Lec03-ADT-heaps

Tuesday, August 25, 2026 2:59 PM

We just finished proving correctness of MST algorithms

Let's circle back around to implementation

Data structures are design specs

- how to store data in memory — storage
- what ops (operations) we can perform on data — ops
- the algs for those ops — algs
- time & space efficiency of algs — complexity

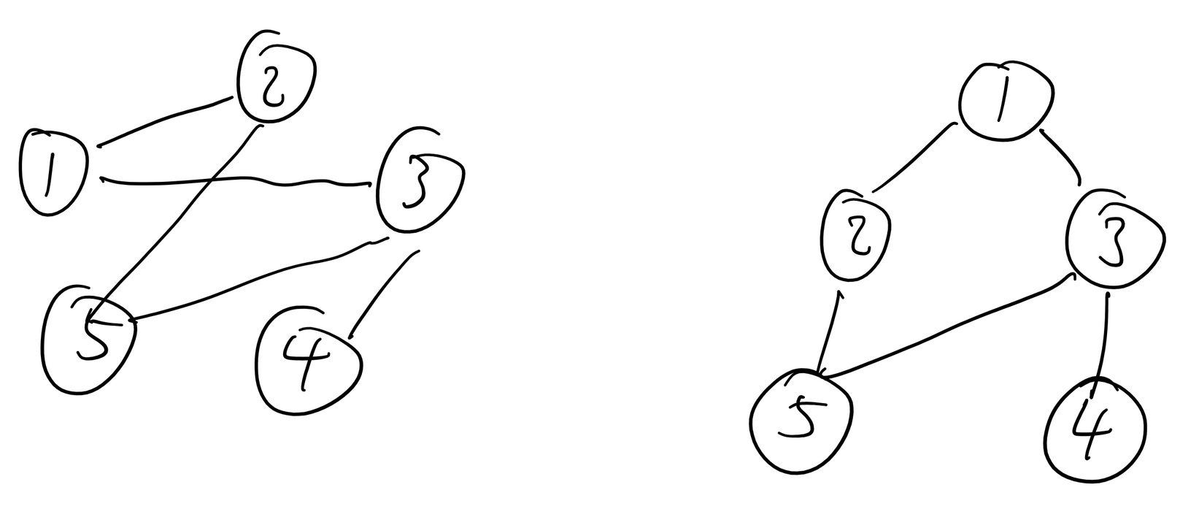
} organization of data can lead to speed

Abstract data types (ADTs) omit implementation, but say what should happen

Ex. Graph ADT G

- G. add_vertex (u) — adds a vertex to G with label u
- G. add_edge (u, v, d) — adds edge $\{u,v\}$ with weight d
- G. has_edge (u, v) — returns True iff $\{u,v\} \in E_G$
- G. neighbors (u) — returns list of vertices adjacent to u in G
- G. weight (u, v)

Multiple drawings can represent same graph



Option 1: Adjacency matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$A_{ij} = 1$ iff $\{i,j\} \in E_G$

or

$A_{ij} = w(\{i,j\})$

Option 2: Adj list

1 → [2 → 3]

2 → [1 → 5]

3 → [1 → 4 → 5]

4 → [3]

5 → [2 → 3]

Option 3: Edge list

$\{1,2\}$
$\{1,3\}$
$\{2,5\}$
$\{3,4\}$
$\{3,5\}$

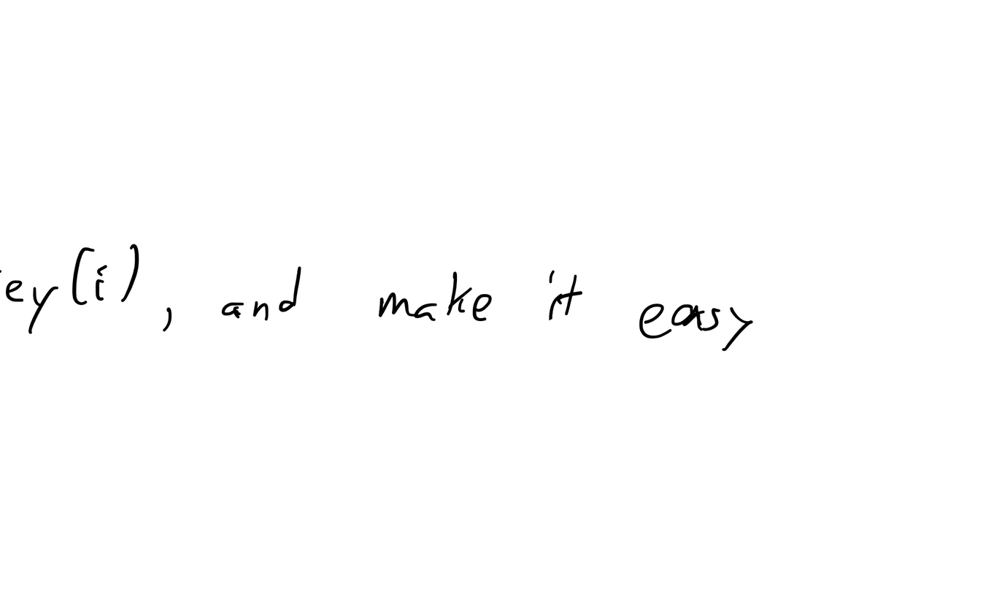
How to implement weights?

How long to find neighbors?

What about adding/deleting an edge/node?

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

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



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

Priority queue ADT & heaps

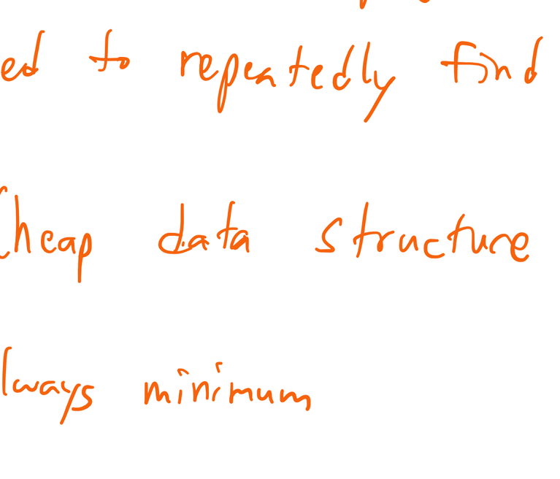
A heap H holds items i that have keys key(i), and make it easy to find the smallest key.

redundant

- H. insert (i)
- H. deletemin (i) — return smallest i & delete it } exactly the closestVertex op
- H. makeheap (S)
- H. findmin ()
- H. delete (i)

} redundant

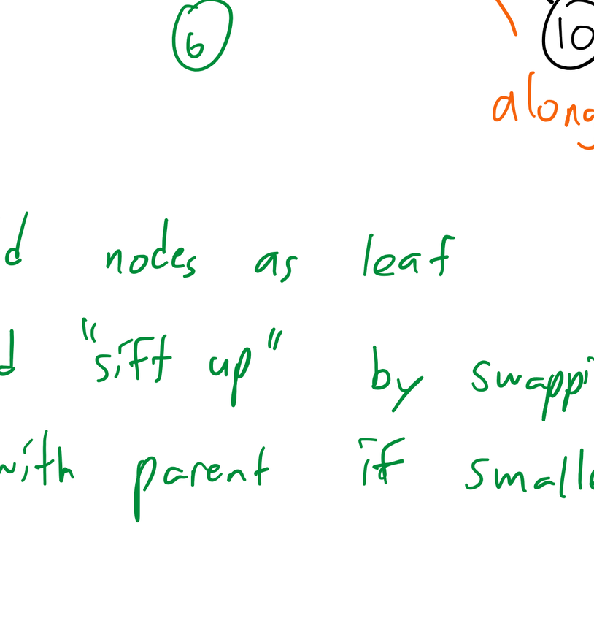
Other applications includes process priority on computers or plane sweep



Store points in priority queue by x-coordinate

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)



along every path keys monotonically non-decreasing

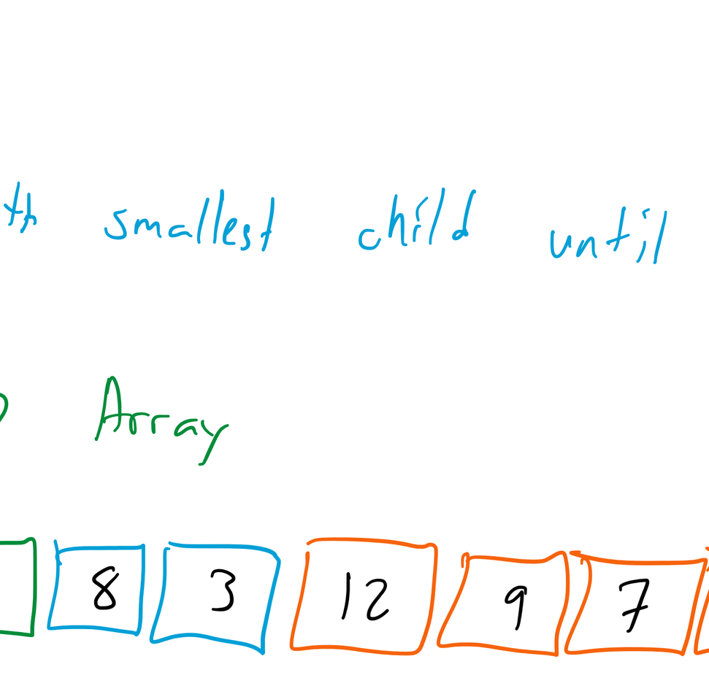
- keys of child nodes ≥ keys of parent

Reminder: nodes have both keys & values/items

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: $O(1)$

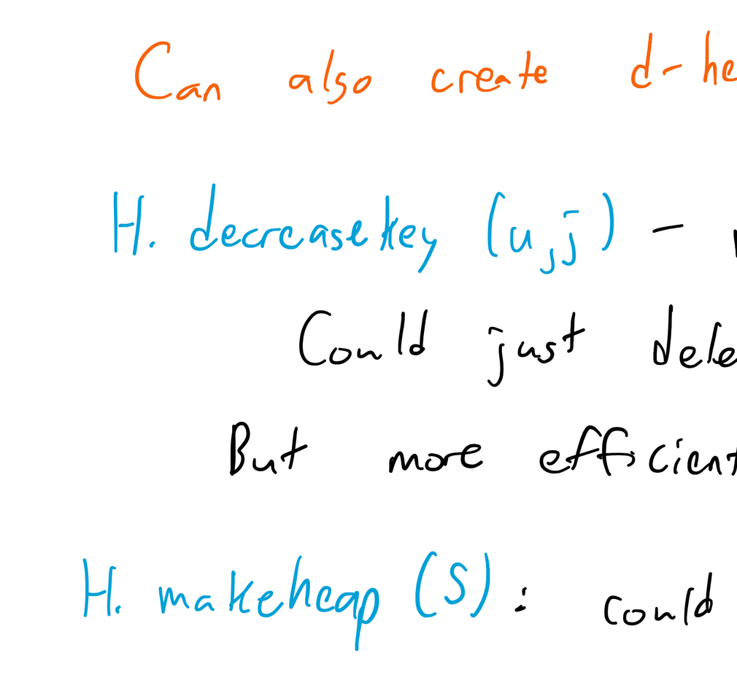
insert/delete: $O($tree height$)$ + time to find leaf

how to keep low?

how to find

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

Complete binary tree ⟺ Array



2	8	3	12	9	7	10	15	21						

$left(i) = 2i$ if $2i \le n$ else None

$right(i) = 2i+1$ if $2i+1 \le n$ else None

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

} 1-indexed

Can also create d-heap with higher fan-cost

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. makeheap (S): could insert n times, but that takes $O(n \log n)$

Can do better $O(n)$:

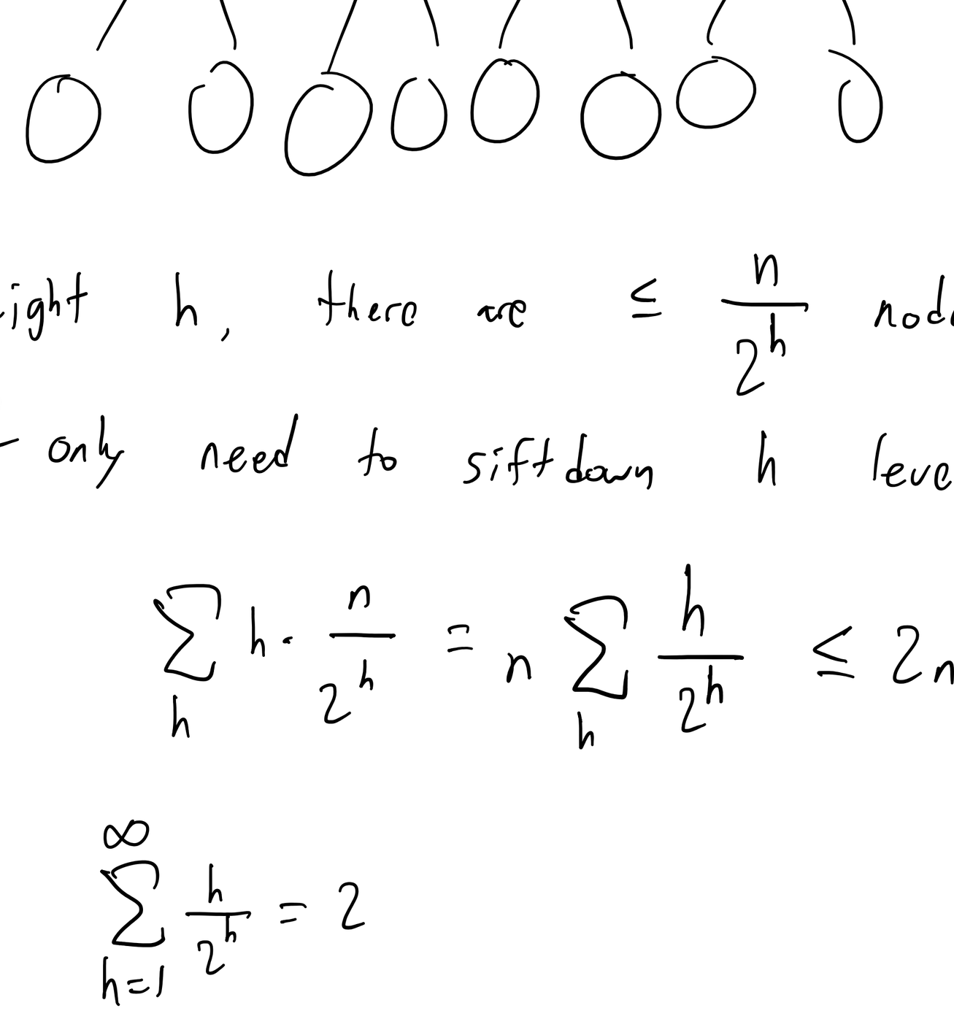
1. put items arbitrarily in array (i.e. complete tree)
2. for all elements in reverse order, sift down — notice that the bottom is always heap-ordered compared to active node

height 4 — level 1

height 3 — level 2

height 2 — level 3

height 1 — level 4



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

- only need to sift down h levels

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

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

(or could just use ratio test for convergence)

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

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

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

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

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

$\vdots$

$= 2$

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

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