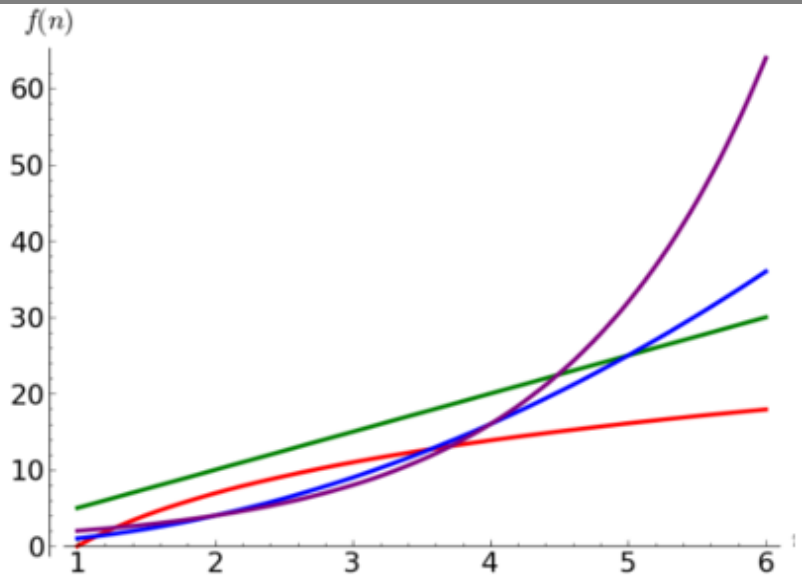
Karlsruher Institut für Technologie

Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 24. Juni 2013

INSTITUT FÜR THEORETISCHE INFORMATIK, PROF. SANDERS




```
if (num1 < 10) or (num2 < 10)
    return num1*num2
/* calculates the size of the numbers */
m = max(size(num1), size(num2))
low1, low2 = lower half of num1, num2
high1, high2 = higher half of num1, num2
/* 3 calls made to numbers approximately half the size */
z0 = karatsuba(low1,low2)
z1 = karatsuba((low1+high1),(low2+high2))
z2 = karatsuba(high1,high2)
return (z2*10^(m))+( (z1-z2-z0) *10^(m/2))+(z0)
```

Übungsblatt

Aufgabenstellung lesen!

- Rang in (a,b)-Bäumen
- Korrektheit begründen

Kreativaufgabe

Gegeben sei ein gerichteter, stark zusammenhängender Graph in folgender Darstellung:

- Ein Knotenarray, das zu jedem Knoten v einen Eintrag mit seiner ID und einen Zeiger auf ein Array mit den von v ausgehenden Kanten enthält. Die Knoten haben eindeutige IDs.
- Das Kantenarray mit den ausgehenden Kanten von v enthält für jede Kante e einen Eintrag mit ihrer ID und der ID des Zielknotens der Kante.

Auf diesem Graphen soll nun eine BFS ausgeführt werden, wobei zu jedem Zeitpunkt nur O(1) zusätzlicher Speicher verwendet werden soll. Die Laufzeit darf dabei schlechter als bei der üblichen BFS sein. Die Art der Darstellung des Graphen soll während der BFS erhalten bleiben, es ist jedoch erlaubt z.B. die Knoten im Knotenarray zu permutieren.

Kreativaufgabe (Forts.)

Geben Sie eine Pseudocode-Implementierung der BFS an die diese Bedingungen erfüllt und begründen Sie warum diese Implementierung nur O(1) zusätzlichen Speicher benötigt. Es soll dabei für jeden Knoten eine unbekannte Funktion f aufgerufen werden, die als Eingabe die Knoten-ID und die Ebene (Entfernung zum Startknoten) des Knotens hat. Es kann angenommen werden, dass f während der Ausführung O(1) und nach der Ausführung keinen Speicher benötigt.

Kürzeste Wege

Problemstellung

- Gegeben seien ein Graph $G = (V, E)$, eine Kosten- oder Kantengewichts-Funktion $c : E \to \mathbb{R}$ und ein Startknoten $s \in V$
- Gesucht sei für alle $v \in V$ die Länge $\mu(v)$ des kürzesten Pfades von s nach v
- oft auch der Pfad selbst

Grundlegendes

- Gibt es immer einen kürzesten Pfad?

Grundlegendes

- Gibt es immer einen kürzesten Pfad? Nein, nicht bei negativen Kreisen.

Grundlegendes

- Gibt es immer einen kürzesten Pfad? Nein, nicht bei negativen Kreisen.
- Einfache Lösung, falls $c(e) \equiv c \geq 0$ für alle $e \in E$?

Grundlegendes

- Gibt es immer einen kürzesten Pfad? Nein, nicht bei negativen Kreisen.
- Einfache Lösung, falls $c(e) \equiv c \geq 0$ für alle $e \in E$? Breitensuche

Grundlegendes

- Gibt es immer einen kürzesten Pfad? Nein, nicht bei negativen Kreisen.
- Einfache Lösung, falls $c(e) \equiv c \geq 0$ für alle $e \in E$? Breitensuche
- Einfache Lösung für DAGs?

Grundlegendes

- Gibt es immer einen kürzesten Pfad? Nein, nicht bei negativen Kreisen.
- Einfache Lösung, falls $c(e) \equiv c \geq 0$ für alle $e \in E$? Breitensuche
- Einfache Lösung für DAGs? Topologische Sortierung nutzen.

Dijkstras Algorithmus

Datenstrukturen, Initialisierung:

- d[v]:= vorläufige Distanz von s nach v (d[v]$\geq \mu$(v))
- parent[v]:= Vorgänger von v auf vorläufigem kürzesten Pfad

Dijkstras Algorithmus

Datenstrukturen, Initialisierung:

- d[v]:= vorläufige Distanz von s nach v (d[v]$\geq \mu$(v))
- parent[v]:= Vorgänger von v auf vorläufigem kürzesten Pfad
- für $v \in V \setminus \{s\}$: d[v]:=∞, parent[v]:=$\perp$
- d[s]:=0; parent[s]:=s

Relaxierung einer Kante (u,v)

- falls d[u] + c(u,v) $\leq$ d[v]: setze d[v] := d[u] + c(u,v) und parent[v]:=u
- Distanzen und parent ändern sich i.A. mehrmals
- Distanz wird dabei nie länger

Relaxierung einer Kante (u,v)

- falls d[u] + c(u,v) $\leq$ d[v]: setze d[v] := d[u] + c(u,v) und parent[v]:=u
- Distanzen und parent ändern sich i.A. mehrmals
- Distanz wird dabei nie länger

Relaxierung einer Kante (u,v)

- falls d[u] + c(u,v) $\leq$ d[v]: setze d[v] := d[u] + c(u,v) und parent[v]:=u
- Distanzen und parent ändern sich i.A. mehrmals
- Distanz wird dabei nie länger

Algorithmus

```
1 initialize d, parent
2 all nodes are non-scanned
3 while ∃ non-scanned node u with d[u]<∞
4     u:=non-scanned node v with minimal d[v]
5     relax all edges (u,v) out of u
6     u is scanned now
```

Implementierung

- verwende adressierbare Prioritätsliste
- Schlüssel ist d[v]
- "Dijkstra = BFS mit PQ statt FIFO"

Implementierung

- verwende adressierbare Prioritätsliste
- Schlüssel ist d[v]
- "Dijkstra = BFS mit PQ statt FIFO"

Korrektheit

- Nur für nichtnegative Kantengewichte garantiert!
- Zeige: jeder erreichbare Knoten wird gescannt...
- und für jeden gescannten Knoten stimmt die Entfernung

Korrektheit

- Nur für nichtnegative Kantengewichte garantiert!
- Zeige: jeder erreichbare Knoten wird gescannt...
- und für jeden gescannten Knoten stimmt die Entfernung

Laufzeit

m-Mal decreaseKey und jeweils n-Mal insert und deleteMin

- Dijkstra (normale Arrays): $O(m+n^2)$
- Binärer Heap (Prioritätsliste): $O((m+n)\log n)$
- Fibonacci-Heaps: $O(m+n \log n)$
- Monotone ganzzahlige Prior.: $O(n+m)$

Laufzeit

m-Mal decreaseKey und jeweils n-Mal insert und deleteMin

- Dijkstra (normale Arrays): $O(m+n^2)$
- Binärer Heap (Prioritätsliste): $O((m+n)\log n)$
- Fibonacci-Heaps: $O(m+n \log n)$
- Monotone ganzzahlige Prior.: $O(n+m)$

Laufzeit

m-Mal decreaseKey und jeweils n-Mal insert und deleteMin

- Dijkstra (normale Arrays): $O(m+n^2)$
- Binärer Heap (Prioritätsliste): $O((m+n)\log n)$
- Fibonacci-Heaps: $O(m+n \log n)$
- Monotone ganzzahlige Prior.: $O(n+m)$

Bellman-Ford

- relaxiere alle Kanten (n-1)-mal
- funktioniert auch mit negativen Kantengewicht
- erkennt negative Kreise
- Laufzeit O(mn)

Bellman-Ford

- relaxiere alle Kanten (n-1)-mal
- funktioniert auch mit negativen Kantengewicht
- erkennt negative Kreise
- Laufzeit O(mn)