Graph Algorithms

Shortest Paths

Michael Lampis

September 20, 2025

Shortest Path Problems

Input: **Edge-weighted** (di)graph

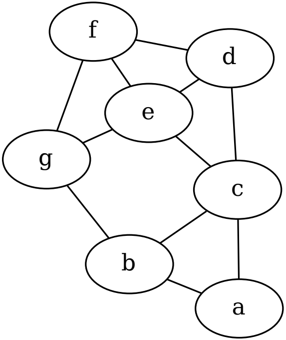
Shortest Path Problems

Input: **Edge-weighted** (di)graph

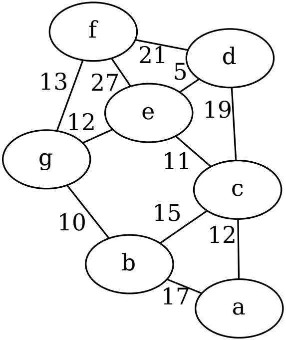
Problems:

- Single-Pair Shortest Path (SPSP): find shortest path from s to t.
- Single-Source Shortest Path (SSSP): find shortest path from s to everyone.
- All-Pairs Shortest Paths: find shortest paths from everyone to everyone.

Example

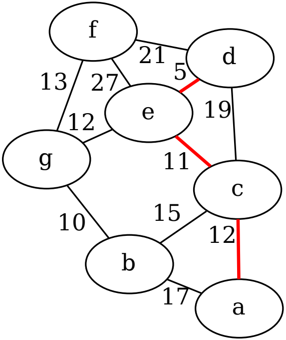


Example



Find shortest path from a to d.

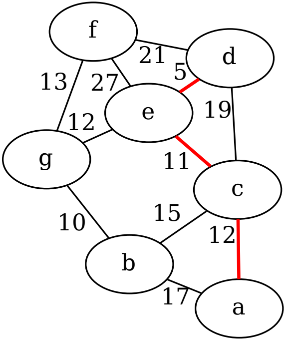
Example



Find shortest path from a to d.

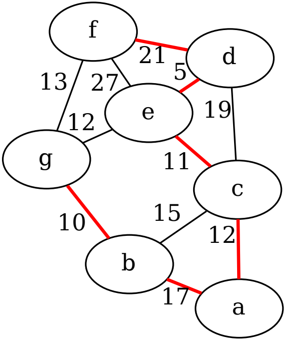
$a \to c \to e \to d$ (cost$= 28$)

Example



Find shortest path from a to everyone

Example

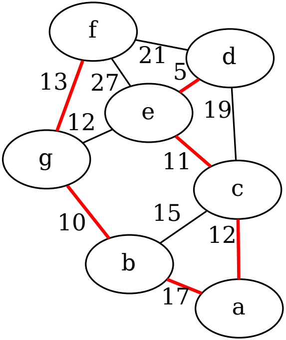


Find shortest path from a to everyone

Costs:

	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27

Example

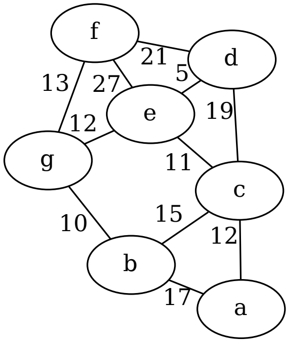


Find shortest path from *a* to everyone

Costs:

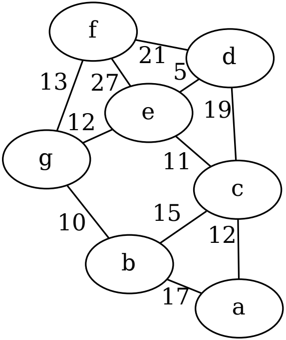
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27

Example



Find all shortest paths

Example

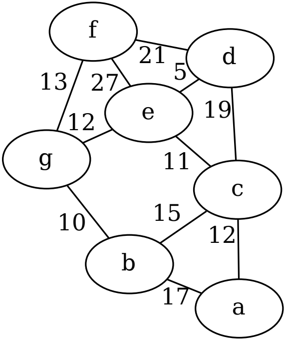


Find all shortest paths

Costs:

	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b	17	0	15	27	22	23	10
c	12	15	0	16	11	36	23
d	28	27	16	0	5	21	17
e	23	22	11	5	0	27	12
f	40	23	36	21	27	0	13
g	27	10	23	17	12	13	0

Example

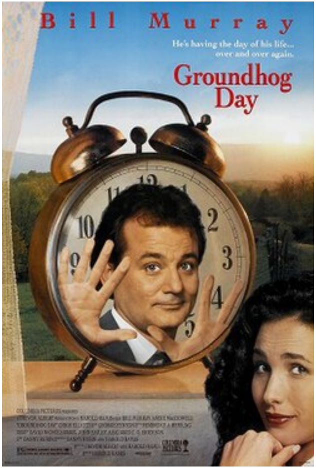


Find all shortest paths

Costs:

	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b	17	0	15	27	22	23	10
c	12	15	0	16	11	36	23
d	28	27	16	0	5	21	17
e	23	22	11	5	0	25	12
f	40	23	36	21	25	0	13
g	27	10	23	17	12	13	0

Why not BFS?

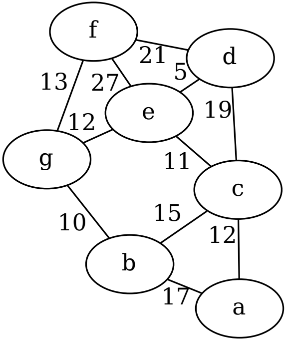

Bill Murray
He's having the day of his life...
over and over again.
Groundhog
Day

Why not BFS?

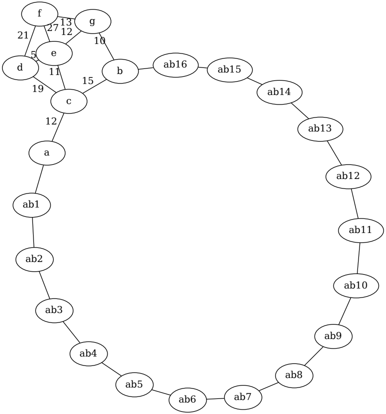
We could use BFS:

- Handle edges of weight w by sub-dividing them w times.
- But then complexity goes from $O(m+n)$ to $O(n+mW)$, where W is max weight.
    - This algorithm is **exponential**-time in the input size!!
- Goal: attain complexity polynomial in $n, m, \log W$.

Why not BFS?

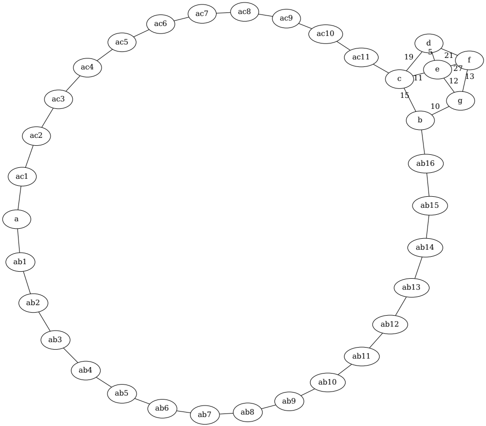


Why not BFS?



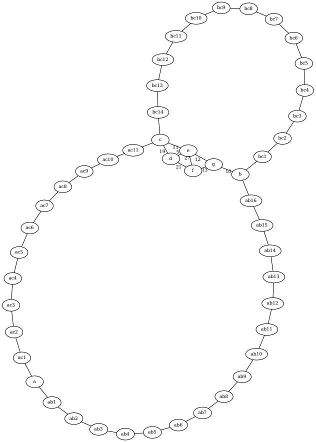
f
g
13
27
12
21
10
e
5
d
b
ab16
ab15
11
15
19
ab14
c
12
ab13
a
ab12
ab1
ab11
ab2
ab10
ab3
ab9
ab4
ab8
ab5
ab7
ab6

Why not BFS?



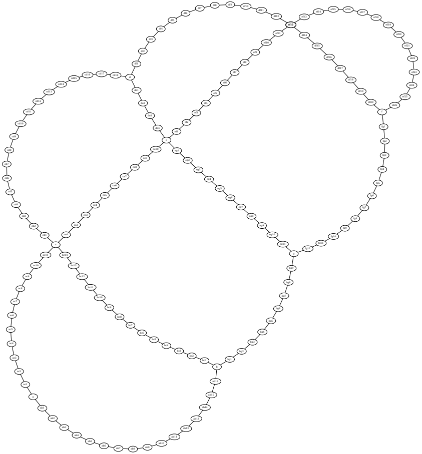
ac7
ac8
ac6
ac9
ac5
ac10
d
ac11
19
5
21
f
ac4
e
27
c
11
13
12
ac3
15
g
10
b
ac2
ab16
ac1
ab15
a
ab14
ab1
ab13
ab2
ab12
ab3
ab11
ab4
ab10
ab5
ab9
ab6
ab8
ab7

Why not BFS?

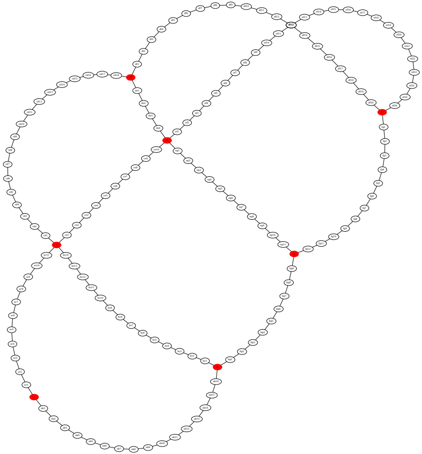


bc9
bc8
bc10
bc7
bc11
bc6
bc12
bc5
bc13
bc4
bc14
bc3
bc2
c
ac11
11
e
19
5
d
27
12
g
bc1
ac10
21
f
13
10
b
ac9
ac8
ab16
ac7
ac6
ab15
ac5
ab14
ac4
ab13
ac3
ab12
ac2
ab11
ac1
ab10
a
ab9
ab8
ab1
ab7
ab2
ab6
ab3
ab4
ab5

Why not BFS?



Why not BFS?



Running BFS on this is a terrible idea!!

Dijsktra's algorithm

Dijkstra's algorithm

Dijsktra's algorithm for SSSP

- Works for instances with positive edge weights.
- Computes shortest path tree from given source s.
- Main idea: **Greedy**. Consider closer vertices first.
  - (in a sense, similar in spirit to BFS)
- Directed or Undirected Graphs.
- Complexity: $O((n+m)\log n)$ time and $O(n)$ space
  - Above depends slightly on some assumptions and data structures, see later.
  - Also assume arithmetic operations take $O(1)$, otherwise we must add a $O(\log W)$ factor, where W is maximum weight.

Basic Setup

- Vertices are White, Gray, or Black
  - White vertex $\rightarrow$ no path found.
  - Gray vertex $\rightarrow$ some path found.
  - Black vertex $\rightarrow$ shortest path found.
- We maintain current Distances and Tree
  - Update each time we find a **shortcut**.
- We maintain a **Priority** Queue
  - Queue contains Gray vertices.
  - Prioritize by **min** distance from source.

Priority Queue – Abstract View

Basic Properties:

- Queue contains a collection of (key, values) pairs.
  - For us: key $\rightarrow$ vertex, value $\rightarrow$ distance.
- Operations:
  - Insert(k,v)
  - Extract $\rightarrow$ k
    - **Caution:** returns k with **min** value in queue! Not FIFO!
  - Decrease(k,v')
    - Updates value of key k from v to v'.
    - **Edge case:** if k not present, calls Insert(k,v')

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (b,3), (c,8), (d, 1), (e, 2)
Output:

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (b,3), (c,8), (e, 2)
Output:
d

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (b,3), (c,2), (e, 2)
Output:
d

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (b,3), (e, 2)
Output:
d, c

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (b,3), (e, 2), (f, 7)
Output:
d, c

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (b,3), (f, 7)
Output:
d, c, e

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
(a,5), (f, 7)
Output:
d, c, e, b

Priority Queue – Example

Operations:

1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:

(f, 7)

Output:

d, c, e, b, a

Priority Queue – Example

Operations:

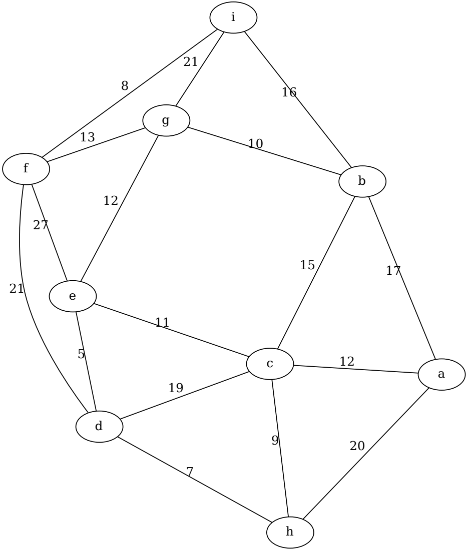
1. Insert (a,5), (b,3), (c,8), (d, 1), (e, 2)
2. Extract
3. Decrease(c,2)
4. Extract
5. Insert (f,7)
6. Extract x4

Queue state:
$\emptyset$
Output:
d, c, e, b, a, f

Dijkstra's algorithm

```
Initialize (dist[1...n] = ∞, parent[1...n] ← NULL, etc. )
Initialize Q ← ∅
Q.Insert(s,0)
while Q not empty do
    u ← Q.Extract()
    for v ∈ N⁺(u) do
        if dist[v] > dist[u] + w(u, v) then          ▷ Shortcut found
            Parent[v] ← u
            dist[v] ← dist[u] + w(u, v)
            Q.Decrease(v,dist[v])
        end if
    end for
end while
```

Example



Distances:

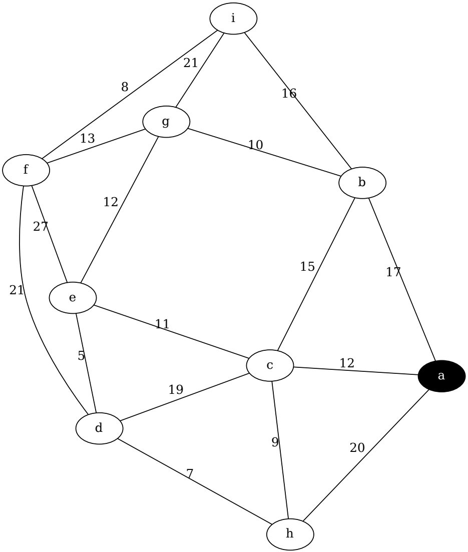
a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(0, a)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

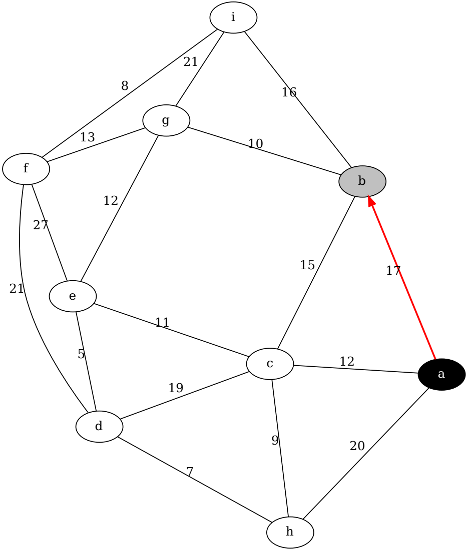
a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
$(0, a)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

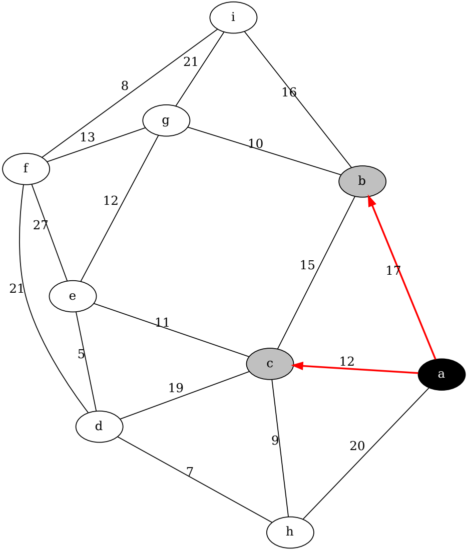
a	0
b	17
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(17, b)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

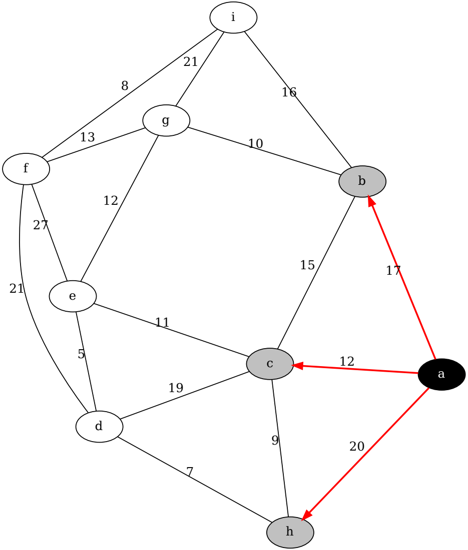
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(12, c), (17, b)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

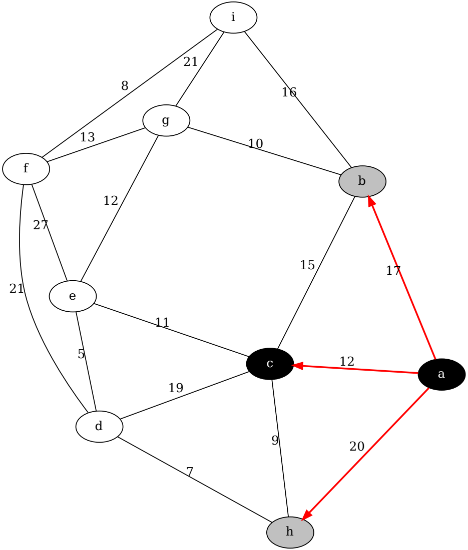
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
$(12, c), (17, b), (20, h)$

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

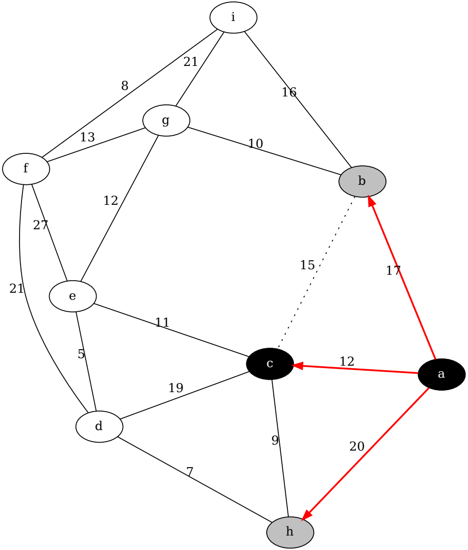
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
(17, b), (20, h)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



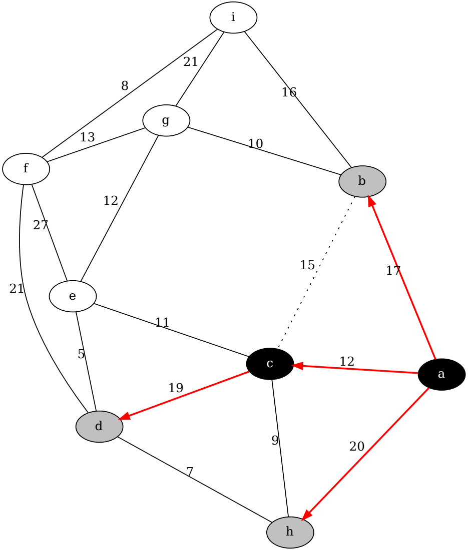
Distances:	
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
(17, *b*), (20, *h*)

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



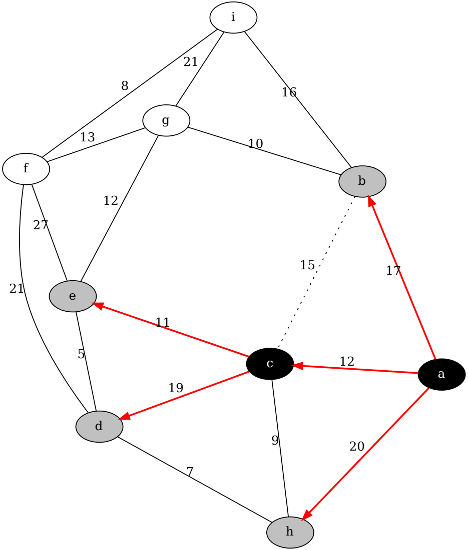
Distances:	
a	0
b	17
c	12
d	31
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
(17, b), (20, h), (31, d)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



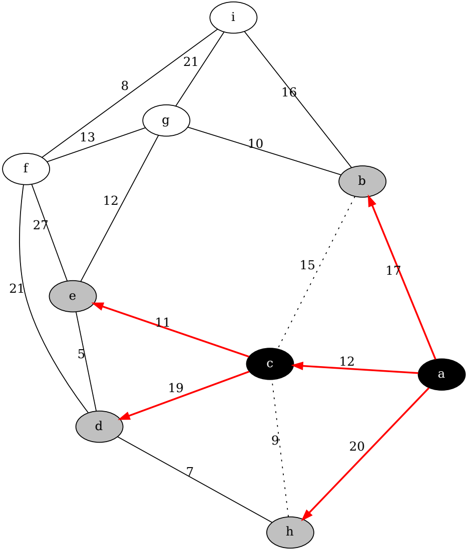
Distances:	
a	0
b	17
c	12
d	31
e	23
f	∞
g	∞
h	20
i	∞

Priority Queue:
$(17, b), (20, h), (23, e), (31, d)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

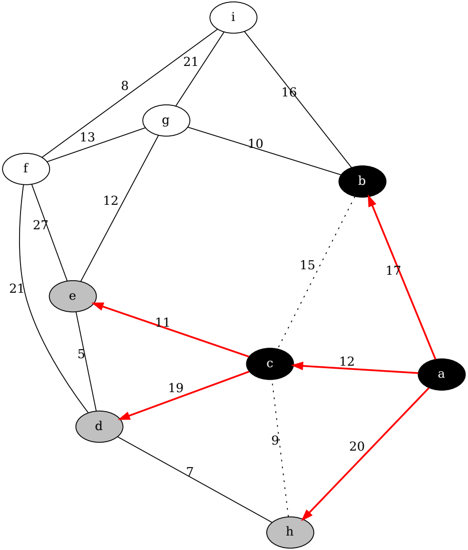
a	0
b	17
c	12
d	31
e	23
f	∞
g	∞
h	20
i	∞

Priority Queue:
$(17, b), (20, h), (23, e), (31, d)$

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

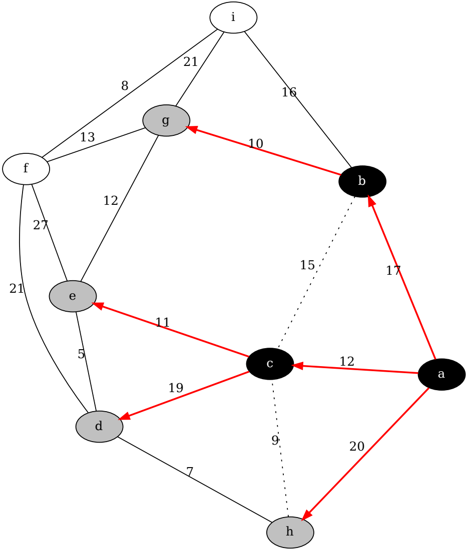
a	0
b	17
c	12
d	31
e	23
f	∞
g	∞
h	20
i	∞

Priority Queue:
(20, h), (23, e), (31, d)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

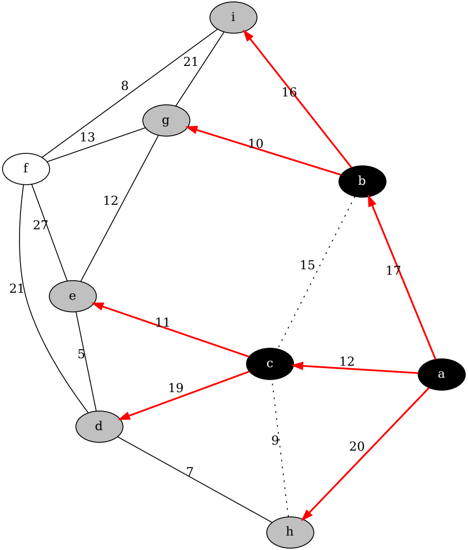
a	0
b	17
c	12
d	31
e	23
f	∞
g	27
h	20
i	∞

Priority Queue:
(20, h), (23, e), (27, g), (31, d)

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

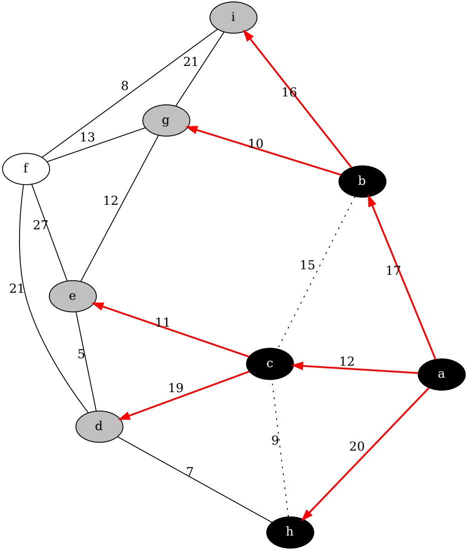
a	0
b	17
c	12
d	31
e	23
f	∞
g	27
h	20
i	33

Priority Queue:
$(20, h), (23, e), (27, g), (31, d), (33, i)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



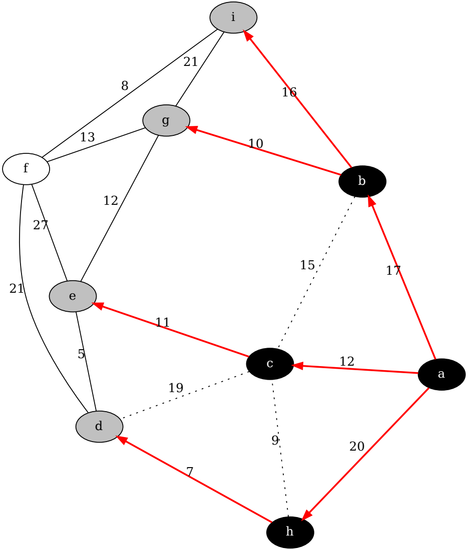
Distances:	
a	0
b	17
c	12
d	31
e	23
f	∞
g	27
h	20
i	33

Priority Queue:
(23, e), (27, g), (31, d), (33, i)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

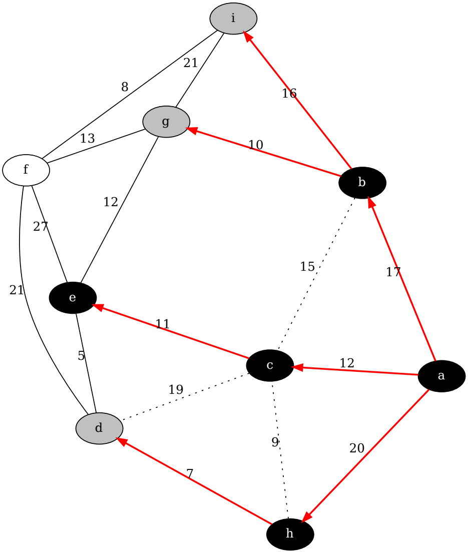
a	0
b	17
c	12
d	27
e	23
f	∞
g	27
h	20
i	33

Priority Queue:
(23, e), (27, d), (27, g), (33, i)

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

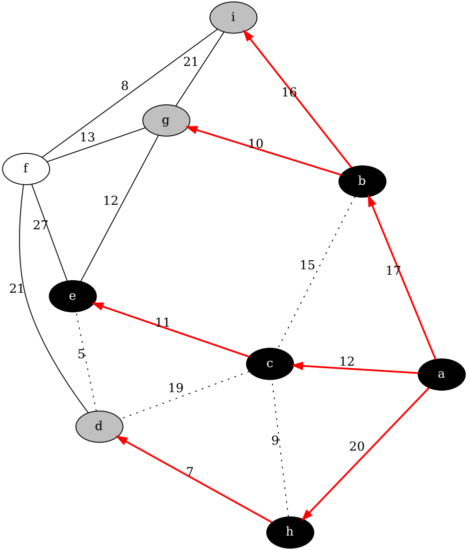
a	0
b	17
c	12
d	27
e	23
f	∞
g	27
h	20
i	33

Priority Queue:
$(27, d), (27, g), (33, i)$

Legend:

- White vertex $\to$ no path found.
- Gray vertex $\to$ some path found.
- Black vertex $\to$ shortest path found.
- Black edge $\to$ not yet considered.
- Red edge $\to$ shortest path edge.
- Dotted edge $\to$ non-optimal edge.

Example



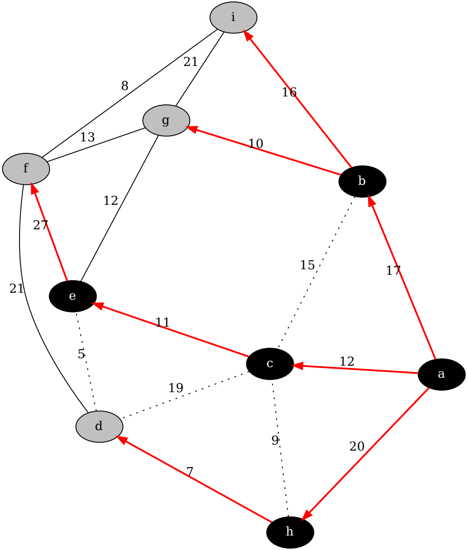
Distances:	
a	0
b	17
c	12
d	27
e	23
f	∞
g	27
h	20
i	33

Priority Queue:
$(27, d), (27, g), (33, i)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

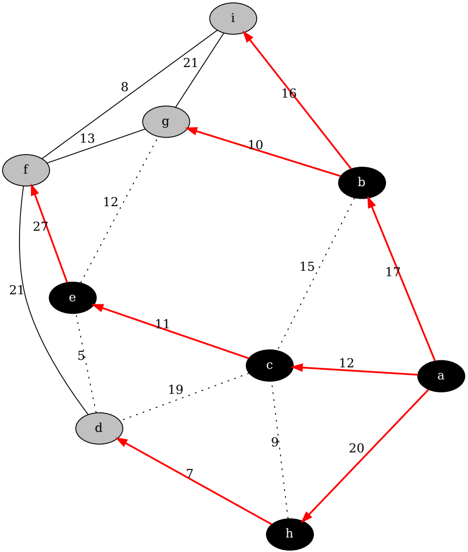
a	0
b	17
c	12
d	27
e	23
f	50
g	27
h	20
i	33

Priority Queue:
$(27, d), (27, g), (33, i), (50, f)$

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ shortest path found.
- Black edge $\longrightarrow$ not yet considered.
- Red edge $\longrightarrow$ shortest path edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

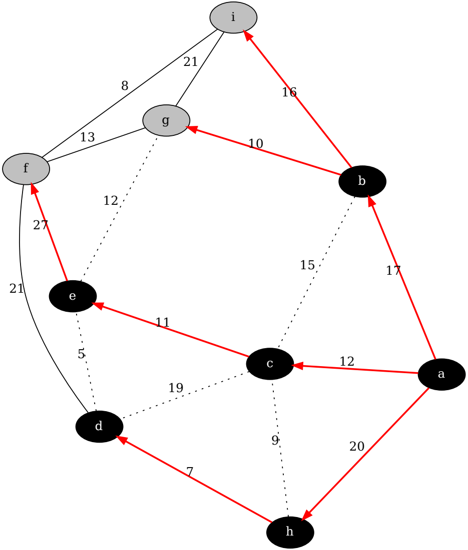
a	0
b	17
c	12
d	27
e	23
f	50
g	27
h	20
i	33

Priority Queue:
$(27, d), (27, g), (33, i), (50, f)$

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

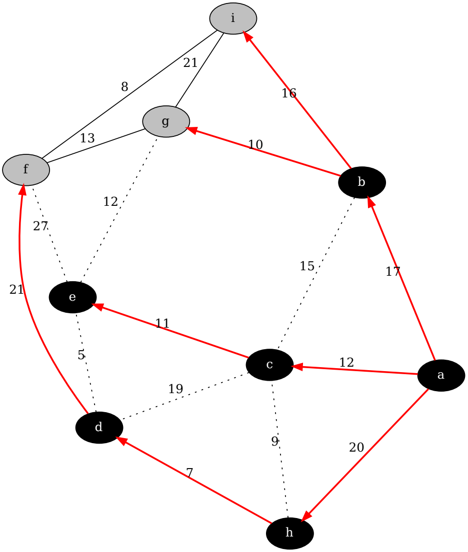
a	0
b	17
c	12
d	27
e	23
f	50
g	27
h	20
i	33

Priority Queue:
$(27, g), (33, i), (50, f)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

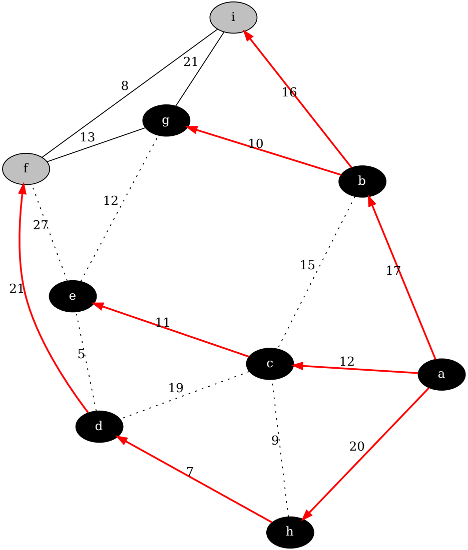
a	0
b	17
c	12
d	27
e	23
f	48
g	27
h	20
i	33

Priority Queue:
$(27, g), (33, i), (48, f)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

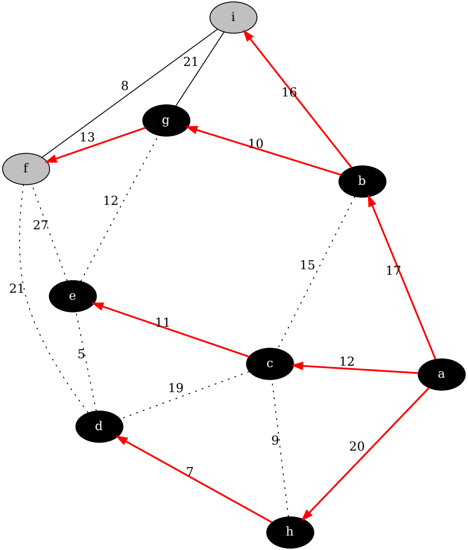
a	0
b	17
c	12
d	27
e	23
f	48
g	27
h	20
i	33

Priority Queue:
(33, i), (48, f)

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

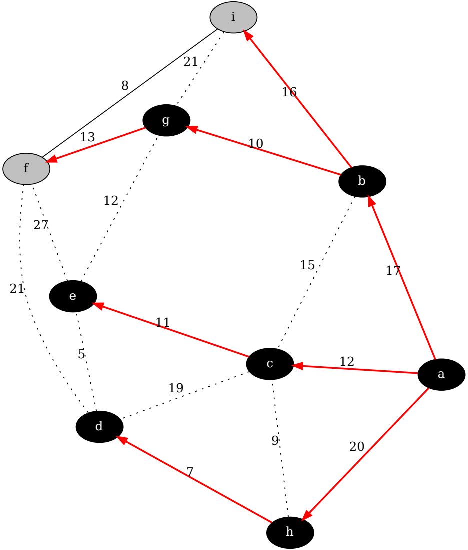
a	0
b	17
c	12
d	27
e	23
f	40
g	27
h	20
i	33

Priority Queue:
(33, i), (40, f)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

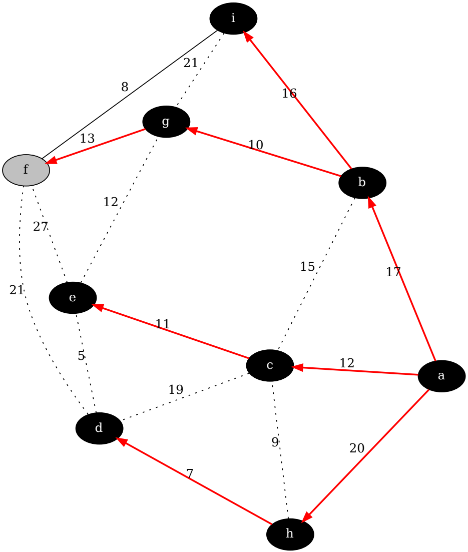
a	0
b	17
c	12
d	27
e	23
f	40
g	27
h	20
i	33

Priority Queue:
(33, i), (40, f)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

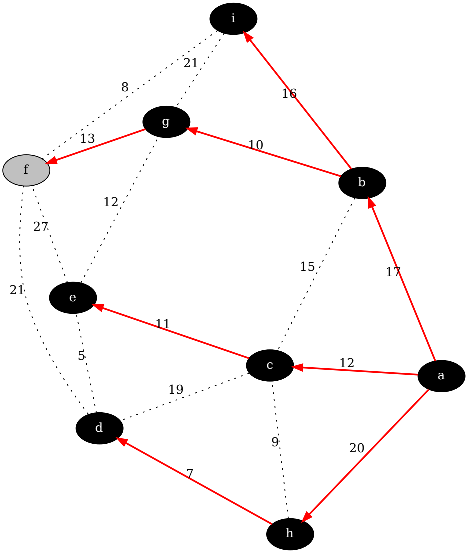
a	0
b	17
c	12
d	27
e	23
f	40
g	27
h	20
i	33

Priority Queue:
(40, f)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ shortest path found.
- Black edge $\longrightarrow$ not yet considered.
- Red edge $\longrightarrow$ shortest path edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

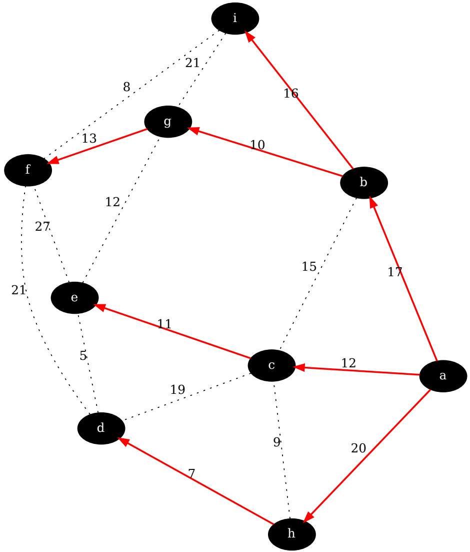
a	0
b	17
c	12
d	27
e	23
f	40
g	27
h	20
i	33

Priority Queue:
(40, f)

Legend:

- White vertex → no path found.
- Gray vertex → some path found.
- Black vertex → shortest path found.
- Black edge → not yet considered.
- Red edge → shortest path edge.
- Dotted edge → non-optimal edge.

Example



Distances:

a	0
b	17
c	12
d	27
e	23
f	40
g	27
h	20
i	33

Priority Queue:
$\emptyset$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ shortest path found.
- Black edge $\rightarrow$ not yet considered.
- Red edge $\rightarrow$ shortest path edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Correctness Proof

Easy direction

Definition

We denote $d_D(v)$ the final distance computed by Dijkstra for v.

We want to prove that $\forall v \in V$ we have:

$$d_D(v) = \mathrm{dist}(s, v)$$

Easy direction

Definition

We denote $d_D(v)$ the final distance computed by Dijkstra for v.

We want to prove that $\forall v \in V$ we have:

$$d_D(v) = \mathrm{dist}(s, v)$$

Lemma

For all $v \in V$ we have $\mathrm{dist}(s, v) \le d_D(v)$.

Proof of easy direction

Lemma

For all $v \in V$ *we have* $\mathrm{dist}(s, v) \leq d_D(v)$.

Proof of easy direction

Lemma

For all $v \in V$ we have $\mathrm{dist}(s, v) \leq d_D(v)$.

Proof.

- Initially dist$[v] = \infty \geq \mathrm{dist}(s, v)$.
- dist$[v]$ only modified on line 7.
- Prove by induction: $\forall$ modification, $\exists$ path of desired length.

Proof of easy direction

Lemma

For all $v \in V$ *we have* $\mathrm{dist}(s, v) \le d_D(v)$.

Proof.

- Initially $\text{dist}[v] = \infty \ge \mathrm{dist}(s, v)$.
- $\text{dist}[v]$ only modified on line 7.
- Prove by induction: $\forall$ modification, $\exists$ path of desired length.

Relevant line:

$\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$

- If by induction there exists $s \to u$ path of length at most $\text{dist}[u]$
- $\Rightarrow$ there exists $s \to v$ path of length at most $\text{dist}[u] + w(u, v)$

□

Optimal Sub-structure

Lemma

Shortest paths satisfy triangle inequality: $\forall a, b, c$ *we have* $\mathrm{dist}(a, b) \leq \mathrm{dist}(a, c) + \mathrm{dist}(c, b)$

Optimal Sub-structure

Lemma

Shortest paths satisfy triangle inequality: $\forall a, b, c$ *we have* $\mathrm{dist}(a, b) \leq \mathrm{dist}(a, c) + \mathrm{dist}(c, b)$

(**NB:** Above also hold for digraphs. Also, at the moment we assume all weights are positive!)

Optimal Sub-structure

Lemma

Shortest paths satisfy triangle inequality: $\forall a, b, c$ *we have* $\mathrm{dist}(a, b) \leq \mathrm{dist}(a, c) + \mathrm{dist}(c, b)$

(**NB:** Above also hold for digraphs. Also, at the moment we assume all weights are positive!)

Lemma (Optimal sub-struture)

If there is a shortest $a \to b$ *path that goes through* x*, then* $\mathrm{dist}(a, b) = \mathrm{dist}(a, x) + \mathrm{dist}(x, b)$.

Optimal Sub-structure

Lemma

Shortest paths satisfy triangle inequality: $\forall a, b, c$ *we have* $\mathrm{dist}(a, b) \leq \mathrm{dist}(a, c) + \mathrm{dist}(c, b)$

(**NB:** Above also hold for digraphs. Also, at the moment we assume all weights are positive!)

Lemma (Optimal sub-struture)

If there is a shortest $a \rightarrow b$ *path that goes through* x*, then* $\mathrm{dist}(a, b) = \mathrm{dist}(a, x) + \mathrm{dist}(x, b)$.

(In other words, we can construct a shortest $a \rightarrow b$ path by gluing a shortest $a \rightarrow x$ path with a shortest $x \rightarrow b$ path.)

Proof of hard direction

Two basic observations:

Lemma

The value dist[v] may only decrease as the algorithm progresses.

Proof of hard direction

Two basic observations:

Lemma

The value $dist[v]$ *may only decrease as the algorithm progresses.*

Proof.

Easy! □

Proof of hard direction

Two basic observations:

Lemma

The value dist$[v]$ may only decrease as the algorithm progresses.

Proof.

Easy! □

Lemma

When v exits the queue, dist$[v] \leq \mathrm{dist}(s, v)$.

Proof of hard direction

Two basic observations:

Lemma

The value dist[v] may only decrease as the algorithm progresses.

Proof.

Easy! □

Lemma

When v exits the queue, $dist[v] \leq \mathrm{dist}(s, v)$.

Therefore, $d_D(v) \leq \mathrm{dist}(s, v)$ as desired.

Main lemma

Lemma

When v exits the queue, $dist[v] \leq \mathrm{dist}(s, v)$.

Main lemma

Lemma

When v exits the queue, $dist[v] \leq \mathrm{dist}(s, v)$.

Proof.

Induction on number of vertices extracted from queue.

- OK for s (base case)

Main lemma

Lemma

When v exits the queue, $dist[v] \leq \mathrm{dist}(s, v)$.

Proof.

Induction on number of vertices extracted from queue.

- OK for s (base case)
- Suppose u was just extracted, so has minimum dist[u] over all remaining vertices.
- Let P be a shortest $s \to u$ path, x_1 be the **last** Black vertex of P.
- Let x_2 be the next vertex of P (not Black).

Main lemma

Lemma

When v exits the queue, $dist[v] \leq \mathrm{dist}(s, v)$.

Proof.

Induction on number of vertices extracted from queue.

- OK for s (base case)
- Suppose u was just extracted, so has minimum dist[u] over all remaining vertices.
- Let P be a shortest $s \to u$ path, x_1 be the **last** Black vertex of P.
- Let x_2 be the next vertex of P (not Black).

$$\mathrm{dist}[u] \leq \mathrm{dist}[x_2] \leq \mathrm{dist}(s, x_1) + w(x_1, x_2) = \mathrm{dist}(s, x_2) \leq \mathrm{dist}(s, u)$$

□

Complexity Analysis

- Each vertex is extracted once.
  - By previous lemma, when we extract, we have correct distance.
  - We only insert/decrease when we find a shortcut; impossible if we have correct distance.
- Therefore, $O(n)$ Extract operations and $O(m)$ Decrease/Insert operations.
  - $O(d(v))$ Decrease operations per vertex.

Complexity Analysis

- Each vertex is extracted once.
  - By previous lemma, when we extract, we have correct distance.
  - We only insert/decrease when we find a shortcut; impossible if we have correct distance.
- Therefore, $O(n)$ Extract operations and $O(m)$ Decrease/Insert operations.
  - $O(d(v))$ Decrease operations per vertex.
- What is the cost of each operation?
  - Naïve implementation: $O(n)$ per operation $\Rightarrow O(mn + n^2)$ time in total.

Min-Heaps

Reminder: min-heaps

- Initialize an array H of size n
  - Keep track of how many items we have
- Invariant: for all i we have $H[i] \leq H[2i+1]$ and $H[i] \leq H[2i+2]$
  - Min element always at $H[0]$
- Insert: add element to the end of the array, **Fix**
- Decrease: **Fix**
- Fix: if inequality is violated, exchange $H[i]$ with smaller of $H[2i+1], H[2i+2]$, repeat.
  - $O(\log n)$ complexity to completely fix one offending element.

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

*	*	*	*	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

5	*	*	*	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

5	3	*	*	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

3	5	*	*	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

3	5	8	*	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

3	5	8	1	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\to$ 1

Heap state:

3	1	8	5	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\to$ 1

Heap state:

1	3	8	5	*	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	3	8	5	2	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	8	5	3	*	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	8	5	3	7	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	7	5	3	8	*	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	7	5	3	8	6	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	6	5	3	8	7	*	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	6	5	3	8	7	4	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	6	4	3	8	7	5	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	2	6	4	3	8	7	5	9	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

9	2	6	4	3	8	7	5	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

2	9	6	4	3	8	7	5	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

2	3	6	4	9	8	7	5	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

2	3	6	4	9	1	7	5	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

2	3	1	4	9	6	7	5	*	*

Min-Heap example

Operations:

1. Insert 5, 3, 8, 1, 2, 7, 6, 4, 9
2. Extract
3. Decrease 8 $\rightarrow$ 1

Heap state:

1	3	2	4	9	6	7	5	*	*