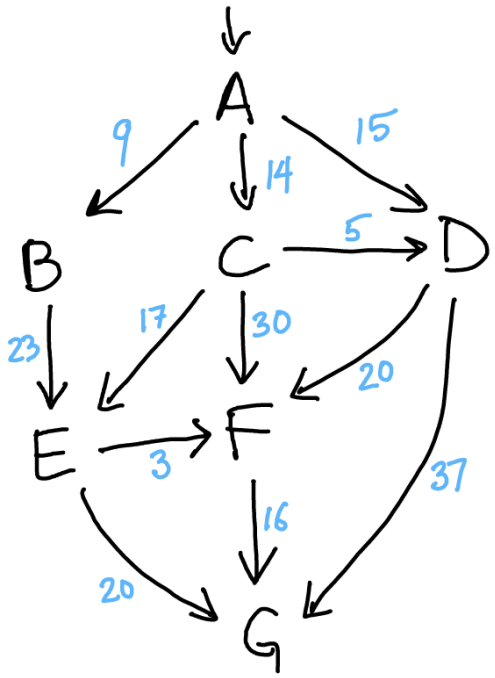
Single-Source Shortest Paths (Dijkstra's Algorithm)

A7 release today — start soon!

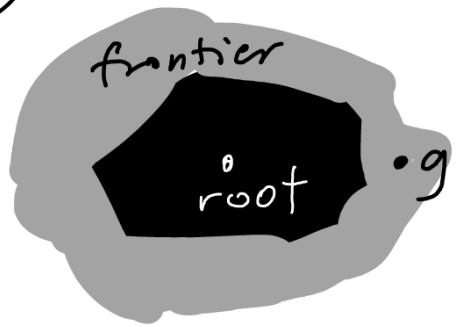


- shortest path in unweighted graph? BFS
- weighted edges ⇒ minimize sum of weights = distance
- Dijkstra's algorithm
  - greedy algorithm: "naive" choice works
- Input: directed graph with nonnegative edge weights
- Output: minimum path distance to all nodes from root
  ⇒ gives path too.
- Applications — weights = distance, time, cost, probabilities, ...

BFS

frontier = FIFO queue

- set all distances to ∞
- root.dist = 0
  frontier.push(root)
- while (!frontier.empty()) {
  g = frontier.pop();
  foreach (g → v) {
  if (v.distance == ∞) {
  frontier.push(v);
  v.dist = g.dist + 1;
  }
  }



undiscovered.

invariant: no black → white edges

→ Dijkstra's: use priority queue instead of FIFO.

extractMin

Priority queue: pop() removes highest-priority element.
min-queue: highest-priority = minimum value

v.dist = shortest distance to v so far

white : v.dist = ∞

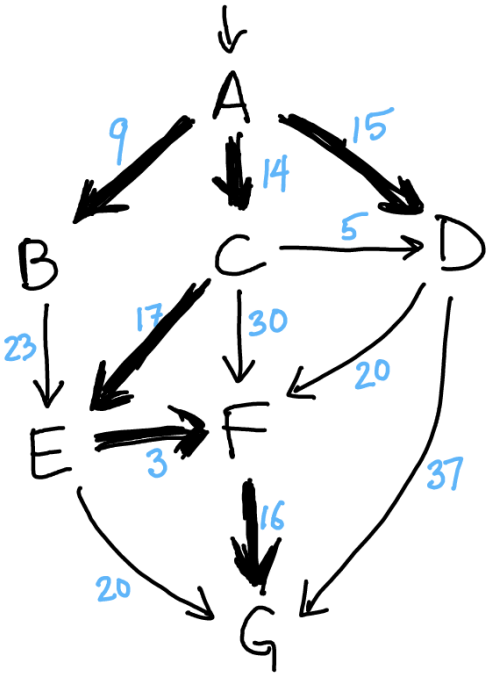
gray : v.dist < ∞, v ∈ frontier

black : v.dist < ∞, v ∉ frontier

Algorithm

```
set all distances to ∞
root.dist = 0
frontier.push(root)
while (!frontier.empty()) {
    Node g = frontier.pop();   ← greedy
    foreach (g --d--> v) {
        if (v.dist == ∞) {
            v.dist = g.dist + d;
            frontier.push(v);
        } else {
            v.dist = min(v.dist, g.dist + d);   // note: affects priority of v.
        }
    }
}
```

priority = dist



V	Finished	v.dist
A	✓	0
B	✓	9
C	✓	14
D	✓	15
E	✓	~~32~~ 31
F	✓	~~44~~ ~~35~~ 34
G	✓	~~52~~ ~~51~~ 50

Edge used for final distance update lies on best path, form a tree

⇒ record best edge to each vertex, work backward from target node

Run time

Outer loop: once per vertex

Finding min element in priority queue: $O(\lg V)$ (b/c sorting)

$\Rightarrow$ outer loop is $O(V \lg V)$

Inner loop: $O(E)$ iterations

Update distance in priority queue?

- $O(\lg V)$ with BST or binary heap
- $O(1)$ with Fibonacci heap ← bad constant factors

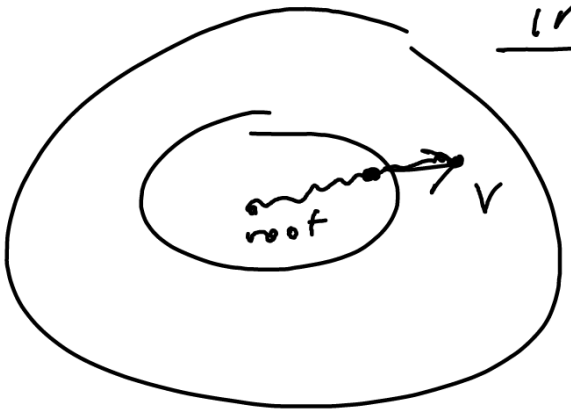
Total: $O(V \lg V + E)$

or: $O(E \lg V)$ with simpler data structures

— just as good for sparse graph.

Correctness

loop invariant = tricolor (black ↛ white) + ...



interior path to v : path from root to v only via black nodes (exc. v)

invariant :
① For every black v, gray g
v.dist ≤ g.dist
black ≤ gray
② v.dist is shortest interior path₁ to v, or ∞ if v white
distance

1. Establishment
   ① no black node! ✓
   ② Interior paths = (root) ✓
2. Postcondition
   ① No gray nodes ✓
   ② all paths are interior paths
   ⇒ v.dist is shortest path distance.

Preservation

① $\text{black}' \leq \text{gray}'$

change: g becomes black.

g is min gray node

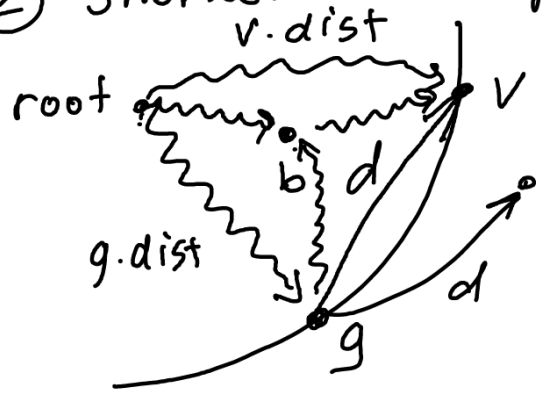
$\text{black} \leq g \leq \text{gray}$

$\text{black}' \leq \text{gray}'$

change #2: v becomes gray

$d \geq 0 \Rightarrow v.dist \geq g.dist$ ✓

② Shortest interior paths?



New shorter i.p. to ... ?

- To g? No.
- To black v? No, $g.dist \geq v.dist$
- To white v? tricolor inv. $\Rightarrow$ only i.p. is via g. $v.dist = g.dist + d$ ✓
- To gray v? 3 paths:
  - existing ($v.dist$)
  - new via $g \xrightarrow{d} v$
  - via some black b? $b.dist \leq g.dist$, so $v.dist$ path must be at least as good.

Invariant

① For every black v, gray g
$v.dist \leq g.dist$
$\text{black} \leq \text{gray}$

② $v.dist$ is shortest interior path distance to v, or ∞ if v white

Code

```
while (!frontier.empty()) {
    Node g = frontier.pop();   ← greedy
    foreach (g --d--> v) {
        if (v.dist == ∞) {
            v.dist = g.dist + d;
            frontier.push(v);
        } else {
            v.dist = min(v.dist, g.dist+d);
        }
    }
}
```