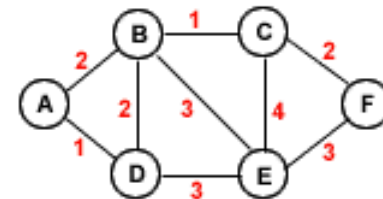




# Effiziente Algorithmen

Hartmut Klauck  
Universität Frankfurt  
SS 06  
27.6.





# Laufzeit Fast Cut

- Jedesmal haben wir 2 rekursive Aufrufe mit  $\lceil n/\sqrt{2} \rceil$  Knoten
- Zusätzlich Laufzeit  $O(n^2)$  für Kontraktion
- Rekursion also:

$$T(n) \leq 2 T\left(\lceil n/\sqrt{2} \rceil\right) + O(n^2)$$

- Lösung:

$$T(n) = O(n^2 \log n)$$



# Korrektheit

- Und interessiert die Wahrscheinlichkeit, mit der tatsächlich ein minimaler Schnitt ausgegeben wird
- Habe also der Min Cut Größe  $k$
- Es gebe in einer Iteration noch einen Schnitt der Größe  $k$ , der Graph habe  $t$  Knoten
- $H_1$  und  $H_2$  seien die Ergebnisse der folgenden Kontraktionen
- Der Algorithmus wird korrekt sein, wenn
  - Ein Schnitt der Größe  $k$  in  $H_1$  oder  $H_2$  „überlebt“
  - Und der entsprechende rekursive Aufruf korrekt ist
- Man kann zeigen:
  - Wenn wir von  $n$  Knoten bis zur Größe von  $t$  Knoten kontrahieren, ist die Wahrscheinlichkeit, dass ein fester minimaler Cut nicht kontrahiert wird mind.

$$\binom{t}{2} / \binom{n}{2} .$$

# Korrektheit

- Wahrscheinlichkeit, daß der Cut beim Übergang  $t$  Knoten nach  $\lceil 1+t/\sqrt{2} \rceil$  Knoten nicht nicht kontrahiert wird:  
$$\frac{\lceil 1+t/\sqrt{2} \rceil \cdot (\lceil 1+t/\sqrt{2} \rceil - 1)}{t(t-1)} \geq 1/2$$
- $P(t)$  sei Wahrscheinlichkeit, daß der Algorithmus auf  $t$  Knoten korrekt ist (Min Cut in  $H_1$  oder  $H_2$  korr.)
- $P(t) \geq 1 - (1 - 1/2 \cdot P(\lceil 1+t/\sqrt{2} \rceil))^2$  (\*)  
[mit Ws  $\frac{1}{2}P(\lceil 1+t/\sqrt{2} \rceil)$  ist in  $H_1$  der MinCut bestimmt]
- Auflösung ergibt  $P(n) = \Omega(1/\log n)$
- Also reichen  $O(\log n)$  Iterationen des Schemas aus und wir erhalten Laufzeit
- $O(n^2 \log^2 n)$  für konstante Korrektheitswahrscheinlichkeit



# Die Abschätzung

- $k=O(\log t)$  sei die Tiefe der Rekursion
- Betrachte  $p(k)$ , eine untere Schranke für Erfolgsws. bei Tiefe  $k$
- $p(k+1)$  und  $p(k)$  „entsprechen“  $P(t)$  u.  $P(\lceil 1+t/\sqrt{2} \rceil)$
- $p(0)=1$
- $p(k+1)=p(k)-p(k)^2/4$       Auflösung von (\*)
- Setze  $q(k)=4/p(k)-1$ :
  - Umstellung ergibt  $q(k+1)=q(k)+1+1/q(k)$
  - Per Induktion:  $k < q(k) < k+H_{k-1}+3$
  - mit  $H_i=\Theta(\log i)$   $i$ -te harmonische Zahl
- Somit ist  $q(k)=k+O(\log k)$  und  $p(k)=\Theta(1/k)$
- Also  $P(t)=\Theta(1/\log t)$



# Datenstrukturen und amortisierte Analyse



# Binomial Heaps

- Die Datenstruktur speichert Datensätze mit Schlüsseln
- Die Datenstruktur unterstützt folgende Operationen:
  - `MakeHeap()`: erzeuge leere Struktur
  - `Insert(H,x,k)`: füge Element  $x$  mit Schlüssel  $k$  ein
  - `Minimum(H)`: bestimme Element mit min. Schlüssel
  - `ExtractMin(H)`: lösche Minimum
  - `Union(H1,H2)`: vereinige zwei solche Mengen
  - `DecreaseKey(H,x,k)`: verringere Schlüssel von  $x$  auf  $k$
  - `Delete(H,x)`: entferne Element  $x$



# Vergleich:

- Heaps:
  - alle Operationen in  $O(\log n)$  oder schneller  
ausser Union:  $O(n)$
- Binomial Heaps erreichen alle Operationen in  $O(\log n)$  [make in  $O(1)$ ]
- Fibonacci Heaps werden Insert, Minimum, Union, Decrease\_Key in amortisierter Zeit  $O(1)$  erreichen



# Definition

- Der binomische Baum ist ein geordneter Baum
- $B_0$  ist ein einzelner Knoten
- $B_k$  besteht aus zwei  $B_{k-1}$ , die verbunden sind:
  - Die Wurzel des einen wird zum linkesten Kind des anderen





# Beobachtung

- Lemma:
  - In  $B_k$ :
    - gibt es genau  $2^k$  Knoten
    - ist die Tiefe  $k$
    - gibt es  $\binom{k}{i}$  Knoten in Tiefe  $i$
    - hat die Wurzel Grad  $k$
- Beweis einfach per Induktion



# Definition

- In binomischen Heaps sind Schlüssel an Knoten gespeichert
- Ein binomischer Heap  $H$  ist eine Menge von binomischen Bäumen mit den Bedingungen binomischer Heaps:
  - Jeder Baum in  $H$  erfüllt:
    - Der Schlüssel eines Knotens ist größer gleich dem Schlüssel seiner Eltern
  - Für jedes  $k$  gibt es höchstens einen Baum mit Wurzelgrad  $k$



# Beobachtung:

- In einem binomischen Heap mit  $n$  Knoten/Schlüsseln gibt es höchstens  $\log n + 1$  Bäume:
  - Betrachte die Binärdarstellung von  $n$
  - Es gibt für jedes Bit höchstens einen Baum
- Damit kann das Minimum in  $O(\log n)$  gefunden werden, indem alle Wurzeln abgesucht werden

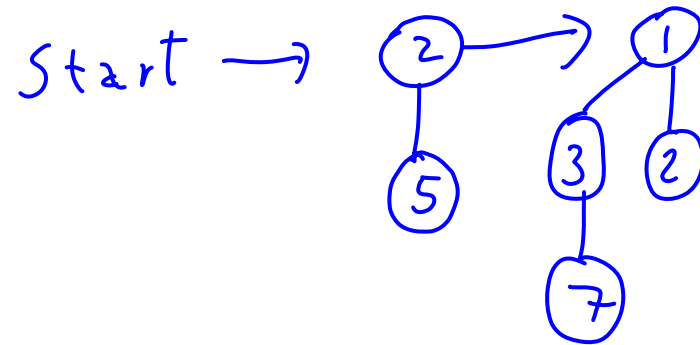


# Repräsentierung binomischer Heaps

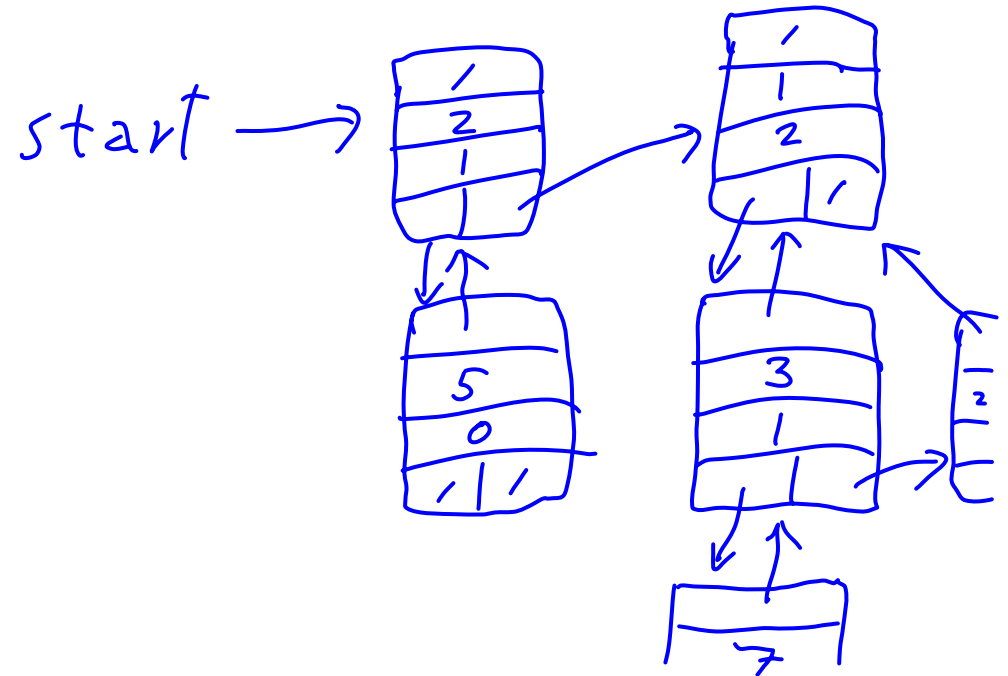
- Jeder Knoten speichert folgende Information:
  - Key: speichert Schlüssel und eventuell weitere Information
  - $\pi$ : Pointer zum Elternknoten
  - child: Pointer zum linkesten Kind
  - sibling: Pointer zu seinem rechten Nachbarn
  - degree: Anzahl der Kinder
- Zusätzlich sind die Wurzeln in einer verketteten Liste
  - Mit steigendem Grad der Wurzel
  - Verwendet Sibling Feld
- Es gibt einen Startpointer: zeigt auf Beginn der Wurzelliste



# Beispiel



| T      |     |
|--------|-----|
| key    |     |
| degree |     |
| child  | sib |





# Operationen

- Make Heap ist leicht in konstanter Zeit zu implementieren
- FindMin: Durchlaufe Wurzelliste, bestimme Minimum, Zeit  $O(\log n)$



# Die Union Operation

- Gegeben sind zwei binomische Heaps  $G, H$ , diese sollen verschmolzen werden
- Wenn ein Baum  $B_i$  nur in  $G$  oder in  $H$  liegt: kein Problem
- $B_i$  in beiden: verschmelze
  - Prozedur  $\text{Link}(y, z)$
  - $y$  und  $z$  seien Wurzel von zwei  $B_i$  und  $\text{key}(z) \leq \text{key}(y)$
  - $\pi(y) := z$
  - $\text{sibling}(y) := \text{child}(z)$
  - $\text{child}(z) := y$
  - $\text{degree}(z)$  um 1 erhöhen
- So entsteht ein  $B_{i+1}$  Baum, jetzt muss eventuell weiter verschmolzen werden!



# Die Union Operation

- High-Level Beschreibung:
  - Bilde eine gemeinsame, nach Grad sortierte Wurzelliste (Merge), eventuell mit doppelten  $B_i$
  - Durchlaufe Wurzelliste
    - Wenn  $B_i$  eindeutig: gehe weiter
    - Sonst: es gibt 2  $B_i$  und 2  $B_{i+1}$ :  
Link für die  $B_i$  (somit gibt es 3  $B_{i+1}$ , von denen die letzten 2 einen  $B_{i+2}$  ergeben werden) und gehe weiter
    - Sonst: Es gibt 2  $B_i$  und 1  $B_{i+1}$ :  
Link für die zwei  $B_i$ , dann für die  $B_{i+2}$
- Laufzeit:  $O(\log n)$



# Insert

- Wir bilden einen neuen binomischen Heap mit einem Knoten, dann Union
- Laufzeit:  $O(\log n)$



# Extract Min

- Bestimme das Minimum  $x$  in  $H$
- Im Baum mit Wurzel  $x$  entferne die Wurzel und füge deren Kinder in einen neuen binomischen Heap  $G$  ein, mit umgedrehter Reihenfolge der Kinder
- Führe Union Operation aus
- Laufzeit: wenn Minimum in  $B_k$  liegt, gilt: Grad an der Wurzel ist  $k$ , daher Erzeugung des neuen Heaps  $O(k) = O(\log n)$
- Also  $O(\log n)$  insgesamt



# Decrease Key

- Eingabe: Bin Heap  $H$ , Knoten  $x$ , Wert  $k$ 
  - Wenn  $k$  größer als Schlüssel: error
  - Verringere Schlüssel von  $x$
  - Ändere Position:
    - Vergleiche Schlüssel mit Elternknoten, tausche wenn kleiner, sonst stop
    - Wenn dabei Wurzel erreicht wird: stop
- Analog zu heaps
- Tiefe  $\leq \log n$ , daher Laufzeit  $O(\log n)$
- Bemerkung: wir brauchen noch ein Array von Pointern, die für Element  $x$  auf den Knoten im bin. Heap zeigen [Vergleiche Disjkstra:  $x$ =Knotenname im Graph für SSSP)



# Delete

- Decrease Key, bis Schlüssel Minimum ist
- Delete Min
  
- Laufzeit  $O(\log n)$