

1 Intro

Welcome.

- why study this course:
 - theoretical elegance, insight on hardness of problems
 - ability to research in any area of algorithms
 - ability to recognize problems that arise, apply past techniques, and develop new ones
- Varieties of problems and algorithms
 - numerical analysis/linear algebra
 - number theoretic. drives cryptography
 - combinatorial—focus of this course.
 - * things involving permutations (sorting), graphs (shortest paths) and subsets (linear programming).
 - * many *optimization problems*—find the best possible solution
 - * almost always, finitely many solutions. brute force always works. we want something better.
 - * combinatorial optimization: major subarea, but not all we cover (vempala course)
 - aspects of all will arise in others
 - some problems/algorithms draw from multiple areas—eg comp. geom.
- Why deterministic/randomized split
- course summary sheet
- I will teach fast. Slow me down with questions.

2 Heaps

Shortest path/MST motivation. Discuss prim's algorithm.

Note: lots more decrease-key than delete.

d -heaps:

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

2.1 Fibonacci Heaps

Fredman-Tarjan, JACM 34(3) 1987. <http://www.acm.org/pubs/citations/journals/jacm/1987-34-3/p59>
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:

- During insertions, do nothing (why bother)
- During linear work for del-min, also simplify structure

Heap ordered trees

- definition
- represent using left-child and sibling pointers
- in constant time, can link two of them
- in constant time, can add item
- in constant time, can decrease key
- time to delete-min equal number of children.

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

- method: ensure that any node has descendant count exponential in degree.
- how? only link heaps of same degree
- must keep a bunch of separate heap-trees.
- lemma: if only link heaps of same degree, than any degree- d heap has 2^d nodes.
- creates "binomial trees" (draw)

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

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

- analysis: potential function equal to number of roots.
- 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?

- basically 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

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.
- idea: cascading cuts for tree of min-degree 2. number nodes less than twice number of leaves, which is number of decrease keys.

Analysis of tree size:

- node x . consider children in order were added.
- claim: i^{th} added child had degree at least $i - 2$
- proof:
 - when added, x had at least $i - 1$ children (some maybe lost later).
 - So i^{th} child y had 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 $F_k \geq \phi^k$

2.2 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 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 $O(m\alpha(n))$
- recently ramachandran gave optimal algorithm (runtime not clear)
- randomization gives linear.

3 Buckets

In shortest paths, often have edge lengths small integers (say max C).
Observe heap behavior:

- heap min increasing (monotone property)
- max C distinct values.

How to exploit? Idea 1: standard heaps. $O(m \log C)$ or $O(m + n \log C)$.

Dial’s algorithm.

2-level buckets.