Lecture in a minute.

Breadth First Search of graph:
Search with queue instead of stack.
Get “distances” from source.

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

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

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.

Prove queue property.

**Base:** $s$ is explored first.

$$=\quad \Rightarrow \text{all its neighbors in queue}$$

$$=\quad \Rightarrow \text{queue has all distance 1 nodes}.$$ 

$$d + 1 \text{ node has a distance } d \text{ neighbor def of distance last distance } d \text{ node explored (queue!)}$$

$$=\quad \Rightarrow \text{every node at distance } d + 1 \text{ is visited}$$

$$=\quad \Rightarrow \text{queue has all dist. } d + 1 \text{ nodes}.$$ 

Has no $l > d + 1$ nodes since only $\leq d$ level nodes explored.

Has no $l < d$ nodes, since removed by induction.

$$=\quad \Rightarrow \text{only distance } d + 1 \text{ nodes}.$$ 

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.

$$A \quad 1,000,000 \quad B$$

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?

In action!

**Breadth First Search:**

- $d(s) = 0$
- $\text{visited}[s] = \text{true}$
- put $S$ in $Q$

While $u = Q$:

- $\text{foreach } (u,v)$:
  - if $\text{visited}[v] = \text{false}$:
    - $\text{visited}[v] = \text{true}$
    - $d[v] = d[u] + 1$
    - put $v$ in $Q$

Queue: $\text{SEE,DE,D,A,CD,A,CA,CCC,BB}$

Queue: $011,11,1,1,11,1,11,111,22$

S

E D A C

S EE DD A

B

A CC

BB

Edge lengths: BFS?

Sacramento

Reno

San Francisco

Bakersfield

Los Angeles

Las Vegas

95

133

445

275

409

112

291

290

271

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 $w$ w/en $l_e$ path.

Run BFS on $G'$.

Looking again.

A

20

30

C

3

D

3

B

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

Nothing interesting until 20 steps.

Wake me then!
**Dijkstra’s Algorithm.**

- **foreach** \( v \): \( d(v) = \infty \).
- \( d(s) = 0 \).
- \( Q.\text{insert}(s,0) \).
- While \( u = Q.\text{deleteMin}() \):
  - **foreach** edge \((u,v)\):
    - If \( d(v) > d(u) + l(u,v) \):
      - \( d(v) = d(u) + l(u,v) \)
      - \( Q.\text{insert}() \) or \( Q.\text{decreaseKey}() \).

**Runtime:**
- \( |V| \) DeleteMins.
- \( |V| \) Inserts.
- \( \leq |E| \) DecreaseKeys.

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

---

**Binary Heap.**

- **Heap** \(^1\): bigger children. \( \Rightarrow \) smallest at root.
- **Insert**(7): Bubble up: check parent. \( \Rightarrow \) depth comp.
- **DeleteMin**: Replace. Bubble down: check both children. \( 2 \times \text{depth} \) – comparisons.

**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 \( T \); find 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 \( T + l(u,v) \) 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

**d-ary heap**

- **Degree** – \( d \);
- **Depth** – \( \log_d n \).
- Insert/DecreaseKey – \( \log n / \log d \).
- DeleteMin – \( \log n / \log d \). (Check all children.)
- **Dijkstra**: \( O(|V|) \) deletions. \( O(d \log n / \log d) \) 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.
- \( O(|V| \log n / |V| + |E|) \).
- 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.