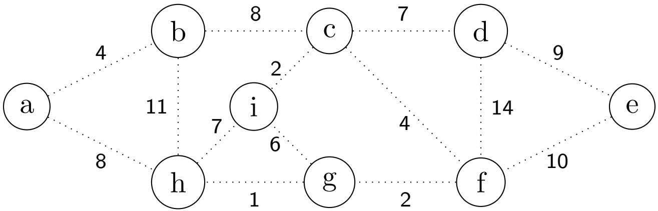
CSCB63 – Design and Analysis of Data Structures

Akshay Arun Bapat[1]

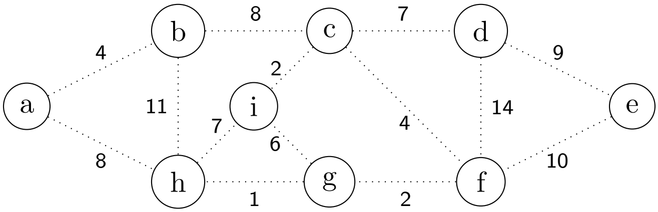
[1] based on notes by Anya Tafliovich, Anna Bretscher and Albert Lai

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)`

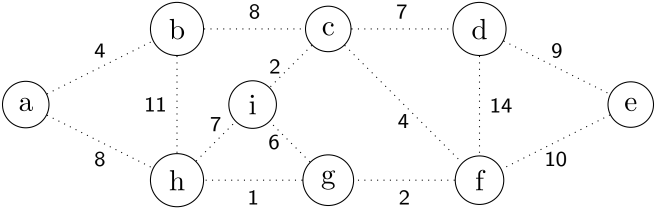
Keep unvisited vertices in the priority queue:

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

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



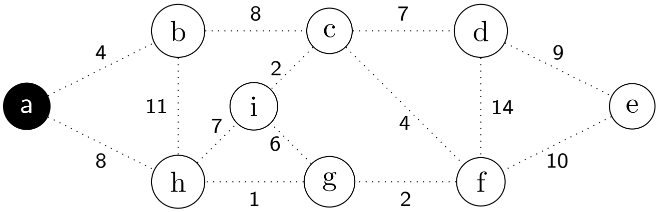
Priority queue contains vertices *not* in tree:

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

Distance tree:

$\{ \qquad \}$

Dijkstra's algorithm: example



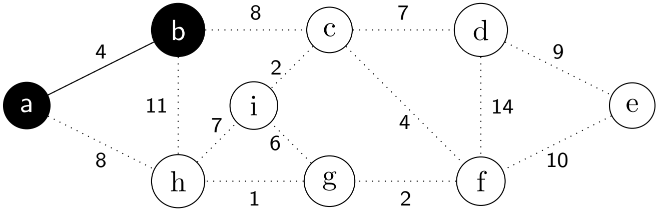
Priority queue contains vertices *not* in tree:

vertex	b	h	c	d	e	f	g	i
priority	4	8	∞	∞	∞	∞	∞	∞
pred	a	a						

Distance tree:

$\{ \quad \}$

Dijkstra's algorithm: example



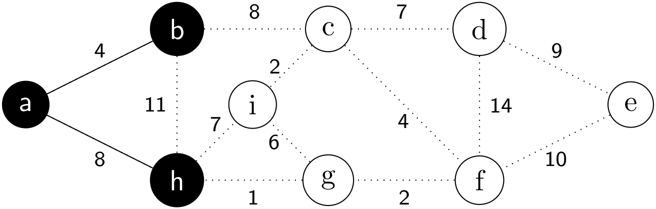
Priority queue contains vertices *not* in tree:

vertex	h	c	d	e	f	g	i
priority	8	12	∞	∞	∞	∞	∞
pred	a	b					

Distance tree:

{ (a,b,4), }

Dijkstra's algorithm: example

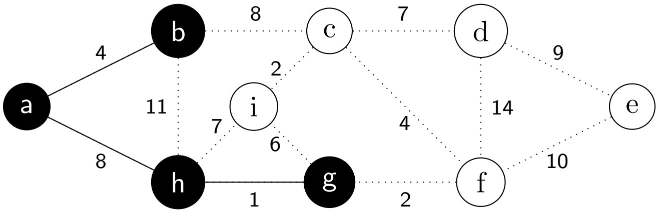


Priority queue contains vertices *not* in tree:

vertex	g	c	i	d	e	f
priority	9	12	15	∞	∞	∞
pred	h	b	h			

Distance tree:
{ (a,b,4), (a,h,8), }

Dijkstra's algorithm: example



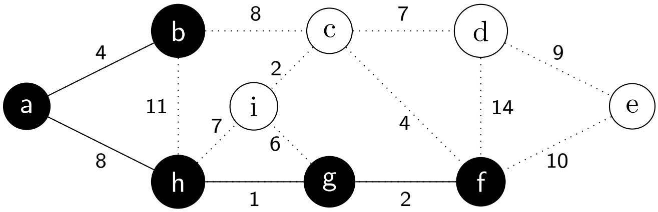
Priority queue contains vertices *not* in tree:

vertex	f	c	i	d	e
priority	11	12	15	∞	∞
pred	g	b	h		

Distance tree:

{ (a,b,4), (a,h,8), (h,g,9), }

Dijkstra's algorithm: example



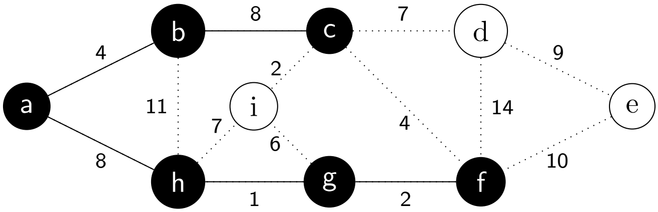
Priority queue contains vertices *not* in tree:

vertex	c	i	e	d
priority	12	15	21	25
pred	b	h	f	f

Distance tree:

{ (a,b,4), (a,h,8), (h,g,9), (g,f,11), }

Dijkstra's algorithm: example



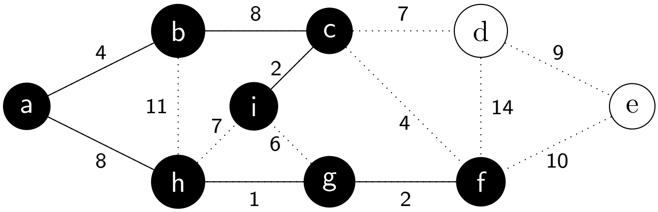
Priority queue contains vertices *not* in tree:

vertex	i	d	e
priority	14	19	21
pred	c	c	f

Distance tree:

{ (a,b,4), (a,h,8), (h,g,9), (g,f,11), (b,c,12), }

Dijkstra's algorithm: example



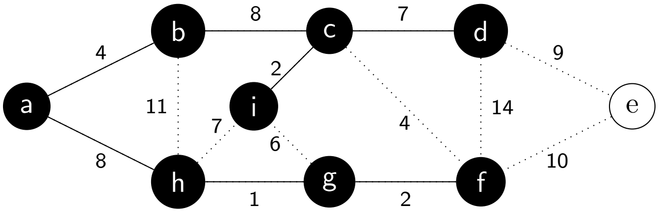
Priority queue contains vertices *not* in tree:

vertex	d	e
priority	19	21
pred	c	f

Distance tree:

{ (a,b,4), (a,h,8), (h,g,9), (g,f,11), (b,c,12), (c,i,14), }

Dijkstra's algorithm: example

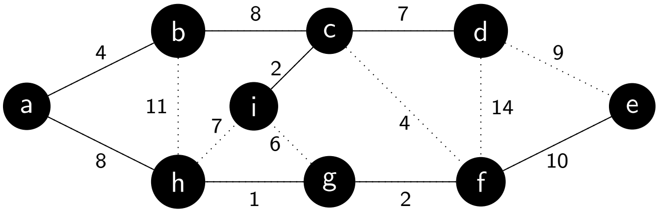


Priority queue contains vertices *not* in tree:

vertex	e
priority	21
pred	f

Distance tree:
{ (a,b,4), (a,h,8), (h,g,9), (g,f,11), (b,c,12), (c,i,14), (c,d,19), }

Dijkstra's algorithm: example



Priority queue contains vertices *not* in tree:

Distance tree:
{ (a,b,4), (a,h,8), (h,g,9), (g,f,11), (b,c,12), (c,i,14), (c,d,19), (f,e,21) }

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
  - enters in the beginning only; continue until heap is empty
  - $\mathcal{O}(\log n)$ each, for a total of $\mathcal{O}(n \log n)$
- with every edge may call `decrease-priority`
  - $\mathcal{O}(\log n)$ each, for a total of $\mathcal{O}(m \log n)$
- the rest can be done in $\Theta(1)$ per vertex or per edge

Total time worst case: $\mathcal{O}((n + m) \log n)$

Dijkstra's algorithm: proof

Let

- $\delta(v)$ 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 (only paths via 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.

- Line 7 adds a new finished vertex u (moves from $u \in PQ$ to $u \notin PQ$).
- Before line 7 we had $p(u) = u.d = \delta_{fin}(u)$.
- Take artibtrary vertex $v \in PQ$.
  Before line 7 we had $p(v) = v.d = \delta_{fin}(v)$.
- If v adjacent to u:
  - look at the path $p_v = p_u + (u, v)$ where p_u is shortest via finished vertices to u
  - have
    $w(p_v) = w(p_u) + w(u, v) = \delta_{fin}(u) + w(u, v) = u.d + w(u, v)$
  - if $w(p_v) < v.d$ then it is the shortest via finished vertices to v
  - then condition on line 10 is true and we set
    $p(v) = v.d = w(p_v) = \delta_{fin}(v)$
  - otherwise, no change
  - so (1) is still true after line 13

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 $u.d = \delta(u)$.

- consider the time just before u is dequeued on line 7
- there is some (overall) shortest path p_u from s to u
- at some point p_u crosses from $V - PQ$ (finished vertices) to PQ (not finished vertices) for the first time via some edge (x, y) with $x \notin PQ$ and $y \in PQ$

$$p_u = \underbrace{s \to \ldots \to x \to y}_{p_y} \to \ldots \to u$$

- have $w(p_y) = \delta(y) = \delta_{fin}(y) = y.d = p(y)$ from (1)
- have both $u, y \in PQ$ and u dequeued first, so $p(u) \leq p(y)$
- then $u.d \leq y.d = \delta(y) \leq \delta(u)$ (p_u has added edges)
- $\therefore u.d = \delta(u)$