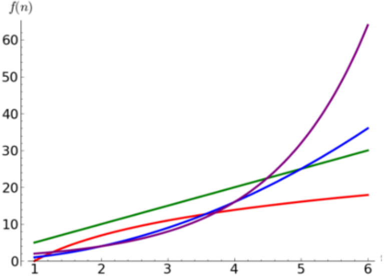
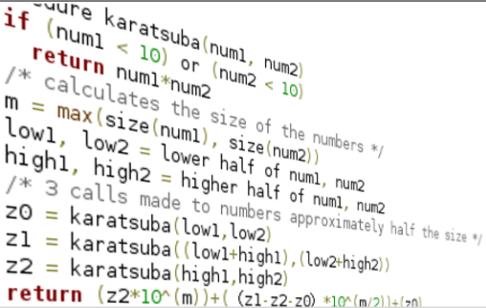
Karlsruher Institut für Technologie

Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 1. Juli 2013

INSTITUT FÜR THEORETISCHE INFORMATIK, PROF. SANDERS





Übungsblatt

- Aufgabenstellung lesen!
- "Gegenkanten" von Baumkanten sind auch Rückwärtskanten

Kreativaufgabe

Betrachte eine Menge von Währungen C mit einem Umtauschkurs von r_{ij} (man erhält r_{ij} Einheiten von Währung j für eine Einheit von Währung i).
Eine Währungs-Arbitrage ist möglich, wenn es eine Folge von elementaren Umtauschoperationen (Transaktionen) gibt, die mit einer Einheit einer Währung beginnt, und mit mehr als einer Einheit derselben Währung endet.
Beschreibe einen Algorithmus, mit dem man für eine gegebenen Umtauschmatrix bestimmen kann, ob Währungs-Arbitrage möglich ist.
Beweise die Korrektheit des Algorithmus.
Hinweis: $\log(xy) = \log x + \log y$, $\log(1) = 0$, siehe auch Buch Seite 207

Minimale Spannbäume (MST)

Gegeben:

- ungerichteter (zusammenhängender) Graph (V,E)
- Kanten als zweielementige Teilmengen $e \subseteq E$
- Kantengewichte c(e)$\in \mathbb{R}_+$

Finde Baum (V,T) mit minimalem Gesamtgewicht, der alle Knoten verbindet

Minimale Spannbäume (MST)

Gegeben:

- ungerichteter (zusammenhängender) Graph (V,E)
- Kanten als zweielementige Teilmengen $e \subseteq E$
- Kantengewichte c(e)$\in \mathbb{R}_+$

Finde Baum (V,T) mit minimalem Gesamtgewicht, der alle Knoten verbindet

Auswahl der Kanten

Schnitteigenschaft

- Sei $S \subseteq V$ beliebig
- Betrachte die Schnittkanten $C = \{\{u, v\} \in E : u \in S, v \in V \setminus S\}$
- Die leichteste Kante in C kann in einem MST verwendet werden

Kreiseigenschaft: Die schwerste Kante auf einem Kreis wird nicht für einen MST benötigt

Auswahl der Kanten

Schnitteigenschaft

- Sei $S \subseteq V$ beliebig
- Betrachte die Schnittkanten $C = \{\{u, v\} \in E : u \in S, v \in V \setminus S\}$
- Die leichteste Kante in C kann in einem MST verwendet werden

Kreiseigenschaft: Die schwerste Kante auf einem Kreis wird nicht für einen MST benötigt

Auswahl der Kanten

Karlsruher Institut für Technologie

Schnitteigenschaft

- Sei $S \subseteq V$ beliebig
- Betrachte die Schnittkanten $C = \{\{u, v\} \in E : u \in S, v \in V \setminus S\}$
- Die leichteste Kante in C kann in einem MST verwendet werden

Kreiseigenschaft: Die schwerste Kante auf einem Kreis wird nicht für einen MST benötigt

Jarník-Prim

```
T := ∅
S := {s} for arbitrary start node s
repeat n-1 times
  find (u,v) fulfilling the cut property for S
  S := S ∪ {v}
  T := T ∪ {(u,v)}
```

Laufzeit

- ähnlich Dijkstra
- O((m+n)log n) mit binären Heaps
- O(m+n log n) mit Fibonacci Heaps
- Wichtigster Unterschied: monotone PQs reichen nicht

Laufzeit

- ähnlich Dijkstra
- O((m+n)log n) mit binären Heaps
- O(m+n log n) mit Fibonacci Heaps
- Wichtigster Unterschied: monotone PQs reichen nicht

Kruskal

```
T := ∅
foreach (u,v)∈E in ascending order of weight do
  if u and v are in different subtrees of (V,T)
  T := T ∪ {(u,v)}
return T
```

Union-Find-Datenstruktur

Verwalte Partition der Menge 1..n, d.h. Mengen (Blocks) $M_1, \ldots, M_k$ mit

- $M_1 \cup \ldots \cup M_k = 1..n$
- $\forall i \neq j : M_i \cap M_j = \emptyset$

Jede Menge hat einen Repräsentanten

Union-Find-Datenstruktur

Verwalte Partition der Menge 1..n, d.h. Mengen (Blocks) $M_1, \ldots, M_k$ mit

- $M_1 \cup \ldots \cup M_k = 1..n$
- $\forall i \neq j : M_i \cap M_j = \emptyset$

Jede Menge hat einen Repräsentanten

Pseudocode

Karlsruher Institut für Technologie

```
Class UnionFind(n:ℕ)
  Procedure union(i,j:1..n)
  Function find(i:1..n):1..n
```

Implementierungen

- Bäume mit parent-Zeigern: find im worst case in $\Theta(n)$
- Pfadkompression (bei find: parent von allen durchlaufenen Knoten auf Repräsentanten umbiegen): find amortisiert in O(log n)
- Union-By-Rank (Rank = Tiefe; halte damit den Baum einigermaßen balanciert): find in O(log n)

Implementierungen

- Bäume mit parent-Zeigern: find im worst case in $\Theta(n)$
- Pfadkompression (bei find: parent von allen durchlaufenen Knoten auf Repräsentanten umbiegen): find amortisiert in O(log n)
- Union-By-Rank (Rank = Tiefe; halte damit den Baum einigermaßen balanciert): find in O(log n)

Implementierungen

- Bäume mit parent-Zeigern: find im worst case in $\Theta(n)$
- Pfadkompression (bei find: parent von allen durchlaufenen Knoten auf Repräsentanten umbiegen): find amortisiert in O(log n)
- Union-By-Rank (Rank = Tiefe; halte damit den Baum einigermaßen balanciert): find in O(log n)

Pfadkompression + Union-By-Rank

- m find- und n link-Operationen brauchen O($m\alpha_T(m, n)$)
- α_T ist die inverse Ackermannfunktion (welche SEHR schnell wächst)
- $\alpha_T(m, n) \in \omega(1)$ aber $\leq$ 4 für "sinnvolle" m,n

Laufzeit Kruskal

- Durch Kantensortierung O(m log m)
- Besser bei ganzzahligen Kantengewichten

Vergleich Jarník-Prim vs. Kruskal

Pro Jarník-Prim

- Asymptotisch gut für alle m,n
- Sehr schnell für m$\gg$n

Pro Kruskal

- Gut für m$\in$ O(n)
- Braucht nur Kantenliste
- Profitiert von schnellen Sortierern (ganzzahlig, parallel,...)
- Verfeinerungen auch gut für große $\frac{m}{n}$