This material takes 1:15

1 Heaps

Shortest path/MST motivation. Discuss Prim/Dijkstra algorithm.
Note: lots more decrease-key than delete.
Response: *balancing*

- trade off costs of operations
- making different parts equal time.

d-heaps:

- $m \log_d n + nd \log_d n$.
- set $d = m/n$
- $O(m \log_{m/n} n)$

1.1 Fibonacci Heaps

Fredman-Tarjan, JACM 34(3) 1987.
`http://www.acm.org/pubs/citations/journals/jacm/1987-34-3/p596-fredman/`
Key principles:

- Lazy: don't work till you must
- If you must work, use your work to "simplify" data structure too
- force user to spend lots of time to make you work
- analysis via potential function measuring "complexity" of structure. user has to do lots of insertions to raise potential, so you can spread cost of complex ops over many insertions

Basic idea:

- Keep collection of heap-ordered trees
- During insertions, do nothing (why bother)
- During linear work for del-min, also simplify structure
- As compare items to find min, "remember" comparisons by heap-ordering compared items.

Heap ordered trees

- definition
- represent using left-child, parent, and sibling pointers

- in contant time, can link two of them (Fibonacci heaps are *mergeable* in constant time)
- in constant time, can add item
- in constant time, can decrease key (split key off, then merge)
- time to delete-min equal number of children and roots.

Goal: use heap-ordered trees, but keep degree small!

- method: ensure that any node has descendant count exponential in degree.
- how?
  - keep a bunch of separate heap-trees
  - only link heaps of same degree
- lemma: if only link heaps of same degree, than any degree-d heap has 2^d nodes.
- creates “binomial trees” (draw)
- “Binomial heaps” do this aggressively—when delete items, split up trees to preserve exact tree shapes.

Algorithm:

- Maintain list of heap ordered trees.
- insert: add to list, update min if necessary
- delete-min:
  - remove smallest root
  - add its children to list of roots
  - scan all roots to find next min
  - consolidate treelist by merging pairs of same-degree trees
  - (note, since scanning anyway, consolidation is free)

Idea: adversary has to do many insertions to make consolidation expensive.

- analysis: potential function ϕ equal to number of roots.
  - $\text{amortized}_i = \text{real}_i + \phi_i - \phi_{i-1}$
  - then $\sum a_i = \sum r_i + \phi_n - \phi_0$
  - upper bounds real cost if $\phi_n \geq \phi_0$.
- insertion real cost 1, potential cost 1. total 2.

- deletion: of r roots at start, $\log n$ roots at end. difference pays for scanning and consolidating all but $\log n$ roots, so amortized cost $O(\log n)$.

Result: constant insert, $O(\log n)$ amortized delete
What about decrease-key?

- bascially easy: cut off node from parent, make root.
- problem: may violate exponential-in-degree property
- fix: if a node loses more than one child, cut it from parent, make it a root (adds 1 to root potential—ok).
- implement using "mark bit" in node if has lost 1 child (clear when becomes root)
- may cause "cascading cut" until reach unmarked node
- why 2 children? We'll see.

Analysis: must show

- cascading cuts "free"
- tree size is exponential in degree

Second potential function: number of mark bits.

- if cascading cut hits r nodes, clears r mark bits
- adds 1 mark bit where stops
- amortized cost: $O(1)$
- note: if cut without marking, couldn't pay for cascade!
  - this is binomial heaps approach. may do same $O(\log n)$ consolidation and cutting over and over.

Analysis of tree size:

- node x. consider *current* children in order were added.
- claim: i^{th} remaining child (in addition order) has degree at least $i-2$
- proof:
  - Let y be i^{th} added child
  - When added, the $i-1$ items preceding it in the add-order were already there
  - i.e., x had degree $\geq i-1$
  - So i^{th} child y had (same) degree $\geq i-1$

    - y could lose only 1 child before getting cut

- let S_k be minimum number of descendants (inc self) of degree k node. Deduce $S_0 = 1$, $S_1 = 2$, and

$$S_k \geq \sum_{i=0}^{k-2} S_i$$

- deduce $S_k \geq F_{k+2}$ fibonacci numbers
- reason for name
- we know $F_k \geq \phi^k$

Practical?

- Constants not that bad
- ie fib heaps reduces comparisons on moderate sized problems
- but, regular heaps are in an array
- fib heaps use lots of pointer manipulations
- lose locality of reference, mess up cache.
- non-amortized versions with same bounds exist.

1.2 Minimum Spanning Tree

minimum spanning tree (and shortest path) easy in $O(m + n \log n)$.
More sophisticated MST:

- why $n \log n$? Because deleting from size-n heap
- idea: keep heap small to reduce cost.
    - choose a parameter k
    - run prim till region has k neighbors
    - set aside and start over elsewhere.
    - heap size bounded by k, delete by $\log k$
    - "contract" regions (a la Kruskal) and start over.

Formal:

- phase starts with t vertices.
- set $k = 2^{2m/t}$.
- unmark all vertices and repeat following

  - choose unmarked vertex
  - Prim until attach to marked vertex or heap reaches size k
  - mark all vertices in region

- contract graph in $O(m)$ time and repeat

Analysis:

- time for phase: m decrease keys, t delete-mins from size-k heaps, so $O(m + t \log k) = O(m)$.

- number of phases:

  - At end of phase, each compressed vertex "owns" k edges (one or both endpoints)
  - so next number of vertices $t' \leq 2m/k$
  - so $k' = 2^{2m/t'} \geq 2^k$
  - when reach $k = n$, done (last pass)
  - number of phases: $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq 2m/n\} \leq \log^* n$.

Remarks:

- subsequently improved to $O(m \log \beta(m,n))$ using edge packets

- chazelle recently improved to $O(m\alpha(n) \log \alpha(n))$

- ramachandran gave optimal algorithm (runtime not clear)

- randomization gives linear.