MATH 333 Discrete Mathematics
Chapter 6 Graph and Network

Jianan Lin

linj21@rpi.edu

May 24, 2022

Definition

A graph $G = (V, E)$ includes vertex set V and edge set E.

Directed graph VS undirected graph: depends on direction of edge.

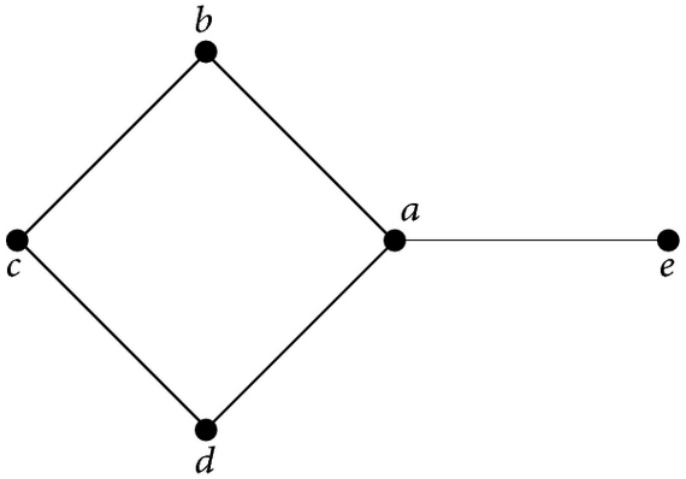


Figure 1: an undirected graph

Definition

If there is an edge between vertex v_1, v_2 in some undirected graph, then both $v_1 \to v_2$ and $v_2 \to v_1$.

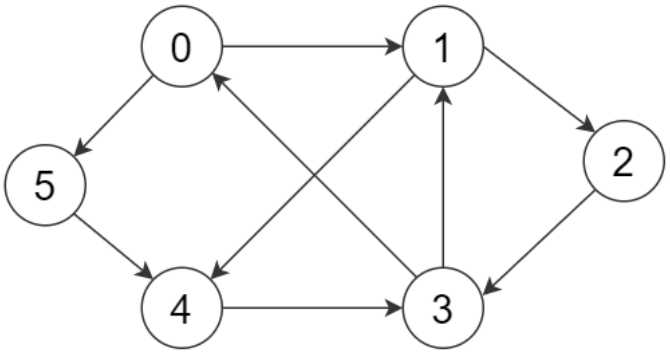


Figure 2: a directed graph

Definition

A cycle is: we can start from one vertex and finally return this vertex without using the same edge.

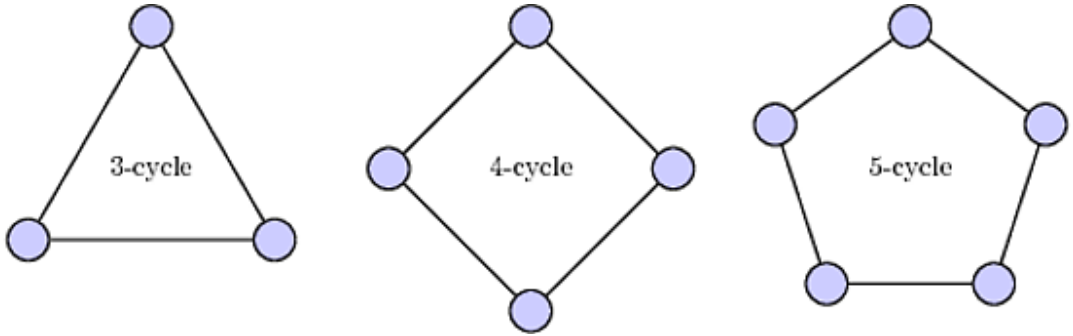


Figure 3: a cycle

Even if there are only 2 vertices in graph, we can also have a cycle. Consider there are two or more edges between the two vertices.

Definition

Tree: If there is no cycle in an undirected graph, then it is a tree.

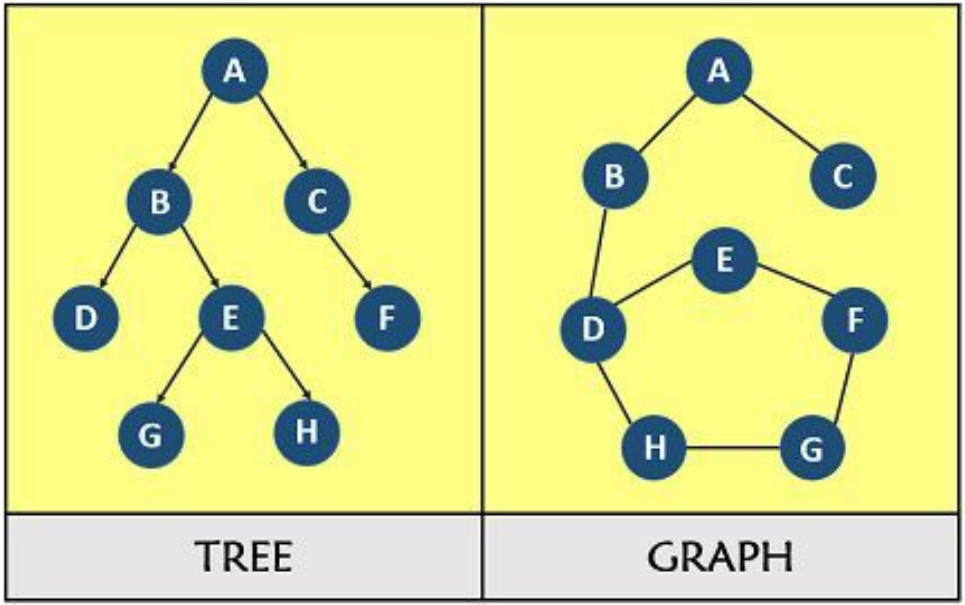


Figure 4: graph and tree

Definition

Usually we only consider simple graph, i.e. there is only one edge between any pair of vertices.

We use $n = |V|$ to denote the number of vertices, and $m = |E|$ to denote the number of edges.

In a simple graph, obviously we have

$$0 \leq m \leq \frac{n(n-1)}{2}$$

Connected graph: We can start from any vertex and reach any vertex.

A connected graph needs at least $m \geq n - 1$ edges.

Definition

There are 2 ways to denote an edge.

1. Use index $e_j \in E$.

2. Use pair of edges (v_i, v_i'). Notice if the graph is directed, then order of the two vertices is important.

Degree of a vertex v: how many edges connected.

In-degree: how many edges coming into it.

Out-degree: how many edges leaving it.

See graph in the previous pages to calculate the degree.

Definition

We can add a weight to each edge. The weight can denote cost, or utility, or nothing.

Also there may be weight on each vertex. But this is not common.

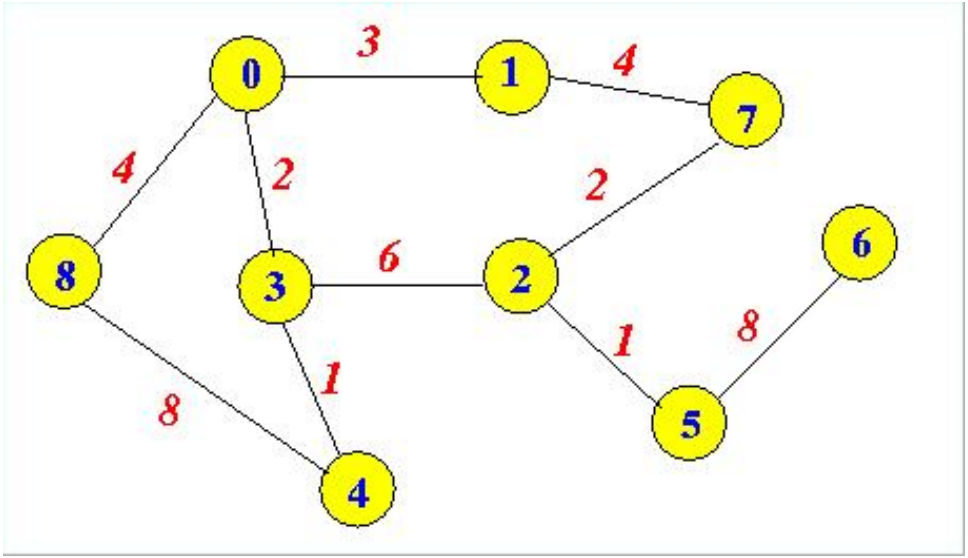


Figure 5: weighted graph

Definition

Use $d(v)$ to denote the degree of a vertex v. We have

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

There are even number of vertices with odd degree. We can use contradiction proof.

Suppose there are $2k + 1$ vertices with odd degree, then the sum of degree of them is also odd. Also we know the sum of degree of vertices with even degree is still even. We get into a contradiction.

MST

Sub-graph: A graph $G' = (V', E')$ is sub-graph of $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$.

Spanning Tree: A tree which is sub-graph of G.

A spanning tree has just $m = n - 1$ edges.

Minimum Spanning Tree (MST): In a weighted graph, MST is a spanning tree with smallest sum of edge weight.

There are many algorithms to get MST. We introduce two famous.

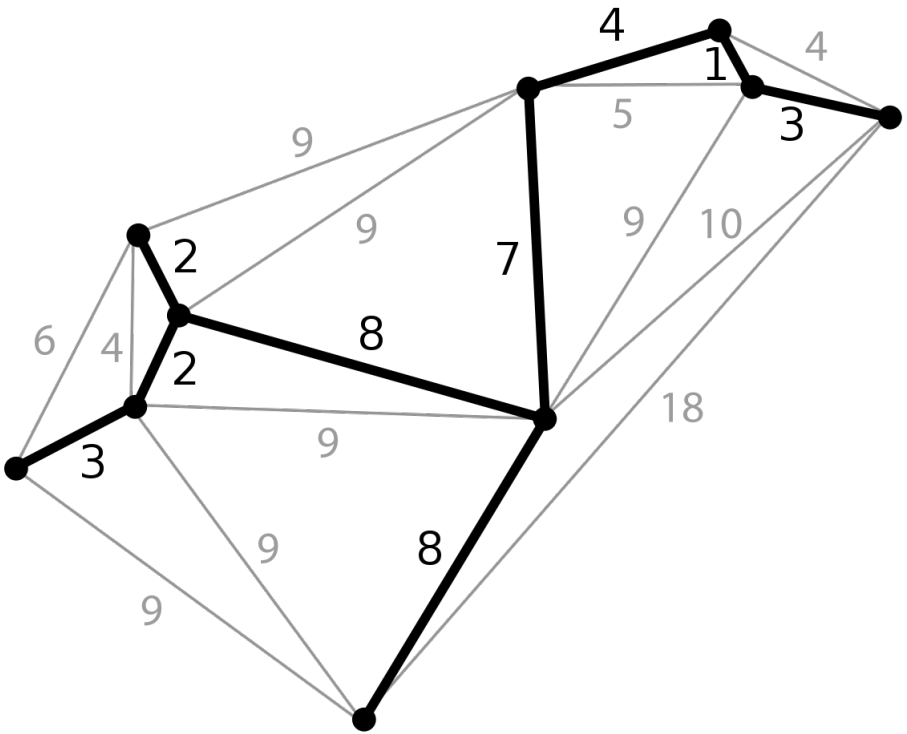


Figure 6: minimum spanning tree

Prim's algorithm.

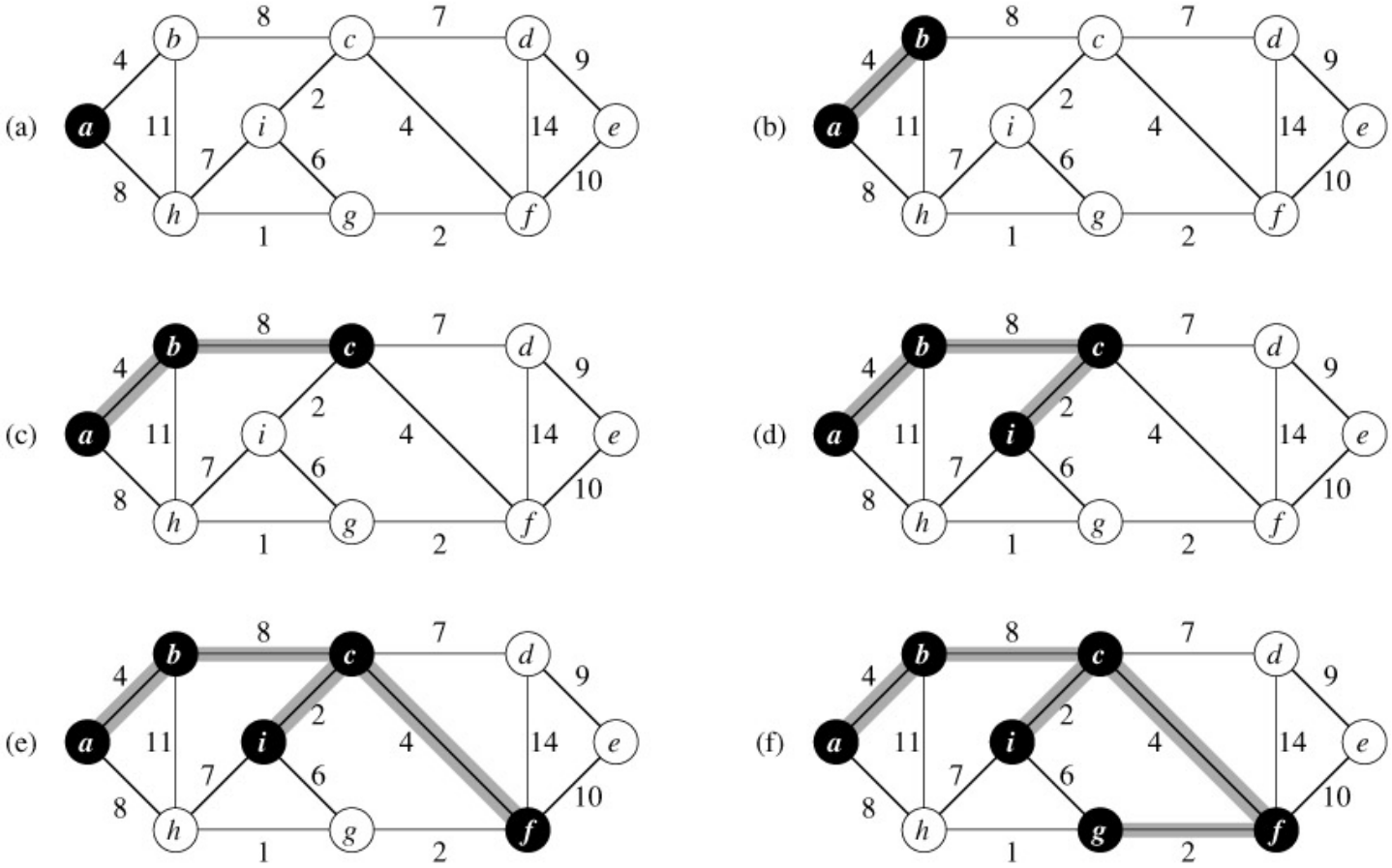
1. Start with arbitrary vertex v. We have a set S of vertices and at first only $v \in S$. Also we have a set P of edges and at first $P = \emptyset$.

2. Find the edge satisfying: I. one end connecting S and the other not; II. with smallest weight.

3. Add the other end vertex to S, add the edge to P.

4. Repeat step 2 and 3 until we get a tree (or repeat $n - 1$ times).

MST



Kruskal's algorithm.

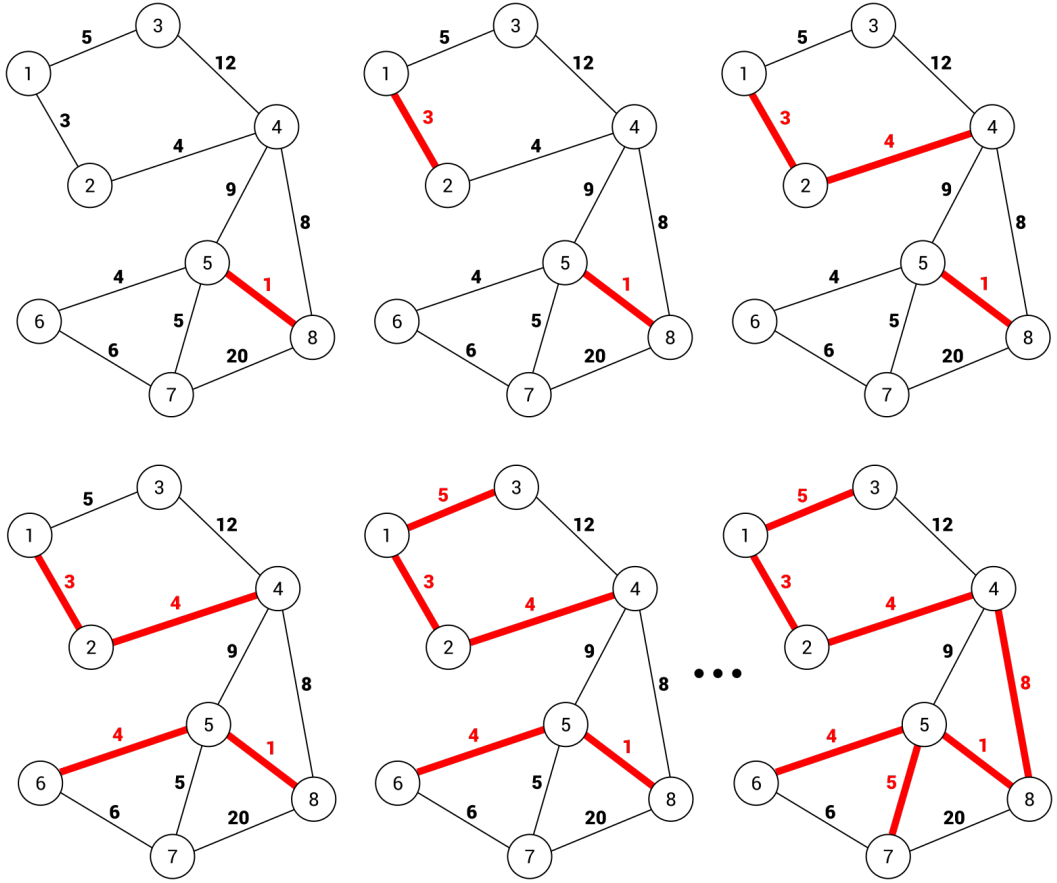
1. We have a set S of vertices which is empty at first, and a set P of edges which is empty at first.

2. Find the edge satisfying: I. does not form a cycle with edges in P; II. with smallest weight.

3. Add the vertices into S and add the edge into P.

4. Repeat step 2 and 3 until we get a tree (or repeat $n-1$ times).

MST



Problem: if we let all the weight +1, will the MST be changed?

Hint: For convenience, we can suppose all the weights are different, so we have only 1 MST (this does not need to prove).

Problem: if we let all the weight +1, will the MST be changed?

Hint: For convenience, we can suppose all the weights are different, so we have only 1 MST (this does not need to prove).

Answer: No. Since the sum of weight of each tree just $+(n-1)$ where $n = |V|$.

Problem: if we let all the weight w_e be w_e^2, will the MST be changed?

Hint: For convenience, we can suppose all the weights are different, so we have only 1 MST (this does not need to prove).

Problem: if we let all the weight w_e be w_e^2, will the MST be changed?

Hint: For convenience, we can suppose all the weights are different, so we have only 1 MST (this does not need to prove).

Answer: No. We can use any of the two algorithms above and the result of each step is the same.

What's the time complexity of the two algorithms?

Prim: $O(n^2)$

Kruskal: $O(m \log(m))$

Here, $n = |V|, m = |E|$. In practice, Prim's algorithm is suitable for dense graph (i.e. many edges) and Kruskal's algorithm is suitable for sparse graph (i.e. few edges).

PS: in order to reach this complexity, we should use some special data structure. If you only write code directly, you cannot get this complexity.

Search

Does a node belong to the graph? We have two common algorithms to find it.

Breadth First Search (BFS)

Depth First Search (DFS)

For convenience, we only consider tree structure. This is because the existence of cycle is a big trouble when using the two algorithms.

The time complexity of these two algorithms are both $O(n)$ (obviously).

Some terms of tree: root, leaf, parent, child.

Search

Depth First Search (DFS)

1. We start from the root.

2. Each time we pass a vertex, if it has children, then search them one by one; else return its parent vertex.

3. Return true or false according to the search result.

Search

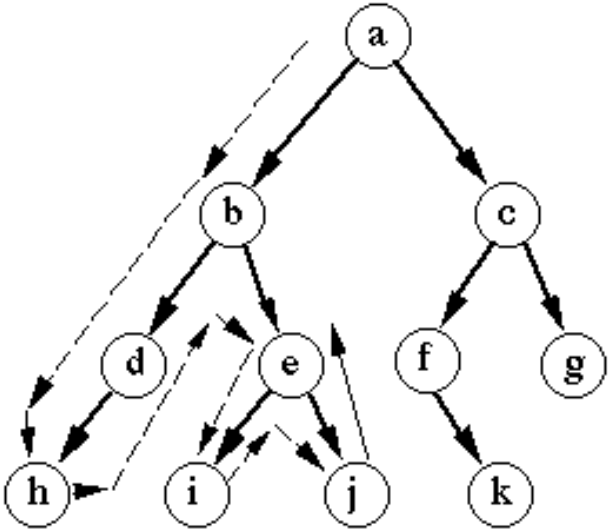
Breadth First Search (BFS)

1. We start from the root. We have a set S of vertices which at first has only the root vertex.

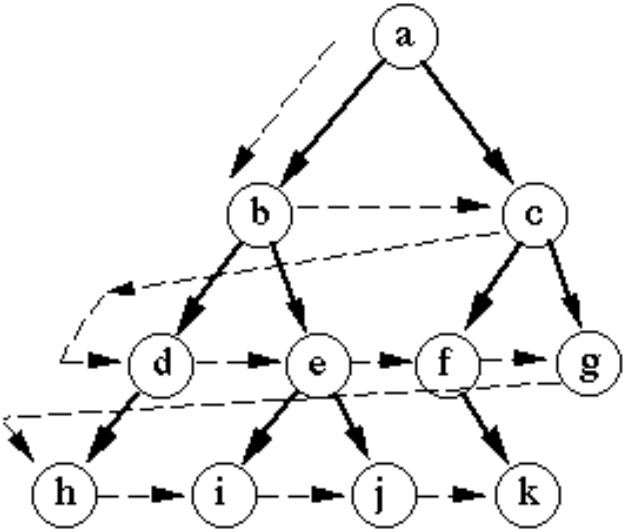
2. Each time if there is some child of vertices in S, we update S. We delete the vertices, and add all the children vertices.

3. Return true or false according to the search result.

Search



Depth-first search



Breadth-first search

Shortest Path

A shortest path from s to t is the path with smallest sum of weight. For convenience, we consider graph with no cycle and all the weights are non-negative. We use Dijsktra algorithm.

1. We maintain a set of vertices S which at first only include our start vertex. Let distance $d(s, v_i) = \infty$ for any $v_i \in V$.

2. Each time there are vertices $\notin S$, we find a vertex v so that:

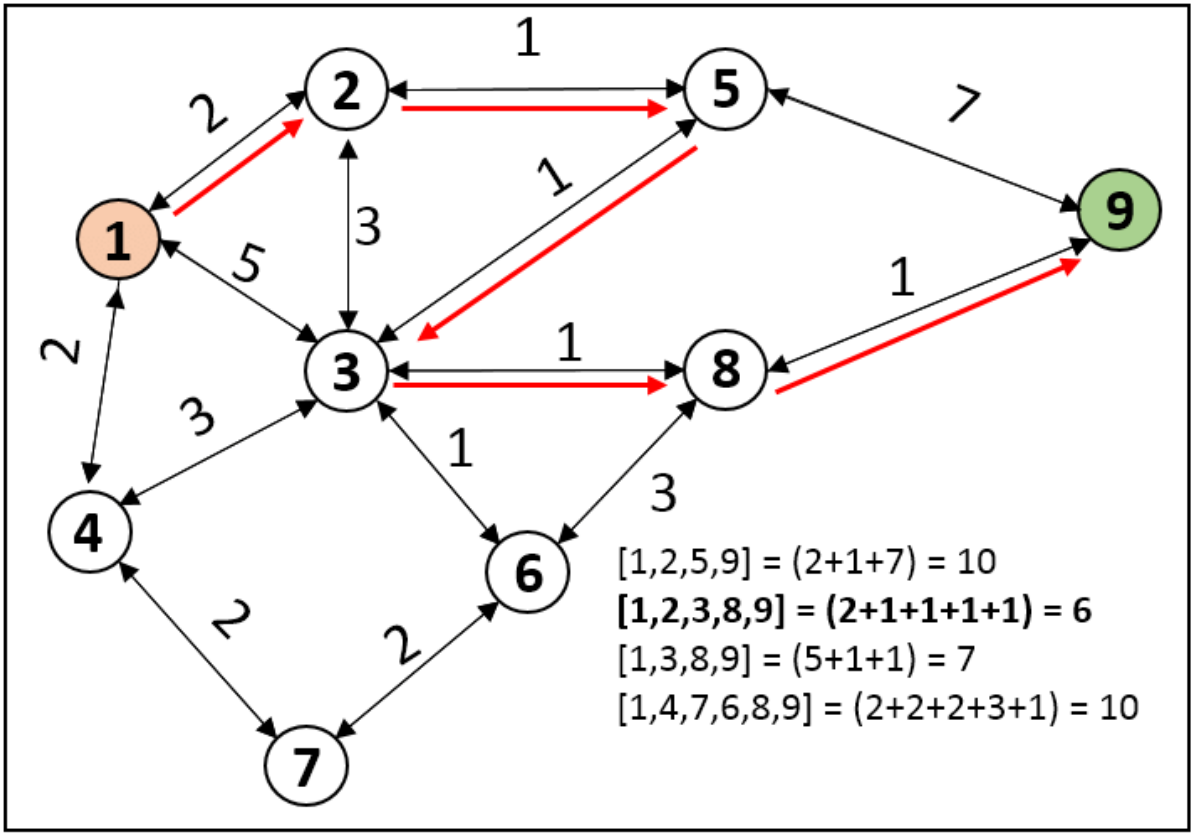
$$\text{I.}v \notin S, \quad \text{II. it has } \min_{v \notin S, v' \in S} \big(d(s, v') + w(v', v)\big)$$

Here $w(v', v)$ is the weight of edge (v', v). We let $d(v, v) = 0$.

3. Add v into S and update $d(s, v)$ as the formula above.

4. Return $d(s, t)$. The time complexity is $O(n \log(n) + m)$ with heap structure.

Shortest Path



Shortest Path

We can write a table to record the result in each step.

Time	$1-2$	$1-3$	$1-4$	$1-5$	$1-6$	$1-7$	$1-8$	$1-9$
1	∞	∞	∞	∞	∞	∞	∞	∞
2	2	∞	∞	∞	∞	∞	∞	∞
3	2	∞	2	∞	∞	∞	∞	∞
4	2	∞	2	3	∞	∞	∞	∞
5	2	4	2	3	∞	∞	∞	∞
6	2	4	2	3	∞	4	∞	∞
7	2	4	2	3	∞	4	5	∞
8	2	4	2	3	6	4	5	∞
9	2	4	2	3	6	4	5	6

Shortest Path

Suppose we get a shortest path $v_1 \to v_2 \to ... \to v_i$ by Dijsktra algorithm. This time we let the weight of each edge change from w_e to w_e^2. Will the path still be (one of) the shortest path?

Hint: This is different from minimum spanning tree (MST).

Shortest Path

Suppose we get a shortest path $s \to v_1 \to ... \to t$ by Dijsktra algorithm. This time we let the weight of each edge change from w_e to w_e^2. Will the path still be (one of) the shortest path?

Hint: This is different from minimum spanning tree (MST).

Answer: No. For example, the original graph has two paths from s to t: the first one is $1+1+1+1=4$ and the second one is 3. So the second one is the shortest path.

But this time, the first one is still $1+1+1+1=4$, but the second one becomes $3^2=9$. So it will not be the shortest path any more.

Note: but if we only let $w'_e = 2w_e$, then it will not change.

We define the height and size of binary tree as follows.

Height: An empty tree is -1, a one-vertex tree is 0. If T has children T_1, T_2, then

$$h(T) = 1 + \max\left(h(T_1), h(T_2)\right)$$

Size: An empty tree is 0, a one-vertex tree is 1. If T has children T_1, T_2, then

$$s(T) = 1 + s(T_1) + s(T_2)$$

Size if the number of vertices. Height is the longest distance between root and one leaf.

Problem: Prove $s(T) \geq h(T) + 1$ for any tree.

Hint: Use induction proof.

Problem: Prove $s(T) \geq h(T) + 1$ for any tree.

Hint: Use induction proof.

Proof.

Base Case: $n = 0, 1$ obviously hold.

Suppose a tree holds, then when add a new vertex to some leaf, then s must $+1$, but h either $+1$ or $+0$. Therefore $s(T) \geq 1 + h(T)$ still holds.

A full binary tree: Each vertex either has 2 children or 0.

Problem: Prove $s(T) \geq 2h(T) + 1$ for any full binary tree.

Hint: Use induction proof.

A full binary tree: Each vertex either has 2 children or 0.

Problem: Prove $s(T) \geq 2h(T) + 1$ for any full binary tree.

Hint: Use induction proof.

Proof.

Base Case: $n = 0, 1$ obviously holds.

Suppose a full tree holds. Then if want to add children for one vertex, we have to add two in one time. In this way, $s(T)$ must $+2$, but $h(T)$ either $+1$ or $+0$. Therefore it still holds.

A graph is $k-$regular if all the vertices have degree k.

Prove: if k is even and $n > k$, then there is a $k-$regular graph with n vertices.

Hint: Show how to draw the graph.

A graph is $k-$regular if all the vertices have degree k.

Prove: if k is even and $n > k$, then there is a $k-$regular graph with n vertices.

Hint: Show how to draw the graph.

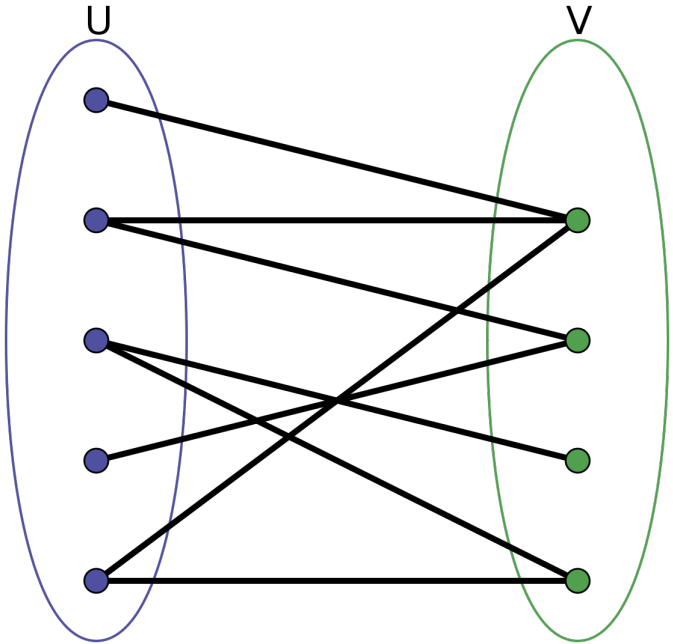
Proof.

If $k = 2$, then we only need to draw a cycle with n vertices.

If $k = 2r$, then we connect vertex i with $i \pm 1, i \pm 2, ..., i \pm r$.

Matching

Bipartite Graph: We can divide V into two parts X and Y. All the edges have one end in X and the other in Y.



Matching

If a cycle with n vertices is a bipartite graph, then what should n satisfy?

Is a $k-$regular graph a bipartite graph? Here, $k \leq 0.5n$.

A tree is a bipartite graph. Does this always hold?

Matching

If a cycle with n vertices is a bipartite graph, then what should n satisfy?

By drawing graph, we know n is even.

Is a $k-$regular graph a bipartite graph? Here, $k \leq 0.5n$.

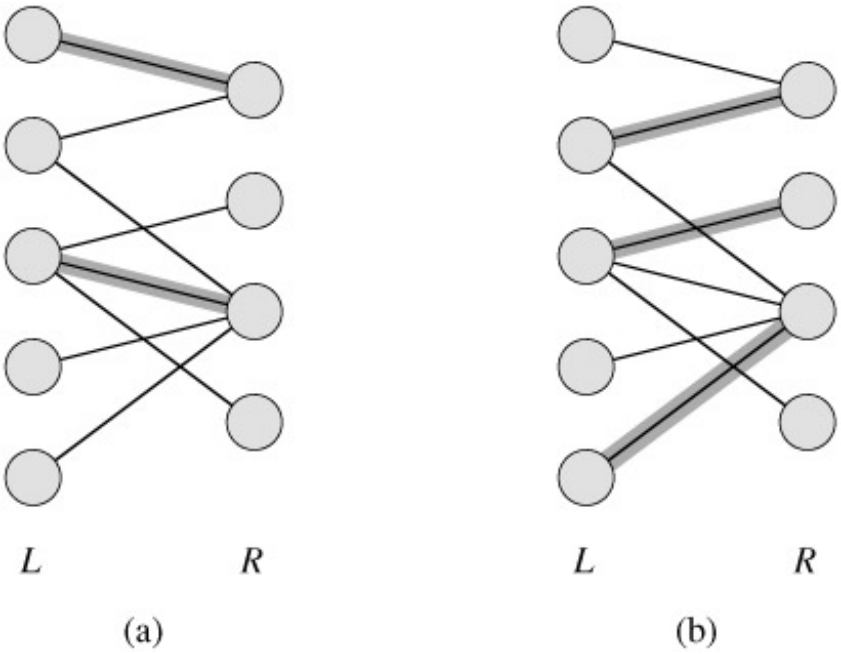
The answer is yes. It's not easy to prove, but we can draw it.

A tree is a bipartite graph. Does this always hold?

Yes. We let odd index layer belong to X, and even index layer belong to Y.

Matching

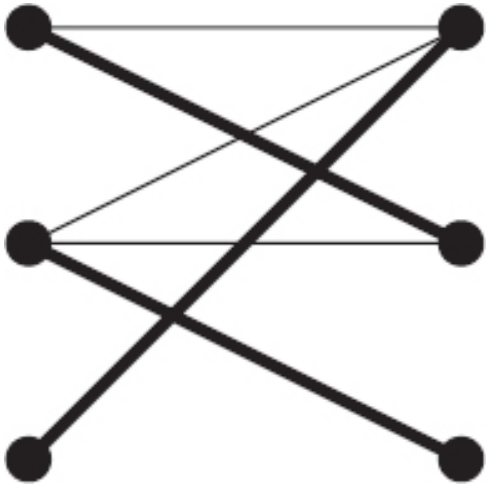
A matching is a set of edges: 1. one vertex can only be used once; 2. the edge in matching belong to E.



Matching

Largest matching: At least all the vertices in left or right part are used.

Perfect matching: All the vertices are used.



Matching

We use neighbor $N(X)$ to denote a set which includes all the vertices connected with some vertex in X.

Hall's Theorem: Suppose a bipartite graph $G = (V, E)$ and $V = X + Y$

$$\text{there is a largest matching for } X \iff \forall X' \subseteq X : |X'| \leq |N(X')|$$

If Both holds for X and Y, then there is a perfect matching.

We can use induction proof to prove it. Try it.

Proof of Hall's Theorem.

Base case : $|X| = 1$ obviously holds.

Suppose $|X| \leq k$ holds. We consider the case $|X| = k + 1$.

1. If there is no set $S \subseteq X$ so that $|N(S)| = |S|$, then we can select arbitrary $x \in X$ and let it match arbitrary neighbor $y \in Y$. In this way we reduce it to $|X| = k$. This is because (N is neighbor in Y and N' is neighbor in $Y - \{y\}$)

$$\forall A \subseteq X - \{x\}, |N'(A)| \geq |N(A)| - 1 \geq (|A| + 1) - 1 = |A|$$

2. If there is some set $S \subseteq X$ so that $|N(S)| = |S|$, then we find the neighbor $N(S) = T$ and delete S, T from X, Y. The remaining part will still be reduced to a smaller case because (N is neighbor in Y and N' is neighbor in $Y - T$)

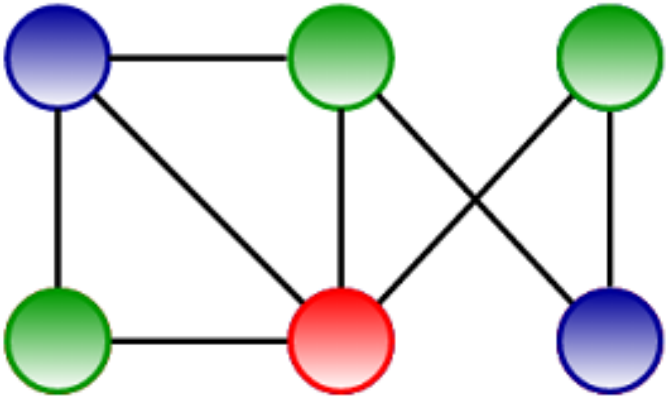
$$\forall A \subseteq X - S, |N(A + S)| \geq |A + S| = |A| + |S|$$

Since we know $|N(S)| = |S|$, we have

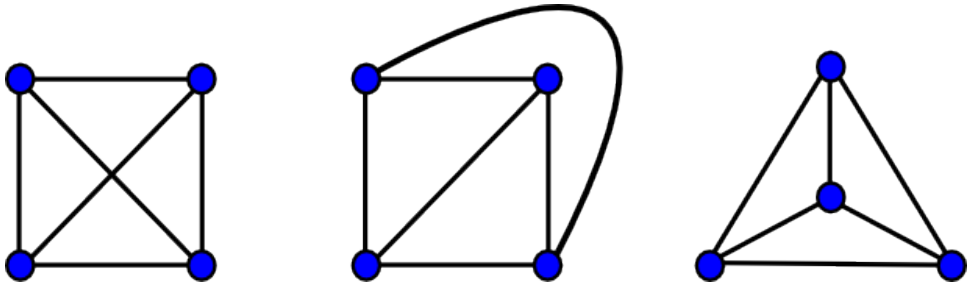
$$|N'(A)| = |N(A + S)| - |N(S)| \geq |A|$$

Color

A graph is $k-$colorable if we can use k different colors to paint the vertices, so that adjacent vertices will never have the same color.

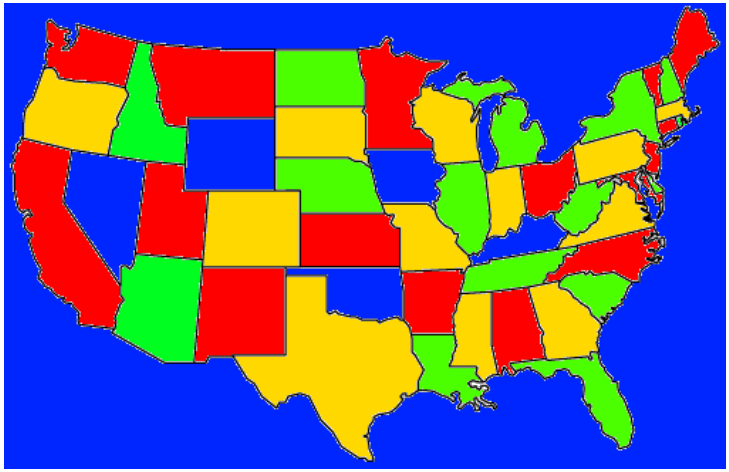


Planar Graph: We can draw the graph in the plane without crossing edges.



Color

4-color Theorem: Any planar graph is 4-colorable.



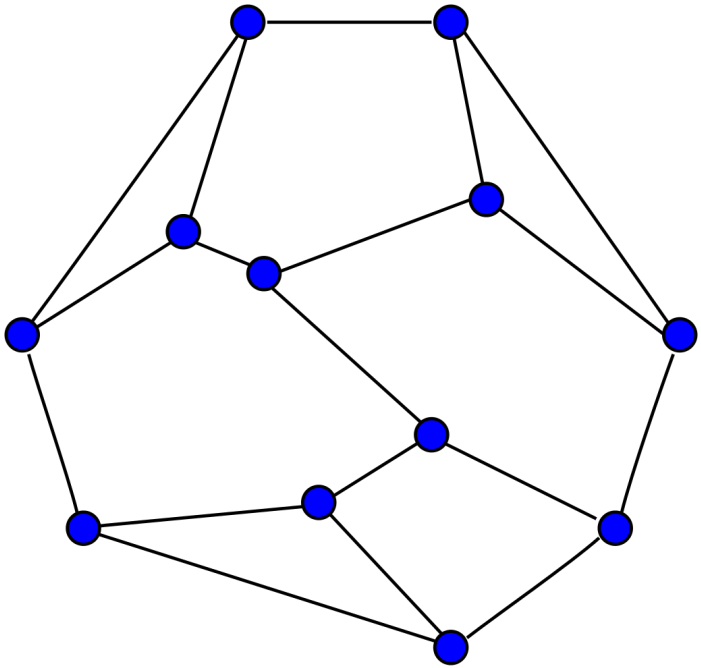
This is hard to prove so we will not introduce.

But it is not hard to prove that any planar graph is 5-colorable.

Color

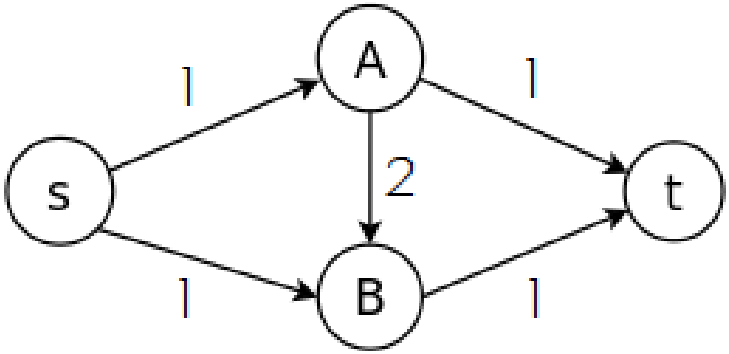
A planar graph has V vertices, E edges and F faces. Then we have the Euler's formula

$$V - F + E = 2$$



Network Flow

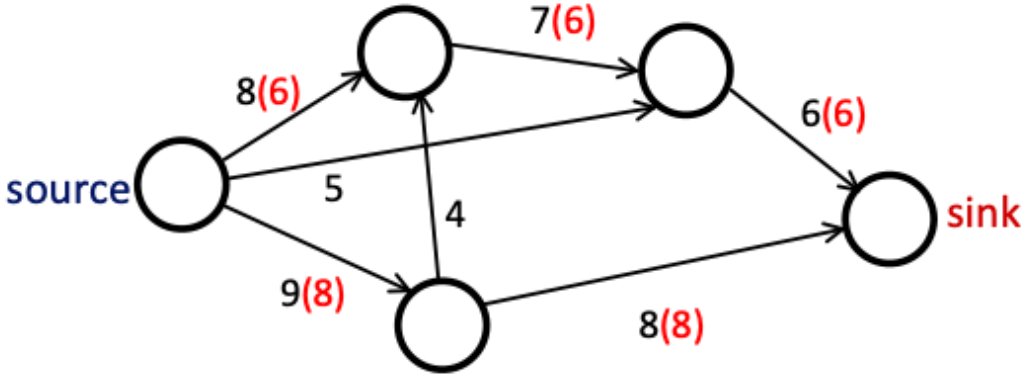
The weight of the edge is the capacity, which is usually integer (we do not consider non-integer case). A flow has source vertex s and sink vertex t. It may have different paths.



For the graph above, the $s - t$ flow can be ? Try it.

What about the max flow?

Network Flow



This is a max $s-t$ flow example and the flow is $6+8=14$. How can we get this?

Use Ford-Fulkerson algorithm.

Network Flow

Before we introduce the Ford-Fulkerson algorithm, we should first know the requirement of the flow.

1. The flow in each edge is no more than the capacity.

$$\forall e \in E, f(e) \leq c(e)$$

2. Symmetry.

$$\forall (u, v) \in E, f(u, v) = -f(v, u)$$

3. Any vertex except s, t must have 0 flow.

$$\forall v \neq s, t, f_{in}(v) = f_{out}(v)$$

Ford-Fulkerson algorithm. Time: $O(EF)$.

1. At first we let all $f(u, v) = 0$.

2. Each time we can find a path P, on which all the edge has capacity > 0, we do the following things.

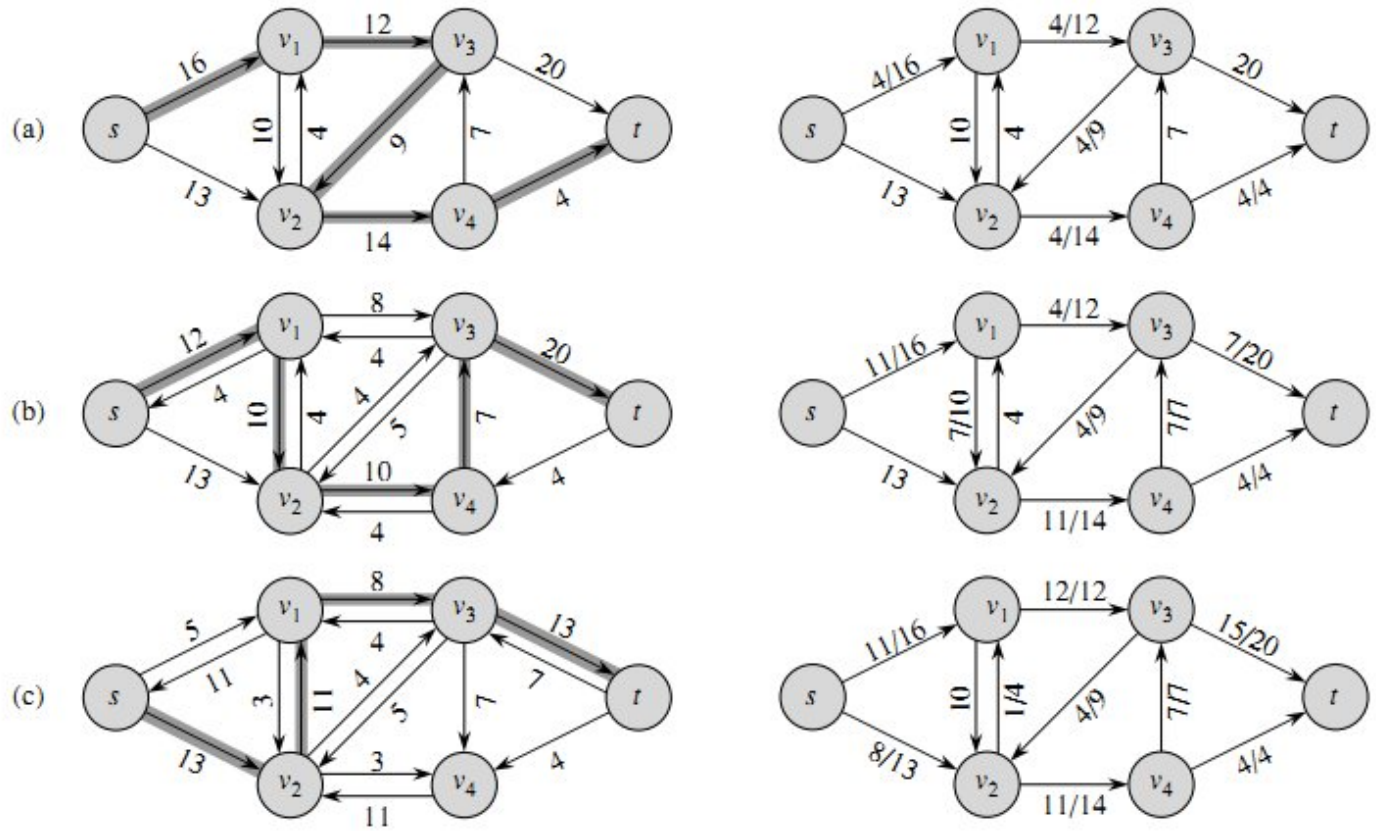
I. Choose the smallest capacity $c_0 = \min_{e \in P}(c(e))$ on this path.

II. Update flow $f(u, v) = f(u, v) + c_0$ for each $(u, v) \in P$.

III. Update capacity $c(u, v) = c(u, v) - c_0$ for each $(u, v) \in P$.

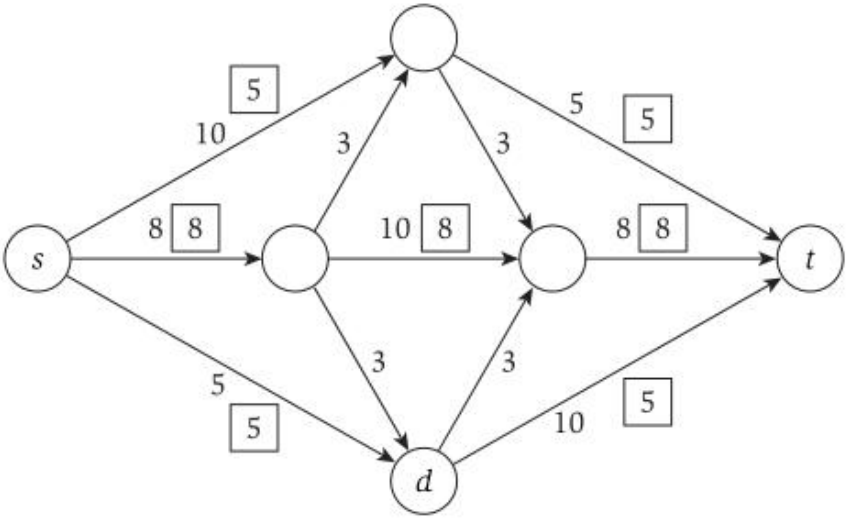
Notice f and c can be negative according to the symmetry.

Network Flow



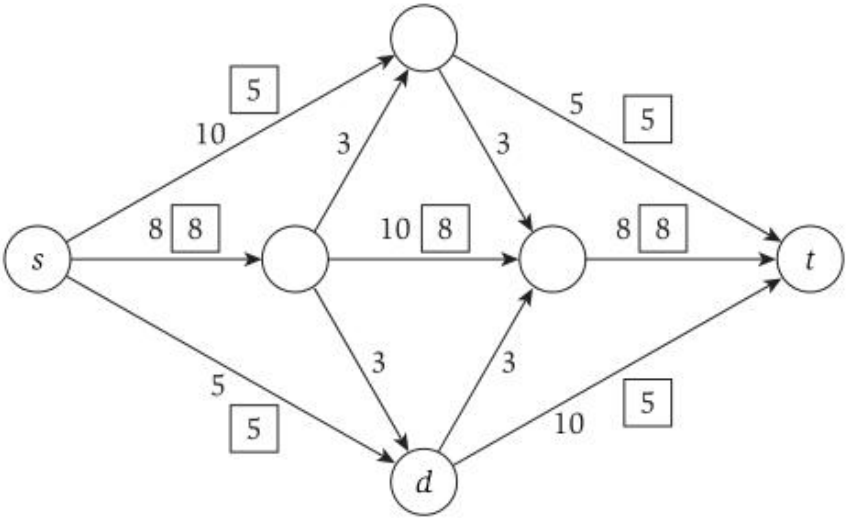
Exercise

Is this a max $s - t$ flow?



Exercise

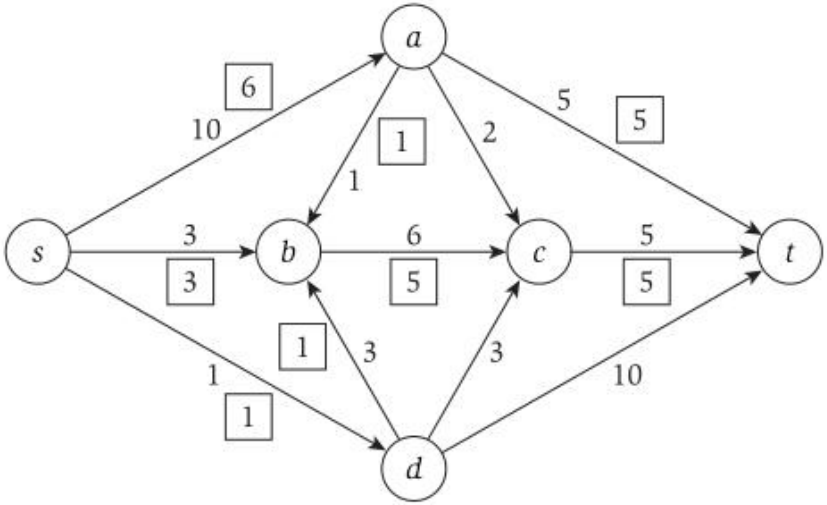
Is this a max $s - t$ flow?



No, the max is 21, but this is 18.

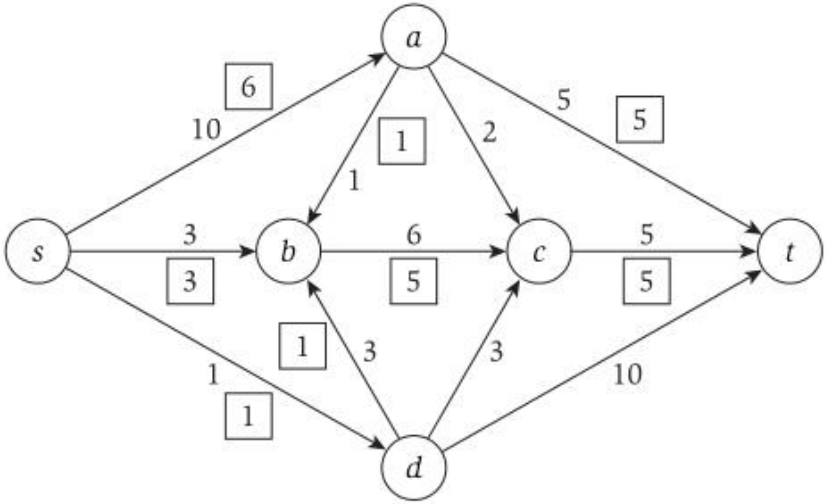
Exercise

Is this a max $s - t$ flow?



Exercise

Is this a max $s - t$ flow?



No, the max is 11, but this is 10.

Exercise

Problem: Suppose we have a large house and there are many walls in it. We have n lights and n light switches. A switch can control one light to turn on or off. We require that, if we let a switch a_i control a light b_j, then we can see the light in the location of this switch.

Design an algorithm to determine whether we can satisfy this requirement. Here, suppose we know for $\forall a_i, b_j$, we know whether we can see the light from the switch.

Hint: Use network flow.

Exercise

Problem: Suppose we have a large house and there are many walls in it. We have n lights and n light switches. A switch can control one light to turn on or off. We require that, if we let a switch a_i control a light b_j, then we can see the light in the location of this switch.

Design an algorithm to determine whether we can satisfy this requirement. Here, suppose we know for $\forall a_i, b_j$, we know whether we can see the light from the switch.

Solution: we should construct a graph. We have a source node s, n switches $a_i \in A$, n lights $b_j \in B$ and a sink node t. For each a_i, we connect (s, a_i) with capacity 1. For each b_j, we connect (b_j, t) with capacity 1. For each a_i, b_j, if we can see the light from the switch, then we connect them with capacity 1.

We can satisfy the requirement if and only if the max $s - t$ flow is n.

Exercise

A hospital needs some blood and it also has some. Does this satisfy the requirement?

Hint: O can supply all, A can supply A and AB, B can supply B and AB, AB can only supply AB.

blood type	supply amount	demand amount
O	50	45
A	36	42
B	11	8
AB	8	3

Exercise

A hospital needs some blood and it also has some. Does this satisfy the requirement?

Hint: O can supply all, A can supply A and AB, B can supply B and AB, AB can only supply AB.

blood type	supply amount	demand amount
O	50	45
A	36	42
B	11	8
AB	8	3

Answer: No. We need 87 O and A, but only supply 86.

Exercise

Problem: We have n TAs for a large course. We have m time slots and each slot need exactly one TA to serve. A TA is available for some time slots and we know this. A TA can serve at most c time slots. Design an algorithm to determine whether we can satisfy this requirement.

Hint: Use network flow.

Exercise

Problem: We have n TAs for a large course. We have m time slots and each slot need exactly one TA to serve. A TA is available for some time slots and we know this. A TA can serve at most c time slots. Design an algorithm to determine whether we can satisfy this requirement.

Hint: Use network flow.

Answer: We construct a graph. We have a source vertex s, n TA a_i, m time slots b_j and a sink vertex t.

For each a_i, we connect (s, a_i) with capacity c. For each b_j, we connect (b_j, t) with capacity 1. For each a_i, b_j, if TA i can serve time slot j, then we connect them with capacity 1.

We can satisfy the requirement if and only if the max $s - t$ flow is m.

Exercise

1. Can you use network flow to determine whether there is a perfect matching in a bipartite graph?

2. What's the time complexity of these algorithms?

Cut

We can cut the vertices in network with two sets A, B. The capacity of $A \to B$ is:

$$C = \sum_{v_i \in A, v_j \in B} c_{ij}$$

The minimum cut is equal to the maximum flow.