Contents

1. Lecture 1

1.1 Running time

Definition If there is a constant C and a number k such that

$$\forall n \geq k, \quad f(n) \leq C \cdot g(n)$$

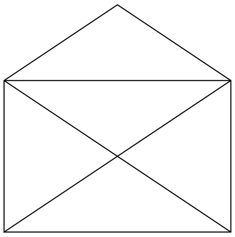
We say that a function f is of order of a function g and write:

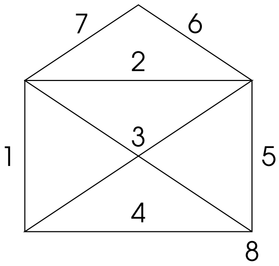
$$f(n) = O(g(n))$$

It is common to say this problem can be solved in O(f(n)).

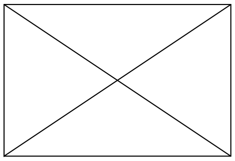
1.2 Euler path

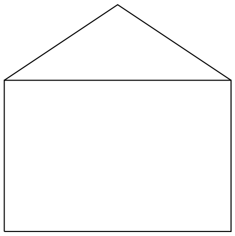
Example of decision problem: Is it possible to draw this figure without going over the same line?



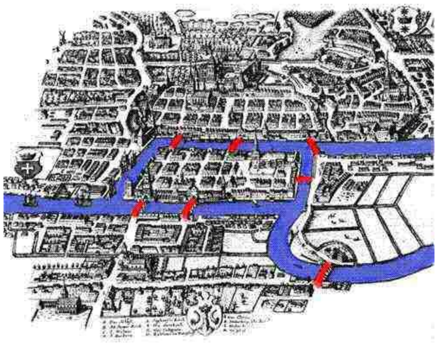


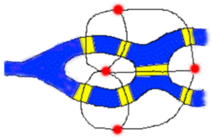
What if you get the following cases?

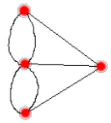




Is it possible to have a walk around the city, crossing each of the seven bridges only once ?







Definition **Eulerian path**: a path in a finite graph that visits every edge exactly once.

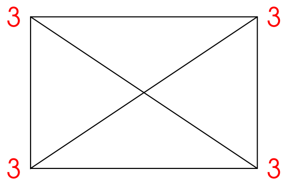
By transforming our problem into a graph, our question amounts to asking whether there is an Eulerian path.

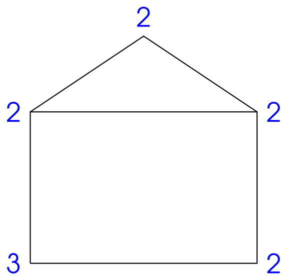
Theorem

A connected graph has an Eulerian path $\Longleftrightarrow$ the number of nodes with odd degree is 0 or 2.

May the number be 0 then the path is a cycle: it starts and finishes on the same node.

The theorem gives a certificate for the existence of an Eulerian path but doesn't provide a effective solution.



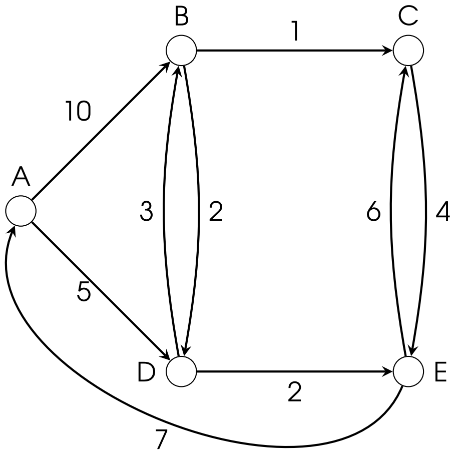


We can also conclude that the people of the city will not be able to cross the city without going over one of the bridges.

We can add-delete bridges to make it Eulerian.This idea leads to the chinese postman problem: What is the minimum number of edges we need to add to make the graph Eulerian? (Efficient algorithm by Jack Edmonds).

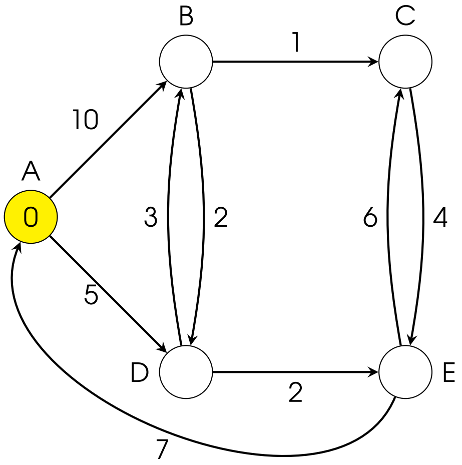
1.3 Dijkstra's algorithm

What is the shortest path from A to C ?



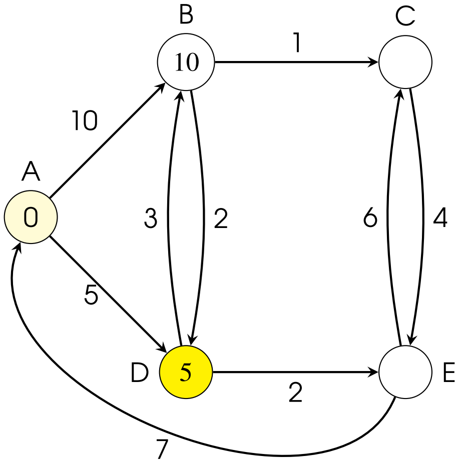
Start by setting the distance from A to A to zero in the node.

The color yellow indicates the final answer.



For the node that was just made yellow, check what you can reach by using its outgoing arcs.

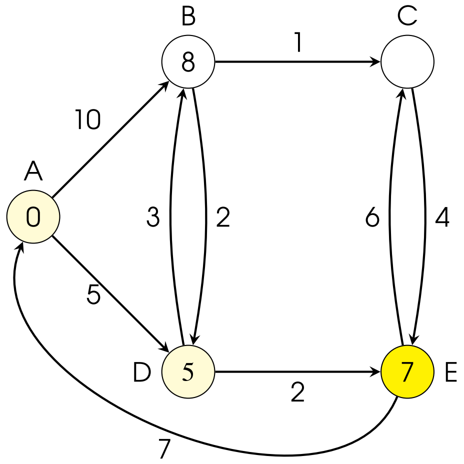
Finalize the smallest number that is not yellow yet



Check everything you can reach by using 1 arc from the node that was last made yellow.

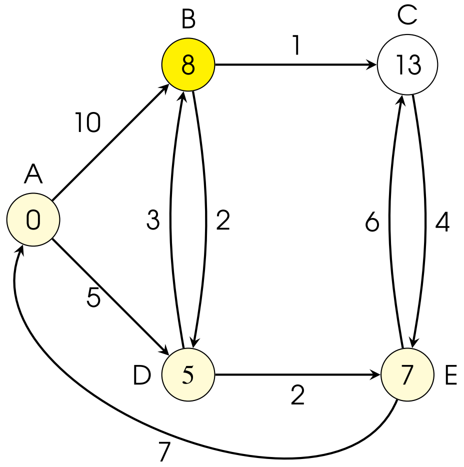
Update numbers if you found a shorter path.

Finalize the smallest number that is not yet yellow.



Ignore arcs that point back to something that was already finalized

Finalize the smallest number that is not yet yellow

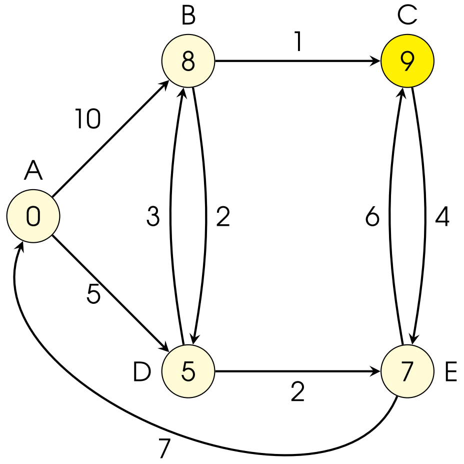


Ignore arcs that point back to something that was already finalized

Finalize the smallest number that is not yet yellow

When the destination node is yellow, we are done.

Note: in this case the destination was the last node to become yellow, but this is not always the case.

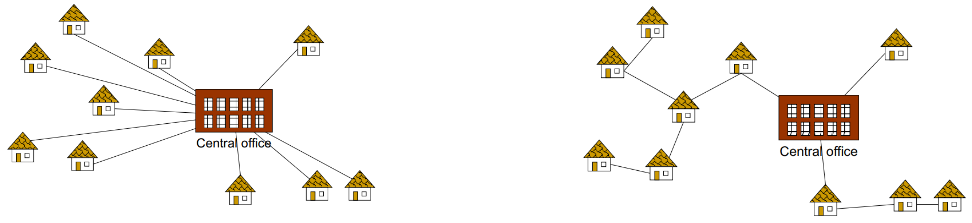


Does Dijkstra's algorithm give the optimal solution in any graph?

1.4 Greedy algorithms

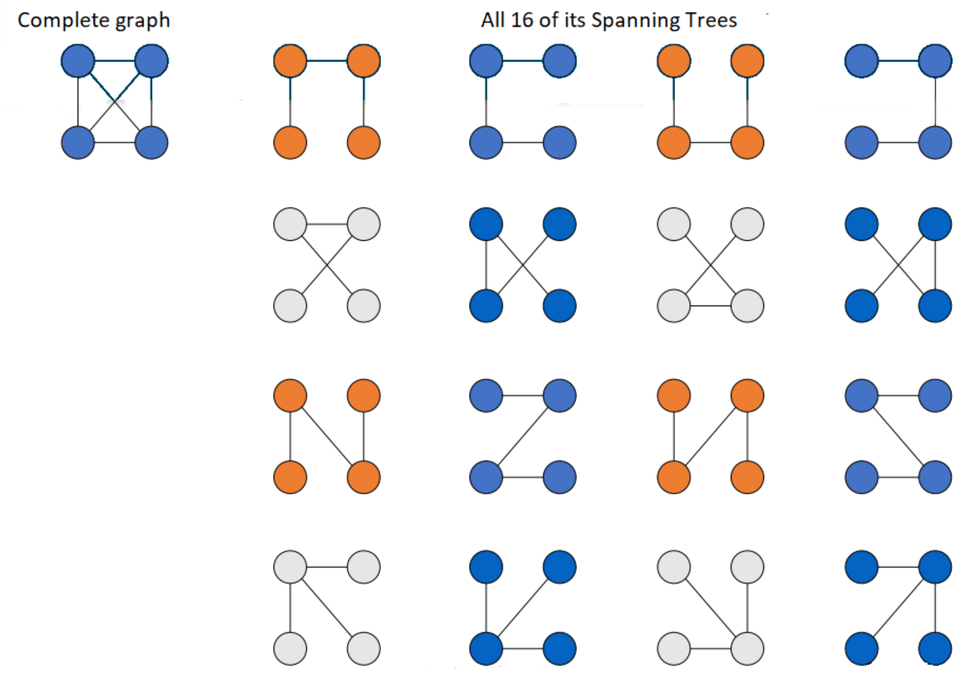
A greedy algorithm extends its wealth (the solution being constructed) by taking the step with highest immediate gain.Greedy algorithms are often sub-optimal but they can be optimal.We will see that through Minimal Spanning Tree problem.

Problem: laying cables

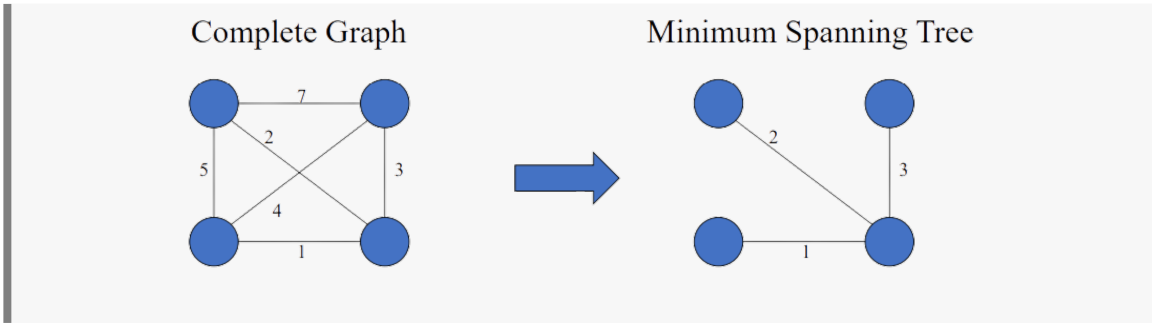


> **Definition**
> A **tree** is a connected graph without cycles.
> A **spanning tree** of a graph is a tree that contains all the vertices of the graph and a subset of its edges.

A graph may have many spanning trees:



> **Definition** The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph.



We can imagine several versions of such a greedy algorithms and they may have different efficiencies.

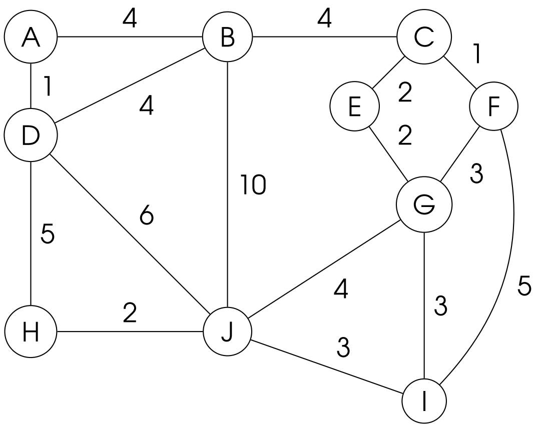
1.5 Kruskal's Algorithm

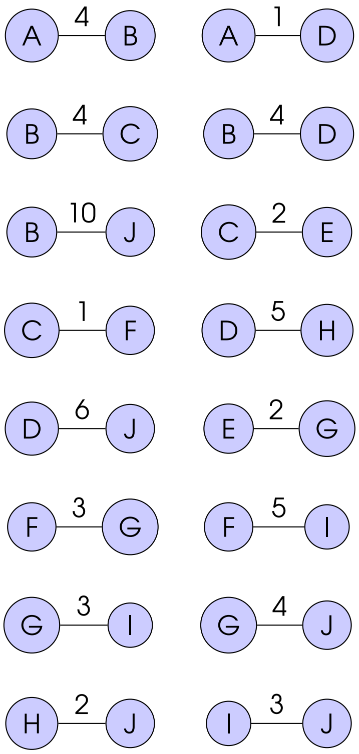
This algorithm creates a forest of trees. Initially the forest consists of n single node trees (and no edges). At each step, we add one edge (the cheapest one) so that it joins two trees together. If it were to form a cycle, skip that edge (it would not be needed).

The steps are:

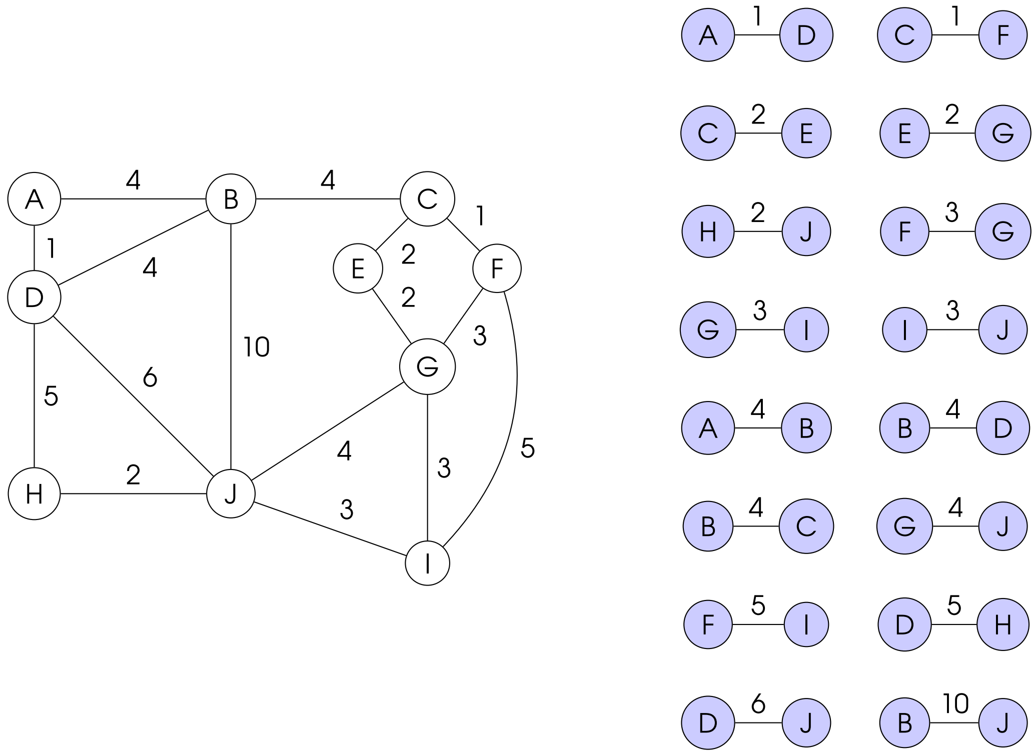
- The forest is constructed with each node in a separate tree.
- The edges are placed in a priority queue.
- Until we've added $n-1$ edges,
  1. Extract the cheapest edge from the queue
  2. If it forms a cycle, reject it
  3. Else add it to the forest. Adding it to the forest will join two trees together.

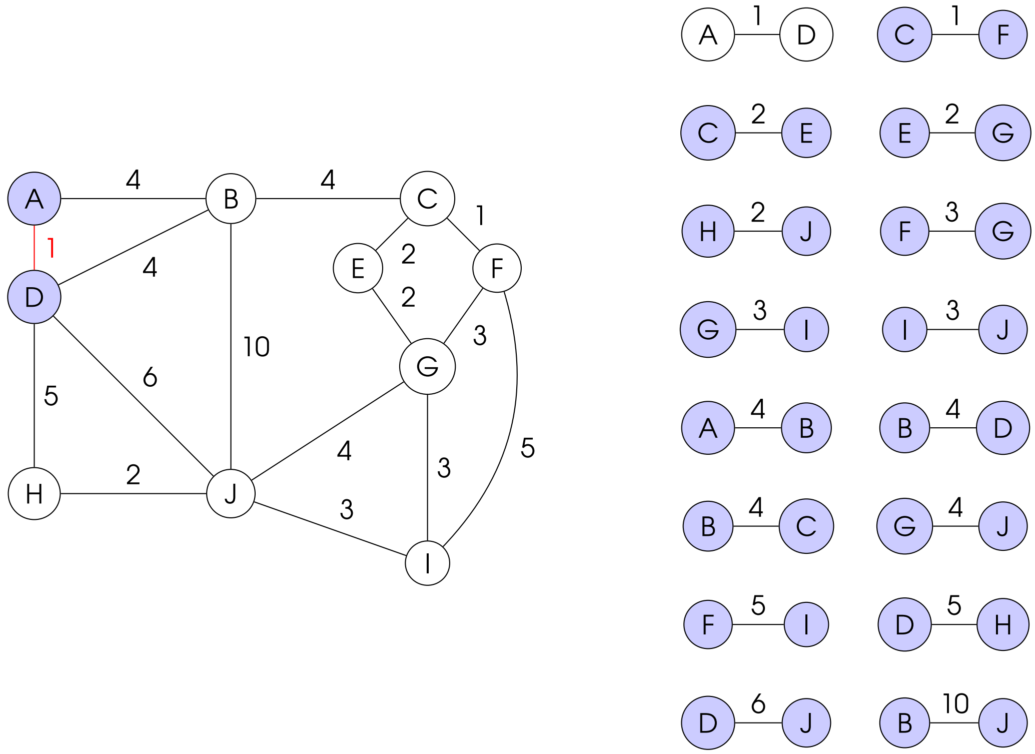
Every step will have joined two trees in the forest together, so that at the end, there will only be one tree.

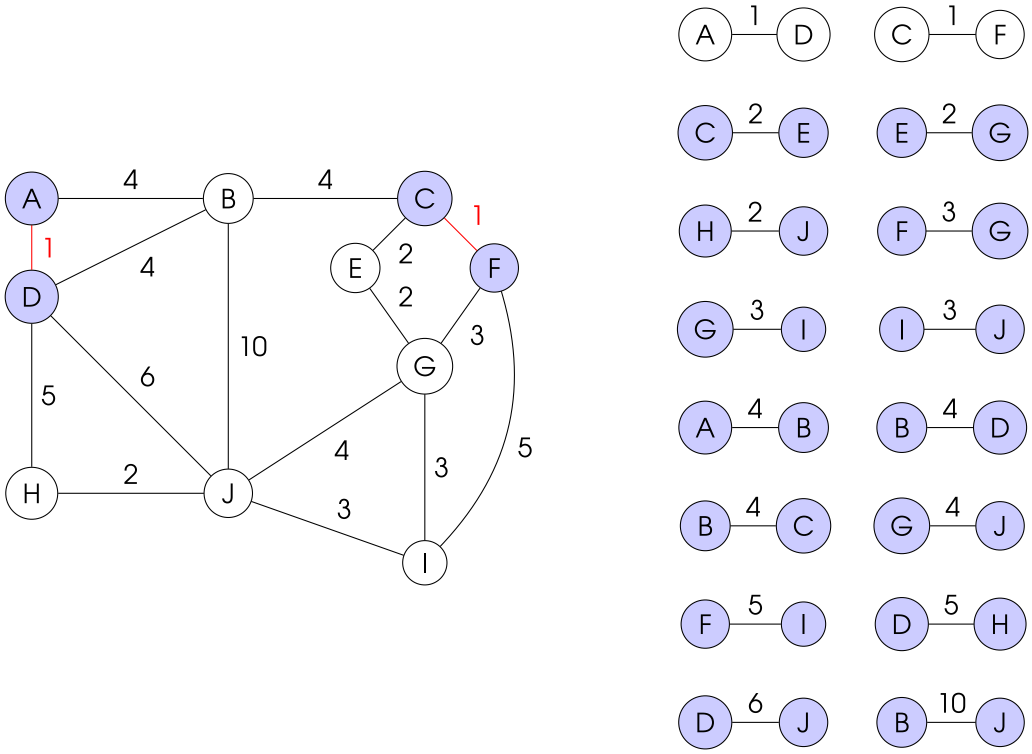


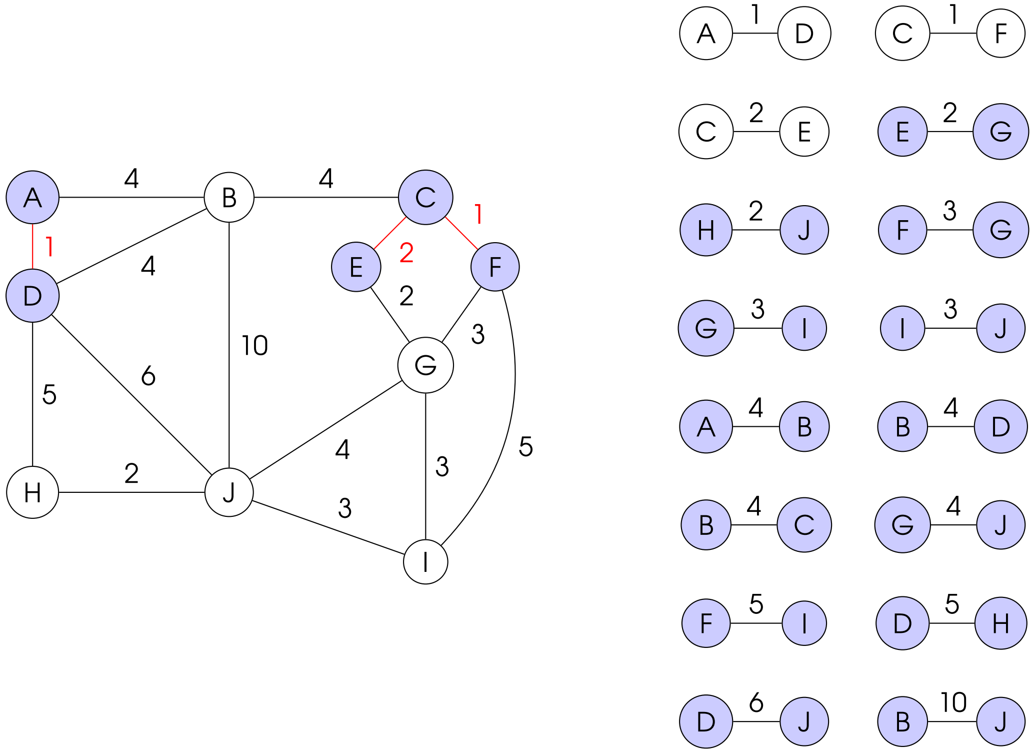


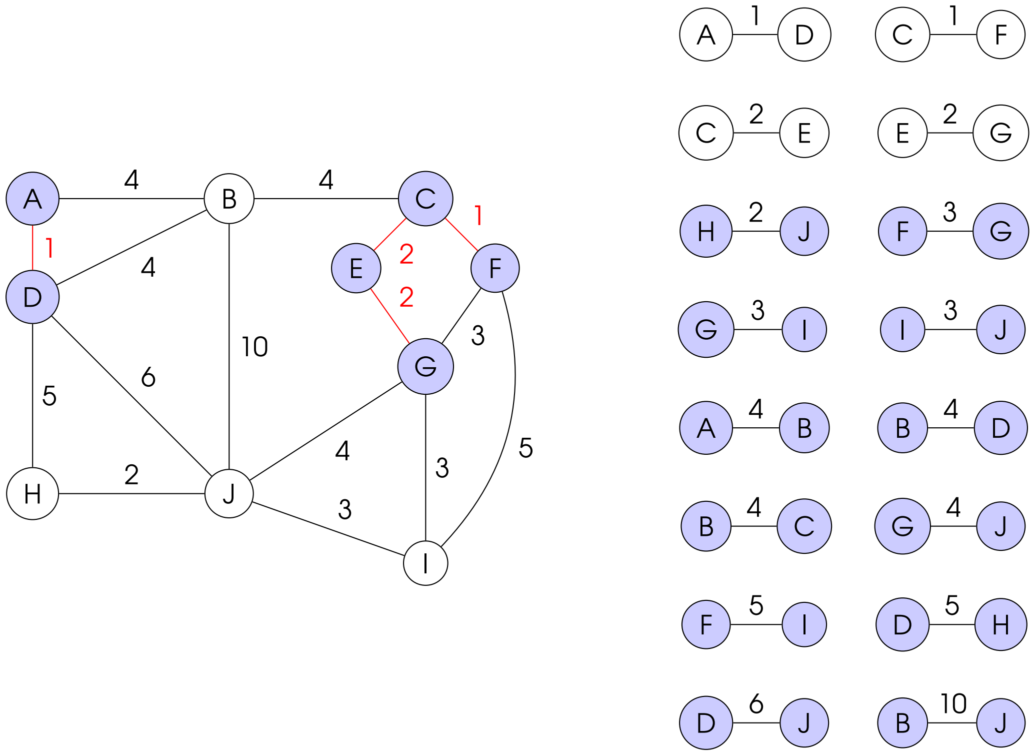
The edges are placed in a priority queue: sorting them makes the algorithm easier to visualize.

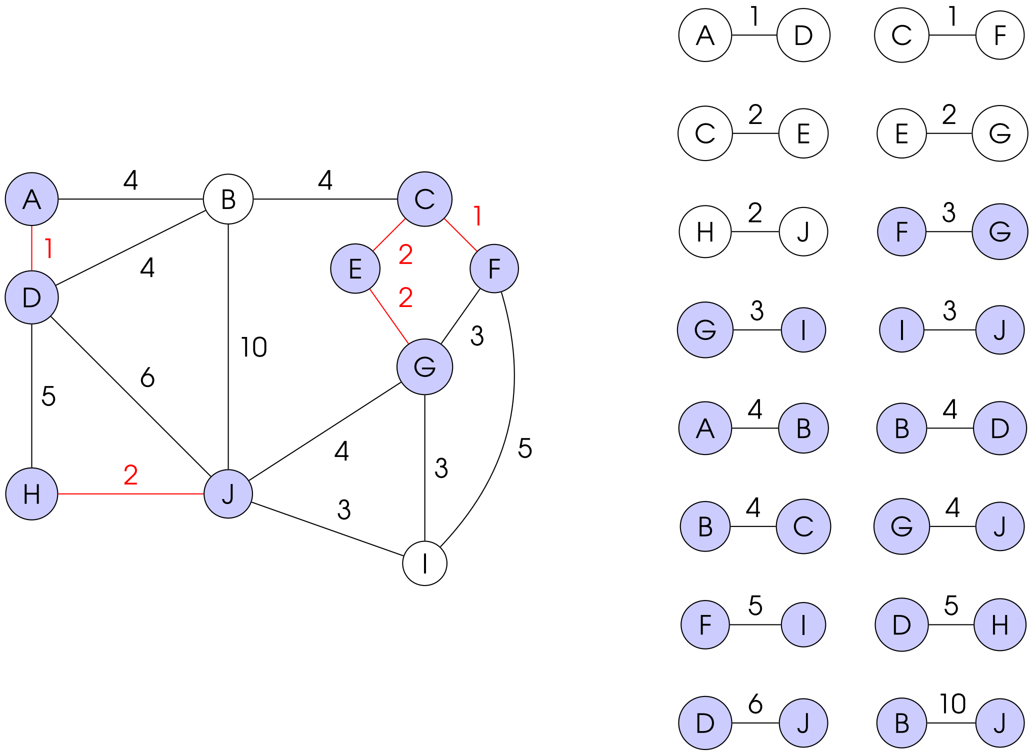


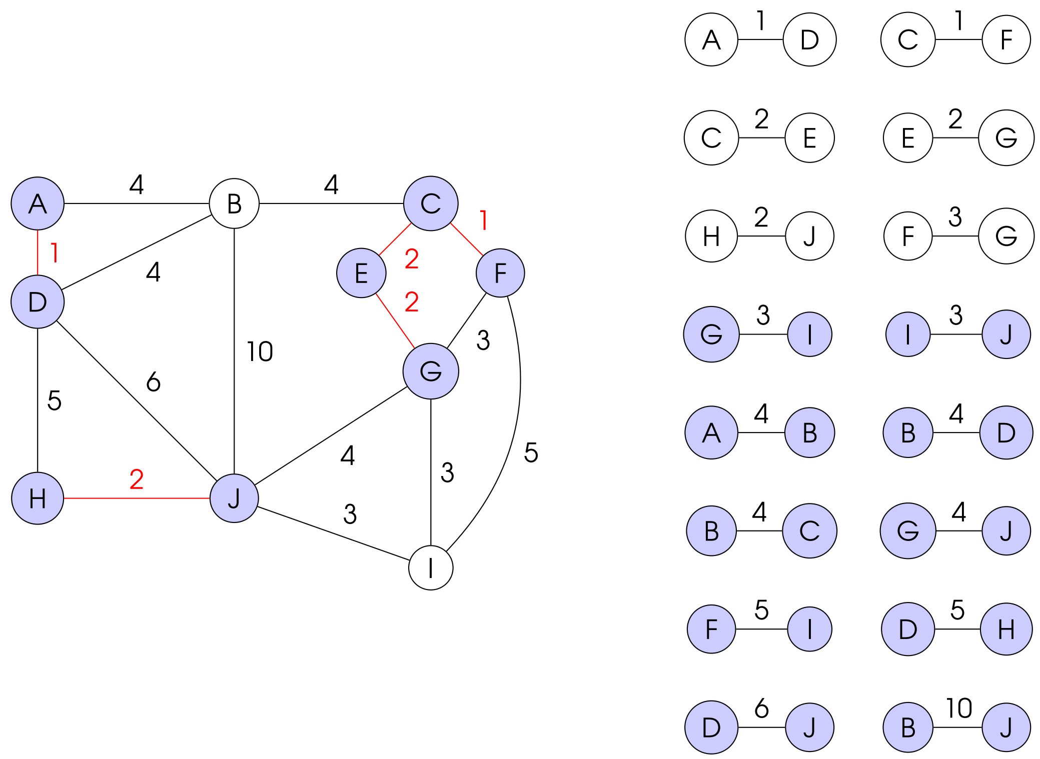


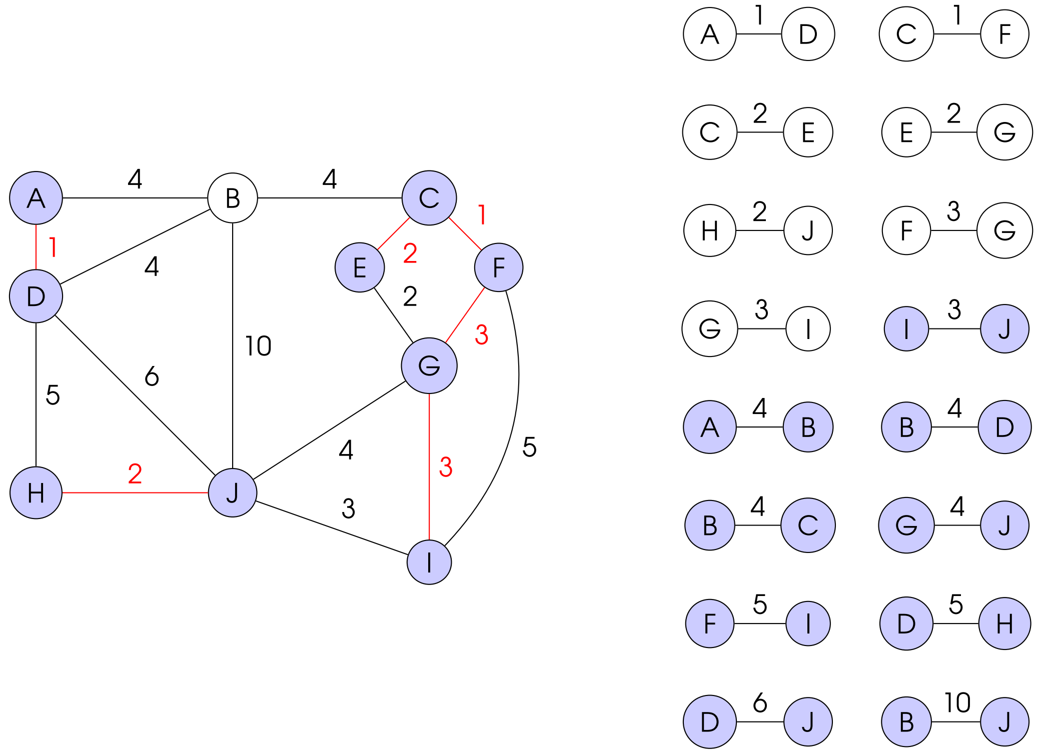

A
B
C
D
E
F
G
H
I
J
4
4
1
1
4
2
2
10
3
6
5
4
3
5
2
3
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

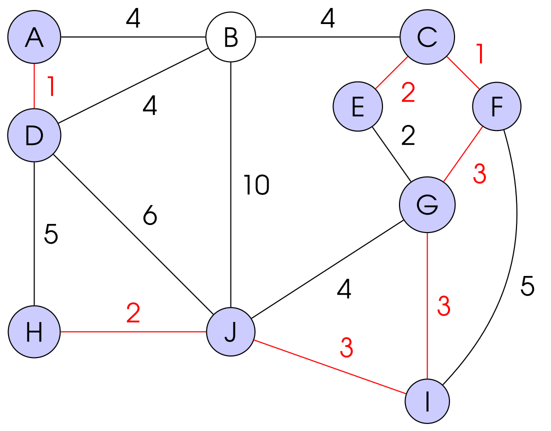

A
B
C
D
E
F
G
H
I
J
4
4
1
1
4
2
2
10
3
6
5
4
3
5
2
3
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

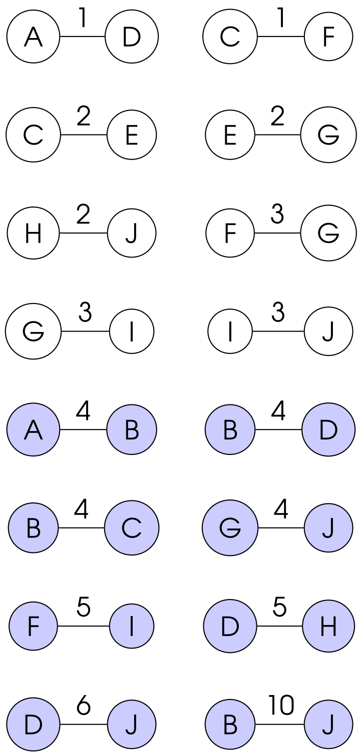

A
B
C
D
E
F
G
H
I
J
4
4
1
1
4
2
2
3
10
6
5
4
3
5
2
3
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

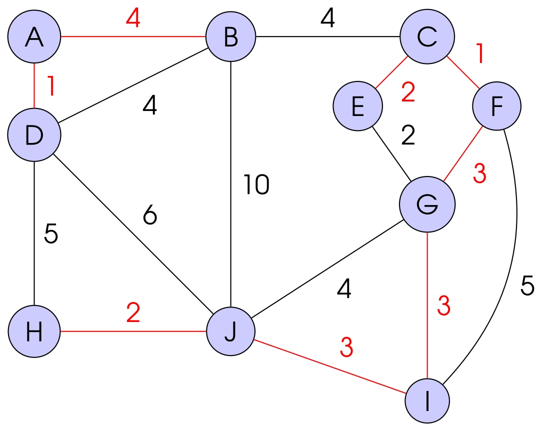

A
B
C
D
E
F
G
H
I
J
4
4
1
1
4
2
2
3
10
6
5
4
3
5
2
3
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

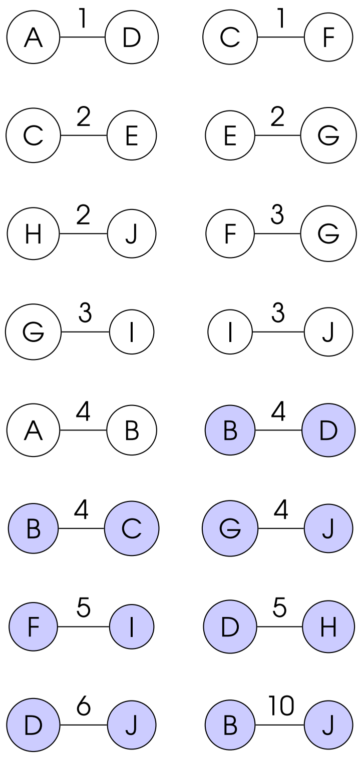

A
B
C
D
E
F
G
H
J
I
4
4
1
4
2
2
1
10
3
5
6
5
4
3
2
3
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

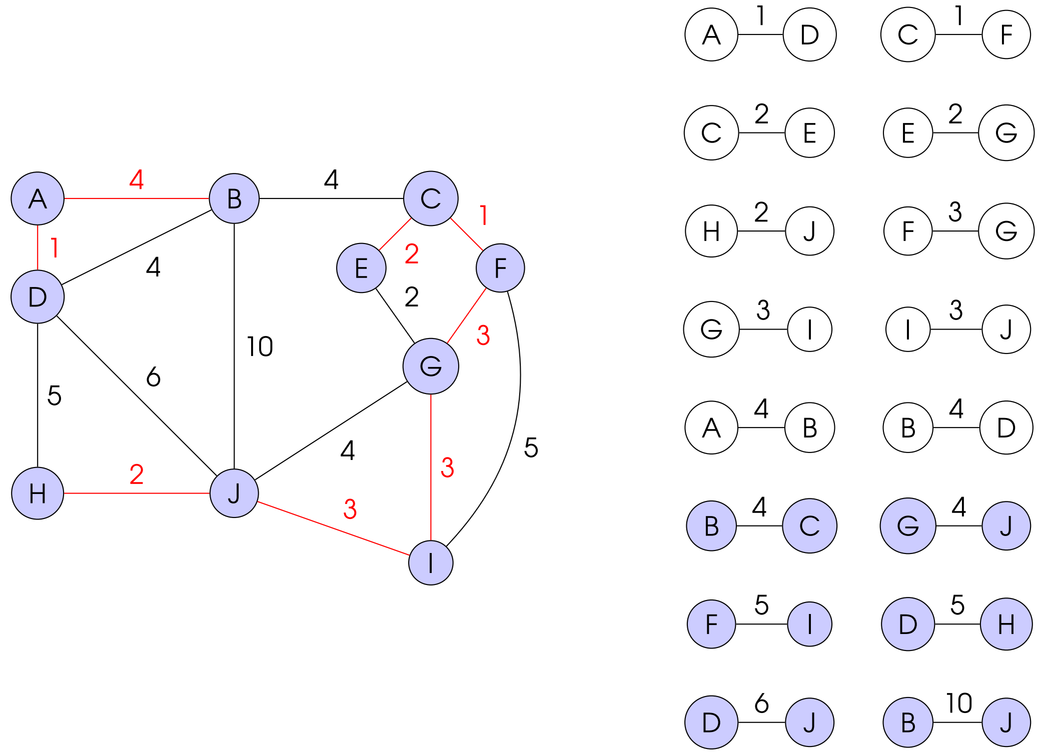

A
B
C
D
E
F
G
H
J
I
4
4
1
4
2
2
1
10
3
5
6
5
4
3
2
3
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

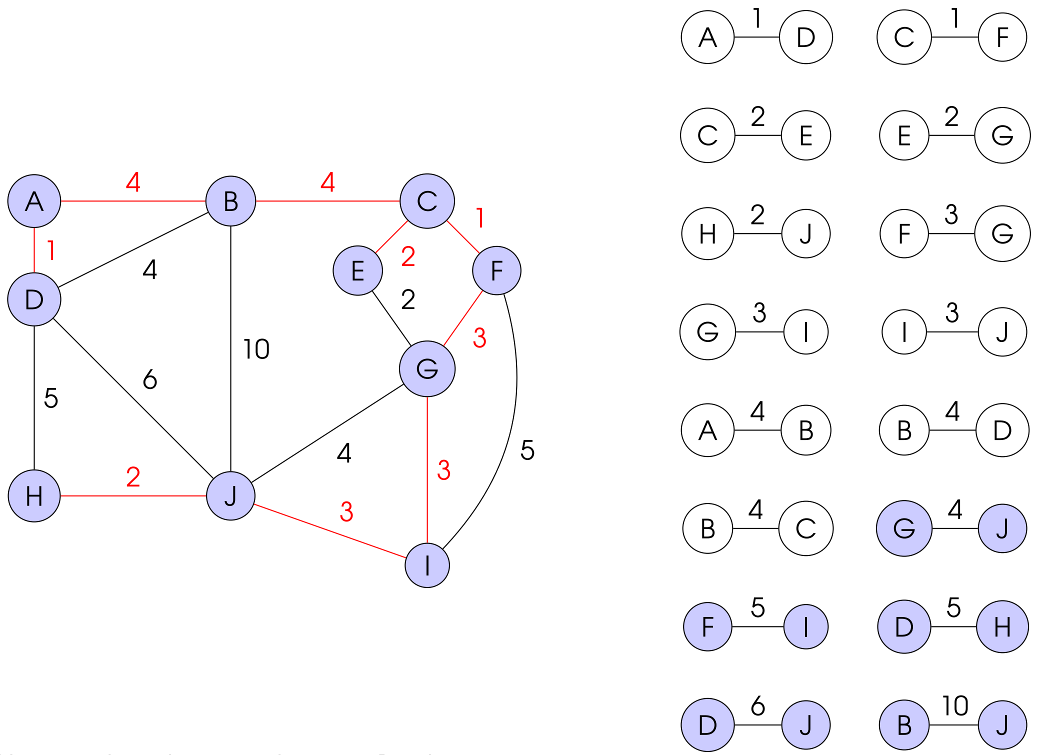

A
B
C
D
E
F
G
H
J
I
4
4
1
4
2
1
2
10
6
3
5
4
3
5
2
3


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J


A
B
C
D
E
F
G
H
J
I
4
4
1
4
2
1
2
10
6
3
5
4
3
5
2
3


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J





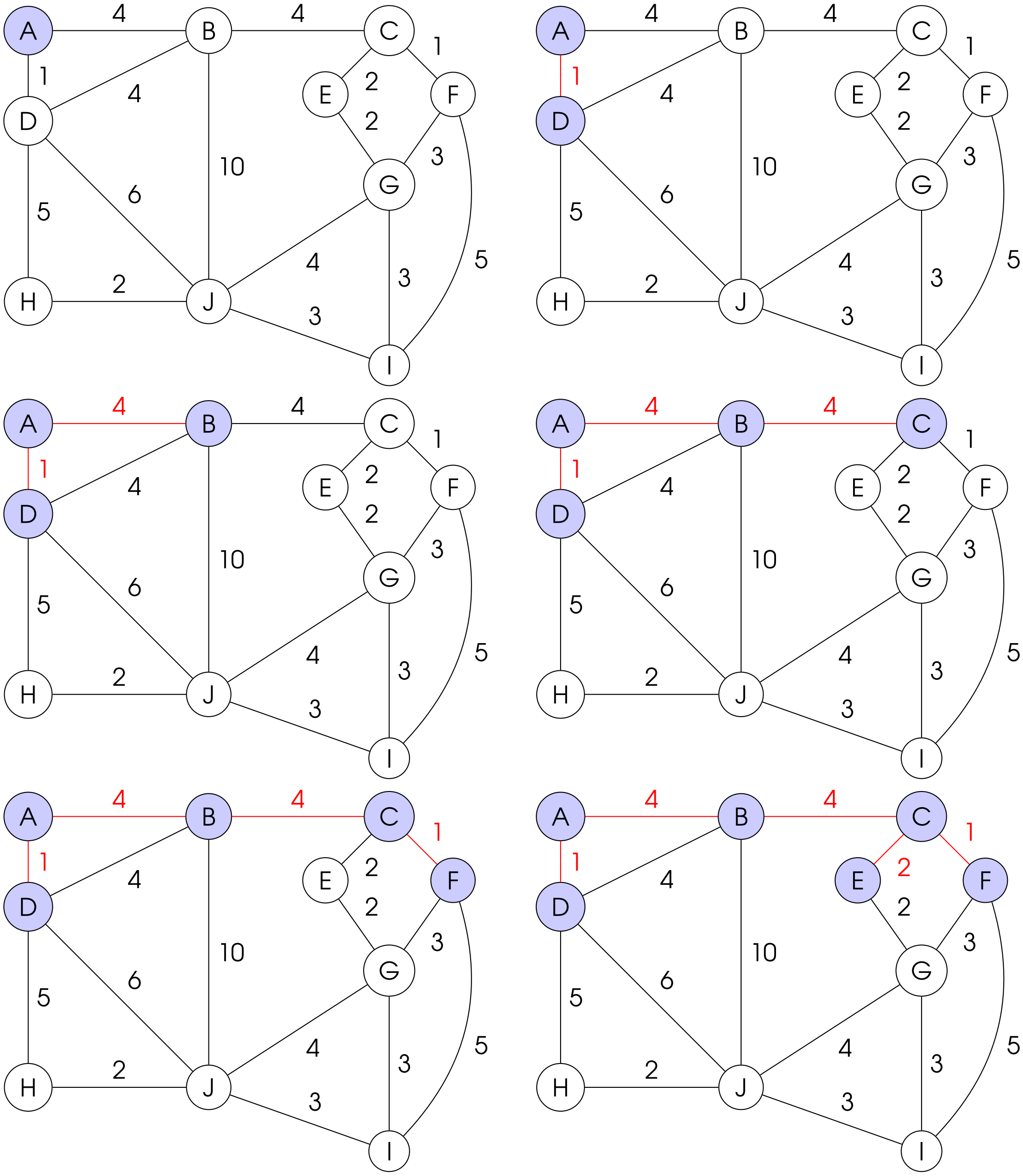
We can stop since we have n-1 edges.

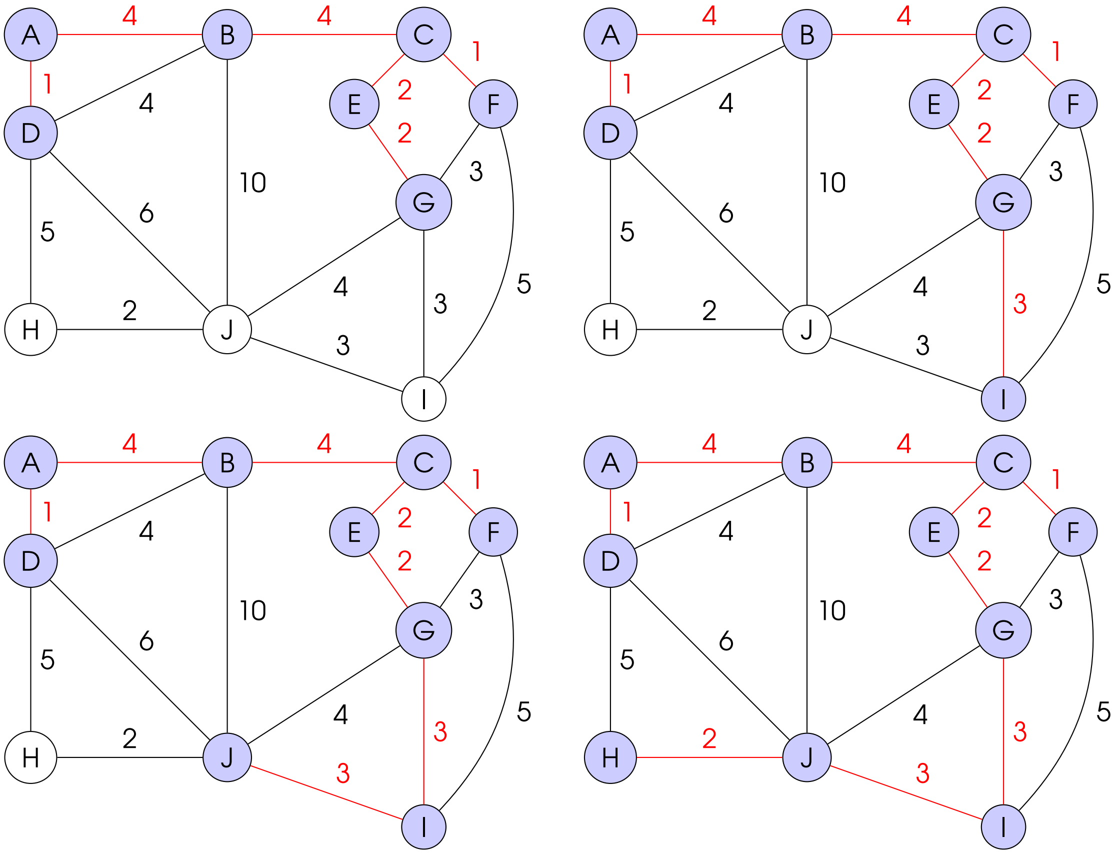
1.6 Prim's Algorithm

The steps are:

- Initialize new graph with one (arbitrary) node from the old graph.
- While new graph has fewer than n nodes:
  1. Find the node from the old graph with the smallest connecting edge to the new graph
  2. Add it to the new graph

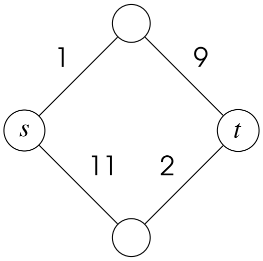
Every step will have joined one node, so that at the end we will have one graph with all n nodes and it will be a minimum spanning tree of the original graph.





1.7 Max flow

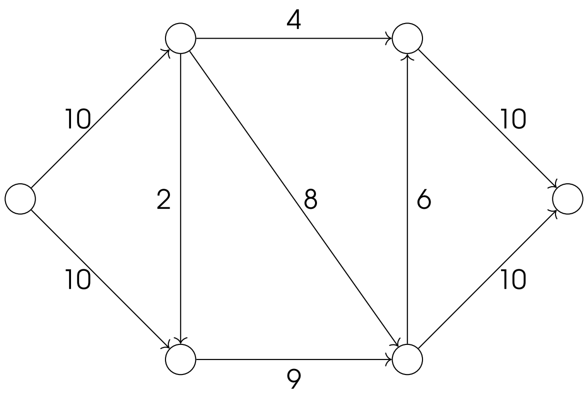
Given a network directed from a source vertex to a target vertex with a maximum capacity on each arc, determine the highest value of a flow from the source to the target which respects the capacities and is preserved at each intermediate vertex.

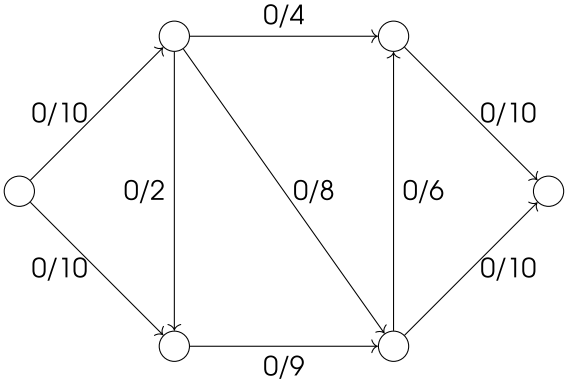


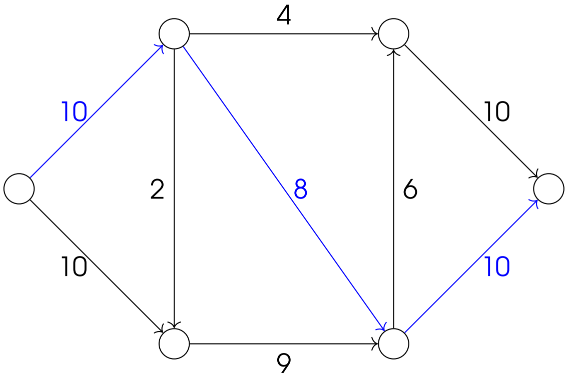
1.8 Method of Ford & Fulkerson

1. Start with a flow of zero.
2. At each iteration consider the residual graph composed of
   - the original arcs with capacity = original capacity - the flow on the arc. This is called the residual capacity.
     ⇝if this value is zero, the arc is eliminated.
   - Arcs opposite to the original ones with capacity equal to the flow passing on the corresponding arc
     ⇝if this value is zero, the arc is eliminated.

Try to find a path from source to target on the residual graph and update the flow with the minimum capacity along the path (increase flow for original arcs, decrease for opposite arcs).
Update the residual graph.
construct the re a path not exist, we're done: the flow is maximal.

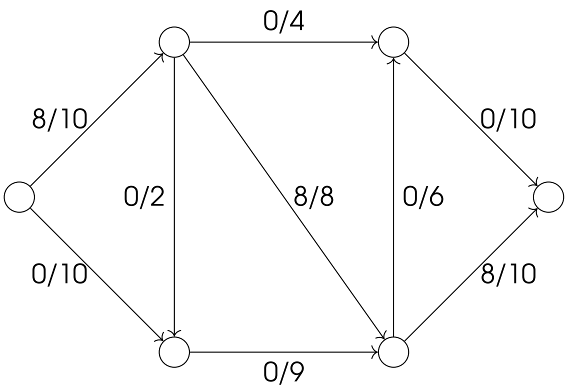




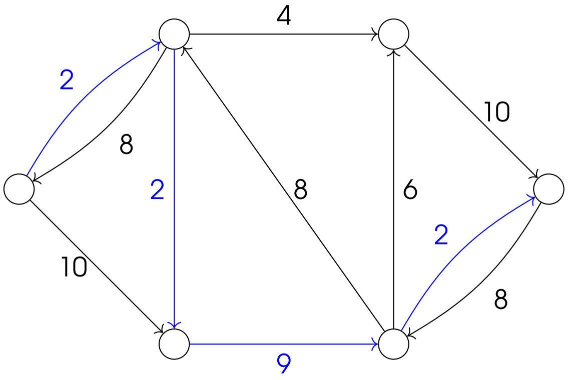


Start with a flow of zero.

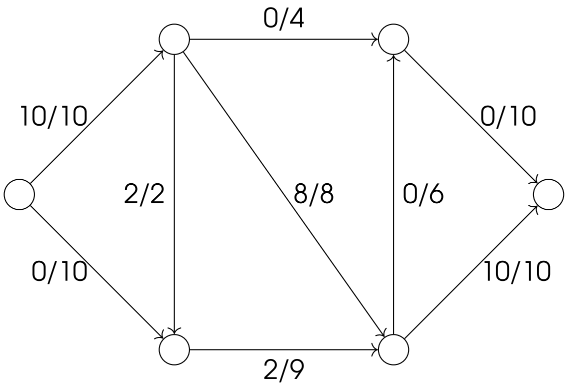
Construct the residual graph as follow:
-Keep the same edges
-Capacity of edges is actual capacity - flow
-Backward arrow if we have flow
Select a path from s to t.
Therefore the residual capacity is 8.



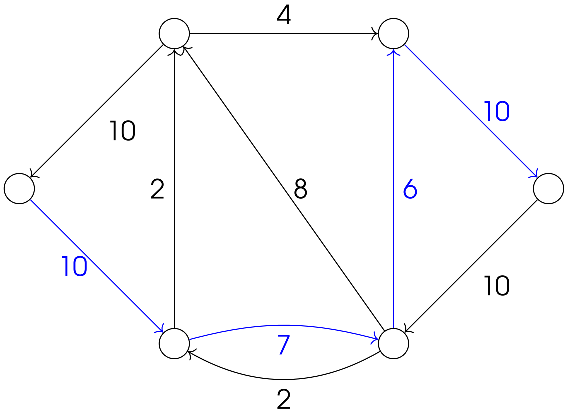
Add the residual capacity to the flow.



Construct the residual graph and select a path.Therefore the residual capacity is 2.

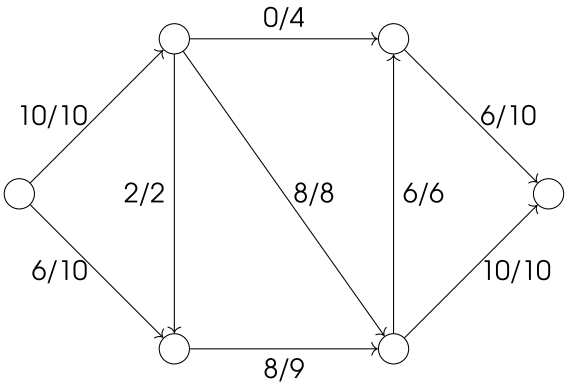


Add the residual capacity to the flow.

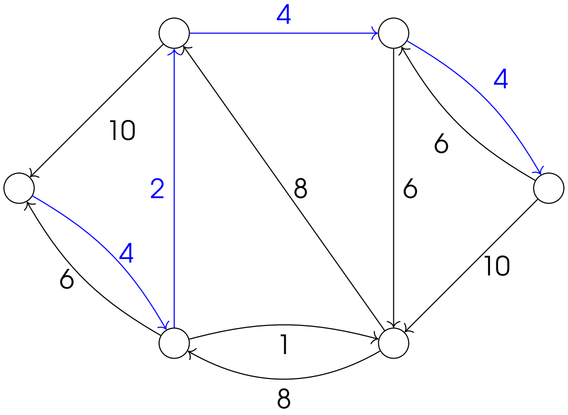


Construct the residual graph and select a path.

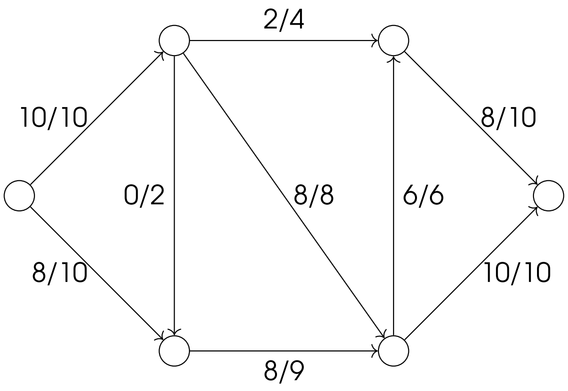
Therefore the residual capacity is 6.

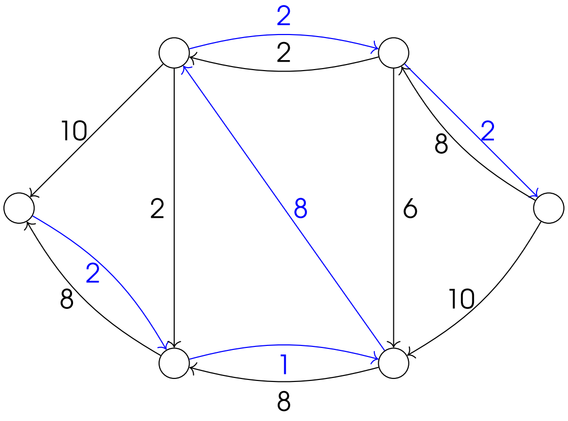


Add the residual capacity to the flow.



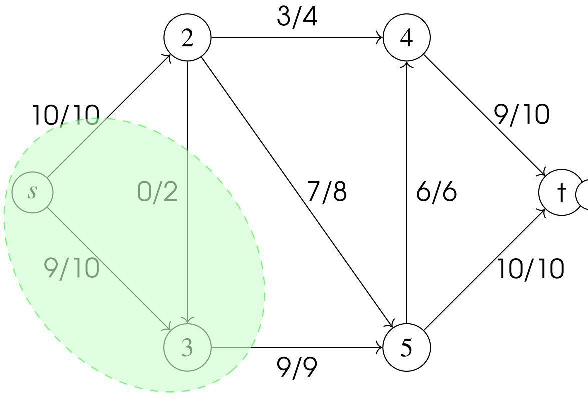
Construct the residual graph and select a path. Therefore the residual capacity is 2.

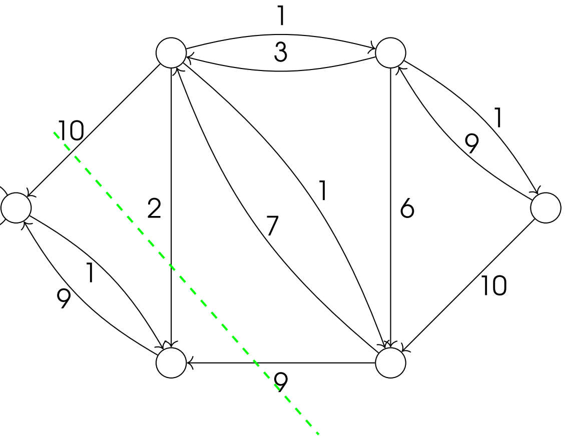




Add the residual capacity to the flow.

Therefore the residual capacity is 1.





The max flow is $1+2+6+2+8=19$.

Decide or not if there is no more path available it's not a good condition for a computer.We need to introduce an another condition.

Definition A **cut** corresponds to a set of edges (arcs) whose removal leaves a graph disconnected.

Sometimes a cut is denoted by a partitioning of the nodes:
Example above: s,3 and 2,4,5,t.

An **s-t cut** is a division of the nodes into two parts, with the source in one part and the target in the other.

The **value of a cut** is defined as the combined capacity of the outgoing arcs.
For instance, the green cut has a value/capacity of 10+9.
The **minimum cut**is among all cuts the one with the smallest value.

Theorem Min cut = Max flow and FF algorithm finds them.

Proof. Clearly Min cut $\geq$Max flow since any flow that we push from s to t has to go through the green area. Claim: when FF algorithm terminates, it gives a cut C and a flow F for which value(C) = value(F).

It follows min cut $\leq$ value(C) $=$ value(F) $\leq$ max flow. □

Definition By Edmonds-Karp-Dinitz, Apply the FF algorithm but always choose the path with the minimum number of edges allows to terminate in O(mn).

1.9 Min cost flow

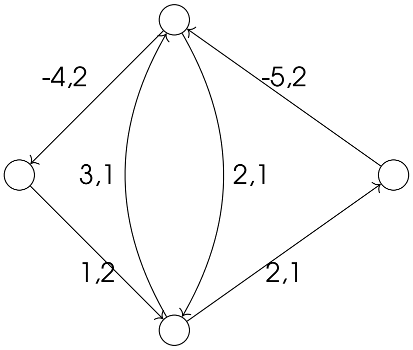
Given the network with costs and capacities and a flow value v, we need to find an $s-t$ flow of value v of minimum cost.

Algorithm:

1. Find a feasible flow of value v (for example by FF). Construct residual graph (negative of the cost for reverse arcs)
2. While there is a negative-cost cycle in residual graph:
   - Add flow over the cycle.
   - Update the residual graph.

Definition A flow is a minimum cost flow (of given value v) $\Longleftrightarrow$ Residual graph has no negative-cost cycles.

Example: Find a min cost flow of value 2.



2. Exercises

2.1 Tutorial 1: Basics of Graph Theory

Definition

1. A **graph** G is defined by a pair (V,E) where:
   - V is a finite set of points
   - E is a set of pairs of two distinct points.

   A graph is drawn by depicting the points in V as dots and the pairs in E as lines between two points.
2. The points defining a graph are called **vertices** and the lines are called **edges**.
   $\rightsquigarrow$ The number of vertices is denoted by $|V| = n$ and the number of edges by $|E| = m$.
3. A **graph** is called **simple** if there is at most one edge between every pair of points.
4. If for two vertices $u,v \in V$ we have the edge $e = \{u,v\} \in E$ then we say that u and v are **adjacent** or u and v are incident to e / e is incident to u and v.
   Two edges that share a vertex are also said to be adjacent.
5. The number of edges incident to vertex $v \in V$ is called the **degree** of v and denoted by $d(v)$.
6. A **graph** is called **regular** if all vertices have the same degree.
7. A **graph** is called **k-regular** if all vertices have degree k.

Exercise 1.7 How many 2-regular graphs exist with 5 vertices?

Exercise 1.8 How many 3-regular graphs exist with 5 vertices?

Exercise 1.11 How many edges has a 5-regular graph on 16 vertices?

Exercise 1.12 How many edges has a k-regular graph on n vertices?

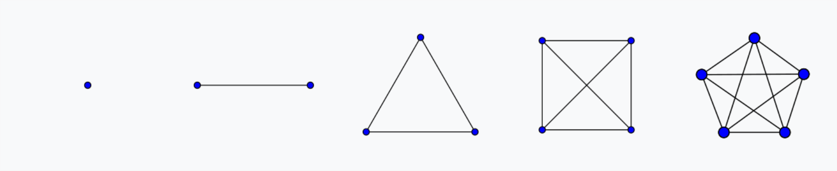
Exercise 1.16 Prove that every graph has an even number of points with odd degree.

Exercise 1.19 Prove that a graph $G = (V,E)$ with $|V| = n$ and $|E| = m$ has a vertex with degree $\leq \frac{2m}{n}$ and a vertex with degree $\geq \frac{2m}{n}$.

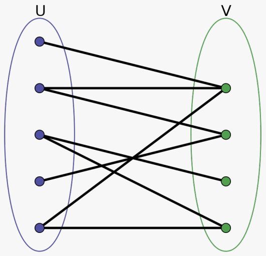
Exercise 1.21 Prove that every graph $G = (V,E)$ with $|V| \geq 2$ has two vertices of the same degree.

Definition

1. A **graph** $G = (V, E)$ is **complete** if each pair of points is adjacent. A complete graph on n points is denoted by K_n.



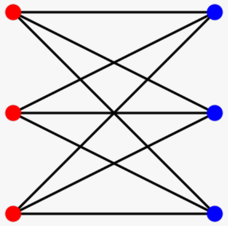
2. A **graph** G is **bipartite** if V can be split into two sets V_1 and V_2 such that for each edge $e = \{u, v\} \in E$, we have $|e \cap V_1| = |e \cap V_2| = 1$.



Informally, bipartite means that there are only arrows going across V_1 and V_2.

3. A **complete bipartite graph** with $V = V_1 \cup V_2$ has edge set

$$E = \{\ \{v_1, v_2\}\ |\ v_1 \in V_1, v_2 \in V_2\}$$



The complete bipartite graph with $|V_1| = m$ and $|V_2| = n$ is denoted by $K_{m,n}$

Exercise 1.26 For which values of m and n is $K_{m,n}$ regular?

Definition

1. A **walk** in a graph $G = (V, E)$ is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $\{v_{i-1}, v_i\} \in E, \quad$ for $1 \leq i \leq k$. It is called a walk is from v_0 to v_k. We say that this walk has (combinatorial) length k.
2. If all vertices on the walk are distinct we call it a **path**.

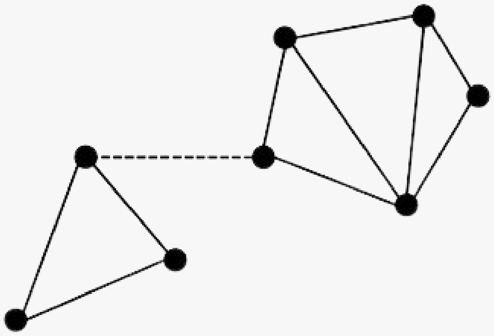
Exercise 1.41 How many paths are there from vertex 1 to vertex 3 in K_3?

Exercise 1.42 How many paths are there from vertex 1 to vertex n in K_n?

Exercise 1.43 Prove that that a graph of which each vertex has degree at least k, has a path of length k.

Definition

1. A **graph** is called **connected** if there is a path between any two of its vertices.



This graph becomes disconnected when the dashed edge is removed.

Exercise 1.47 Prove that every connected graph on n vertices contains at least$n-1$ edges.

Exercise 1.48 Does there exist a non-connected graph on 6 vertices containing 11 edges?

Exercise 1.50 Prove that every non-connected graph on n vertices contains at most $\frac{1}{2}(n-1)(n-2)$ edges.

Definition

1. A **graph** $G' = (V', E')$ is called a **subgraph** of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.
2. A **component** of $G = (V, E)$ is a maximal connected subgraph $G' = (V', E')$. That means, it is a connected subgraph and there is no other connected subgraph $G'' = (V'', E'')$ with $V' \subseteq V''$ and $E' \subseteq E''$.
   Hence, a graph G is connected if and only if it consists of exactly one component.

Exercise 1.59 Prove that if $G1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two distinct components of G then $V_1 \cap V_2 = \emptyset$.

Exercise 1.63 Prove that a graph $G = (V, E)$ with each vertex having degree at least $\frac{1}{2}(n-1)$ is connected.

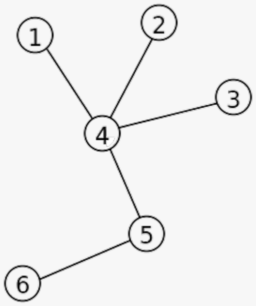
Definition

1. A **walk** $(v_0, v_1, \ldots, v_k)$ is called a closed walk or a **circle** if $v_0 = v_k$.
2. A cycle with all vertices distinct is called a **circuit**.
3. A circuit of length 3 is called a **triangle**

Exercise 1.67 Let G be a graph for which every vertex has a degree of at least 2. Prove that G contains a circuit.

Definition

1. $G = (V, E)$ is called a **forest** if G does not contain any circuit.
2. A connected forest is called a **tree**.



Exercise 1.75 Prove that between any pair of vertices in a tree there is exactly one path.

Definition

1. A vertex of degree 1 in a tree is called a **leaf** of the tree.

Exercise 1.76 Prove that every tree with at least two vertices contains a leaf.

Exercise 1.77 Derive from the previous exercise that every tree on n vertices has exactly $n-1$ edges.

Exercise 1.80 Prove that every tree with at least two vertices contains at least two leaves

Definition

1. An **Euler cycle** in a graph $G = (V, E)$ is a cycle $C = (v_0, v_1, \ldots, v_k)$ (remember that $v_0 = v_k$), with the property that every edge $e \in E$ is traversed exactly once; i.e., for each edge $e \in E$ there exists exactly one $i \in \{1, \ldots, k\}$ such that $e = \{v_{i-1}, v_i\}$.
2. A graph is called an **Euler graph** if it contains an Euler cycle.
3. An **Euler path** in a graph $G = (V, E)$ is a path $C = (v_0, v_1, \ldots, v_k)$ with the property that every edge $e \in E$ is traversed exactly once, but it is not required that $v_0 = v_k$.

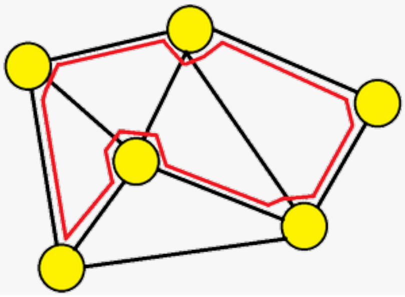
Theorem Let $G = (V, E)$ be a graph.

G is an Euler graph $\Longleftrightarrow$ G is connected and each of its vertices has even degree.

Exercise 1.88 Let G be a connected graph with exactly two points of odd degree. Use Euler's Theorem to prove that G contains a walk that traverses each edge exactly once.

Definition

1. A circuit C of G is called a **Hamilton circuit** (or Hamilton cycle) if each vertex of G appears on C.
2. A graph G is called a **Hamilton graph** if it contains a Hamilton circuit. Hamiltonian cycles / graphs are also called Rudrata cycles / graphs.

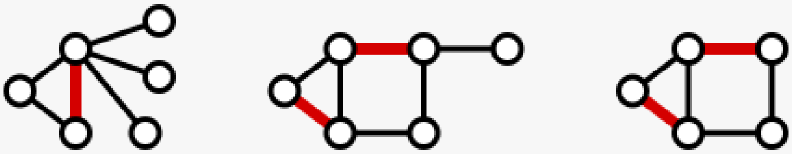


Exercise 1.90 Let n be an odd number. Show that on an $n \times n$ chess board, it is not possible for a knight (horse) to move over the board, hitting each square exactly once, while starting and ending in the same square.

Exercise 1.91 Show that for each n there exists a graph on n vertices such that each vertex has degree at least $\frac{1}{2}(n-1)$ and such that it is not a Hamilton graph.

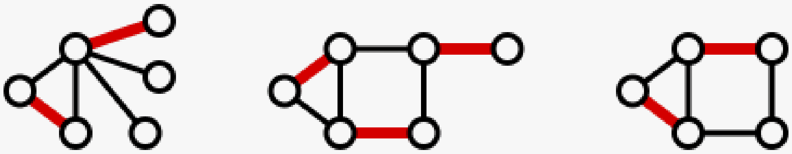
Definition

1. A **matching** M is a subset of the edges ($M \subseteq E$) such that no the edges in M have an end point (=nodes) in common.



Edges cannot touch each other.

2. The **size of the matching** M is the number of edges in it: $|M|$.
3. A **maximum macthing** in a graph is a matching of maximum size.



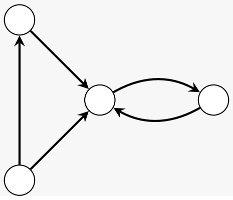
4. A **perfect matching** is a matching for which each vertex is the end point of some edge in the matching, i.e., $2|M| = |V|$.
   $\rightsquigarrow$Clearly, a graph with an odd number of vertices can not have a perfect matching.

Exercise X.1 Give an example of a connected graph with an even number of vertices that does not have a perfect matching.

Definition

1. A **directed graph** G is defined by a pair (V,A) where V is a finite set of points and A is an ordered set of pairs (=arcs) of two (distinct) points.
   An arc (u,v) is considered to be directed from u to v; v is called the head and u is called the tail of the arc. We use the following notation for directed graphs:

   $\delta^-(v)$: the set of arcs with head v (the arcs towards v)
   $\delta^+(v)$: the set of arcs with tail v (the arcs leaving v)
   $d^-(v)$: the number of arcs with head v (so $d^-(v) = |\delta^-(v)|$)
   $d^+(v)$: the number of arcs with tail v (so $d^+(v) = |\delta^+(v)|$)



2.2 Tutorial 2: Network flow algorithms

2.2.1 Flows

Definition

The networks we consider consists of a directed graph $G = (V,A)$, two special nodes $s,t \in V$ which are, respectively, a source and sink of G, and capacities $c_a > 0$ on each arc $a \in A$. (Notation: we also write c_{uv} for $a = (u,v)$.)

A **flow** f consists of a number $f_a \geq 0$ for each $a \in A$ such that:

- the capacity constraints are met: $0 \leq f_a \leq c_a$ for each $a \in A$,
- flow conservation is met, that means, the amount of flow leaving a point u (other than the source or sink) is equal to the amount of flow entering u:

$$\sum_{(u,v)\in A} f_{uv} = \sum_{(v,u)\in A} f_{vu} \text{ for all } u \in A \setminus \{s,t\}$$

The value of the flow is the nett flow that leaves s:

$$\sum_{(s,v)\in A} f_{sv} - \sum_{(v,s)\in A} f_{vs}$$

To simplify notation, we define from now on $f_{uv} = 0$ if $(u,v) \notin A$. Then we can write the value of the flow a

$$\sum_{(s,v)\in A} f_{sv} - \sum_{(v,s)\in A} f_{vs}$$

2.2.2 Cuts

Definition For a set $U \subset V$ define:
$\delta^-(U)$: the set of arcs with head in U and tail in $V \setminus U$ (arcs towards U)

$\delta^+(U)$: the set of arcs with tail in U and head in $V \setminus U$ (arcs leaving U)

If $U \subset V$ with $s \in U$ and $t \in V$ U then $\delta^+(U)$ is called an s-t cut. We call this the cut defined by U .

The next theorem says that the value of the flowis equal to the nett flow across any s-t cut.

Theorem Let f be a flow and $\delta^+(U)$ an s-t cut then

$$\text{value}(f) = \sum_{a \in \delta^+(U)} f_a - \sum_{a \in \delta^-(U)} f_a$$

Equivalently, when we define f_{uv} = 0 if $(u,v) \notin A$, then we can write this theorem as

$$\text{value}(f) = \sum_{u \in U} \left(\sum_{v \in V \setminus U} f_{uv} - f_{vu} \right)$$

Definition The **capacity of a cut** $\delta^+(U)$ is the total capacity of the arcs in the cut:

$$\text{cap}(\delta^+(U)) = \sum_{a \in \delta^+(U)} c_a = \sum_{\substack{(u,v) \in A: \\ u \in U \\ v \in V \setminus U}} c_{uv}$$

To simplify notation we write cap($\delta^+(U)$) as cap(U) and define $c_{uv} = 0$ if $(u,v) \notin A$. Then we can write the capacity of the cut defined by U as

$$\text{cap}(U) = \sum_{u \in U} \left(\sum_{v \in V \setminus U} c_{uv} \right)$$

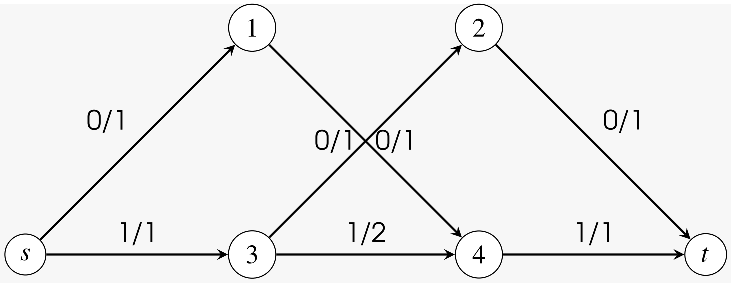
Theorem For any s-t flow f and s-t cut $\delta^+(U)$:

$$\text{value}(f) \leq \text{cap}(U)$$

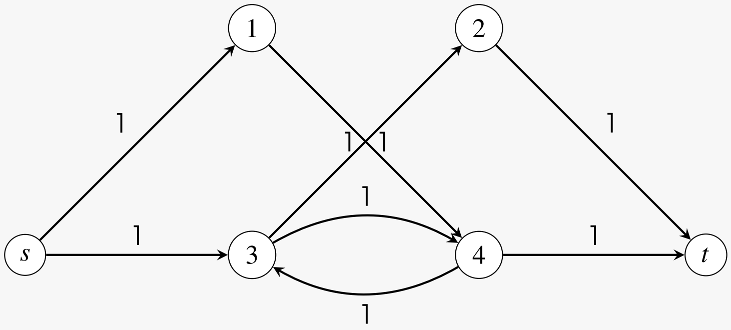
The theorm implies that the maximum value of a flow is no more than the minimum capacity of a cut. In fact, equality holds and the proof follows from the algorithm to find a maximum flow

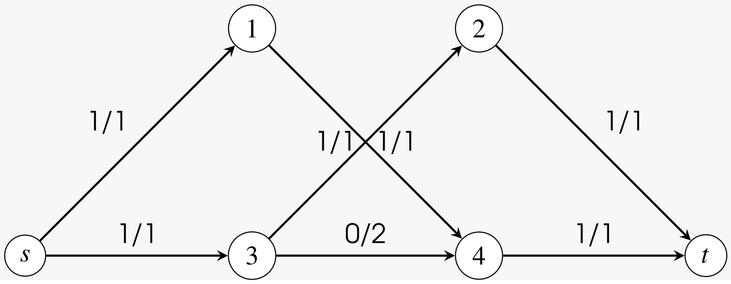
2.2.3 MaxFlow: The Ford-Fulkerson (FF) algorithm

Definition The algorithm starts with zero flow: $f_e = 0$ for all $e \in A$. Then, it repeatedly chooses an appropriate path from s to t and increases the flow along the arcs of this path as much as possible.
It is important to note that it may be necessary to reduce flow along an arc in order to increase the flow over the path. In the example below, the first number is the flow and the second number is the capacity. The maximum flow is 2. However, the initial flow f1 of value 1 as shown in the figure cannot be extended without reducing the flow on the arc in the middle.



Given a flow f, the **residual graph** (or residual network) gives for any arc the amount by which the current flow on the arc can be altered. For any arc $(u,v) \in A$, there is an arc (u, v) in the residual graph if there is still capacity left: $f_{uv} < c_{uv}$. But also, there is a reversed arc (v, u) in the residual graph if $f_{uv} > 0$. The residual graph for our example is as follows. The numbers show the amount by which the current flow can be extended, or, if the arc is reversed, the amount by which the current flow can be reduced. The graph has an st-path, namely $s,1,4,3,2,t$. The minimum capacity on the path is 1. So we add 1 over this path. The new flow has value 2 which is the maximum. A more complex example of the FF algorithm is given in the slides.





Theorem If all capacities are integer then the FF algorithm terminates and the flow f_a is integer for each arc $a \in A$.

Theorem The maximum value of an s-t flow is equal to the minimum capacity of s-t cut and FF-algorithm returns both a maximum flow and a minimum cut.

2.2.4 MaxFlow: Edmonds-Karp-Dinitz (EKD) algorithm

The algorithm applies the FF algorithm but in each iteartion it chooses the s-t path in the residual graph with the minimum number of arcs.

Theorem The number of iterations of the EKD algorithm is $O(nm)$

2.2.5 Mincost flow

Definition In the minimum cost flow problem we are given a network $G = (V,A)$ with $s,t \in V$ and a capacity c_a for any arcs a and, in addition, a cost that we denote by cost$_a$. The **cost of a flow f** is:

$$\sum_{a \in A} \text{cost}_a \cdot f_a$$

By this definition, the minimum cost (zero) is attained by sending no flow. In the minimum cost flow problem we want to find, for a given flow value v, a flow of minimum cost among the flows of value v.

A mincost flow can be computed as follows:
First, find any flow f of value v, which can be computed by, for example, the FF algorithm. Next, make the residual network as in FF but now for each reversed arc also make the cost negative. (Note that sending a flow over a reversed arc in the residual corresponds with reducing the flow.) Let C be a cycle in the residual graph. If we augment f by sending flow over C then the value of the flow remains v. However the cost may change: If the sum of the cost of the arcs in C is negative then the cost of the flow will decrease.

Mincost flow algorithm: (The cycle cancelling algorithm):

- Step 1: Find a feasible flow of value v. Make the residual graph.
- Step 2: While there is a negative-cost cycle C in the residual graph:
  - add the largest possible flow over C,
  - update the residual graph.

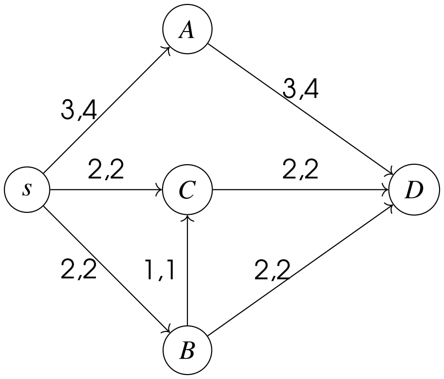
Theorem The mincost flow algorithm returns a minimum cost flow.

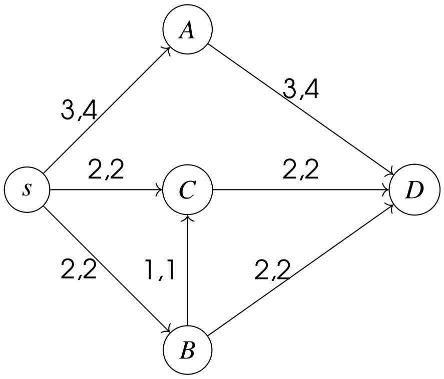
2.2.6 Flow exercises

Exercise 1 Consider the following problem: There are p families going out for dinner and together they use q tables. No two members of a family should sit at the same table. Family i has a_i people ($i = 1,2,\dots,p$) and table j has b_j chairs ($j = 1,\dots,q$). Formulate this problem as a maximum flow problem. (Make a sketch of the network including the capacities on the arcs.)

Exercise 2 Explain how you can find a maximum matching in a bipartite graph by using a maxflow algorithm. That means, formulate the matching problem as a flow problem and explain how you can deduce the maximum matching from the maximum flow.

Exercise 3 In the network below, each first number is the arc's capacity and the second number is its cost per unit flow. Find a minimum cost flow of value 3 by following the steps of the cycle cancelling algorithm. Start with a flow of value 3 over the path s, a, t.

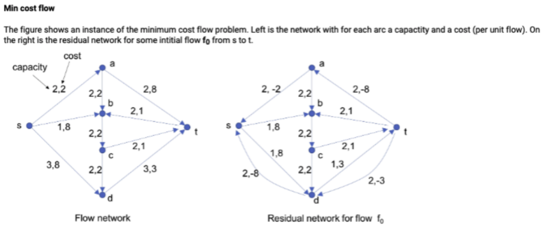




Exercise 4 Consider a flow network with integer capacities. Prove or disprove the following statements.
(a) If all capacities are even, then there is maximum flow in which f_a is even for all $a \in A$.
(b) If all capacities are odd, then there is maximum flow in which f_a is odd for all $a \in A$.

Exercise A Draw a small graph for which Dijkstra's algorithm does not give the shortest path from node A to node B.

Exercise B Fill in the blanks.



Min cost flow

The figure shows an instance of the minimum cost flow problem. Left is the network with for each arc a capactity and a cost (per unit flow). On the right is the residual network for some intitial flow **f₀** from s to t.

The value of the initial flow f_0 is:
The cost of the initial flow is:
The ressiual network has a negative cost cycle.This is cycle:
Assume we add the maximum possible amount of flow overthis cycle to the initial flow f_0.
Then he cost of the flow decreases by

3. Solutions

Theorem 1 The sum of all the degrees of all the vertices of an graph is equal to twice its number of edges:

$$\sum_{v \in V} d(v) = 2 \cdot |E|$$

Proof. We can count the number of edge ends in two different ways:

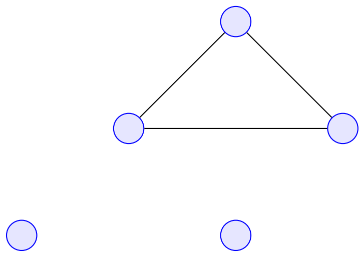
- it is the double of the number of edges since each edge having two ends
- it is also the sum of the degrees of each vertex

. □

This theorem leads to the handshake lemma:

The number of odd-degree vertices is even.

Solution 1.7 One graph which is a cycle on all the 5 points denoted by C_5:



Solution 1.8 Each node of 3-regular graph has degree 3 so odd. If there is 5 vertices then the number of odd-degre vertices is odd which is absurd according to the handshake lemma. Therefore, none.

Solution 1.11 Using the Theorem 1, we have:

$$\sum_{k=1}^{16} d(v_k) = 2m \Longleftrightarrow \sum_{k=1}^{16} 5 = 2m \Longleftrightarrow 5 + \cdots + 5 = 2m \Longleftrightarrow 5 \cdot 16 = 2m \Longleftrightarrow m = \frac{5 \cdot 16}{2}$$

Solution 1.12 Using the Theorem 1, we have:

$$\sum_{i=1}^{n} d(v_i) = 2m \Longleftrightarrow \sum_{i=1}^{n} k = 2m \Longleftrightarrow k + \cdots + 5 = 2m \Longleftrightarrow k \cdot n = 2m \Longleftrightarrow m = \frac{k \cdot n}{2}$$

Solution 1.16 To prove the handshaking lemma, we'll use the Theorem 1:

$$\sum_{i=1}^{n} d(v_i) = 2m \Longleftrightarrow \sum_{\text{d(v) even}} d(v) + \sum_{\text{d(v) odd}} d(v) = 2m$$

even + even = even $\Longrightarrow \left(\sum_{\text{d(v) even}} d(v)\right)$ is an even number.

even + odd = odd but 2m is always even. Therefore $\left(\sum_{\text{d(v) even}} d(v)\right)$ has to be even.

Solution 1.19 Using the Theorem 1, we have:

$$\sum_{i=1}^{n} d(v_i) = 2m \Longleftrightarrow d(v_1)+\cdots+d(v_n) = 2m \Longleftrightarrow \frac{d(v_1)+\cdots+d(v_n)}{n} = \frac{2m}{n}$$

We proved that on average a vertex is of degree $\frac{2m}{n}$.Hence, there must be a vertex less than $\frac{2m}{n}$ and a vertex of degree above $\frac{2m}{n}$.

Solution 1.21 Assume all vertices has a different degree.

There is n degrees different therefore $\forall v \in V$, $d(v) \in \{0,\ldots,n-1\}$.

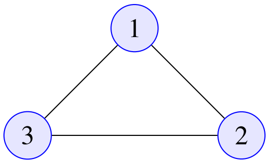
However, there cannot be two vertices u, v with $d(u) = 0$ **and** $d(v) = n-1$.

So, there are at most $n-1$ different degrees.

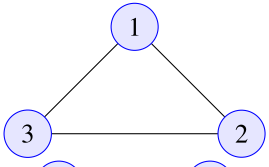
Since we have more vertices than different degrees, by the Dirichlet's principle there must be at least two vertices that have the same degree.

Solution 1.26 Only regular if m = n.

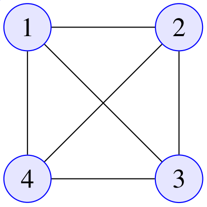
Solution 1.41 There are 2 paths: (1,3) and (1,2,3)



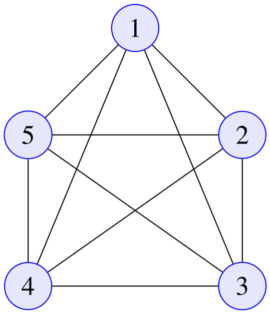
Solution 1.42 Given a starting node and a final node in K_n, how many paths there is of size ? Let's work on examples to find a logic:



path of size 1: 1
path of size 2: 2



path of size 1: 1
path of size 2: 2
path of size 3: 2



path of size 1: 1
path of size 2: 5-2 = 3
path of size 3: (5-2)(5-3)=6
path of size 4: (5-2)(5-3)(5-4) = 6

Hence, in general:

path of size 1: 1

$\vdots$

path of size i: $(n-2)\cdots(n-i)$

$\vdots$

path of size n-1: $(n-2)\cdots 1$

To find the total number of paths, you would have to add all these numbers:

$$1+\sum_{l=2}^{n-1}\left(\prod_{k=2}^{l}(n-k)\right)$$

Solution 1.43 Construct a path as follow:

Step 1: Take any vertex and call it v_0. We build the path (v_0)
Step 2: Define i to be the index of the last vertex of our path.
Step 3: We add v_{i+1} to the path where v_{i+1} is in the neighborhood of v_i and $v_{i+1} \notin \{v_0,\dots,v_i\}$
Step 4: Repeat 2 & 3 while $i \leq k$

The existence of v_{i+1} is guaranteed since v_i has at least k neighbors and $i < k$.When $i = k$, we have the path $(v_0,\dots,v_k)$

Solution 1.47 By induction on n, we will prove

P(n) : "A connected graph on n vertices contains at least $n-1$ edges

Base case: for n = 1, we have 0 edges. Hence P(1) is true.

Induction hypothesis: Assume P(n) is true: every connected graph with n vertices has at least n-1 edges.

Induction step: Let $G = (V,E)$ be a connected graph with n+1 vertices: $v_1,\dots,v_{n+1}$.

Define $G' = (V \setminus \{v_l\}, E \setminus \{v_l, v_i\})$ with $1 \leq i \leq n+1$ and v_l any vertice of G such that $d(v_k) = \min\{d(v) \mid v \in E\}$. G has n vertices and G is connected.

By our induction hypothesis, it follows that G′ has n-1 edges. Moreover G is a connected graph therefore there is at least 1 edge between v_l and another vertice.Therefore G has at least $n-1+1 = n$ edges: P(n+1) is true.

Hence by mathematical induction P(n) is correct for all positive integers n.

Solution 1.48 No since a non-connected graph on 6 vertices contains at most 10 edges.

Solution 1.50 Let $G = (V, E)$ be a non-connected graph on n vertices.
Define $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with:

$$(V_1, V_2 \subset V) \wedge (\forall \{u, v\} \in V,\ \ u \in V_1 \text{ and } v \in V_2 \Rightarrow \{u, v\} = \emptyset)$$

In other words, there is no edge linking a vertice of V_1 to a vertice of V_2.
Let $|V_1| = k$. Therefore $|V_2| = |V \setminus V_1| = n - k$.

Solution 1.59 Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two distinct components of G.

G_1 is a maximal connected subgraph so there is no other connected subgraph $G' = (V', E')$ with $V_1 \subseteq V'$ and $E_1 \subseteq E'$.

In particular G_2 is a (maximal) connected subgraph thus $V_1 \not\subseteq V_2$ and $E_1 \not\subseteq E_2$.

G_2 is a maximal connected subgraph so there is no other connected subgraph $G' = (V', E')$ with $V_2 \subseteq V'$ and $E_2 \subseteq E'$.

In particular G_1 is a (maximal) connected subgraph thus $V_2 \not\subseteq V_1$ and $E_2 \not\subseteq E_1$.

$$(V_1 \not\subseteq V_2 \wedge V_2 \not\subseteq V_1) \Longleftrightarrow V_1 \cap V_2 = \emptyset \ \ \wedge \ \ (E_1 \not\subseteq E_2 \wedge E_2 \not\subseteq E_1) \Longleftrightarrow E_1 \cap E_2 = \emptyset$$

Solution 1.63 Let $G = (E, V)$ be a graph with $\forall v \in V,\ d(v) \geq \frac{n-1}{2}$.

Assume it's not connected. G can be partitioned into 2,3,.., n components.
Let C be a component of G therefore it's a (maximal) connected subgraph.
If we split into 2 components, $|C| = \frac{n}{2}$.
If we split into 3 components, $|C| = \frac{n}{3}$. (and so on until n)
The best case scenario is into 2 components therefore C has at most $\frac{n}{2}$ vertices.
However, all vertices of C can have at most $\frac{n}{2} - 1$ edges. But $\frac{n}{2} - 1 < \frac{n}{2} - \frac{1}{2} = \frac{n-1}{2}$.

Since all vertices of G had to be greater or equal $\frac{n-1}{2}$, it's absurd to have at most our vertices in C of degree $\frac{n}{2} - 1$. Therefore G is connected.

Solution 1.67 Let $G = (V, E)$ be a graph where $\forall v \in V,\ d(v) \geq 2$.

Step 1: Start anywhere by coloring a vertex.
Step 2: Take any usued edges and color the vertex.
Step 3: Repeat step 2 until return to a vertex.

We'll be able to continue in the worst case until we have covered all the vertices once. Since each vertex has degree at least 2, so there will be an unused edge to exit on. If you ever return to a vertex where you've been, you've got a cycle.

Solution 1.75 Assume that between any pair of vertices in a tree there is 2 path: $p_1 = (v_k, ..., v_l)$ and $p_2 = (v_k, ..., v_l)$
Therefore, by merging p_1 and the reversed path p_2 without its first vertice, we got a circle.
That's absurd. Therefore in any pair of vertices in a tree there is exaclty one path.

Solution 1.76 Let G be a tree without leaf. Therefore all vertices are of degree 2. By the exercise 1.67, G must contains a circuit.Absurd since G is a (connected) forest.
Thus G always has a leaf.

Solution 1.77 By induction, we will prove:

P(n):" Every tree on n vertices has exactly $n-1$ edges".

Base case: for n = 1, we have 0 edges. Hence P(1) is true.

Induction hypothesis: Assume P(n) is true: every tree on n vertices has exactly n-1 edges.

Induction step: Let $G = (V, E)$ be a tree with n+1 vertices.By the exercise 1.76 there must be a leaf somewhere in G, remove it. By our induction hypothesis, the new graph has n-1 edges, if we put back our leaf it's clear that G has n-1+1 = n edges. Therefore P(n+1) is true.

Hence by mathematical induction P(n) is correct for all positive integers n.

Solution 1.80 By the Theorem 1,

$$\sum_{v \in V} d(v) = 2 \cdot |E| = 2(n-1)$$

Therefore the average degree of a vertice in a tree is $\frac{2n-2}{n}$.

Assume there is only one leaf in a tree.Then all other vertices should be of at least degree 2. (not 0 since a tree is a connected forest)
Therefore the minimum average of a tree is:

$$\min\left(\frac{1 + d(v_2) + \cdots + d(v_2)}{n}\right) = \frac{1 + 2(n-1)}{n} = \frac{2n+1}{n}$$

However

$$\frac{2n-2}{n} < \frac{2n+1}{n}$$

Our average is too big given our assumption therefore there must be more than 1 leaf.

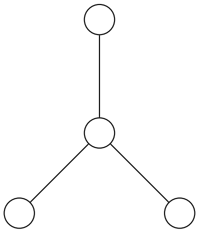
Solution 1.88 Let G be a connected graph with exactly two vertices of odd degree.
Add one linking edge between those two vertices of odd degree.Now all vertices have an even degree.
Therefore, by the Euler Theorem there is an Euler cycle.
By deleting the linking edge, the Euler cycle became an Euler path.

Solution 1.90 Let n be an odd number. We want to show that on an $n \times n$ chess board,it is not possible for a knight (horse) to move over the board, hitting each square exactly once, while starting and ending in the same square.

A knight moves from white to black and vice versa. Since n is odd, n^2 is odd. If it starts on black, then it is on white after n 2 moves. But then it cannot be back at its starting point.

Solution 1.91 We want to prove the existence of a graph G with n vertices with $\forall v \in V, d(v) \geq \frac{n}{2} - 1$ such that there is no Hamilton Circuit.
If n is even then take two components with n/2 vertices of degree $\frac{n}{2} - 1$.
If n is odd then take two complete graphs on $n/2$ vertices and merge two vertices to have only one.All vertices will be of degree $\frac{n}{2} - 1$ except the merged vertex that will have $n - 2$

Solution X.1

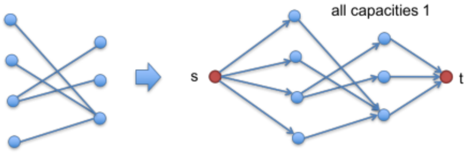


Solution 1 Take vertices $(v_1, v_2, \dots, v_p)$ where v_i corresponds with family i.
For each table j, take a vertex w_j . Further add points s and t.

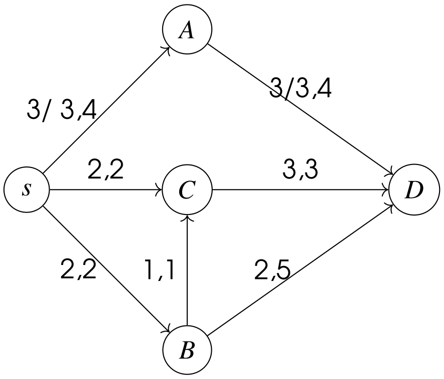
There is an arc (s, v_i) with capacity a_i for each $i \in \{1, \dots, p\}$.
There is an arc (w_j, t) with capacity b_j for each $j \in \{1, \dots, q\}$.
For each pair (i, j), there is an arc (vi, wj) with capacity 1.

An upper bound on the maximum flow vale is $\sum_i a_i$ since that is the maximum flow that can leave s. If there exists a flow of value $\sum_i a_i$ then this immediately give a solution to the dinner problem since, by Theorem 3, the flow on each arc (v_i, w_j) is either 0 or 1. If the flow value on (v_i, w_j) is one then a person from family i is seated at table j.

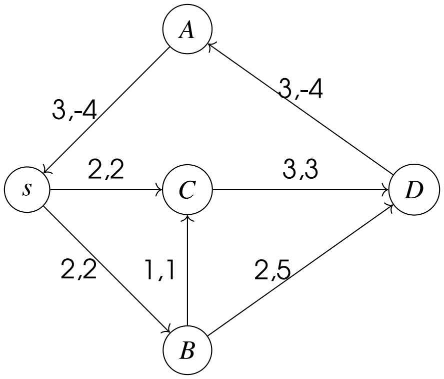
Solution 2



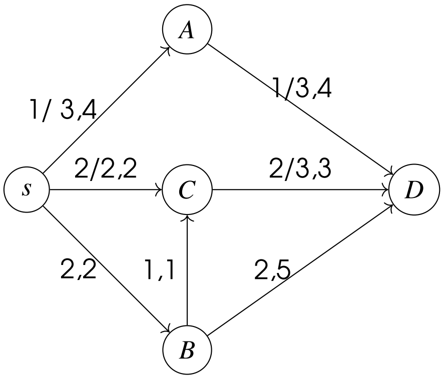
Solution 3



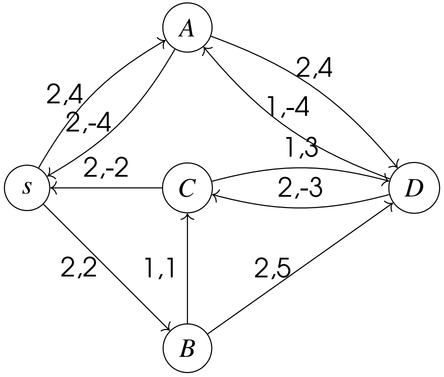
Cost: $3 \cdot 4 + 3 \cdot 4 = 24$



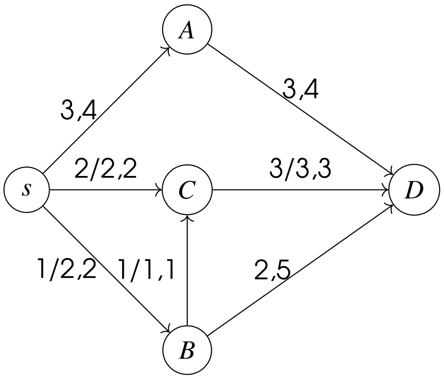
Residual graph
Look for a negative circle: $-4+2+3-4=-3$ and the residual capacity is 2.



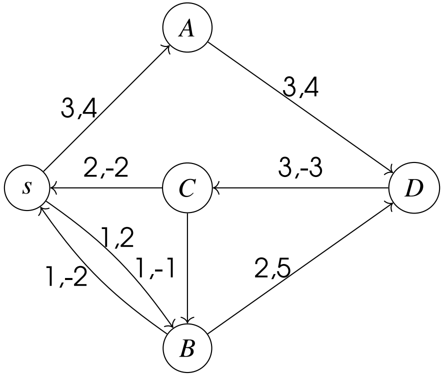
Cost: $24-2 \cdot 3 = 18$



Residual graph
Look for a negative circle: $-4-4+2+1+3 = -2$ and the residual capacity is 1.



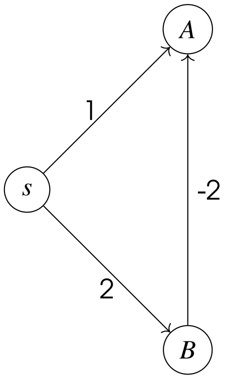
Cost: $18-2=16$



Residual graph
No negative circle hence 16 is the minimal cost flow.

Solution 4 (a) True. Divide all capacities by 2. Theorem 3 says that there is an optimalflow with integer flow values fa on each arc. Now multiply all f_a by two.
(b) Not true. This is a counter example where All edges have odd capacity, but the maxflow is even.

Solution A



Solution B

The value of the initial flow f_0 is:$2+2$
The cost of the initial flow is:$16+6=22$
The ressiual network has a negative cost cycle.This is cycle:$abct$
Assume we add the maximumpossible amount of flow overthis cycle to the initial flow f_0.
Then he cost of the flow decreases by 6.