Breadth First Search/Dijkstra.
Lecture in a minute.

Breadth First Search of graph:
Search with queue instead of stack.
Get “distances” from source.
Proof idea: queue has distance 0, then level 1, ...

Dijkstra: shortest paths in weighted graph:
Replace weights by paths + BFS
Implement using priority queue.
Idea: ignores “new” nodes.
Runtime: $|V|$ extracts, $|E|$ reduce-key for p-queue.

Priority Queue:
Implementation: degree $d$ tree.
Heap Property: children larger than parent.
Minimum at top.
Remove min: $O(d \log_d n)$ time: Replace min/percolate down.
Reduce Key: $O(\log_d n)$: percolate up.
 Paths in graphs.

$G = (V, E)$.  
Distance - length of shortest path between $u$ and $v$.  
Source $s$.

**Definition:**

$\text{Distance}(s) = 0$, $\text{Distance}(v) = \min_{N(v)} d(v) + 1$

Try depth first search.  
Do you think this will work?

D and A at distance 1.
Algorithm 2: Pick it up!

\[ G = (V, E) . \]
Distance - length of shortest path between \( u \) and \( v \).

Pick it up by \( s \)!
So proceed layer by layer.
Distance 0.
Neighbors of “0” node are distance 1 nodes.

Untouched Neighbors of $d$ nodes are $d + 1$ nodes.
What data structure should we use to organize this?
Queue: Breadth First Search

d(s) = 0;
visited[s] = true
put S in Q
While u = Q.pop():
    foreach (u,v):
        if (visited[v] == false):
            visited[v] = true
            d[v] = d[u]+1
            put v in Q
Nodes “explored” in order of distance from s.
Correctness.

**BFS:** while node \( u \) in queue;  
add unvisited neighbors of \( u \) to queue.

**Inductive statement:**  
Queue only has distance \( d \) and \( d + 1 \) nodes.  
When all distance \( d \) nodes explored,  
queue has all (and only) distance \( d + 1 \) nodes.

**Prove queue property.**  
**Base:** \( s \) is explored first.  
\[ \implies \text{all its neighbors in queue} \implies \text{queue has all distance 1 nodes.} \]

\( d + 1 \) node has a distance \( d \) neighbor def of distance  
last distance \( d \) node explored (queue!)  
\[ \implies \text{every node at distance } d + 1 \text{ is visited} \implies \text{queue has all dist. } d + 1 \text{ nodes.} \]

Has no \( l > d + 1 \) nodes since only \( \leq d \) level nodes explored.  
Has no \( l < d \) nodes, since removed by induction.  
\[ \implies \text{only distance } d + 1 \text{ nodes.} \]
In action!

**Breadth First Search:**

\[
d(s) = 0 \\
\text{visited}[s] = \text{true} \\
\text{put } S \text{ in } Q \\
\text{While } u = Q: \\
\quad \text{foreach } (u,v): \\
\quad \quad \text{if } (\text{visited}[v] == \text{false}): \\
\quad \quad \quad \text{visited}[v] = \text{true} \\
\quad \quad \quad d[v] = d[u]+1 \\
\quad \quad \text{put } v \text{ in } Q
\]
S.F. to L. A.: One hop.
S.F. to Vegas: Two hops (which two).
Reno to L.A.: Three hops. not two!
BFS with edge lengths?

Graph: \( G = (V, E) \).
Length of \( e \): \( l_e \).

Find shortest paths from \( s \).

Make \( G' \) from \( G \).
For each edge \( e \)
Replace w/len \( l_e \) path.

Run BFS on \( G' \).
BFS in exploded graph.

Correct: follows from the correctness of BFS.

Time? $O(\sum_e l_e)$.

Very very large.

Compared to size of problem.

Size of representation: 6 digits plus one edge. 10 ish.

Time: 1,000,000. ...ish.

Hmmm...

The distance is ..obviously 1,000,000.

Could it be easier?
Looking again.

Queue next node on long edges again and again...

Nothing interesting until 20 steps.

Wake me then!
Alarms.

\[
\begin{align*}
\text{d}(A) &= 0 \\
\text{d}(B) &= 20 \\
\text{d}(C) &= 20 \\
\text{d}(D) &= 23
\end{align*}
\]

Process A: \( \text{d}(A) = 0 \)
For Edge \((A, C)\): Set \( \text{d}(C) = \text{d}(A) + 20 = 20 \) and alarm for 20.
For Edge \((A, B)\): Set \( \text{d}(B) = \text{d}(A) + 30 = 30 \) and alarm for 30.

At time 20:
Process C.
For Edge \((C, D)\): Set \( \text{d}(D) = \text{d}(C) + 3 = 23 \) and alarm for 23.
For Edge \((C, B)\): Reset \( \text{d}(B) = 24 \) and alarm for 24.

At time 23:
Process D. Set \( \text{d}(D) = 23 \).
For Edge \((D, B)\):
\[
\text{d}(D) = 24 \text{ which is less than } \text{d}(D) + 3 = 26 \text{ so leave it.}
\]

At time 24: Process B. Done.
..what needed to be done?

Set a distance. Set an alarm.

d(A) = 0

\[ d(C) = 20 \quad d(D) = 23 \]

\[ d(B) = 20 \]
Alarm Algorithm.

Set an alarm clock for node $s$ at time 0.
Repeat until there are no more alarms:
Next alarm goes off at time $T$, for node $u$. Then:
- The distance from $s$ to $u$ is $T$.
- For each neighbor $v$ of $u$ in $G$:
  * If no alarm for $v$, set alarm for $T + l(u, v)$.
  * If $v$’s alarm is $\geq T + l(u, v)$
    then reset it $T + l(u, v)$.

Implementation:
Need to maintain alarm for each node.
Possibly need to decrease alarm for a node.
Find next alarm time.

Insert: $(v, key)$
DecreaseKey: $(v, newkey)$
DeleteMin: $Q$ returns $v$ with min. key
Dijkstra’s Algorithm.

\[
\text{foreach } v: \quad d(v) = \infty.
\]
\[
d(s) = 0.
\]
\[
\text{Q.Insert}(s, 0)
\]
\[
\textbf{While} \quad u = \text{Q.DeleteMin}():
\]
  \[
  \text{foreach edge } (u, v):
  \]
  \[
  \quad \text{if } d(v) > d(u) + l(u, v):
  \]
  \[
  \quad \quad d(v) = d(u) + l(u, v)
  \]
  \[
  \quad \text{Q.InsertOrDecreaseKey}(v, d(v))
  \]

**Runtime:**

\[
|V| \text{ DeleteMins.}
\]
\[
|V| \text{ Inserts.}
\]
\[
\leq |E| \text{ DecreaseKeys.}
\]

Binary heap: \(O((|V| + |E|) \log |V|))\)
Binary Heap.

Heap\(^1\): bigger children.  
\[\implies\] smallest at root.

![Binary Heap Diagram]

Insert(7): Bubble up: check parent. \(\text{depth}\) comp.  
DeleteMin: Replace. Bubble down: check both children.  
\(2 \times \text{depth}\) – comparisons.

\(^1\)values only
**d-ary heap**

Degree – $d$, Depth – $\log_d n$.
Insert/DecreaseKey – $\log n / \log d$.
DeleteMin – $d \log n / \log d$. (Check all children.)

Dijkstra:
$O(|V|)$ deletemins. $O(d \log n / \log d)$ each.
$O(|E|)$ insert/decrease-keys. $O(\log n / \log d)$ each.

$O(|V|d \log n / \log d + |E| \log n / \log d)$.

Optimal Choice: Choose $d = |E| / |V|$ (average degree/2)
$O(|E| \log n / \log d)$

For dense graphs it approaches linear.

Fibonacci Heaps:
$O(\log n)$ per delete.
$O(1)$ average decrease-key.

$O(|V| \log |V| + |E|)$.
Linear for moderately dense graphs!
Lecture in a minute.

Breadth First Search of graph:
Search with queue instead of stack.
Get “distances” from source.
Proof idea: queue has distance 0, then level 1, …

Dijkstra: shortest paths in weighted graph:
Replace weights by paths + BFS
Implement using priority queue.
Idea: ignores “new” nodes.
Runtime: $|V|$ extracts, $|E|$ reduce-key for p-queue.

Priority Queue:
Implementation: degree $d$ tree.
Heap Property: children larger than parent.
Minimum at top.
Remove min: $O(d \log_d n)$ time: Replace min/percolate down.
Reduce Key: $O(\log_d n)$: percolate up.