Grundzüge der Informatik 1

Vorlesung 28

Überblick gesamte Vorlesung

- Grundlagen
- Teile&Herrsche Verfahren
- Dynamische Programmierung
- Gierige Algorithmen
- Datenstrukturen
- Graphalgorithmen

Überblick Vorlesung

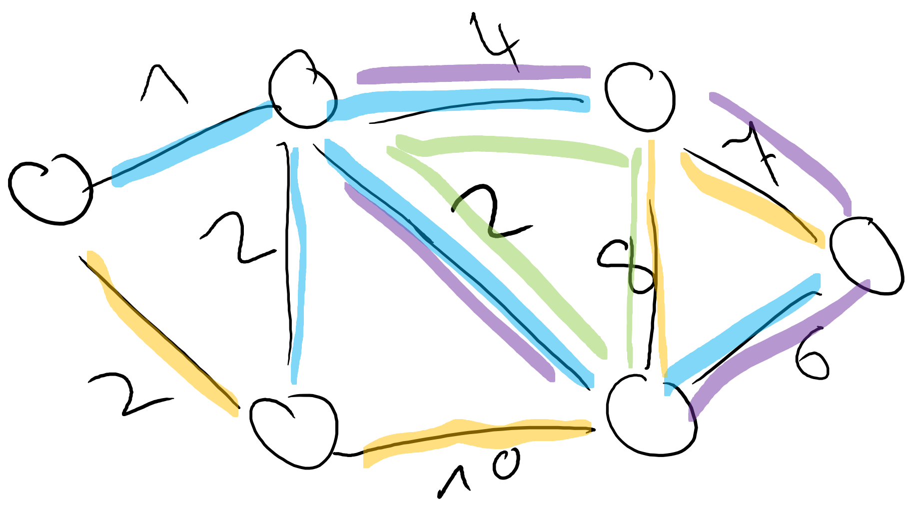
Graphenalgorithmen

- Minimale Spannbäume
  - Definition
  - Die Kreiseigenschaft
  - Algorithmus von Kruskal
  - Die Union-Find Datenstruktur
  - Algorithmus von Prim

Graphenalgorithmen

Minimale Spannbäume

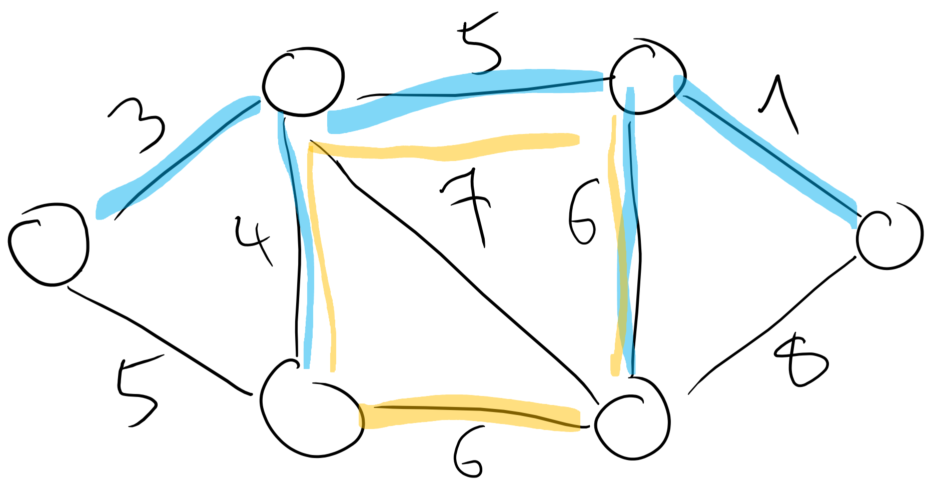
- Gegeben: Gewichteter, ungerichteter, zusammenhängender Graph G=(V,E)
- Gesucht: Ein aufspannender Baum mit minimalem Gewicht
  (Summe der Kantengewichte des Baums)
- Aufspannender Baum: Baum mit Knotenmenge V und Kanten aus E



Graphenalgorithmen

Satz 23.4

- Sei G=(V,E) ein ungerichteter, gewichteter, zusammenhängender Graph. Sei $(v_0,..,v_k)$ ein einfacher Kreis in G und sei e eine Kante des Kreises mit maximalem Gewicht. Dann ist der Graph G=(V,E\{e}) zusammenhängend und der minimale Spannbaum hat dasselbe Gewicht wie der minimale Spannbaum von G.



Graphenalgorithmen

MST-1(G)

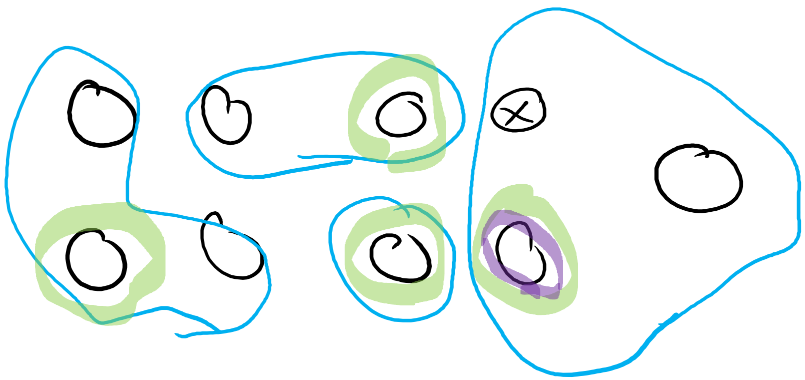
1. T=G
2. **while** T ist kein Baum **do**
3. Finde Kreis in T
4. Entferne die Kante mit maximalem Gewicht aus dem Kreis
5. **return** T

Graphenalgorithmen

Kruskal(G)

1. A =$\emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** (u,v)$\in$E geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in derselben Zusammenhangskomponente in Graph H=(V,A) **then**
5. A = A$\cup${(u,v)}
6. **return** A

Graphenalgorithmen



Union-Find Datenstrukturen

- Grundmenge V
- Speichert Partition $\mathcal{S}$ ={S_1,…, S_k} von V
  (Aufteilung in disjunkte Mengen, deren Vereinigung V ergibt)
- Für jede Menge gibt es einen Repräsentanten
- Make-Set(x): Erzeuge neue Menge, die nur x enthält und fügt x in V ein
- Union(x,y): Vereinigung der Mengen, die x bzw. enthalten
- Find(x): Gibt Referenz auf den Repräsentanten der Menge, die x enthält

Graphenalgorithmen

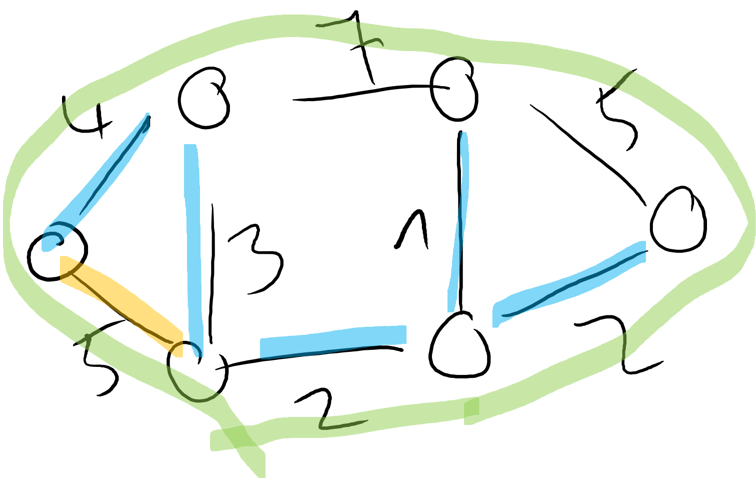
Idee

- Im Algorithmus von Kruskal entsprechen die disjunkten Mengen der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die Kanten aus A erzeugten Graphen

Graphenalgorithmen

Kruskal(G)

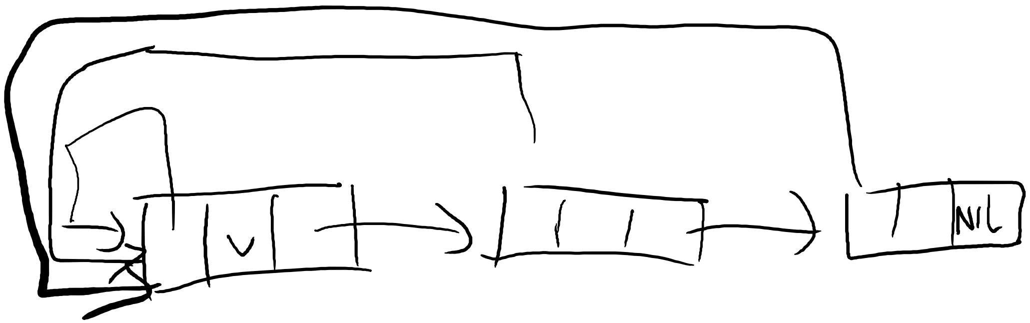
1. A = ∅
2. **for each** vertex v∈V **do** Make-Set(v)
3. Sortiere Kanten nach Gewicht
4. **for each** (u,v)∈E geordnet nach aufsteigendem Gewicht **do**
5.   **if** Find(u)≠Find(v) **then**
6.   A = A∪{(u,v)}
7.   Union(u,v)
8. **return** A



Graphenalgorithmen

Eine einfache Union-Find Datenstruktur

- Jede Menge ist Liste
- Erstes Element ist Repräsentant
- Jedes Listenelement enthält Zeiger auf den Repräsentanten

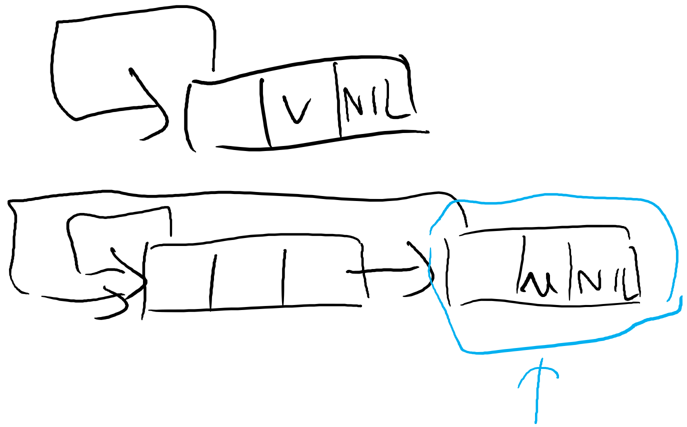


Graphenalgorithmen

Aufgabe

- Wie würden Sie die Operationen makeset, union und find für unsere Datenstruktur realisieren?

Graphenalgorithmen



Implementierung

- Make-Set in O(1) Zeit einfach
- Find in O(1) Zeit einfach
- Union: Hänge die eine Liste hinter die andere und aktualisiere alle Zeiger

Laufzeit

- Betrachte Sequenz von m Operationen aus Make-Set, Find, und Union
- Laufzeit für Union O(m) (da wir höchstens m Elemente haben)

Graphenalgorithmen

Beobachtung

- Wir hängen evtl. immer eine sehr lange Liste an eine sehr kurze
- Wenn wir immer die kurze hinter die lange hängen, müssen wir nur die Referenzen in der kurzen Liste aktualisieren
- Aber bringt das etwas (mehr als Konstanten)?

Graphenalgorithmen

Satz 24.1

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, O(m+ n log n) Zeit.

Beweis

- Wir analysieren zunächst, wie oft der Repräsentantenzeiger eines Elements einer Menge der Größe k maximal aktualisiert wurde
- Betrachte Element x
- Jedes mal, wenn der Repräsentantenzeiger von x aktualisiert wurde, war x in der kleineren der vereinigten Mengen
- Damit hat sich die Größe der Menge mindestens verdoppelt

Graphenalgorithmen

Satz 24.1

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $O(m+ n \log n)$ Zeit.

Beweis

- Damit gilt für jedes $k \le n$, dass nach $\lceil \log k \rceil$ Aktualisierungen des Repräsentantenzeigers von x, die Menge, die x enthält, mindestens k Elemente besitzt
- Da die größte Menge maximal n Elemente besitzt, wurde jeder Repräsentantenzeiger maximal $O(\log n)$ mal aktualisiert (über alle Union-Operationen)
- Damit ist die Gesamtlaufzeit für die Aktualisierungen der n Objekte $O(n \log n)$

Graphenalgorithmen

Satz 24.1

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $O(m+ n \log n)$ Zeit.

Beweis

- Jedes MakeSet und Find benötigt $O(1)$ Zeit und es gibt $O(m)$ davon
- Damit ist die gesamte Laufzeit für die Sequenz $O(m+n \log n)$

Graphenalgorithmen

Satz 24.2

Der Algorithmus von Kruskal berechnet in O(|E| log |E|) Zeit einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen G=(V,E).

Beweis

- Sei A die Menge der Kanten am Ende von Kruskals Algorithmus
- Jede Kante, die nicht durch den Algorithmus von Kruskal ausgewählt wird, schließt mit den Kanten aus A einen Kreis und ist eine maximale Kante in diesem Kreis
- Wir können also mit E beginnen und alle Kanten aus E\A schrittweise entfernen, indem wir in jedem Schritt eine maximale Kante aus einem Kreis entfernen
- Durch iteratives Anwenden von Satz 23.4 folgt, dass der Graph H(V,A) ein minimaler Spannbaum ist

Graphenalgorithmen

Satz 24.2

Der Algorithmus von Kruskal berechnet in $O(|E| \log |E|)$ Zeit einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

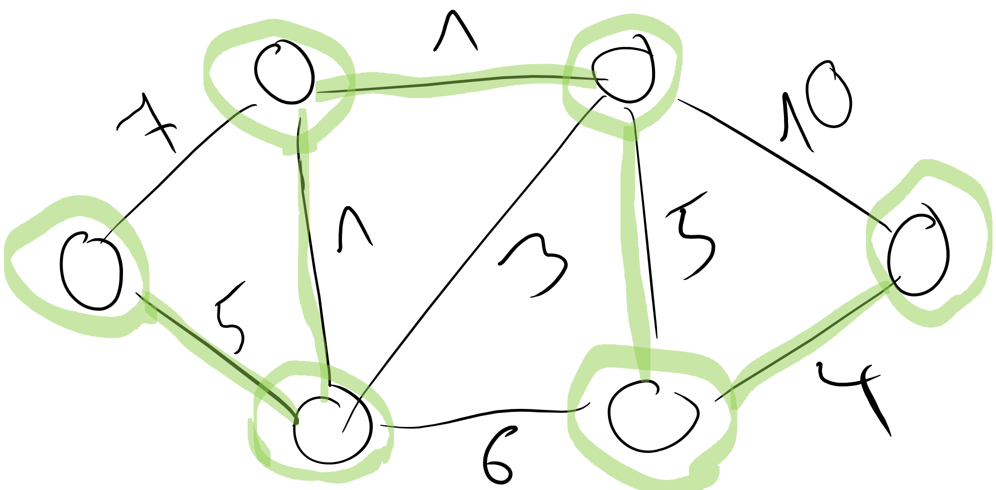
Beweis

- Laufzeit: Der Algorithmus benötigt $O(|E| \log |E|)$ Zeit zum Sortieren
- Er führt $O(|E|+ |V|)$ Operationen mit der Union-Find-Datenstruktur durch
- Davon sind $|V|$ Operationen MakeSet
- Somit ist die Laufzeit für die Operationen der Union-Find-Datenstruktur $O(|E|+|V| \log |V|)$
- Diese dominieren die Laufzeit der zweiten for-Schleife
- Insgesamt ist daher die Laufzeit $O(|E| \log |E|)$
(da $|V| = O(|E|)$ für zusammenhängende Graphen)

Graphenalgorithmen

Idee des Algorithmus von Prim

- Algorithmus lässt einen Baum T von einem Knoten aus wachsen
- In jeder Runde: Nimm immer eine Kante mit minimalem Gewicht, die einen Knoten in Baum T mit einem Knoten verbindet, der nicht in Baum T ist und füge diese zu T hinzu
- Die Kantenmenge von T bezeichnen wir mit A



Graphenalgorithmen

Prim(G,r)

1. $Q = V \setminus \{r\}$
2. $A = \emptyset$
3. **while** $Q \neq \emptyset$ **do**
4.   Finde Kante (u,v) mit minimalem Gewicht, wobei $u \in Q$ und $v \in V \setminus Q$ ist
5. $Q = Q \setminus \{u\}$
6. $A = A \cup \{(u,v)\}$
7. **return** A

Graphenalgorithmen

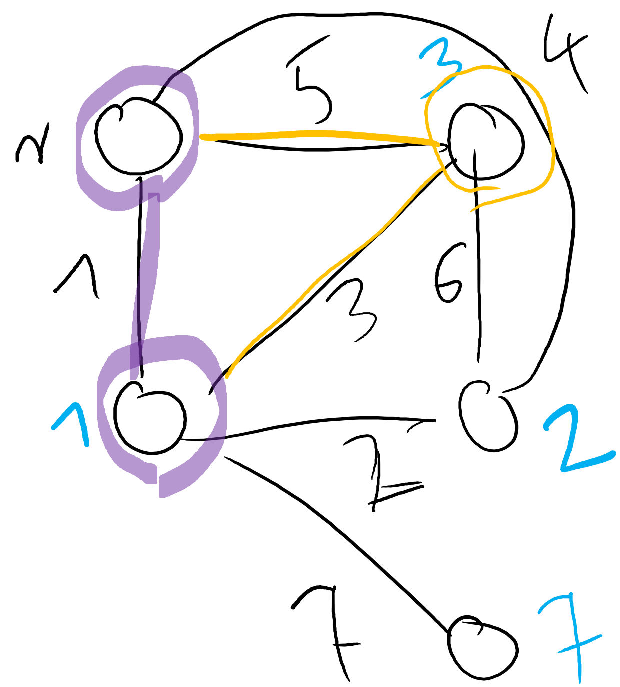
Prioritätenschlange

- Alle Knoten, die noch nicht zum Baum gehören, werden in Prioritätenschlange Q abgespeichert
- key[v]: minimales Gewicht einer Kante, die v mit Baum verbindet
- parent[v]: Vorgänger von v im Baum
- Menge A implizit gegeben durch $A = \{(v, parent[v]) \mid v \in V \setminus \{r\} \setminus Q\}$

Graphenalgorithmen

Prim(G,r)

1. Q = V
2. **For each** vertex u∈Q **do** key[u] = ∞
3. key[r] = 0, parent[r] = NIL
4. **while** Q ≠∅ **do**
5. u = Extract-Min(Q)
6. **For each** v∈Adj[u] **do**
7. **If** v∈Q **and** w(u,v)<key[v] **then**
8. key[v] = w(u,v), parent[v]=u



Graphenalgorithmen

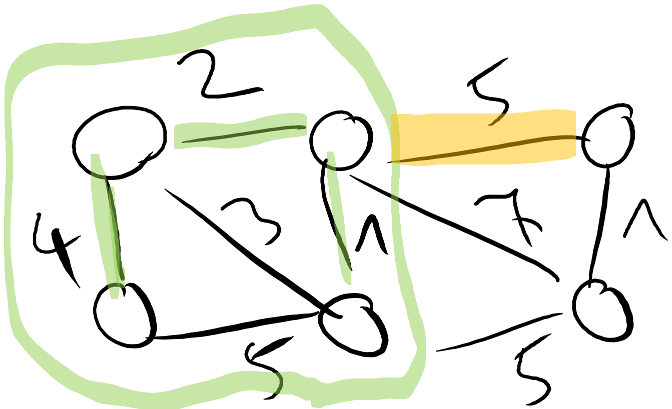
Implementierung

- Q als Rot-Schwarz-Baum

Laufzeit

- Initialisierung von Q: O(|V|)
- Ausführung der **while**-Schleife: O(|V|)-mal
- Extract-Min: O(log |V|)
- Länge aller Adjazenzlisten: O(|E|)
- Test v∈Q: O(1)
- Änderung eines Schlüsselwertes: O(log |V|)
- Gesamtlaufzeit: O(|V| log |V| + |E| log |V|) = O(|E| log |V|)

Graphenalgorithmen



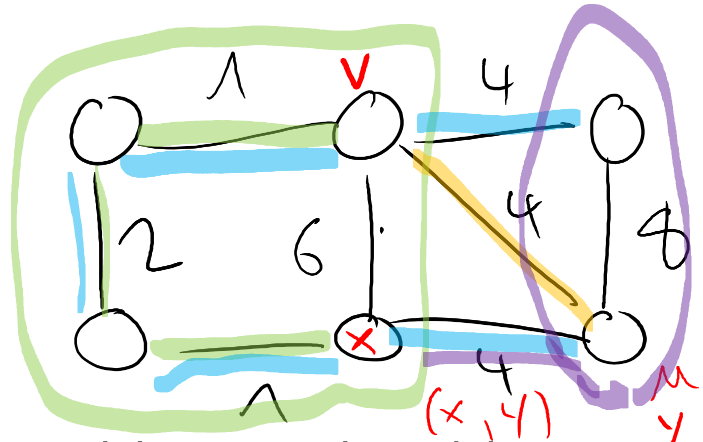
Satz 24.3

Sei G=(V,E) ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G. Sei Q eine Knotenmenge, so dass keine Kante aus A einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Sei (u,v) eine Kante mit minimalem Gewicht, die einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Dann ist $A \cup \{(u,v)\}$ Teilmenge eines minimalen Spannbaums von G.

Beweis

- Für einen Baum T bezeichne $w(T) = \Sigma_{e \in T} w(e)$ sein Gewicht
- Sei T min. Spannbaum, der A enthält
- Annahme: Seien Q und (u,v) wie im Satz
- Wir konstruieren min. Spannbaum T‘, der A und (u,v) enthält
- Wenn (u,v) in T ist, so sind wir fertig

Graphenalgorithmen



Satz 24.3

Sei G=(V,E) ein zusammenhängender, ungerichteter und gewichteter Graph. Sei A⊆E Teilmenge eines minimalen Spannbaums von G. Sei Q eine Knotenmenge, so dass keine Kante aus A einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Sei (u,v) eine Kante mit minimalem Gewicht, die einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Dann ist A ∪ {(u,v)} Teilmenge eines minimalen Spannbaums von G.

Beweis

- Ansonsten: Kante (u,v) bildet Kreis mit Weg p von u nach v in T
- Da ein Knoten von (u,v) in Q liegt und der andere in V\Q, gibt es noch mindestens eine weitere Kante aus p, die Q und V\Q verbindet
- Sei (x,y) eine solche Kante
- (x,y) ist nicht in A nach unserer Annahme

Graphenalgorithmen

Satz 24.3

Sei $G=(V,E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G. Sei Q eine Knotenmenge, so dass keine Kante aus A einen Knoten aus Q mit einem Knoten aus $V \setminus Q$ verbindet. Sei (u,v) eine Kante mit minimalem Gewicht, die einen Knoten aus Q mit einem Knoten aus $V \setminus Q$ verbindet. Dann ist $A \cup \{(u,v)\}$ Teilmenge eines minimalen Spannbaums von G.

Beweis

- Da (x,y) auf dem eindeutig bestimmten Weg von u nach v in T ist, wird T durch Entfernen von (x,y) in zwei Komponenten aufgeteilt.
- Hinzunahme von (u,v) verbindet diese Komponenten wieder (da Weg p und (u,v) einen Kreis bilden)
- Definiere: $T' = T - \{(x,y)\} \cup \{(u,v)\}$
- Wir zeigen, dass T' min. Spannbaum ist

Graphenalgorithmen

Satz 24.3

Sei G=(V,E) ein zusammenhängender, ungerichteter und gewichteter Graph. Sei A⊆E Teilmenge eines minimalen Spannbaums von G. Sei Q eine Knotenmenge, so dass keine Kante aus A einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Sei (u,v) eine Kante mit minimalem Gewicht, die einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Dann ist A ∪ {(u,v)} Teilmenge eines minimalen Spannbaums von G.

Beweis

- T‘ ist ein Spannbaum, da T‘ nach Konstruktion kreisfrei ist und T‘ genauso viele Kanten wie der Spannbaum T hat
- Da (u,v) Q und V\Q verbindet und (x,y) ebenfalls, gilt w(u,v) ≤ w(x,y). Daher
- w(T‘) = w(T) – w(x,y) + w(u,v) ≤ w(T)
- T ist aber min. Spannbaum und somit gilt w(T) ≤ w(T‘)

Graphenalgorithmen

Satz 24.3

Sei G=(V,E) ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G. Sei Q eine Knotenmenge, so dass keine Kante aus A einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Sei (u,v) eine Kante mit minimalem Gewicht, die einen Knoten aus Q mit einem Knoten aus V\Q verbindet. Dann ist $A \cup \{(u,v)\}$ Teilmenge eines minimalen Spannbaums von G.

Beweis

- Daher muss T‘ ebenfalls min. Spannbaum
- Der Satz folgt nun direkt aus $A \subseteq T'$, da $A \subseteq T$, $(x,y) \notin A$ und $(u,v) \in T'$ und weil T‘ min. Spannbaum ist

Graphenalgorithmen

Satz 24.4

Der Algorithmus von Prim berechnet einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen in $O(|E| \log |V|)$ Zeit.

Beweis

- Die Laufzeit haben wir bereits analysiert.
- Die Korrektheit folgt aus Satz 24.3

Graphenalgorithmen

Zusammenfassung

- Minimale Spannbäume
  - Definition
  - Die Kreiseigenschaft
  - Algorithmus von Kruskal
  - Die Union-Find Datenstruktur
  - Algorithmus von Prim

Referenzen

- T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.