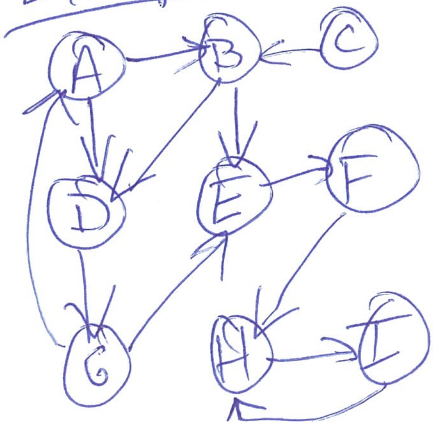
DFS = depth first search

We used DFS for:

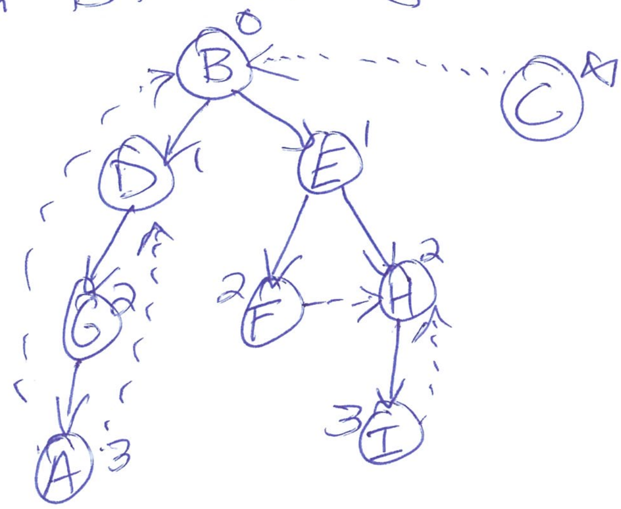
1) connected components of undirected graphs
2) topological sorting of DAGs
3) strongly connected components (SCCs) of directed graphs

BFS = breadth first search
= explore graph in layers

Example:



Run BFS starting at B:



BFS(G, s)

<u>input:</u> $G=(V,E)$ in adj. list. representation (G can be directed or undirected)

<u>output:</u> forall $w \in V$, $dist(w) =$ min # of edges to go from s to w

for all $w \in V$, $dist(w) = \infty$

$dist(s) = 0$

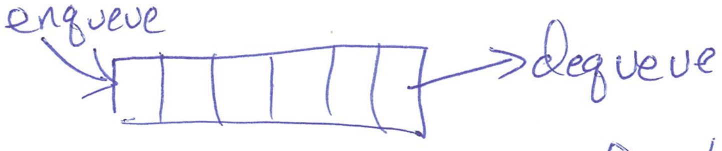
$Q = \{s\}$ (create a new queue just containing $\{s\}$)

While $Q \neq \emptyset$

- $w = \text{dequeue}(Q)$
- for all $(w,z) \in E$
  - if $dist(z) = \infty$
    - then
      - enqueue(Q, z)
      - $dist(z) = dist(w) + 1$

Running time: $O(|V| + |E|) = O(n+m)$

Queue:



FIFO = first-in first-out.

Generalize BFS to allow positive lengths on edges.

Let $\ell(e)$ = length of edge e.

$\ell(e) > 0$.

For a path $P: w_0 \to w_1 \to w_2 \to \cdots \to w_k$

its length is: $\ell(P) = \sum_{i=0}^{k-1} \ell(w_i, w_{i+1})$

$dist(v,w)$ = length of shortest path from v to w

$= \min_P \{ \ell(P) : P$ is a path from v to $w \}$

Goal: for given $s \in V$, find $dist(s,w)$ for all $w \in V$.

BFS solves it when $\ell(e)=1$ for all $e \in E$.

Suppose every $\ell(e)$ is a positive integer.

For every edge e, replace e by a path of length $\ell(e)$ & give each new edge length 1.

Run BFS on this new graph.

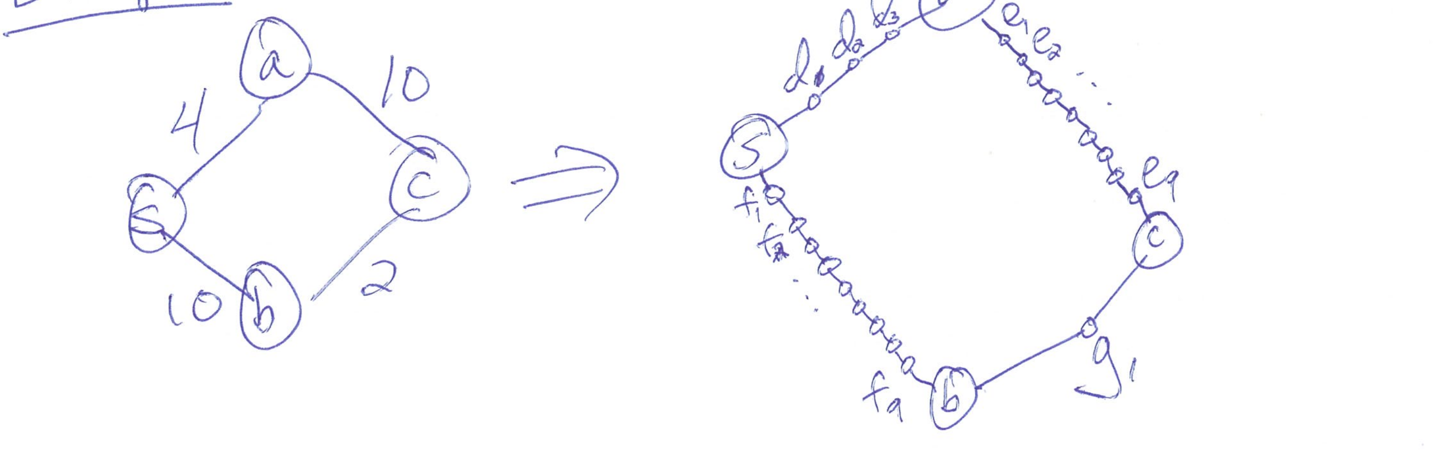
<u>Problem:</u> running time depends on lengths $\ell(e)$ which might be HUGE.

Most of the time BFS is processing "dummy" vertices that we added.

Can we skip such times & only consider times to process original vertices?

Set "alarm clocks" for times of interest.

<u>Example:</u>



time 0: see s

1: see d_1 & f_1

2: see d_2 & f_2

3:

→ 4: see a & f_4

⋮

→ 10: see b & e_6

⋮

→ 12: see c

idea: each vertex w has an alarm clock set at $d(w)$ when we see a new path to w, check if need to adjust w's clock.

High-level algorithm:

Initialize: for s set its alarm for $d(s)=0$
for all $w \in V-s$, set $d(w)=\infty$

Repeat until no more alarms (ignore ∞-time alarms)
Let w be the next alarm clock to go off at time T.
Then: set $dist(w)=T$
for every $(w,z) \in E$:
if $d(z) > T + \ell(w,z)$
then $d(z) = T + \ell(w,z)$.

To implement alarm clocks, use min-heaps data structure

priority queue.

Min-heap data structure H:

H = contains a set of elements (in our case, element = vertex)

each element has a key (in our case, Key = dist(w))

Operations:

Insert(H, w, dist(w)) = add element w with key dist(w) to H

DecreaseKey(H, w, dist(w)) = for $w \in H$, decrease its key to dist(w)

DeleteMin(H) = output the element in H with smallest & delete it from H

For a heap with $\leq n$ elements, $O(\log n)$ time per operation.

Dijkstra(G, ℓ, s)

input: $G=(V,E)$ with $\ell(e)>0$ for each $e \in E$,
& vertex $s \in V$

output: for all $w \in V$,
$dist(w)$ = length of shortest $s \rightarrow w$ path
$prev(w)$ = parent of w on this path

for all $w \in V$, $dist(w) = \infty$
$prev(w) = NULL$

$dist(s) = 0$
$H = \emptyset$
for all $w \in V$, Insert$(H, w, dist(w))$
while $H \neq 0$:
  w = deletemin(H)
  for all $(w,z) \in E$:
    if $dist(w) > dist(w) + \ell(w,z)$
      then: $dist(z) = dist(w) + \ell(w,z)$
        $prev(z) = w$
        DecreaseKey$(H, z, dist(z))$

Running time:

n inserts & n deletes $\Rightarrow O(n \log n)$ time.

$\leq m$ decreaseKeys $\Rightarrow O(m \log n)$ time

$\Rightarrow O((n+m) \log n)$ total time.

if G is connected then

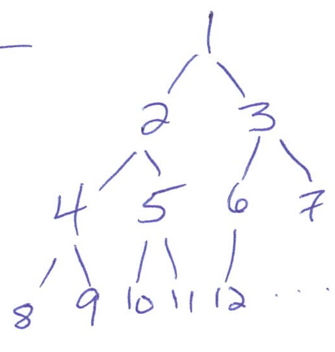
$m \geq n-1$, so it's $O(m \log n)$ total time.

How to implement min-heaps?

Use complete binary tree: every level is full except possibly the bottom level

fill bottom level from left to right

Example:



Can use an array $A[1 \ldots 12]$ to store the keys for the tree

For node in position i of $A[\ ]$

Parent is in position $\lfloor \frac{i}{2} \rfloor$

left child is at $2i$,

right child is at $2i+1$.

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

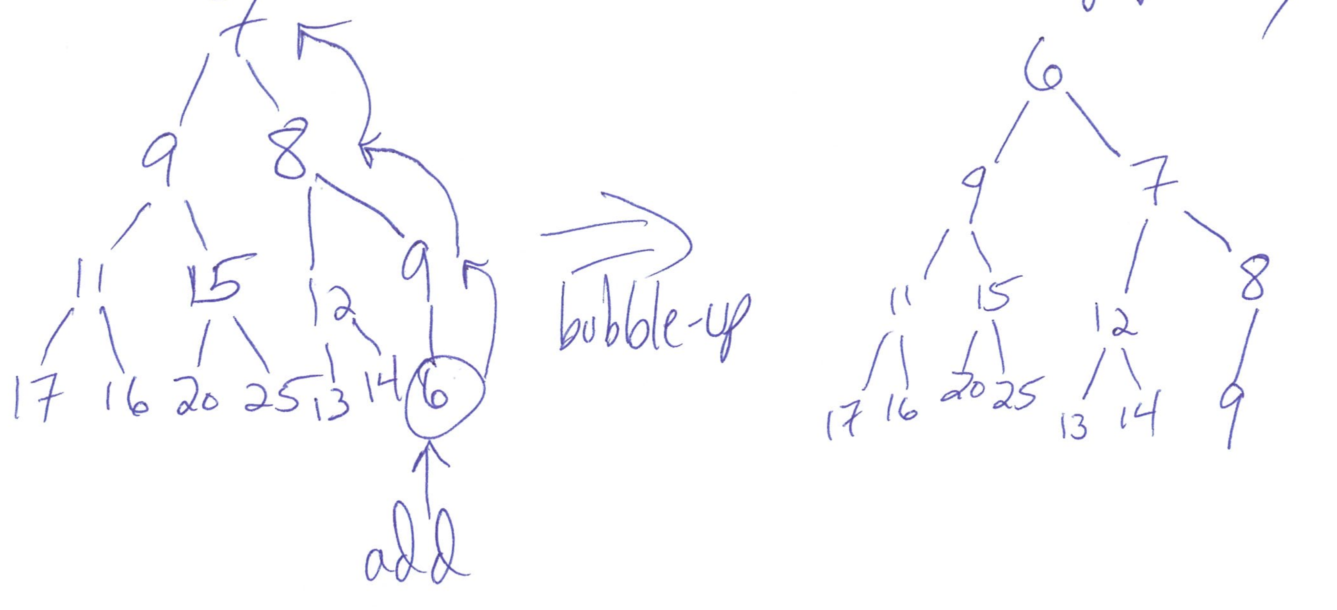
Min-heap property: $A[p(i)] \leq A[i]$

Min key is at root.

Insert: add new element to bottom level, leftmost open spot

this maintains desired shape

then "bubble-up" to maintain min-heap property



To decrease-key: lower its value & then Bubble-up

To Delete-Min:

1) output value at root
2) Delete root
3) Take element from bottom-right & put it at the root
4) Sift-down:

Example: