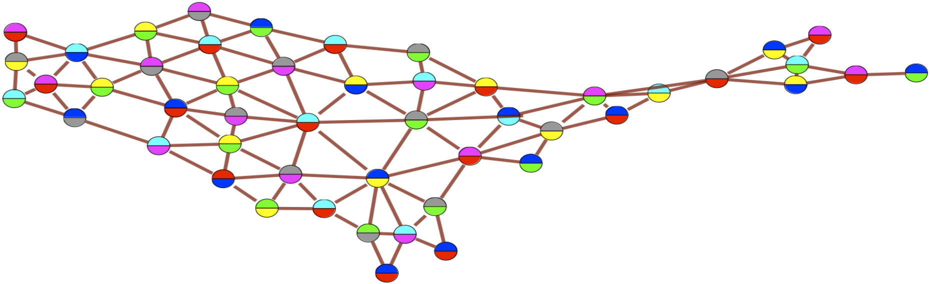
Karlsruher Institut für Technologie

Tutorium 11: Kürzeste Wege und MST

Holger Ebhart | 1. Juli 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

9. Übungsblatt

Aufgabe 2) b)

Die Idee ist eine modifizierte Tiefensuche zu verwenden:

```
procedure getSequence(a:Adjazenzlist):List of ℕ0
    s:DoublyLinkedList of ℕ0
    unmark all nodes in a
    foreach n:node in a do
        if n is unmarked then
            mark n
            DFS(n)

procedure backtrack(u,v:ℕ0)
    s.insertFront(v)
```

Laufzeit: $\mathcal{O}(|V| + |E|)$

Dijkstras Algorithmus

Wir suchen nun kürzeste Wege in einem Graphen.
Dijkstras Algorithmus liefert einen Baum kürzester Pfade von einem bestimmten Startknoten aus zu allen anderen Knoten des Graphen, sowie das Gewicht des jeweiligen Pfades.
Konvention: Knoten die vom Startknoten aus nicht erreichbar sind haben Distanz ∞ zu diesem.
Voraussetzungen:

- nur nichtnegative Kantengewichte
- kürzeste Pfade von einem Startknoten zu allen anderen

Dijkstras Algorithmus

```
procedure dijkstra(s:NodeId):Array of NodeId × Array of NodeId
    d = {∞, ··· , ∞};d[s]:=0
    parent[s]:=s
    Q.insert(s):PriorityQueue
    while Q ≠ ∅ do
        u:=Q.deleteMin
        foreach e = (u,v) ∈ E do
            if d[u]+c(e)¡d[v] then
                d[v] := d[u]+c(e)
                parent[v] := u
                if v ∈ Q then Q.decreaseKey(v)
                else Q.insert(v)
    return (d,parent)
```

Laufzeit:

$\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$

Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

Laufzeit:
$\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$

Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

Laufzeit:
$\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$

Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

Bellman-Ford Algorithmus

Warum reicht Dijkstra nicht aus?

- negative Kantengewichte
- negative Kreise $\rightarrow$ Es gibt Knoten die Distanz $-\infty$ zu anderen Knoten haben

Idee:
Ein kürzester Pfad hat maximal Länge $|V| - 1$; wenn wir also jede Kante $|V| - 1$ mal relaxieren erhalten wir alle kürzesten Pfade

Bellman-Ford Algorithmus

Warum reicht Dijkstra nicht aus?

- negative Kantengewichte
- negative Kreise $\rightarrow$ Es gibt Knoten die Distanz $-\infty$ zu anderen Knoten haben

Idee:
Ein kürzester Pfad hat maximal Länge $|V| - 1$; wenn wir also jede Kante $|V| - 1$ mal relaxieren erhalten wir alle kürzesten Pfade

Aufgabe

Wenden Sie Dijkstras Algorithmus auf den Graphen an der Tafel an.
Der Startknoten sei durch C gegeben.

Wenden Sie nun den Algorithmus von Bellman-Ford auf den 2. Graphen an der Tafel an.
Der Startknoten sei wieder durch C gegeben.

Aufgabe

Wenden Sie Dijkstras Algorithmus auf den Graphen an der Tafel an.
Der Startknoten sei durch C gegeben.

Wenden Sie nun den Algorithmus von Bellman-Ford auf den 2. Graphen an der Tafel an.
Der Startknoten sei wieder durch C gegeben.

MST

Es sei ein beliebiger zusammenhängender Graph G gegeben.
Nun such man einen Baum der alle Knoten von G verbindet und minimales Gewicht hat.
Im Allgemeinen ergibt sich ein Wald aus genauso vielen Bäumen wie G Zusammenhangskomponenten hat $\rightarrow$ minimal spannender Wald

- **Schnitteigenschaft**: Teilt man die Knoten einer Zusammenhangskomponente in zwei Mengen auf, so kann man die leichteste Kante die diese beide Mengen verbindet im MST verwenden.
- **Kreiseigenschaft**: Die schwerste Kante eines Kreises gehört nicht zum MST.

MST

Es sei ein beliebiger zusammenhängender Graph G gegeben.
Nun such man einen Baum der alle Knoten von G verbindet und minimales Gewicht hat.
Im Allgemeinen ergibt sich ein Wald aus genauso vielen Bäumen wie G Zusammenhangskomponenten hat $\rightarrow$ minimal spannender Wald

- **Schnitteigenschaft**: Teilt man die Knoten einer Zusammenhangskomponente in zwei Mengen auf, so kann man die leichteste Kante die diese beide Mengen verbindet im MST verwenden.
- **Kreiseigenschaft**: Die schwerste Kante eines Kreises gehört nicht zum MST.

MST - Jarník-Prim

procedure jpMST():Set of Nodes
    pick random $s \in V$
    $d = \{\infty, \cdots, \infty\}$;d[s]:=0
    parent[s]:=s
    Q.insert(s):PriorityQueue
    while $Q \neq \emptyset$ **do**
        u:=Q.deleteMin
        d[u]:=0
        foreach e = (u,v) $\in$ E **do**
            if *c(e)* $<$ d[v] **then**
                d[v] := *c(e)*
                parent[v] := u
                if v $\in$ Q **then** Q.decreaseKey(v)
                else Q.insert(v)
    return $\{(v, parent[v]) : v \in V \setminus \{s\}\}$

MST - Jarník-Prim

Der MST wird Stück für Stück aufgebaut.
Algorithmus ist sehr ähnlich zu Dijkstra.

Laufzeit:
$\mathcal{O}((|V| + |E|) + |V| \cdot T_{deleteMin}(|V|) + |E| \cdot T_{decreaseKey}(|V|)$
Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

MST - Jarník-Prim

Der MST wird Stück für Stück aufgebaut.
Algorithmus ist sehr ähnlich zu Dijkstra.

Laufzeit:
$\mathcal{O}((|V| + |E|) + |V| \cdot T_{deleteMin}(|V|) + |E| \cdot T_{decreaseKey}(|V|)$
Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

MST - Jarník-Prim

Der MST wird Stück für Stück aufgebaut.
Algorithmus ist sehr ähnlich zu Dijkstra.

Laufzeit:
$\mathcal{O}((|V| + |E|) + |V| \cdot T_{deleteMin}(|V|) + |E| \cdot T_{decreaseKey}(|V|)$
Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

MST - Jarník-Prim

Der MST wird Stück für Stück aufgebaut.
Algorithmus ist sehr ähnlich zu Dijkstra.

Laufzeit:
$\mathcal{O}((|V| + |E|) + |V| \cdot T_{deleteMin}(|V|) + |E| \cdot T_{decreaseKey}(|V|)$
Binärer Heap: $\mathcal{O}((|V| + |E|) \log |V|)$
Fibonacci Heap: $\mathcal{O}(|E| + |V| \log |V|)$

Union-Find-Datenstruktur

Die Datenstruktur soll Mengen von Knoten verwalten und folgende Operationen möglichst kostengünstig zur Verfügung stellen:

- new / build - eine neue Datenstruktur erstellen die aus n einelementigen Mengen besteht
- find(x) - gibt an in welcher Menge x sich befindet
- union(x,y) - vereinigt die Mengen in denen x und y sind

Union-Find-Datenstruktur

- speichere $Array[1, \cdots, n]$ of $\{1, 2, \cdots, n\}$
- initialisiere das Array mit $\langle 1, 2, \cdots, n \rangle$
- find(i):$\{1, 2, \cdots, n\}$
  if parent[i] = i then return i
  else return find(parent[i])
- union(i,j)
  if find(i) $\neq$ find(j) then parent[i] := j

Laufzeit:
union: $\mathcal{O}(1)$
find: $\mathcal{O}(n)$

Union-Find-Datenstruktur

- speichere $\mathit{Array}[1, \cdots, n]$ of $\{1, 2, \cdots, n\}$
- initialisiere das Array mit $\langle 1, 2, \cdots, n \rangle$
- find(i):$\{1, 2, \cdots, n\}$
  if parent[i] = i then return i
  else return find(parent[i])
- union(i,j)
  if find(i) $\neq$ find(j) then parent[i] := j

Laufzeit:
union: $\mathcal{O}(1)$
find: $\mathcal{O}(n)$

Union-Find-Datenstruktur

- speichere $Array[1, \cdots, n]$ of $\{1, 2, \cdots, n\}$
- initialisiere das Array mit $\langle 1, 2, \cdots, n \rangle$
- find(i):$\{1, 2, \cdots, n\}$
  if parent[i] = i then return i
  else return find(parent[i])
- union(i,j)
  if find(i) $\neq$ find(j) then parent[i] := j

Laufzeit:
union: $\mathcal{O}(1)$
find: $\mathcal{O}(n)$

Union-Find-Datenstruktur

Karlsruher Institut für Technologie

- speichere $Array[1, \cdots, n]$ of $\{1, 2, \cdots, n\}$
- initialisiere das Array mit $\langle 1, 2, \cdots, n \rangle$
- find(i):$\{1, 2, \cdots, n\}$
  if parent[i] = i then return i
  else return find(parent[i])
- union(i,j)
  if find(i) $\neq$ find(j) then parent[i] := j

Laufzeit:
union: $\mathcal{O}(1)$
find: $\mathcal{O}(n)$

Union-Find-Datenstruktur

Zur Laufzeitverbesserung kommen zwei Techniken zum Einsatz:

- **Pfadkompression**

```
procedure find(i):{1, 2, ..., n}
    if parent[i] = i then return i
    else k := find(parent[i])
    parent[i] := k
    return k
```

  $\rightarrow$ find amortisiert in $\mathcal{O}(log n)$

- **Union-by-Rank**

  speichere noch zu jedem Knoten seine Höhe im aktuellen Baum (Array)

  verkette die Bäume bei link nun so, dass sie möglichst flach sind

  $\rightarrow$ find in $\mathcal{O}(log n)$

Union-Find-Datenstruktur

Zur Laufzeitverbesserung kommen zwei Techniken zum Einsatz:

- **Pfadkompression**

  procedure find(i):$\{1, 2, \cdots, n\}$
  if parent[i] = i **then return** i
  else k := find(parent[i])
  parent[i] := k
  return k

  $\rightarrow$ find amortisiert in $\mathcal{O}(logn)$

- **Union-by-Rank**

  speichere noch zu jedem Knoten seine Höhe im aktuellen Baum (Array)
  verkette die Bäume bei link nun so, dass sie möglichst flach sind
  $\rightarrow$ find in $\mathcal{O}(logn)$

MST - Kruskal

```
procedure kMST():Set of Nodes
    Tc:UnionFind(|V|)
    s:Set of Edges
    sort(E) in ascending order of weight
    foreach (u,v) ∈ E do // E is sorted
        if Tc.find(u) ≠ Tc.find(v) then
            s.add((u,v))
            Tc.union(u,v)
    return s
```

Laufzeit: $\mathcal{O}(|E| \log |E|)$

Kruskal benötigt nur eine Kantenliste.

MST - Kruskal

```
procedure kMST():Set of Nodes
    Tc:UnionFind(|V|)
    s:Set of Edges
    sort(E) in ascending order of weight
    foreach (u,v) ∈ E do // E is sorted
        if Tc.find(u) ≠ Tc.find(v) then
            s.add((u,v))
            Tc.union(u,v)
    return s
```

Laufzeit: $\mathcal{O}(|E| \log |E|)$

Kruskal benötigt nur eine Kantenliste.

MST - Kruskal


```
procedure kMST():Set of Nodes
    Tc:UnionFind(|V|)
    s:Set of Edges
    sort(E) in ascending order of weight
    foreach (u,v) ∈ E do // E is sorted
        if Tc.find(u) ≠ Tc.find(v) then
            s.add((u,v))
            Tc.union(u,v)
    return s
```

Laufzeit: $\mathcal{O}(|E| \log |E|)$

Kruskal benötigt nur eine Kantenliste.

Aufgabe zu MST

Wenden Sie den Jarník-Prim Algorithmus und den Kruskal Algorithmus auf den Graphen an der Tafel an und konstruieren Sie jeweils den MST (Startknoten sei 1).

Kreativaufgabe zu MST

Streaming MST

Gegeben sei ein zusammenhängender Graph G mit n Knoten und m Kanten, dessen Knoten lokal gespeichert sind, und dessen Kanten über eine Netzwerkverbindung o.ä. gestreamt werden. Man hat nicht genug Speicherplatz um alle Kanten lokal zu Speichern, da lokal nur $\mathcal{O}(|V|)$ Platz verfügbar ist. Die Kanten kommen in einer beliebigen Reihenfolge an, sie sind insbesondere nicht sortiert. Die Kanten werden aber einzeln angefordert, wir haben hier kein Echtzeitproblem.

a) Gib einen Algorithmus an, der einen MST von G unter diesen Einschränkungen bestimmt.

b) Verbessere diesen Algorithmus so, dass er nur $\mathcal{O}(|E| log |V|)$ Rechenzeit benötigt.

Kreativaufgabe zu MST

Streaming MST

Gegeben sei ein zusammenhängender Graph G mit n Knoten und m Kanten, dessen Knoten lokal gespeichert sind, und dessen Kanten über eine Netzwerkverbindung o.ä. gestreamt werden. Man hat nicht genug Speicherplatz um alle Kanten lokal zu Speichern, da lokal nur $\mathcal{O}(|V|)$ Platz verfügbar ist. Die Kanten kommen in einer beliebigen Reihenfolge an, sie sind insbesondere nicht sortiert. Die Kanten werden aber einzeln angefordert, wir haben hier kein Echtzeitproblem.

a) Gib einen Algorithmus an, der einen MST von G unter diesen Einschränkungen bestimmt.

b) Verbessere diesen Algorithmus so, dass er nur $\mathcal{O}(|E| log |V|)$ Rechenzeit benötigt.

Zu Übungsblatt 10 - Graphalgorithmus

Karlsruher Institut für Technologie

Es sei ein Graph $G = (V, E)$ gegeben. Außerdem eine Abbildung φ und γ die jedem Knoten $v \in V$ ein Knotengewicht bzw. jeder Kante $e \in E$ ein Kantengewicht folgendermaßen zuordnet: $\varphi : V \to \mathbb{R}_+, v \mapsto \varphi(v)$ bzw. $\gamma : E \to \mathbb{R}_+, e \mapsto \gamma(e)$.
Nun sei ein Startknoten $s \in V$ gegeben. Es wird nun der Baum der kürzesten Pfade (geringstes Gewicht) in G gesucht.
Geben Sie einen Algorithmus an der das Problem in Zeit $\mathcal{O}(|V| + |E|)$ löst.

Nun seien keine Kantengewichte mehr gegeben, sondern nur noch die Knotengewichte mittels φ.
Finden sie nun einen Algorithmus der das obige Problem auf diesem Graphen löst.

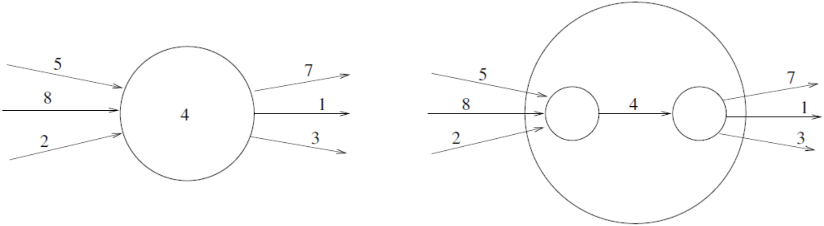
Zu Übungsblatt 10 - Graphalgorithmus

Es sei ein Graph $G = (V, E)$ gegeben. Außerdem eine Abbildung φ und γ die jedem Knoten $v \in V$ ein Knotengewicht bzw. jeder Kante $e \in E$ ein Kantengewicht folgendermaßen zuordnet: $\varphi : V \to \mathbb{R}_+, v \mapsto \varphi(v)$ bzw. $\gamma : E \to \mathbb{R}_+, e \mapsto \gamma(e)$.
Nun sei ein Startknoten $s \in V$ gegeben. Es wird nun der Baum der kürzesten Pfade (geringstes Gewicht) in G gesucht.
Geben Sie einen Algorithmus an der das Problem in Zeit $\mathcal{O}(|V| + |E|)$ löst.

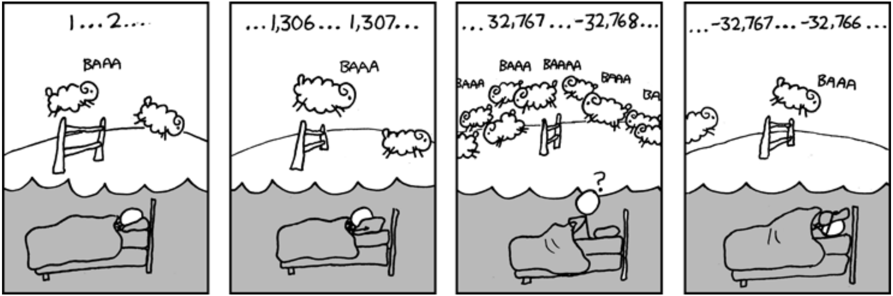
Nun seien keine Kantengewichte mehr gegeben, sondern nur noch die Knotengewichte mittels φ.
Finden sie nun einen Algorithmus der das obige Problem auf diesem Graphen löst.

Lösungsskizze

Kürzeste Wege mit Knotengewichten:



Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.



stackoverflow.com