Graph theory

about graph theory

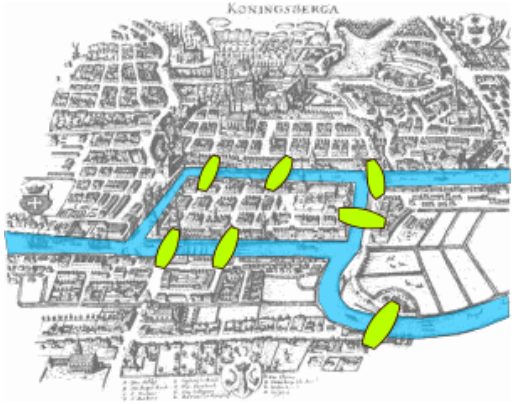
graph

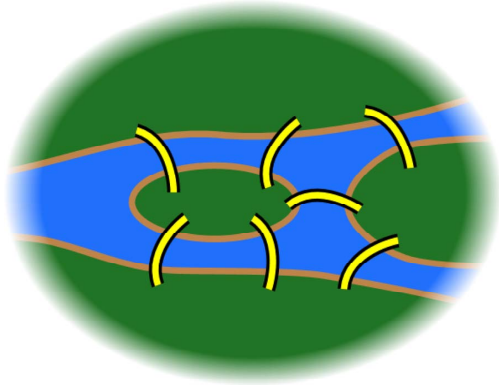
minimum spanning tree

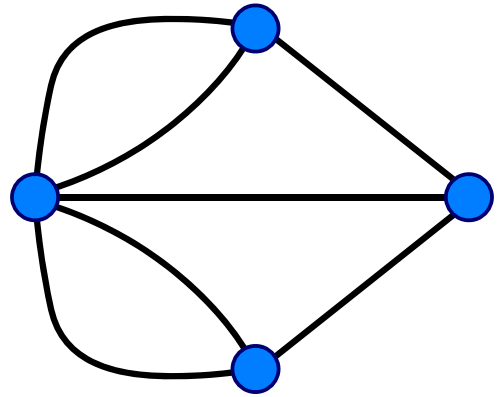
shortest path problem

maximum flow

seven bridges of Königsberg







- a necessary condition for the walk of the desired form is that the graph be connected and have exactly zero or two nodes of odd degree
- this condition turns out also to be sufficient
- such a walk is now called an Eulerian path or Euler walk in his honor
- if there are nodes of odd degree, then any Eulerian path will start at one of them and end at the other, such a walk is called an Eulerian circuit or an Euler tour
- such a circuit exists if, and only if, the graph is connected, and there are no nodes of odd degree at all

chinese postman problem

was originally studied by the Chinese mathematician Kwan Mei-Ko in 1960

Alan Goldman of the U.S. National Bureau of Standards first coined the name 'Chinese Postman Problem'

- to find a shortest closed path or circuit that visits every edge of a (connected) undirected graph
- when the graph has an Eulerian circuit (a closed walk that covers every edge once), that circuit is an optimal solution
- otherwise, the optimization problem is to find the fewest number of edges to add to the graph so that the resulting multigraph does have an Eulerian circuit

travelling salesman problem

given a list of cities and the distances between each pair of cities
what is the shortest possible route that visits each city exactly once and returns to the origin city?

- TSP can be modelled as an undirected weighted graph
- cities are the graph's vertices
- paths are the graph's edges, a path's distance is the edge's length

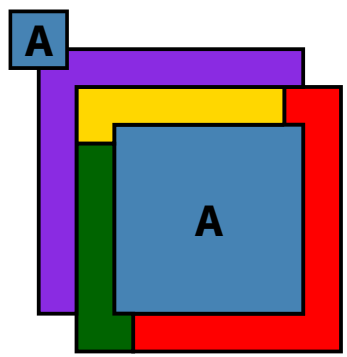
it is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once

- a Hamiltonian path (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once
- a Hamiltonian cycle is a Hamiltonian path that is a cycle
- determining whether such paths and cycles exist in graphs is the Hamiltonian path problem, which is NP-complete
- Tutte (1956): a 4-connected planar graph has a Hamiltonian cycle

four color theorem

given any separation of a plane into contiguous regions, producing a figure called a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color

- two regions are called adjacent if they share a common boundary that is not a corner
- bizarre maps (using regions of finite area but infinite perimeter) can require more than four colors
- for the purpose of the theorem, every "country" has to be a connected region, or contiguous



Graph

graph: a graph is an ordered pair $G = (V, E)$ comprising a set V of vertices, nodes or points together with a set E of edges, arcs or lines, which are 2-element subsets of V

undirected graph: an undirected graph is a graph in which edges have no orientation.

directed graph: a directed graph or digraph is a graph in which edges have orientations.

simple graph: a simple graph, as opposed to a multigraph, is an undirected graph in which both multiple edges and loops are disallowed.

weighted graph: a weighted graph is a graph in which a number (the weight) is assigned to each edge.

connected graph:

- In an undirected graph, an unordered pair of vertices $\{x, y\}$ is called connected if a path leads from x to y. Otherwise, the unordered pair is called disconnected.
- A connected graph is an undirected graph in which every unordered pair of vertices in the graph is connected. Otherwise, it is called a disconnected graph.
- In a directed graph, an ordered pair of vertices (x, y) is called strongly connected if a directed path leads from x to y. Otherwise, the ordered pair is called weakly connected if an undirected path leads from x to y after replacing all of its directed edges with undirected edges. Otherwise, the ordered pair is called disconnected.
- A strongly connected graph is a directed graph in which every ordered pair of vertices in the graph is strongly connected. Otherwise, it is called a weakly connected graph if every ordered pair of vertices in the graph is weakly connected. Otherwise it is called a disconnected graph.

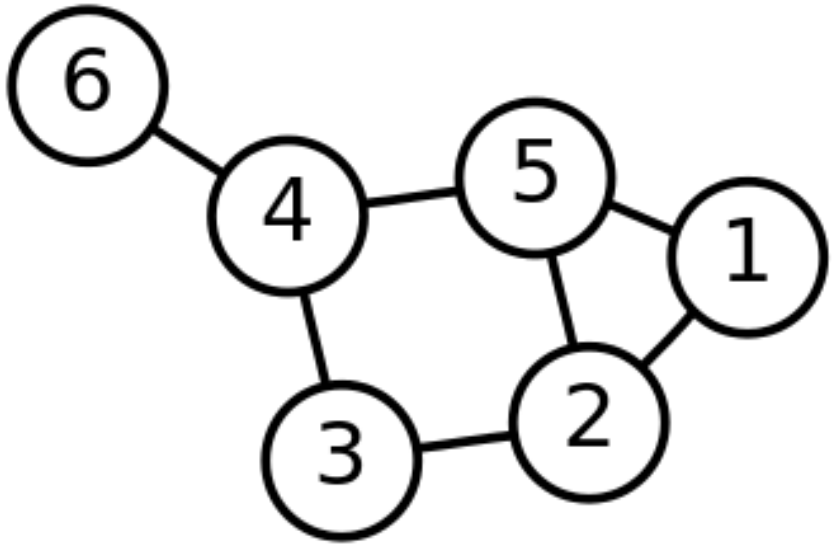
regular graph: a regular graph is a graph in which each vertex has the same number of neighbours, i.e., every vertex has the same degree.

complete graph: a complete graph is a graph in which each pair of vertices is joined by an edge.

bipartite graph: A bipartite graph is a graph in which the vertex set can be partitioned into two sets.

planar graph: a planar graph is a graph whose vertices and edges can be drawn in a plane such that no two of the edges intersect.

tree: a tree is a connected graph with no cycles. a forest is a graph with no cycles, i.e. the disjoint union of one or more trees.



The diagram is a graphic representation of the following graph $G = (V, E)$:

$V = \{1, 2, 3, 4, 5, 6\}$

$E = \big\{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 4\}, \{4, 5\}, \{4, 6\}\big\}$

adjacency matrix

For a simple graph of n vertices, the adjacency matrix is a square $n \times n$ matrix A such that

$$A[i,j] = \begin{cases} 1, & \text{there is an edge from vertex } i \text{ to vertex } j \\ 0, & \text{otherwise} \end{cases}$$

The diagonal elements of the matrix are all zero, since edges from a vertex to itself (loops) are not allowed in simple graphs.

The adjacency matrix of an undirected simple graph is symmetric, and therefore has a complete set of real eigenvalues and an orthogonal eigenvector basis. The set of eigenvalues of a graph is the spectrum of the graph

incidence matrix

The incidence matrix (or unoriented incidence matrix) of $G(V, E)$ is a $|V| \times |E|$ matrix B, such that

$$B_{ij} = \begin{cases} 1, & \text{if the vertex } v_i \text{ and edge} x_j \text{ are incident} \\ 0, & \text{otherwise} \end{cases}$$

The incidence matrix of a directed graph $D(V, E)$ is a $|V| \times |E|$ matrix B, such that

$$B_{ij} = \begin{cases} 1, & \text{if the edge } x_j \text{ enters vertex } v_i \\ -1, & \text{if the edge } x_j \text{ leaves vertex } v_i \\ 0, & \text{otherwise} \end{cases}$$

Note that many authors use the opposite sign convention.

An oriented incidence matrix of an undirected graph $G(V, E)$ is the incidence matrix, in the sense of directed graphs, of any orientation of G.

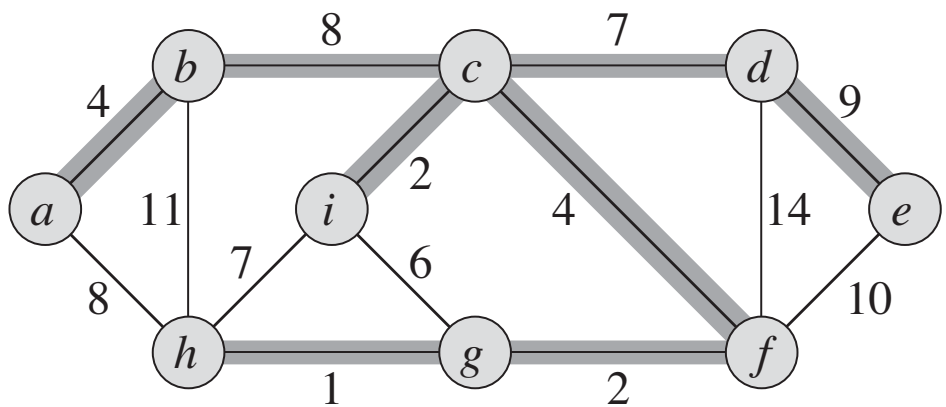
minimum spanning tree

given a connected, undirected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$

a minimum spanning tree for G, i.e.,
an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight

$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

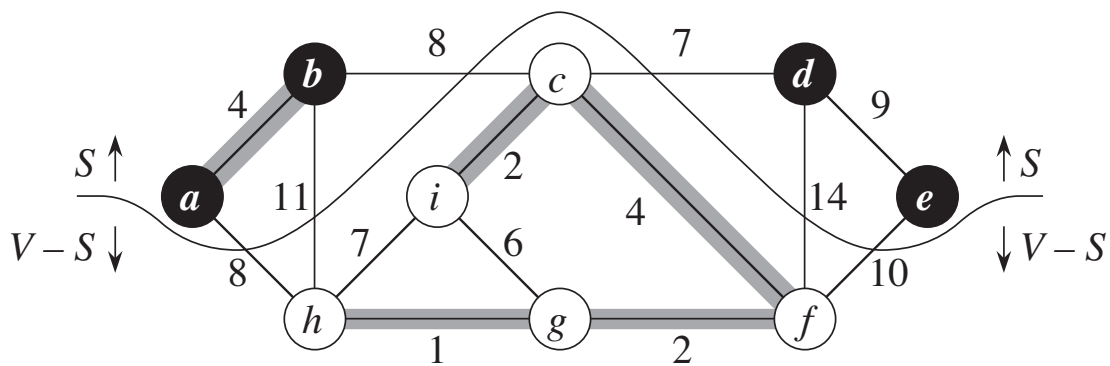
is minimized



rule for recognizing safe edges

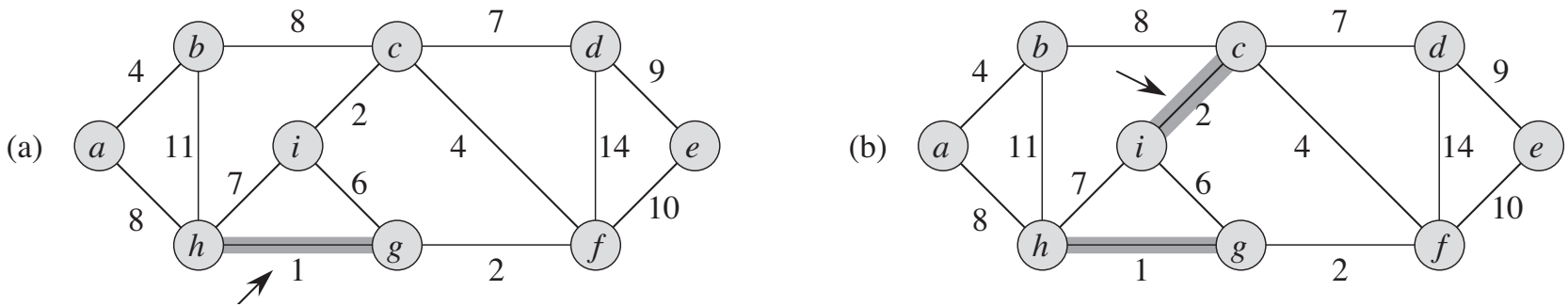
- $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E
- A be a subset of E that is included in some minimum spanning tree for G
- $(S, V \setminus S)$ be any cut of G that respects A
- (u, v) be a light edge crossing $(S, V \setminus S)$

then, edge (u, v) is safe for A.

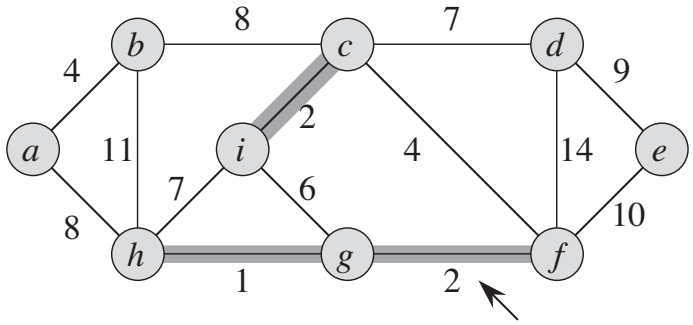


Kruskal's algorithm

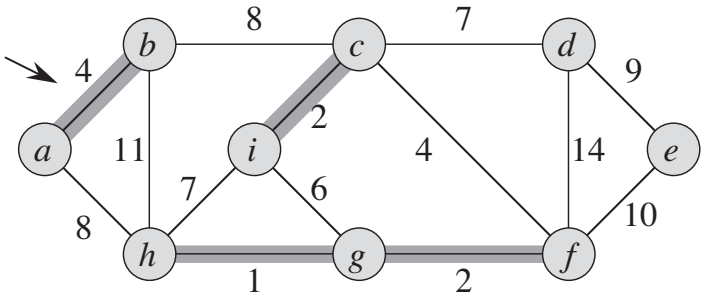
Kruskal's algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u, v) of least weight.



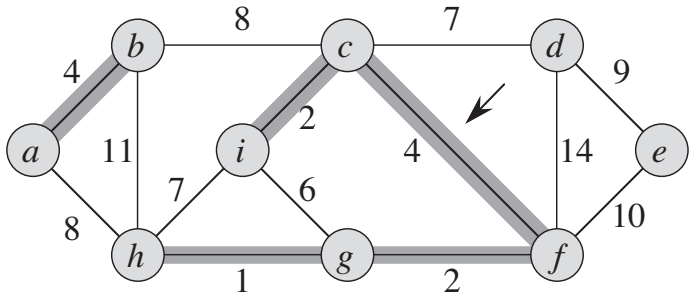
(c)


a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
1
2

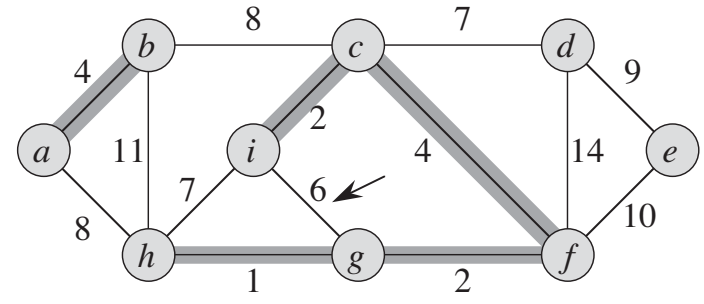
(d)


a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
1
2

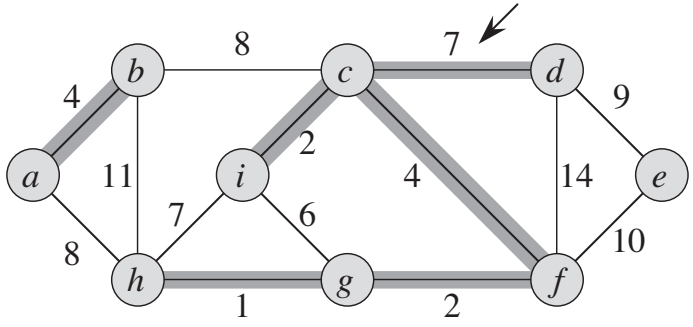
(e)


a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
1
2

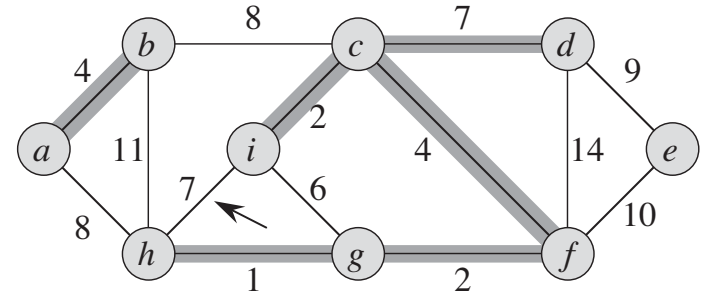
(f)


a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
1
2

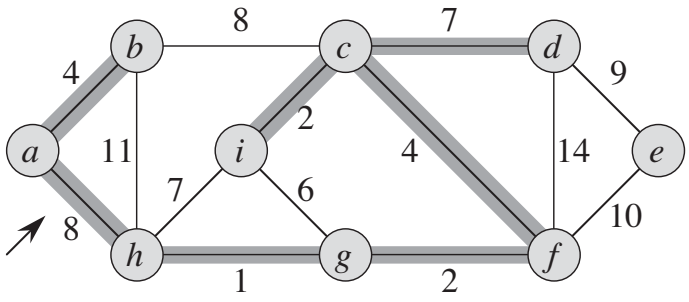
(g)


a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
1
2

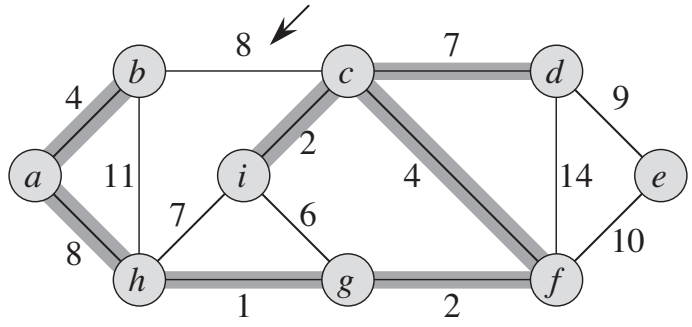
(h)


a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
1
2

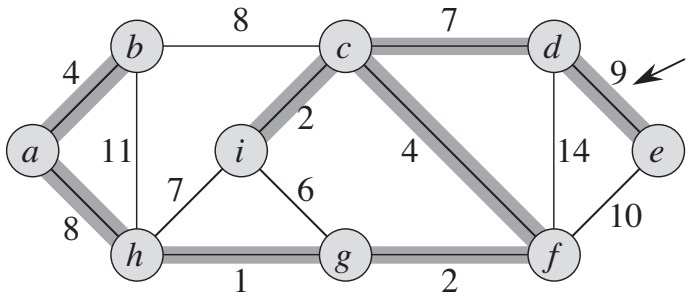
(i)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
2
4
6
7
14
9
10
2

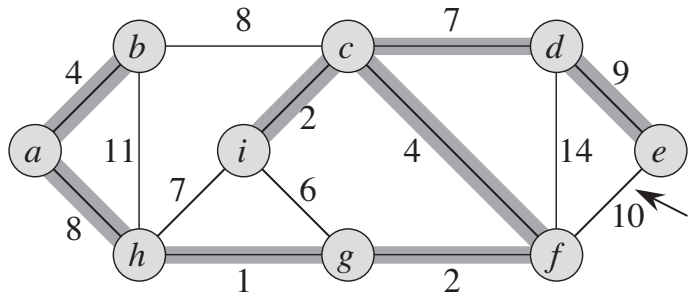
(j)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
2
4
6
7
14
9
10
2

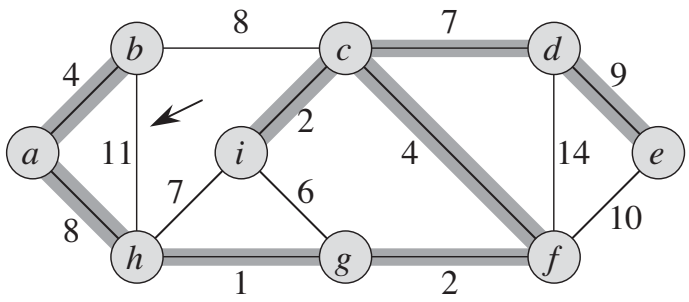
(k)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
2
4
6
7
14
9
10
2

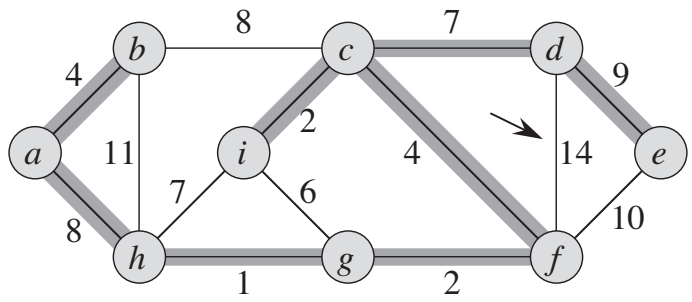
(l)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
2
4
6
7
14
9
10
2

(m)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
2
4
6
7
14
9
10
2

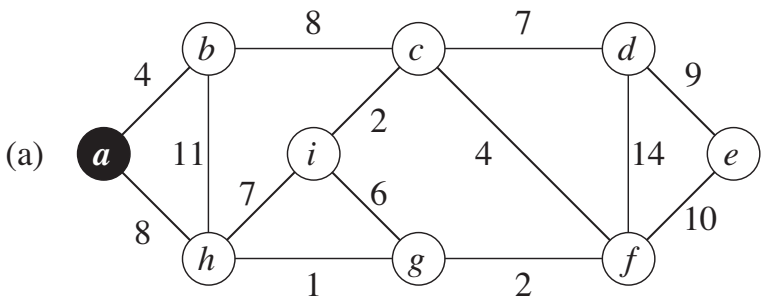
(n)

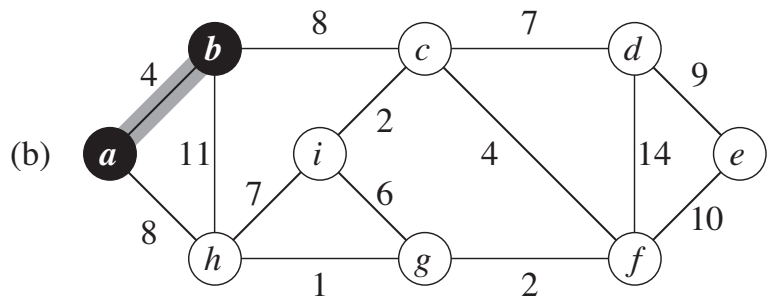

a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
2
4
6
7
14
9
10
2

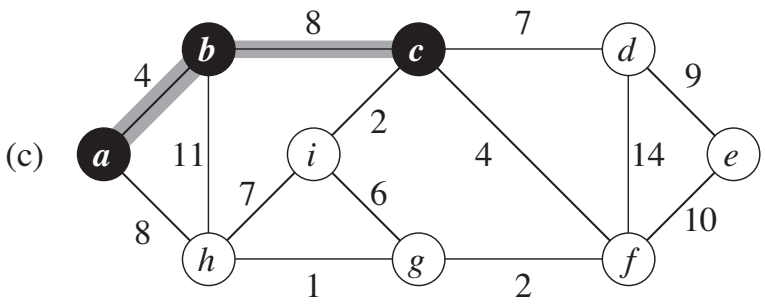
Prim's algorithm

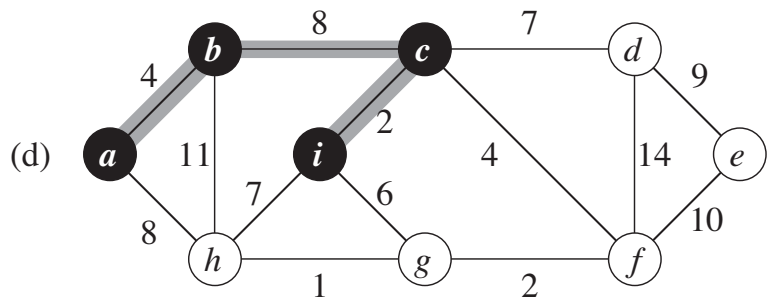
- the tree starts from an arbitrary root vertex r
- each step adds to the tree A a light edge that connects A to an isolated vertex
- grows until the tree spans all the vertices in V

Prim's algorithm has the property that the edges in the set A always form a single tree

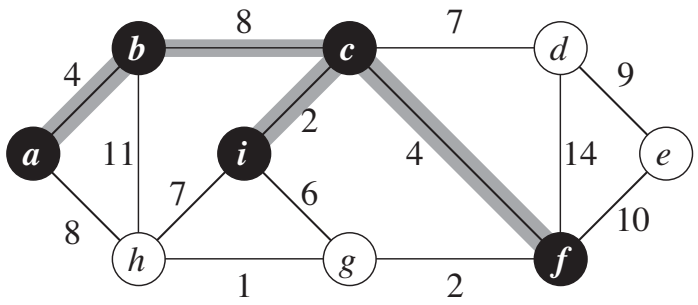




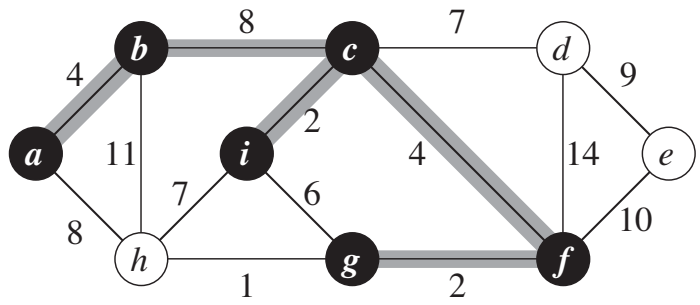




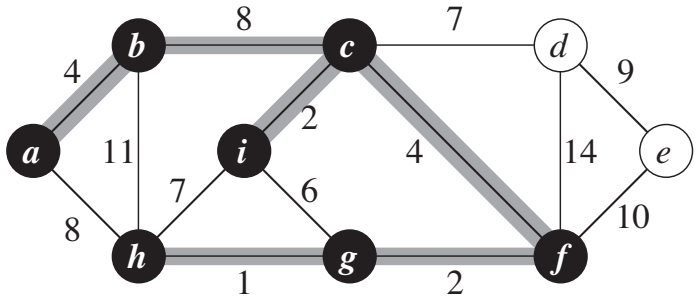
(e)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

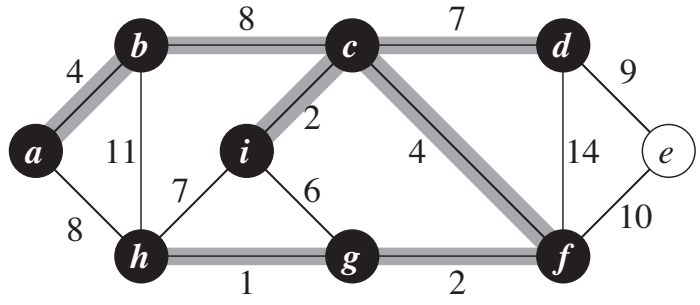
(f)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

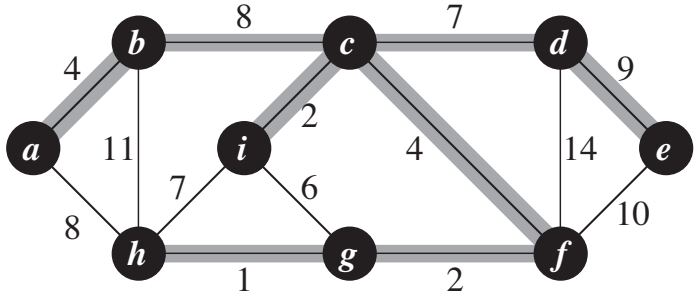
(g)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

(h)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

(i)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

shortest path problem

Let $D = (V, A)$ be a directed graph, and let $s, t \in V$.

A walk is a sequence $P = (v_0, a_1, v_1, \ldots, a_m, v_m)$ where a_i is an arc from v_{i-1} to v_i for $i = 1, \ldots, m$.

If $v_0, \ldots, v_m$ all are different, P is called a path.

- single-pair shortest path problem
- single-source shortest path problem
- single-destination shortest path problem
- all-pairs shortest path problem

The most important algorithms:

- Dijkstra's algorithm solves the single-source shortest path problem
- Bellman-Ford algorithm solves the single-source problem if edge weights may be negative
- A* search algorithm solves for single pair shortest path using heuristics to try to speed up the search
- Floyd-Warshall algorithm solves all pairs shortest paths
- Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd-Warshall on sparse graphs

linear optimization formulation

$$\begin{aligned} \min \quad & \sum_{ij \in A} w_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \\ & x \geq 0 \end{aligned}$$

this LO has the special property that it is integral, i.e.,
every basic optimal solution (when one exists) has all variables equal to 0 or 1

$$\begin{aligned} \max \quad & y_t - y_s \\ \text{s.t.} \quad & y_j - y_i \leq w_{ij} \end{aligned}$$

Dijkstra's algorithm (non-negative weights)

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y.

1. assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

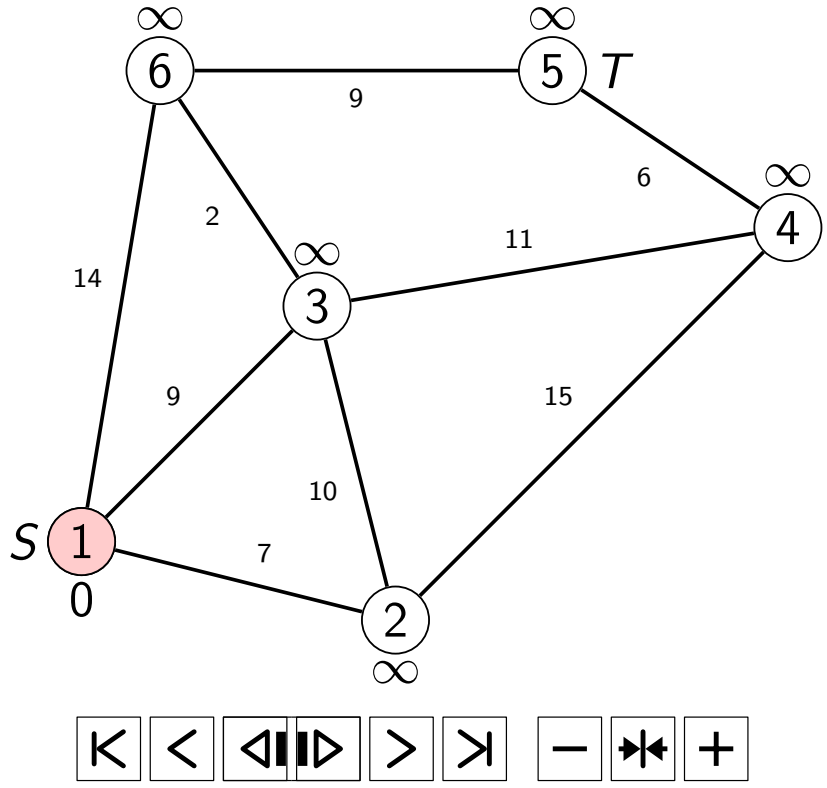
2. set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

3. for the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.

4. when we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set.

5. if the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity, then stop.

6. otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.



Dijkstra's algorithm to find the shortest path between a and b.

(1) picks the unvisited vertex with the lowest-distance

(2) calculates the distance through it to each unvisited neighbor

(3) updates the neighbor's distance if smaller

(4) mark visited (set to red) when done with neighbors

proof by induction

invariant hypothesis:

- for each visited node u, $\mathrm{dist}[u]$ is the shortest distance from source to u;
- for each unvisited v, $\mathrm{dist}[v]$ is the shortest distance via visited nodes only from source to v

proof:

- the base case is when there is just one visited node, trivial
- assume the hypothesis for $n-1$ visited nodes
  we choose an edge uv where v has the least $\mathrm{dist}[v]$ of any unvisited node and the edge uv is such that $\mathrm{dist}[v] = \mathrm{dist}[u] + \mathrm{length}[u, v]$.
- $\mathrm{dist}[v]$ must be the shortest distance because if there were a shorter path
  - if w was the first unvisited node on that path then by hypothesis $\mathrm{dist}[w] \geq \mathrm{dist}[v]$ creating a contradiction
  - if there was a shorter path to v without using unvisited nodes then $\mathrm{dist}[v]$ would have been less than $\mathrm{dist}[u] + \mathrm{length}[u, v]$
- after processing v it will still be true that for each unvisited node w, $\mathrm{dist}[w]$ is the shortest distance from source to w using visited nodes only,
  - if there were a shorter path which doesn't visit v we would have found it previously
  - if there is a shorter path using v we update it when processing v

Running time

- for dense graphs, a running time bound of $O(|V|^2)$ for a shortest path algorithm is best possible, since one must inspect each arc
- the implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O\big(|E| + |V| \log |V|\big)$ (where $|E|$ is the number of edges) is due to Fredman and Tarjan 1984
- this is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights
- specialized cases (such as bounded/integer weights, directed acyclic graphs etc) can indeed be improved further

shortest paths with arbitrary lengths

- if lengths of arcs may take negative values, it is not always the case that a shortest walk exists.
- if the graph has a directed circuit of negative length, then we can obtain s-t walks of arbitrary small negative length (for appropriate s and t).
- it can be shown that if there are no directed circuits of negative length, then for each choice of s and t there exists a shortest s-t walk (if there exists at least one s-t path).
- let each directed circuit have nonnegative length. Then for each pair s, t of vertices for which there exists at least one s-t walk, there exists a shortest s-t walk, which is a path.

Bellman-Ford method (arbitrary lengths)

Let $n := |V|$.

the algorithm calculates functions $f_0, f_1, f_2, \ldots, f_n : V \to \mathbb{R} \cup \{\infty\}$
successively by the following rule:

(i) put $f_0(s) := 0$ and $f_0(v) := \infty$ for all $v \in V \setminus \{s\}$

(ii) for $k < n$, if f_k has been found, put
$f_{k+1}(v) := \min\{f_k(v), min_{(u,v)\in A}(f_k(u) + \text{length}(u, v))\}$ for all $v \in V$

assuming that there is no directed circuit of negative length, $f_n(v)$ is equal to the length of a shortest s-v walk, for each $v \in V$ (if there is no s-v path at all, $f_n(v) = \infty$)

For each $k = 0, \ldots, n$ and for each $v \in V$,

$$f_k(v) = \min\{l(P) \mid P \text{ is an } s - v \text{ walk traversing at most } k \text{ arcs}\}$$

maximum flow

a **flow network** $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative **capacity** $c(u, v) \geq 0$

we further require that

- if E contains an edge (u, v), then there is no edge (v, u)
- if $(u, v) \notin E$, we define $c(u, v) = 0$, and we disallow self-loops
- distinguish two vertices: a source s and a sink t

a **flow** in G is a real-valued function $f : V \times V \to \mathbb{R}$ satisfying

capacity constraint: for all $u, v \in V$, $0 \leq f(u, v) \leq c(u, v)$

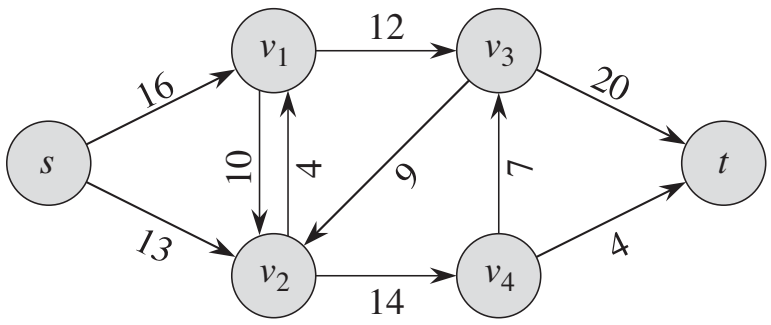
flow conservation: for all $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$

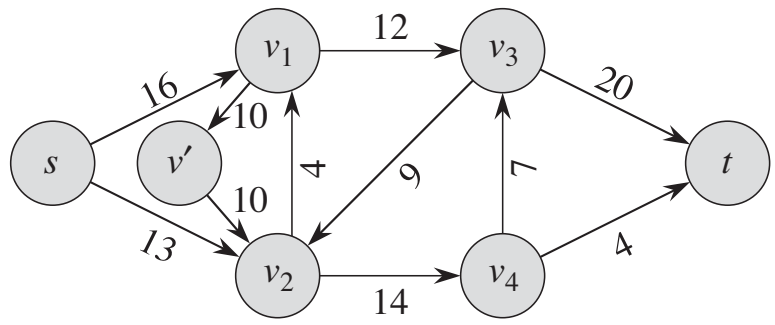
the **value** of a flow f: $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$

the $|\cdot|$ notation denotes flow value, not absolute value, nor cardinality

maximum-flow problem: given a flow network G with source s and sink t, find a flow of maximum value

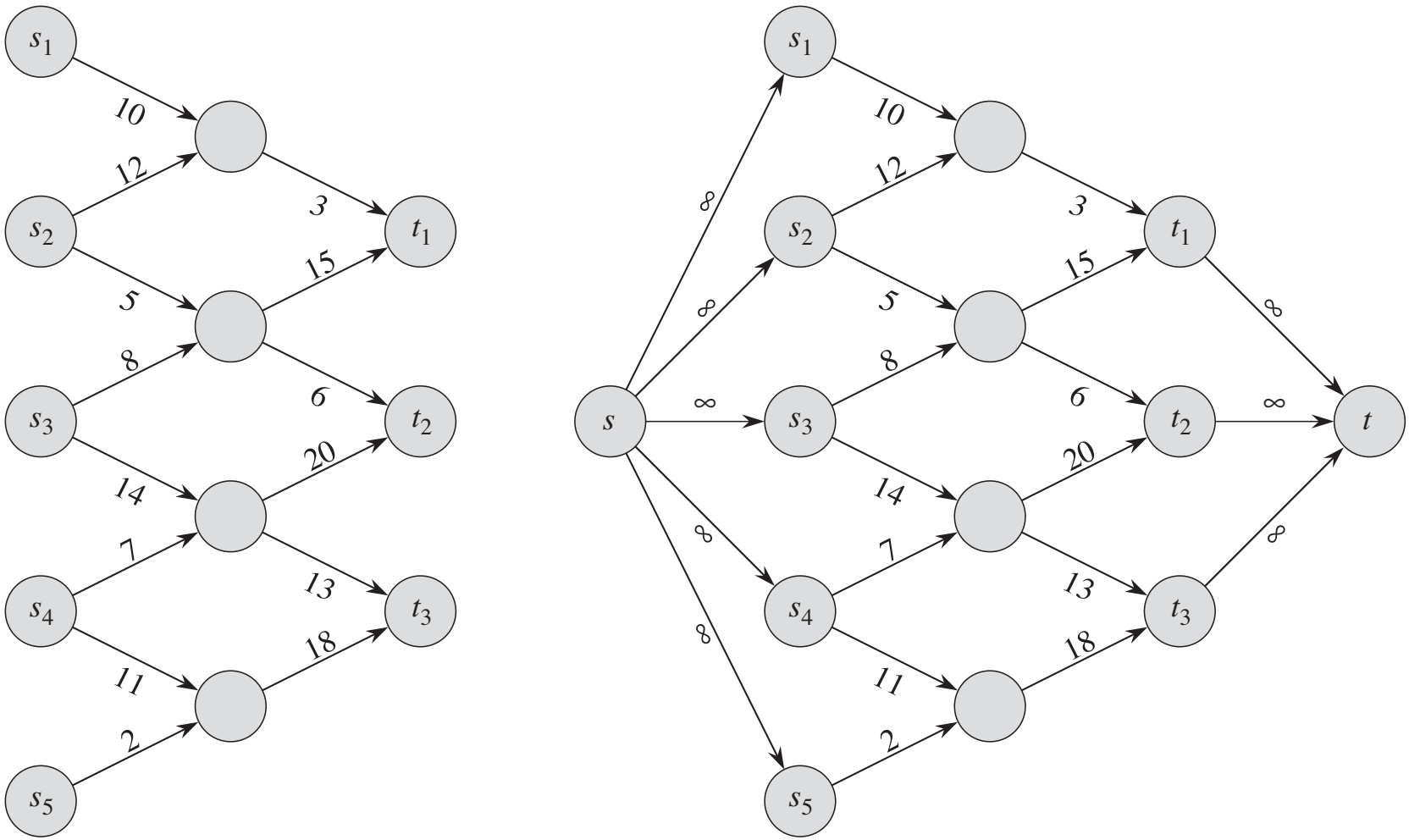
modeling problems with antiparallel edges





networks with multiple sources and sinks

add a **supersource** and a **supersink**



Ford-Fulkerson method

The Ford-Fulkerson method depends on three important ideas

- residual networks
- augmenting paths
- cuts

Ford-Fulkerson method

1. initialize flow f to 0
2. **while** there exists an augmenting path p in the residual network G_f
   augment flow f along p

   return f

residual networks

the residual network G_f consists of edges with capacities that represent how we can change the flow on edges of G

- a flow network $G = (V, E)$ with source s and sink t
- f be a flow in G

residual capacity $c_f(u, v)$:

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

because of antiparallel, $(u, v) \in E$ implies $(v, u) \notin E$, exactly one case applies to each ordered pair

residual network of G induced by f is $G_f = (V, E_f)$, where

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

The edges in E_f are either edges in E or their reversals, $|E_f| \leq 2|E|$

augmentation of flow

- f is a flow in G
- f' is a flow in the corresponding residual network G_f

the **augmentation** of flow f by f'

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

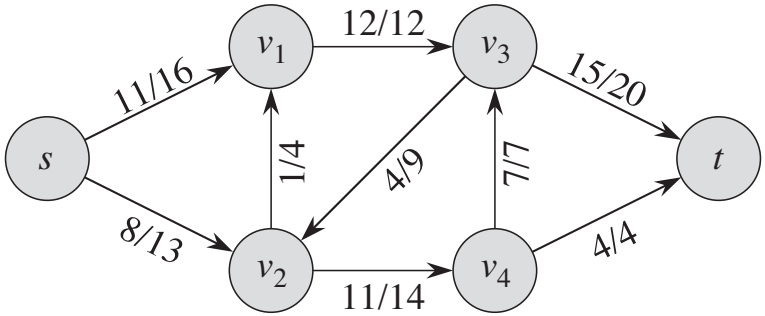
increase the flow on $f(u,v)$ by $f'(u,v)$ but decrease it by $f'(v,u)$

pushing flow on the reverse edge in the residual network is also known as **cancellation**

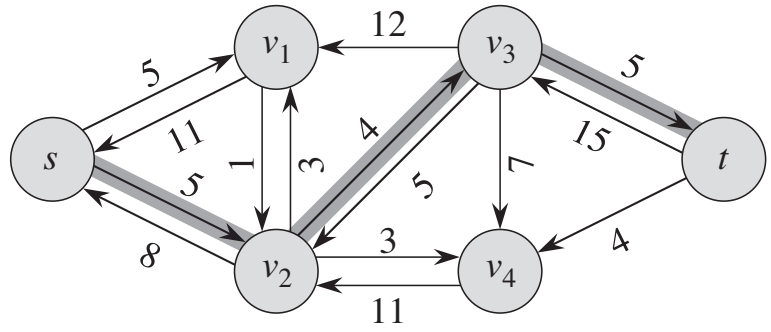
an **augmenting path** p is a simple path from s to t in the residual network G_f

the **residual capacity** of p:

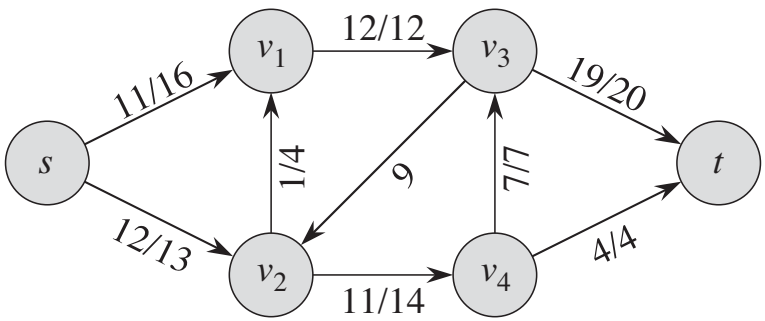
$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is on } p\}$$


v_1
v_3
s
t
v_2
v_4
12/12
11/16
15/20
1/4
4/9
7/7
8/13
4/4
11/14

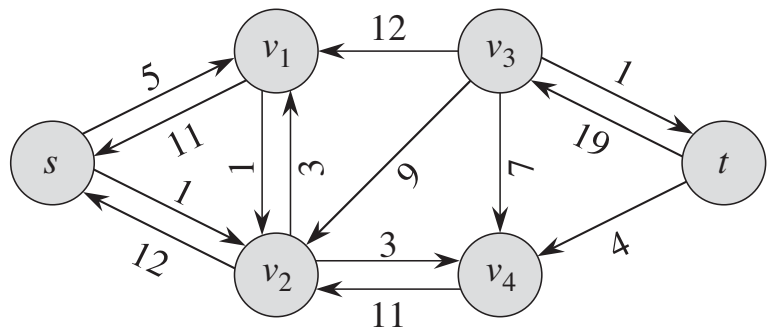
(a)


v_1
v_3
s
t
v_2
v_4
12
5
5
11
4
15
1
3
5
7
5
8
3
4
11

(b)


v_1
v_3
s
t
v_2
v_4
12/12
11/16
19/20
1/4
9
7/7
12/13
4/4
11/14

(c)


v_1
v_3
s
t
v_2
v_4
12
5
1
11
19
1
1
3
9
7
12
3
4
11

(d)

cuts of flow networks

a **cut** (S, T) of flow network $G = (V, E)$ is a partition of V into S and $T = V \setminus S$, such that $s \in S$ and $t \in T$

the **net flow** $f(S, T)$ across the cut (S, T)

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

the **capacity** of the cut (S, T) is

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

a **minimum cut** of a network is a cut whose capacity is minimum over all cuts

max-flow min-cut theorem

If f is a flow in a flow network $G(V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G_f contains no augmenting paths
3. $|f| = c(S, T)$ for some cut (S, T) of G

linear program formulation

$$
\begin{aligned}
\max \quad & |f| \\
\text{s.t.} \quad & \sum_v f(s,v) = |f| \\
& \sum_v f(u,v) - \sum_v f(v,u) = 0, \quad \forall u \notin \{s,t\} \\
& -\sum_v f(v,t) = -|f| \\
& 0 \le f(u,v) \le c(u,v), \quad \forall (u,v) \in A
\end{aligned}
$$

$$
\begin{aligned}
\min \quad & c^T d \\
\text{s.t.} \quad & y_v - y_u \le d(u,v), \quad \forall (u,v) \in A \\
& y_t - y_s = 1 \\
& d \ge 0
\end{aligned}
$$

every (S,T) cut is feasible:

$d(u,v) = 1$, $\forall u \in S, v \in T$, otherwise $d(u,v) = 0$

$y_u = 0$ if $u \in S$ and $y_v = 1$ if $v \in T$

the basic Ford-Fulkerson algorithm

for each edge $(u, v) \in G.E$

$(u, v).f = 0$

while there exists a path p from s to t in the residual network G_f

$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is in } p\}$

for each edge (u, v) in p

if $(u, v) \in E$

$(u, v).f = (u, v).f + c_f(p)$

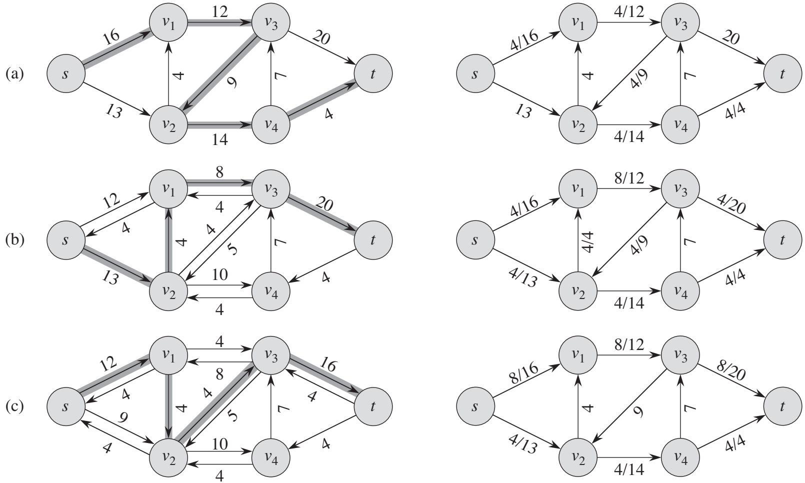
else

$(v, u).f = (v, u).f - c_f(p)$

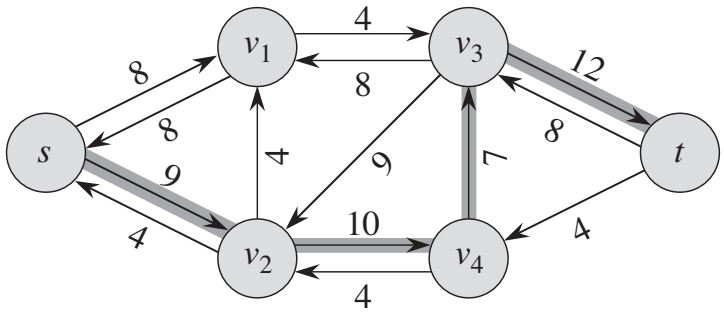
the running time of Ford-Fulkerson depends on how we find the augmenting path p

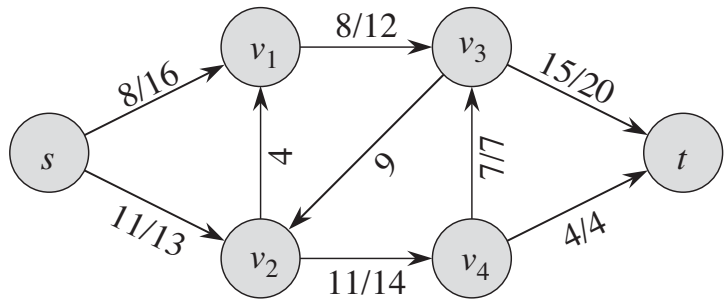
choose the augmenting path arbitrarily and all capacities are integers: $O(E|f^*|)$

example

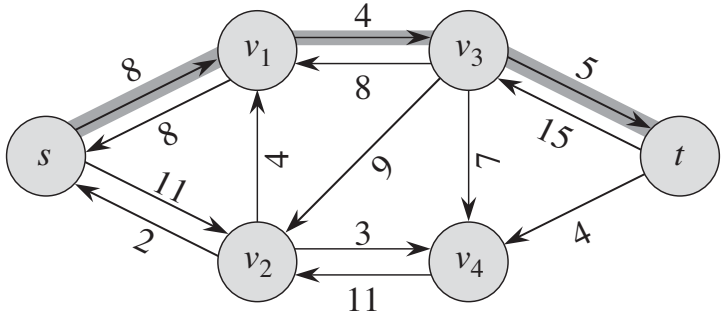

(a)
s
v1
v2
v3
v4
t
16
12
20
13
4
9
7
4
14
4/16
4/12
4/9
4/14
4/4
(b)
8
4/13
4/20
4/4
10
5
8/12
(c)
16
8/16
8/20

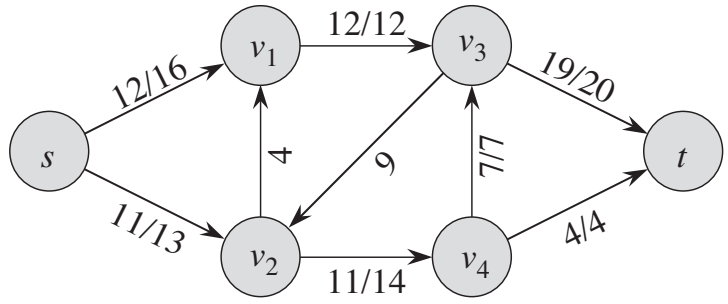
(d)


v_1
v_3
s
t
v_2
v_4
4
8
8
12
8
9
4
9
7
8
10
4
4
4

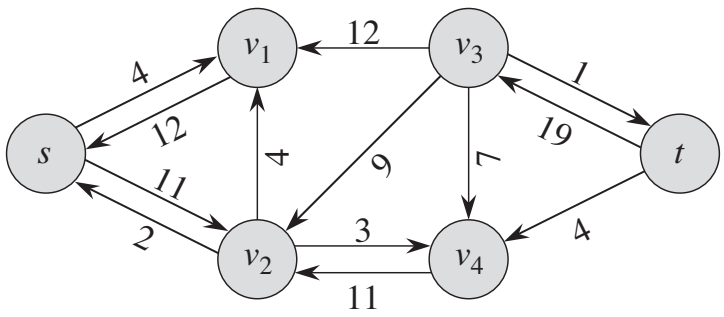

v_1
v_3
s
t
v_2
v_4
8/12
8/16
15/20
4
9
7/7
11/13
4/4
11/14

(e)


v_1
v_3
s
t
v_2
v_4
4
8
8
5
8
15
4
9
7
11
2
3
4
11


v_1
v_3
s
t
v_2
v_4
12/12
12/16
19/20
4
9
7/7
11/13
4/4
11/14

(f)


v_1
v_3
s
t
v_2
v_4
12
4
1
12
19
4
9
7
11
2
3
4
11