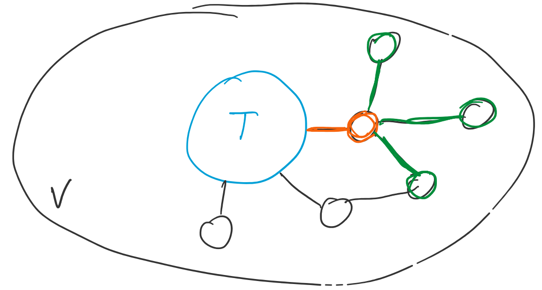
Lec04-heaps-and-clustering

Friday, September 1, 2023 11:01 AM

Recall: Prim's starts from arbitrary node, and grows the tree adding the lightest edge on the frontier

- Need neighbors (u)
- Need Closest Vertex in frontier (updates at each step)



could keep list of all neighbors and their distances, which would take $O(n)$ time to search through and update.

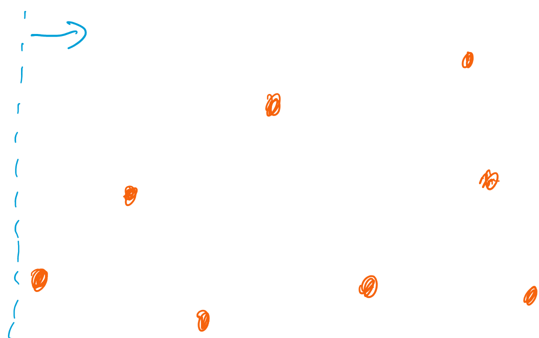
Priority Queue ADT & heaps

A heap (H) holds items i that have keys key(i), and make it easy to find the smallest key.

- H. insert(i)
- H. deletemin() – return smallest key & delete it } exactly the Closest Vertex operation
- H. makeheap(S)
- H. findmin()
- H. delete(i)

(fundamental: insert, deletemin, makeheap; redundant: findmin, delete)

other application includes process priority or plane sweep

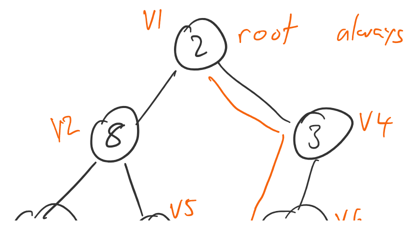


Store points in priority queue, ordered by x-coordinate

more efficient than keeping a sorted list if we only need to repeatedly find the smallest item. (both creation and updating)

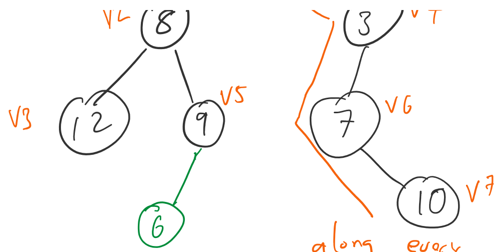
Heap-ordered trees

(heap data structure ≠ heap memory)



- keys of child nodes ≥ key of parent

Reminder: nodes are not just key values



Reminder: nodes are not just key values

along every path, keys monotonically non-decreasing

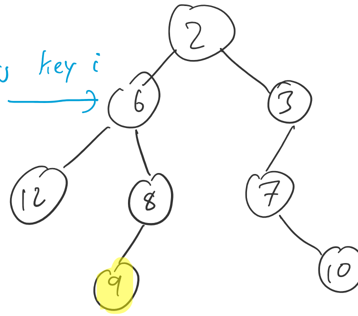
add nodes as leaf
and "sift up" by swapping with
parent if smaller

findmin: $O(1)$
insert / delete / deletemin: $O(\text{tree height}) + \text{time to find leaf}$

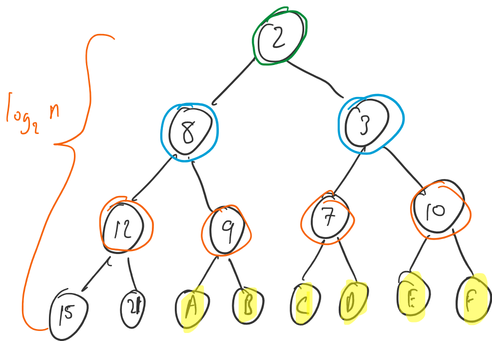
how to keep low — how to find

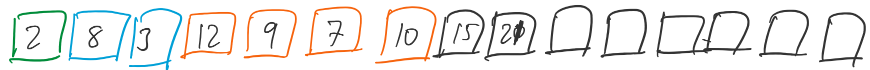
- Use last inserted node for deletes
- Chose next empty spot in complete tree.

delete node containing key i
1. need pointer to node
2. replace key with key j of a leaf
3. delete leaf
4. If $i > j$, sift up.
5. If $i < j$, sift down
   by swapping current node with smallest child until done



Complete Tree $\iff$ Array





$\text{left}(i) = 2i$ if $2i \le n$ else None
$\text{right}(i) = 2i+1$ if $2i+1 \le n$ else None
$\text{parent}(i) = \lfloor i/2 \rfloor$ if $i \ge 2$ else None

Can also create d-heap with higher fan-out

H. decrease key (u, j) – reduce key for item u by j.
Could just delete and reinsert in $O(\log n)$ time.
But more efficient to just reduce the key and siftup in a smaller $O(\log n)$

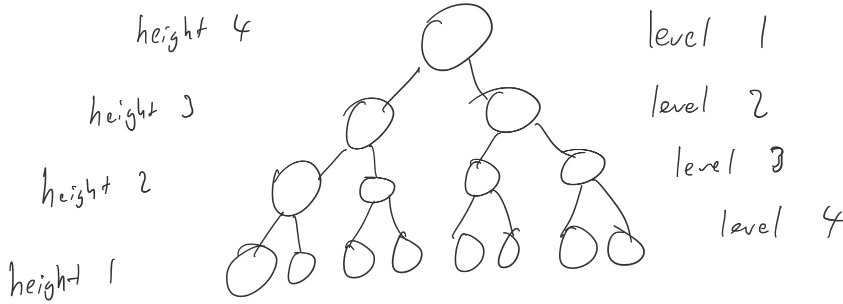
H. makeHeap (S) – could insert n times, but that takes $O(n \log n)$
Can do better $O(n)$:
1. Put items arbitrarily into array (i.e. complete tree)
2. for all elements in reverse order, siftdown. — notice that the bottom is always heap-ordered compared to the active node

Would it work to start from the top and sift up?

2. for all elements ...

Would it work to start from the top and sift up ?

always heap-ordered compared to the active node



at height h, there are $\leq \frac{n}{2^h}$ nodes

↳ only need to siftdown h levels

Runtime: $\sum_h h \cdot \frac{n}{2^h} = n \sum_h \frac{h}{2^h} \leq 2n = O(n)$

ratio test

Claim: $\sum_{h=1}^{\infty} \frac{h}{2^h} = 2$

proof. $\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \cdots$

$= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = 1$

$+ \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = \frac{1}{2}$

$+ \frac{1}{8} + \frac{1}{16} + \cdots = \frac{1}{4}$

$+ \frac{1}{16} + \cdots = \frac{1}{8}$

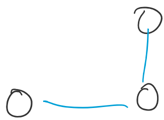
$\vdots$

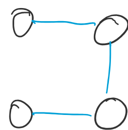
$= 2$

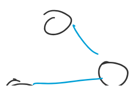
heaps can also be used for sorting by repeatedly deleting the root. $O(n \log n)$

Clustering via MST

represent distance by weighted edges

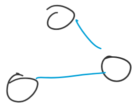






Goal: divide n items into k groups s.t. the minimum distance b/t items in different groups is maximized.

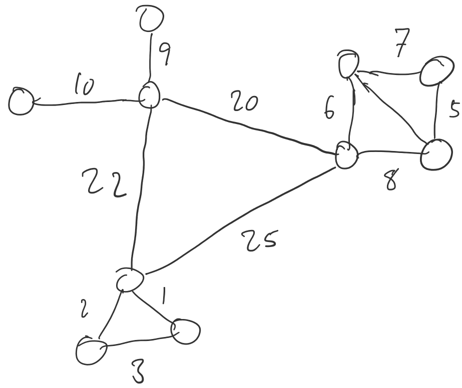
Idea:
- maintain clusters as a set of connected components of a graph
- combine clusters containing closest pair of items by adding an edge
- stop when we have exactly k components



- pair of items by adding an edge
- stop when we have exactly k components

Exactly Kruskal's stopping early before everything is connected

Example of agglomerative clustering.



Correctness proof.

- delete $k-1$ heaviest edges from MST
- The spacing d of the clustering C is the length of the lightest edge deleted ($(k-1)$st.)

Let $C' \neq C$ be another clustering.

Then $\exists\ p_i, p_j$ in the same cluster in C but not in C'.

$\Rightarrow$ must $\exists$ path P from p_i to p_j formed entirely of edges $\leq d$.

$\Rightarrow$ some edge on P must cross clusters of C'.

$\Rightarrow$ separation of $C' \leq d$

$\Rightarrow$ C' not an optimal cluster

Note: picture not perfect because there doesn't have to be an edge directly b/t C_s' and C_t', but it could go to another cluster instead.