Formelsammlung AGT

Inhalt

[1. Grundlagen 1](#_Toc30244701)

[Bipartite Planare Graphen: 1](#_Toc30244702)

[Handshakinglemma: 2](#_Toc30244703)

[Eulersche Polyederformel: 2](#_Toc30244704)

[Topologische Sortierung: 2](#_Toc30244705)

[2. Suche in Graphen 2](#_Toc30244706)

[Breitensuche: 2](#_Toc30244707)

[Tiefensuche: 2](#_Toc30244708)

[3. Minimal aufspannende Bäume 3](#_Toc30244709)

[Prim: 3](#_Toc30244710)

[Kruskal: 4](#_Toc30244711)

[4. Kürzeste Wege 4](#_Toc30244712)

[Bellman Ford: 4](#_Toc30244713)

[Rucksack Problem dynamische Programmierung: 4](#_Toc30244714)

[5. Flüsse in Netzwerken 5](#_Toc30244716)

[Edmons und Karp: 5](#_Toc30244717)

[Push und Relabel: 5](#_Toc30244718)

[6. Matchings 5](#_Toc30244719)

[Blossom Shrinking: 5](#_Toc30244720)

[7. Eulerkreise, Hamiltonkreise und CPP/TSP 5](#_Toc30244721)

[Chinesisches Postboten-Problem (CPP) 5](#_Toc30244722)

[8. Färbung von Graphen 5](#_Toc30244723)

[DSatur: 5](#_Toc30244724)

[RecursiveLargestFirst (RLF): 5](#_Toc30244725)

## Grundlagen

### Bipartite Planare Graphen:

Bei Größeren Graphen einen Knoten und seine Kanten entfernen. Danach Knoten, welche in der Mitte einer Verbindung sind entfernen. Er hält man einen K3,3-Graphen ist man Fertig und kann mit Formel beweisen:

n = Knoten

m = Kanten

Es muss für bipartite planare Graphen gelten.

Handshakinglemma:

### Eulersche Polyederformel:

Sei ein G ein zusammenhängender planarer Graph und seien n, m, f die Anzahl der Knoten, Kanten bzw. Flächen von G. Dann gilt

Topologische Sortierung:   
Graphen welche keinen Kreis beinhalten sind topologisch sortierbar.

Überprüfen auf Kreislauf.

1. Hat kein Knoten Eingangsgrad 0. Fertig Graph ist nicht Sortierbar
2. Entferne immer wieder Knoten mit Grad 0 bis jeder Knoten in |V| aufgebraucht ist. Sind noch Knoten mit eingangsgrad Größer 0 vorhanden => Kreis

Oder

1. Tiefensuche
2. Sind Rückwertskanten vorhanden existiert ein Kreis

## Suche in Graphen

### Breitensuche:

Parameter:

* l(v) = level im Baum
* p(v) = Parent Knoten
* t(v) = Zeit zu der Ein Knoten besucht wurde.

Bsp. :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| v | A | B | C | Queue |
| l(v) | 0 |  |  | (A) |
| p(v) | - |  |  |
| t(v) | 1 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| l(v) | 0 | 1 |  | (B,C) |
| p(v) | - |  |  |
| t(v) | 1 | 2 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| l(v) | 0 | 1 | 1 | (C) |
| p(v) | - | A1 | A1 |
| t(v) | 1 | 2 | 3 |

### Tiefensuche:

Parameter:

f(v) und l(v) teilen sich eine Laufvariable, f(v) und l(v) für einen Knoten geben an in welchem Bereich sich ein Knoten befindet [ f(v), l(v) ].

* f(v) = zählt hoch, wenn Knoten auf den stack kommt
* l(v) = zählt hoch, wenn knoten vom stack runter kommt
* p(v) = Parent Knoten

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| v | A | B | C | D | E | Stack |
| f(v) | 1 |  |  |  |  | (A) |
| l(v) |  |  |  |  |  |
| p(v) | - |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 |  |  |  | (B,A) |
| l(v) |  |  |  |  |  |
| p(v) | - |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 | 3 |  |  | (C,B,A) |
| l(v) |  |  |  |  |  |
| p(v) | - | A | B |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 | 3 | 5 |  | (D,A) |
| l(v) |  |  | 4 |  |  |
| p(v) | - | A | B |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 | 3 | 5 |  | (A) |
| l(v) |  | 7 | 4 | 6 |  |
| p(v) | - | A | B |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 | 3 | 5 | 8 | (E,A) |
| l(v) |  | 7 | 4 | 6 |  |
| p(v) | - | A | B |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 | 3 | 5 | 8 | (A) |
| l(v) |  | 7 | 4 | 6 | 9 |
| p(v) | - | A | B | B | A |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(v) | 1 | 2 | 3 | 5 | 8 |  |
| l(v) | 10 | 7 | 4 | 6 | 9 |
| p(v) | - | A | B | B | A |

Kantenarten beim Digraphen im Tiefensuchbaum:

* Querbogen
* Vorwärtsbogen
* Rückwertsbogen

## Minimal aufspannende Bäume

### Prim:

1. Nehme Startknoten in die Menge S auf
2. Wähle Knoten, welcher den Kleinsten Abstand zur Menge S hat
3. Nehme den Knoten in die Menge S auf
4. Wiederhole solange bis alle Knoten in der Menge S vorhanden sind

### Kruskal:

1. Wähle Kante mit niedrigster Kapazität
2. Wähle die nächste Kante mit niedrigster Kapazität mit der Bedingung, dass kein Kreis entsteht.

## Kürzeste Wege

### Bellman Ford:

Ist ein Algorithmus für den Kürzesten Weg von s = source zu allen anderen Knoten in G. In diesem Algorithmus sind negative Kanten (aber keine negativen Kreise) erlaubt.

Die Tabelle beinhaltet d(v) = distanz zum Startknoten und p(v) seinen Parentknoten, welcher ihn zuletzt aktualisiert hat.

Vorgehen:

1. durchlaufe im Graph jede kante ausgehend von s und aktualisier dabei die Distanzen der Knoten bis alle Kanten aufgebraucht sind.
2. Wiederhole Schritte solange bis
   1. Sich nichts mehr verändert (Fertig)
   2. Es selbst in einer Höheren Iterationsstufe noch etwas verändert (dann existiert ein negativer Kreis)

### Rucksack Problem dynamische Programmierung:

Ist eigentlich das Binäre Rucksack Problem. Man hat 2 Polynomial Funktionen.

f(x) = der wert meiner „ware“ (das was es zu maximieren gilt)

die andere Funktion gibt das „gewicht der waren“ an und ist begrenzt. Im Folgenden gilt es die beiden Funktionen gemeinsam zu optimieren. Funktioniert nach kürzeste Wege verfahren.

Hierfür wird eine Matrix erstellt in der x Koordinate nach rechts d(eigentlich die Distanz) von 0 bis dem maximal „Gewicht“. In der y Koordinate befindet sich nach unten k von 1 bis Anzahl der Polynome.

Erstellt wird die Matrix wie Folgt:

1. Beginne mit der Zeile k = 1 und Trage ab d von x1 (= den Multiplikator von x1 in der Funktion vom „Gewicht“) in die Ganze reihe den Multiplikator von x1 von f(x) ein.
2. Starte bei der nächsten Zeile. Bspw. K = 2. Summiere auf jede spalte d, dass dx2 von x2 und vergleiche in der Spalte dsum, ob der Inhalt der Zelle dxk1 + multiplikator von x2 in f(x) größer ist als der aktuelle Inhalt in der Zelle dsumxk1. Ist es größer wird die Zelle aktualisiert.

## Flüsse in Netzwerken

### Edmons und Karp:

1. Baue den Hilfsgraphen auf
2. Nutze die Breitensuche(lexikographisch) und erstelle die Breitensuchtabelle
3. Gehe in der Tabelle ausgehend von t bis zu s zurück
4. Markiere die Änderungen im Graphen
5. Starte die Breitensuche erneut
6. Max. n/2 Iterationen

Falls der Maximale Fluss gesucht wird.

1. Mache einen schnitt im Graphen (bspw. an t)
2. Max. Fluss = Anzahl des Flusses im Schnitt

### Push und Relabel:

1. Initialisiere den Graphen mit d(s) = n = |V| und für alle anderen Knoten d(v) = 0. d(t) = immer 0. Die Maximalen Flüsse einer Kante sind vorgegeben. Der Excess eines Knotens ist zu Beginn 0.
2. Die Folgeknoten von s bekommen den Maximalen Fluss der Kante. Es verändert sich der Excess der Folge Knoten auf den jeweiligen Wert des Flusses.
3. Der lexikographisch kleinste Folgeknoten von s bekommt ein relabel und erhöht sein d(v) = 1. Das Maximum des Flusses wird an die nachbarknotenweiter gegeben.
4. Hat dieser knoten einen Excess > 0 wird der Knoten erneut relabeled auf 1 höher als der Startknoten s [d(v) = d(s) +1] gesetzt.
5. Im Folgenden wird lexikographisch weiter gemacht.
6. Kann an einem Knoten nicht der Gesamte Excess gesendet werden muss er zum Startknoten zurück.
7. Fertig wenn alle Knoten keinen Excess besitzen.

## Matchings

### Blossom Shrinking:

1. Suche den Kleinsten Knoten, welcher nicht Teil eines Matchings ist.
2. Bilde von dort den alternierenden Baum.
3. Wende dafür von dem Punkt eine art “Breitensuche“ an. Nehme den nächsten Nachbarn und überprüfe:
   1. Der Nachbarknoten ist Teil eines Matchings. Hänge im Baum das matching an und wiederhole schritt 3.
   2. Der Nachbarknoten ist Teil eines Matchings und besitzt eine Kante mit der ein Kreis im Baum entsteht => Blüte im Graphen gefunden. Reduziere die Blüte im Graphen und gib dieser Blüte eine neue Knoten Zahl. Merke dir dabei welche Verbindungen aus der Blüte rausgehen.
   3. Der Nachbarknoten ist kein Teil eines Matchings => Alternierenderpfad wurde gefunden. Auf diesem Pfad werden die Kanten getauscht. Doppelkanten werden zu Einzelkanten und anders herum.
4. Wiederhole 1. Bis es nicht mehr möglich ist.

## Eulerkreise, Hamiltonkreise und CPP/TSP

### Chinesisches Postboten-Problem (CPP):

1. Finde Knoten mit ungeradem Grad
2. Verbinde Knoten paarweise miteinander
3. Die Summe der Paare muss Kostenminimal sein. Durch ausprobieren.
4. Füge für die Verbindung der Paare eine weitere Kante ein => Kontengrade gerade.
5. Finde nun den Euler Kreis durch sukzessives Aufspüren von Kreisen:
   1. Fang bei einem Knoten an und finde einen Eulerkreis. Nehme die Kanten aus dem Graphen raus und gib dem Kreis einen Namen bspw. Cx. Notiere den pfad für später. (Es empfiehlt sich möglichst lange Eulerkreise zu nehmen. Erreicht wird das in dem man zum Beispiel außen am Graph entlang geht.).
   2. Wiederhole bis keine kannten mehr frei sind.
6. Nun besitzt man eine Anzahl an Euler Kreisen. Diese müssen jetzt zusammengefügt werden.
7. Beginne an einem Start knoten eines Euler Kreises. Wird auf dem Pfad ein startknoten eines Anderen Eulerkreises berührt. Füge den anderen Eulerkreis in den jetzigen mit ein. Dabei ist es möglich im anderen Kreislauf wieder einen Startknoten zu Berühren.

#### Minimale-Spannbaum-Heuristik:

1. Bestimme einen Minimalen Spannbaum (Prim oder Kruskal)
2. Verdopple die Kanten des Minimalen Spannbaums
3. Bilde eine Euler Tour
4. Laufe die Euler Tour ab so das Kein Knoten doppelt benutzt wird. Das bedeutet das man abkürzt Sobald man einen Knoten, welchen man bereits besucht hat, nochmal besuchen würde.

#### Christofides-Heuristik:

1. Bestimme Minimalen Spannbaum (Prim oder Kruskal)
2. Bestimme die Menge der Knoten v mit ungeraden Grad
3. Bilde ein Kosten minimales Perfektes Matching. Durch ausprobieren.
4. Verdopple die kanten auf dem Pfad der Knoten
5. Bilde eine Euler Tour
6. Laufe die Euler Tour ab so das Kein Knoten doppelt benutzt wird. Das bedeutet das man abkürzt Sobald man einen Knoten, welchen man bereits besucht hat, nochmal besuchen würde.

## Färbung von Graphen

### DSatur:

### RecursiveLargestFirst (RLF):