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:

### Push und Relabel:

## Matchings

### Blossom Shrinking:

## Eulerkreise, Hamiltonkreise und CPP/TSP

### Chinesisches Postboten-Problem (CPP)

#### Minimale-Spannbaum-Heuristik:

#### Christofides-Heuristik:

## Färbung von Graphen

### DSatur:

### RecursiveLargestFirst (RLF):