Minimum Spanning Trees

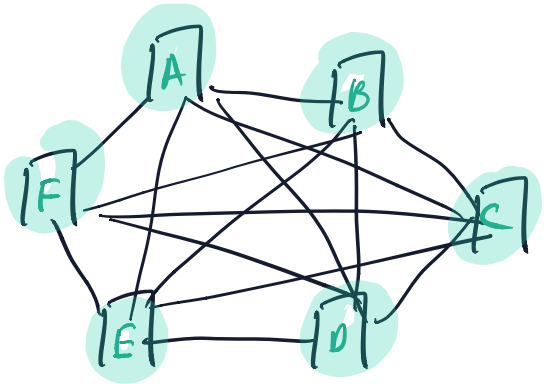
↳ tree has no cycles,

The problem: ↳ all vertices are reachable from each other.

"Given a connected graph G with positive edge weights, find a set of edges that connects all the vertices such that the sum of the weights of the edges is minimum"

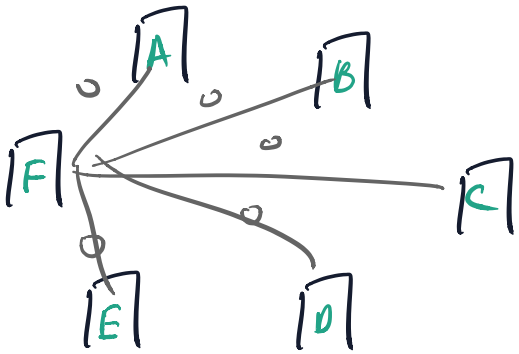
non-negative

Why is it useful?



Collection of servers.

MST →



What would the weights represent?

or finding a minimum spanning tree:

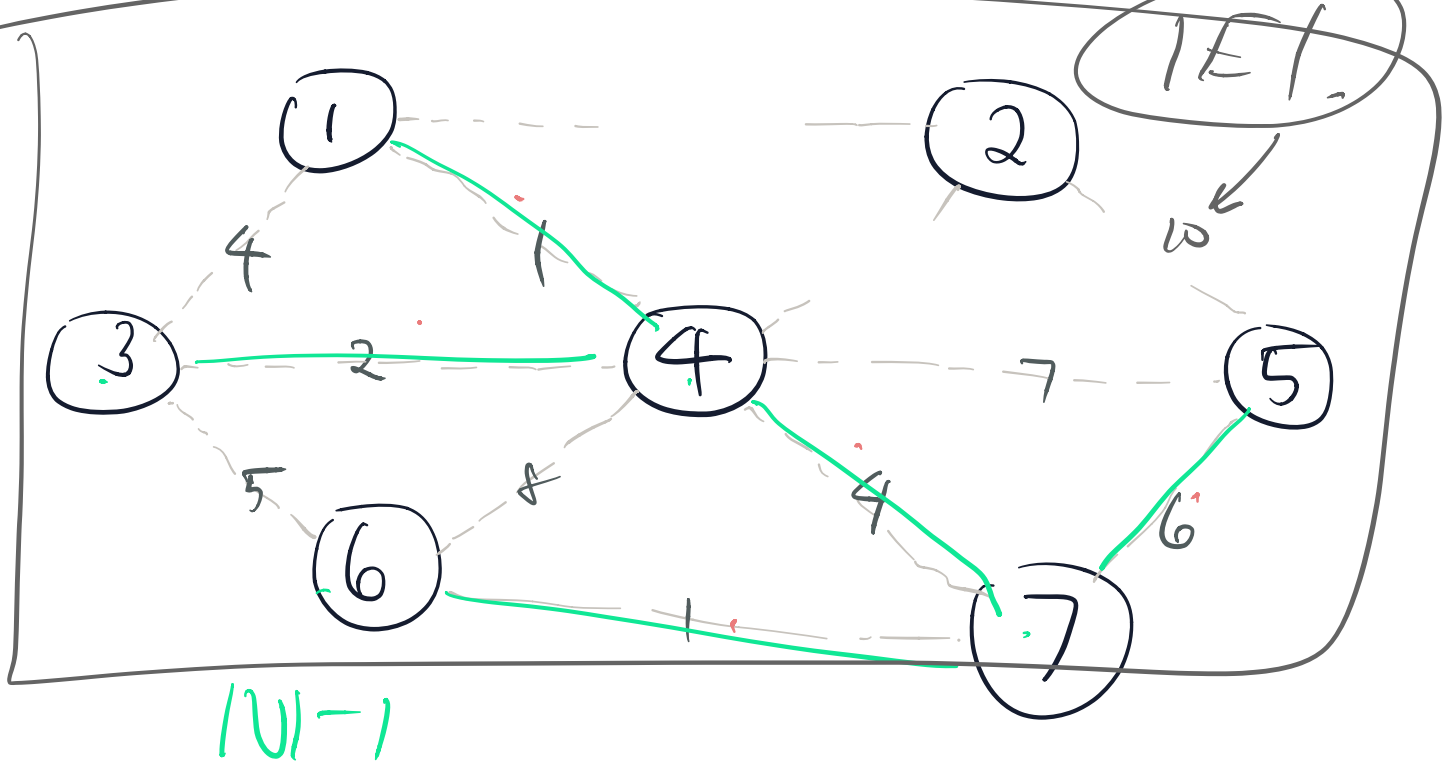
```
typedef Graph MSTree;
MSTree kruskalFindMST (Graph g)
{
    MSTree mst = newGraph (); // MST initially empty
    Edge eList[g->nV]; // sorted array of edges
    edges (eList, g->nE, g);
    sortEdgeList (eList, g->nE);
    for (int i = 0; mst->nE < g->nV - 1; i++) {
        Edge e = eList[i];
        insertE (mst, e);
        if (hasCycle (mst))
            removeE (mst, e);
    }
    return mst;
}
```

complexity?

sort all edges

O(E).

This algorithm effectively constructs the MST by gradually joining together the connected



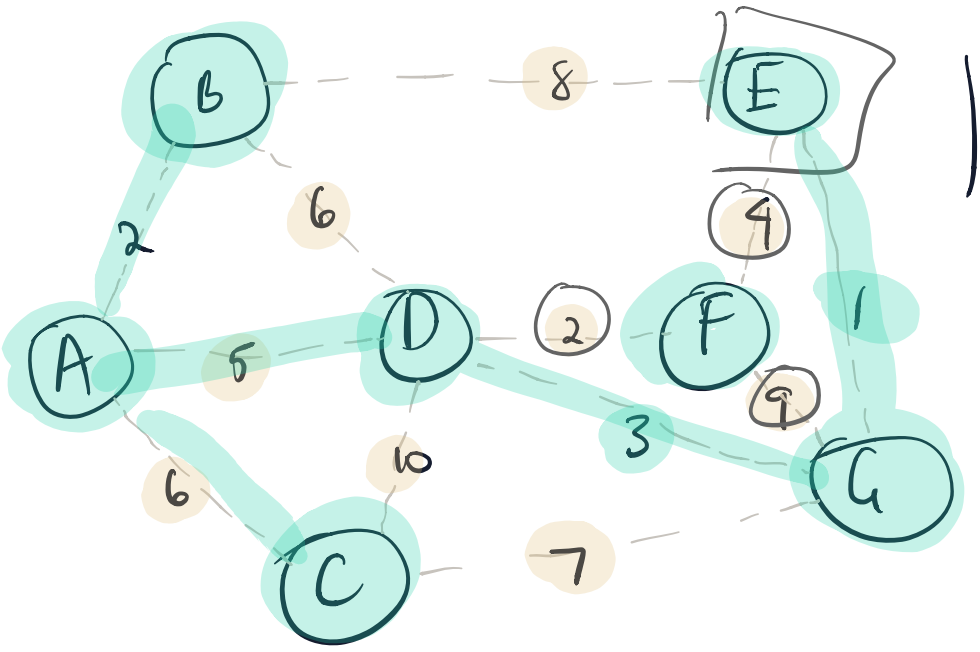
PRIM'S ALGORITHM. complexity? ? (adj. matrix)

❖ ... Prim's Algorithm aka. baby Dijkstra.

Pseudocode:

```
PrimMST(G):
|   Input  graph G with n nodes
|   Output a minimum spanning tree of G
|
|   MST=empty graph
|   usedV={0}
|   unusedE=edges(g)
|   while |usedV|<n do |V|
|   |   find e=(s,t,w)∈unusedE such that { |V|.
|   |      s∈usedV ∧ t∉usedV ∧ w is min weight of all such edges
|   |   }
|   |   MST = MST ∪ {e}
|   |   usedV = usedV ∪ {t}
|   |   unusedE = unusedE \ {e}
|   end while
|   return MST
```

Critical operation: finding best edge



Q: What node does it add?

pq

In both Dijkstra and Prim we want to get "MINIMUM VALUE FROM ARRAY"

```
v = ∞
for (int i=0; i<n; i++)
    if (arr[i] < v) v = arr[i]
```

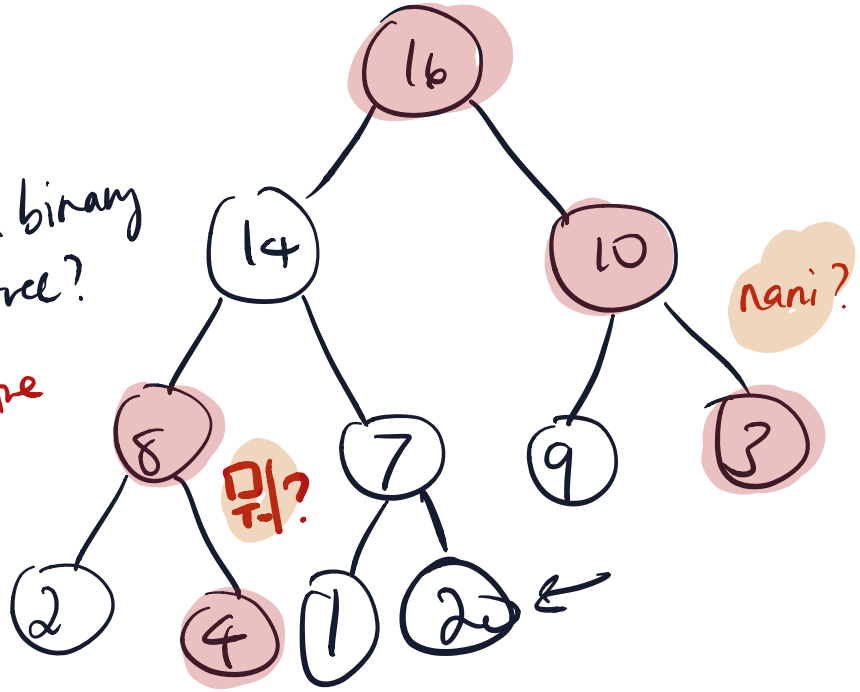
} O(n)

It would be amazing if we have a structure that can:

- Extract Min : O(log n)
- Update key : O(log n)
- Insert : O(log n)

HEAPS

Max heap.



Q: Is this a binary search tree?

NOPE! Look at the red.

(max) HEAP INVARIANTS vs. BST INVARIANTS.

① For a given node x, the nodes under x must have values less than value of x.

① All values in left subtree are less than root. Values in right subtree GREATER.

② Subtrees are BSTs.

- A heap builds an (almost) complete tree.
- Filled from bottom row of the heap, left to right.

Theorem: A heap of n nodes has height ($\log n$?)

Proof: We know that a heap inserts from left to right from the bottom row. Then there must exist some h where:

$$2^h \leq n \leq 2^{h+1} - 1$$

2^k = number of nodes in k deep tree.

Solve for h: $\log_2(2^h) \leq \log_2(n) \leq \log_2(2^{h+1} - 1)$

$$h \leq \log_2(n) < \log_2(h+1)$$

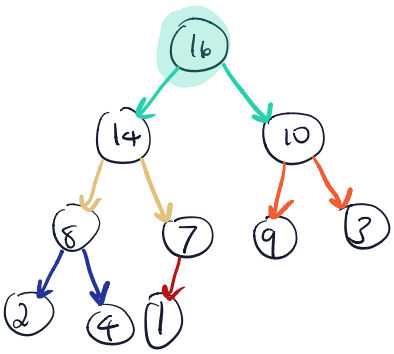
we're happy. we want O(log n) operations, and we have guaranteed O(log n) height on our trees.

Q3. What is the length of the longest path in a heap with 3 nodes? (Tute) 4 nodes? 5 nodes? 25 nodes?

Define path to be num nodes from root to leaf.

IMPLEMENTING HEAPS

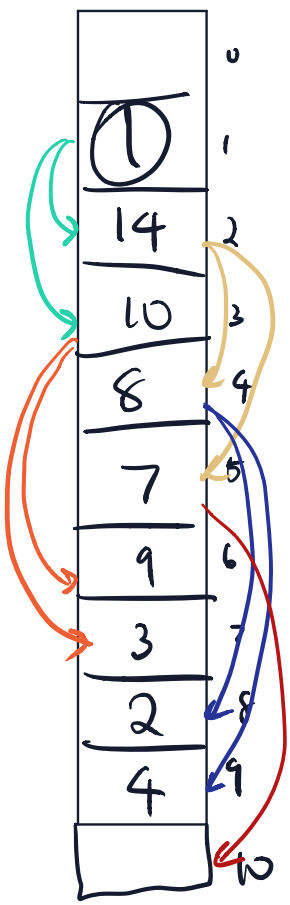
(MAX HEAP)



≡

nelems
10

n max
10



$parent(i) = \lfloor \frac{i}{2} \rfloor$

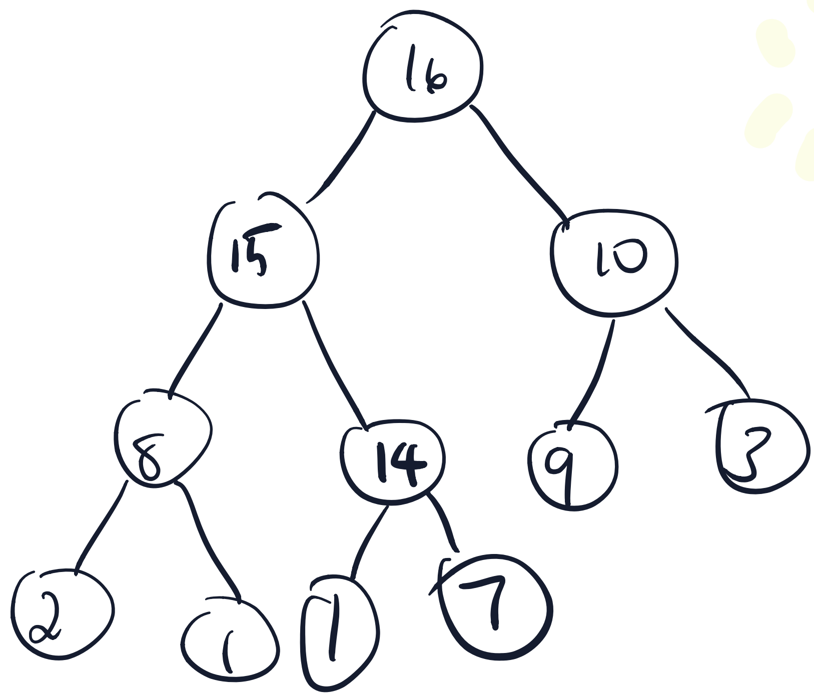
$leftChild(i) = 2i$

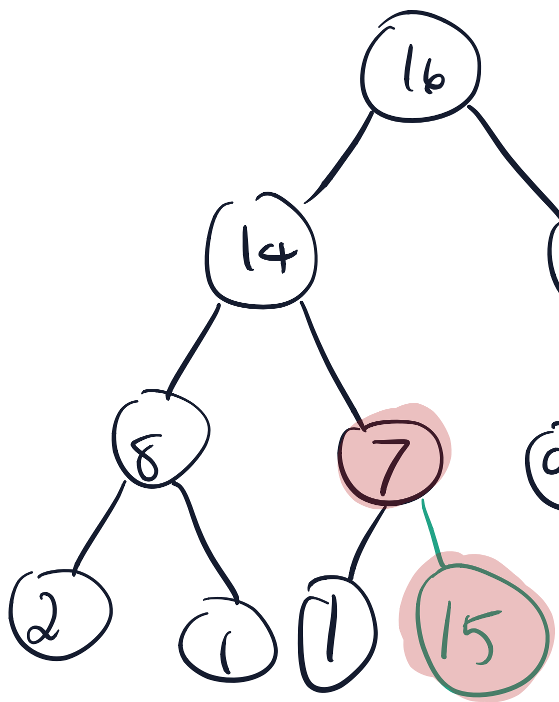
$rightChild(i) = 2i + 1$

MAINTAINING OUR HEAP INVARIANT!

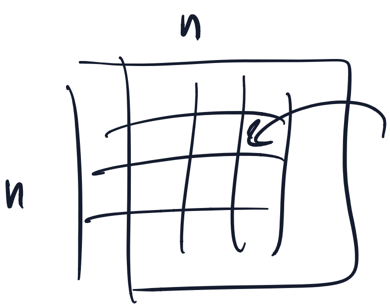
We want to insert an item into our heap. insert(15)

HEAPIFY





choose the bigger of the two!



knight

Q_1: $G = (V, E)$ — V → positions; E → edges are valid moves

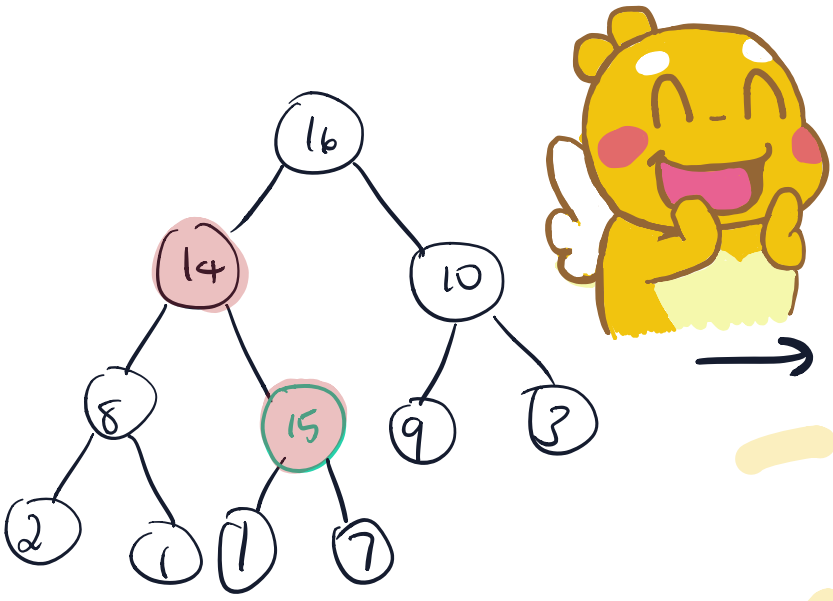
Q_2: Pseudocode.

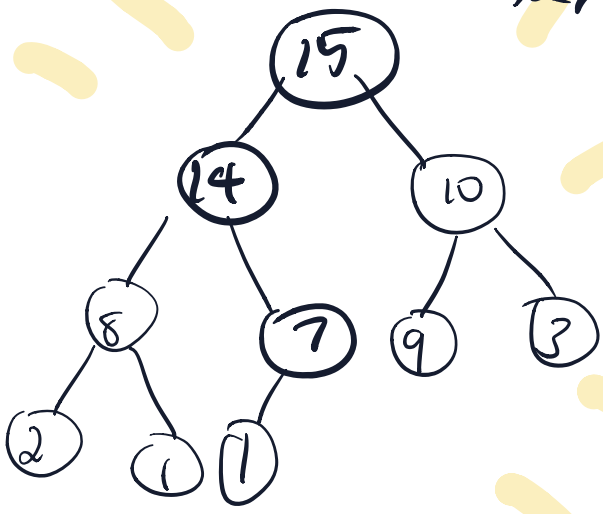
(1) Position start: seen set

↳ traverse on all possible positions

(2) new P. recurse. and mark I've seen

↳ If I can visit.
↳ If not keep the root



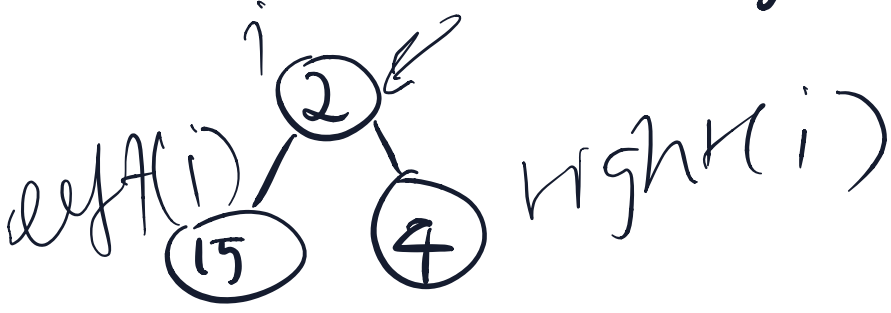


Complexity?

Delete? (always root)

(1) Remove head and move heap.nelems to the root.
Our heap needs to be fixed up.

(2) To maintain maxiness what would you do if you had:



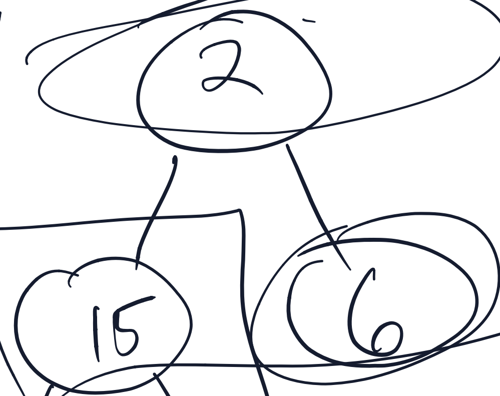
(3) Continue down as element you moved might still violate heap invariant (recursion).

TOP DOWN HEAPIFY

MAX HEAP.

if = 16

smallestIndex = i



initees. = 4

smallest = left(i)

BFS.

swap(A, i, smallestIndex)

G weights 0, 1

fixDown(a, smallestIndex, N)

W. N|.

V →

start shortest path end.

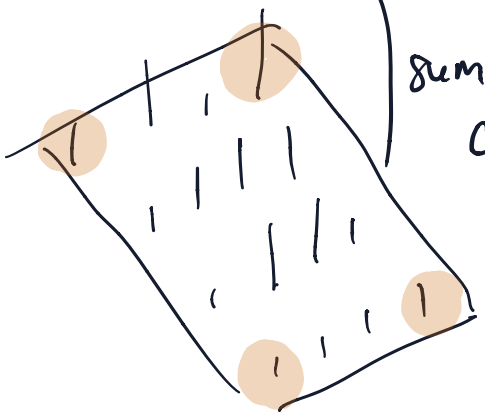
Charges: Pyramid of most water? Choose peak if must end at 0

Compute at every pole $O(N)$

3D container via MOST container.

(1) compute 2D for each point — $O(N^2)$

> Make that as sides need to be on the same 2D plane, do we have



> sum of adj pole
> Choose min to move over?

Merge k sorted lists

$1 \to 4 \to 5$

$1 \to 8 \to 4$

$2 \to 6$

$\xrightarrow{k}$ MERGE.

$1 \to 1 \to 2 \to 3$

$\Rightarrow \hookrightarrow 4 \to 4 \to 5 \to 6.$

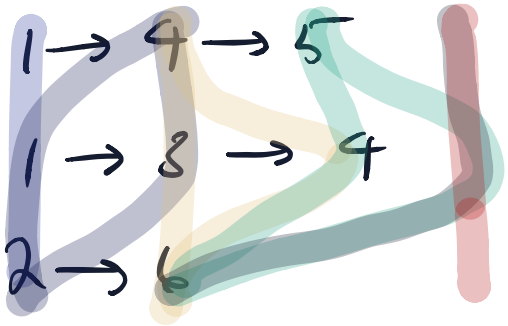
Approach 1: Cumulatively use 2-MERGE on pairs of lists until we get one list. Complexity $O(n^2)$ where n is the sum of all the lists.

$$|k_1|^k + |k_2|^{k-1} + \cdots + |k_n|$$

Say we have $k \times n$ where n is length of maximum.

Then we have $O(n^k)$ algo (polynomial if $k = O(1)$)

Approach 2: Column wise search



(1) Look column wise and have a min structure taking key value and letting you know what index it belongs to.

(2) Min heap gives O(log k) inserts and extractions. Have values be

(v, i)

curr value, curr index.

This would give

$O(nk \log|k|)$

O(nk) elements, for each addition to the list we do O(log k) op. Ideally we would like O(nk) algo.

Approach #3: Divide and conquer