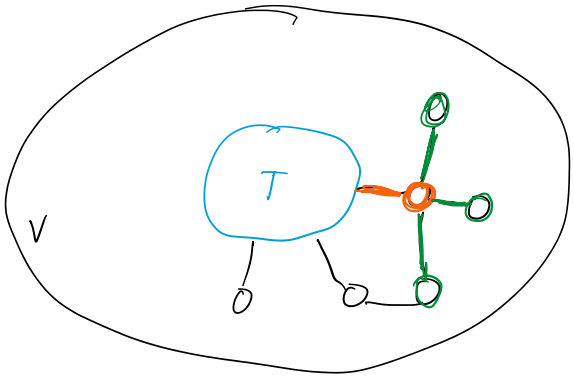
Lec04-heaps

Thursday, August 29, 2024 2:18 PM

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

- Need neighbor(u)
- Need ClosestVertex in frontier
  - updates at each step



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

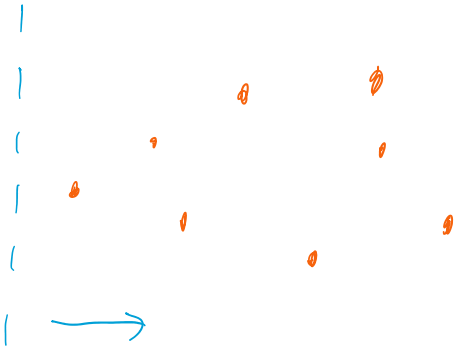
Priority Queue ADT & heaps

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

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

} redundant

other application includes process priority or plane sweep

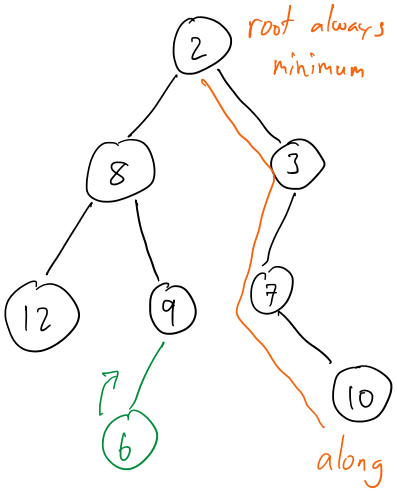


Store points in priority queue ordered by x-coordinate

more efficient than keeping a sorted list if we only need to

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

Heap-ordered trees (heap data structure ≠ heap memory)

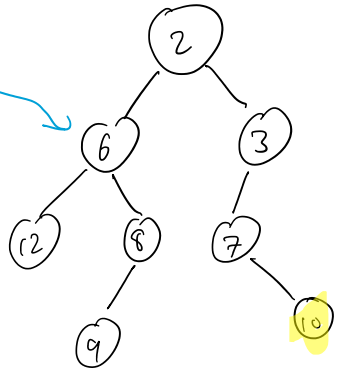


- keys of child nodes ≥ key of parent

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



delete node containing key i

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

findmin: O(1)

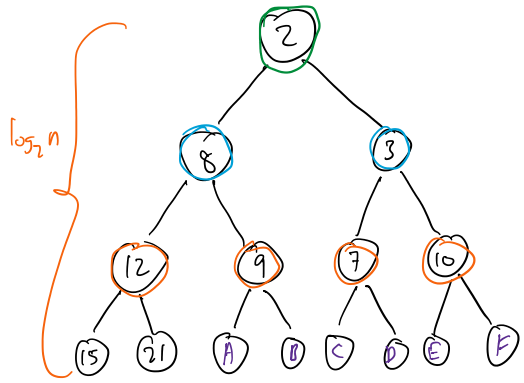
how to keep low

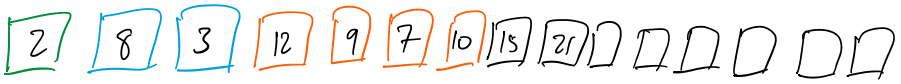
insert/delete/deletemin: O(tree height) + time to find leaf

how to find

- use last inserted node for deletes
- Choose next empty spot in complete tree

Complete Tree ⟺ Array





left(i) = 2i if 2i ≤ n else None

right(i) = 2i+1 if 2i+1 ≤ n else None

parent(i) = $\lfloor i/2 \rfloor$ if i ≥ 2 else None

Can also create a d-heap with higher fan-out

H. decreasekey(u, j) - reduce key for item u to j

Could just delete & reinsert key in $O(\log n)$ time

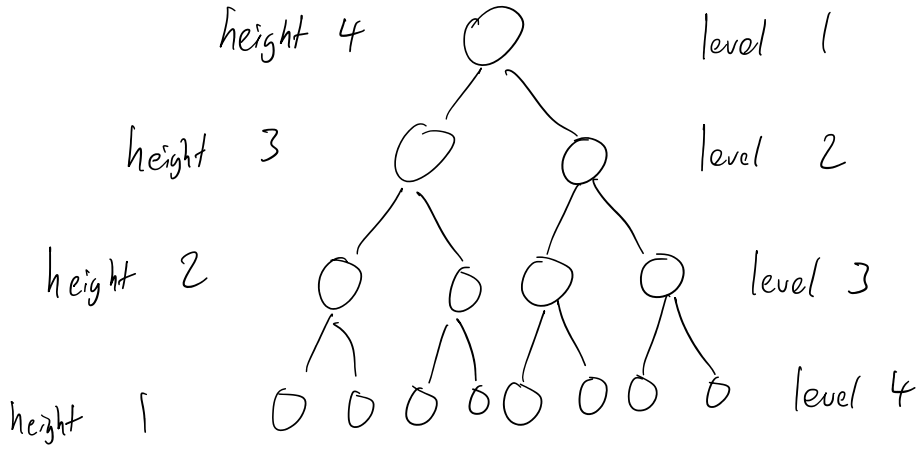
But more efficient to just reduce the key & sift up in smaller $O(\log n)$

H. make heap(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, sift down. - notice that the bottom is always heap-ordered compared to the active node.

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



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

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

↳ only need to sift down h levels

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

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

(or could just use ratio test for convergence)

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

$= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \dots = 1$

$+ \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \dots = \frac{1}{2}$

$+ \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \dots = \frac{1}{4}$

$+ \frac{1}{16} + \frac{1}{32} + \dots = \frac{1}{8}$

$\vdots$

$= 2$

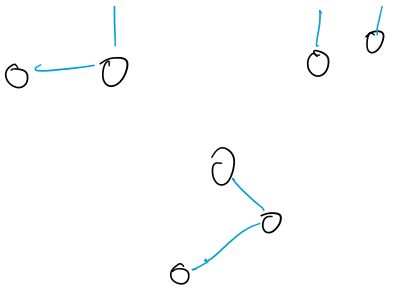
heaps can also be used for sorting by repeatedly deleting the root node $O(n \log n)$

Clustering via MST

represent distance by weighted edges

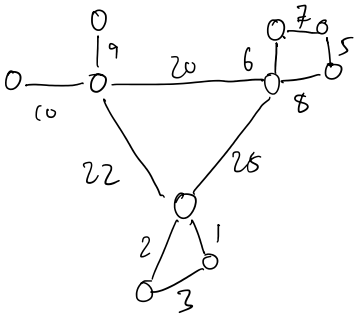
Goal: divide n items into k groups s.t. the minimum distance b/t items in different groups is maximized

Idea: • maintain clusters as a set of connected components of a graph



- Connected components of a graph
- combine clusters containing closest 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 cluster is the length of the lightest edge deleted ($(k-1)$st)

Let $C' \neq C$ be another clustering

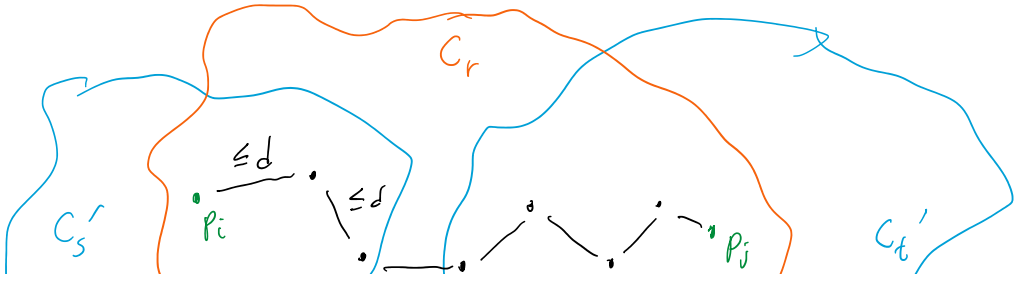
Then $\exists\ p_i, p_j$ in the same cluster in C but not in C'.

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

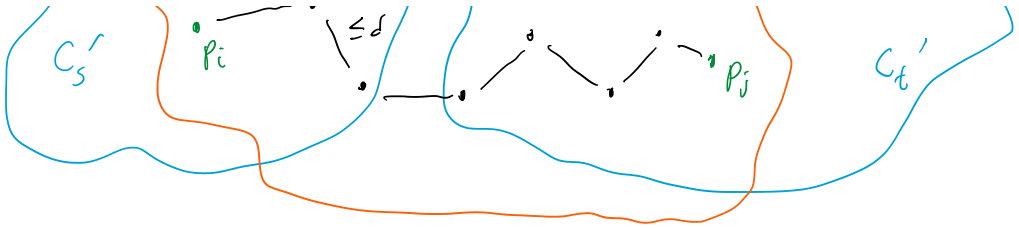
$\Rightarrow$ some edge on P must cross clusters of C'

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

$\Rightarrow$ C' is not an optimal clusterer



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



directly b/t C_s' & C_t',
but it could go to
another cluster instead