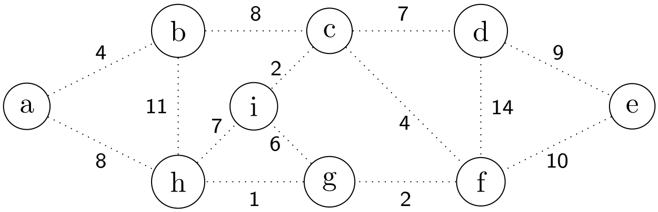
CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich[1]

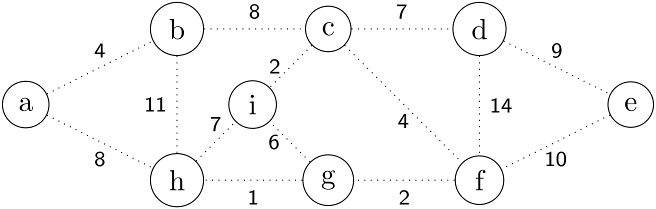
[1] based on notes by Anna Bretscher and Albert Lai

finding the shortest paths



- Given an (edge-)weighted graph and two vertices in it,
- find the cheapest (minimum possible weight) path between them, or
- report that one does not exist.

finding the shortest paths



Even better:

- Given an (edge-)weighted graph and a vertex s in it,
- find the cheapest (minimum possible weight) paths from s to all other vertices.

Dijkstra's algorithm: idea

Dijkstra's algorithm finds shortest paths by a BFS with a twist

- the queue is replaced with a minimum priority queue
- with an additional operation `decrease-priority(vertex, new-priority)`

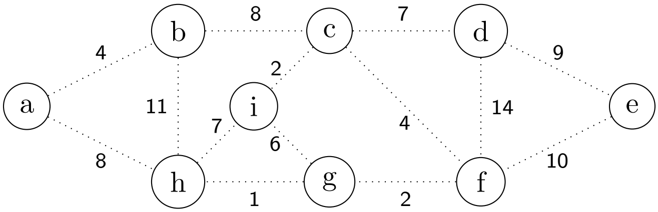
Keep unvisited vertices in the priority queue:

$$priority(v) = distance(start, v) \text{ via finished vertices only}$$
$$priority(v) = \infty \text{ if no such path}$$

The algorithm grows paths by one edge at a time.

Correctness idea: every time we `extract-min`, we get the next vertex closest to `start`.

Dijkstra's algorithm: example



Priority queue contains vertices *not* in tree:

vertex	a	b	c	d	e	f	g	h	i
priority	0	∞	∞	∞	∞	∞	∞	∞	∞
pred									

Distance tree: (X, Y, N) represents total distance to get to vertex Y with predecessor X is N

Dijkstra's algorithm

```
PQ := new min-heap()
PQ.insert(0, start)
start.d := 0
for each vertex v != start:
  PQ.insert(inf, v)
  v.d := inf
while not PQ.is-empty():
  u := PQ.extract-min()
  for each v in u's adjacency list, v in PQ:
    d' := u.d + weight(u, v)
    if d' < v.d:
      PQ.decrease-priority(v, d')
      v.d := d'
      v.pred := u
```

Dijkstra's algorithm: time

Let $n = |V|$ and $m = |E|$. Then:

- every vertex enters and leaves min-heap once
  - $O(\log n) \times n$ vertices $= O(n \log n)$
  -
- with every edge may call `decrease-priority`
  - $O(\log n) \times m$ edges $= O(m \log n)$
- the rest can be done in $\Theta(1)$ per vertex or per edge

Total time worst case: $O[(m+n)\log(n)]$

Dijkstra's algorithm: proof

Let s: start vertex

- $\delta(v)$ be the weight of the shortest path from start vertex s to v,
- $\delta_{fin}(v)$ be the weight of the shortest path from start vertex s to v among paths via finished vertices only (not in PQ), and
- $p(v)$ be priority of v.

Dijkstra's algorithm maintains the loop invariants:

1. for each v in PQ, $p(v) = v.d = \delta_{fin}(v)$, i.e. considering only paths via finished vertices (vertices not in PQ),
2. for each v not in PQ, $v.d = \delta(v)$ over all paths, and $v.pred$ is the vertex before v on the shortest path.

Dijkstra's algorithm: proof

Initially (after lines 0-5):

- PQ contains all of V,
- $s.d = p(s) = 0$, and
- $v.d = p(v) = \infty$, for all $v \neq s$

so (1) and (2) are true.

Dijkstra's algorithm: proof

Suppose (1) and (2) are true on line 6.

Choose arbitrary $v \in PQ$, $u \notin PQ$ just dequeued from PQ

Case 1: v not adjacent to u, no change to $p(v)$, $\delta(v)$, $\delta_{fin}(v)$

Case 2: v adjacent to u:

1. d' new min from s to v, no change to $p(v)$, $\delta(v)$, $\delta_{fin}(v)$
2. d' not new min from s to v, update v to ensure (1)

Dijkstra's algorithm: proof

(cont.)

- If v not adjacent to u:
  - Can we have a shorter path to v via finished vertices that looks like:
    $s \to ... \to x \to u \to y \to ... \to v$?
  - No, because y is finished, so path from s to y must have been shortest.
  - So no change means (1) still true after line (13)

Dijkstra's algorithm: proof

Now to show $u.d = \delta(u)$.