Graph Algorithms

Shortest Paths III

Michael Lampis

October 10, 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.

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.
- Two lectures ago: Dijkstra's algorithm, SSSP with **positive weights**
- Last lecture: SSSP with possibly negative weights (Bellman-Ford), APSP (Matrix Multiplication-like)

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.
- Two lectures ago: Dijkstra's algorithm, SSSP with **positive weights**
- Last lecture: SSSP with possibly negative weights (Bellman-Ford), APSP (Matrix Multiplication-like)
- Today:
  - Floyd-Warshall algorithm for APSP
  - Johnson's algorithm, APSP for sparse graphs, negative weights
  - Conditional Impossibility Results

APSP

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra, Bellman-Ford (SSSP), run n times (APSP).

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra, Bellman-Ford (SSSP), run n times (APSP).
- Last week: $(\min, +)$-product APSP algorithm $\Rightarrow O(n^3 \log n)$ time.
  - OK for negative weights.

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra, Bellman-Ford (SSSP), run n times (APSP).
- Last week: $(\min, +)$-product APSP algorithm $\Rightarrow O(n^3 \log n)$ time.
  - OK for negative weights.
- Can we do better? For sparse graphs?

The Floyd-Warshall-Roy algorithm

Historical Note

- Published by Robert Floyd in 1962.

Historical Note

- Published by Robert Floyd in 1962.
- Also by Stephen Warshall in 1962.

Historical Note

- Published by Robert Floyd in 1962.
- Also by Stephen Warshall in 1962.
- Also by Bernard Roy in **1959**.

Historical Note

- Published by Robert Floyd in 1962.
- Also by Stephen Warshall in 1962.
- Also by Bernard Roy in **1959**.
  - Caveat: published in French.

Remark

. . . De ce théoreme découle un procédé simple et mécanisable permettant de construire la fermeture transitive d'une matrice quelconque $M \in \Omega$. Ce résultat semble susceptible d'applications nombreuses en théorie des graphes. . . .

Historical Note

- Published by Robert Floyd in 1962.
- Also by Stephen Warshall in 1962.
- Also by Bernard Roy in **1959**.
  - Caveat: published in French.

Remark

. . . De ce théoreme découle un procédé simple et mécanisable permettant de construire la fermeture transitive d'une matrice quelconque $M \in \Omega$. Ce résultat semble susceptible d'applications nombreuses en théorie des graphes. . . .

Bernard Roy is the founder of LAMSADE in Université Paris-Dauphine

Reminder: Optimal Sub-structure

Lemma

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

Reminder: 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. Assume no negative-weight cycles.)

Reminder: 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. Assume no negative-weight cycles.)

Lemma (Optimal sub-structure)

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

Reminder: 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. Assume no negative-weight cycles.)

Lemma (Optimal sub-structure)

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

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

Basic Strategy

- Start with some initial distances
  - $\mathrm{dist}[i,j] \leftarrow w(i,j)$.
- For each vertex k check if k can be used as a shortcut.
  - Compare: Bellman-Ford (for each edge e, check if shortcut)
- k is a shortcut if $\mathrm{dist}[i,k] + \mathrm{dist}[k,j] < dist[i,j]$
  - If Yes, update.

Basic Strategy

- Start with some initial distances
  - $\mathrm{dist}[i,j] \leftarrow w(i,j)$.
- For each vertex k check if k can be used as a shortcut.
  - Compare: Bellman-Ford (for each edge e, check if shortcut)
- k is a shortcut if $\mathrm{dist}[i,k] + \mathrm{dist}[k,j] < dist[i,j]$
  - If Yes, update.

Invariant:

- After k iterations, for all i, j, $\mathrm{dist}[i,j]$ contains the shortest-path length using internal vertices from $\{1, \ldots, k\}$.

Algorithm in pseudocode

1: $\forall i, j \in V : \ \mathrm{dist}[i, j] = w(ij)$
2: **for** $k = 1$ to n **do**
3: **for** $i = 1$ to n **do**
4: **for** $j = 1$ to n **do**
5: **if** $\mathrm{dist}[i, j] > \mathrm{dist}[i, k] + \mathrm{dist}[k, j]$ **then**
6: $\mathrm{dist}[i, j] \leftarrow \mathrm{dist}[i, k] + \mathrm{dist}[k, j]$ ▷ k shortcut for $i \to j$
7: **end if**
8: **end for**
9: **end for**
10: **end for**

Algorithm in pseudocode

1: $\forall i, j \in V : \ \mathrm{dist}[i, j] = w(ij)$
2: **for** $k = 1$ to n **do**
3: **for** $i = 1$ to n **do**
4: **for** $j = 1$ to n **do**
5: **if** $\mathrm{dist}[i, j] > \mathrm{dist}[i, k] + \mathrm{dist}[k, j]$ **then**
6: $\mathrm{dist}[i, j] \leftarrow \mathrm{dist}[i, k] + \mathrm{dist}[k, j]$ ▷ k shortcut for $i \to j$
7: **end if**
8: **end for**
9: **end for**
10: **end for**

NB: For loops structured $K - I - J$.

Complexity

- Time: $O(n^3)$ arithmetic operations.
  - Three nested loops, each n iterations.
- Space: $O(n^2)$ for distance matrix.
  - Assume distances in $O(1)$ space.

Correctness

Lemma

After k iterations of outer loop, $\mathrm{dist}[i, j]$ is equal to shortest-path distance from i to j using internal vertices from $\{1, \ldots, k\}$.

Correctness

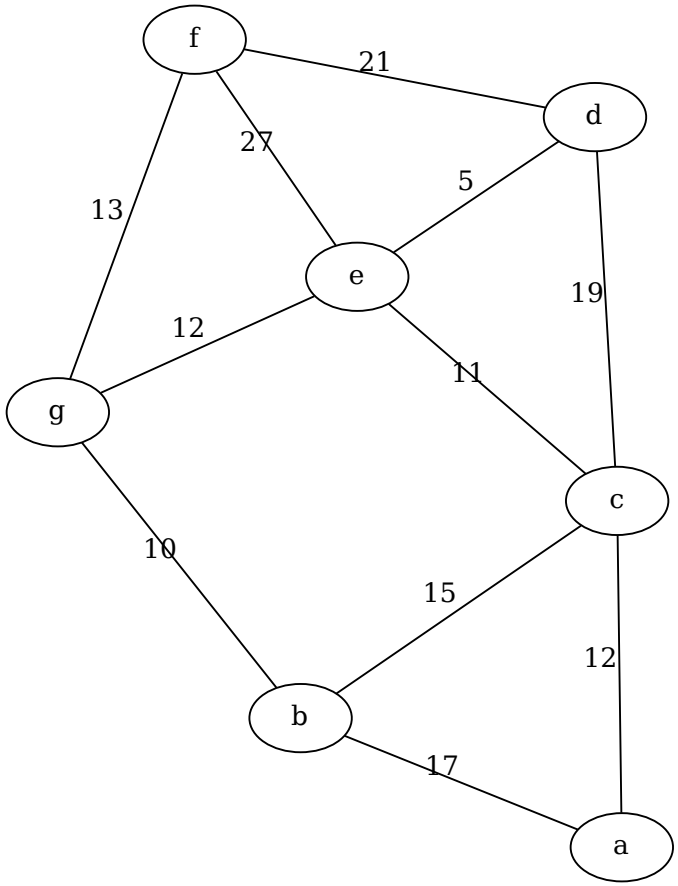
Lemma

After k iterations of outer loop, $\mathrm{dist}[i,j]$ is equal to shortest-path distance from i to j using internal vertices from $\{1,\ldots,k\}$.

Proof.

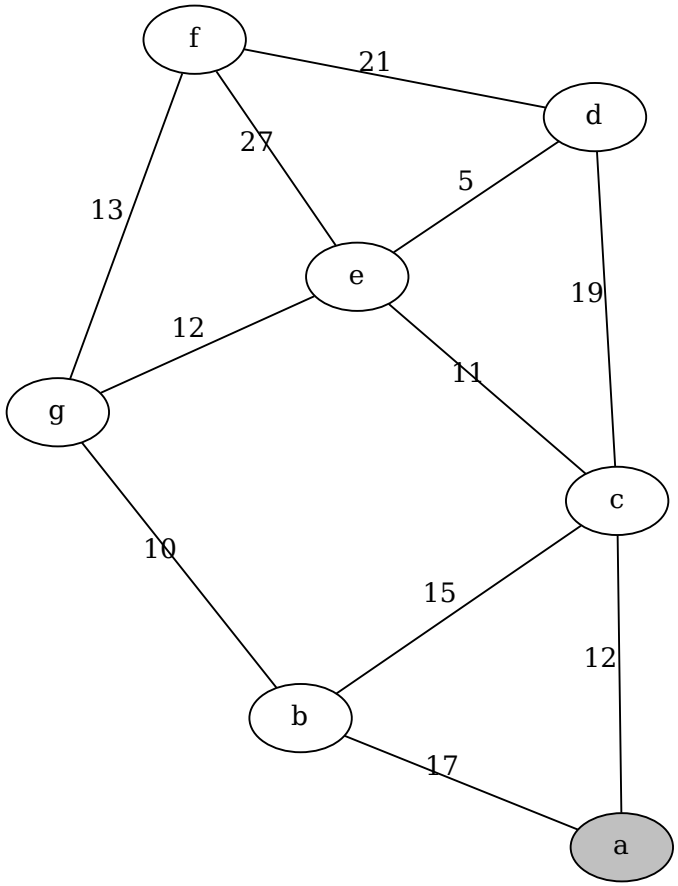
- $k = 0 \Rightarrow$ clear.
- $(k-1) \rightarrow k$: If k part of shortest path from i to j, then $\mathrm{dist}(i,j) = \mathrm{dist}(i,k) + \mathrm{dist}(k,j)$.

Example



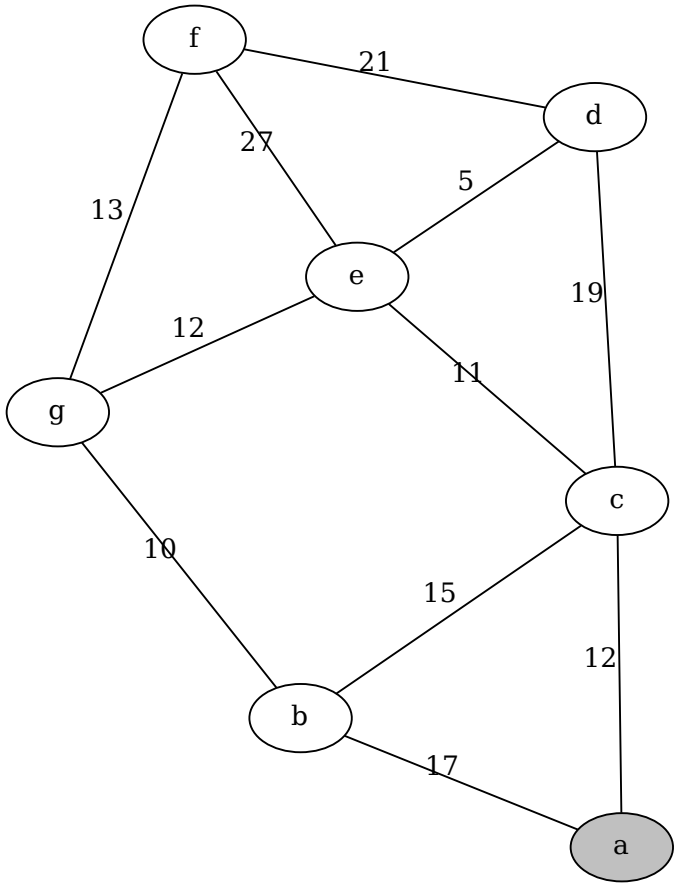
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



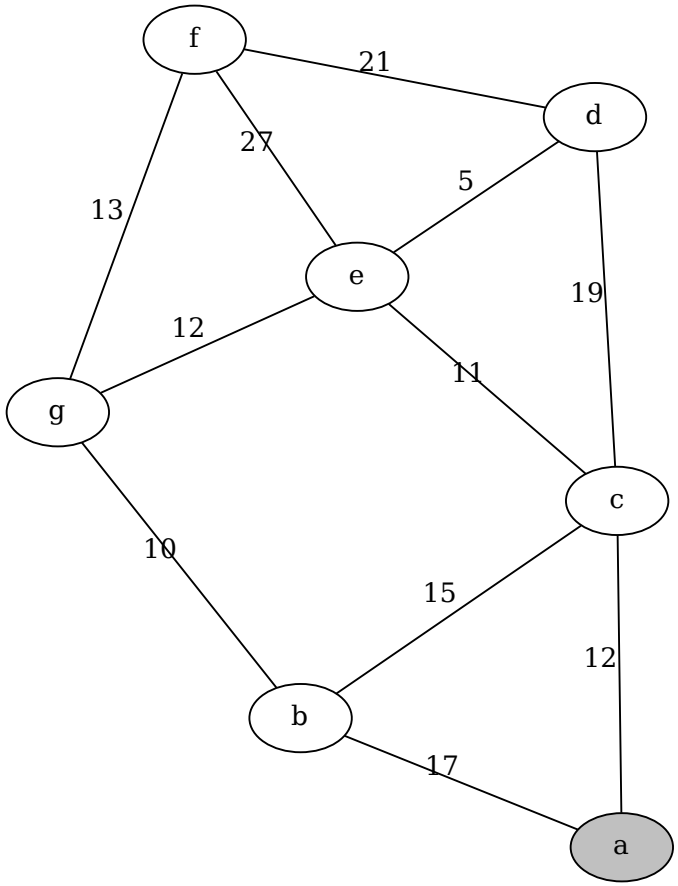
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



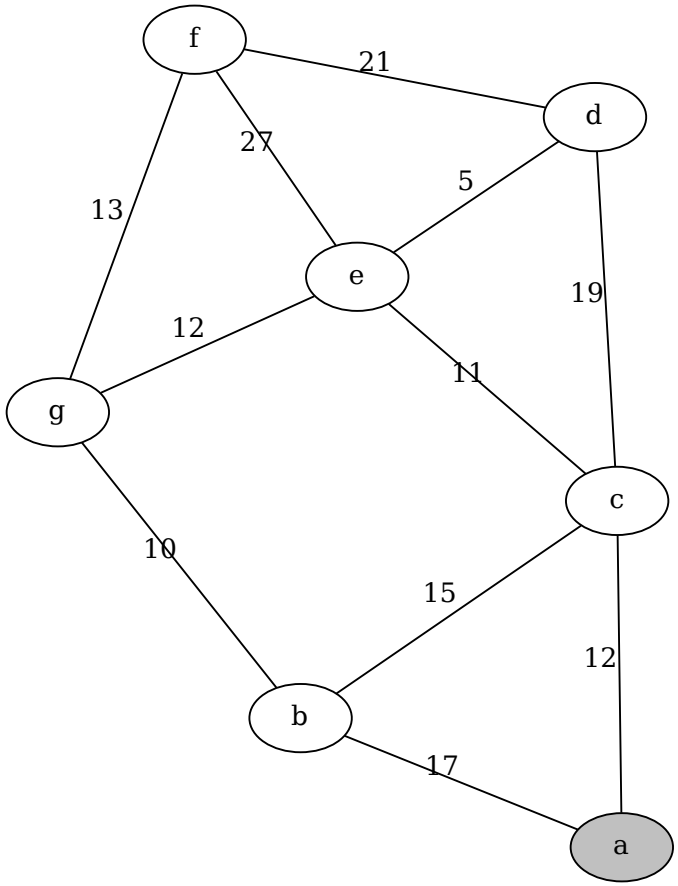
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



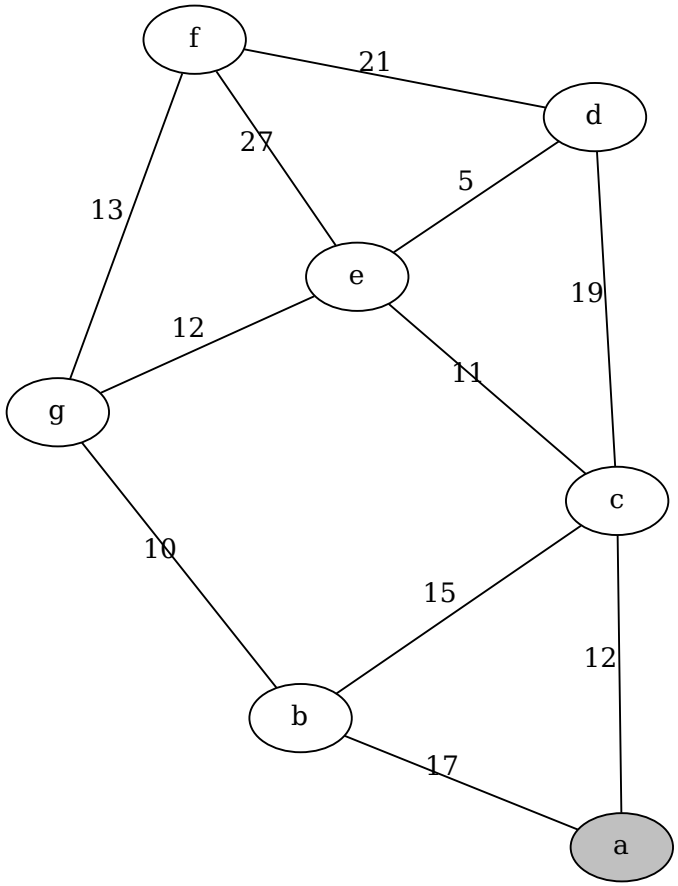
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



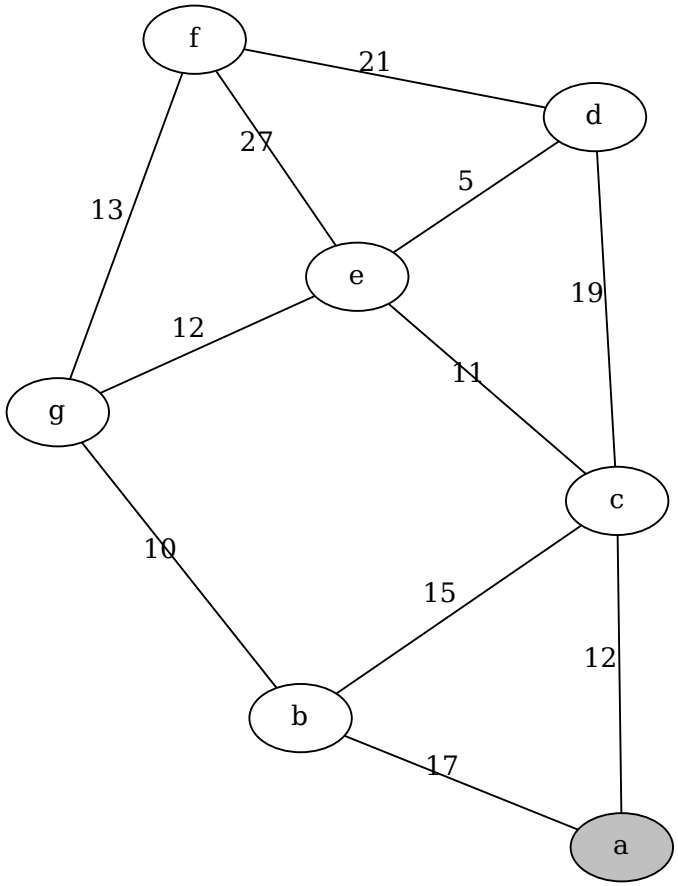
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



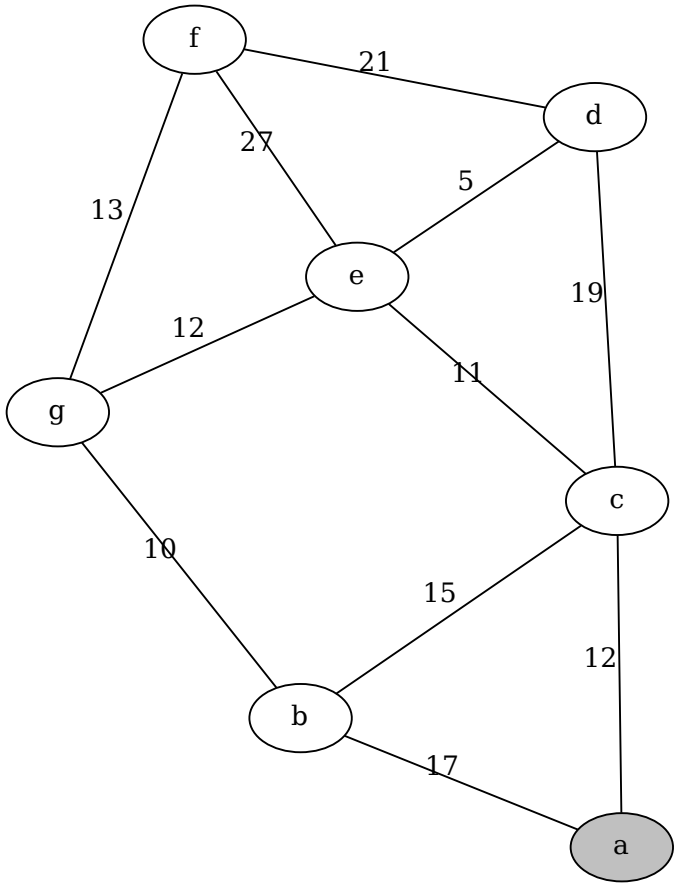
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



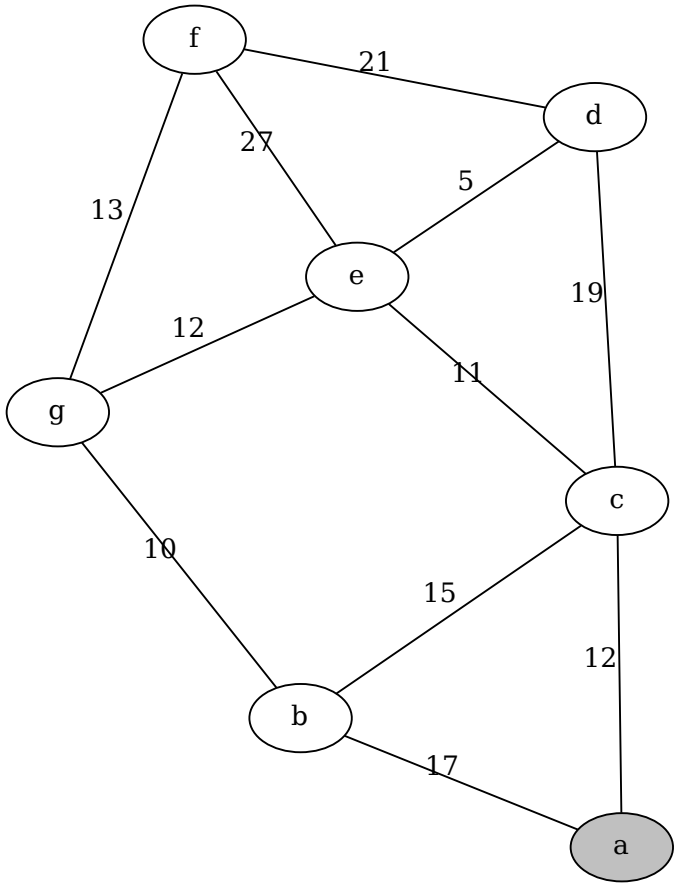
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



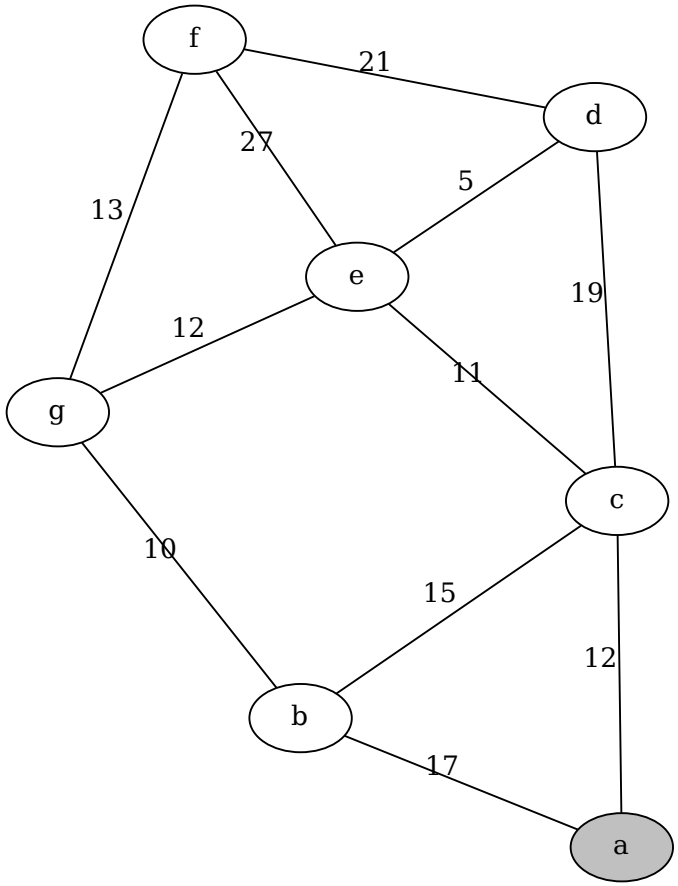
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



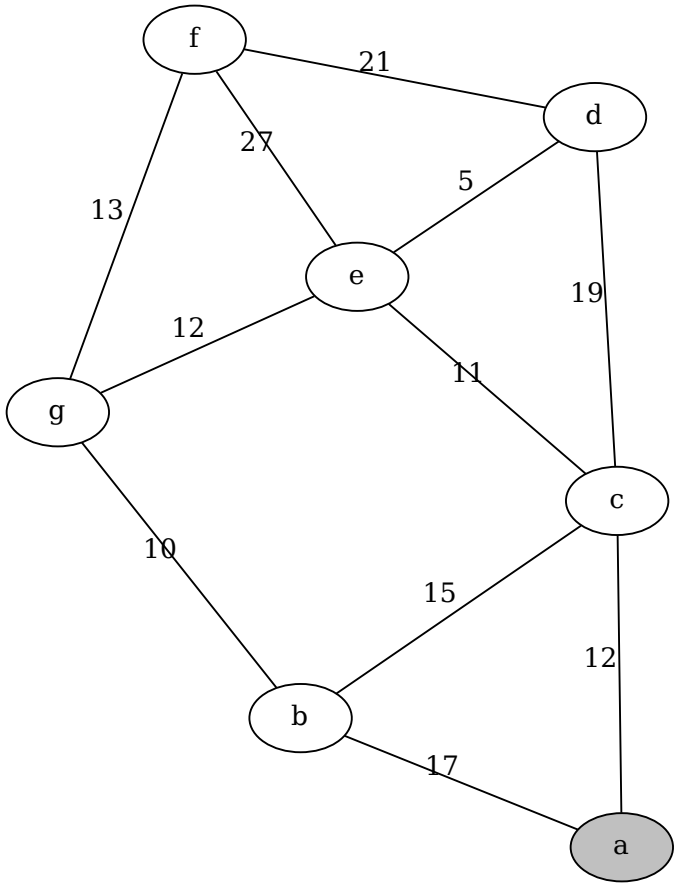
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



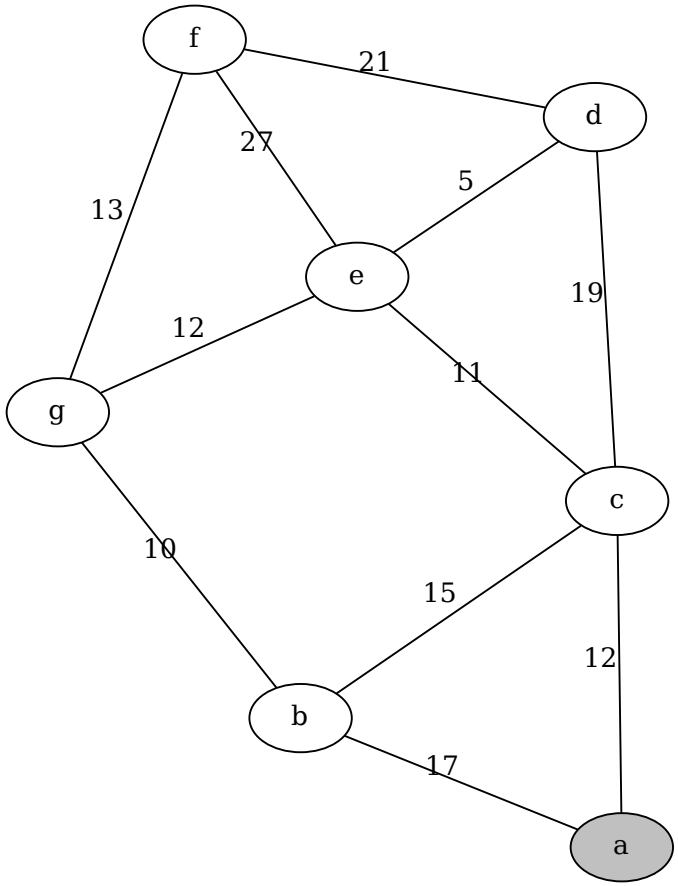
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



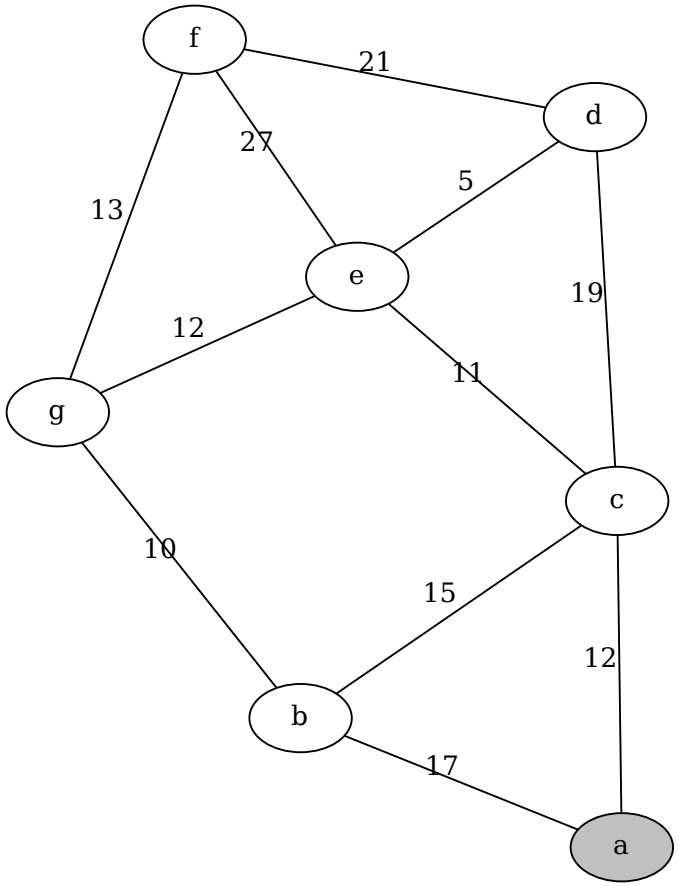
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



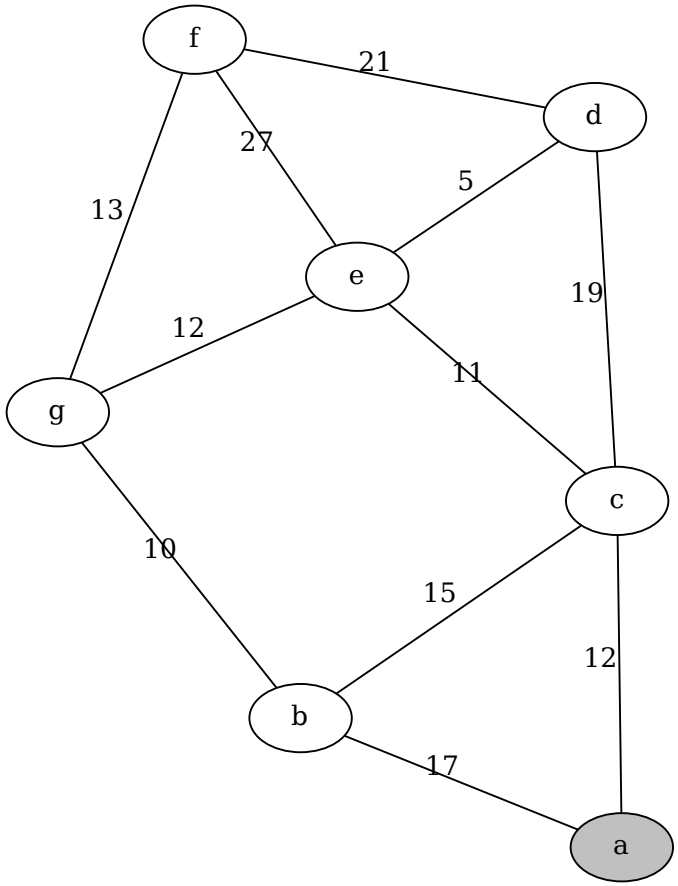
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



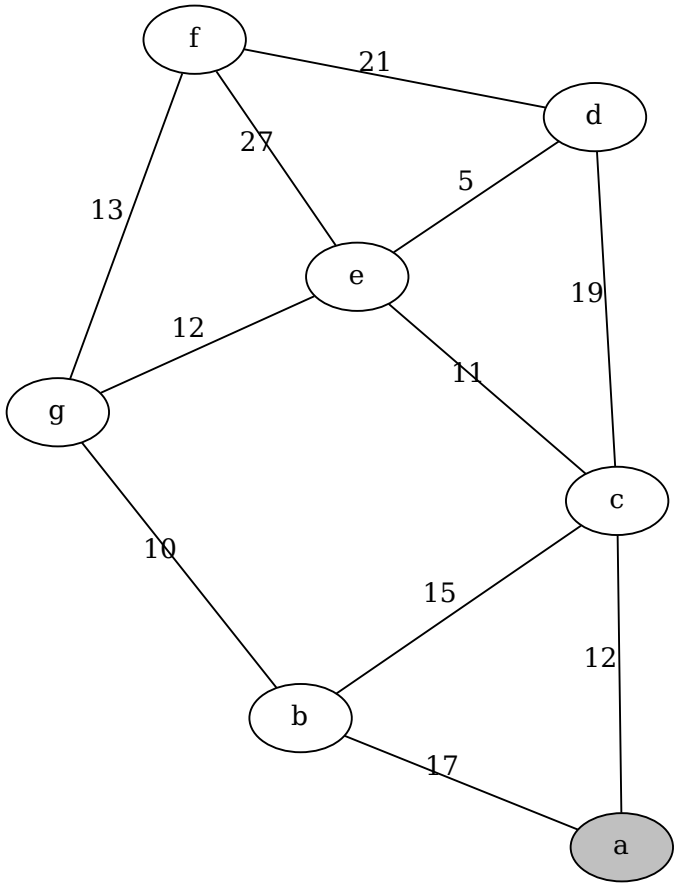
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



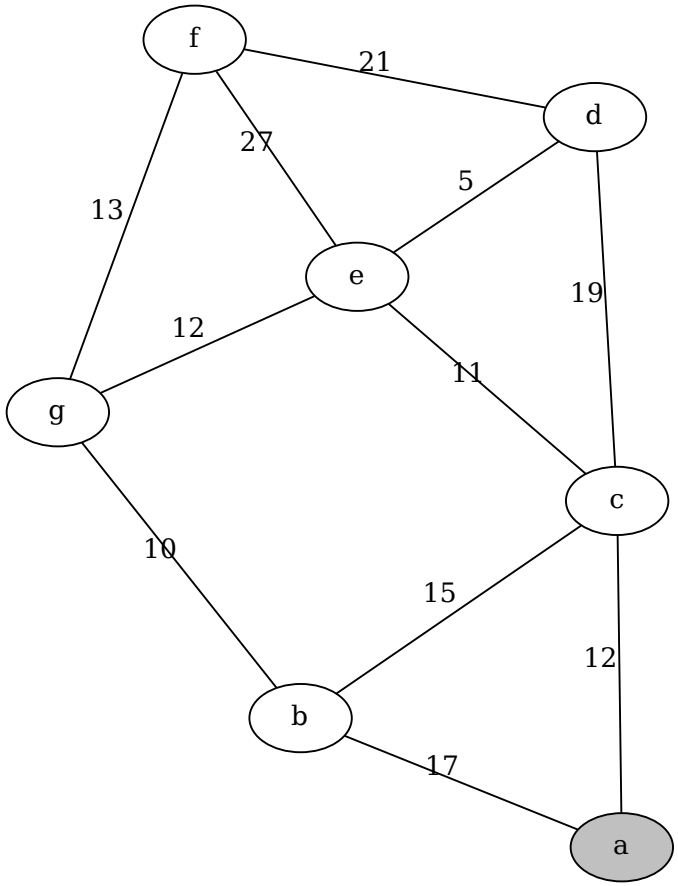
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



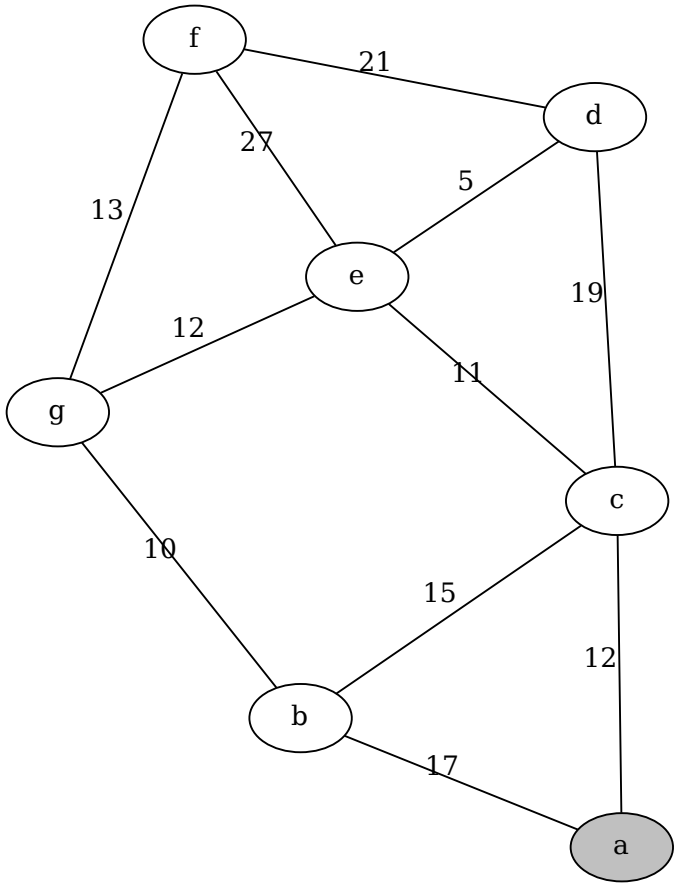
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



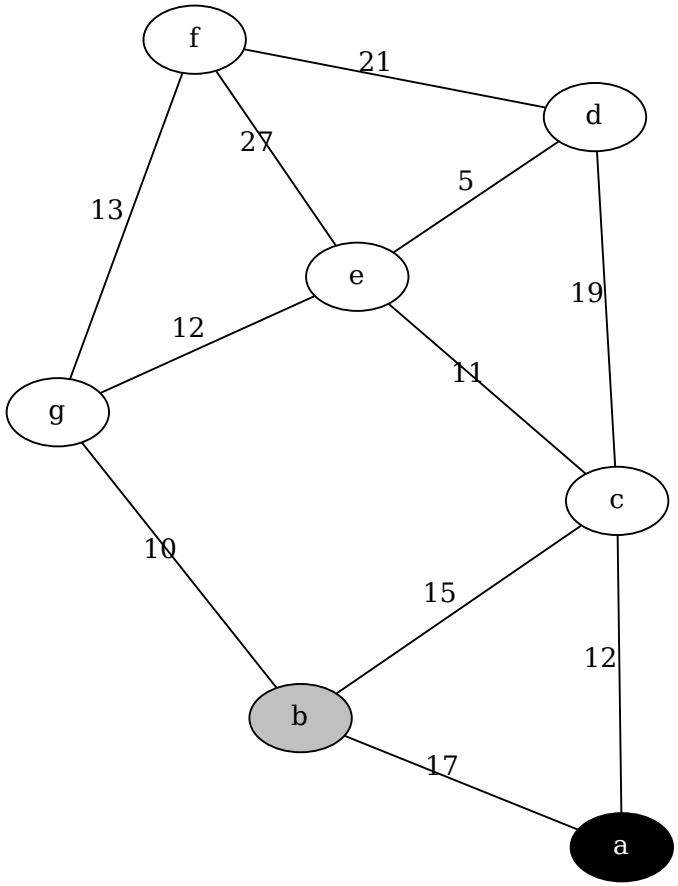
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



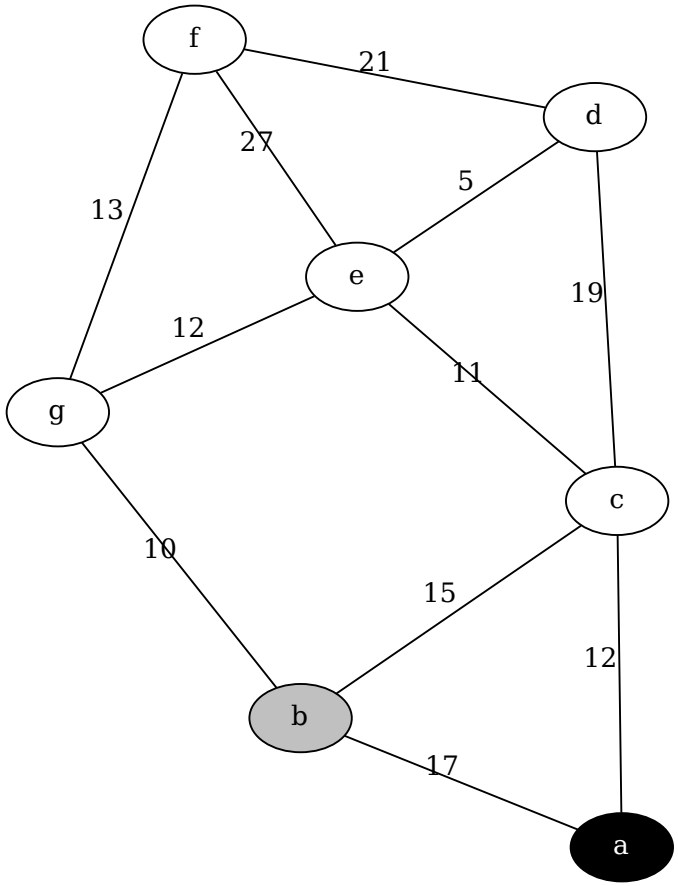
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



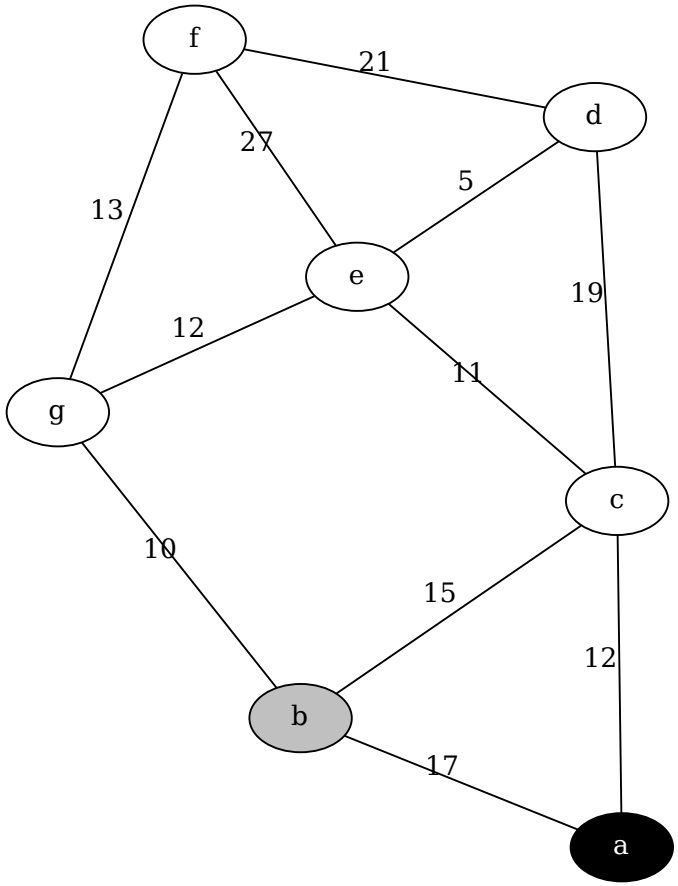
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



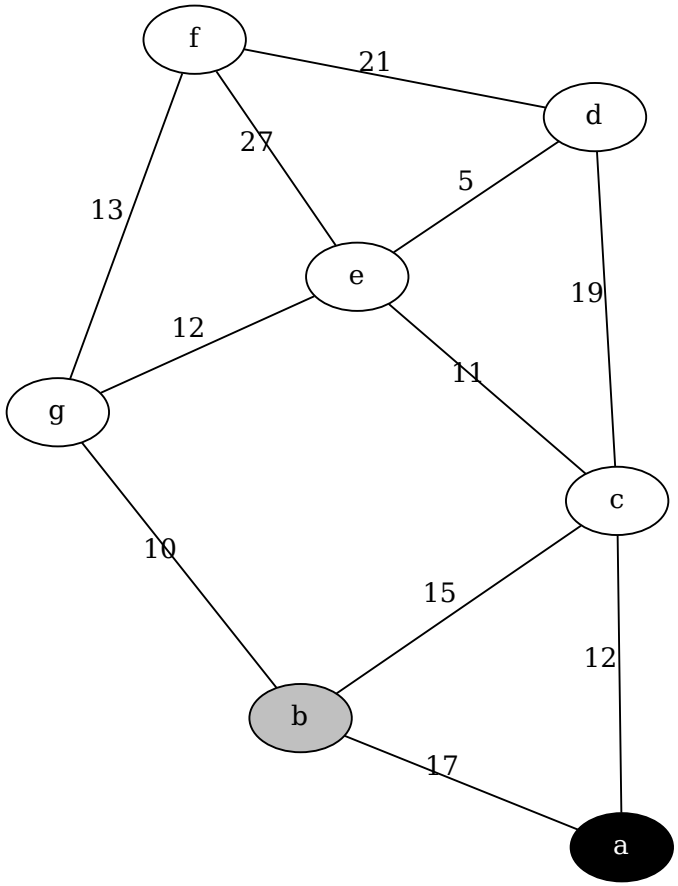
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



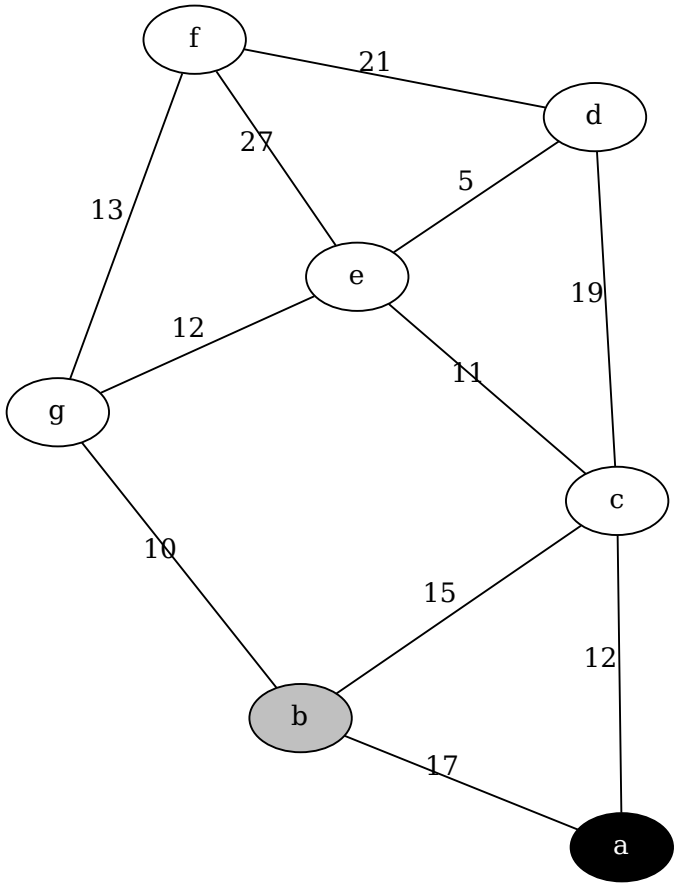
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



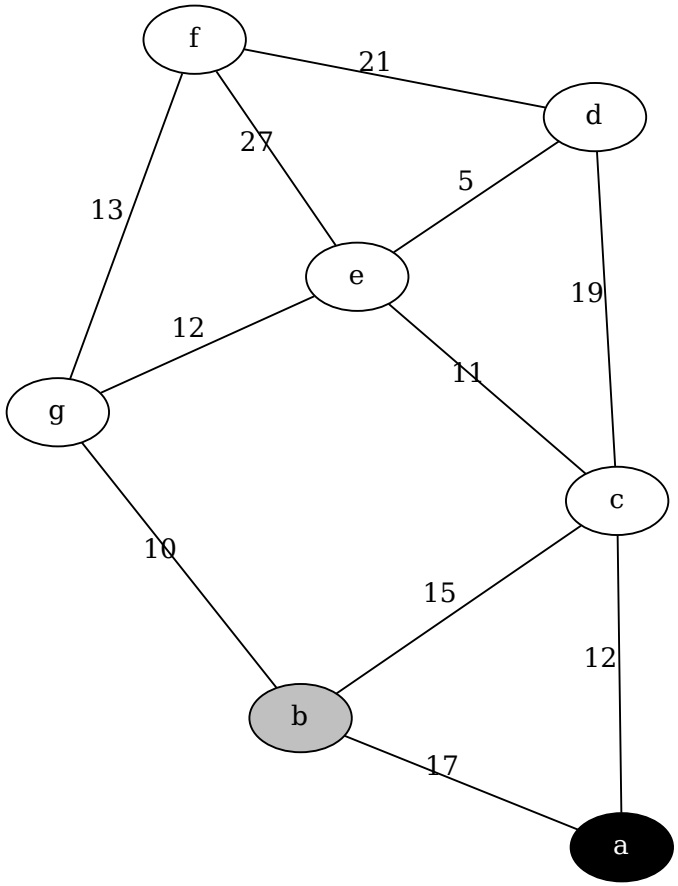
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



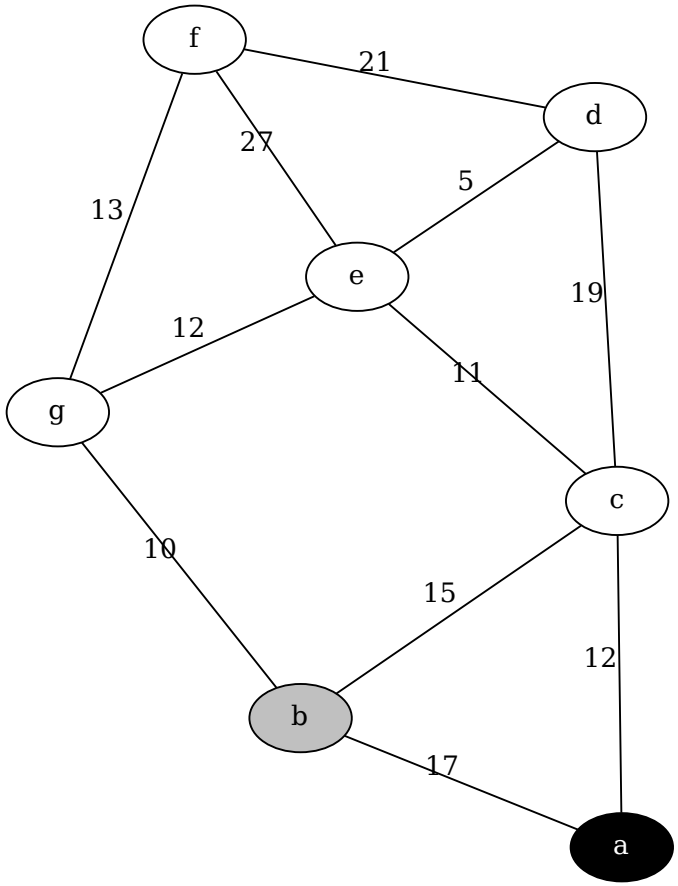
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



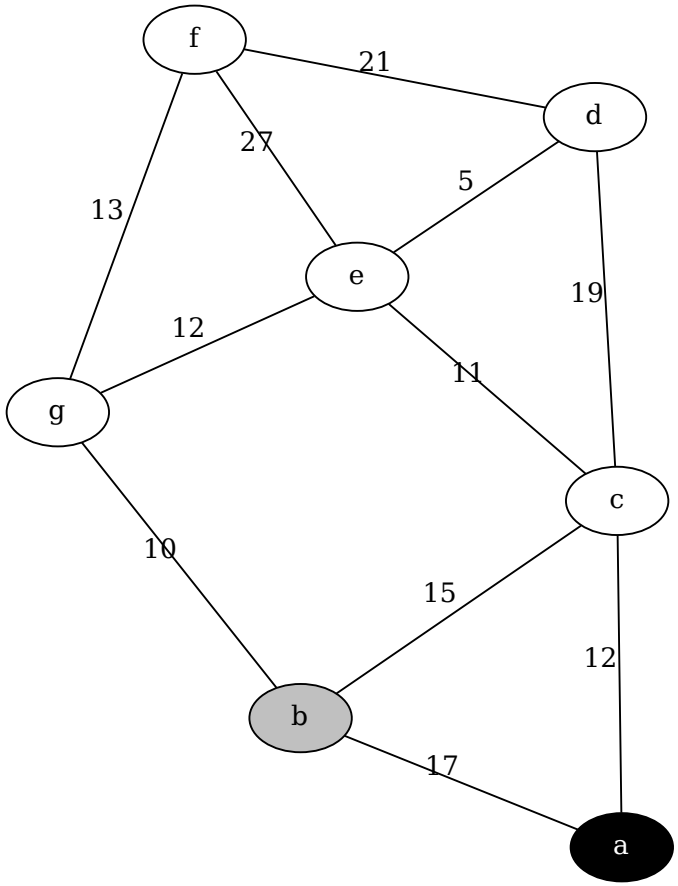
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



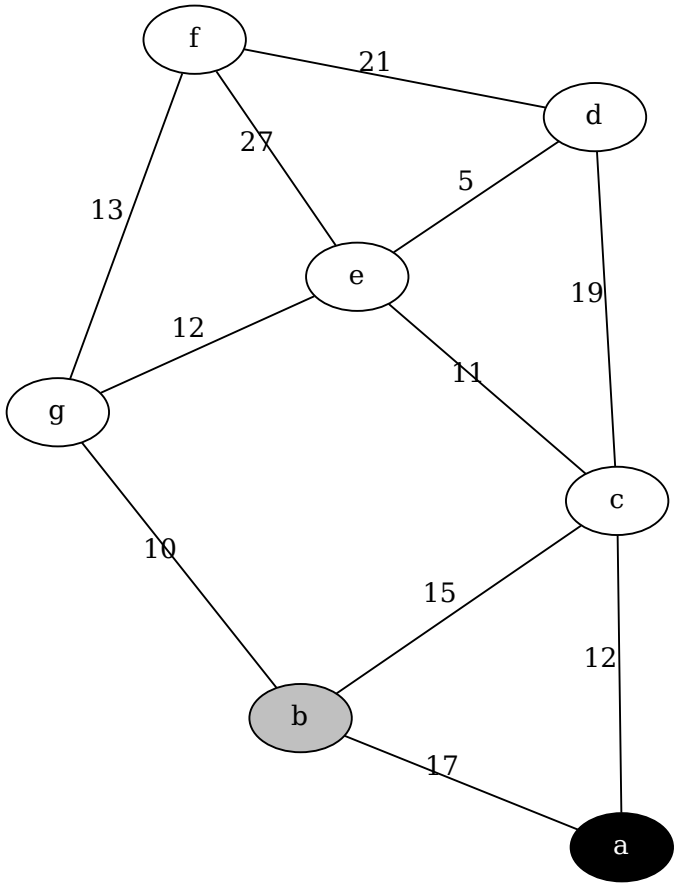
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



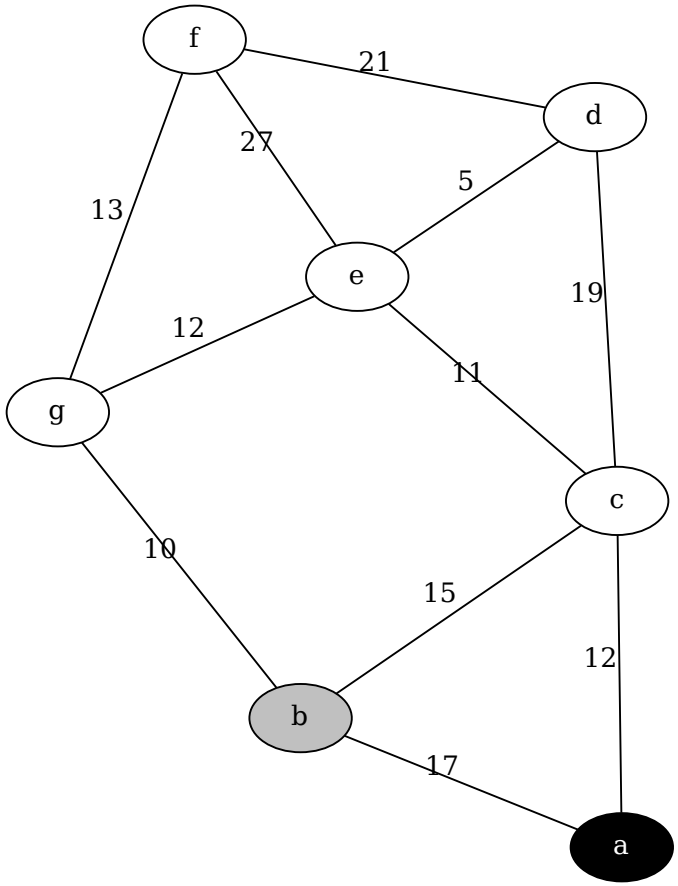
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	∞
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



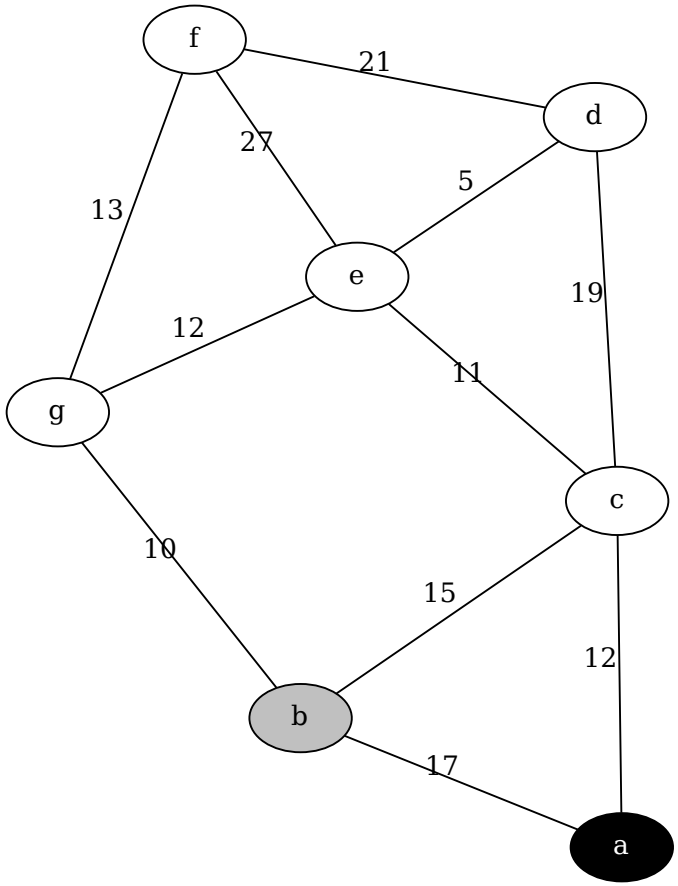
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



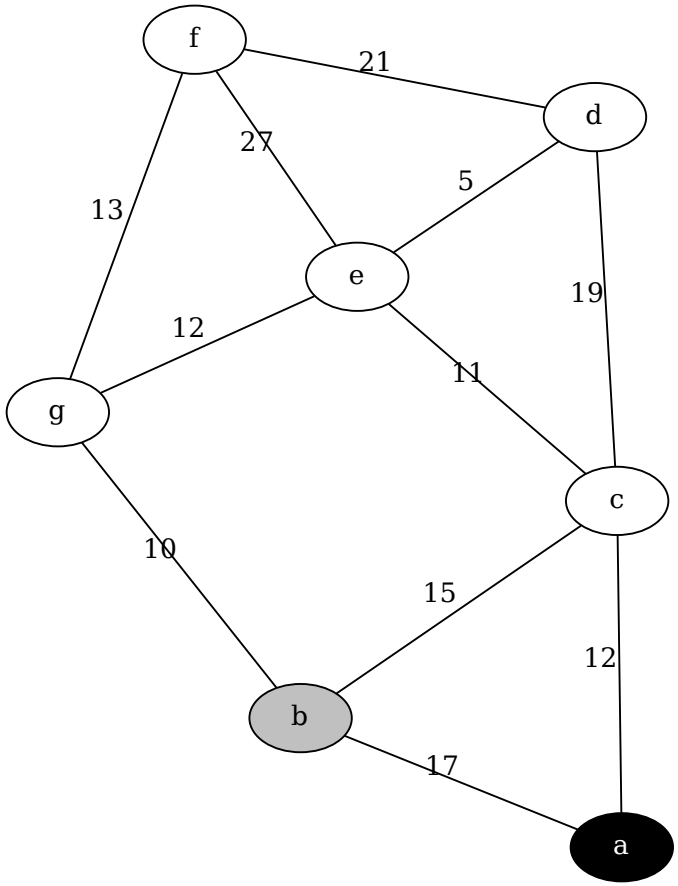
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



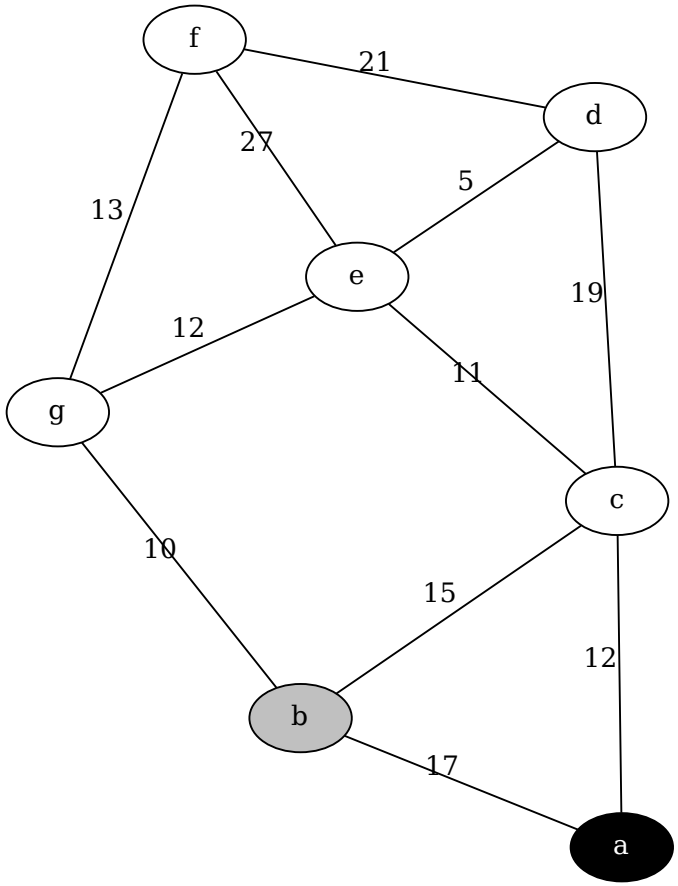
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



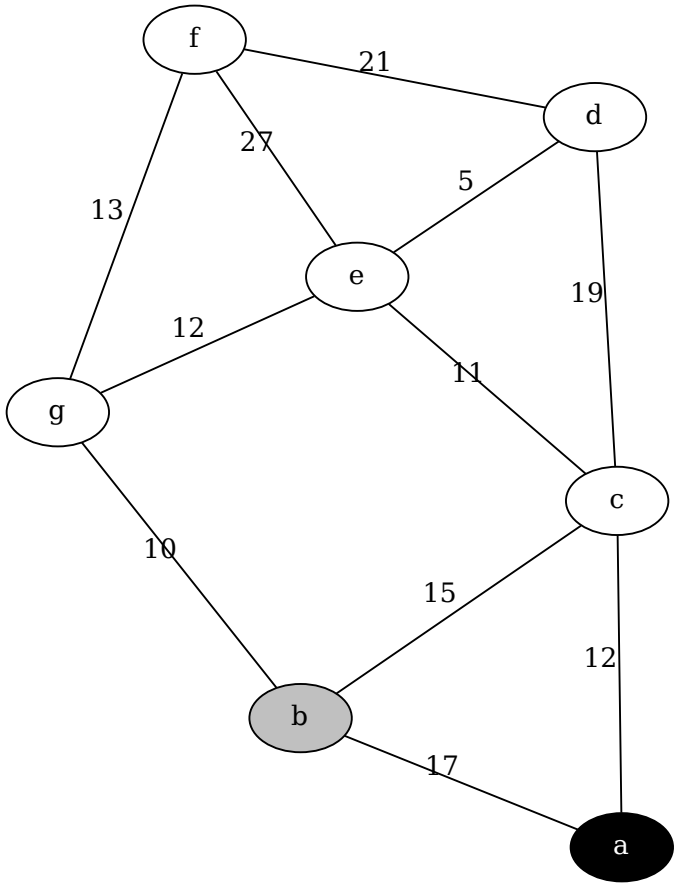
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



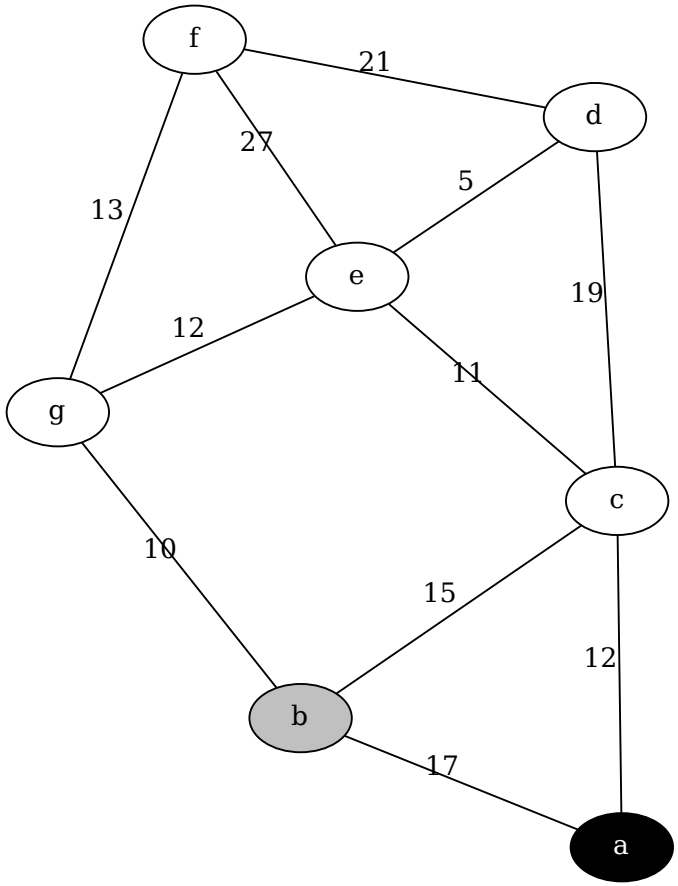
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



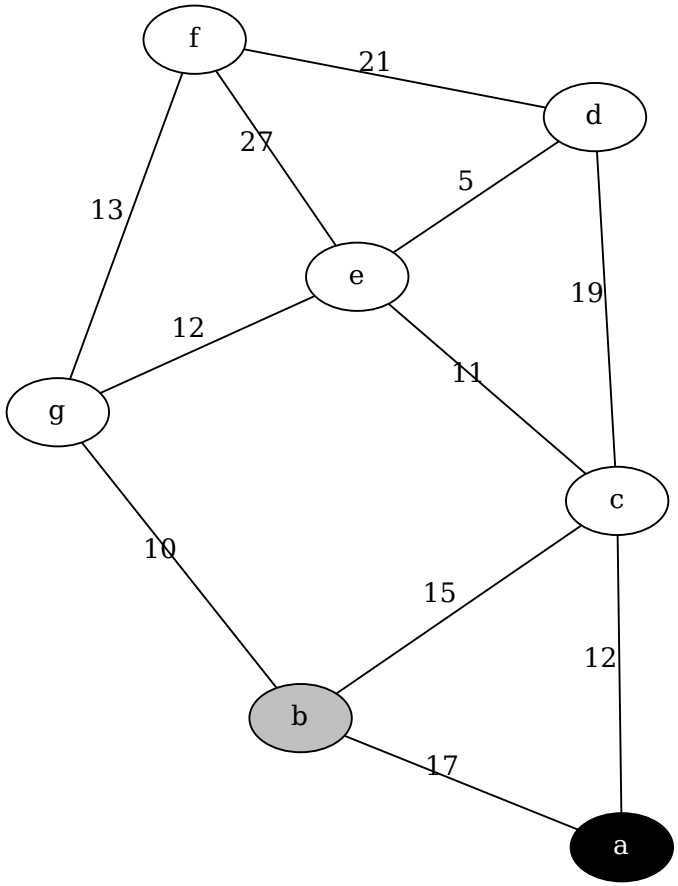
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	∞
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



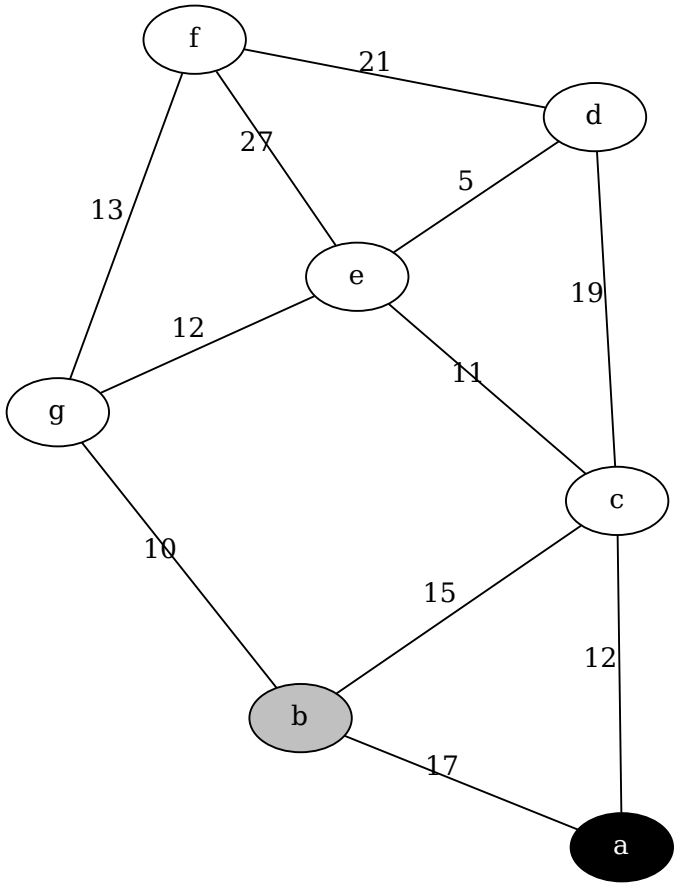
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



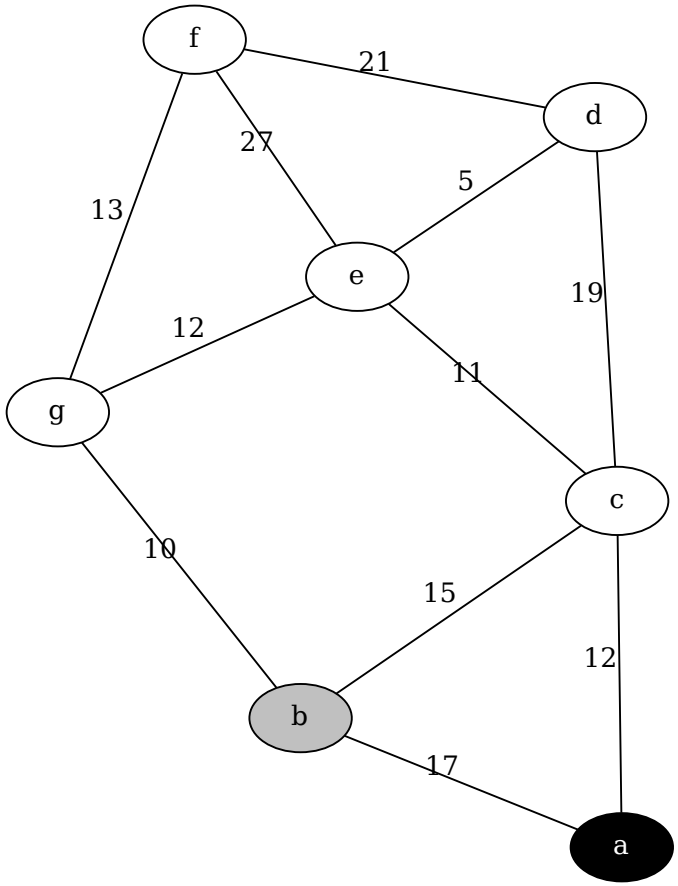
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



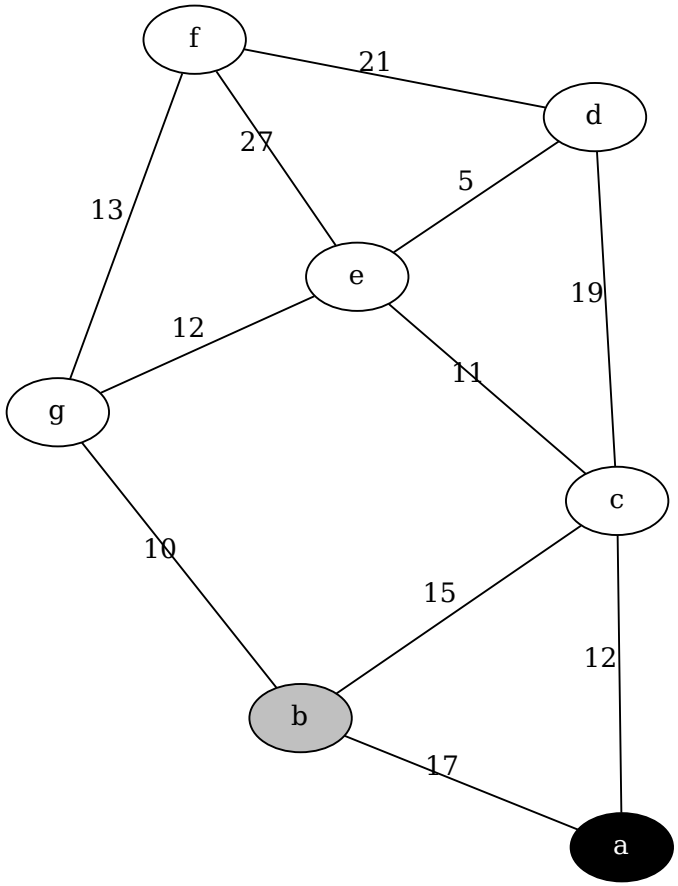
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



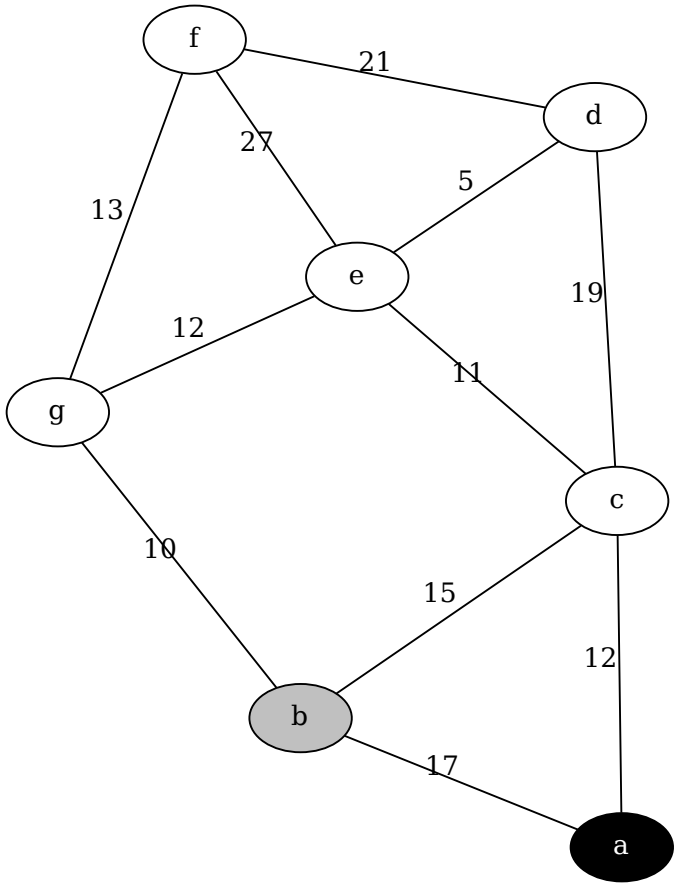
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



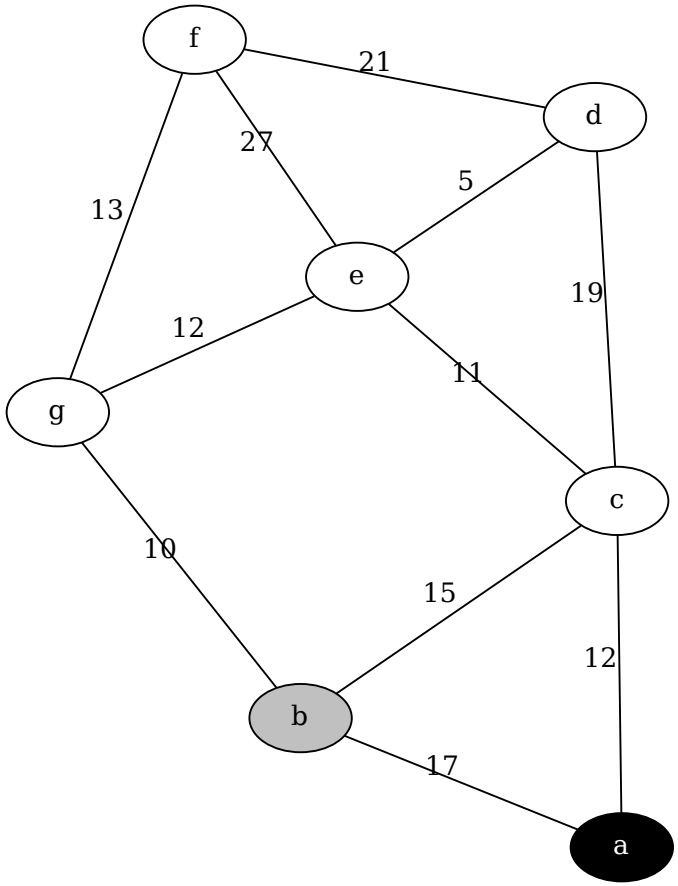
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



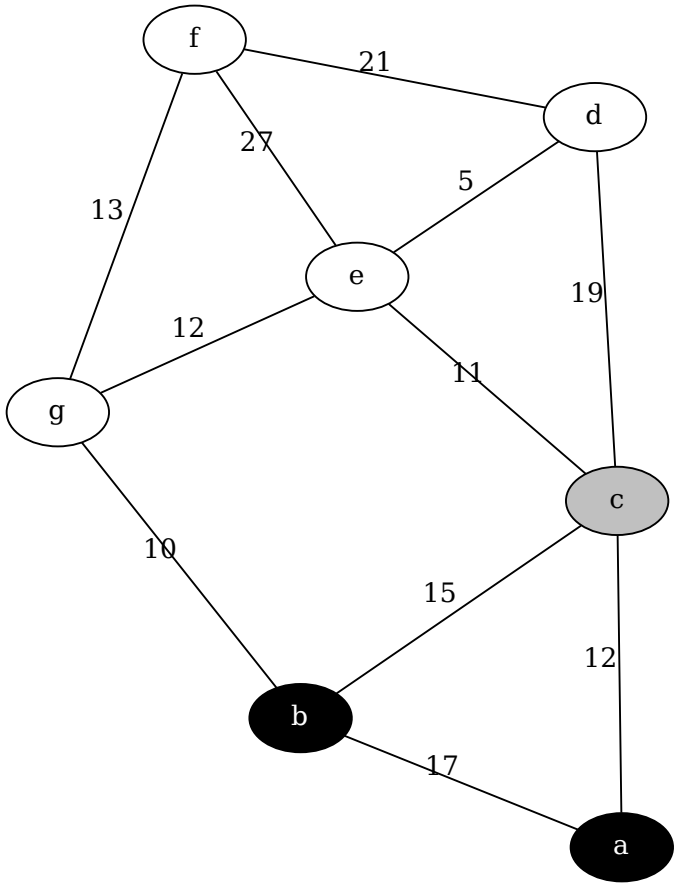
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



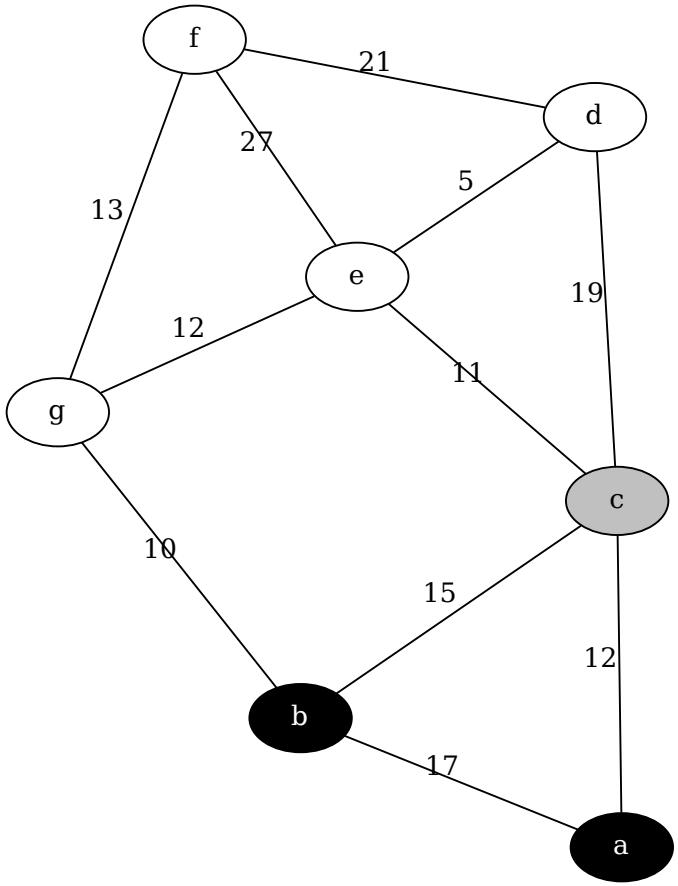
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



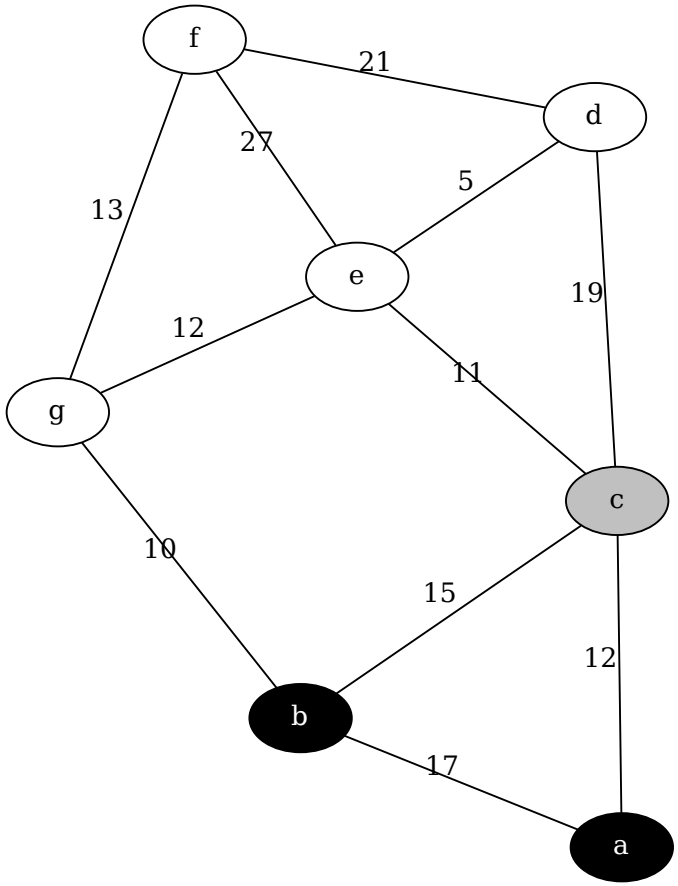
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



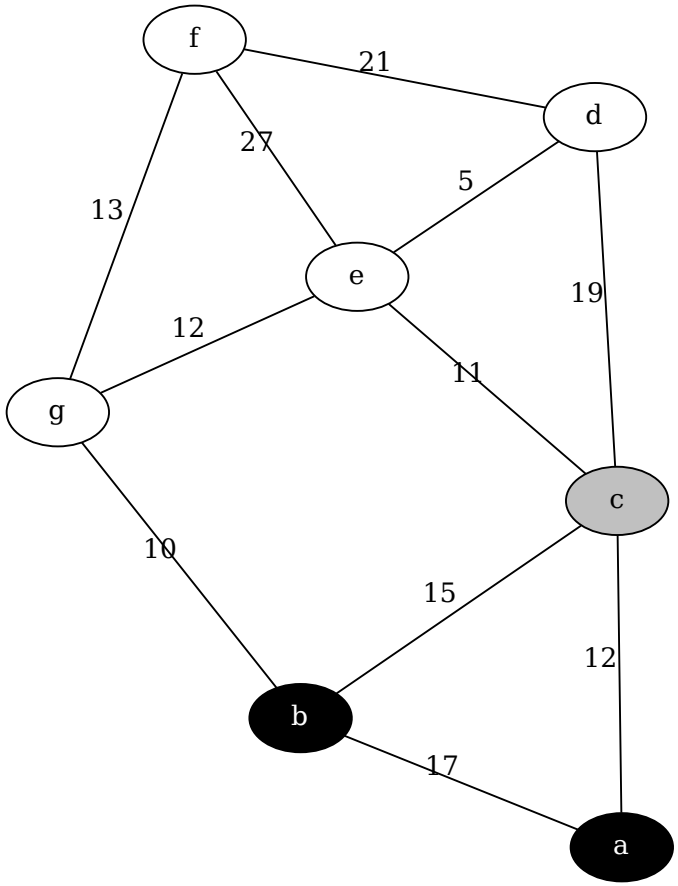
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



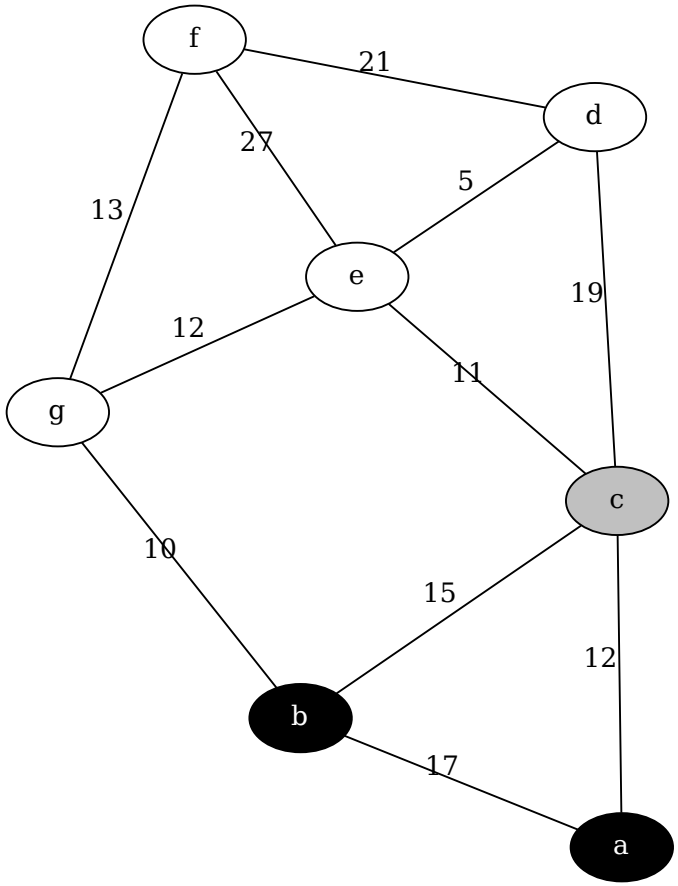
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



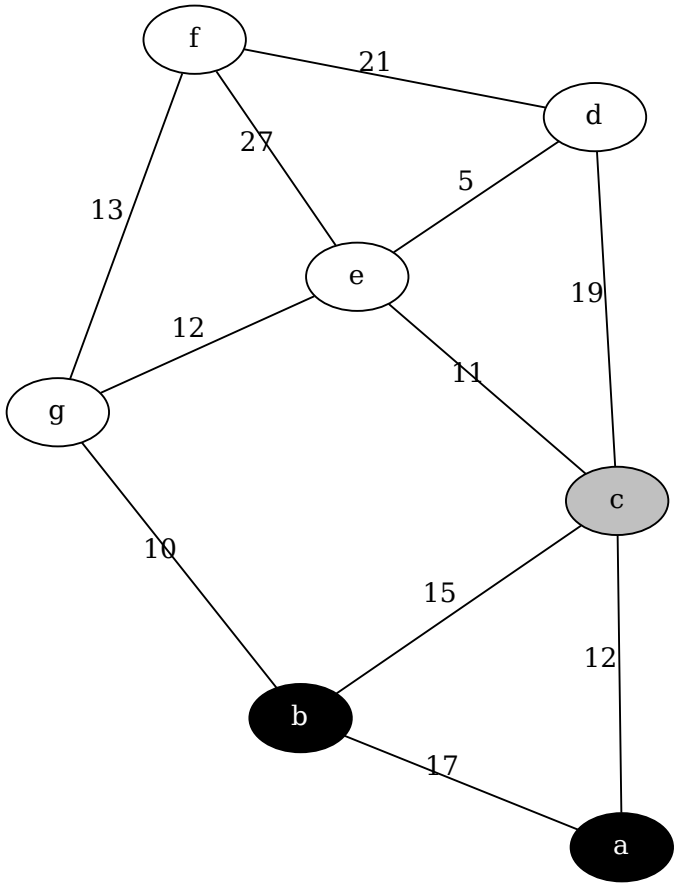
	a	b	c	d	e	f	g
a	0	17	12	∞	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



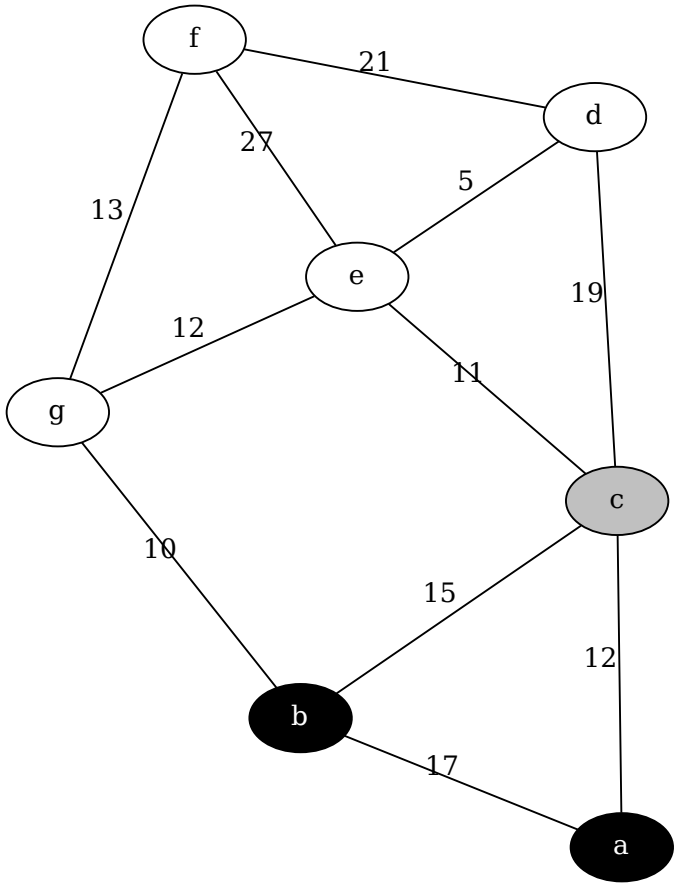
	a	b	c	d	e	f	g
a	0	17	12	31	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



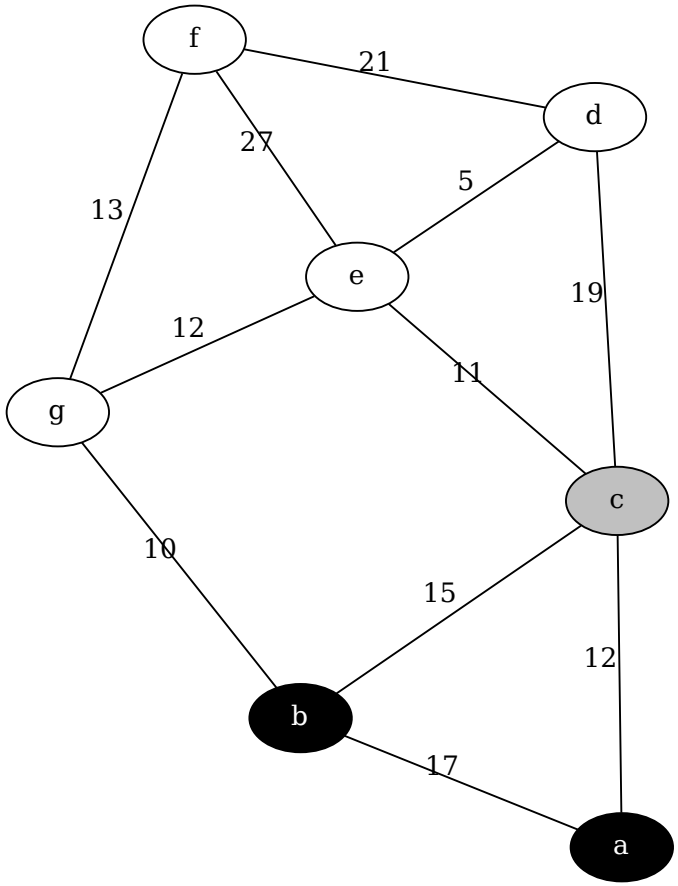
	a	b	c	d	e	f	g
a	0	17	12	31	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



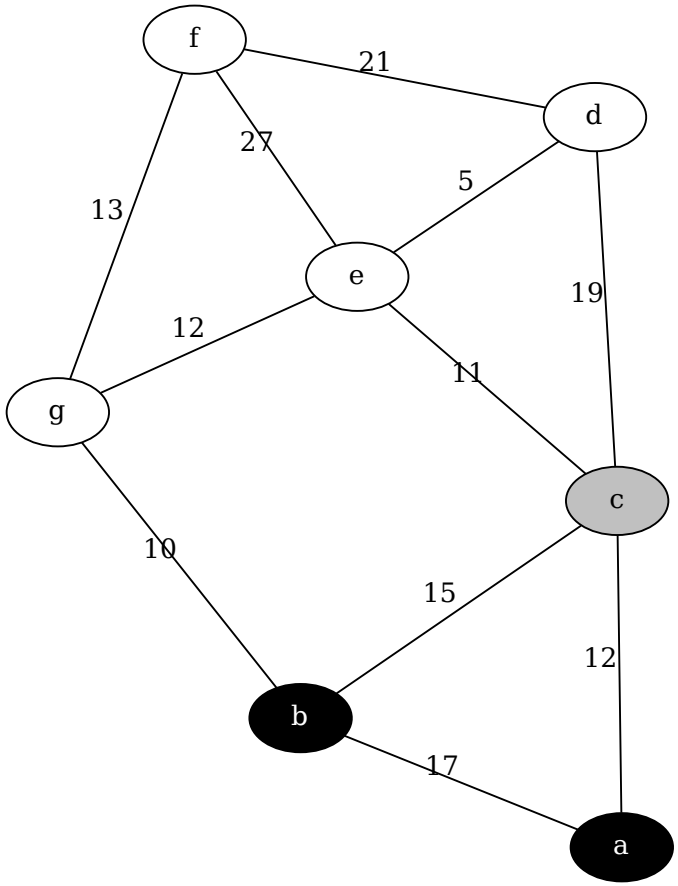
	a	b	c	d	e	f	g
a	0	17	12	31	∞	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



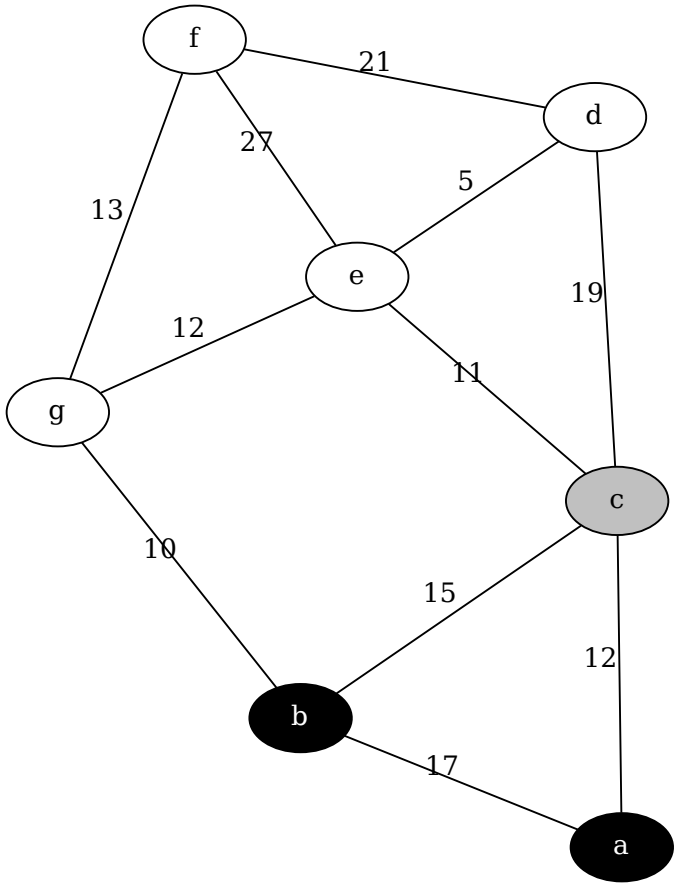
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



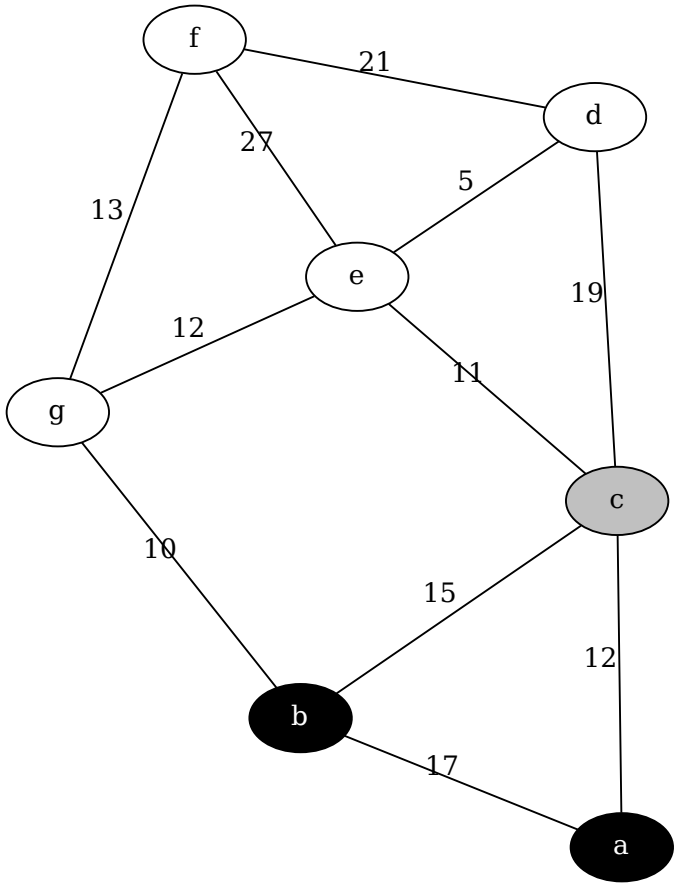
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



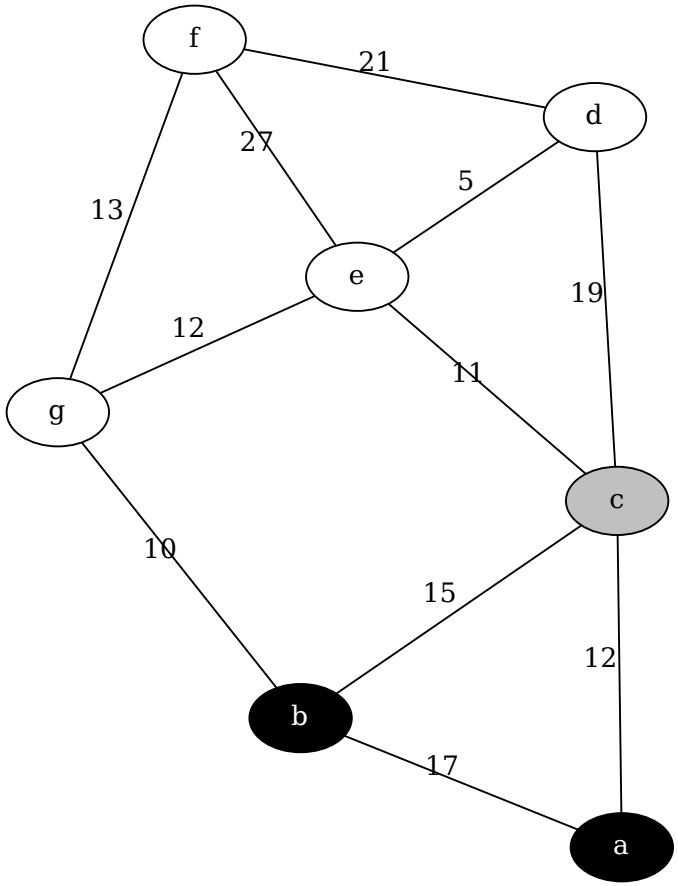
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



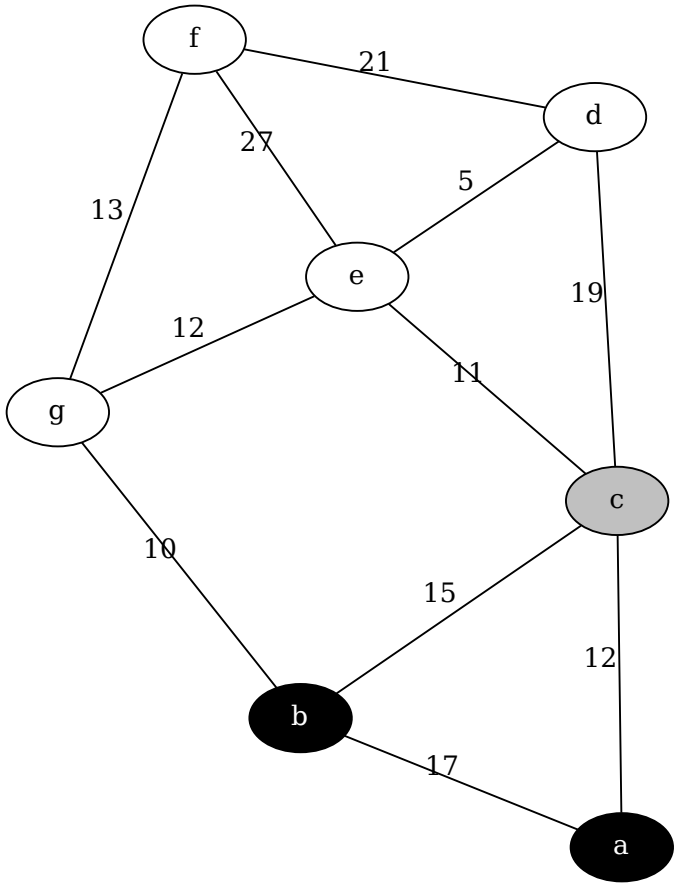
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



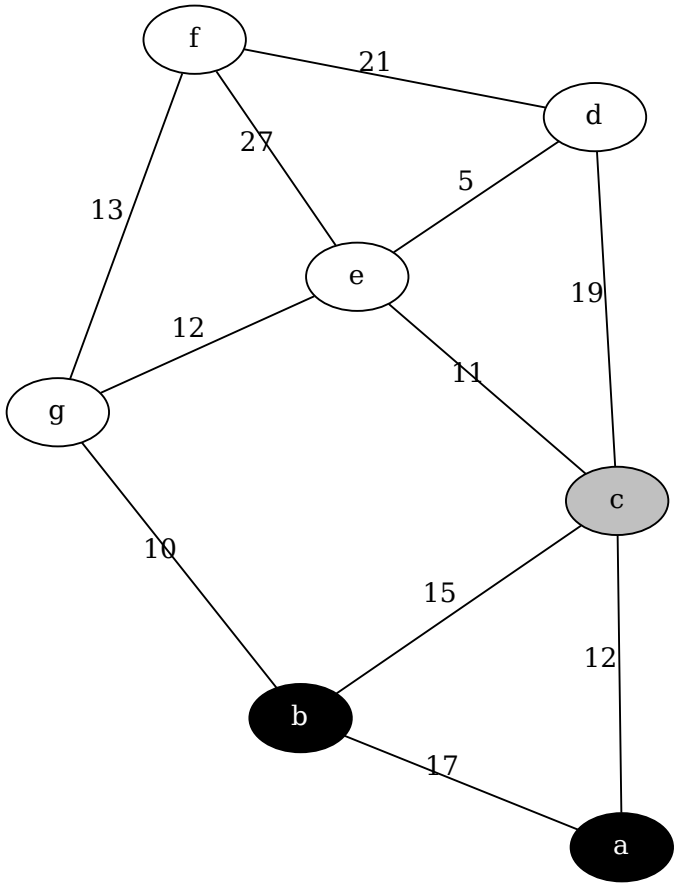
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	∞	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



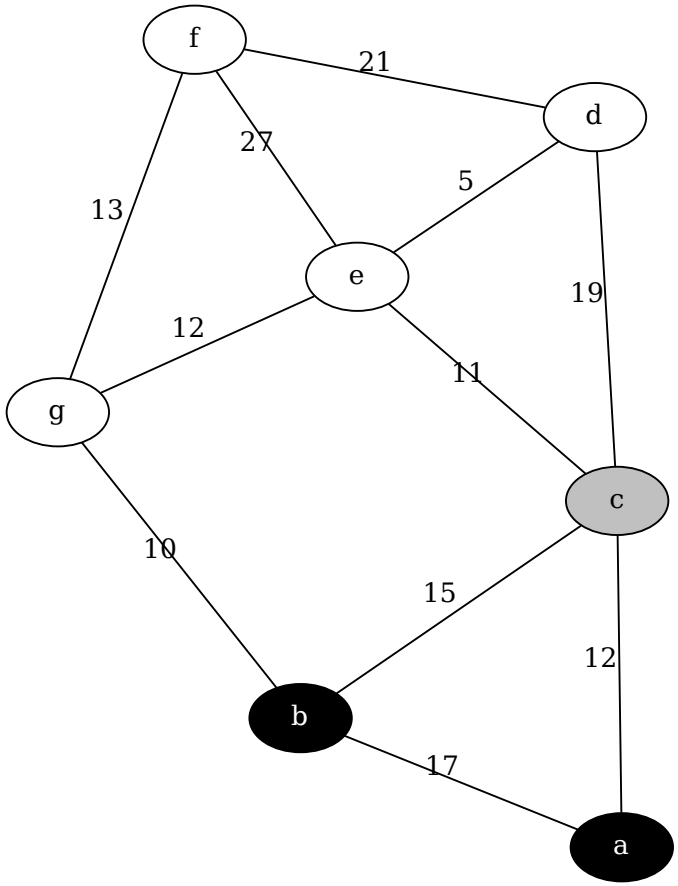
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



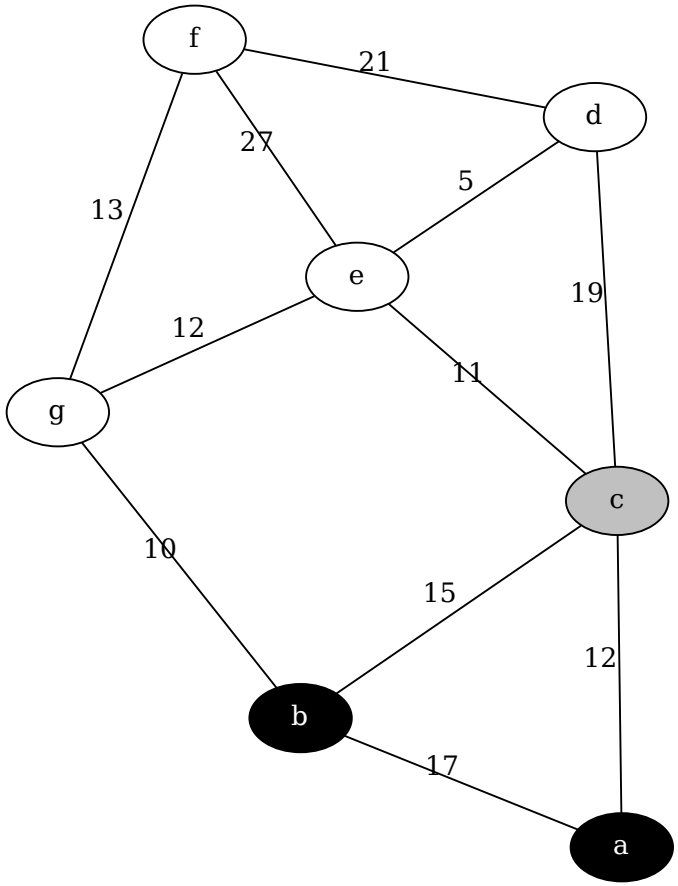
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



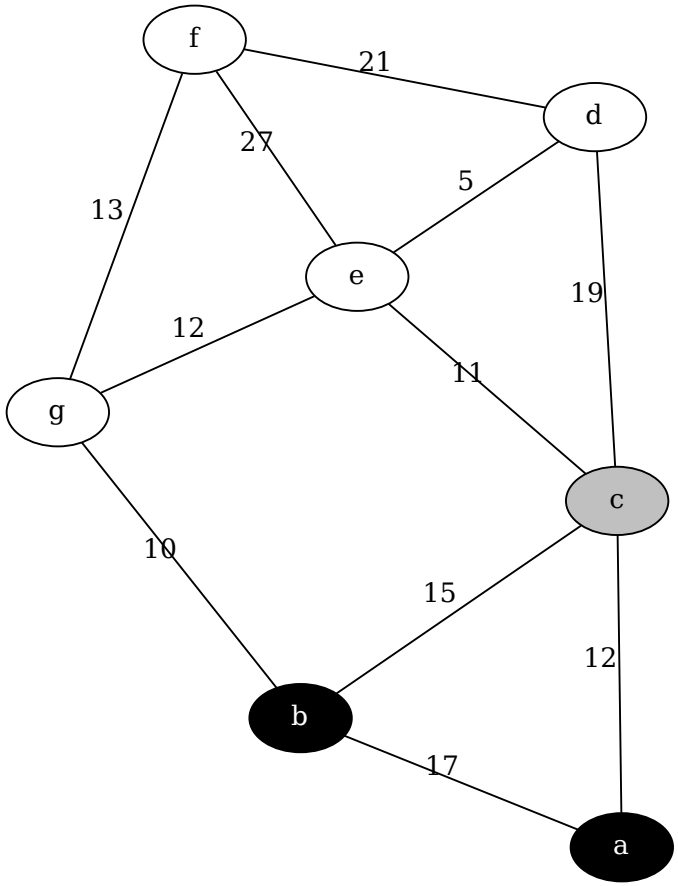
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	∞	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



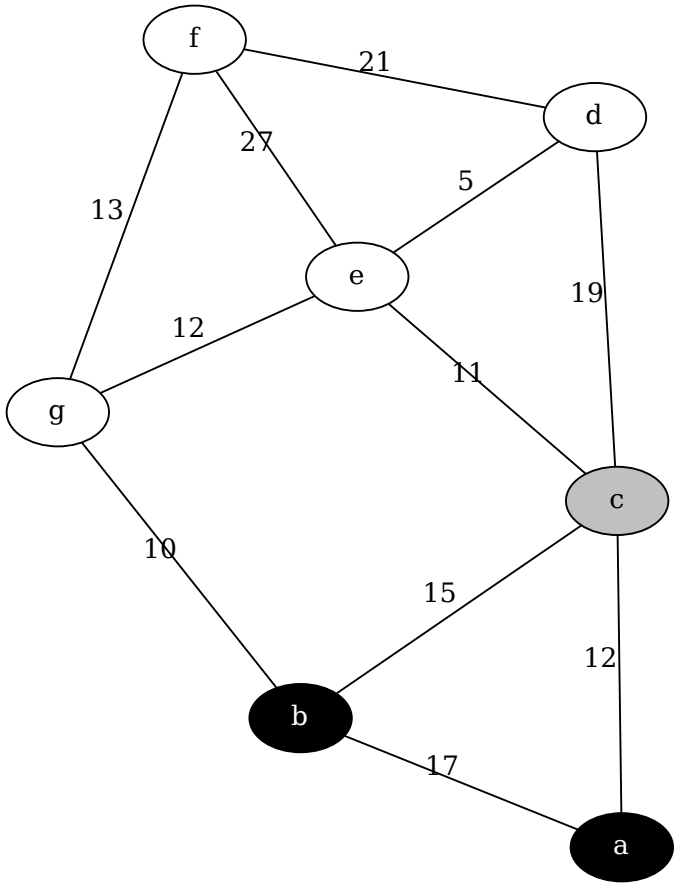
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



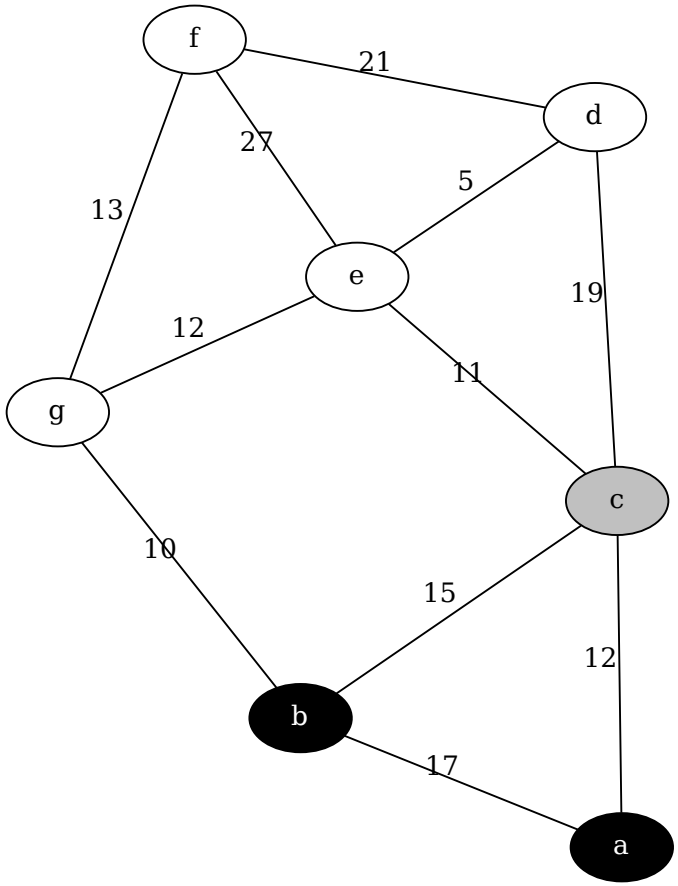
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



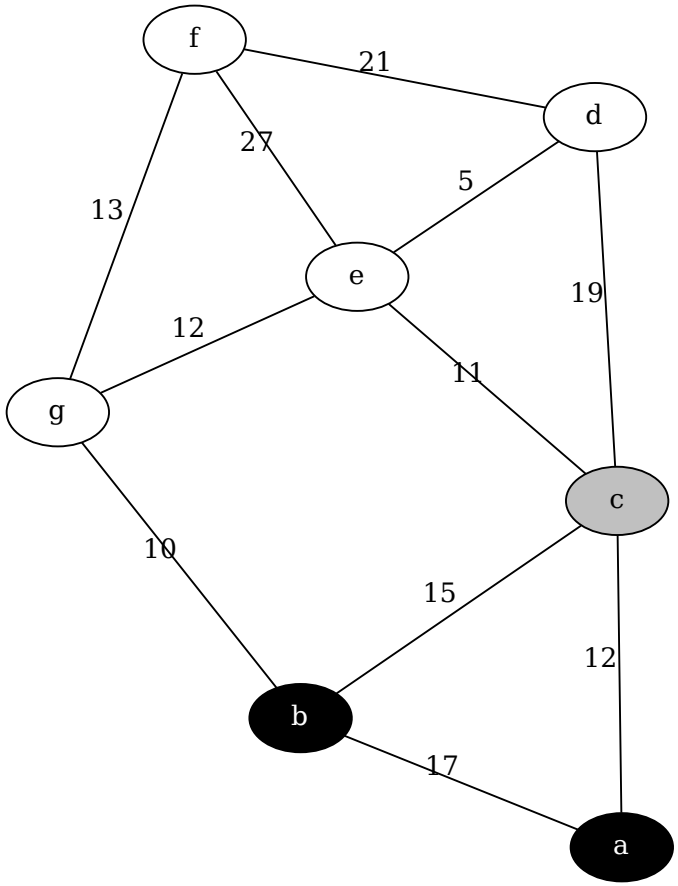
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



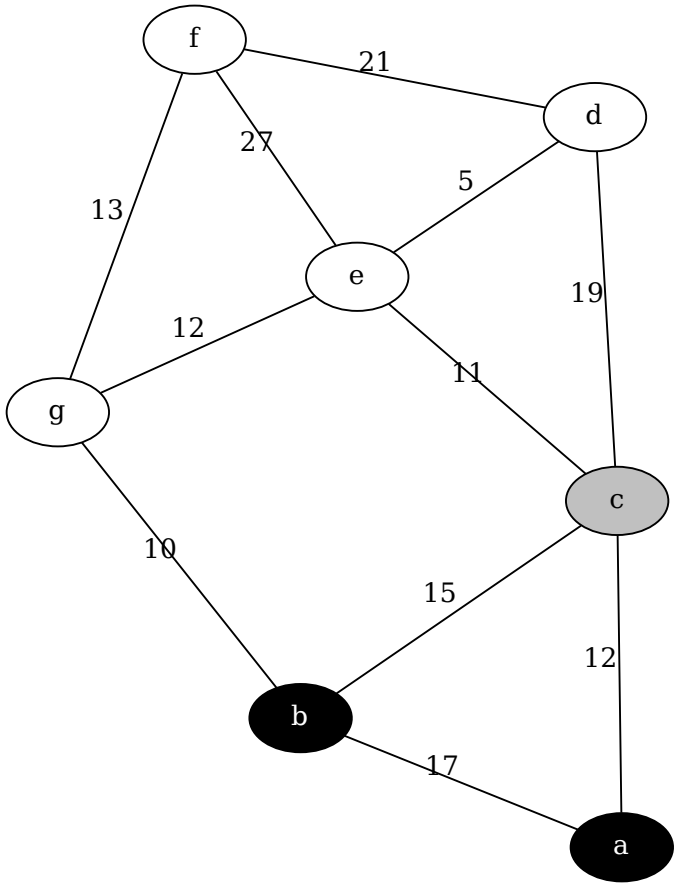
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



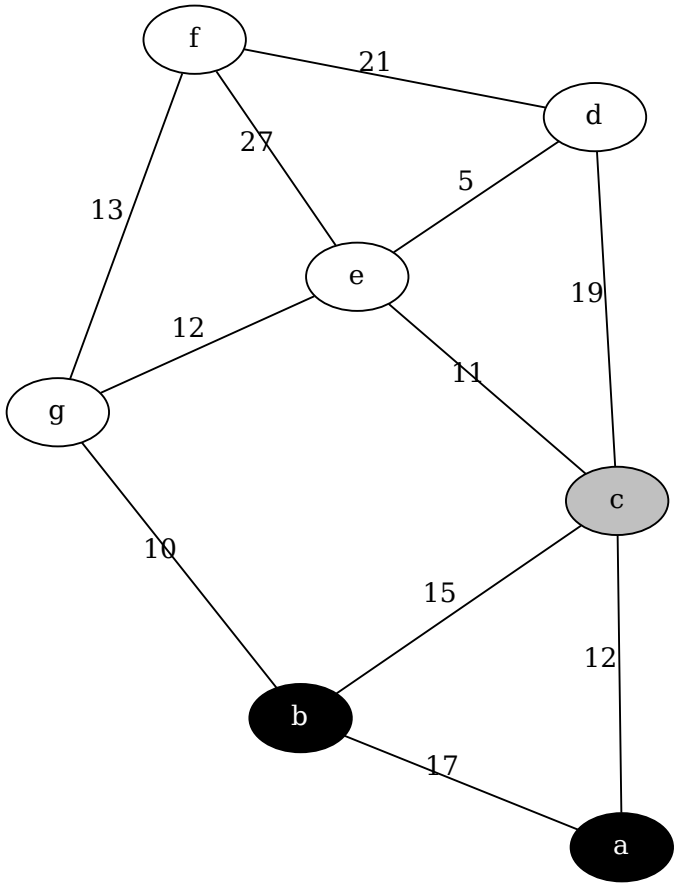
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



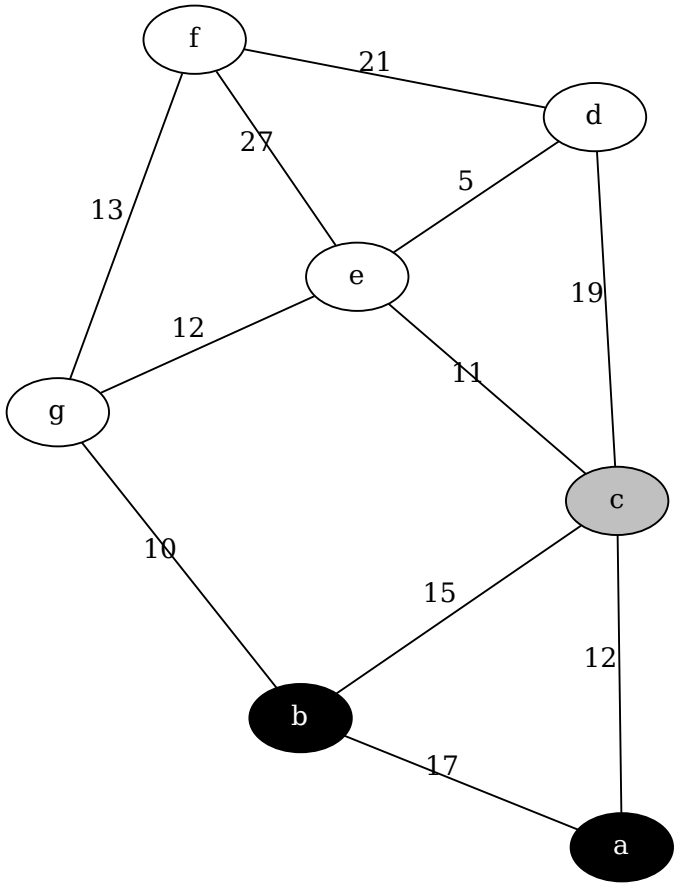
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



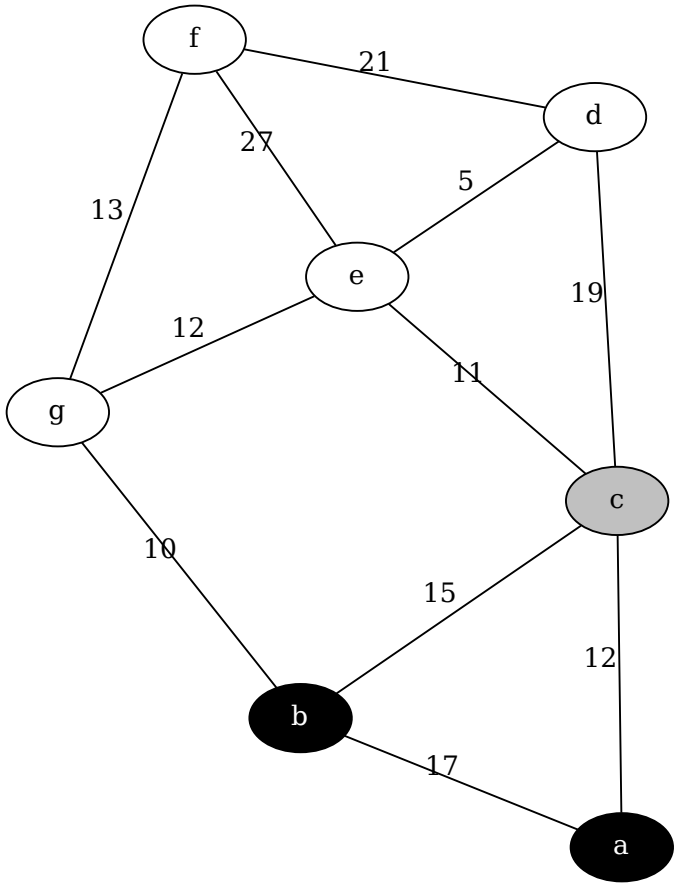
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	∞
e					0	27	12
f						0	13
g							0

Example



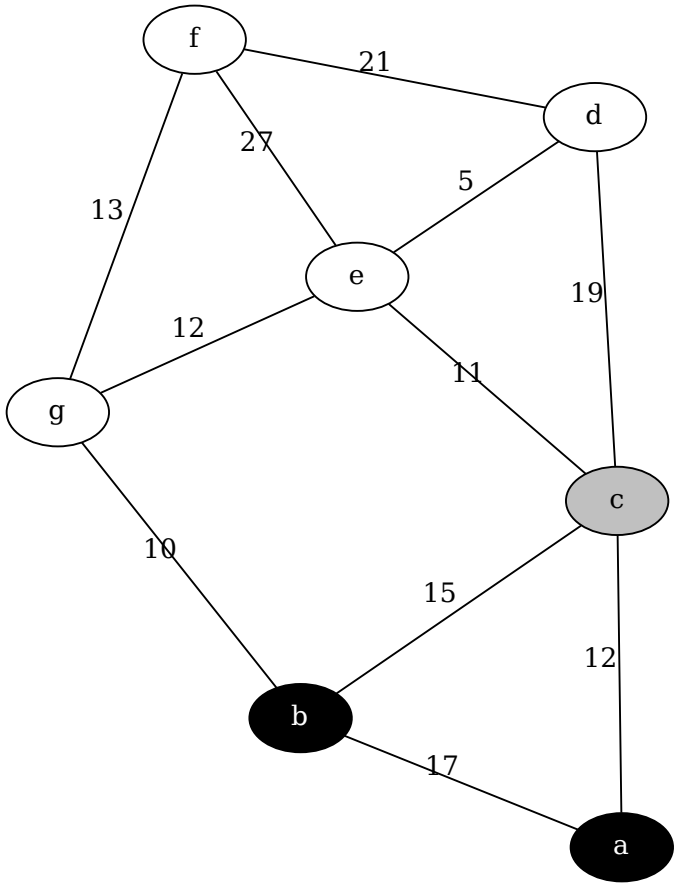
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



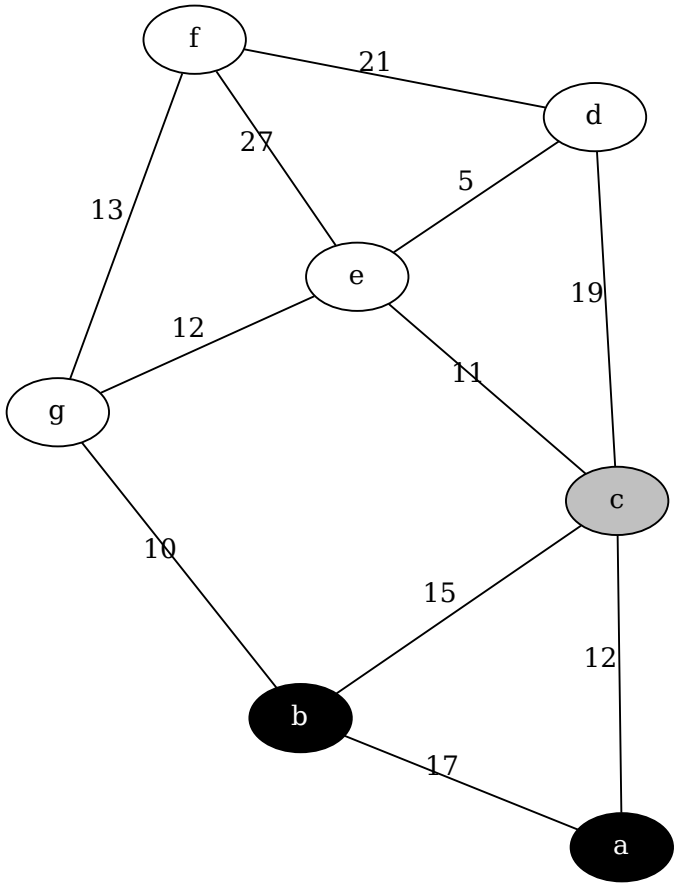
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



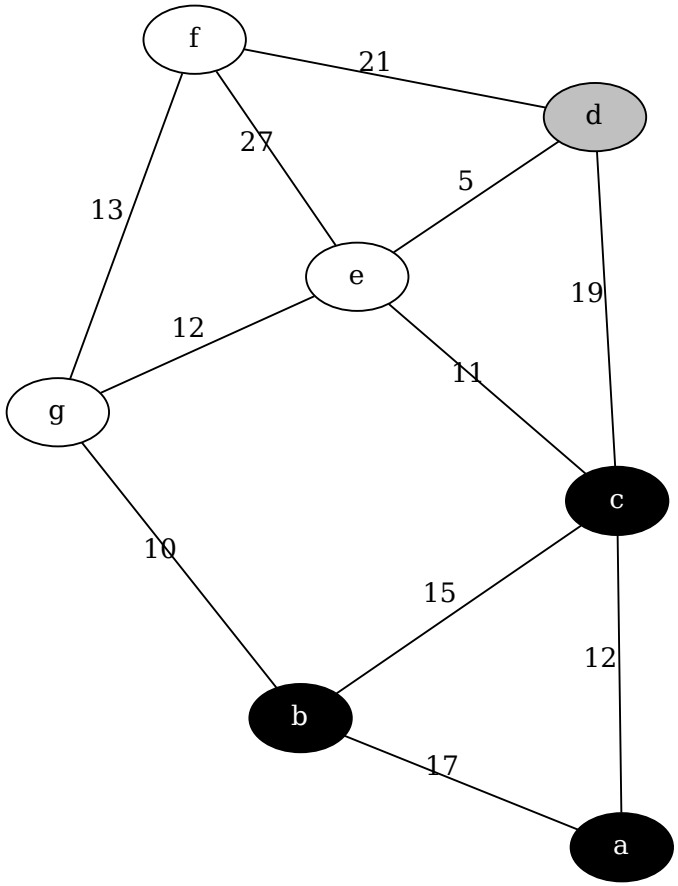
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



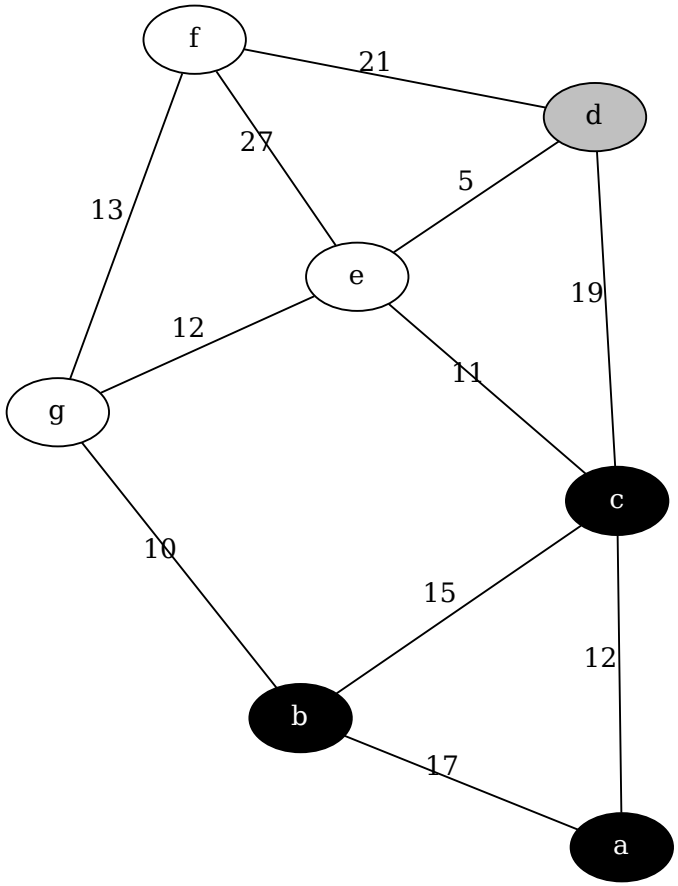
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



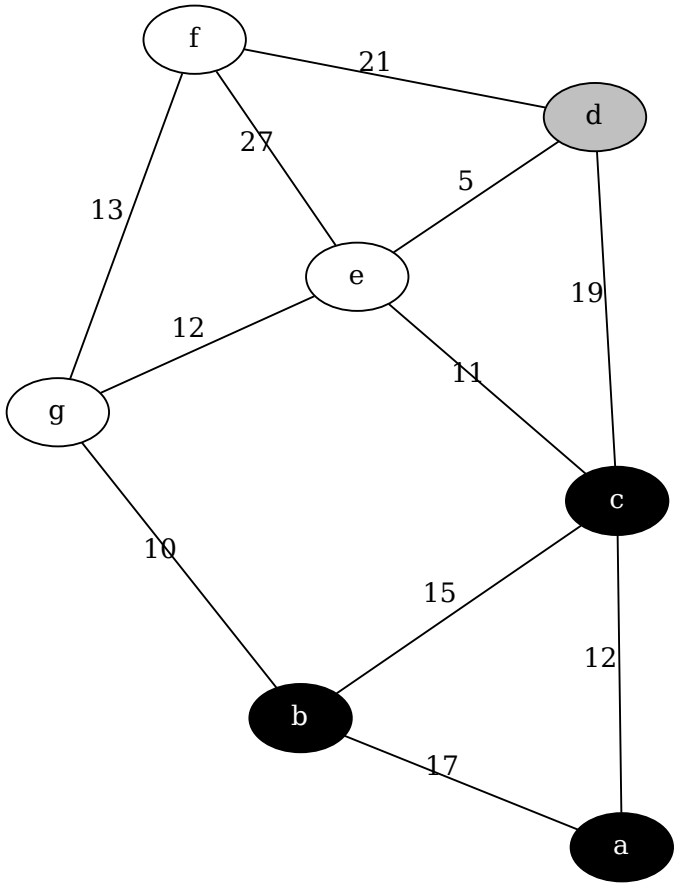
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



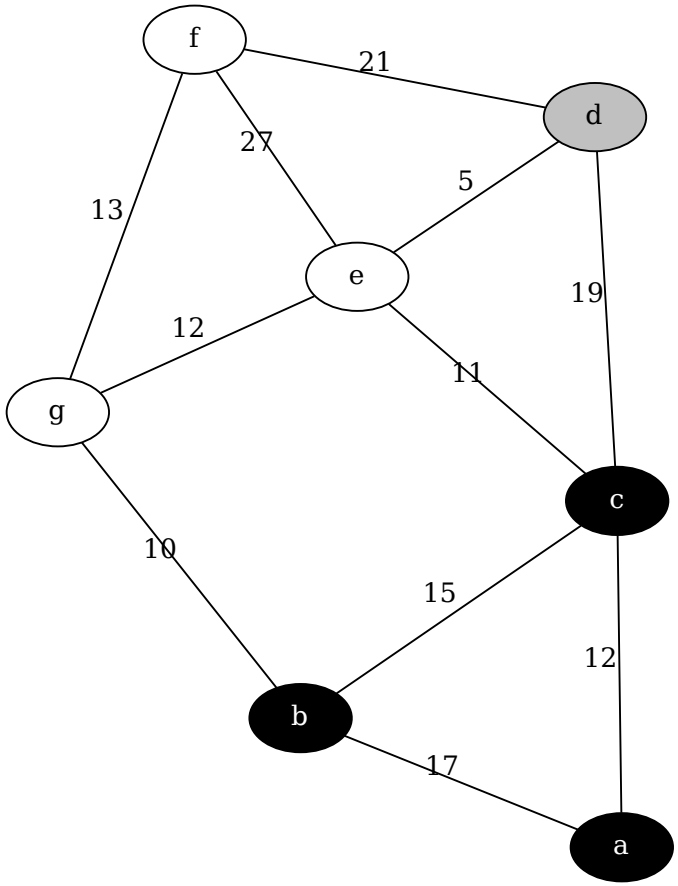
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



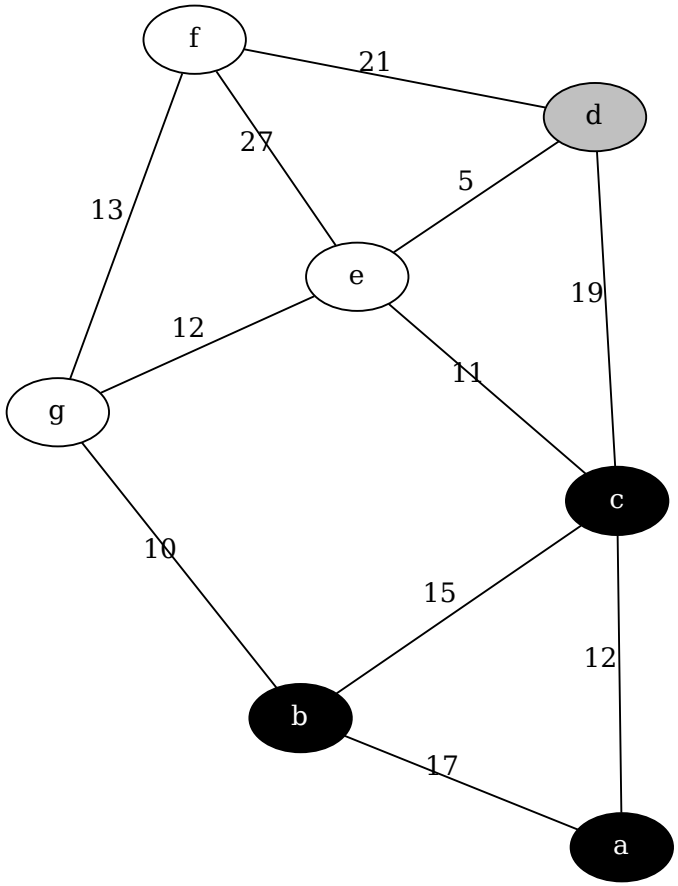
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



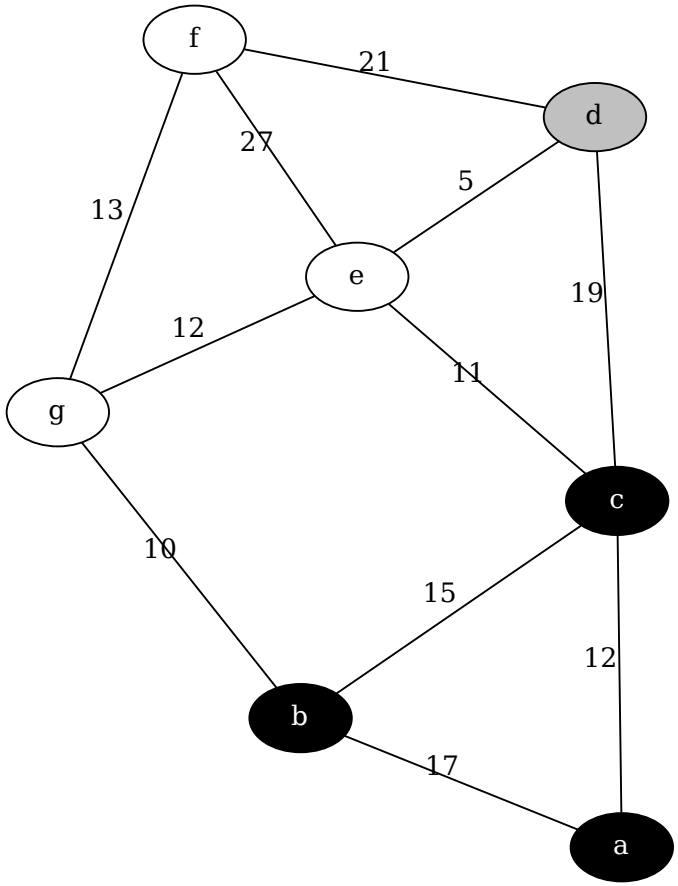
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



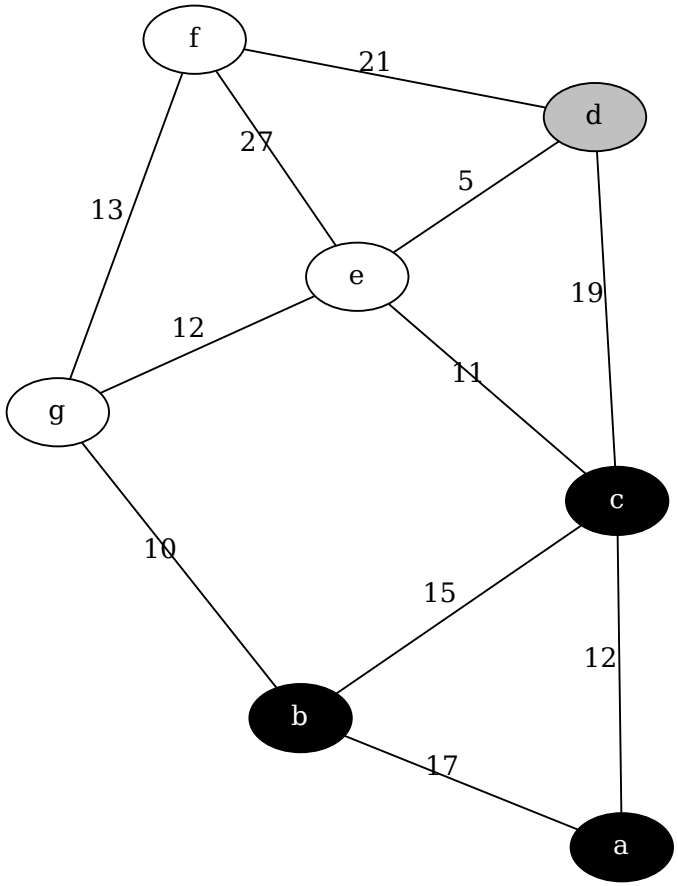
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



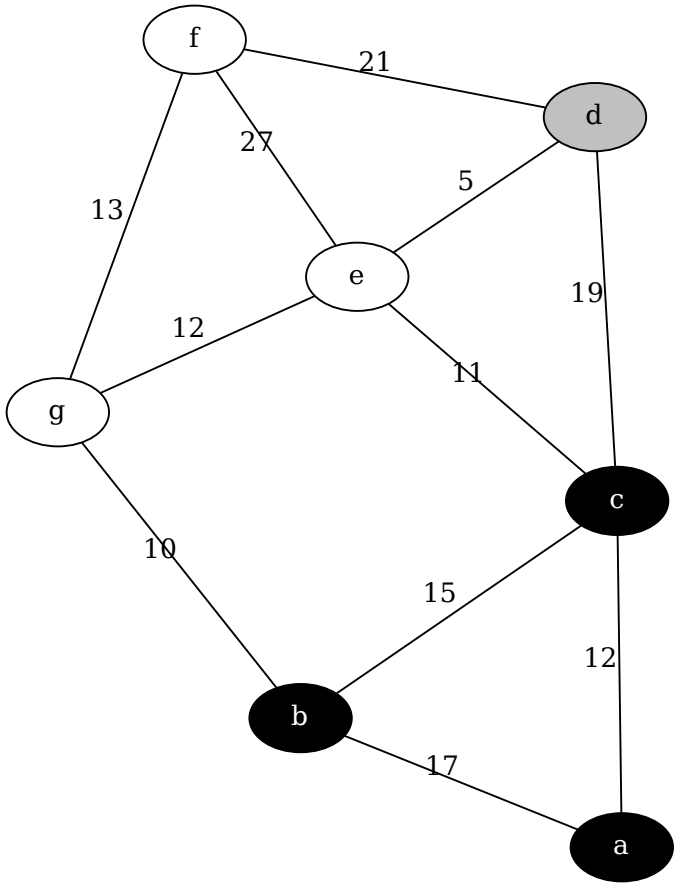
	a	b	c	d	e	f	g
a	0	17	12	31	23	∞	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



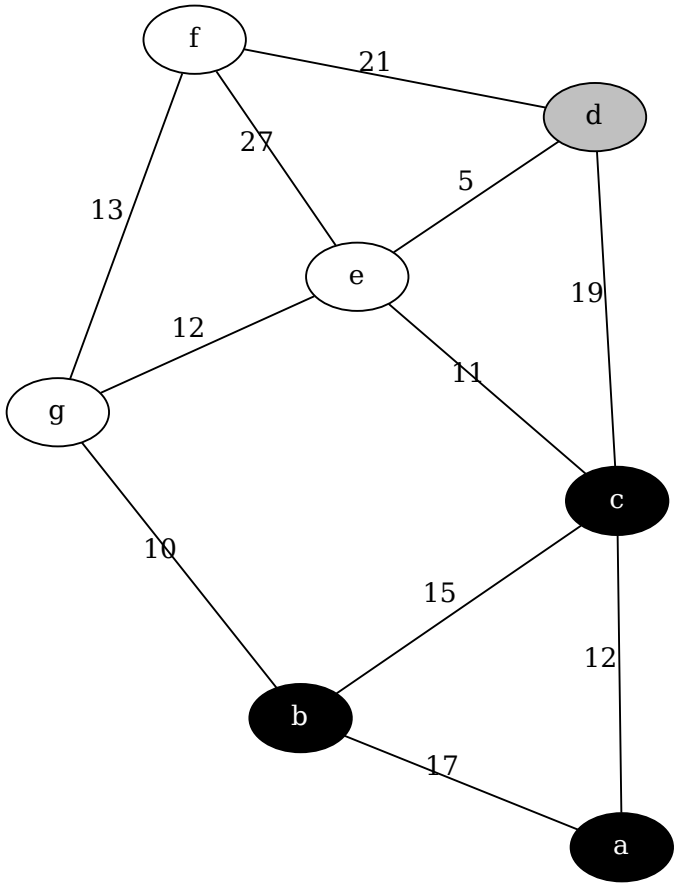
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



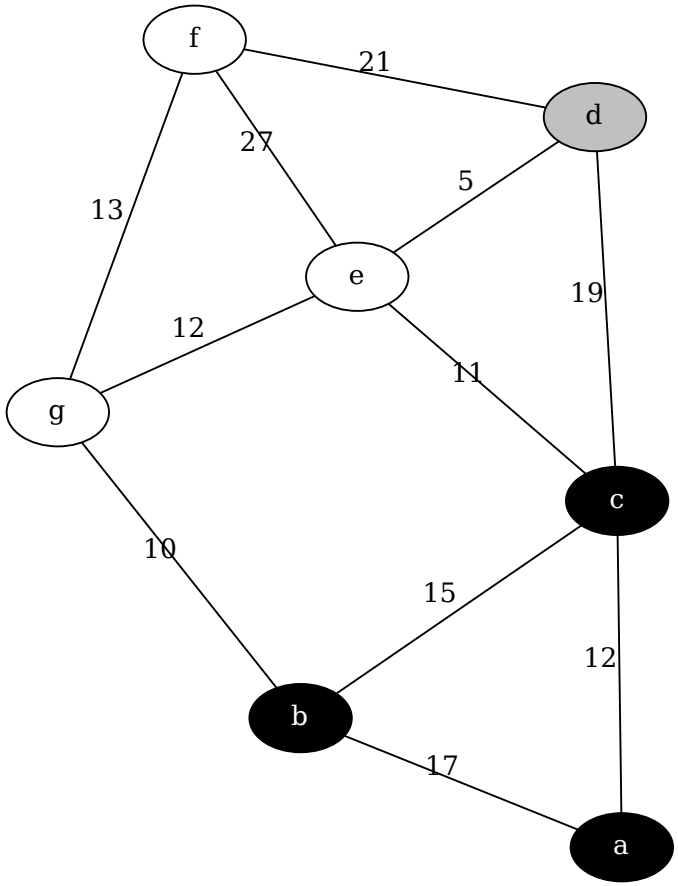
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



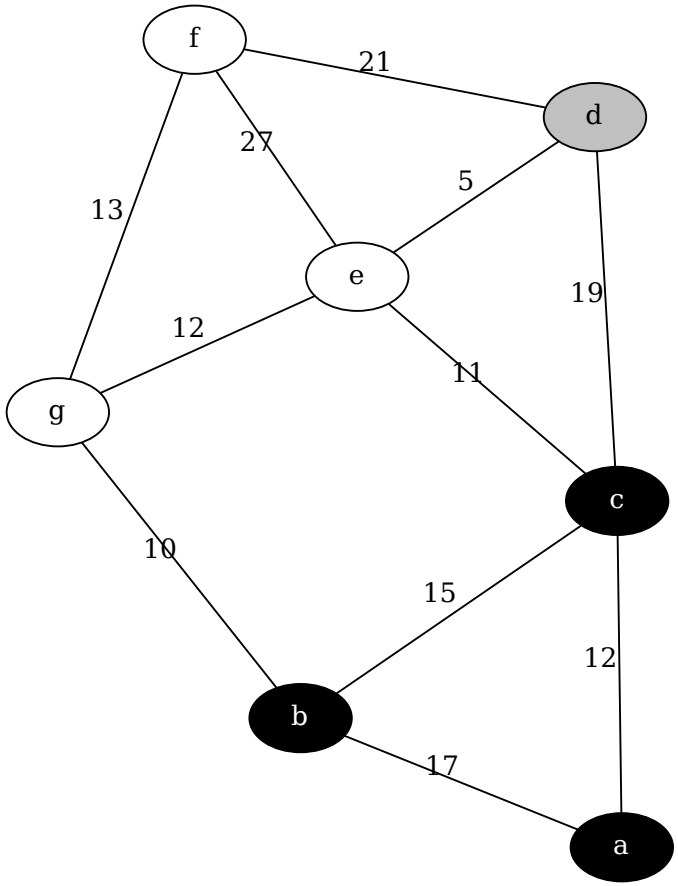
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



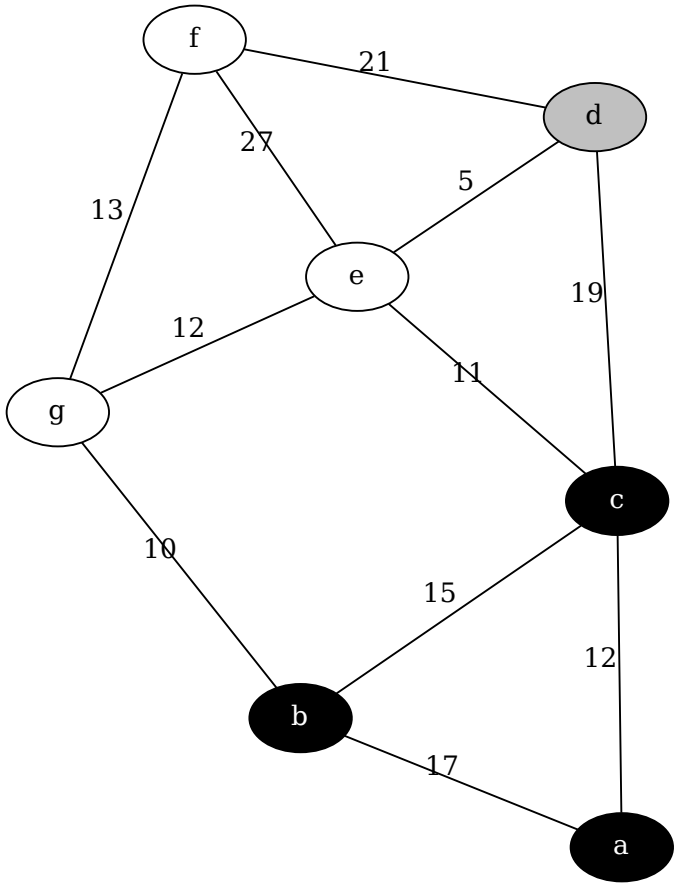
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



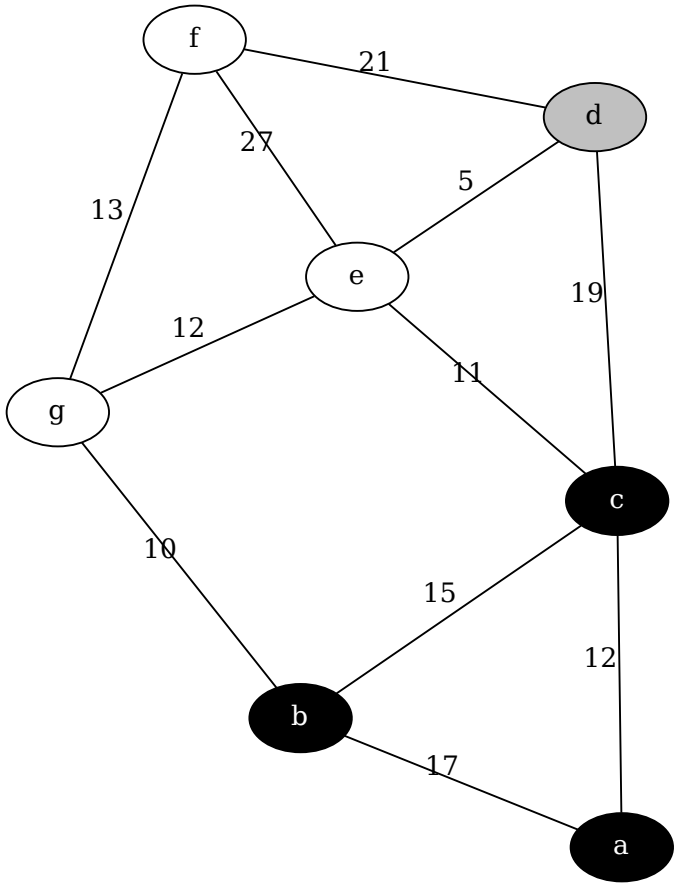
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



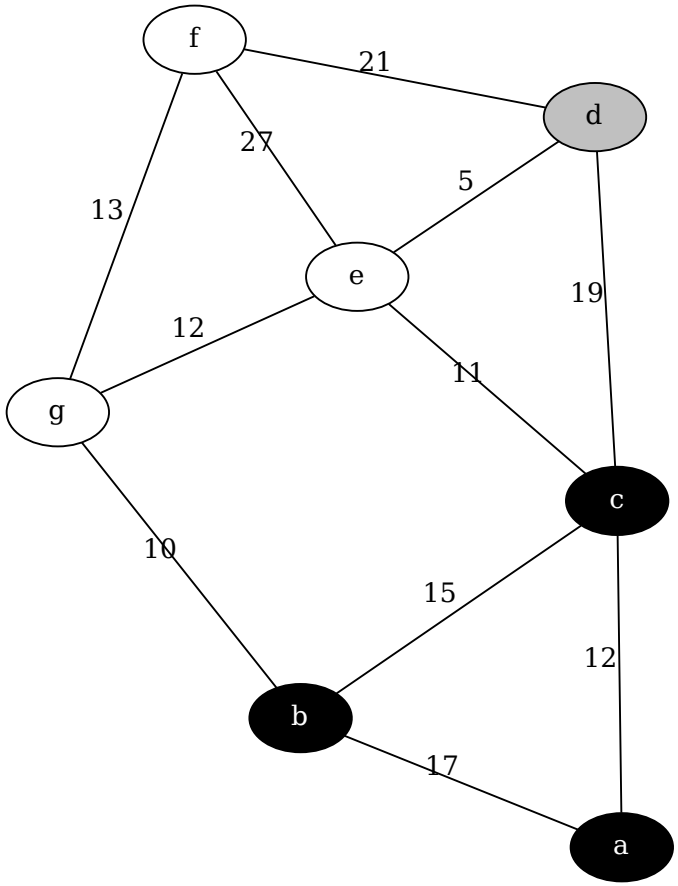
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	∞	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



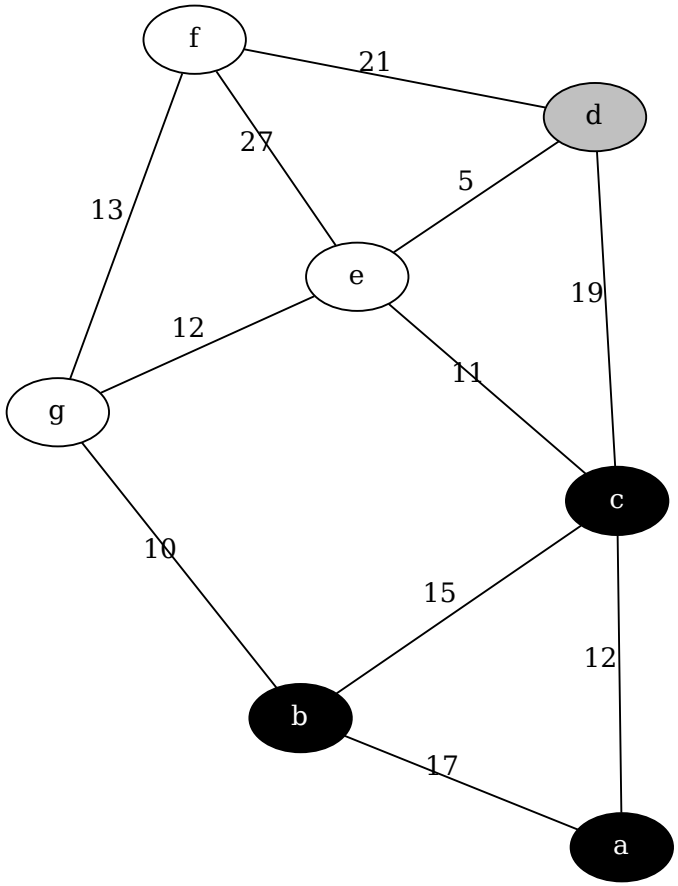
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



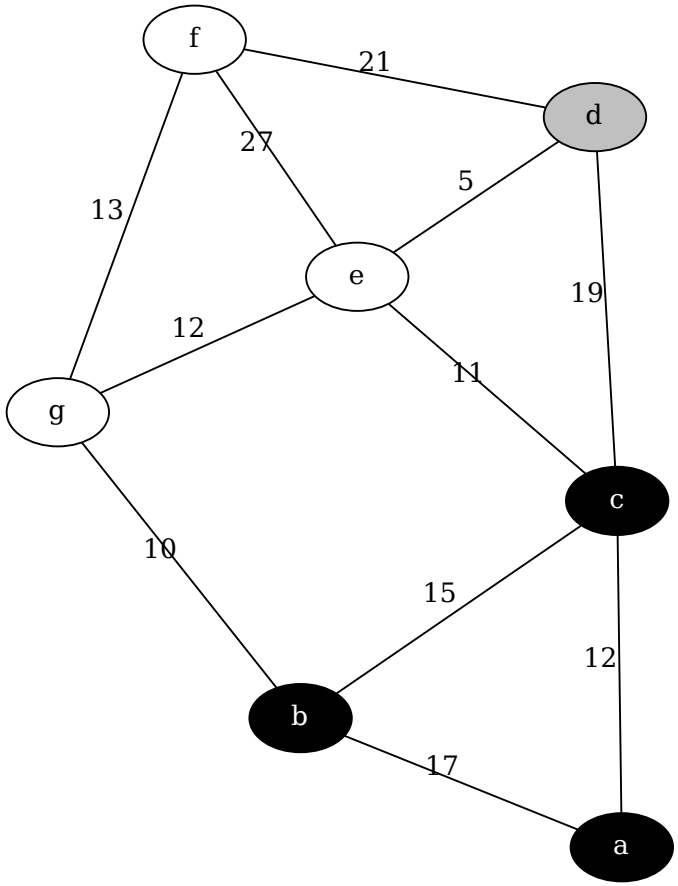
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



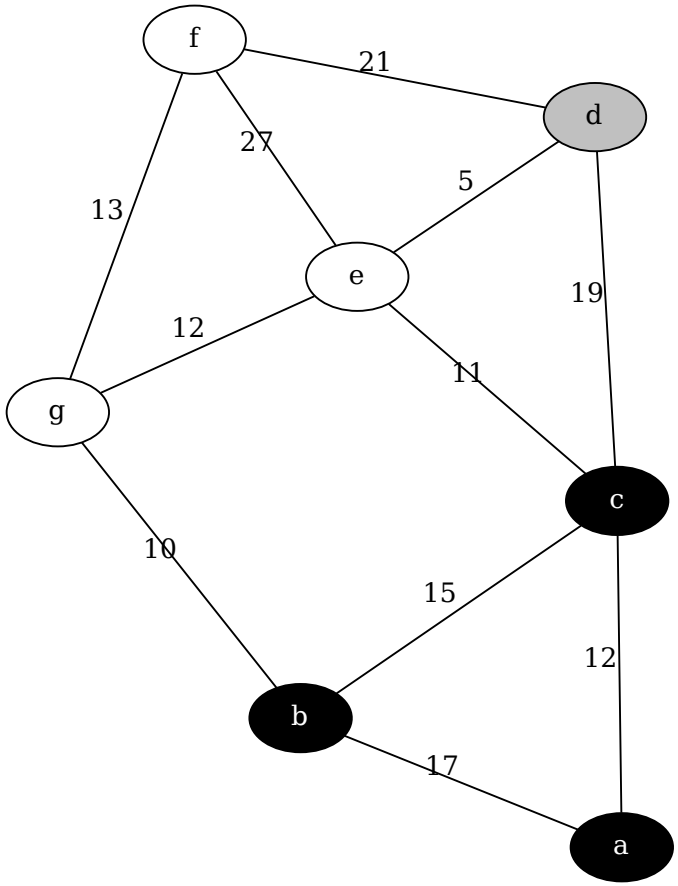
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



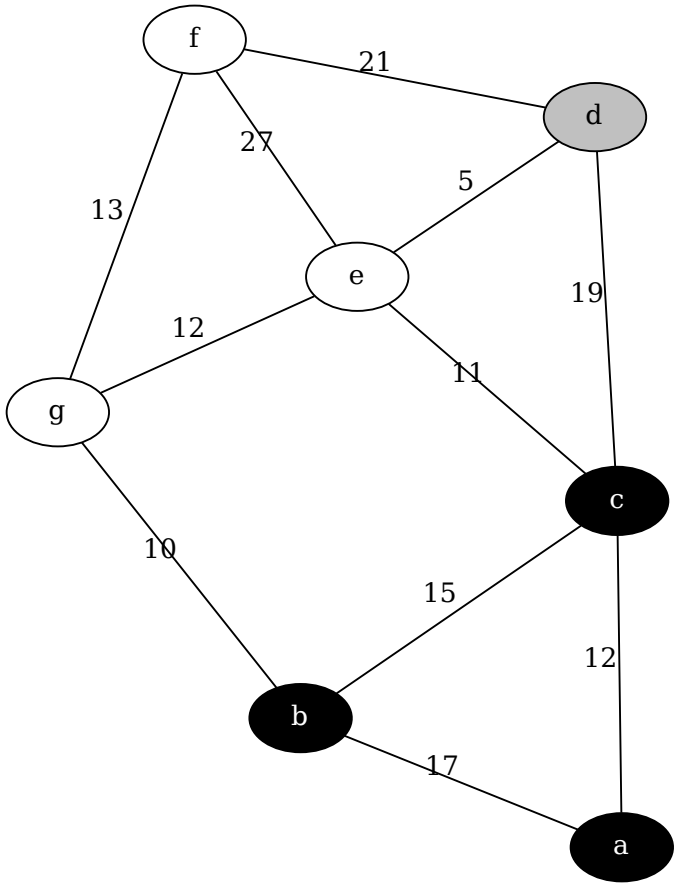
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



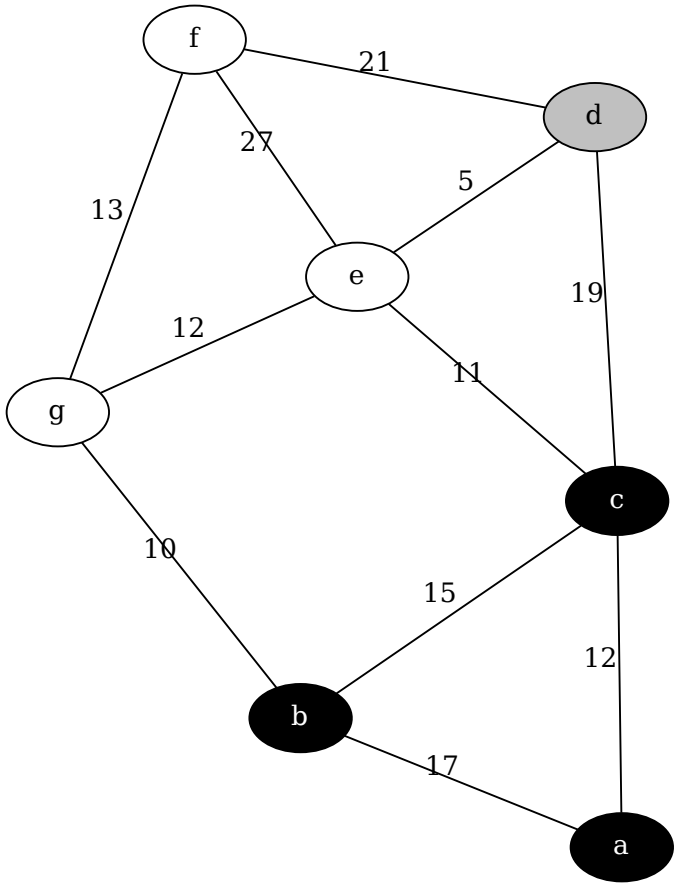
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	∞	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



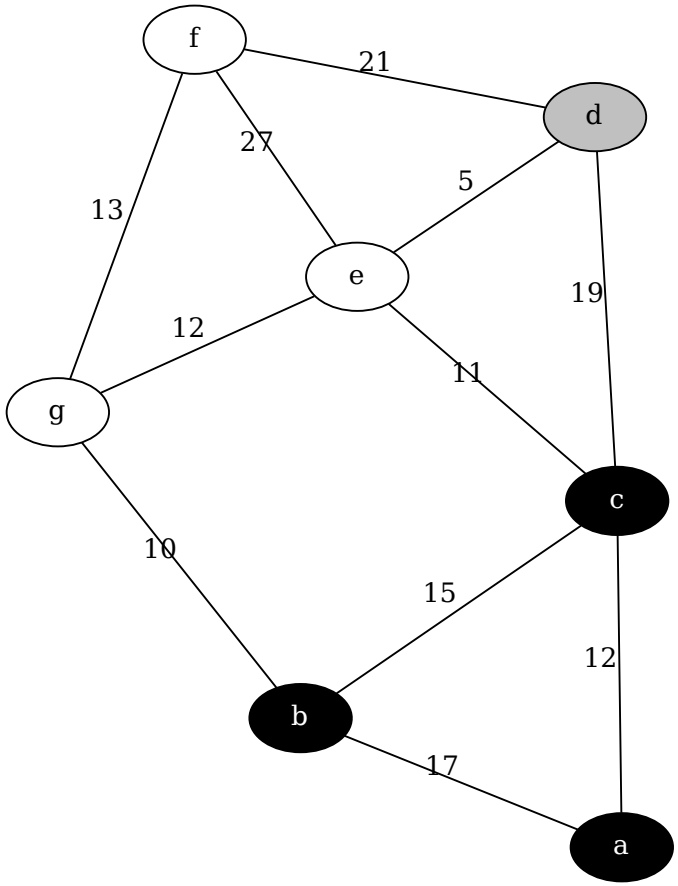
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



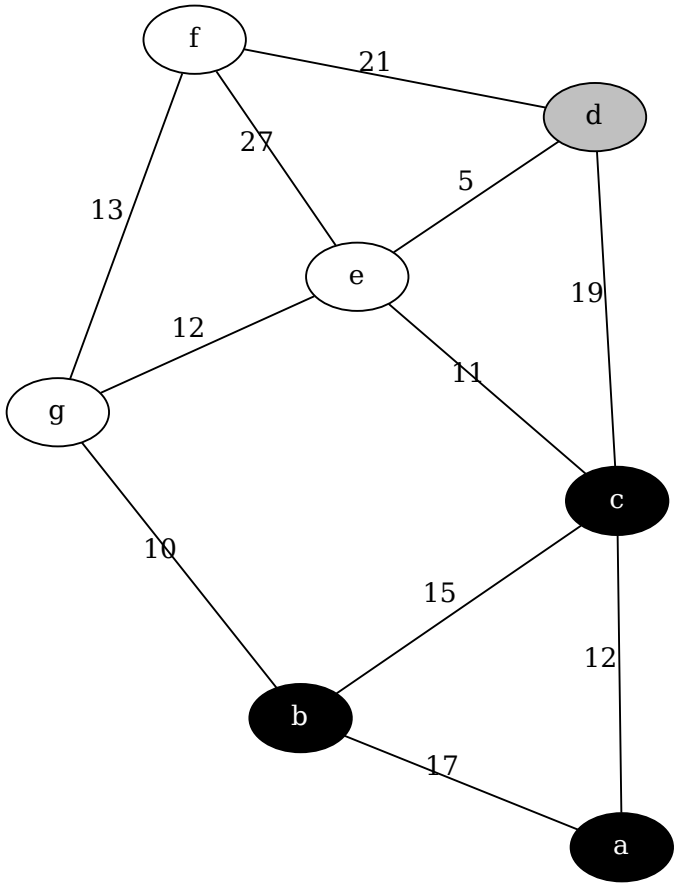
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



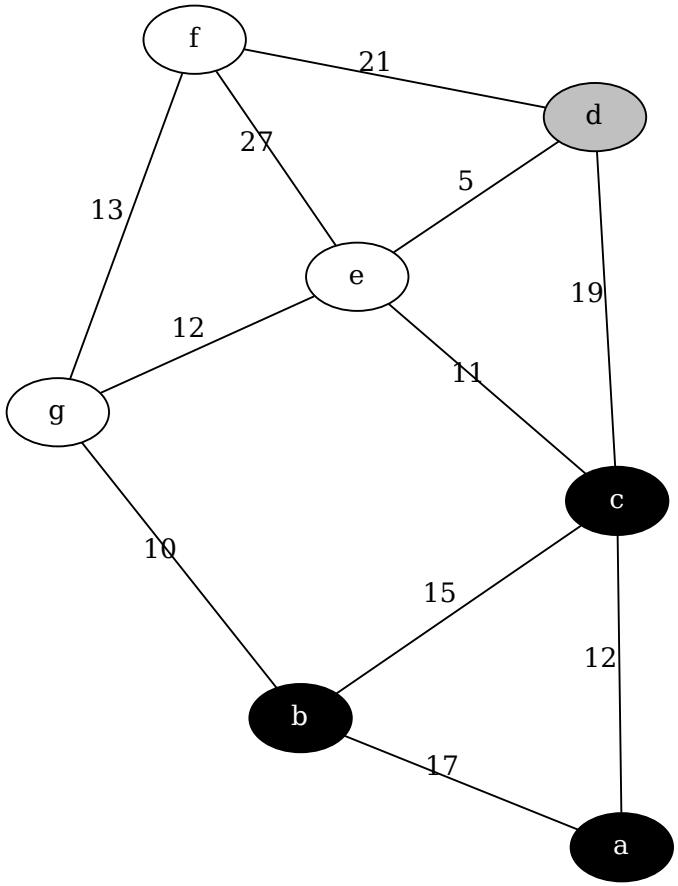
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



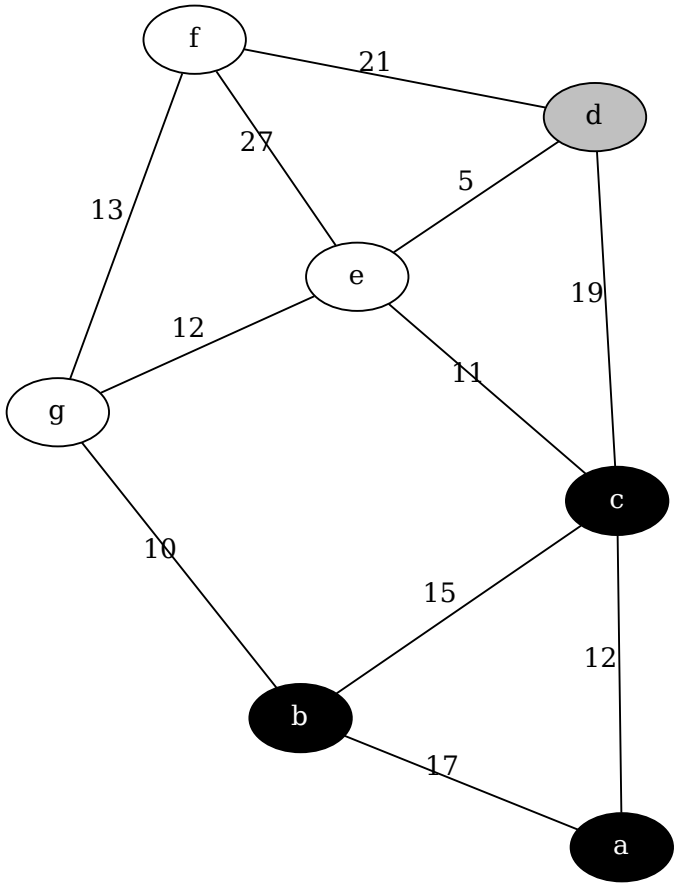
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	27	12
f						0	13
g							0

Example



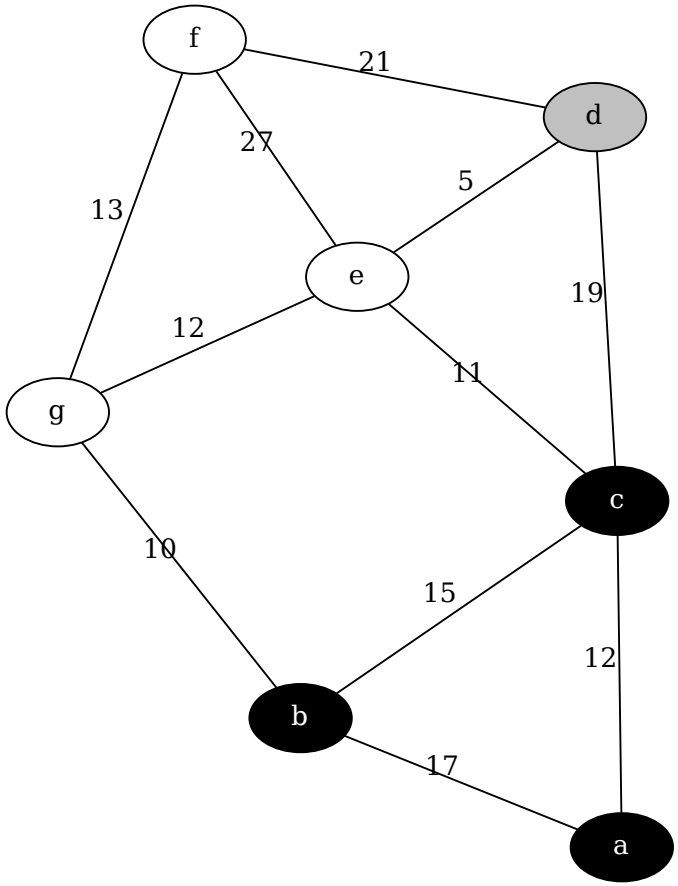
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



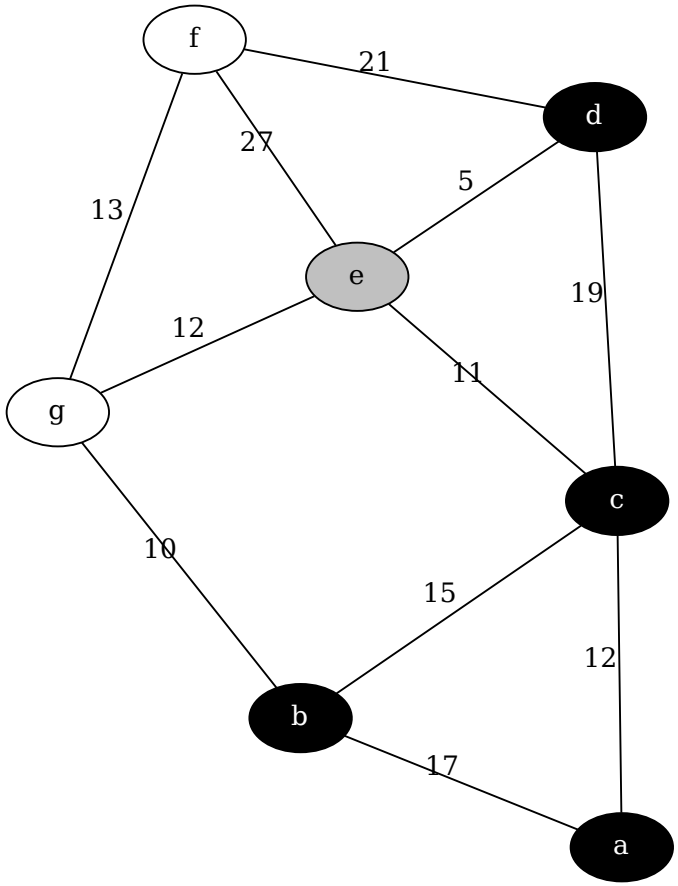
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



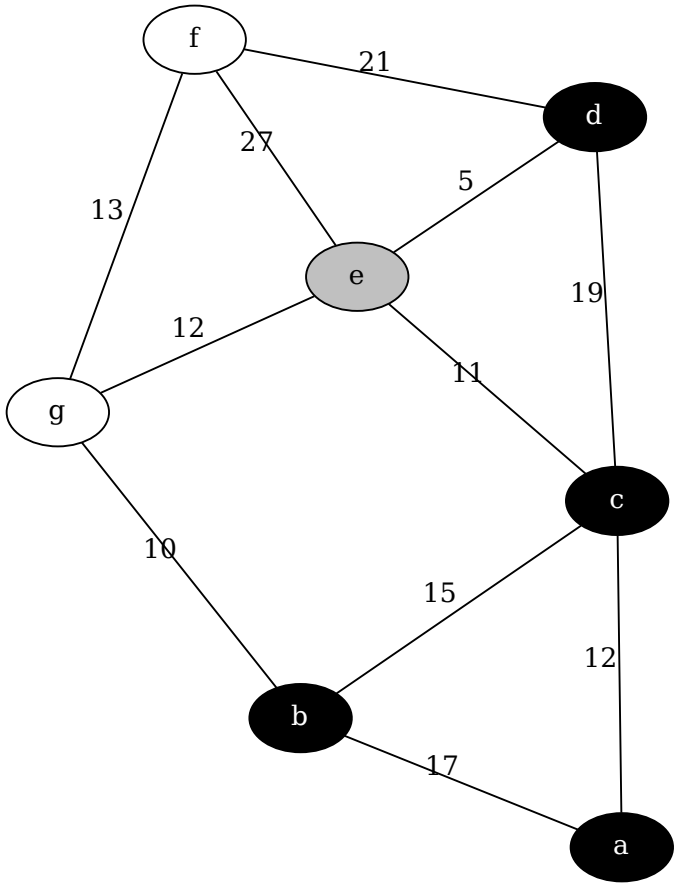
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



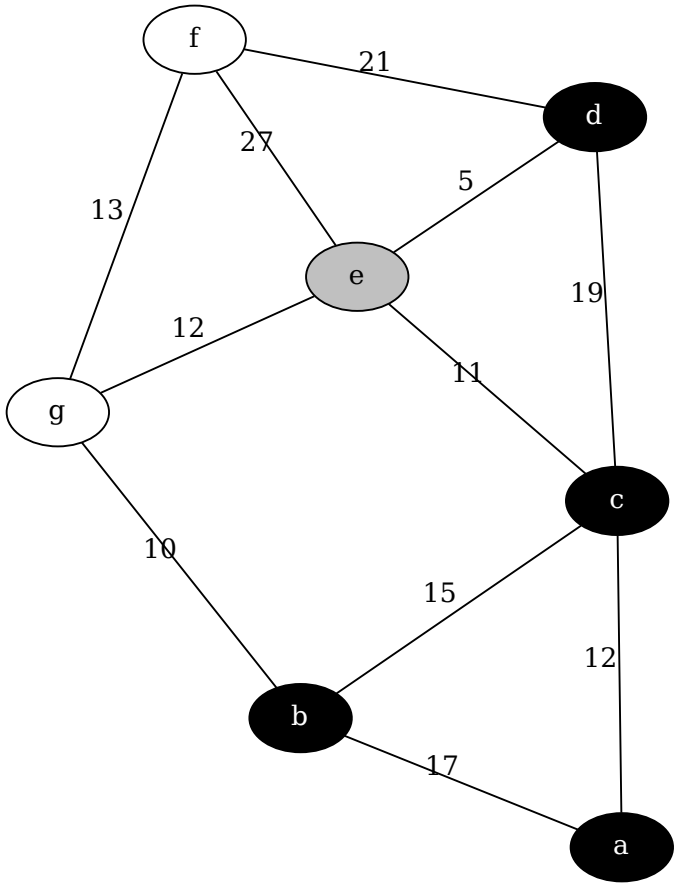
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



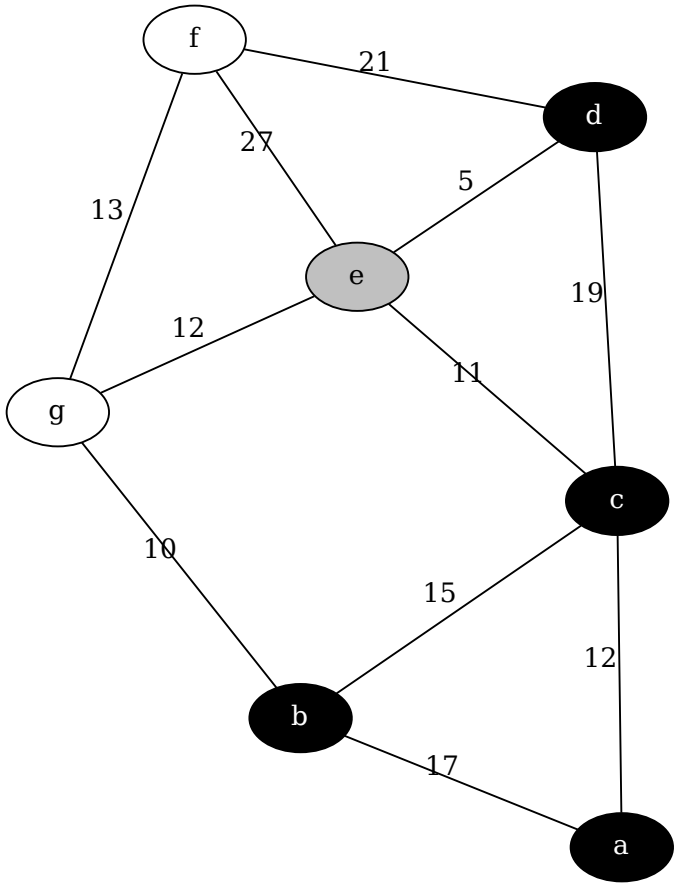
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



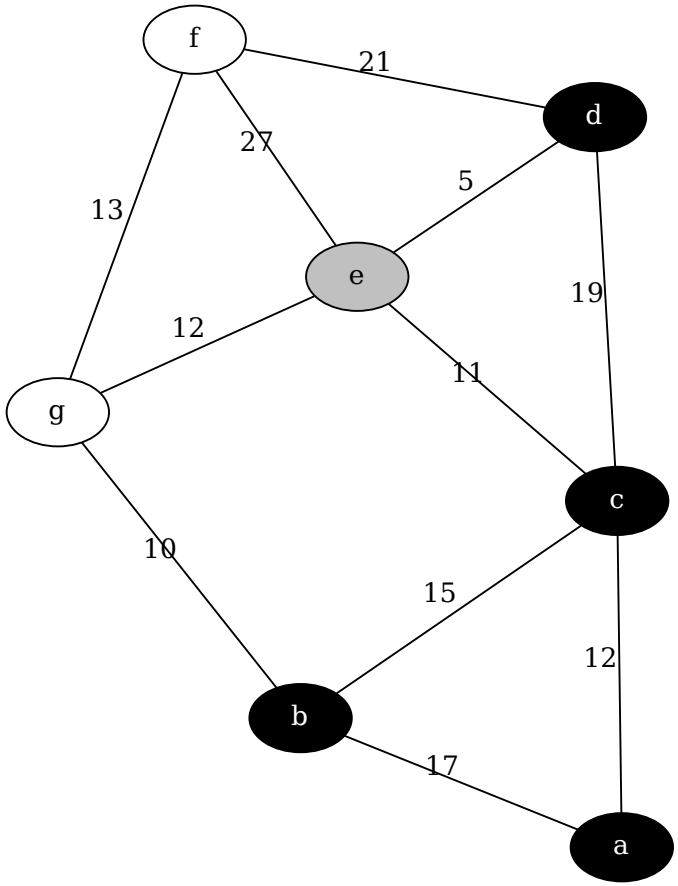
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



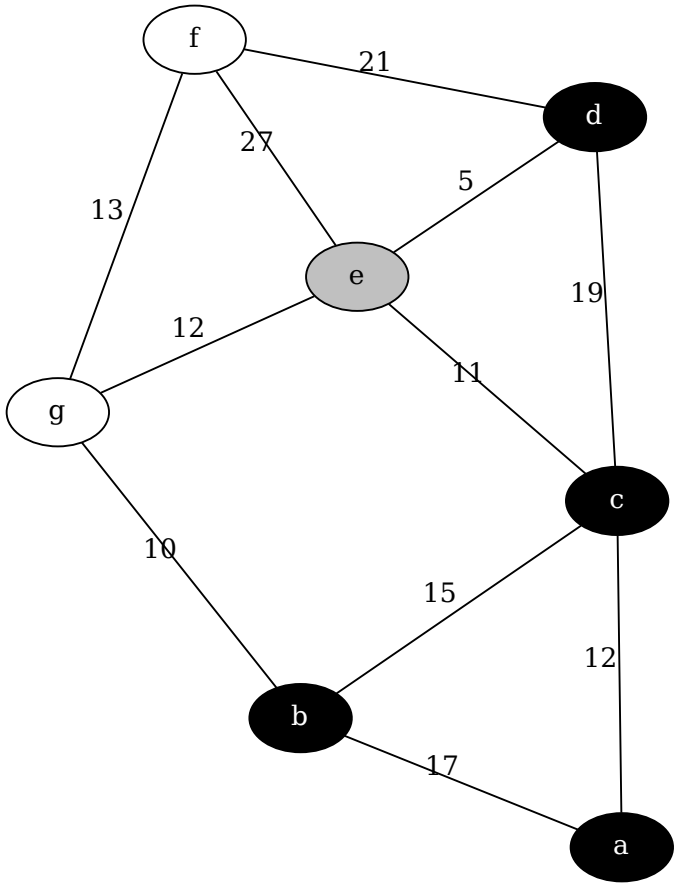
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



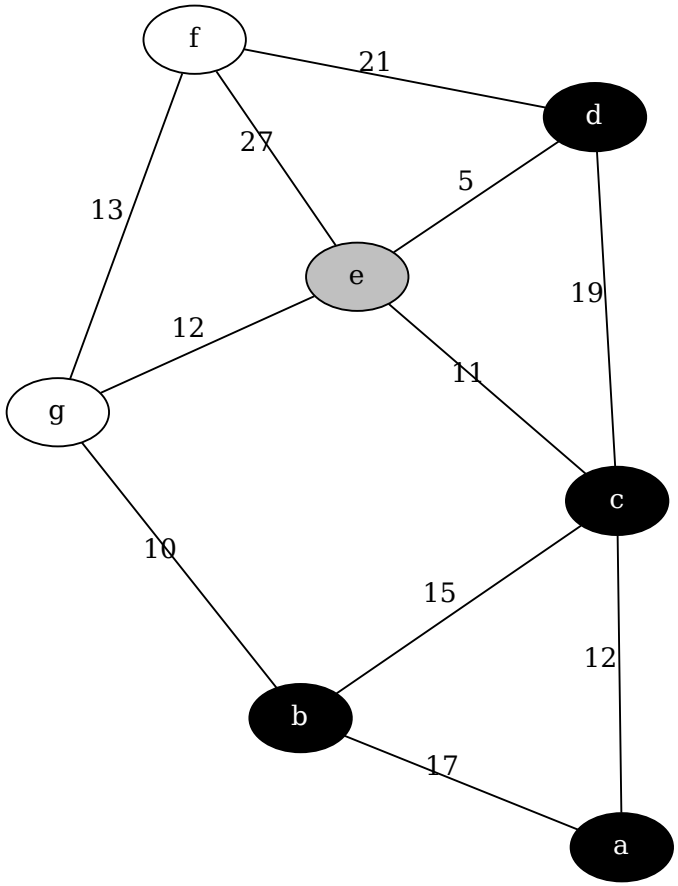
	a	b	c	d	e	f	g
a	0	17	12	31	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



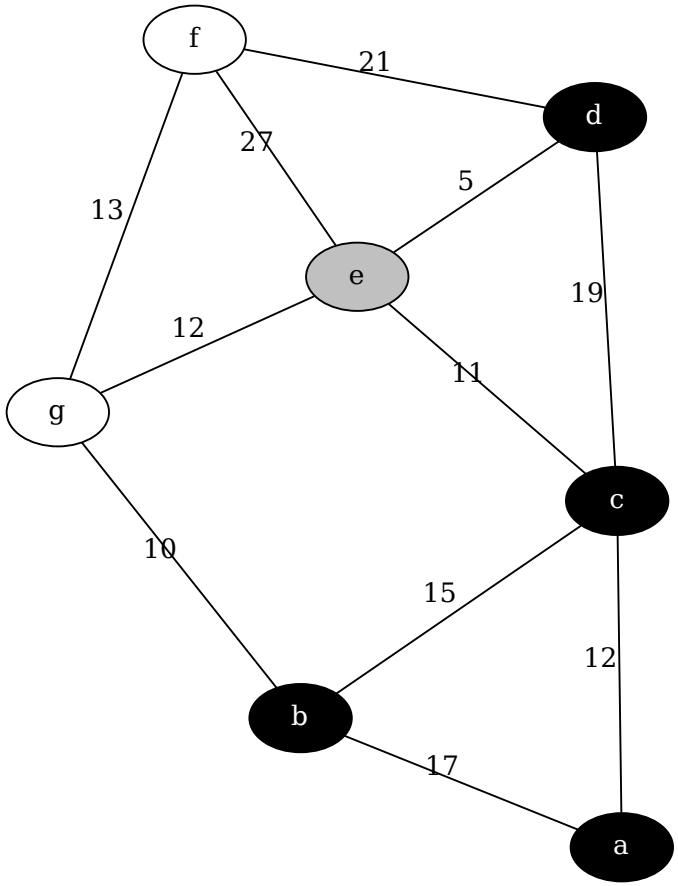
	a	b	c	d	e	f	g
a	0	17	12	28	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



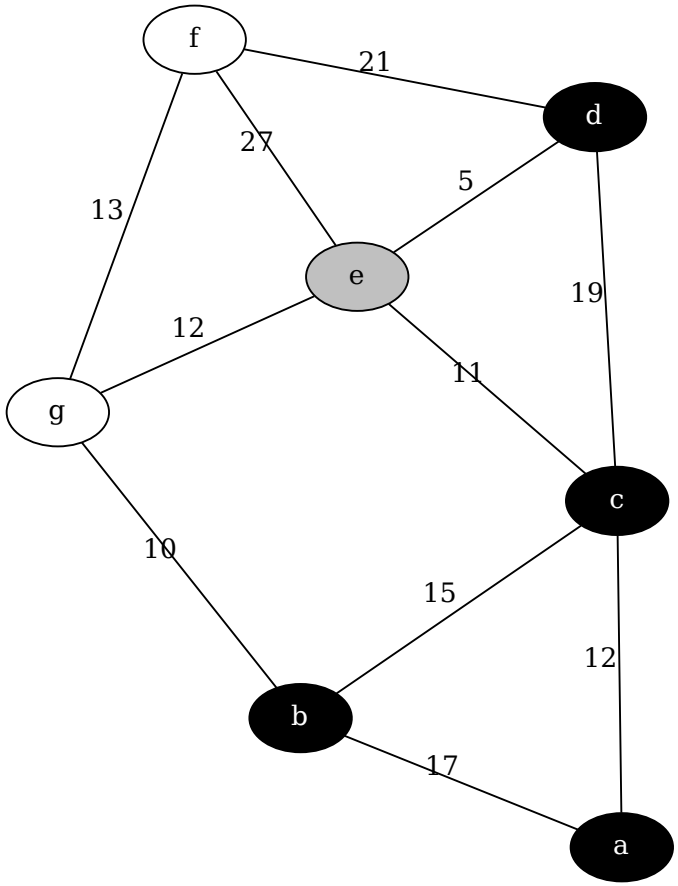
	a	b	c	d	e	f	g
a	0	17	12	28	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



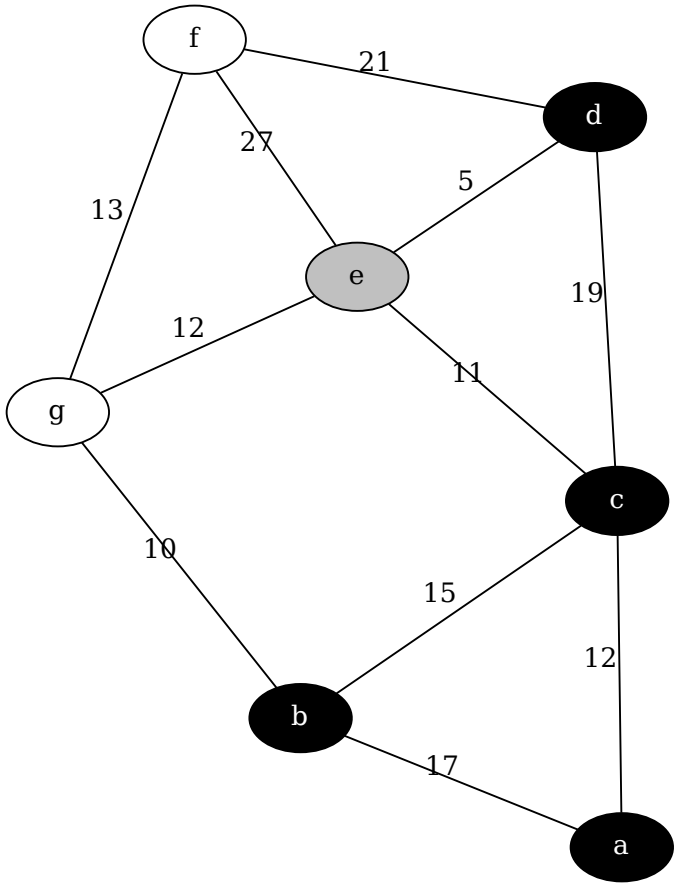
	a	b	c	d	e	f	g
a	0	17	12	28	23	52	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



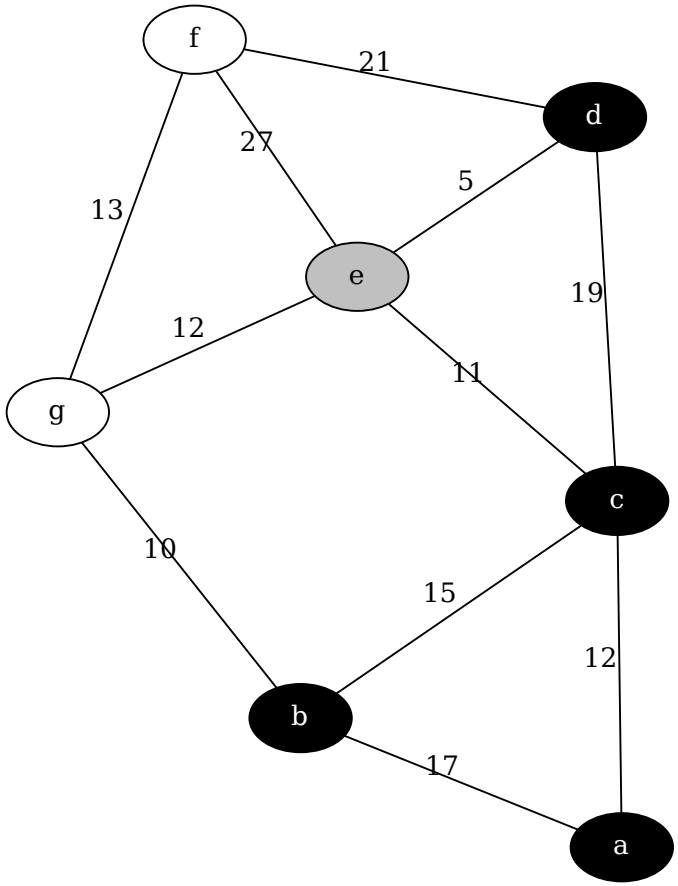
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



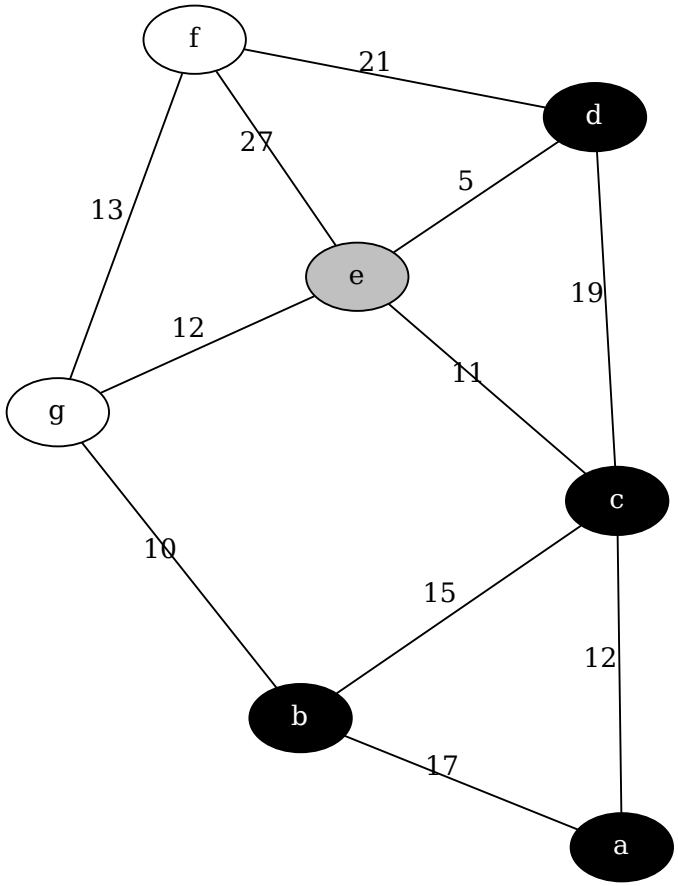
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



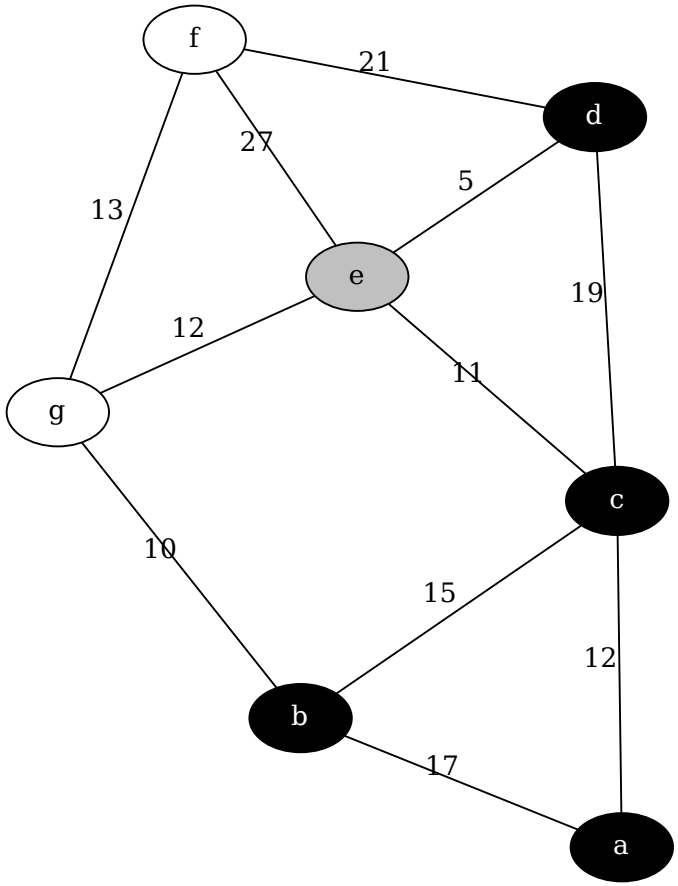
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



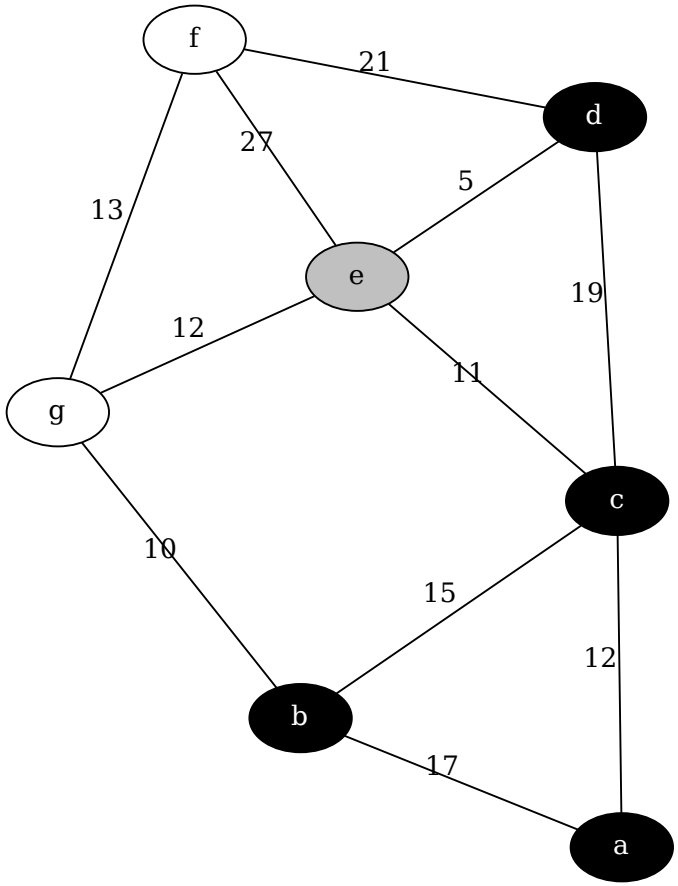
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



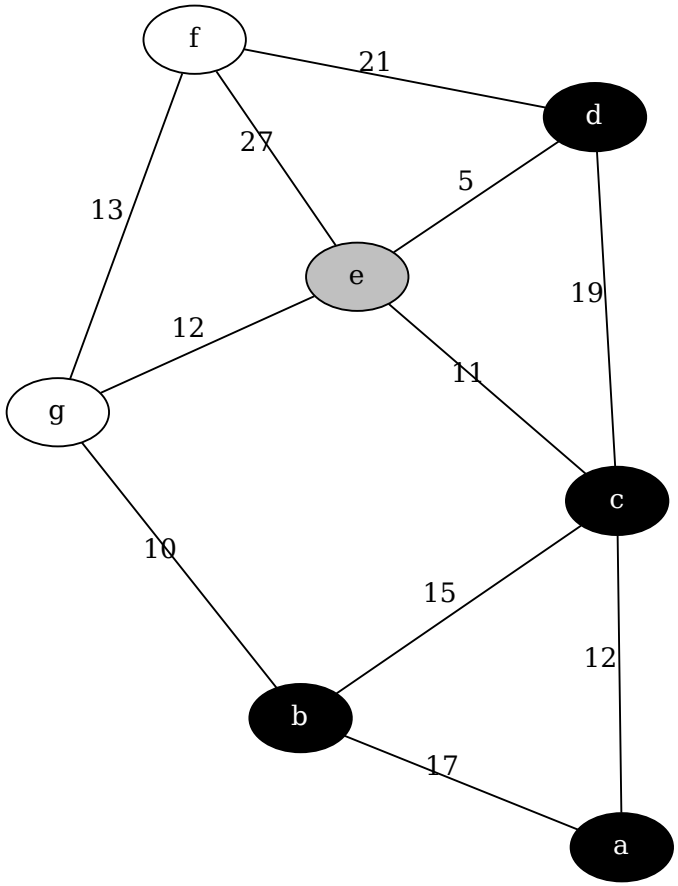
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	34	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



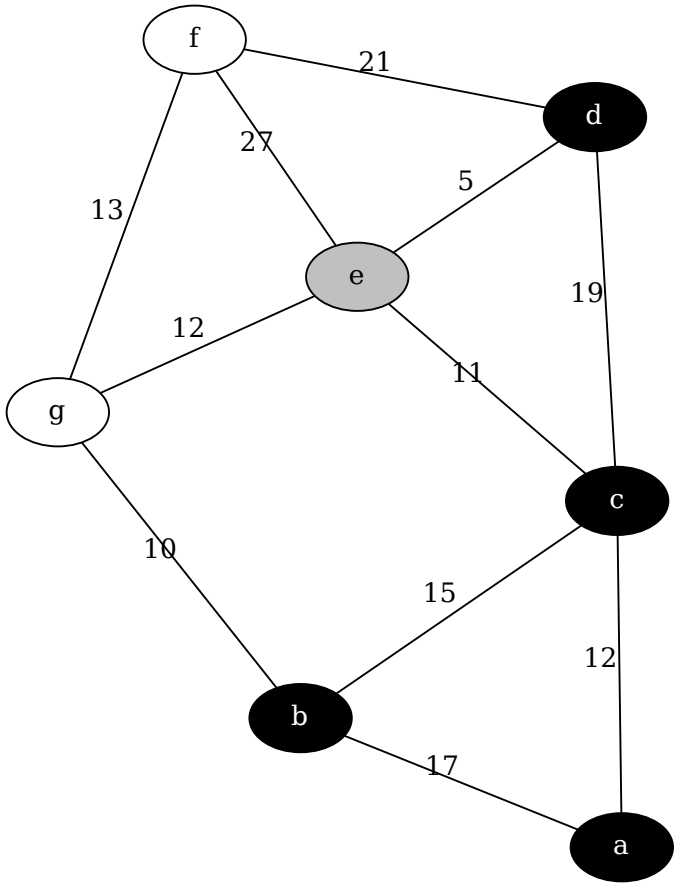
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



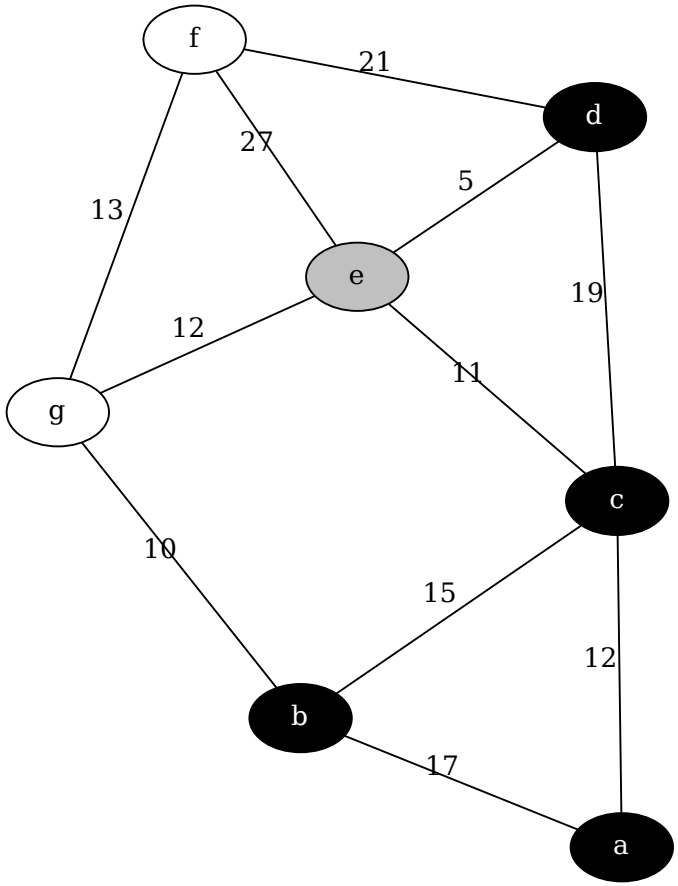
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



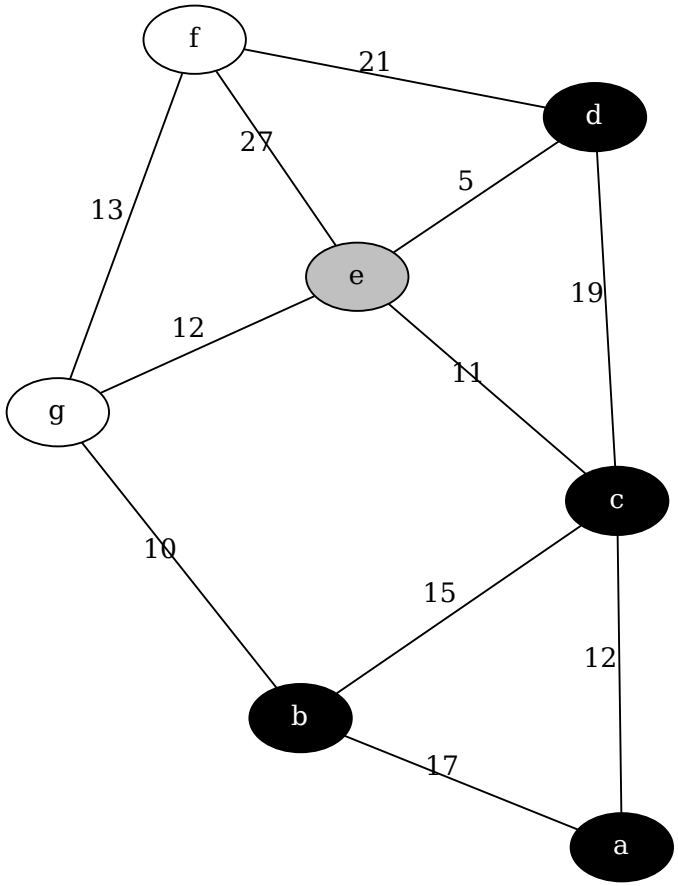
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	55	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



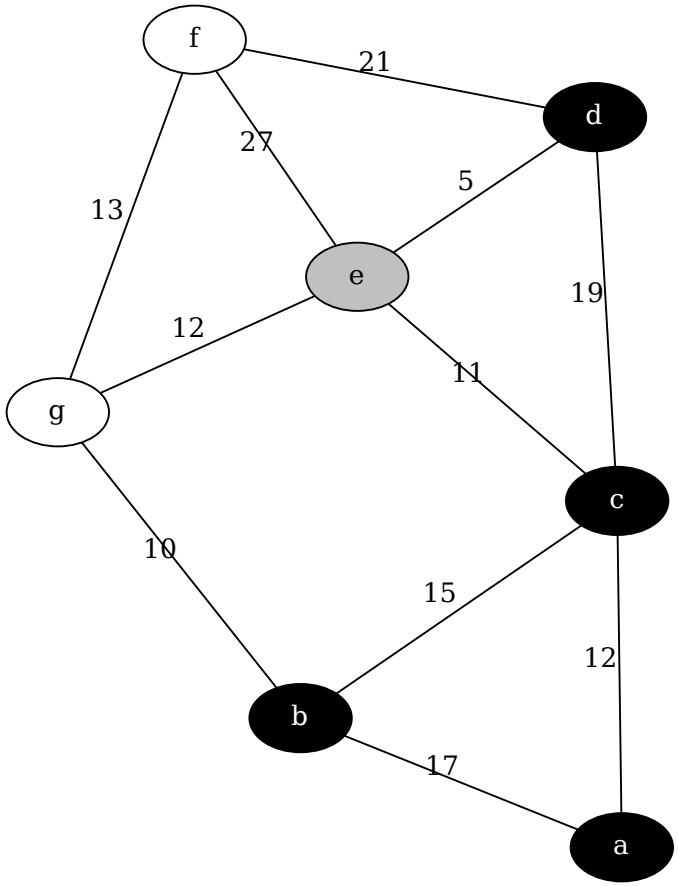
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



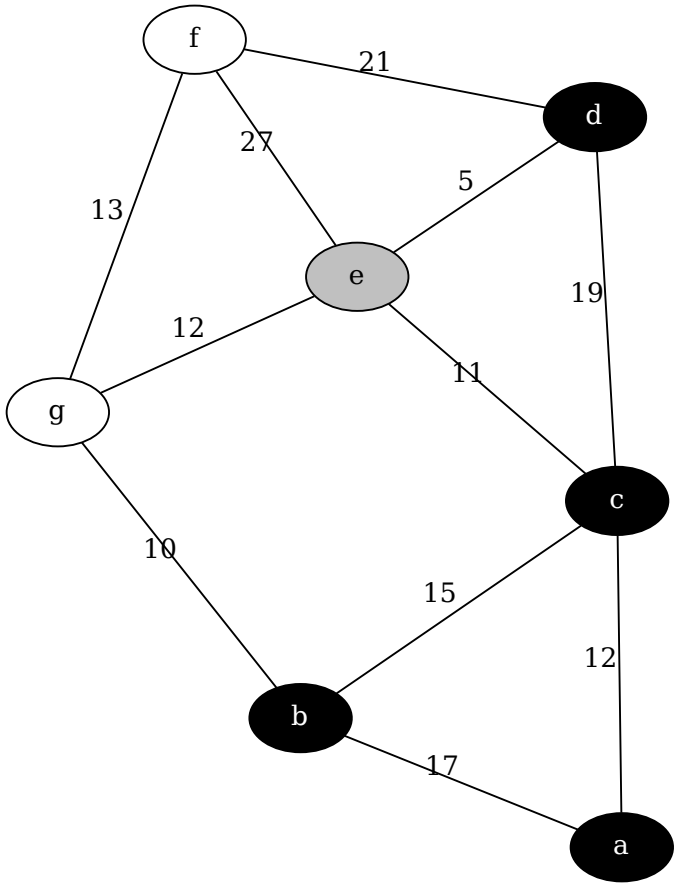
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



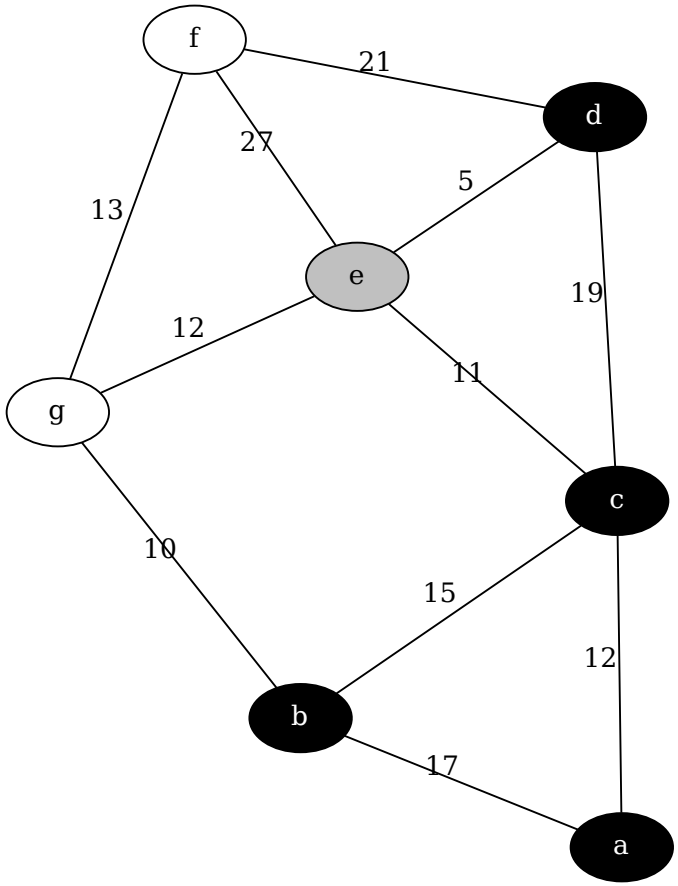
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



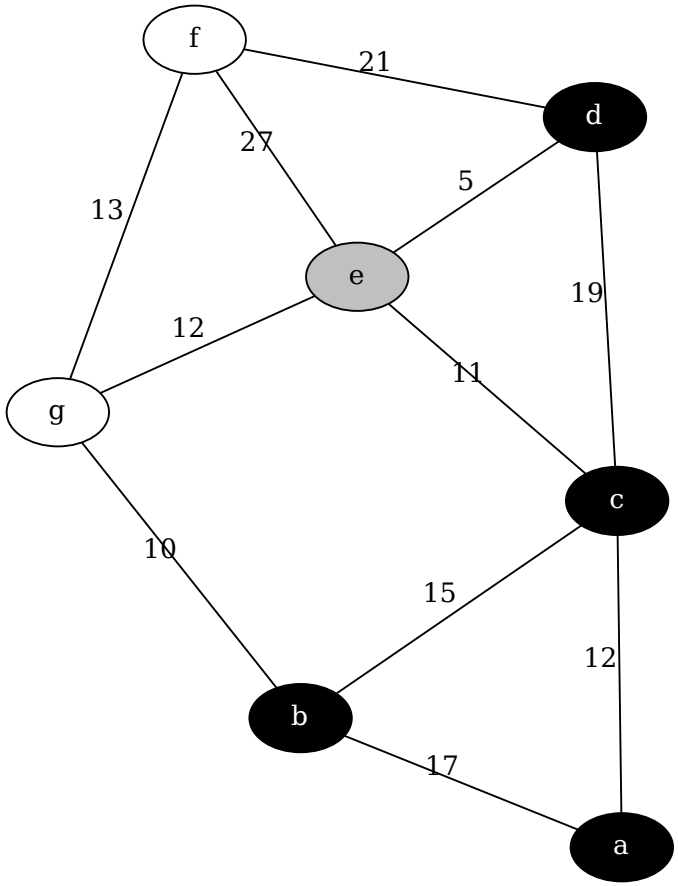
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	19	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



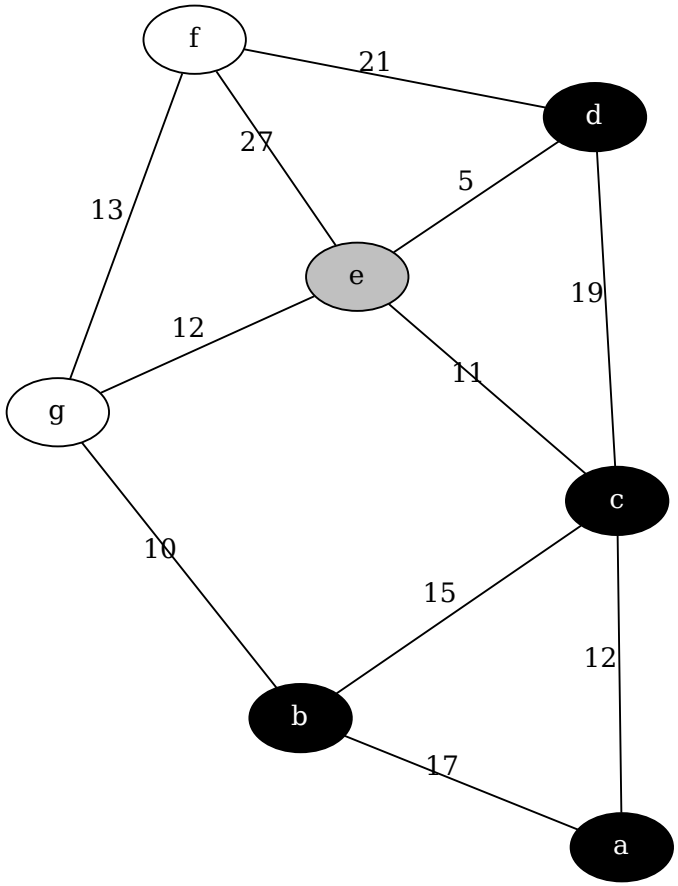
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



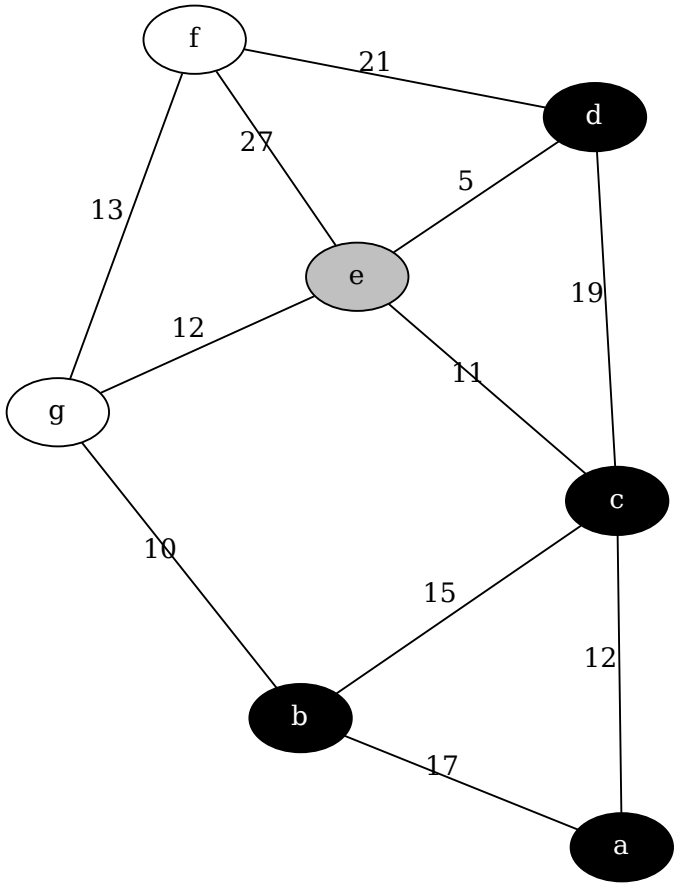
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



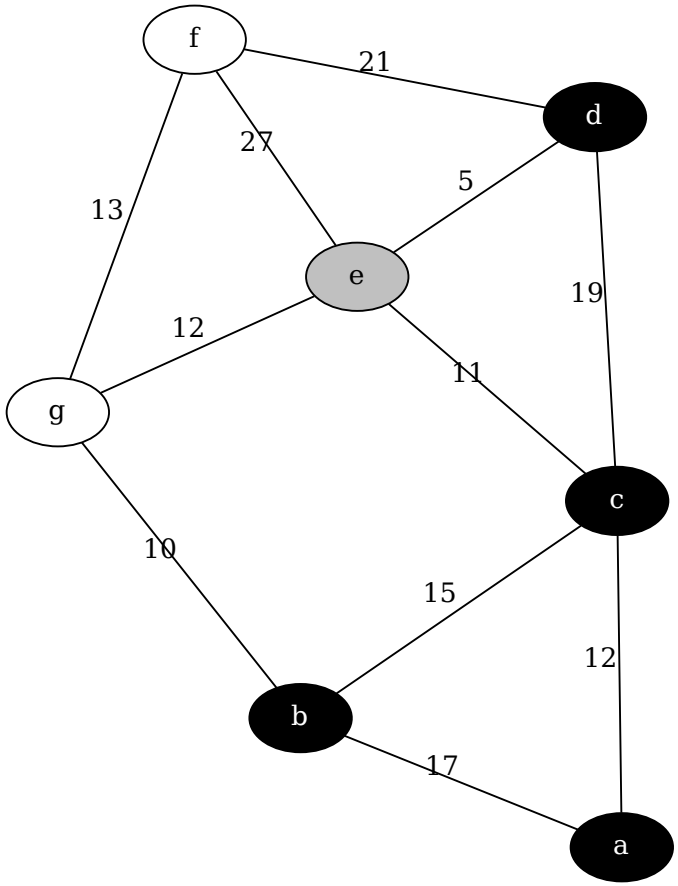
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	40	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



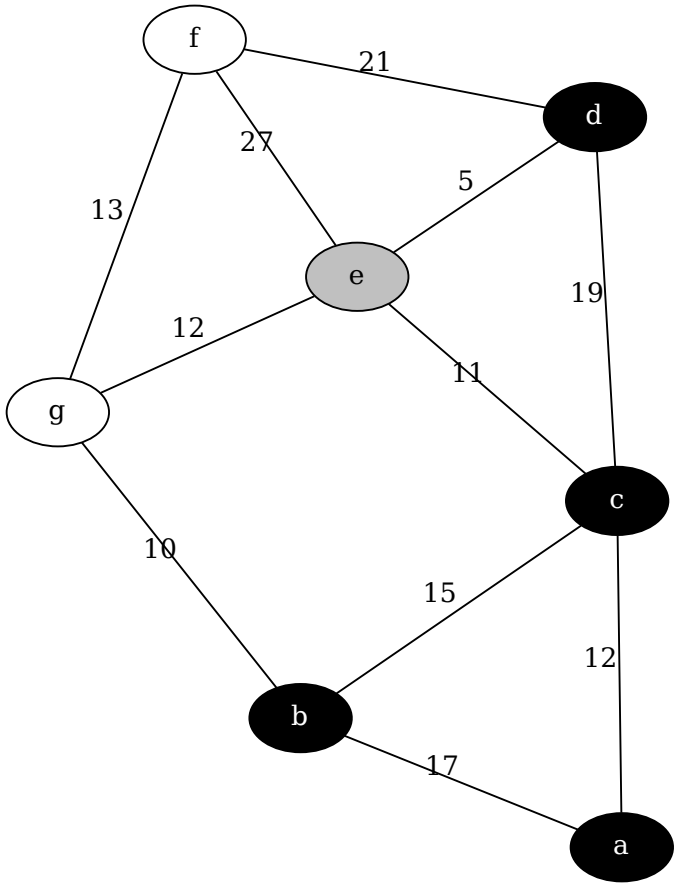
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



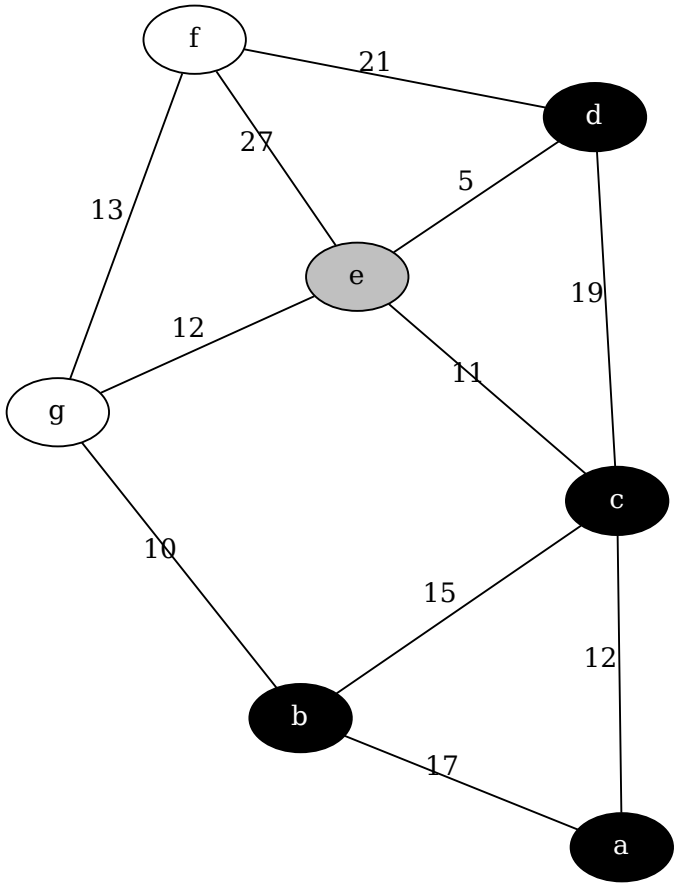
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



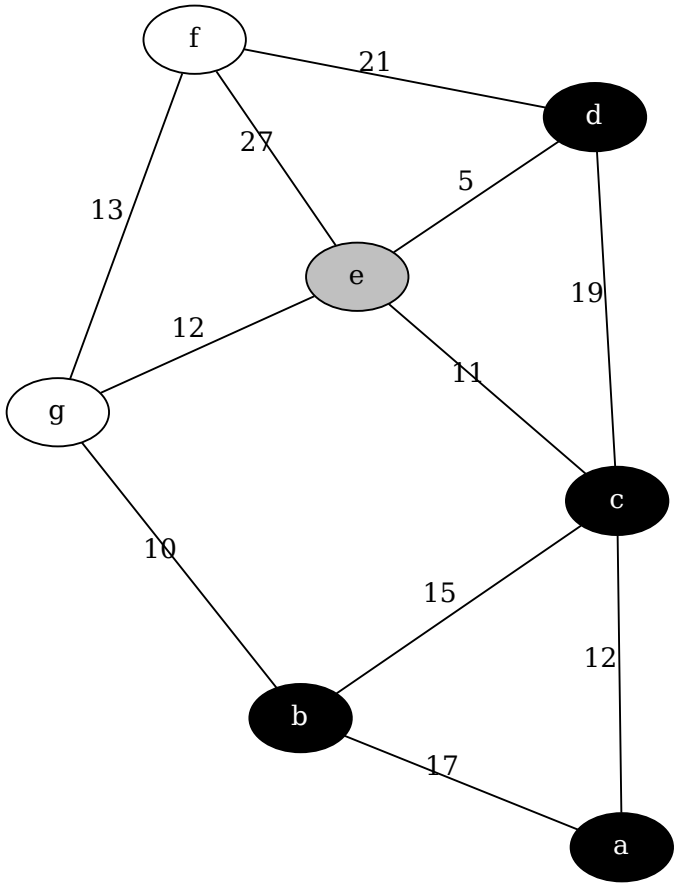
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	25
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



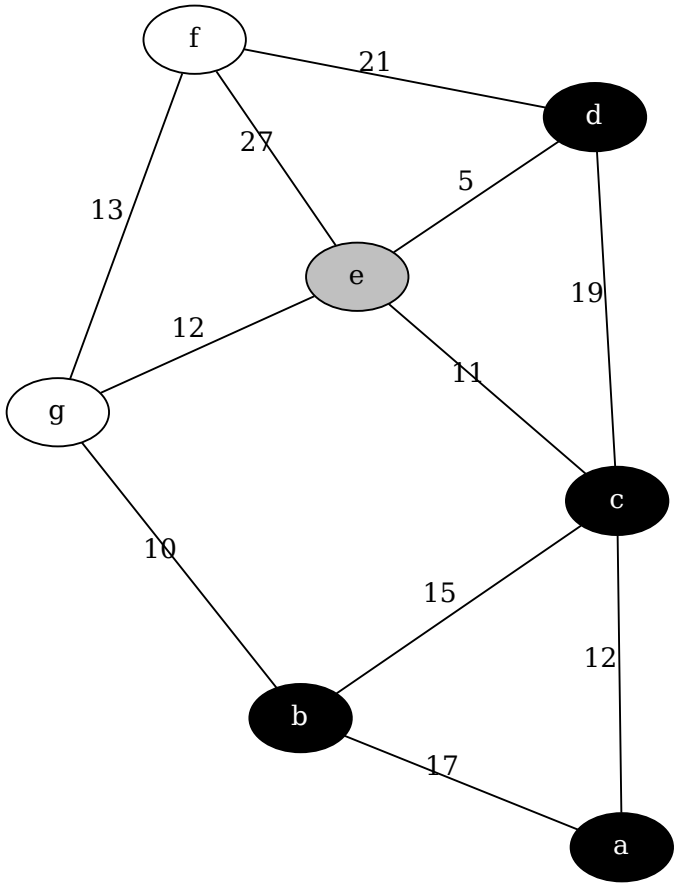
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



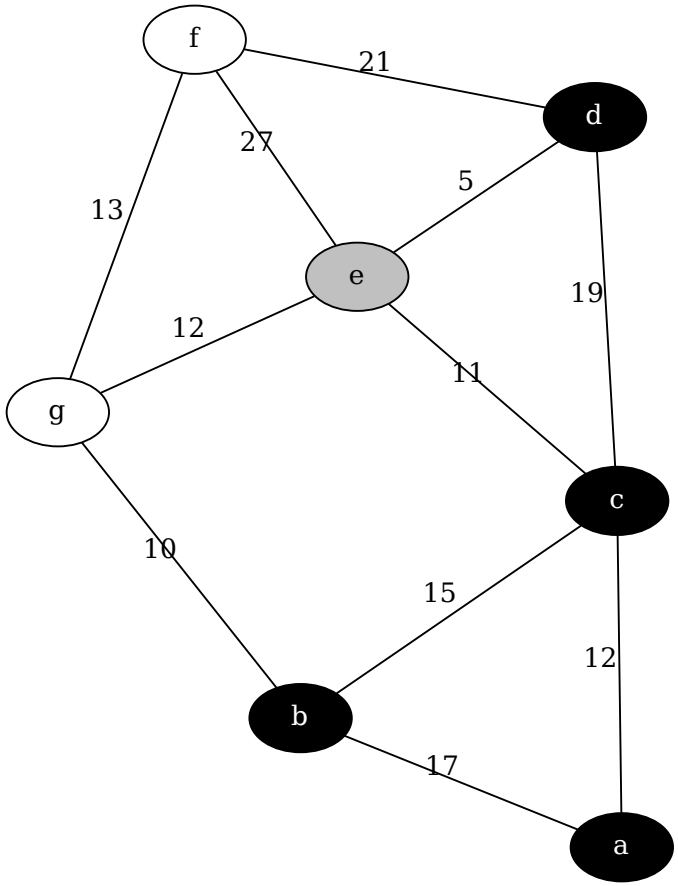
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



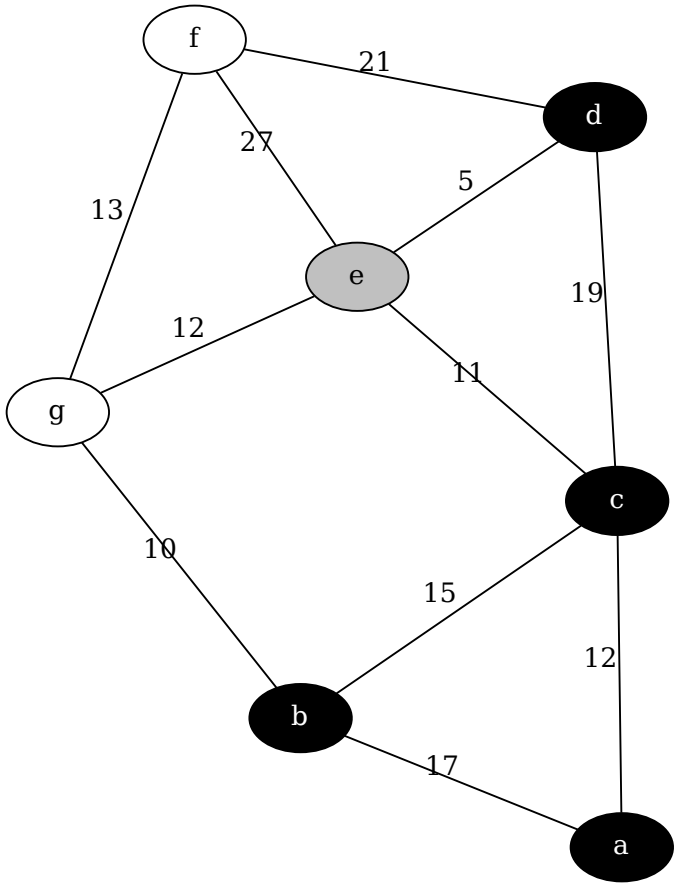
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



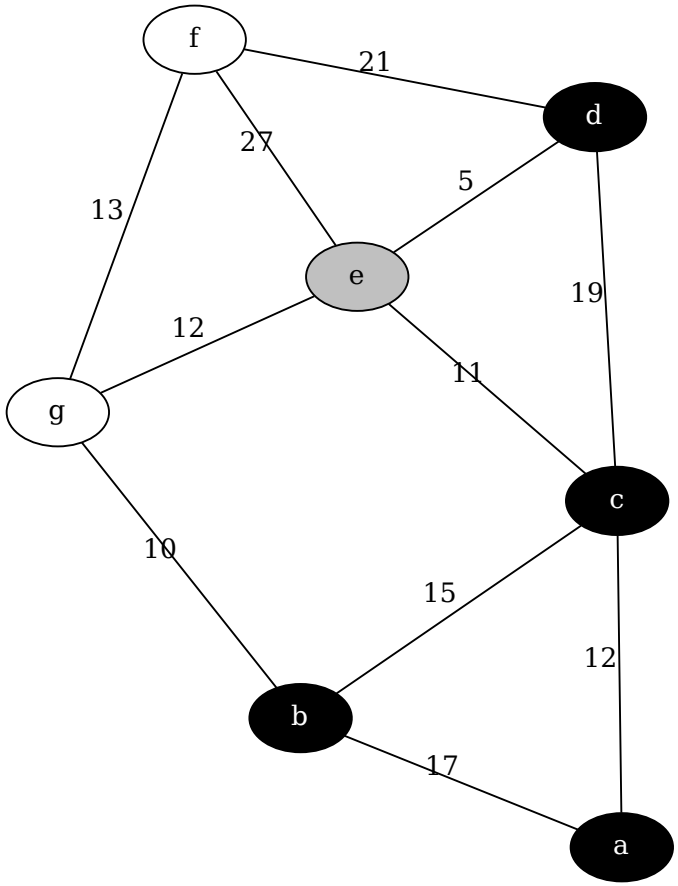
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	44
e					0	26	12
f						0	13
g							0

Example



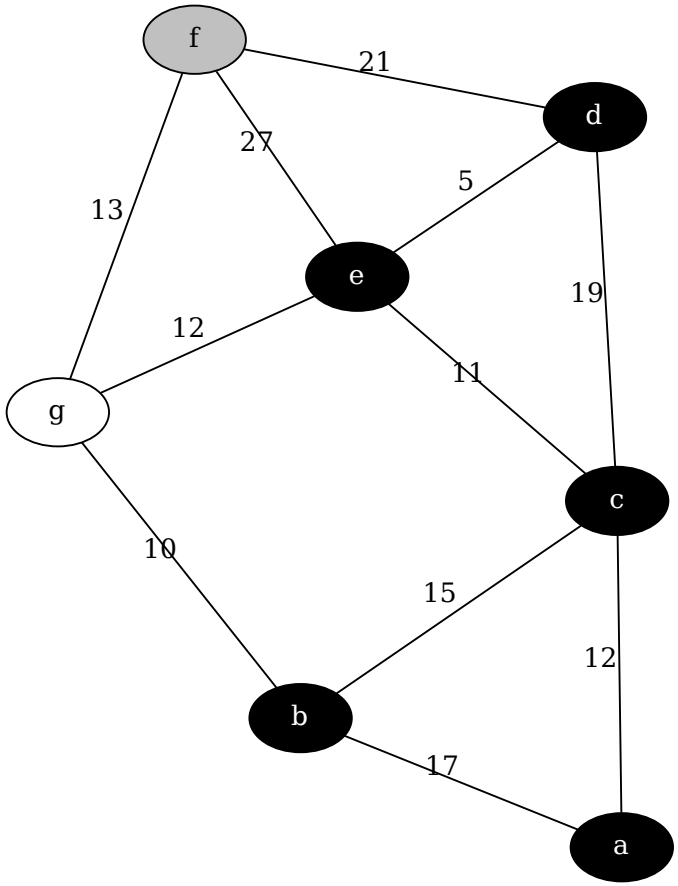
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



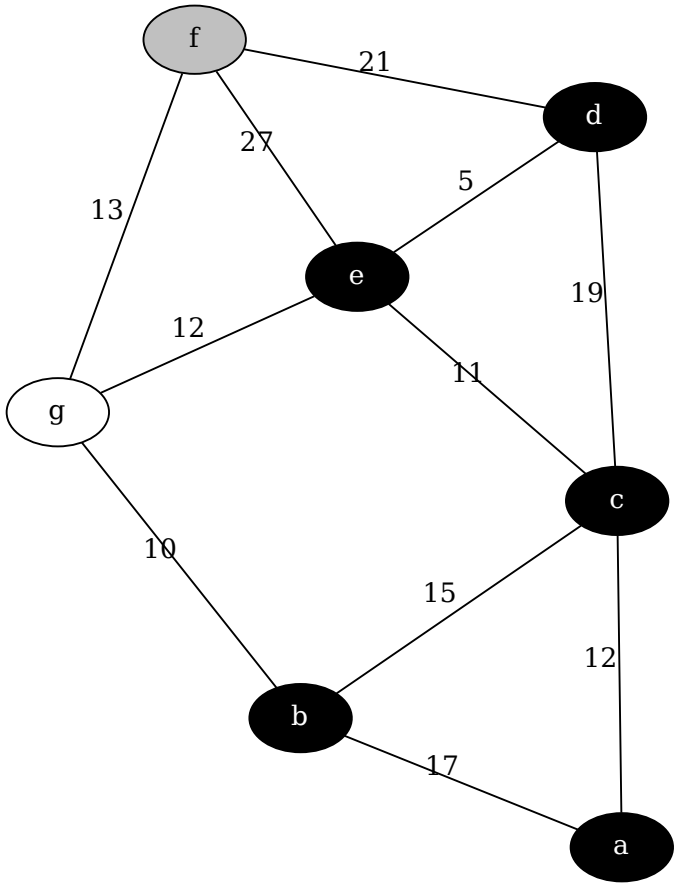
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



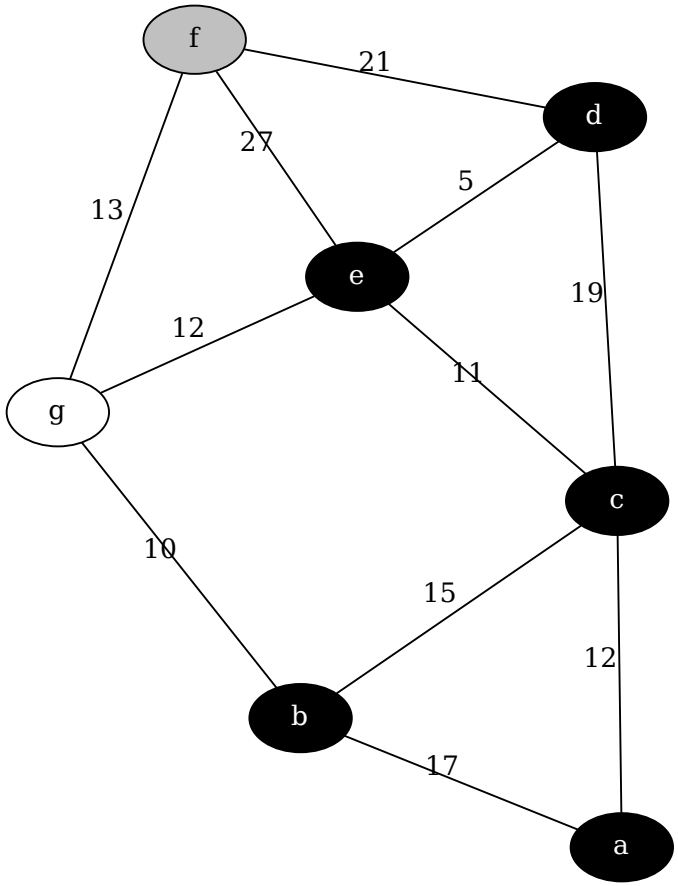
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



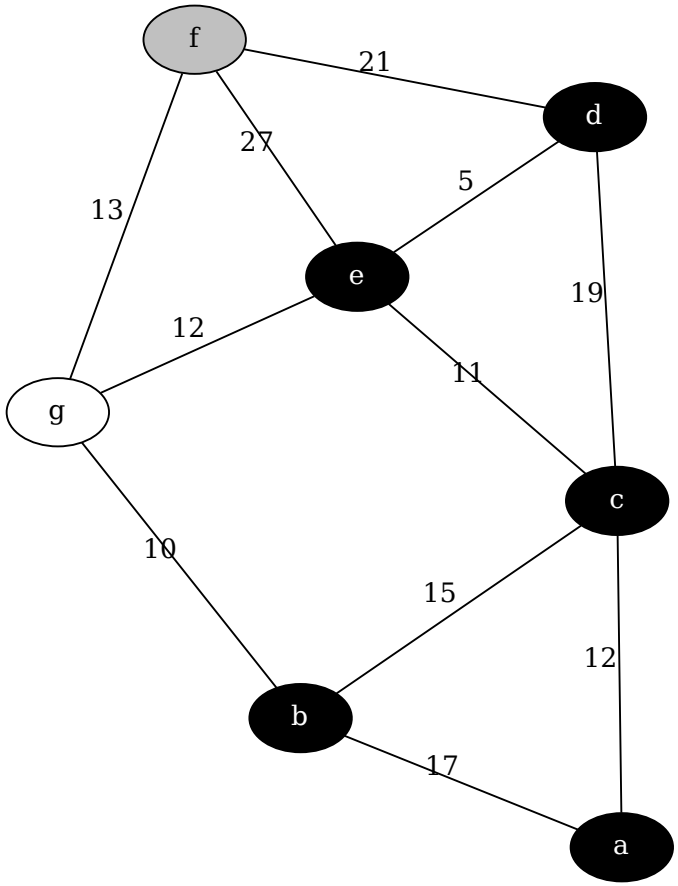
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



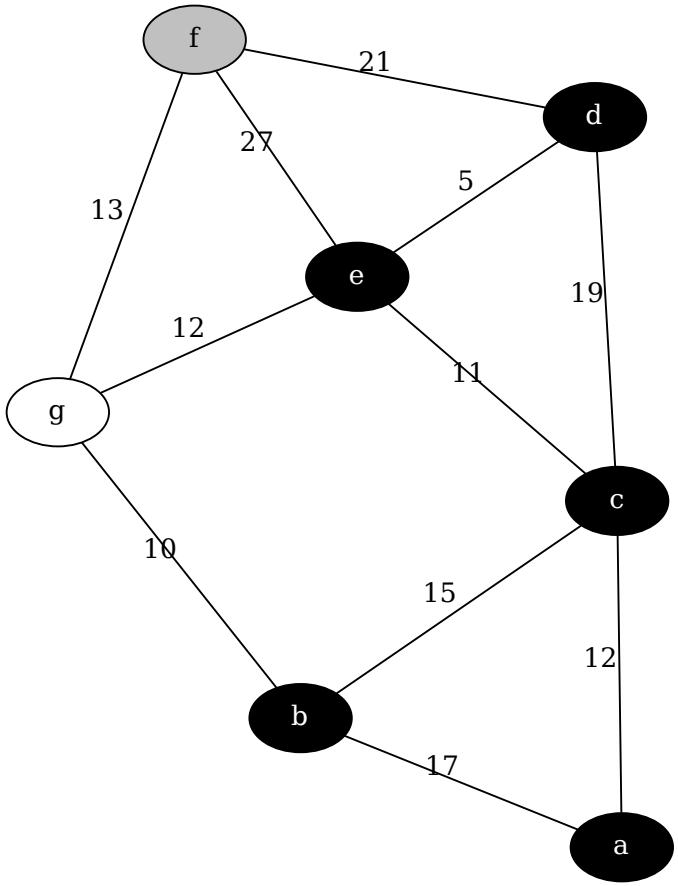
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



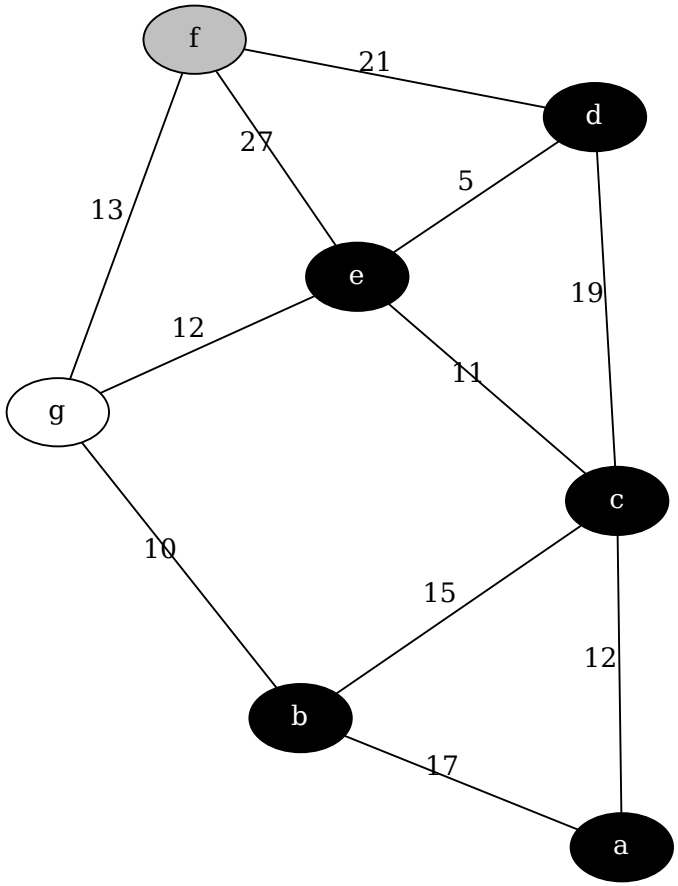
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



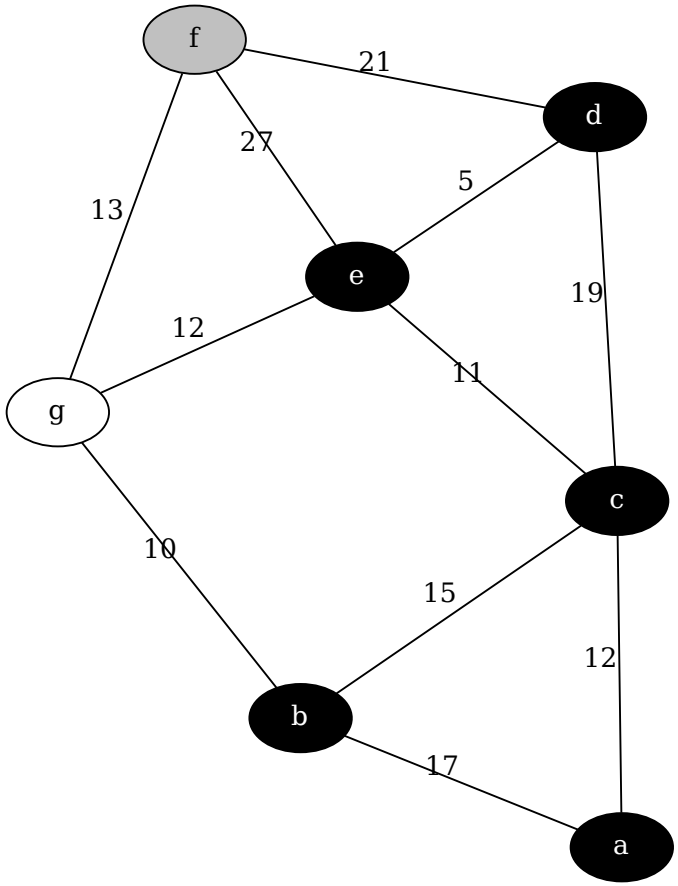
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



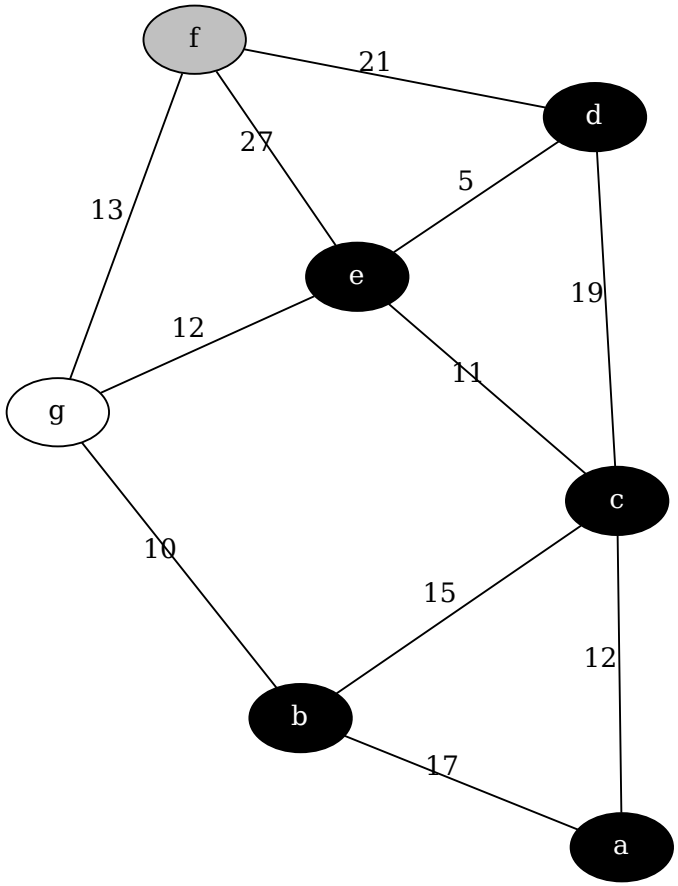
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



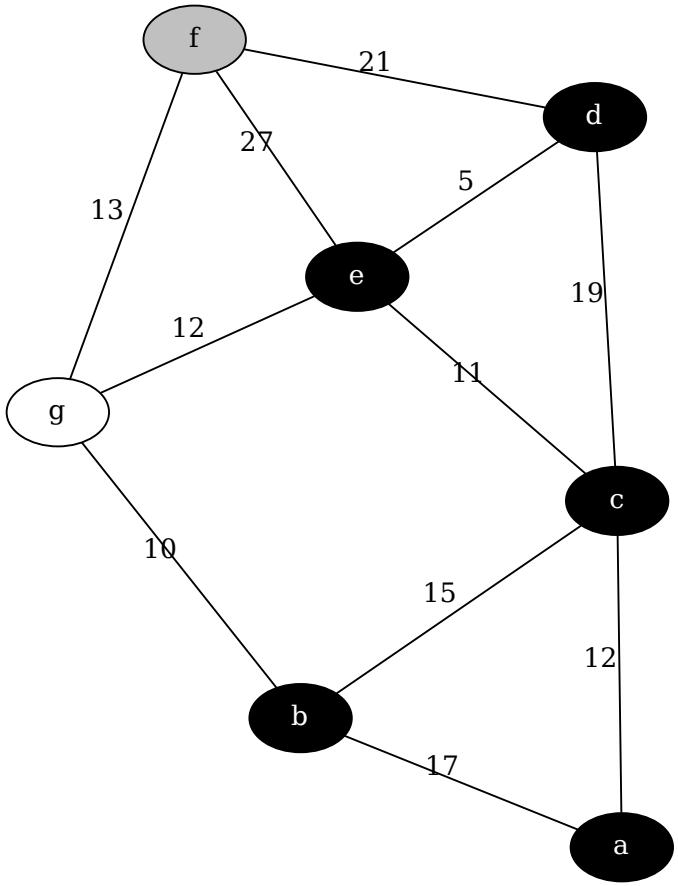
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



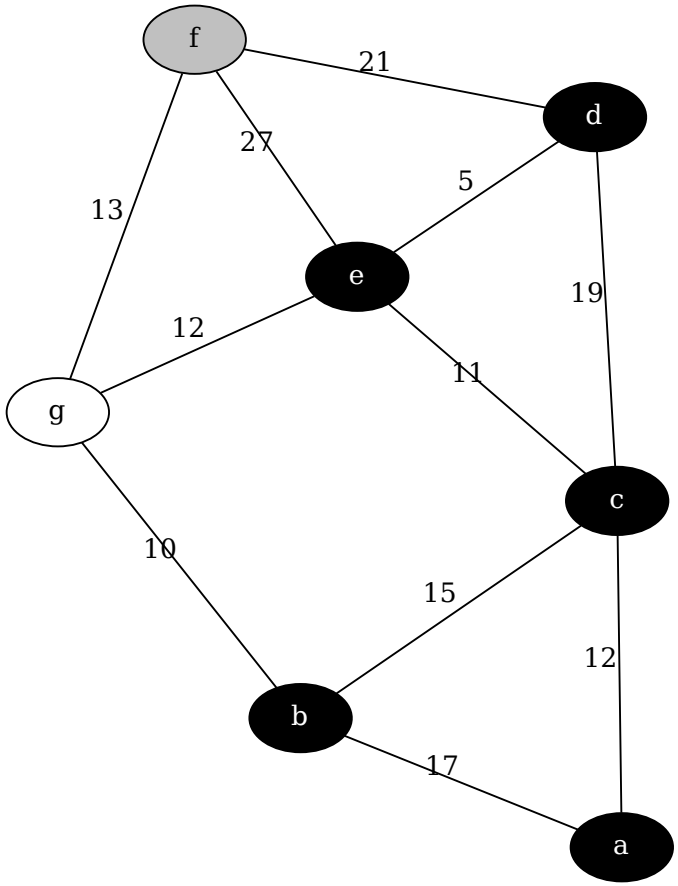
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



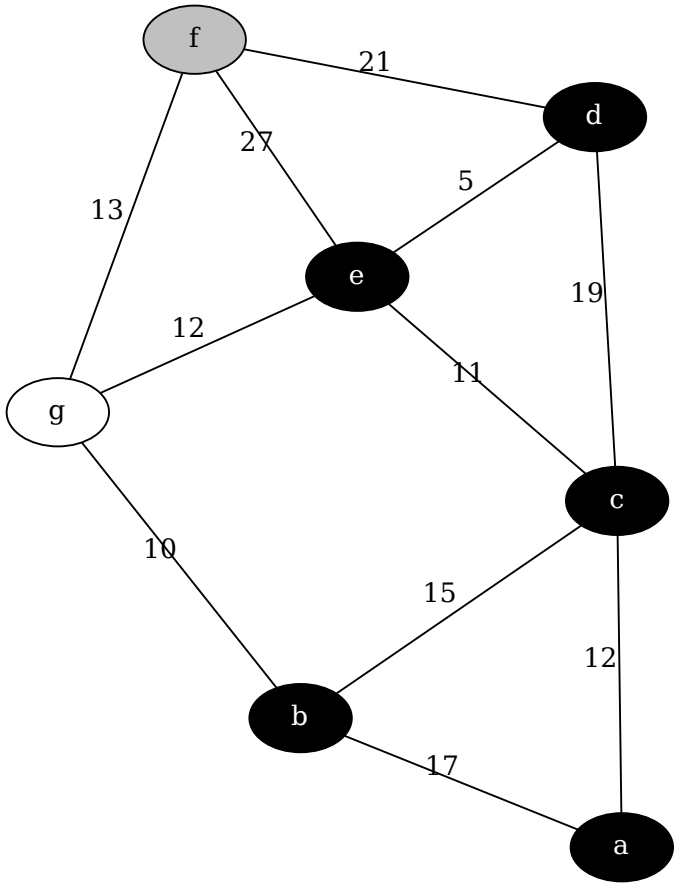
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



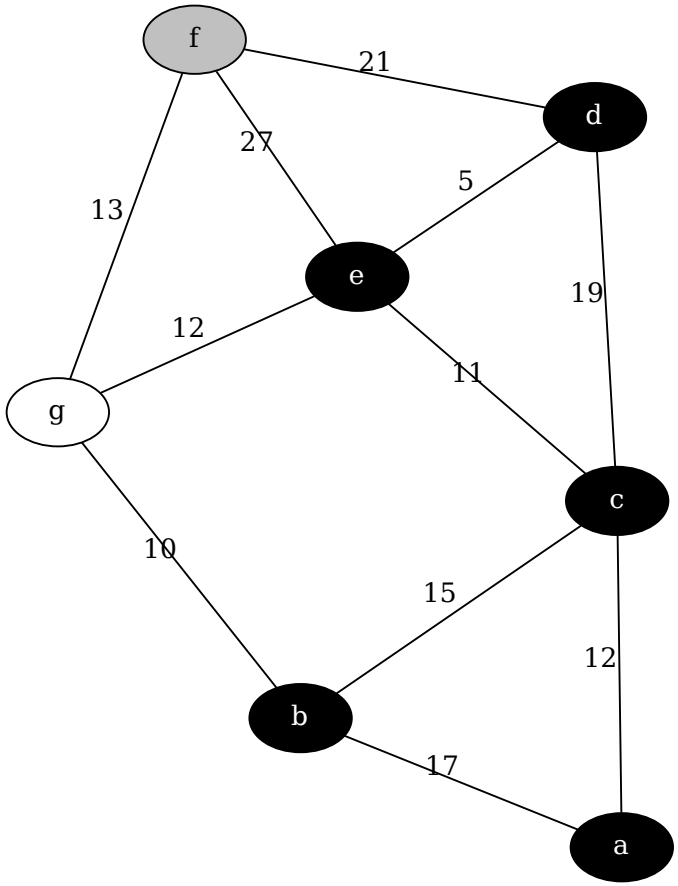
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



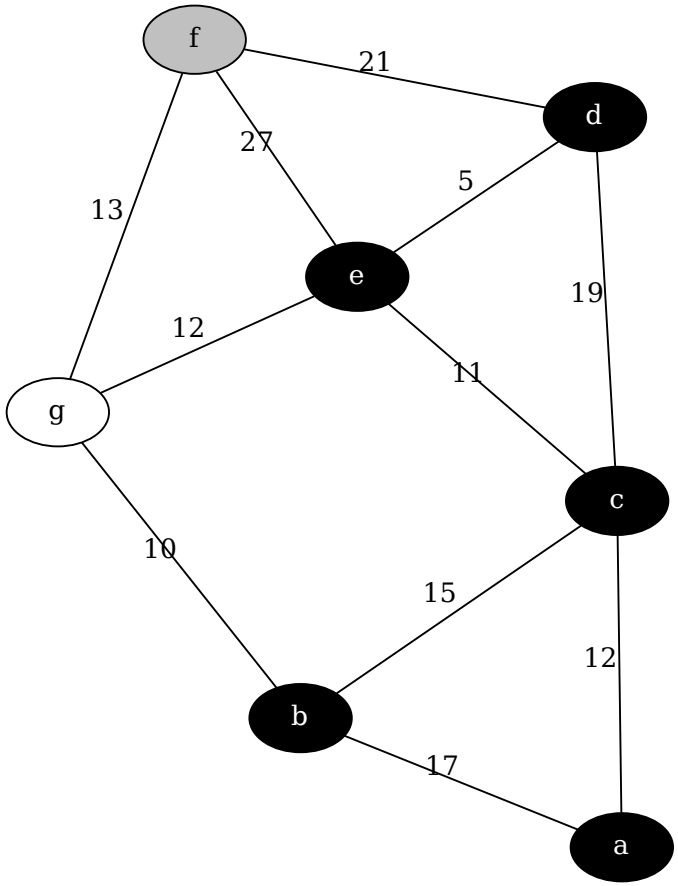
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



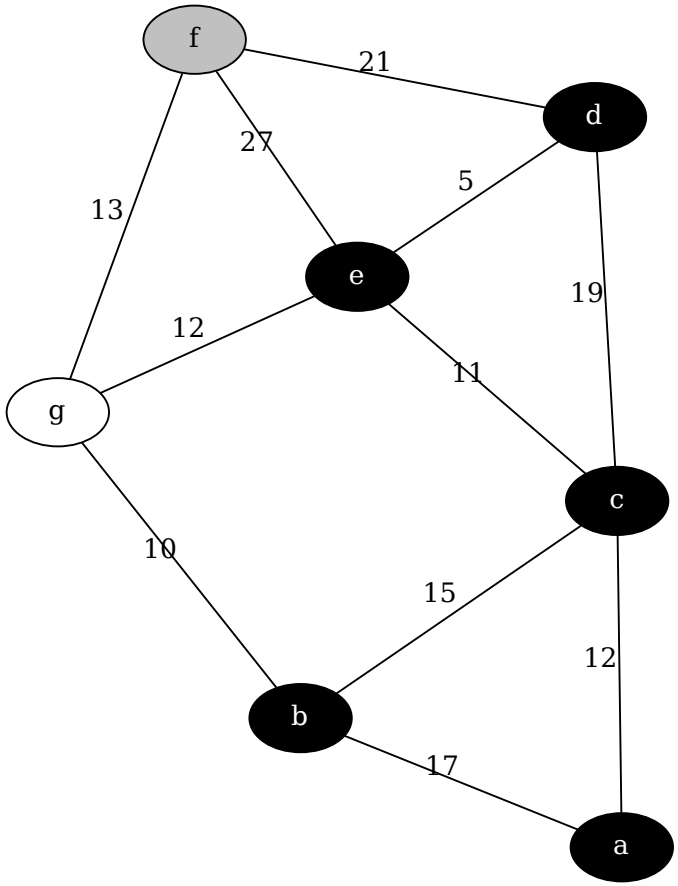
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



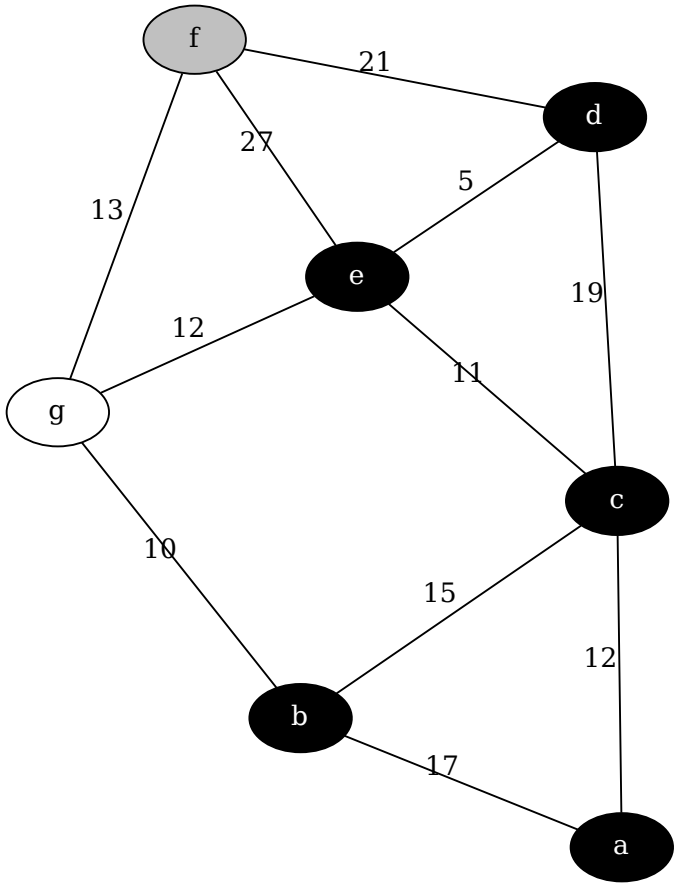
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



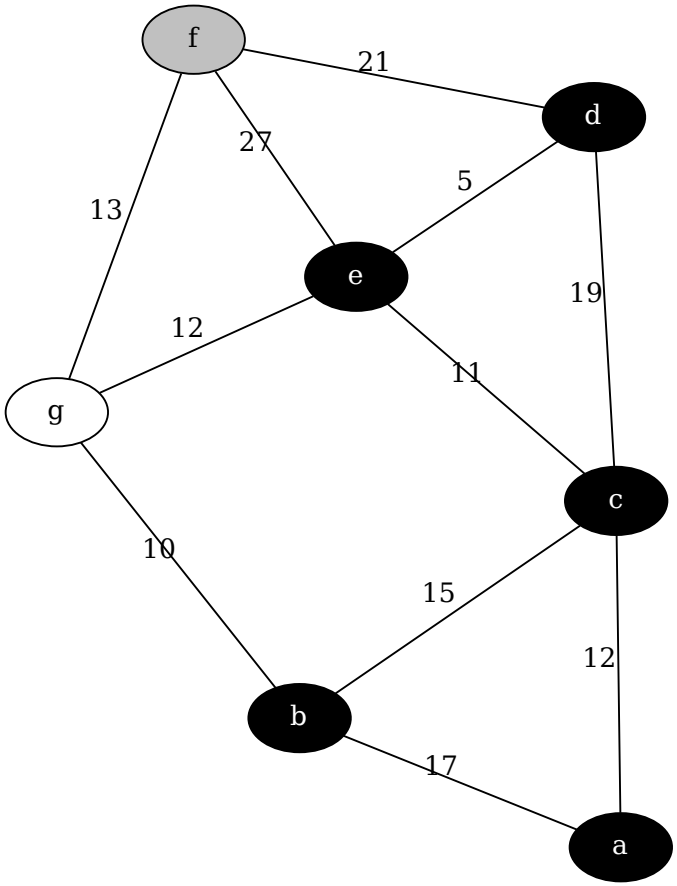
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



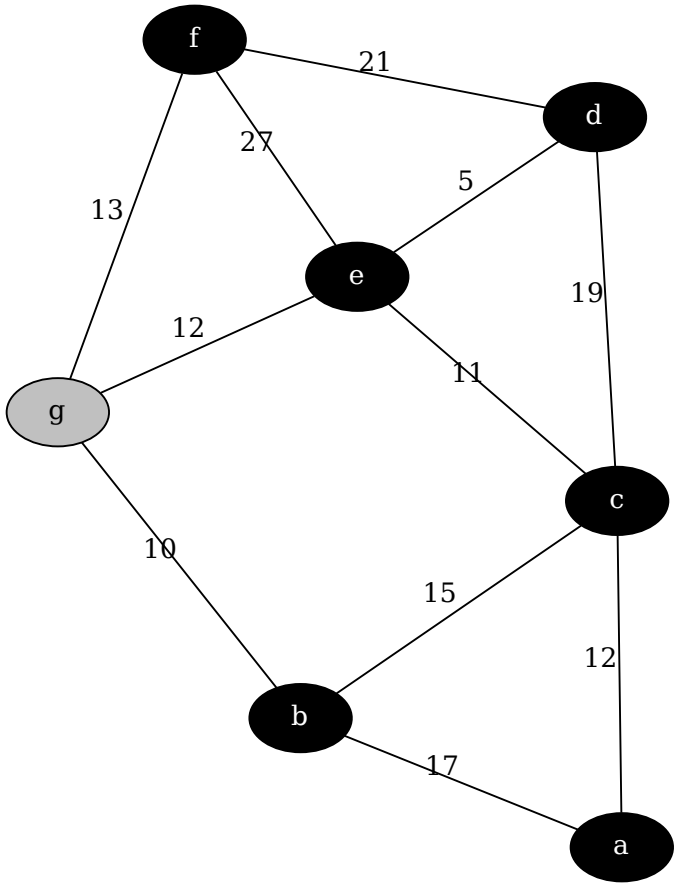
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



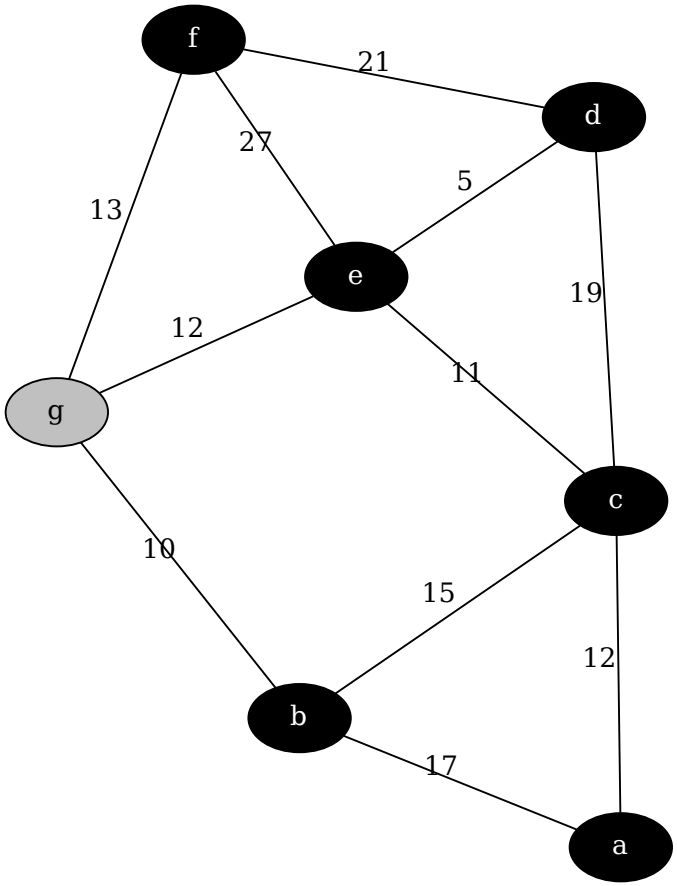
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



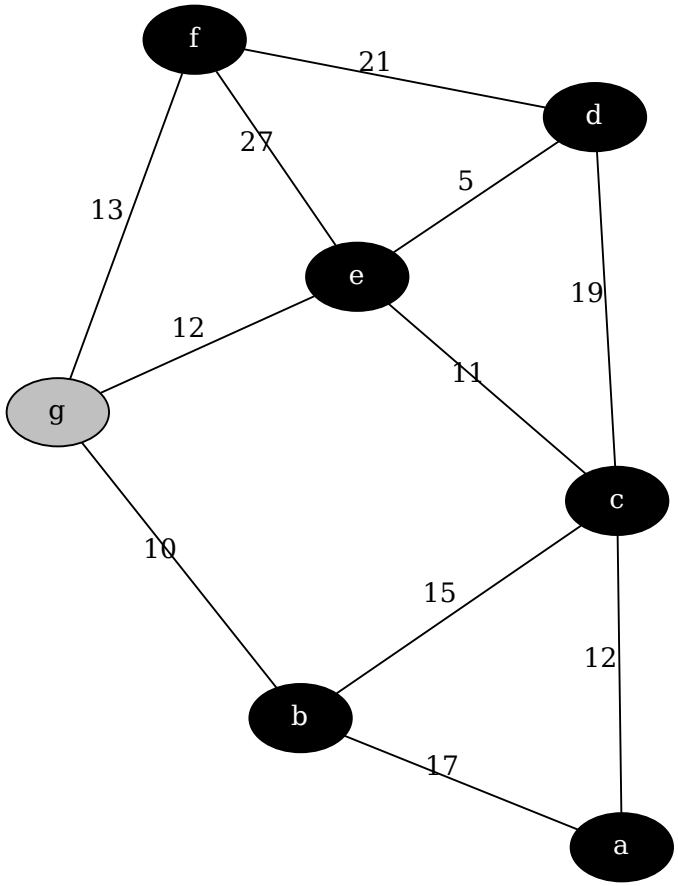
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



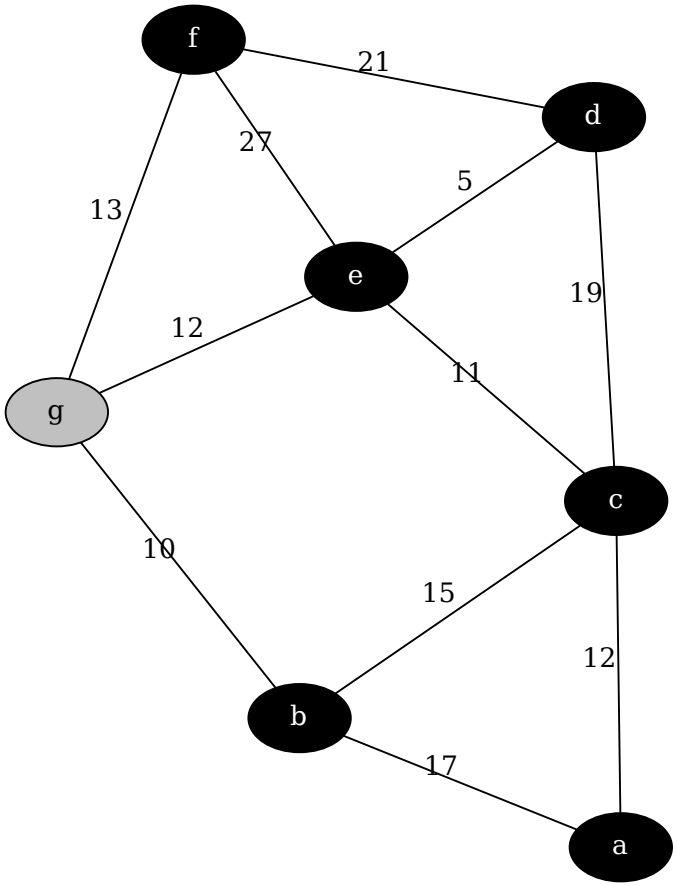
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



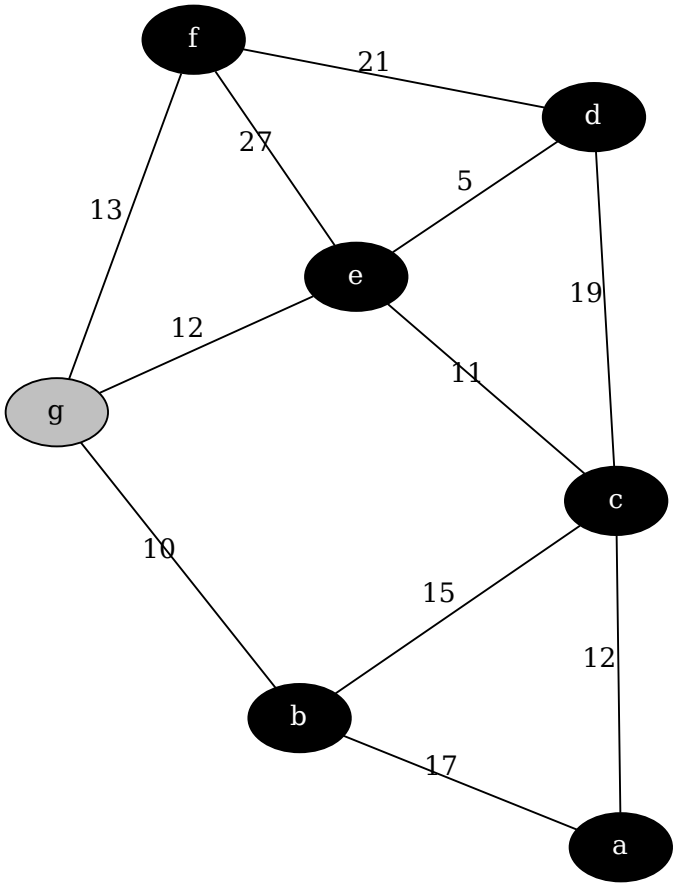
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



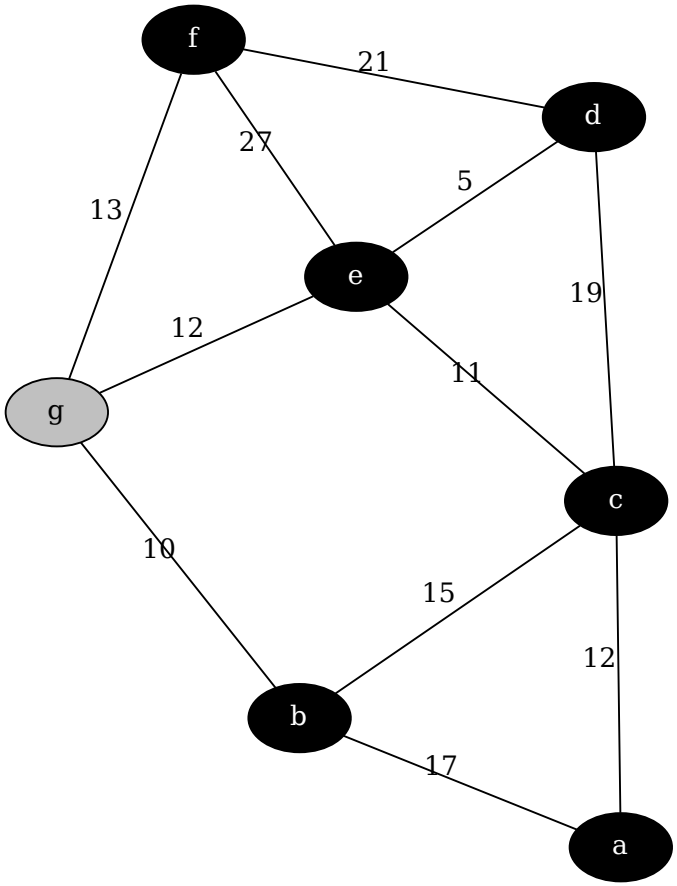
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



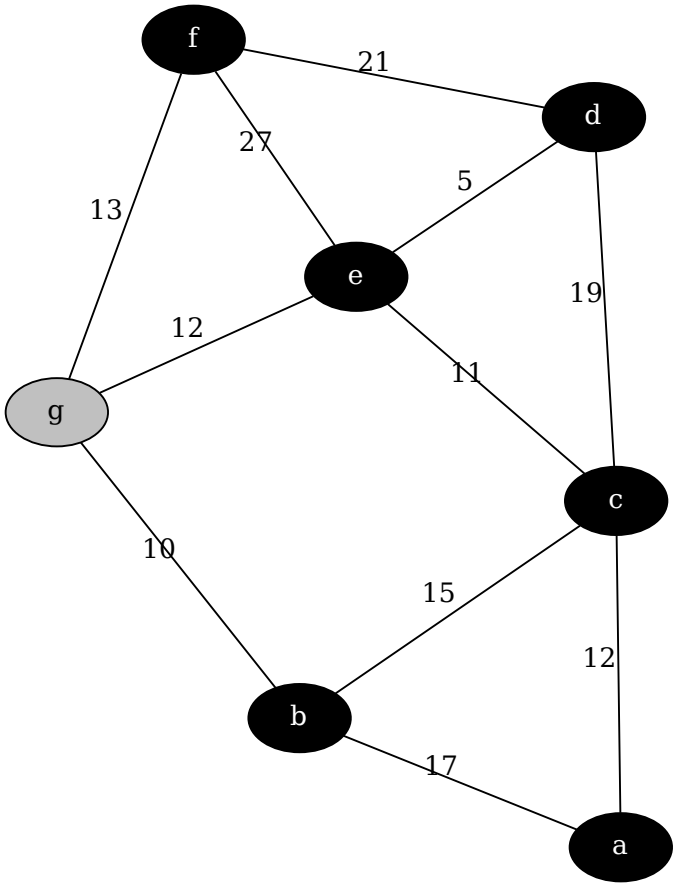
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



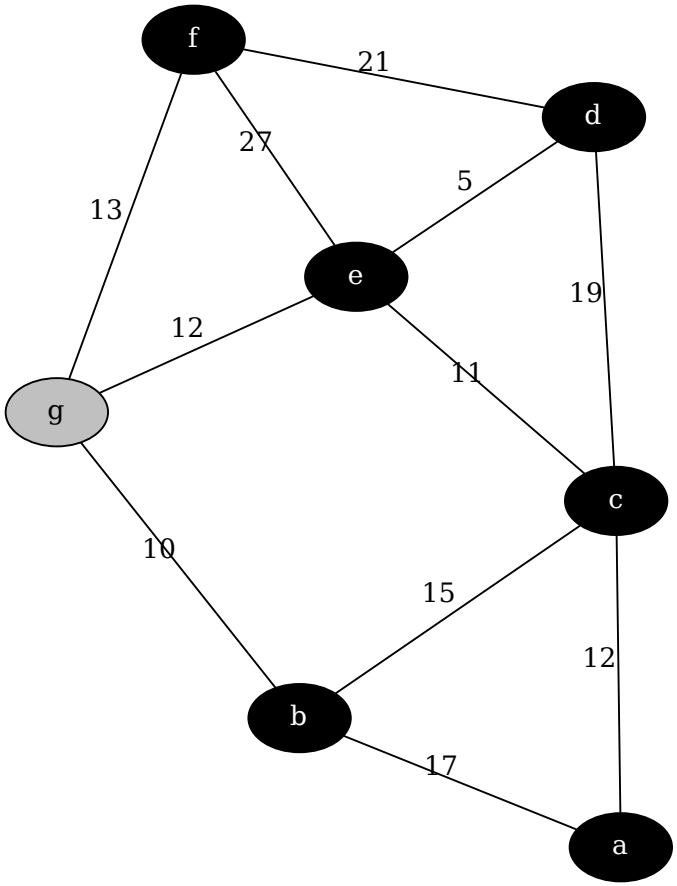
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



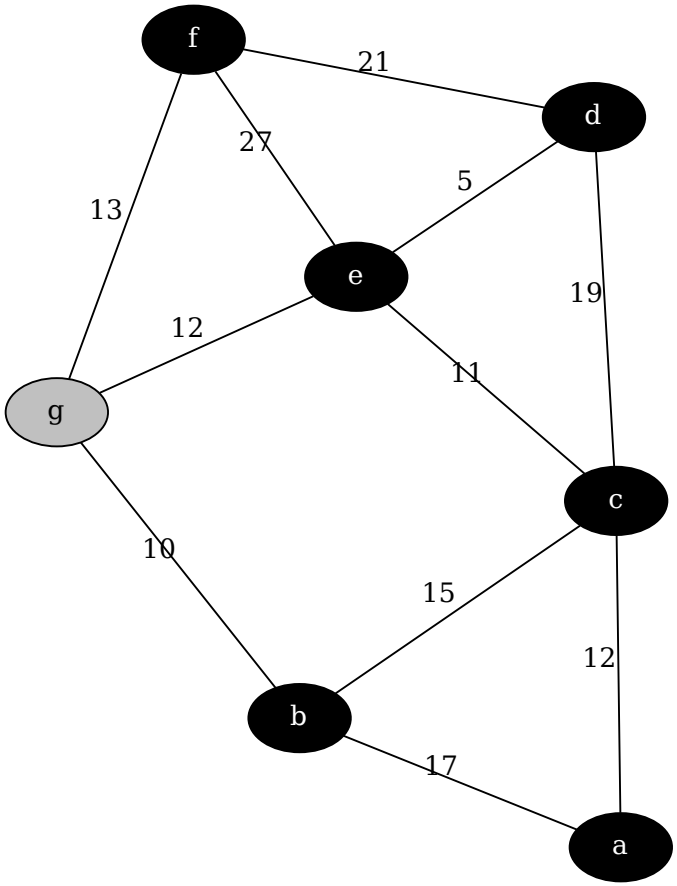
	a	b	c	d	e	f	g
a	0	17	12	28	23	49	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



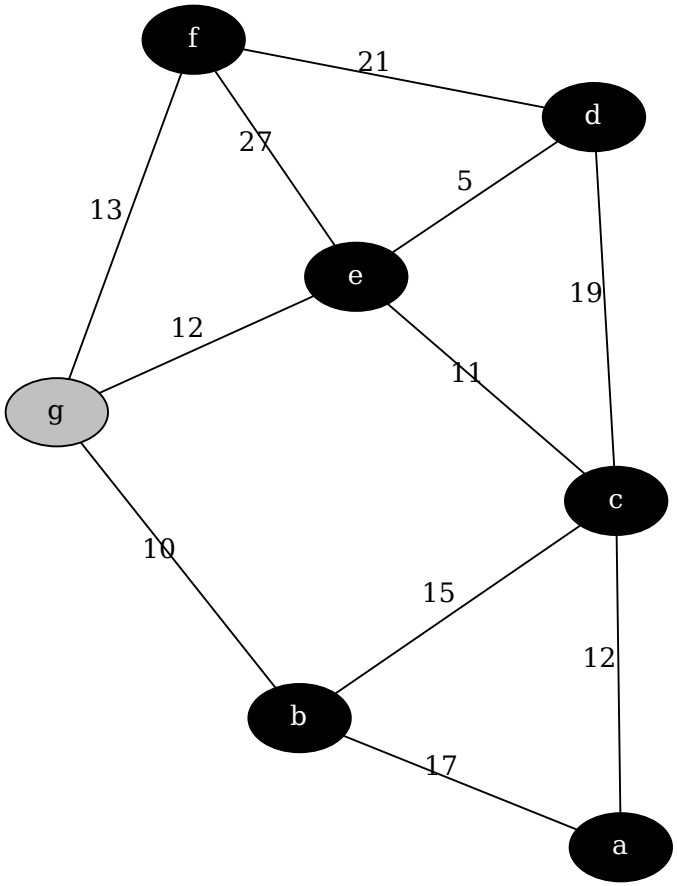
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



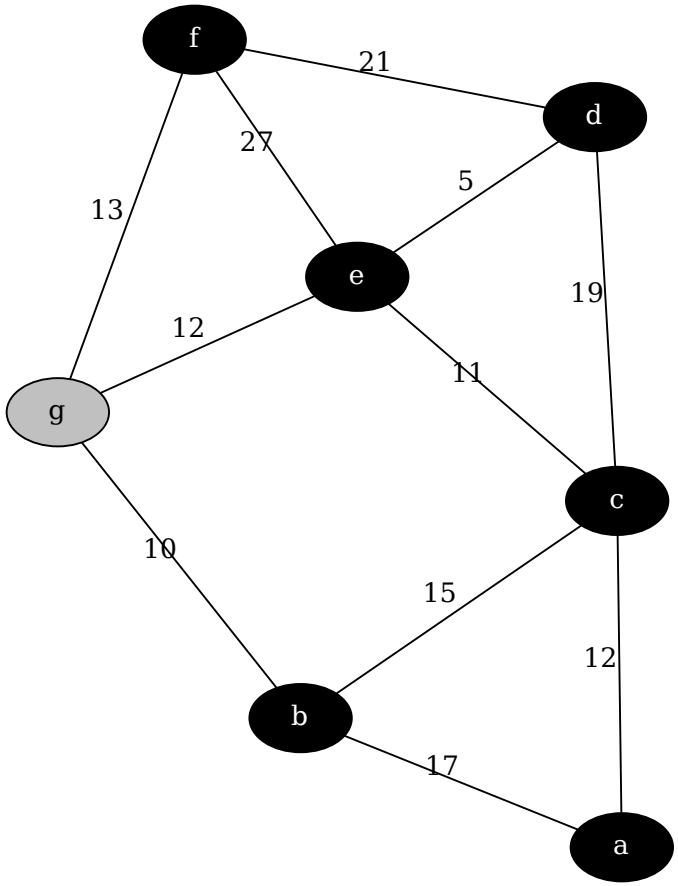
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



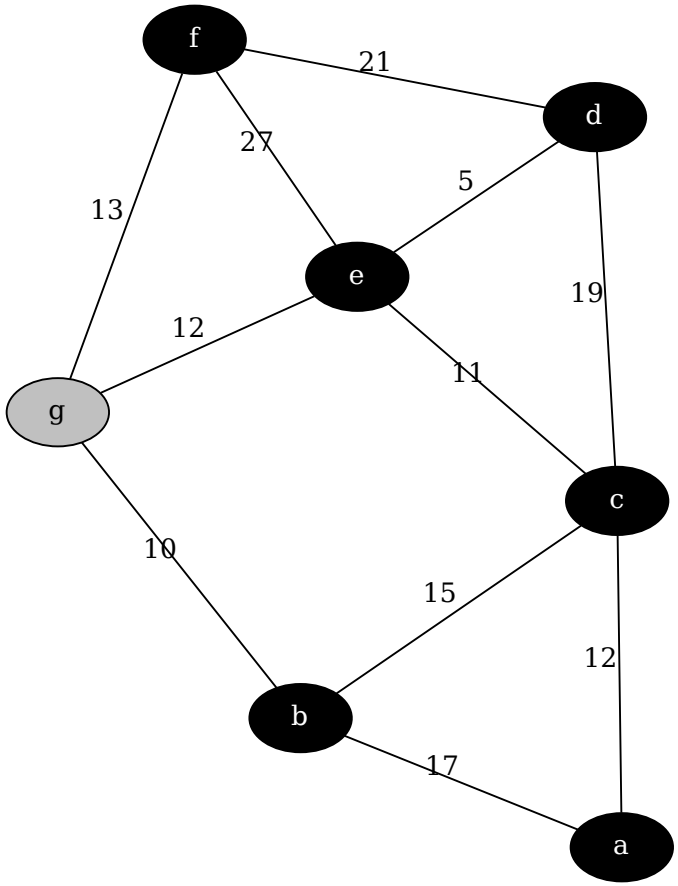
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



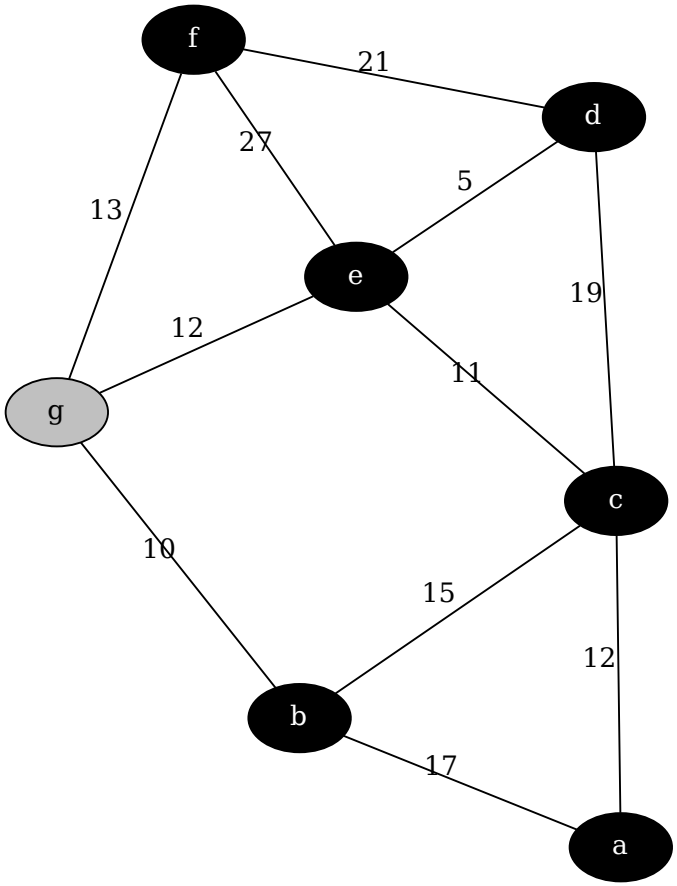
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	31	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



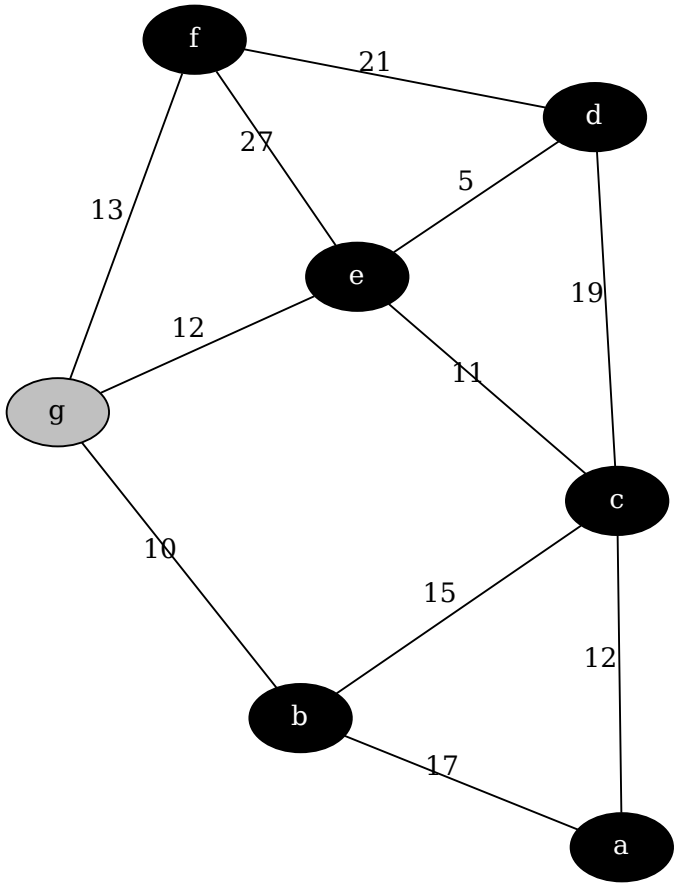
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



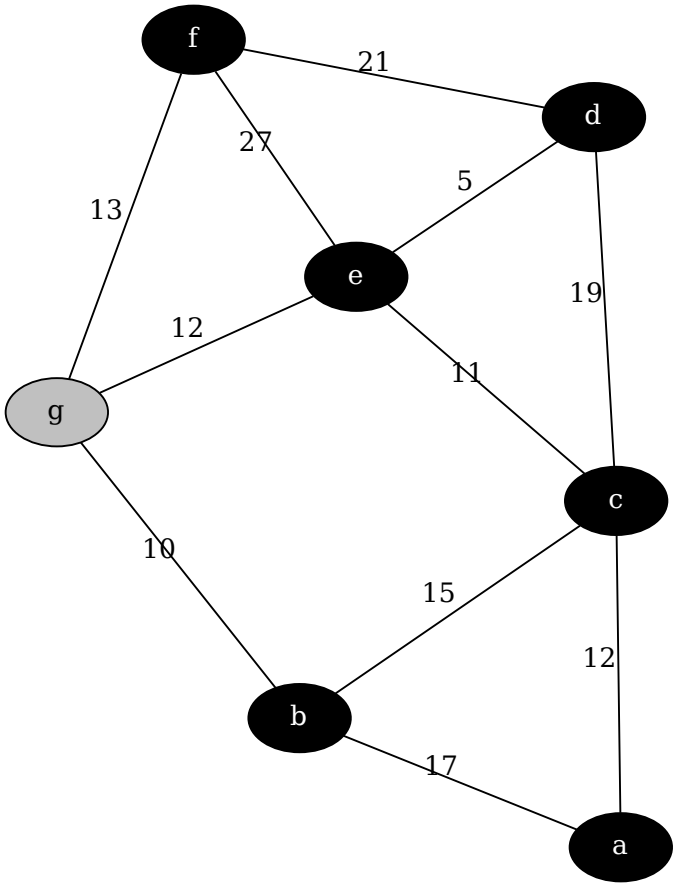
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



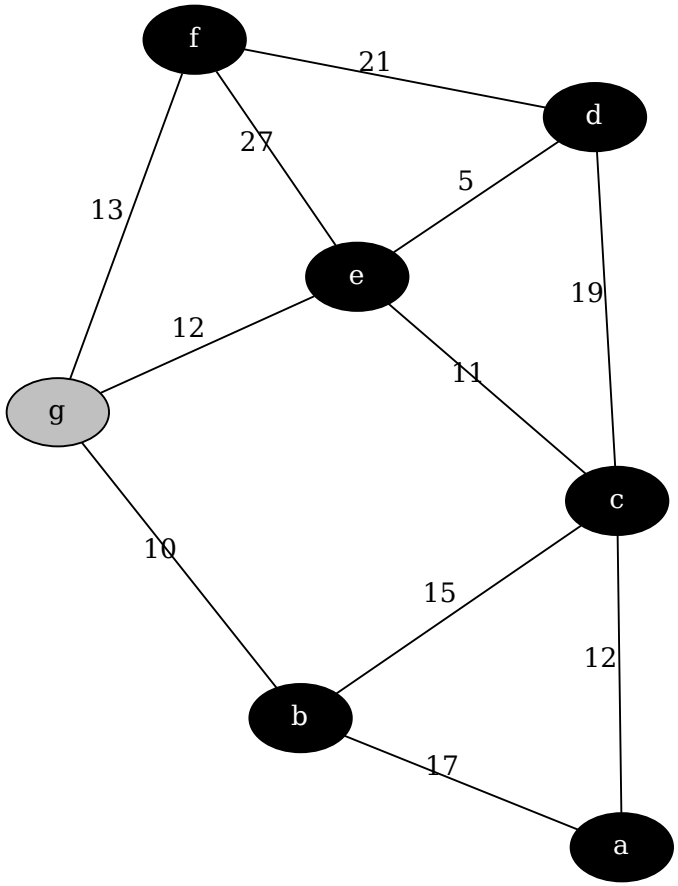
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	26	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



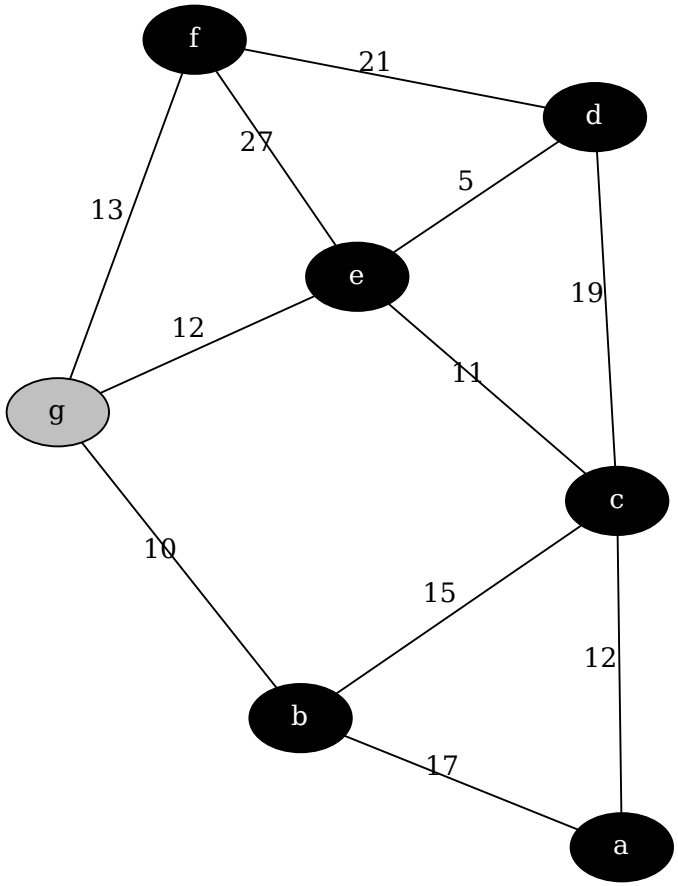
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



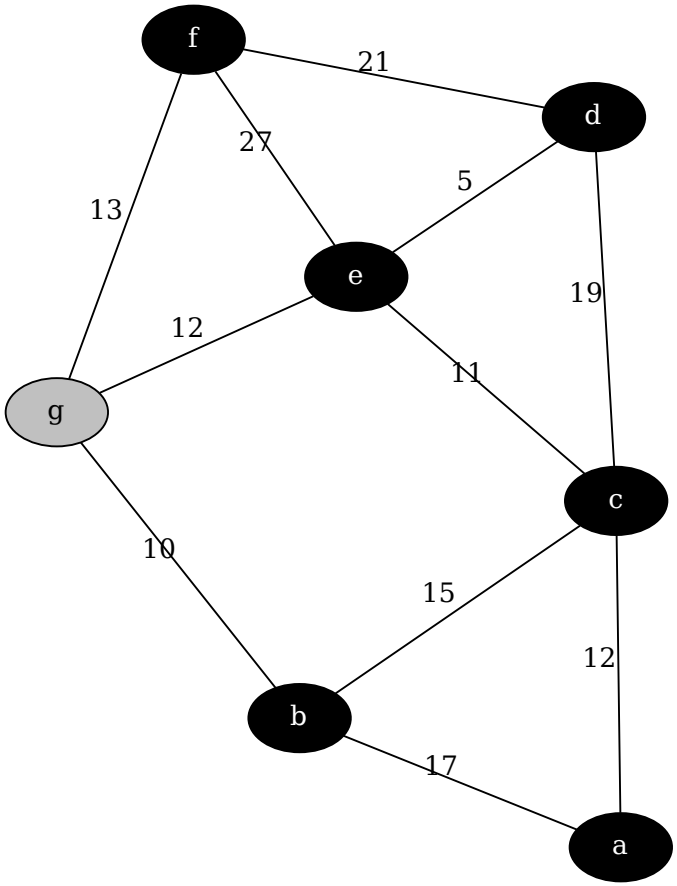
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



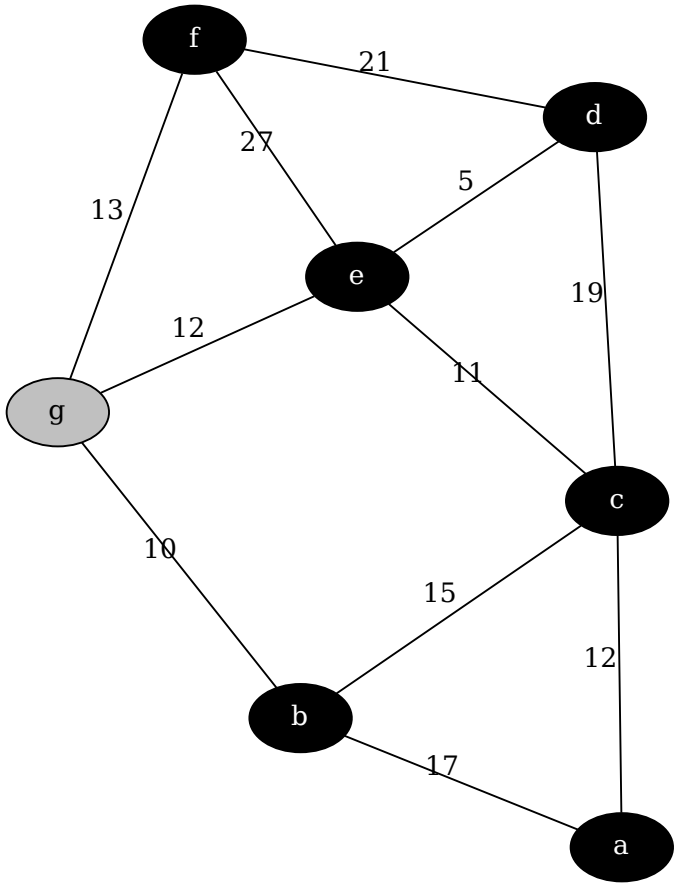
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	52	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



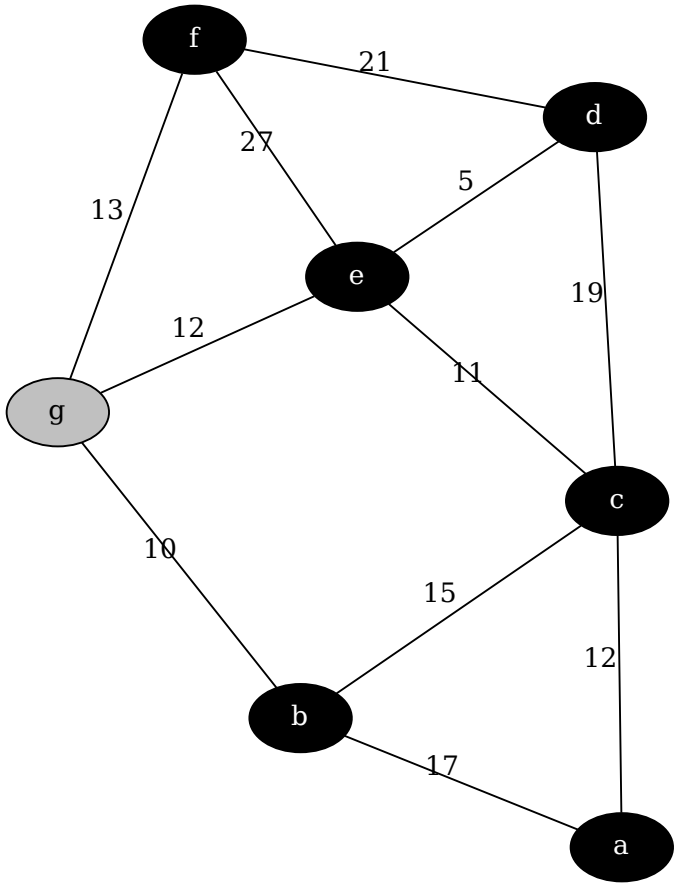
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



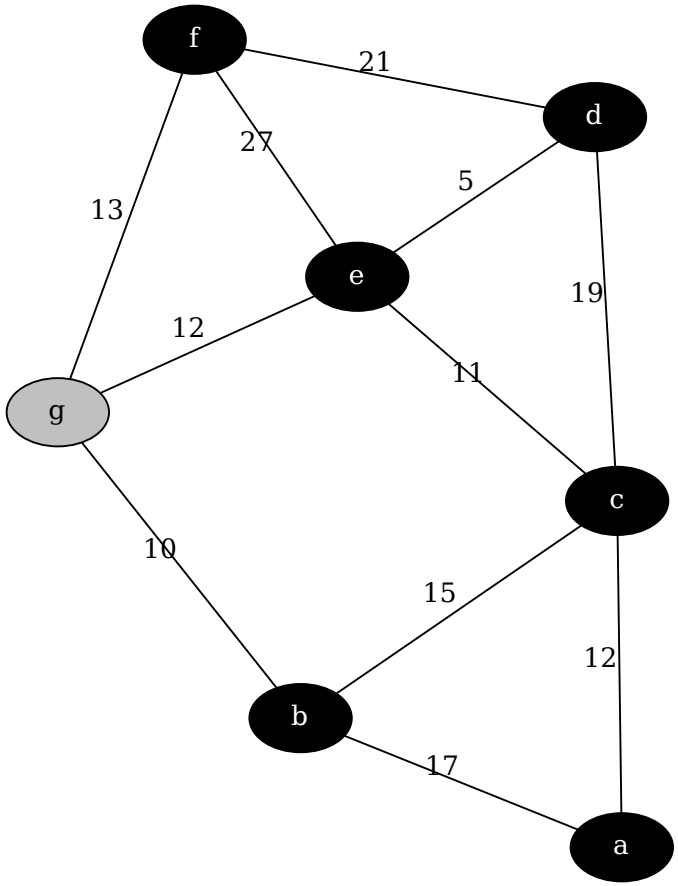
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



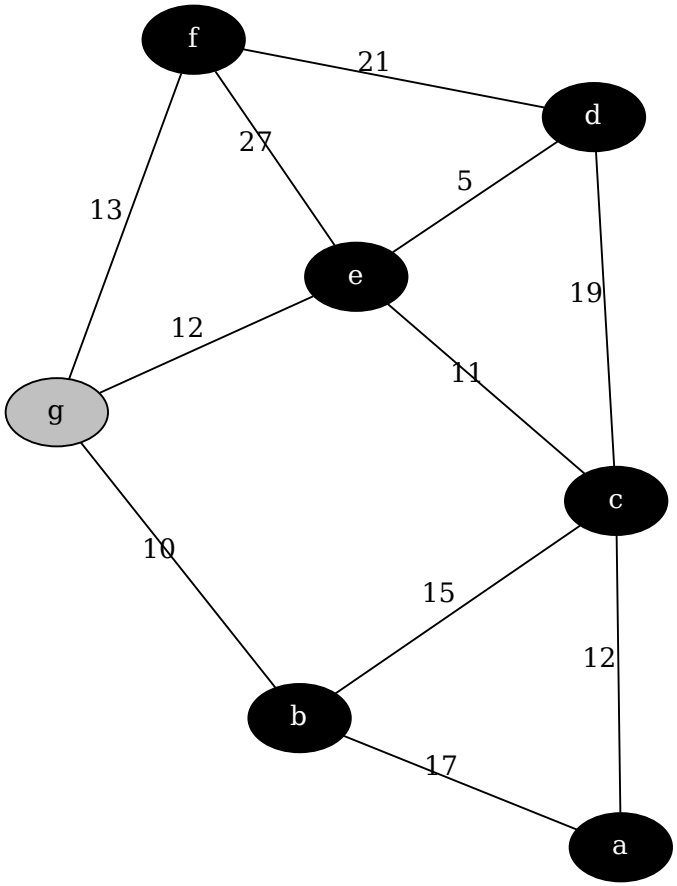
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



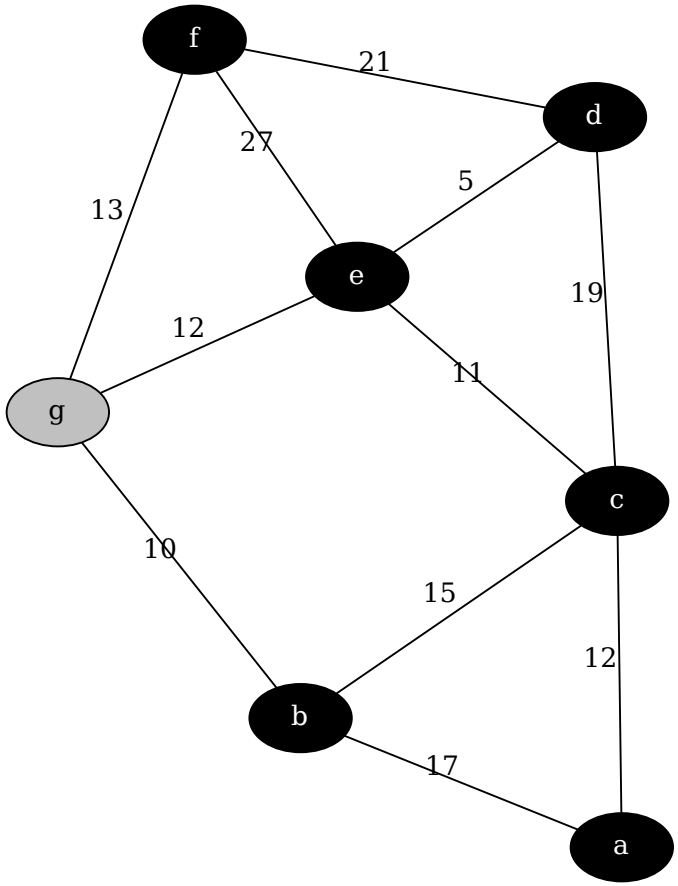
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



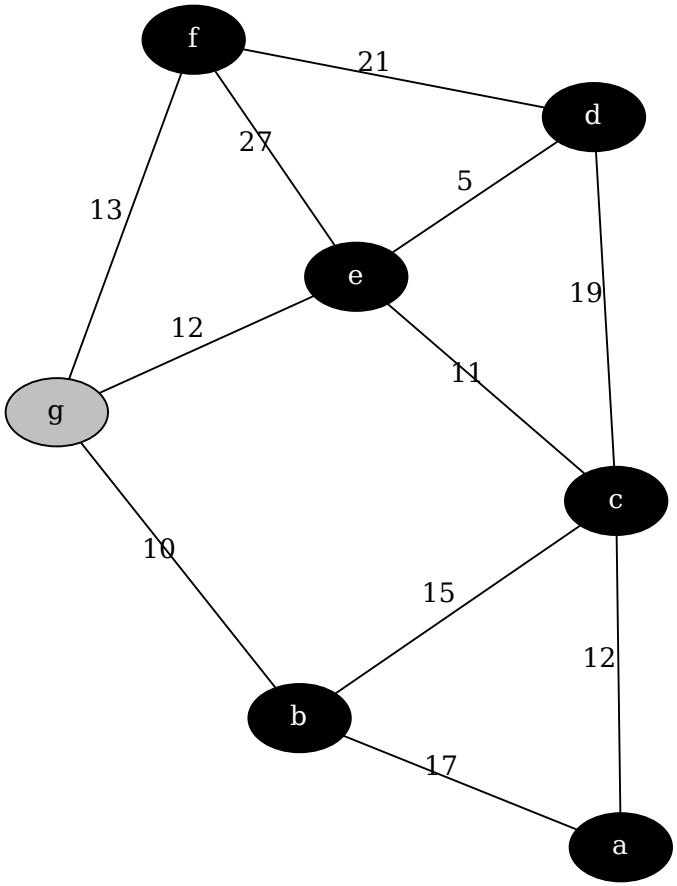
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	37	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



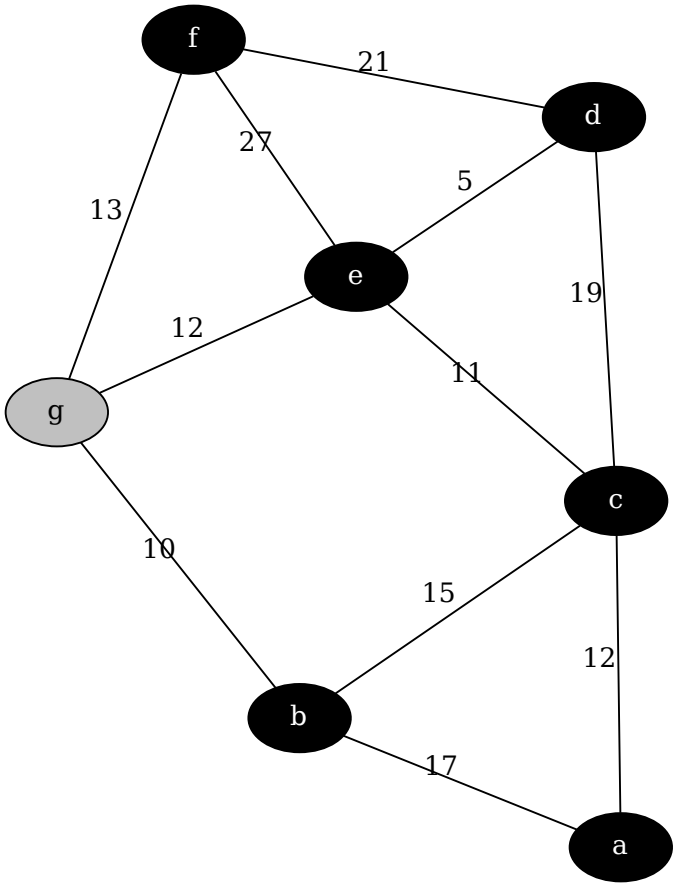
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	36	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



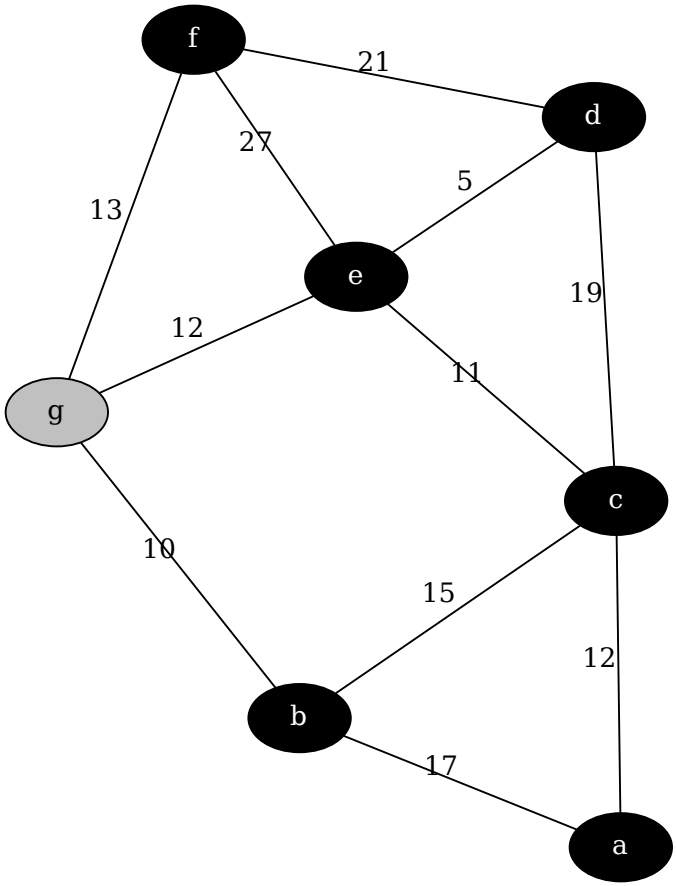
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	36	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



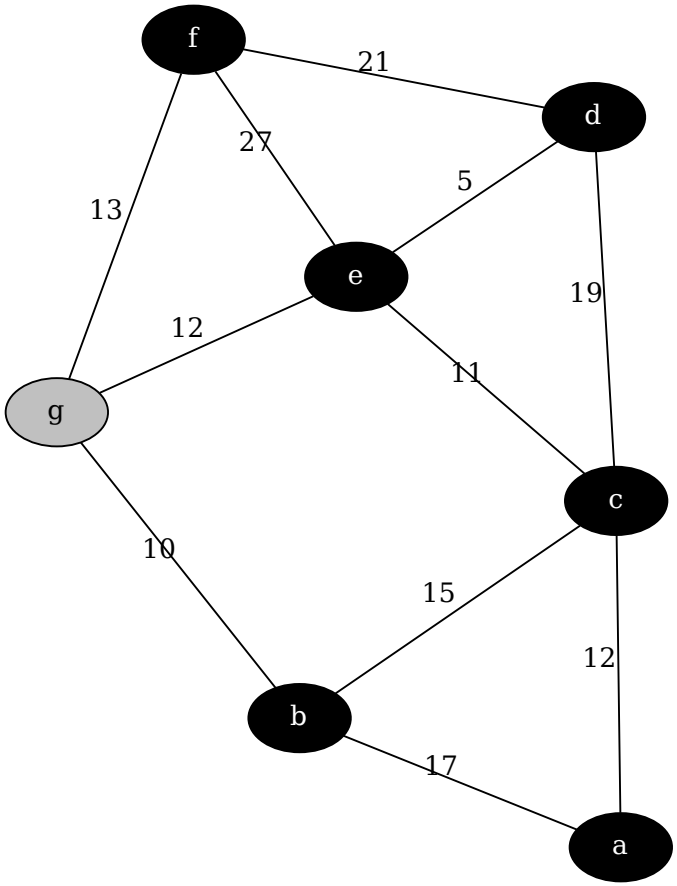
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	36	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



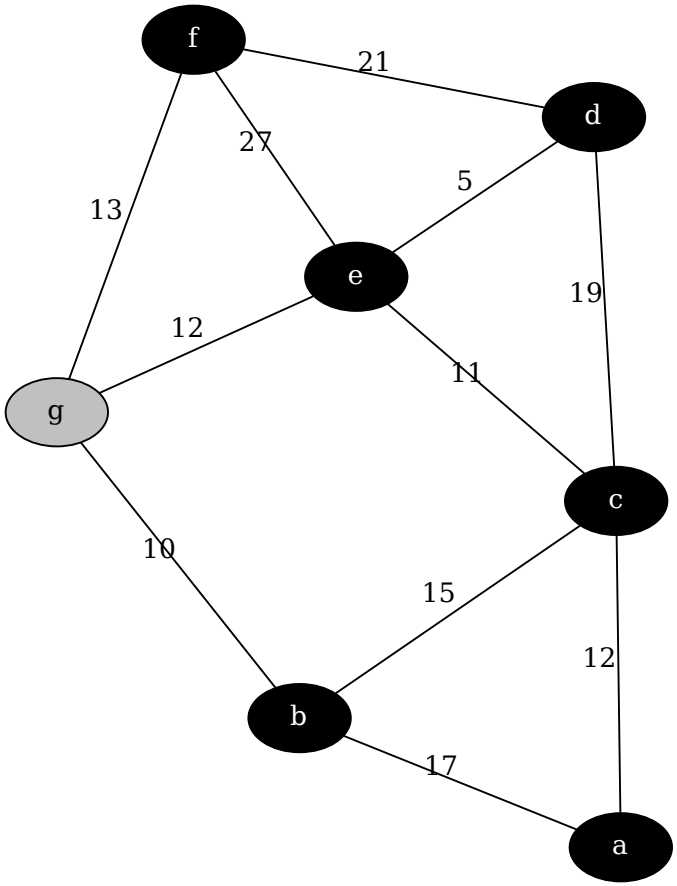
	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	36	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



	a	b	c	d	e	f	g
a	0	17	12	28	23	40	27
b		0	15	27	22	23	10
c			0	16	11	36	23
d				0	5	21	17
e					0	26	12
f						0	13
g							0

Example



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

APSP state of the art

- Last week's algorithm: $O(n^3 \log n)$ arithmetic operations.
- Floyd-Warshall: $O(n^3)$ arithmetic operations.
- Can we do better?

APSP state of the art

- Last week's algorithm: $O(n^3 \log n)$ arithmetic operations.
- Floyd-Warshall: $O(n^3)$ arithmetic operations.
- Can we do better?
- In general, we conjecture that **NO**, but this is a major open problem.
- What if the input graph is sparse (e.g. $m = O(n)$)?
  - $n \times$Dijkstra $\Rightarrow O(n(m+n)\log n)$
  - $n \times$BF $\Rightarrow O(n^2 m)$
- First is quadratic for sparse graphs, but only for positive weights.
- Second is worse than Floyd-Warshall, even for sparse graphs...

APSP state of the art

- Last week's algorithm: $O(n^3 \log n)$ arithmetic operations.
- Floyd-Warshall: $O(n^3)$ arithmetic operations.
- Can we do better?
- In general, we conjecture that **NO**, but this is a major open problem.
- What if the input graph is sparse (e.g. $m = O(n)$)?
  - $n\times$Dijkstra $\Rightarrow O(n(m+n)\log n)$
  - $n\times$BF $\Rightarrow O(n^2 m)$
- First is quadratic for sparse graphs, but only for positive weights.
- Second is worse than Floyd-Warshall, even for sparse graphs...

Questions:

- $O(nm)$ APSP with negative weights?
- $O(m^{2-\varepsilon})$ with positive weights?

Johnson's algorithm

Johnson's algorithm

- APSP in digraphs with negative weights faster than $n \times$BF.
- Strategy:
  - Run Bellman-Ford **once** to ensure no negative cycles.
  - Adjust edge weights so all are non-negative.
  - Run Dijkstra n times.

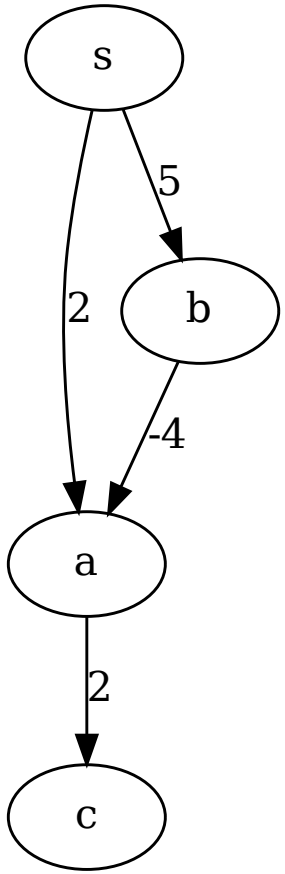
Johnson's algorithm

- APSP in digraphs with negative weights faster than $n \times$BF.
- Strategy:
  - Run Bellman-Ford **once** to ensure no negative cycles.
  - Adjust edge weights so all are non-negative.
  - Run Dijkstra n times.

Do we expect this to work??

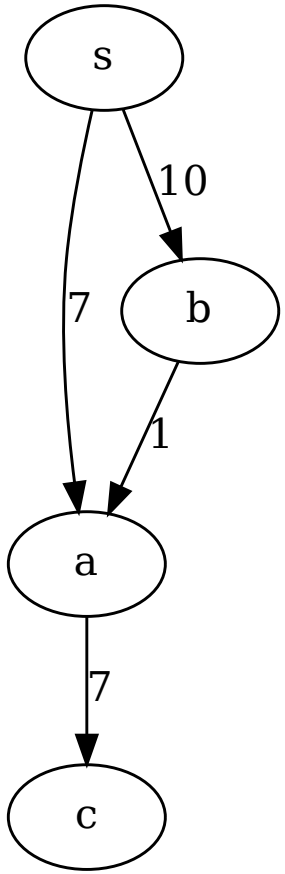
Reminder: An easy FAILED fix

- Suggestion: eliminate negative weights
- Find the max absolute value of any negative weight M.
- Add $M + 1$ to all weights.
- Run Dijkstra.
- Why not?



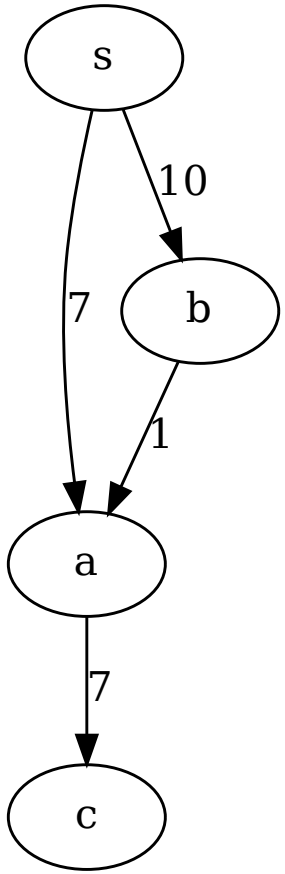
Reminder: An easy FAILED fix

- Suggestion: eliminate negative weights
- Find the max absolute value of any negative weight M.
- Add $M + 1$ to all weights.
- Run Dijkstra.
- Why not?



Reminder: An easy FAILED fix

- Suggestion: eliminate negative weights
- Find the max absolute value of any negative weight M.
- Add $M + 1$ to all weights.
- Run Dijkstra.
- Why not?
- Gives unfair advantage to paths with fewer edges...



Johnson's fix

- Will compute a potential $h(v)$ for every vertex u.
- Modify weights: $w'(uv) \leftarrow w(uv) + h(u) - h(v)$.
- **Key intuition:** in every $s \rightarrow t$ path, potentials of internal vertices **cancel out**.
- We just need to compute $h()$ so that w' is nowhere negative.

Johnson's fix

- Will compute a potential $h(v)$ for every vertex u.
- Modify weights: $w'(uv) \leftarrow w(uv) + h(u) - h(v)$.
- **Key intuition:** in every $s \to t$ path, potentials of internal vertices **cancel out**.
- We just need to compute $h()$ so that w' is nowhere negative.

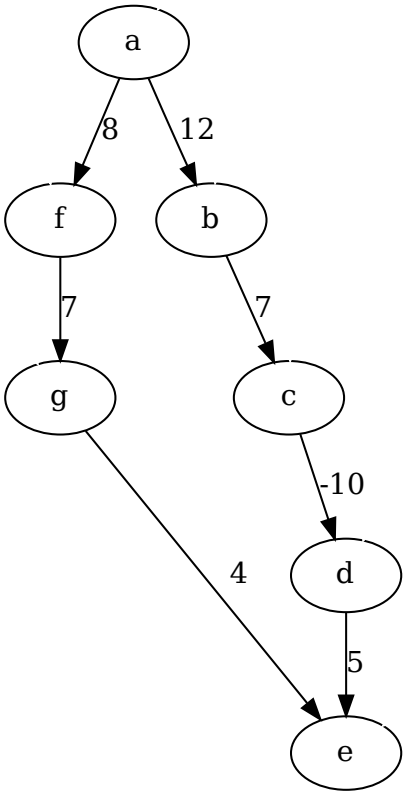
How to compute $h()$:

- Add universal source s to the graph.
- Run Bellman-Ford from $s \Rightarrow \forall v$ set $h(v) = \mathrm{dist}(s, v)$.

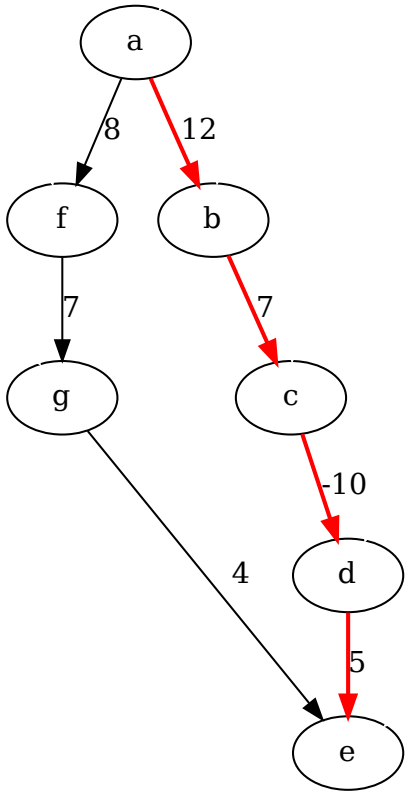
Johnson's algorithm

1: Add to G a universal source s with arcs of weight 0 to everyone.
2: Bellman-Ford(G,s)
3: **for** $ij \in E$ **do**
4: $\quad w'(ij) \leftarrow w(ij) + \mathrm{dist}(s,i) - \mathrm{dist}(s,j)$
5: **end for**
6: **for** $i \in V$ **do**
7: $\quad$ Run Dijkstra from i with weight function w'
8: **end for**

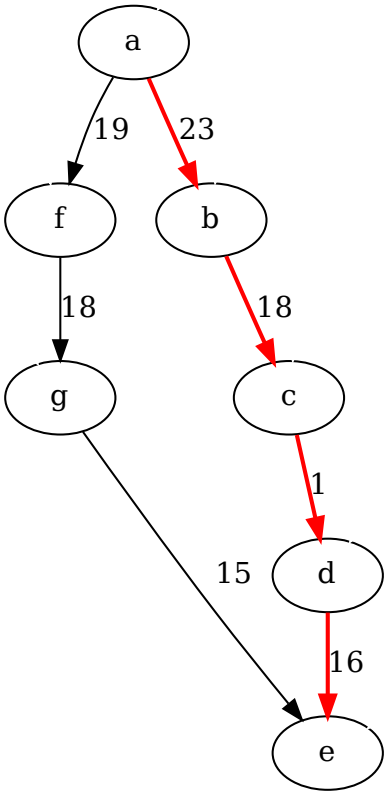
Example



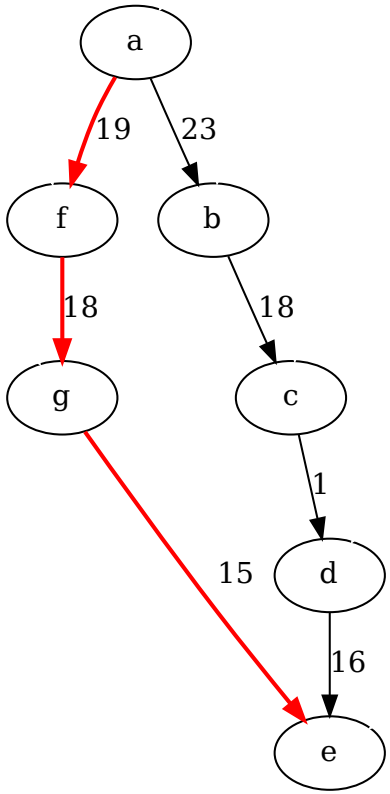
Example



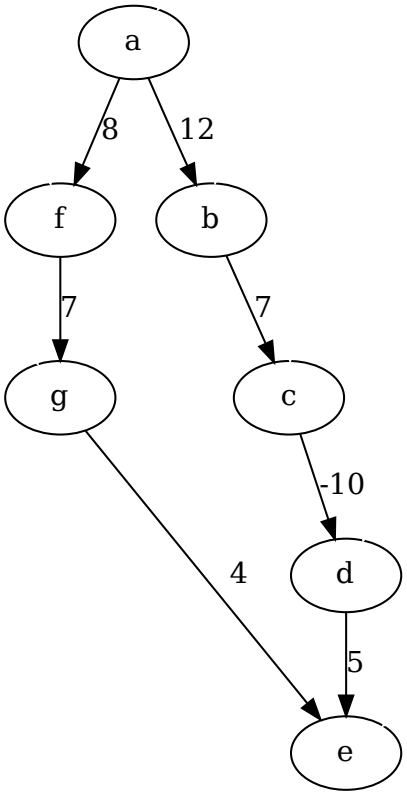
Example



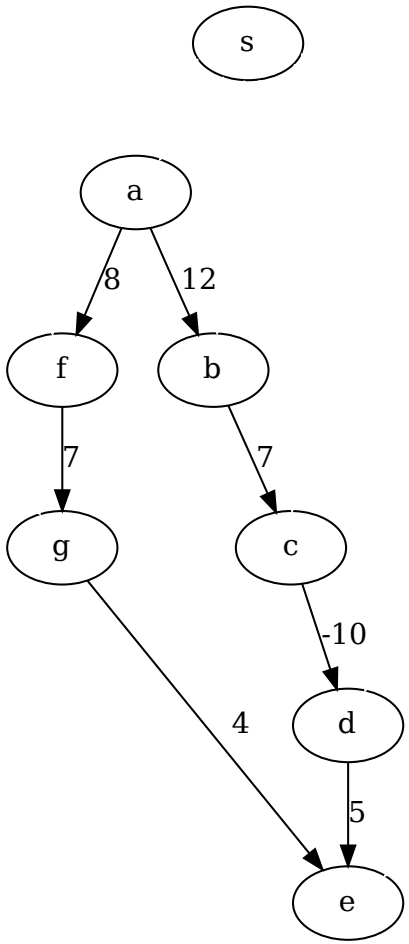
Example



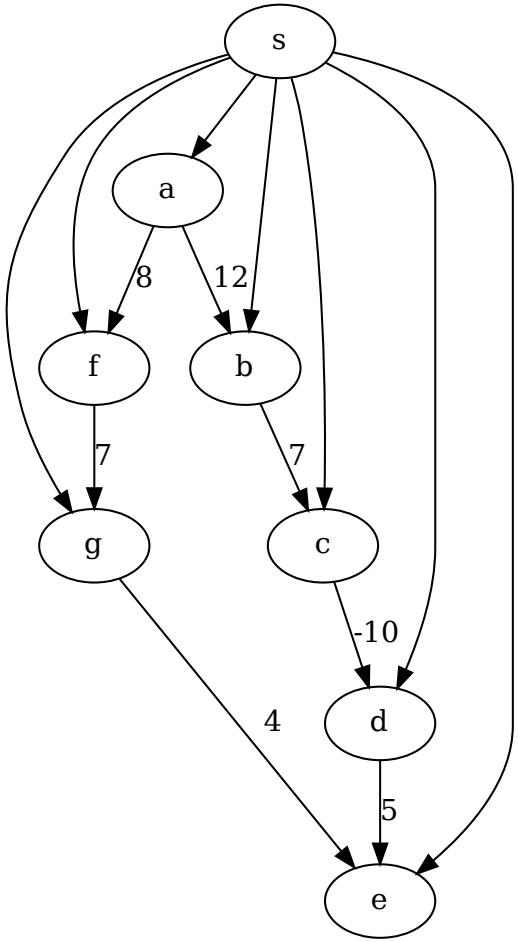
Example



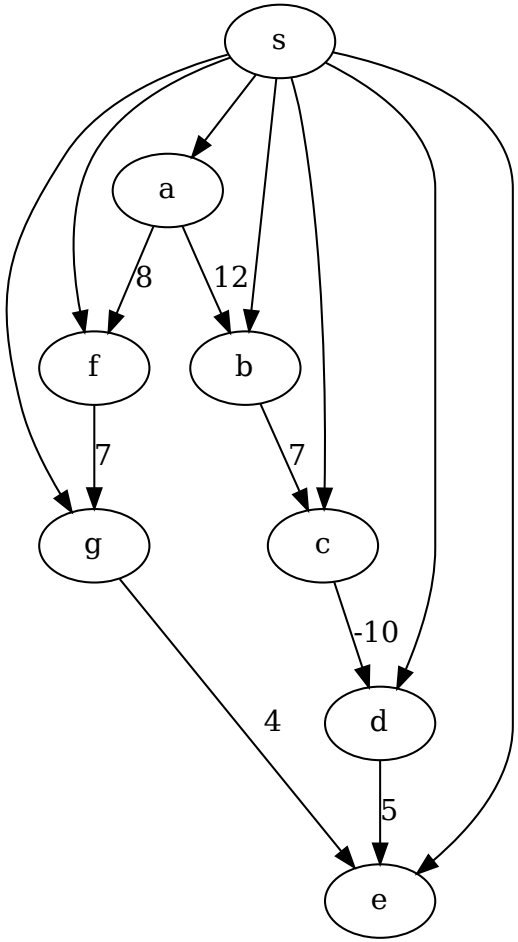
Example



Example



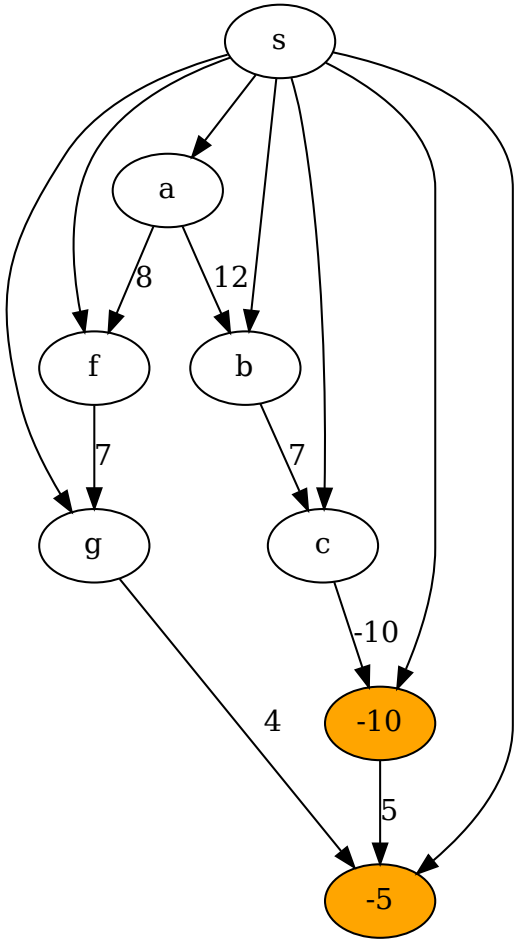
Example



Shortest path distances from s:

a	b	c	d	e	f	g
0	0	0	-10	-5	0	0

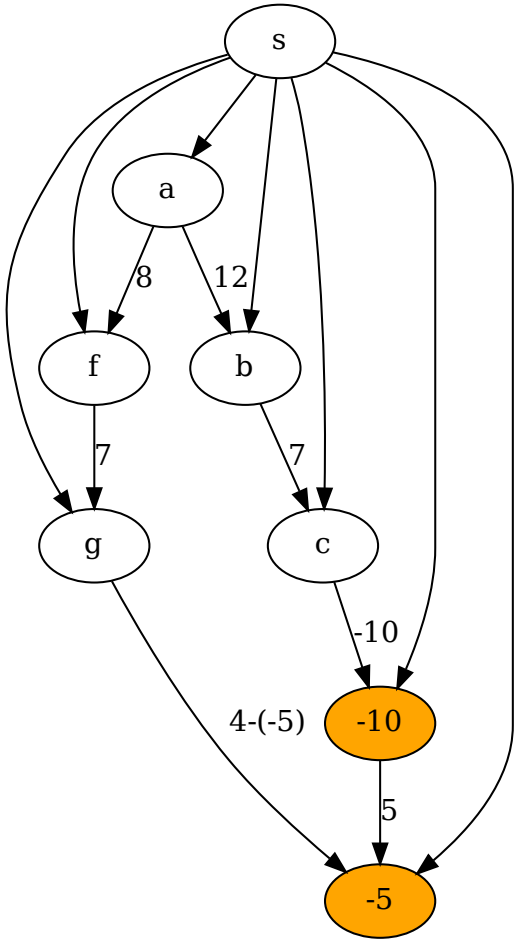
Example



Shortest path distances from s:

a	b	c	d	e	f	g
0	0	0	-10	-5	0	0

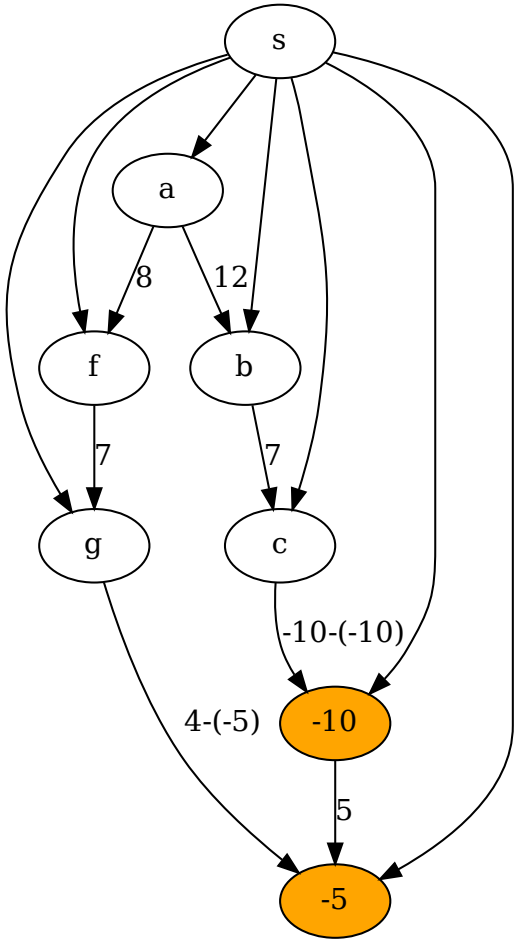
Example



Shortest path distances from s:

a	b	c	d	e	f	g
0	0	0	-10	-5	0	0

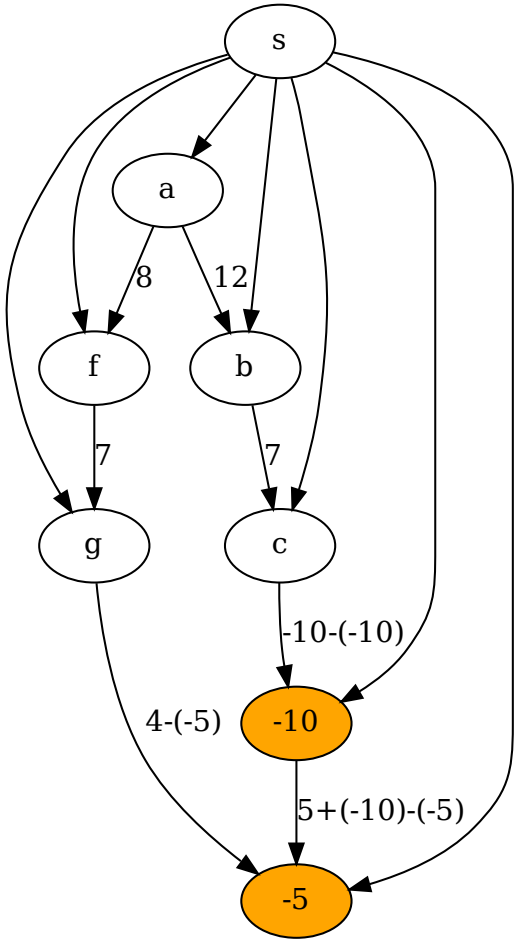
Example



Shortest path distances from s:

a	b	c	d	e	f	g
0	0	0	-10	-5	0	0

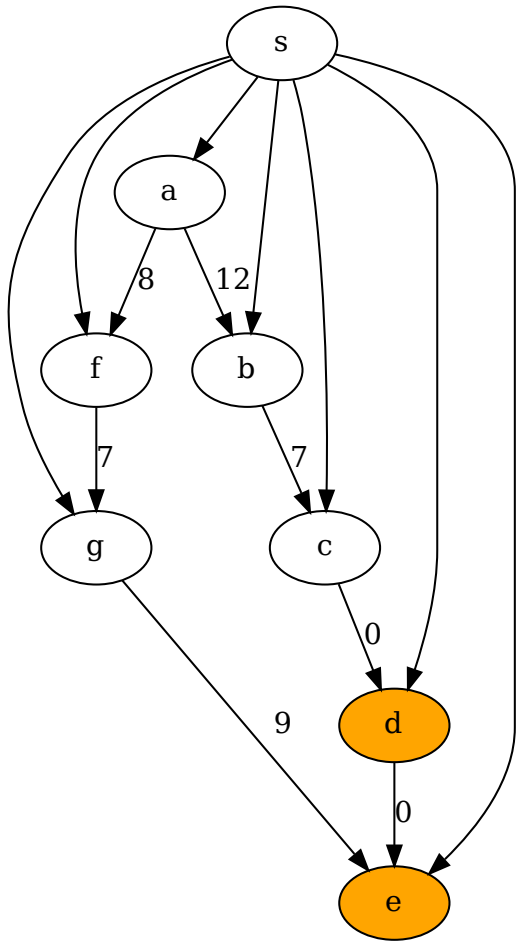
Example



Shortest path distances from s:

a	b	c	d	e	f	g
0	0	0	-10	-5	0	0

Example



Shortest path distances from s:

a	b	c	d	e	f	g
0	0	0	-10	-5	0	0

Analysis

- Running time is $O(nm)$ for BF and $O(n(n+m)\log n)$ for $n\times$Dijkstra
  - $O(nm \log n)$ (compare with $n\times$BF)
- Space complexity is same as for BF and Dijkstra ($O(n)$)
  - Note: output is $n \times n$ matrix $\Rightarrow$ size $O(n^2)$
  - But we generate it row-by-row.

Correctness

Lemma

G has a negative cycle if and only if $G + s$ does.

Lemma

A path P is a shortest $s \to t$ path for w if and only if it is a shortest $s \to t$ path from w'.

Lemma

$w'(ij) \geq 0$ for all $i, j \in V$.

Correctness

Lemma

G has a negative cycle if and only if $G + s$ does.

Correctness

Lemma

G has a negative cycle if and only if $G + s$ does.

Proof.

Easy: no cycle goes through s (only outgoing arcs). □

Correctness

Lemma

G has a negative cycle if and only if $G + s$ does.

Proof.

Easy: no cycle goes through s (only outgoing arcs). □

Lemma

A path P is a shortest $s \to t$ path for w if and only if it is a shortest $s \to t$ path from w'.

Correctness

Lemma

G has a negative cycle if and only if $G + s$ does.

Proof.

Easy: no cycle goes through s (only outgoing arcs). □

Lemma

A path P is a shortest $s \to t$ path for w if and only if it is a shortest $s \to t$ path from w'.

Proof.

Cost of any $s \to t$ paths changes by $h(s) - h(t)$:

- $P = s, x_1, x_2, \ldots, x_k, t \Rightarrow w'(P) = (w(sx_1) + h(s) - h(x_1)) + (w(sx_2) + h(x_1) - h(x_2)) + \ldots + (w(x_k t) + h(x_k) - h(t)) = w(P) + h(s) - h(t)$.

□

Correctness

Lemma

$w'(ij) \geq 0$ *for all* $i, j \in V$.

Correctness

Lemma

$w'(ij) \geq 0$ *for all* $i, j \in V$.

Proof.

- $w'(ij) = w(ij) + h(i) - h(j) \geq 0 \Leftrightarrow h(j) \leq w(ij) + h(i)$
- $\Leftrightarrow \mathrm{dist}(s, j) \leq \mathrm{dist}(s, i) + w(ij)$.
- Follows from triangle inequality.

Conditional Impossibility Results

State of the art

- Best algorithms for APSP take $O(n^3)$ time in worst case (even for positive weights).
- APSP for sparse graphs can be solved in $O(nm \log n)$ time (even for negative weights).

Can we do better?

State of the art

- Best algorithms for APSP take $O(n^3)$ time in worst case (even for positive weights).
- APSP for sparse graphs can be solved in $O(nm \log n)$ time (even for negative weights).

Can we do better?

- Is there a sub-cubic algorithm for APSP?
- Is there a sub-quadratic ($O(m^{2-\varepsilon})$) algorithm if we are promised that $m = O(n)$?

APSP-equivalence

Is there a sub-cubic algorithm for APSP?

APSP-equivalence

Is there a sub-cubic algorithm for APSP?
We don't know!

APSP-equivalence

Is there a sub-cubic algorithm for APSP?
We don't know!

Theorem

There is a sub-cubic algorithm for APSP if and only if there is a sub-cubic algorithm for one of the following:

- *Computing the diameter*
- *Finding a negative triangle*
- *Finding the minimum-weight cycle*
- *Testing if a matrix satisfies triangle inequality*
- *...*

Idea: either **all** these problems are easy, or **all** are hard...

Sub-quadratic on sparse graphs?

Is there a sub-quadratic ($O(m^{2-\varepsilon})$) algorithm for APSP if we are promised that $m = O(n)$?

Sub-quadratic on sparse graphs?

Is there a sub-quadratic ($O(m^{2-\varepsilon})$) algorithm for APSP if we are promised that $m = O(n)$?

We don't know!

Sub-quadratic on sparse graphs?

Is there a sub-quadratic ($O(m^{2-\varepsilon})$) algorithm for APSP if we are promised that $m = O(n)$?

We don't know!

- Some care must be taken in defining the problem, as output has size $n^2 \Rightarrow$ no algorithm can produce it in sub-quadratic time.
- Instead, will consider closely related **diameter** problem: given G, compute the distance between $x, y \in V$ which are at maximum distance.

Sub-quadratic on sparse graphs?

Is there a sub-quadratic ($O(m^{2-\varepsilon})$) algorithm for APSP if we are promised that $m = O(n)$?

We don't know!

- Some care must be taken in defining the problem, as output has size $n^2 \Rightarrow$ no algorithm can produce it in sub-quadratic time.
- Instead, will consider closely related **diameter** problem: given G, compute the distance between $x, y \in V$ which are at maximum distance.
- We have **some** reasons to believe diameter needs at least quadratic time, even if $m = O(n)$ and weights are positive.

Boolean CNF Satisfiability

Boolean CNF Satisfiability

Attention: NOT a GRAPH PROBLEM!

Boolean CNF Satisfiability

- n Boolean variables $x_1, x_2, \ldots, x_n$.
- **Literal**: x_i or $\neg x_i$.
- **Clause**: A disjunction of literals.
  - Examples: $(x_1 \vee \neg x_2 \vee x_3)$, $(x_4 \vee \neg x_5)$
- **CNF Formula**: A conjunction of clauses.
  - CNF: Conjunctive Normal Form
  - Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge \ldots$
- SAT: Given a CNF formula, decide if it is SATISFIABLE:
  - Does there exist a Boolean assignment to the variables that makes all clauses True?

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$ satisfiable?

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 0, x_3 = 1$ satisfies it.

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 0, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_1 \vee \neg x_3)$ satisfiable?

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 0, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_1 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 0, x_2 = 0, x_3 = 1$ satisfies it.

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 0, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_1 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 0, x_2 = 0, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2)$ satisfiable?

Example

- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_2 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 1, x_2 = 0, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_1 \vee \neg x_3)$ satisfiable?
  - Yes! $x_1 = 0, x_2 = 0, x_3 = 1$ satisfies it.
- Is $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2)$ satisfiable?
  - No! (but proving it seems a little more ad-hoc)

SAT - state of the art

- Without (much) exageration, SAT is **the most important** algorithmic problem.
- Intuition: many problems have form "I'm looking for a good solution. I'll know it when I see it!"
  - Example: Bitcoin mining uses Proof-of-Work. We have a simple hash function $f()$ and a target y. We need to find x such that $f(x) = y$.
  - Assumption: Must try out all possible x values.
- Key intuition: verification function f can be encoded as a CNF formula $\Rightarrow$ solving SAT solves a generic search problem!

SAT - state of the art

- Without (much) exageration, SAT is **the most important** algorithmic problem.
- Intuition: many problems have form "I'm looking for a good solution. I'll know it when I see it!"
  - Example: Bitcoin mining uses Proof-of-Work. We have a simple hash function $f()$ and a target y. We need to find x such that $f(x) = y$.
  - Assumption: Must try out all possible x values.
- Key intuition: verification function f can be encoded as a CNF formula $\Rightarrow$ solving SAT solves a generic search problem!

Two possible conclusions:

- If you solve SAT you will be rich!

SAT - state of the art

- Without (much) exageration, SAT is **the most important** algorithmic problem.
- Intuition: many problems have form "I'm looking for a good solution. I'll know it when I see it!"
  - Example: Bitcoin mining uses Proof-of-Work. We have a simple hash function $f()$ and a target y. We need to find x such that $f(x) = y$.
  - Assumption: Must try out all possible x values.
- Key intuition: verification function f can be encoded as a CNF formula $\Rightarrow$ solving SAT solves a generic search problem!

Two possible conclusions:

- If you solve SAT you will be rich!
- Solving SAT efficiently is probably impossible.

SAT in complexity theory

- SAT is the central problem of computational complexity theory.
- Does SAT have a poly-time algorithm?
  - Equivalent to P=NP? question.
  - Technically open ($1,000,000$\$ prize!), consensus is P$\neq$NP.

SAT in complexity theory

- SAT is the central problem of computational complexity theory.
- Does SAT have a poly-time algorithm?
  - Equivalent to P=NP? question.
  - Technically open ($1,000,000$\$ prize!), consensus is P$\neq$NP.
- Best algorithm essentially 2^n time.
- Does SAT have a $(1.99)^n$-time algorithm?

SAT in complexity theory

- SAT is the central problem of computational complexity theory.
- Does SAT have a poly-time algorithm?
  - Equivalent to P=NP? question.
  - Technically open (1,000,000$ prize!), consensus is P≠NP.
- Best algorithm essentially 2^n time.
- Does SAT have a $(1.99)^n$-time algorithm?
  - At the moment **NO!**
  - Conjecture that no such algorithm exists is called Strong Exponential Time Hypothesis (SETH).
  - **Wide open** problem!

SAT in complexity theory

- SAT is the central problem of computational complexity theory.
- Does SAT have a poly-time algorithm?
  - Equivalent to P=NP? question.
  - Technically open (1,000,000$ prize!), consensus is P≠NP.
- Best algorithm essentially 2^n time.
- Does SAT have a $(1.99)^n$-time algorithm?
  - At the moment **NO!**
  - Conjecture that no such algorithm exists is called Strong Exponential Time Hypothesis (SETH).
  - **Wide open** problem!
- Fine, what does this have to do with graph algorithms?

SAT to diameter reduction

- High-level idea: **reduce** SAT to a diameter computation, so that if we have a fast algorithm for computing the diameter, we have a fast algorithm for SAT.

SAT to diameter reduction

- High-level idea: **reduce** SAT to a diameter computation, so that if we have a fast algorithm for computing the diameter, we have a fast algorithm for SAT.
- Given: CNF formula with n variables, m clauses.
- Output: G with $N = O(2^{n/2} + m)$ vertices, $M = O(2^{n/2} + m)$ edges.
- G has diameter 3 if formula is satisfiable.
- G has diameter 2 if formula is not satisfiable.

SAT to diameter reduction

- High-level idea: **reduce** SAT to a diameter computation, so that if we have a fast algorithm for computing the diameter, we have a fast algorithm for SAT.
- Given: CNF formula with n variables, m clauses.
- Output: G with $N = O(2^{n/2} + m)$ vertices, $M = O(2^{n/2} + m)$ edges.
- G has diameter 3 if formula is satisfiable.
- G has diameter 2 if formula is not satisfiable.
- If $M^{1.99}$ time algorithm exists for diameter, use it to decide satisfiability in time $(2^{n/2})^{1.99} < (1.99999)^n$.
- Breakthrough in diameter computation $\Rightarrow$ breakthrough for SAT

SAT to diameter reduction

Sketch:

- Partition variables into two sets $x_1, \ldots, x_{n/2}$ and $x_{n/2+1}, \ldots, x_n$.
- Construct one vertex for each truth assignment to each set $\Rightarrow 2 \cdot 2^{n/2}$ vertices so far. Call these sets L, R.
- Construct a vertex for each clause. For clause c and assignment σ put the edge $c\sigma$ if σ **fails to satisfy** c.
- Add a common neighbor to L, a common neighbor to R, make them adjacent to each other and all clauses.

SAT to diameter reduction

Sketch:

- Partition variables into two sets $x_1, \ldots, x_{n/2}$ and $x_{n/2+1}, \ldots, x_n$.
- Construct one vertex for each truth assignment to each set $\Rightarrow 2 \cdot 2^{n/2}$ vertices so far. Call these sets L, R.
- Construct a vertex for each clause. For clause c and assignment σ put the edge $c\sigma$ if σ **fails to satisfy** c.
- Add a common neighbor to L, a common neighbor to R, make them adjacent to each other and all clauses.
- Only two vertices (possibly) at distance 3: an assignment from L and an assignment from R which have no common neighbor $\Rightarrow$ form a satisfying assignment.