung, 28.11.23

Für ein gegebenes Problem gibt es viele Matchings. Wir wollen das optimale Matching in Bezug auf eine Kostenfunktion c finden.

- Beispiele sind:
  - Kürzeste-Wege-Problem
    - setze $c = 1$, dann minimaler Fluss = kürzester Weg
  - Max-Flow
    - setze $c = 0$, füge Kante (t, s) mit negativen Kosten hinzu, dann minimaler Fluss = Max-Flow
  - Min-Cost-Bipartites-Matching
- MINIMUM-COST-FLOW-PROBLEM
  - Instanz (G, u, b, c)
    - gerichteter Graph G, Kapazitäten u, Balance b, Kosten c
    - Knoten mit $b > 0$ (Angebot) sind Quellen, $b < 0$ (Nachfrage) sind Senken

b "generiert" bzw. "absorbiert" flow

  - b-Fluss
    - erfüllt $b(v) = \sum f(\delta^+(v)) - \sum f(\delta^-(v))$
    - existiert gdw $\sum u(\delta^+(X)) \geq \sum b(X)$ für alle $X \subset V(G)$

    * statt augmentierender Wege habe augmeniterende Kreise
  - Ziel: finde b-Fluss mit geringsten Kosten
- HITCHCOCK-PROBLEM
  - Minimum-Cost-Flow-Problem auf einem bipartiten Graph $V = A \cup B$ und $E \subseteq A \times B$ mit Quellen A und Senken B
  - äquivalent zu MINIMUM-COST-FLOW-PROBLEM

16. Vorlesung, 30.11.23

- Struktursatz für Zirkulationen:
  - jede Zirkulation lässt sich als Summe von Elementarwegen und Kreisen darstellen
  - Spezialfall Flussdekompositionssatz für s-t-Flüsse mit Wert $= 0$
- f-augmentierender Kreis
  - ein Kreis in G_f
- Optimalitätskriterium
  - b-Fluss f kostenminimal $\iff$ es gibt keinen f-augmentierenden Kreis mit negativen Kosten
  - Beweisidee:
    * sei f nicht kostenminimal, dann gibt es f' mit geringeren Kosten. Betrachte Differenzfluss $g = f' - f$ mit $c(g) < 0$ und g Zirkulation. Zerlege g in einzelne Kreise, mindestens eins davon ist f-augmentierend in G_f
    * gibt es einen f-augmentierenden Kreis mit negativen Kosten, so können wir entlang des Kreises augmentieren und f war kein kostenminimaler Fluss
  - äquivalent zur Existenz vom zulässigen Potenzial, da negative Kreise $\iff$ existiert kein zulässiges Potenzial
  - *insgesamt: we care about negative circuits in G_f because they lower the costs of any b-flow and they should obviously be b-flow-compatible, e. g. delta flow < capacity - flow*
- MINIMUM-MEAN-CYCLE-CANCELLING-ALGORITHMUS
  - finde irgendeinen b-Fluss
  - finde einen Kreis mit minimalem Durchschnittsgewicht
  - falls der Kreis positives Durchschnittsgewicht hat oder G_f azyklisch ist, dann fertig

vom Prof: Laufzeitabschätzung im Unterschied zu MaxFlow deutlich schwieriger. Es hat einige Jahrzehnte (Ende der 80er) gedauert, bis ein streng polynomieller Algorithmus überhaupt gefunden wurde.

- Laufzeitabschätzung für MINIMUM-MEAN-CYCLE-CANCELLING-ALGORITHMUS
  - Lemma: Gegeben (G,u,b,c). Sei $f_1, ..., f_t$ eine Folge von b-Flüssen so dass $f_i + 1$ aus f_i entsteht indem wir entlang von C_i ein Kreis mit minimalem Durschschnittsgewicht $\mu(f_i)$ augmentieren. Dann gilt folgendes:
    - das Durchschnittsgewicht steigt oder bleibt gleich
    - blabla wir machen echten Fortschritt
  - Beweis Lemma 9.9 (a):

Vorlesung 17, 05.12.23

- Fortsetzung
  - Beweis Lemma 9.9 (b):
- Lza 2 Beweis 2 für Algorithmus (der erste zeigt schwache polynomielle Laufzeit, der zweite streng polynomielle Laufzeit)

Vorlesung 18, 07.12.23

- Satz für SUKZESSIVE-KÜRZESTE-WEGE-ALGORITHMUS
  - gegeben kostenminimaler b-Fluss f + kürzester s-t-Weg P bezüglich c in G_f, augmentiere f entlang P
  - habe kostenminimalen b'-Fluss f' für ein geeignetes b'
- SUKZESSIVE-KÜRZESTE-WEGE-ALGORITHMUS
  -
- ZUORDNUNGSPROBLEM

3.13 Turingmaschinen, Berechenbarkeit

Vorlesung XX, 09.01.24

- Alphabet $A = \{0, 1\}$, $\overline{A} = A \cup \{\sqcup\}$, String = endliche Folge von Elementen aus A, $A^0 = \{\sqcup\}$, $A^1 = \{0, 1, \sqcup\}$, $A^2 = \{00, 01, 10, 11, 0\sqcup, \ldots\}$, $A^* = \cup_i A^i$, Sprache über $A \subseteq A^*$
- **Turingmaschine** ist eine Funktion Φ

$$\Phi : \underbrace{\{0, \ldots, N\}}_{\text{Befehl}} \times \underbrace{\overline{A}}_{\text{Buchstabe}} \longrightarrow \underbrace{\{-1, \ldots, N\}}_{\text{Befehl}} \times \underbrace{\overline{A}}_{\text{Buchstabe}} \times \underbrace{\{-1, 0, 1\}}_{\text{backwards, nothing, forwards}}$$

  - Berechnung von Φ besteht aus rekursiv definierten Tupeln (n^i, s^i, π^i), wobei
    - i = diskrete Zeit, n^i = i-ter Befehl, $s^i_{\pi^j} = \pi^j$-te Stelle im String zur Zeit i, π^i = Position des Lese-Schreib-Kopfes
    - starte mit $(n^0, s^0, \pi^0) = (0, x, 1)$

  * $\Phi(n^i, s^i_{\pi^i}) = (n^{i+1}, s^{i+1}_{\pi(i)}, \pi^{i+1} - \pi^i)$

- **Berechungsproblem** = ein Paar (X, R), wobei zu jedem $x \in X$ ein $y \in A^*$ mit $(x, y) \in R$ existiert
  - Turingmaschine Φ berechnet (X, R) falls $\text{Zeit}(\Phi, x) < \infty$ und $(x,\text{Output}(\Phi, x)) \in R$
  - Turingmaschine Φ berechnet f falls es genau eine Lösung y gibt. Dann ist $(x, f(x)) \in R$
  - Turingmaschine Φ entscheidet Sprache L wenn $X = A^*$ und $f(x) \in \{0, 1\}$ ist.
  - polynomielle Turingmaschine Φ wenn ein Polynom p mit $\text{Zeit}(\Phi, x) < p(x)$ existiert
- **Entscheidungsproblem** = ein Paar (X, Y), wobei $Y \subseteq X$.
- Entscheidungsprobleme $\subseteq$ Berechnungsprobleme
  - Entscheidungsproblem $\mathcal{P}$ ist Berechnungsproblem $\mathcal{P}' = (X, \{(x, 1) \mid x \in Y\} \cup \{(x, 0) \mid x \in X \setminus Y\})$
  - ein Algorithmus berechnet eine Funktion f mit $f(x) = 1$ falls $x \in Y$ und $f(x) = 0$ falls $x \in X \setminus Y$.
- **2-Band-Turingmaschine** ist eine Funktion Φ

$$\Phi : \underbrace{\{0, \ldots, N\}}_{\text{Befehl}} \times \underbrace{\overline{A}^2}_{\text{Buchstabe}} \longrightarrow \underbrace{\{-1, \ldots, N\}}_{\text{Befehl}} \times \underbrace{\overline{A}^2}_{\text{Buchstabe}} \times \underbrace{\{-1, 0, 1\}^2}_{\text{backwards, nothing, forwards}}$$

  - Berechnung von Φ besteht aus rekursiv definierten Tupeln $(n^i, s^i, t^i, \pi^i, \rho^i)$, wobei
    * i = diskrete Zeit, n^i = i-ter Befehl,
    * starte mit

      $$(n^0, s^0, t^0, \pi^0, \rho^0) = (0, \{\ldots, \sqcup, x_1, \ldots, x_{\text{size}(x)}, \sqcup, \ldots\}, \{\ldots \sqcup \sqcup \sqcup \ldots\}, 1, 1)$$

    * $\Phi(n^i, s^i_{\pi^i}, t^i_{\rho^i}) = (n^{i+1}, s^{i+1}_{\pi^i}, t^{i+1}_{\rho^i}, \pi^{i+1} - \pi^i, \rho^{i+1} - \rho^i)$
- 2-Band-Turingmaschine Φ kann durch 1-Band-Turingmaschine Φ' in $O(\text{Zeit}(\Phi, x)^2)$ simuliert werden
- **Orakel-Turingmaschine** ist eine Funktion Φ

$$\Phi : \underbrace{\{0, \ldots, N\}}_{\text{Befehl}} \times \underbrace{\overline{A}^2}_{\text{Buchstabe}} \longrightarrow \underbrace{\{-2, \ldots, N\}}_{\text{Befehl}} \times \underbrace{\overline{A}^2}_{\text{Buchstabe}} \times \underbrace{\{-1, 0, 1\}^2}_{\text{backwards, nothing, forwards}}$$

  - zusätzlicher Befehl -2, nach Ausführung andere Zeitbemessung $t^{i+1} = t^i + 1 + size(y)$

  - auf zweitem Band steht eine Instanz von (X, R) und wir überschreiben es mit y aus $(x, y) \in R$
  - terminiert *nicht*, springt auf den nächsten Zustand in der Liste
- Beispiele: LINEARE UNGLEICHUNGEN, GANZZAHLIGE LINEARE UNGLEICHUNGEN

3.14 P und NP

- P = Klasse aller **Entscheidungsprobleme**, für die es einen polynomiellen Algorithmus gibt
- NP = Klasse aller **Entscheidungsprobleme**, für die ein Zertifikat c in einer polynomiellen Zeit überprüft werden kann
- $coNP$ = Klasse aller **Entscheidungsprobleme**, deren komplementäre Entscheidungsprobleme in NP sind
  - komplementäres Entscheidungsproblem = $(X, X \setminus Y)$
- $P \subseteq NP$
- **randomisierter Algorithmus**
  - Las-Vegas-Algorithmus = immer richtig
  - Monte-Carlo-Algorithmus = nicht immer richtig
  - mit einseitigem Fehler = für jedes $s \in S$ mit $f(s) = 0$ gilt immer für beliebige r : $g(s\#r) = 0$
  - nichtdeterministischer Algortihmus = mit einseitigem Fehler + für jedes $s \in S$ mit $f(s) = 1$ gibt es immer mindestens ein r mit $g(s\#r) = 1$
    * für jede Nein-Instanz habe nein, für jede Ja-Instanz gibt es eine Chance auf ja.
- polynomielle Reduktion (= Berechnungsprobleme)
- polynomielle Transformation (= Entscheidungsprobleme)
  - jede Transformation ist eine Reduktion
- NP**-vollständig** = Entscheidungsproblem $\mathcal{P}$ + jedes Entscheidungsproblem in NP lässt sich polynomiell in $\mathcal{P}$ transformieren
- NP**-schwer** = Berechnungsproblem $\mathcal{P}$ + jedes Entscheidungsproblem in NP lässt sich polynomiell auf $\mathcal{P}$ reduzieren
- NP-vollständig $\subseteq$ NP-schwer

3.15 Der Satz von Cook

- SATISFIABILITY = gegeben eine Menge X von Variablen und eine Familie Z von Klauseln über X entscheide, ob Z erfüllbar ist.
  - boolesche Variablen X, Wahrheitsbelegung $T : X \to \{1, 0\}$, Klausel $K(T) = \vee_i L_i(T)$, Familie von Klauseln $Z(T) = \bigwedge_j K_j(T)$
  - erfüllbar wenn T existiert mit $Z(T)$ wahr
- SATISFIABILITY ist NP-vollständig
  - SATISFIABILITY ist in NP (gegeben a überprüfe ob $Z(a)$ wahr)
  - $O(Q^2)$ Variablen udn $O(Q^3)$ Klauseln

XX. Vorlesung, 11.01.24

- von Entscheidungsproblemen zu Berechnungsproblemen
  - für Entscheidungsprobleme reicht polynomielle Transformation zwischen zwei Problemen, in dem man Ja-Instanzen auf Ja-Instanzen abbildet und Nein-Instanzen auf Nein-Instanzen
  - für Berechnungsproblemen braucht man Orakel-Turingmaschinen (siehe oben)
  - Berechnungsproblem $\mathcal{P}_1$ auf $\mathcal{P}_2$ **polynomiell reduzierbar** wenn es ein $\mathcal{P}_2$ benutzender polynomieller Orakel-Algorithmus existiert.
- wenn $\mathcal{P}_2$ polynomiell + $\mathcal{P}_1$ polynomiell auf $\mathcal{P}_2$ reduzierbar, dann gibt es einen polynomiellen Algorithmus für $\mathcal{P}_1$
- ein Berechnungsproblem $\mathcal{P}$ ist NP-schwer, wenn jedes Problem aus NP polynomiell auf $\mathcal{P}$ reduzierbar ist
  - falls es einen polynomiellen Algorithmus für ein NP-schweres Problem gibt, dann gilt $P = NP$.
  - NP-vollständige Probleme sind NP-schwer
- 3SAT ist NP-vollständig
  - $3\text{SAT} \in NP$
  - jede Klausel hat **genau** drei Literale
  - reduzieren SAT auf 3SAT
    - Fall 1: Klausel, die mehr als drei Literale haben $\lambda_1 \dots \lambda_k$ mit $k > 3$. Ersetze durch ... mit $(k-3)$ neuen Variablen $y_1 \dots y_{k-3}$
    - Fall 2: Klausel mit genau zwei Literalen λ_1, λ_2. Ersetze durch ...
    - Fall 3: Klausel mit genau einem Literal λ_1. Ersetze durch ...
- 2SAT ist polynomiell lösbar

3.16 Beispiele von NP-vollständigen Problemen

- STABLE-SET
  - hat G eine stabile Menge der Größe k? (k paarweise nicht benachbarte Knoten)
  - ist in NP, Zertifikat = stabile Menge der Größe k
  - zu zeigen: SAT ist polynomiell auf STABLE-SET transformierbar
    - sei Z eine Familie von Klauseln $Z_1, \ldots, Z_n$. Konstruiere Graph G, sodass G enthält stabile Menge der Größe $m \iff$ Z erfüllbar ist.
    - für jede Klausel Z_i enthält G eine Clique mit k_i Knoten, Knoten zwischen verschiedenen Cliquen sind verbunden nur wenn sie sich widersprechen
    - eine stabile Menge $X \subseteq G$ ist dann nach Konstruktion eine Wahrheitsbelegung von SAT
- VERTEX-COVER
  - hat G eine Knotenüberdeckung mit k Knoten?
- CLIQUE
  - hat G eine Clique mit k Knoten?
- HAMILTON-KREIS
  - hat G einen Kreis
- SUBSET-SUM
- PARTITION

3.17 Die coNP Klasse

3.18 NP-schwere Probleme