Amortized Analysis

15-750
1/18/19

Applications: Data Structures (DS)
DS eg: Binomial Heaps, Fibonacci Heaps
Union-Find, Splay Trees

Applications using DS
Dijkstra's Alg (graph shortest paths)
Kruskal's Alg (Min Spanning Tree)

See CLRS Ch17 for intro

Main Trick: Sometimes faster to be
lasy than eager

Timing Analysis Methods

1) Worst Case over all inputs: eg Strassen
2) Average over inputs:
   eg Quick Sort pivoting on first element.
3) Amortized Analysis:
   (worst case input, average over time)
4) Randomized:
   (worst case input, average over coin flips)
   eg Randomized QS

Amortized Analysis methods
1) Aggregation Method (won't use)
2) Accounting Method or token method (Kozen)
3) Potential Method (we will use)

Regular Heap (recall)

Heap: 1) Balanced Binary Tree T (Heap)

2) Priority$(p(k)) \leq$ Prior (k)

(Heap property) $p(k) \equiv$ parent (k)

makeheap $\equiv$ make empty tree

findmin $\equiv$ return root

insert $\equiv$ 1) add new leaf.

2) Let it "float up".

deletemin $\equiv$ 1) replace root with a leaf.

2) Let root "float down".

meld $\equiv$ Combine 2 heaps into one.

By adding meld we speedup othr Ops.

Application: Dijkstra's Alg

Recall Dijkstra computes single source shortest path in weighted graph.

i.e. Computes $dis(s,v)\ \forall v \in V$

vertices $= n$
edges $= m$ $\quad m > n$

It uses a priority queue:

Over time it receives keys $k_1, \cdots k_n$ with priorities $Prior(k_1), \cdots Prior(k_n)$

Dijkstra Alg Does:

Ops	# calls	worst case cost	Total cost
make heap (S)	1	1	1
findmin (S)	n	1	n
insert(k, S)	n	$\log n$	$n \log n$
deletemin (S)	n	$\log n$	$n \log n$
decrease key (k, S)	m	$\log n$	$m \log n$

Dominate cost is decrease key!

Goal in next 2 lectures

Make AC of decrease key to be $O(1)$.

Consider some operation Op

Suppose it is called n times

costing $Op_1 \cdots Op_n$

Goal bd total cost $\equiv \sum_{i=1}^{n} OP_i = TC$

Simplest idea: compute $WC \equiv \max_i Op_i$

then $TC \le n \cdot (WC)$

2-Problems:

1) $n(WC)$ too big (over estimate)

2) We won't even consider better alg.

Idea 2: Compute $Avg = \{OP_1, \ldots, OP_n\}$

then $TC \le n \cdot Avg$

Prob: Hard to estimate Avg!

Faster Priority Queues

Types of Heaps: Reg, Binomial, Fibonacci

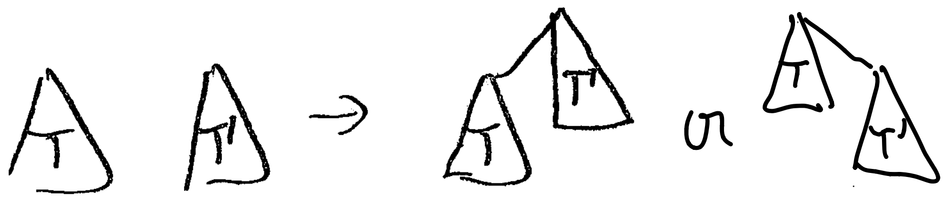
See CLRS Chaps 6, 19, 20 Kozen 8, 9

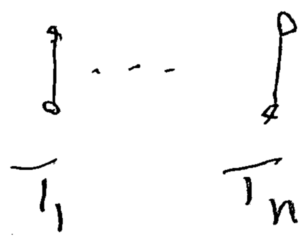
ops	Reg	Binomial Eager / Lazy	Fibonacci Amort
makeheap(S)	O(1)	O(1)	O(1)
findmin (S)	O(1)	O(1)	O(1)
insert(K, S)	O(lg n)	O(lg n)/O(1)	O(1)
deletemin(S)	O(lg n	O(lg n)	O(lg n)
Meld(A, B)	O(n)	O(lg n)/O(1)	O(1)
decrease Key (K, S)	O(lg n)	O(log n)	O(1)

Binomial Heaps (cheap meld)

Idea: 1) balanced → almost balanced

2) binary → (log n)-ary

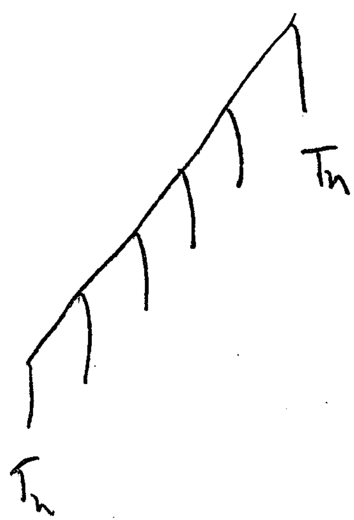
3) Link: 

Problem: 

$T_2' = \text{Meld}(T_1, T_2)$

⋮

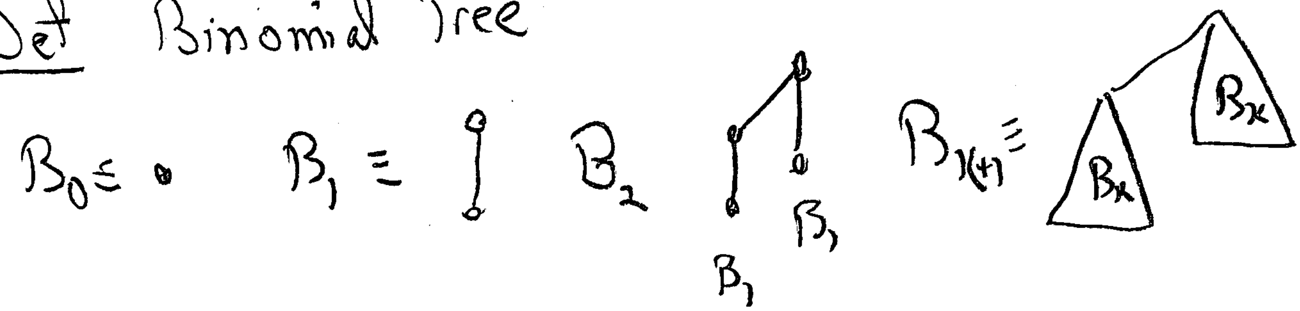
$\text{Meld}(T_{n-1}', T_n)$

We may get: 

(not balanced!)

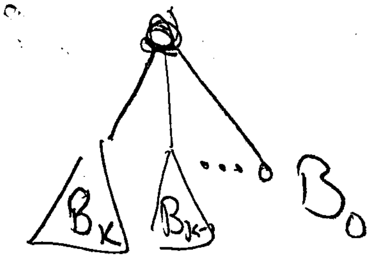
Idea: Only Meld (link) Trees of same size

Def Binomial Tree



---

note: $B_{k+1} \equiv$



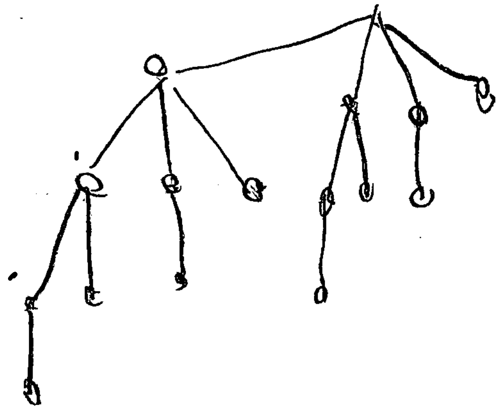
Claim: $\mathrm{depth}(B_k) = k$ &

$|B_k| = 2^k$

Pointers stored

(children)

Why Binomial? $B_4 =$



1
4
6
4
1

Def $B(k,i)$ = # nodes at depth i

Claim: $B(k,i) = \binom{k}{i}$ proof $\binom{n+1}{i} = \binom{n}{i-1} + \binom{n}{i}$

Eager Binomial Heap

$Link(A,B) \equiv$ 1) combine 2 trees of same rank.

2) increment rank of root

$Meld(\bar{A},\bar{B}) \equiv$ Link until at most one tree per rank.

eg $\bar{A} \equiv A_0\ A_1\ \ A_3\ \ A_4$

$\bar{B} \equiv B_0\ B_1\ \ \ \ \ \ B_4$

$\bar{C} \equiv C_1, C_2, A_3, C_5$

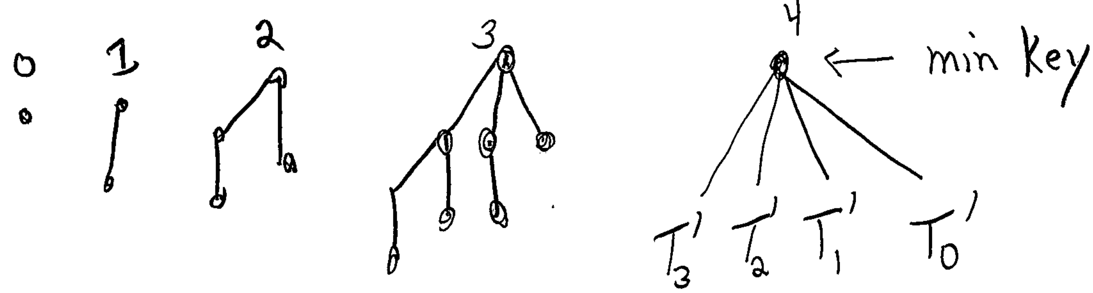
$\text{insert}(k, \bar{A}) \equiv$ 1) makeheap(K, B)
2) meld($\bar{A}$, $\bar{B}$)

$\text{deletemin}(\bar{A}) \equiv$ 1) Suppose $\bar{A} = (T_1, \ldots, T_K)$ (trees)
2) Find tree with min root, T_i.
3) remove T_i from $\bar{A}$
4) remove root(T_i) giving BH $\bar{B}$
5) return meld($\bar{A}$, $\bar{B}$) (Eager)

---

eg

$\bar{A} \equiv (T_0, T_1, T_2, T_3, T_4)$ where



Eager-Deletemin

deletemin(A) $\quad \bar{A} = (T_0, T_1, T_2, T_3)$

$$\bar{B} = (T_0', T_1', T_2', T_3')$$

$$\text{meld}(\bar{A}, \bar{B}) = (T_1'', T_2'', T_3'', T_4'')$$

Note: deletemin is $O(\log n)$ time.

Potential Method

Trick: Add an artificial cost per operation!

Def Φ : state $\rightarrow \mathbb{R}$ a potential

Def unit-cost $= Op_i$

Def Amortized Cost = unit-cost + potential change

$$\sum AC_i = \sum_i \left[Op_i + (\Phi_i - \Phi_{i-1}) \right] = \sum_i Op_i + \Phi_n - \Phi_0$$

$$= TC + \Delta\Phi$$

if $\Delta\Phi \geq 0$ then $TC \leq \sum AC_i$

$AC = \max_i AC_i$ then

$TC \leq n \cdot AC$ if $\Delta\Phi \geq 0$

Amortized analysis of insert

Use potential argument

$$\Phi(\bar{A}) = \#\text{trees}$$

<u>Def</u> unit-cost of insert $= \#$ trees linked $+1$

$= \#$ trees removed $+1$

$$\begin{aligned}\text{Amort-Cost} &= \text{unit-cost} + \Delta\Phi \\ &= 1 + \#\text{trees removed} - (\#\text{tree removed} - 1) \\ &= 2\end{aligned}$$

Note Amort-Cost of Eager-Deletemin still $O(\log n)$

Lazy Melds

1) Tree are kept as a linked list.
2) Lazy-Meld $\equiv$ list concatenation
3) Lazy-Deletemin $\equiv$
   a) Link trees till at most one per rank.
   b) Do Eager-Deletemin
4) Insert using Lazy-Meld.

Amort Analysis of Lazy-Meld DS.

Potential fcn $\Phi \equiv$ # trees.

AC(Insert) = 2

AC(Lazy-Deletemin)

Step a: Unit-Cost $\equiv \log n$ + # trees Deleted

$\Delta\Phi \equiv$ # trees Deleted

AC(Step a) = $\log n$

AC(Step a) = $\log n$ from above

$\equiv O(\log n)$

Next Time

Goal: Find a Lazy DecreaseKey.

Note Deletemin worked because
removing the root of a binomial
shattered it into smaller binomials.