Flow network

A directed graph G=(V,E)
c (capacity) : E -> R
f (flow) : E -> R

For every edge (u,v) that belongs to E, f(u,v) <= c(u,v)

For every vertex v except two vertices s and t,
summation (f (u,v)) = summation (f(v,u))
(Incoming Flow = Outgoing Flow)

s - source , t - sink

Total flow of the network :

$$\sum_{(s,u)\in E} f((s,u)) = \sum_{(u,t)\in E} f((u,t))$$

Summation (f(s,u)) >= 0
Summation (f(u,t)) >= 0

Imagine this as a pipe network,
where s - water source
t - water tank
Edges -> pipes (and they have some capacity)
Vertices -> joints

Max flow -> What is the maximum amount of water that can be transported at a time

Ford Fulkerson Algorithm

Residual network - Given a flow network G with flow f, it is the same network G, where capacities become c_f(u,v) = c(u,v) - f(u,v)

Augmenting path - any simple path from s to t in the residual graph, i.e. along the edges whose residual capacity is positive.

1) Find an augmenting path (this can be done using dfs/bfs)
2) Increase the flow by the min capacity among all the edges in that path
3) Find the residual network and keep applying algo till no augmenting path is found

TIme Complexity: O(F*E)

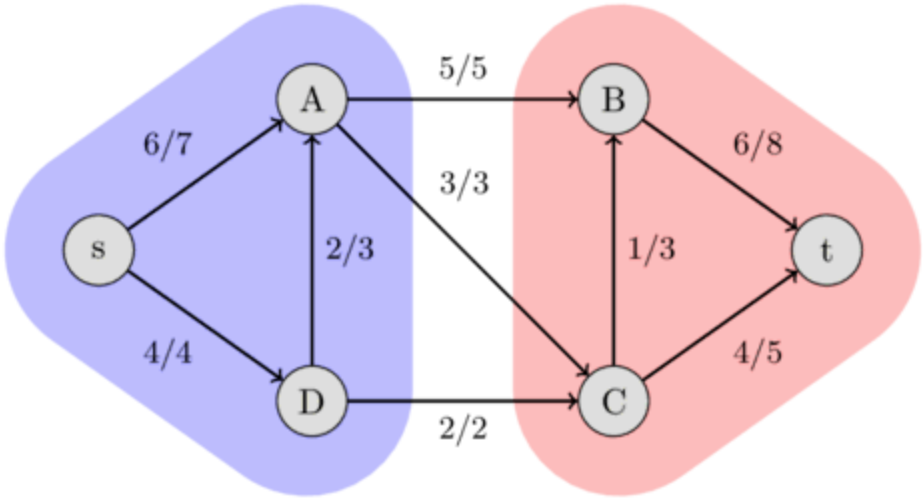
S-T cut - It is a partition of the flow network into two sets S and T, where source s belongs to set S, sink t belongs to set T

Capacity of a cut is sum of all capacities c(u,v) such that (u,v) $\in$ E and u $\in$ S and v $\in$ T

Clearly, Max flow <= c(S,T) (capacity of minimum S-T cut)

Max flow Min cut theorem

(Capacity of the maximum flow has to be equal to the capacity of the minimum cut)



Above image shows that the capacity of the minimum cut S = {s,A,D} and T = {B,C,t} is 5+3+2=10, which is equal to the maximum flow that we found

More statements of this theorem:

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

Finding Minimum Cut

A minimum cut can be found after performing a maximum flow computation using the Ford-Fulkerson method. One possible minimum cut is the following: the set of all vertices that can be reached from s in the residual graph (using edges with positive residual capacity), and the set of all the other vertices. This partition can be easily found using DFS starting at s.

Edmonds Karp algorithm

Only difference with Ford Fulkerson - The augmenting path is the path with shortest length (path with minimum number of edges) . Use BFS instead of DFS for finding such an augmenting path. Rest, the algorithm is same.

For implementation, refer the below link:
https://cp-algorithms.com/graph/edmonds_karp.html

Shortest distance d(s,u) always increases or remains same after each iteration

Critical edge - that edge with minimum capacity in an augmenting path

(u,v) -> becomes critical at some iteration i

d(v) = d(u) + 1

The only possibility of it to become critical again will be when we increase the flow in (v,u)

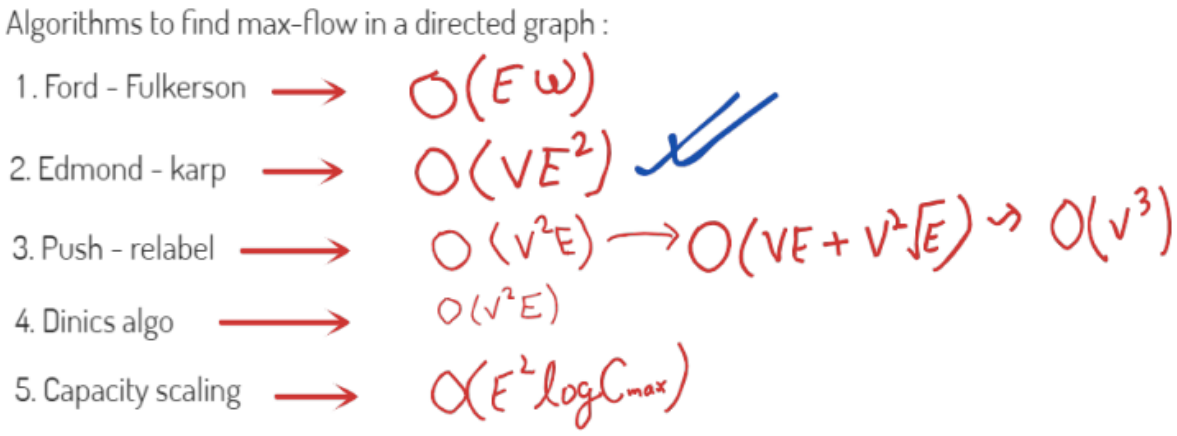
d'(u) = d'(v) + 1
>= d(v) + 1
>= d(u) + 1 + 1
>= d(u) + 2

d(u) <= |V|

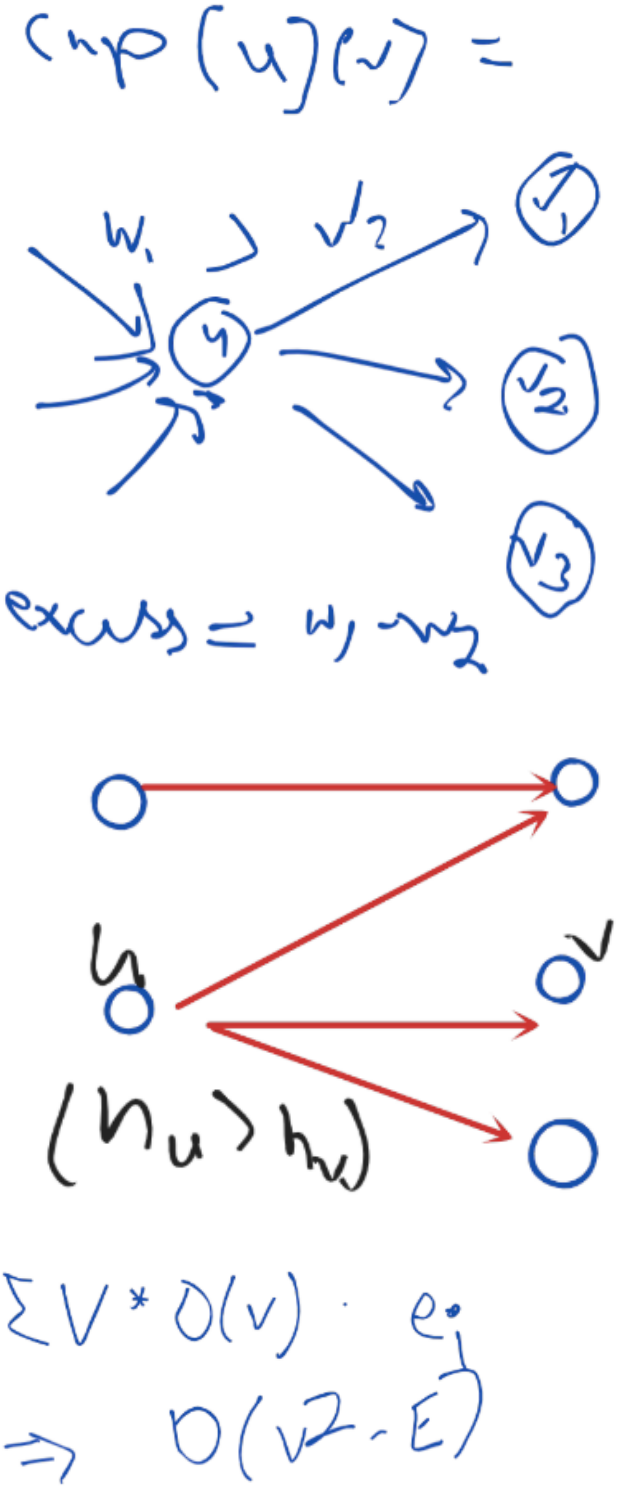
An edge becomes critical atmost (|V| / 2) times

Time Complexity of Edmonds-Karp: O(E*V * E) = O(V*E^2) ≈ O(V^5)

Summary of all algorithms used to find max-flow in a directed graph



Push Relabel Algorithm



For **implementation** of Push relabel algorithm, you can refer the following:
1. https://cp-algorithms.com/graph/push-relabel.html

2. https://cp-algorithms.com/graph/push-relabel-faster.html

Capacity Scaling Algorithm

Dinic Algorithm

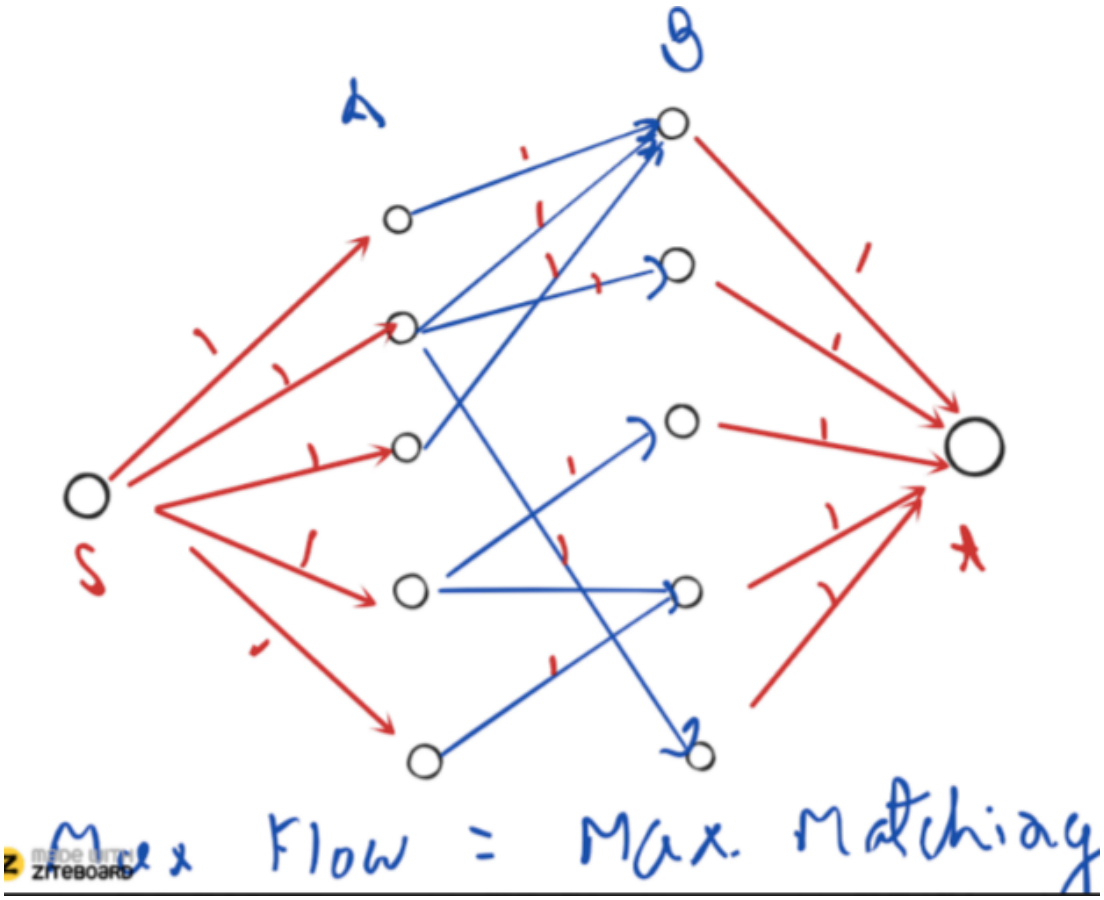
For **understanding** Dinic's algorithm, you can watch this short video: https://youtu.be/M6cm8UeezIl

For **implementation** of Dinic's algorithm, refer: https://cp-algorithms.com/graph/dinic.html

Modeling Problems into Flow Problems

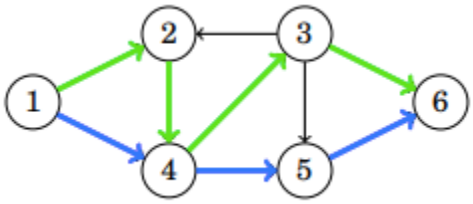
1. Maximum Matching in a bipartite graph

A matching in a graph is a set of edges that do not have any vertices in common.



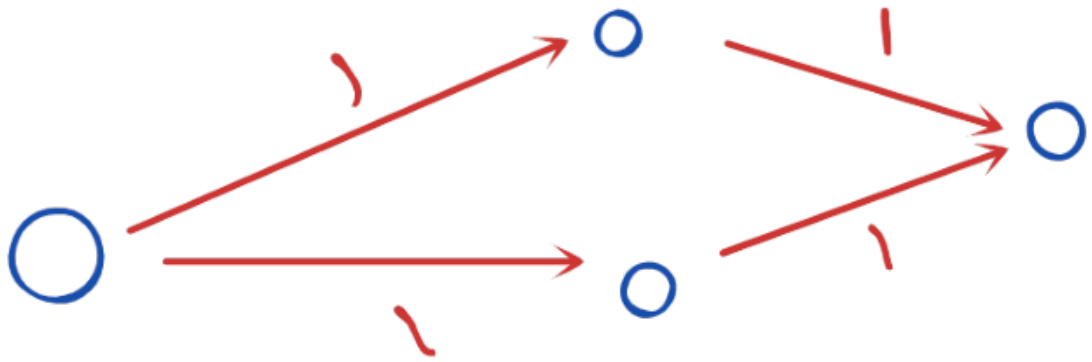
2. Edge-disjoint paths

We need to construct a set of paths (from source node to sink node) such that each edge appears in at most one path.



In above graph, the maximum number of edge-disjoint paths is 2. We can choose the paths $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 6$ and $1 \rightarrow 4 \rightarrow 5 \rightarrow 6$

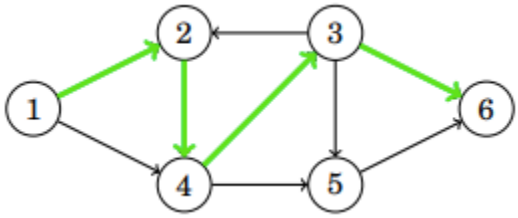
Maximum number of edge-disjoint paths equals the maximum flow of the graph, assuming that the capacity of each edge is one.



After the maximum flow has been constructed, the edge-disjoint paths can be found greedily by following paths from the source to the sink.

3. Node-disjoint paths (or Vertex-Disjoint Paths)

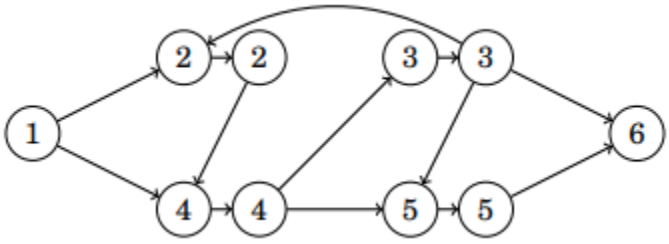
We need to construct a set of paths (from source node to sink node) such that every node, except for the source and sink, may appear in at most one path.



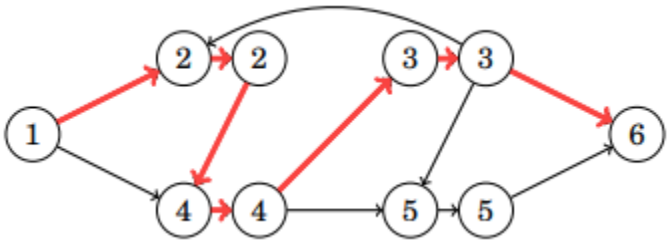
In the above graph, the maximum number of node-disjoint paths is 1.

Since each node can appear in at most one path, we have to limit the flow that goes through the nodes. For this, just divide each node into two nodes such that the first node has the incoming edges of the original node, the second node has the outgoing edges of the original node, and there is a new edge from the first node to the second node.

Like, the above graph becomes:



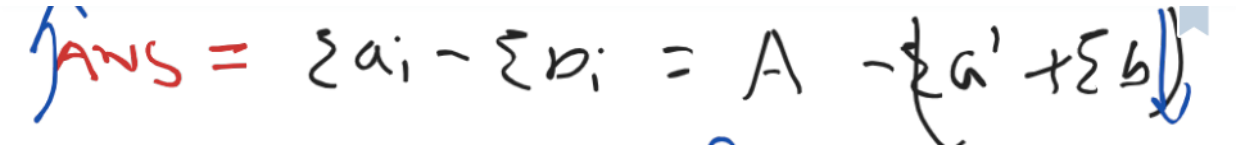
And its max flow would be:

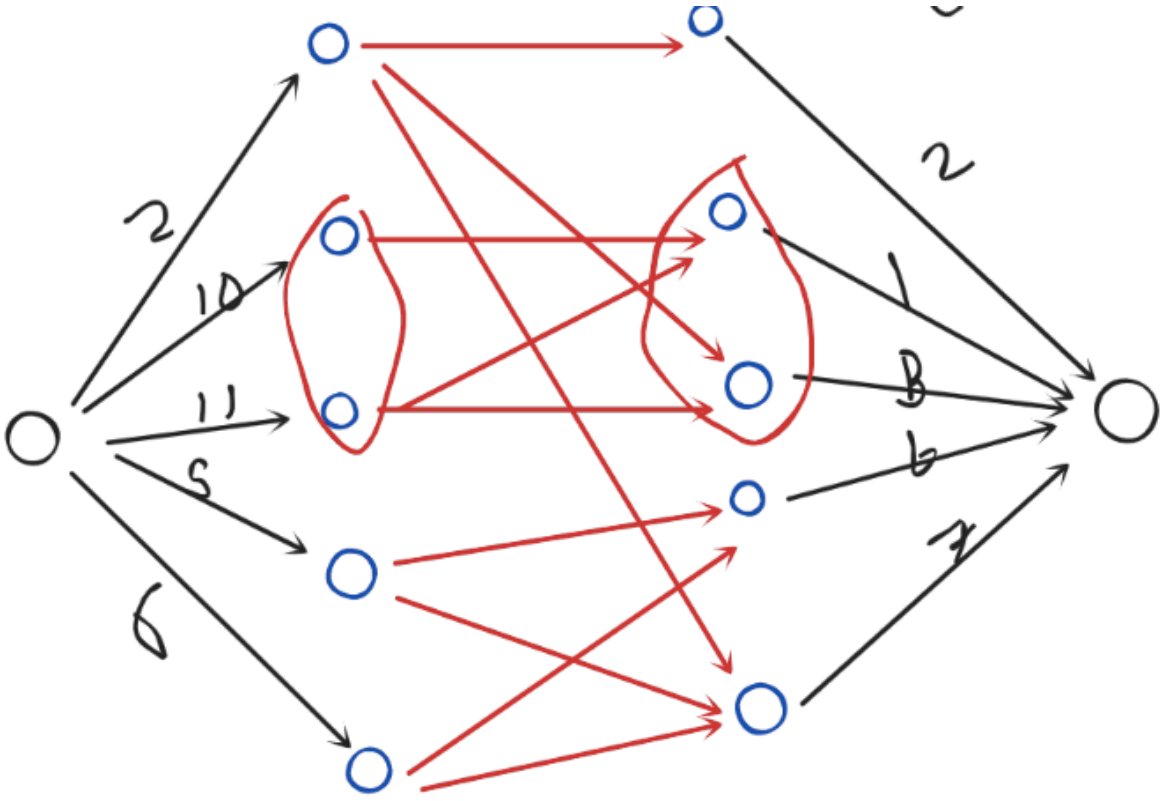


Thus, the maximum number of node-disjoint paths from the source to the sink is 1

4. An example problem

4. "Projects and Instruments". In this problem, we have a set of projects we can do, each with its cost, and a set of instruments (each also having some cost). Each project depends on some instruments, and each instrument can be used any number of times. We have to choose a subset of projects and a subset of instruments so that if a project is chosen, all instruments that this project depends on are also chosen, and we have to maximize the difference between the sum of costs of chosen projects and the sum of costs of chosen instruments.





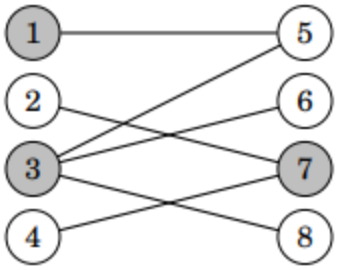
Also try this **similar problem on codeforces:**
https://codeforces.com/problemset/problem/1082/G

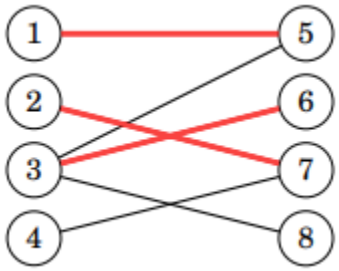
5. Minimum Vertex Cover of a bipartite graph

A Node cover (Vertex Cover) of a graph is a minimum set of nodes such that each edge of the graph has at least one endpoint in the set.

Theorem 1 - K¨onig's Theorem.

Let G(V, E) be a bipartite graph. The size of a maximum matching in G equals the size of a minimum vertex cover of G





Example: In above graph, size of minimum vertex cover = 3 = Size of Maximum Matching

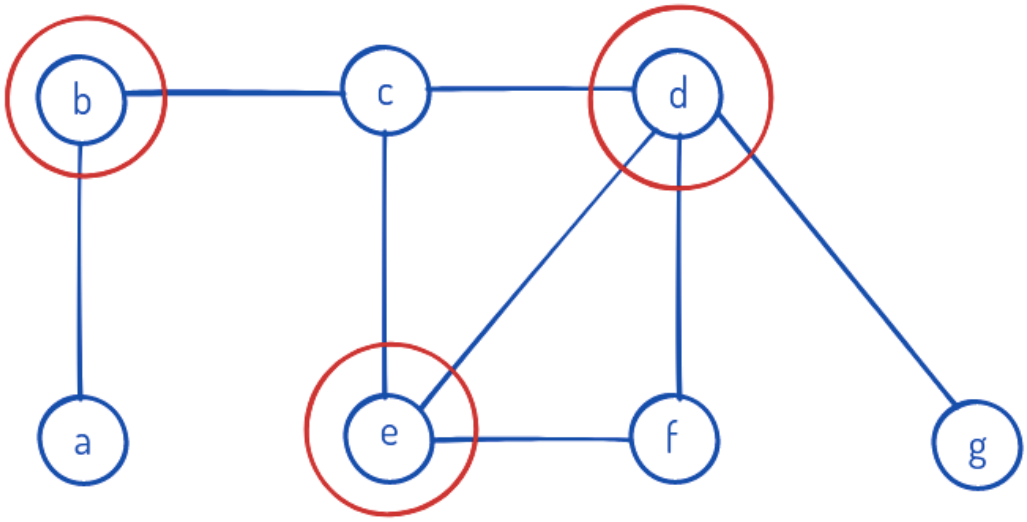
Some more practice problems on Max Flow & Min Cut & Its Applications

1. https://cses.fi/problemset/task/1694
2. https://cses.fi/problemset/task/1695/
3. https://www.hackerearth.com/practice/algorithms/graphs/maximum-flow/practice-problems/algorithm/easy-game/
2. https://www.codechef.com/MAY20A/problems/NRWP
3. https://codeforces.com/problemset/problem/1016/D
4. https://codeforces.com/problemset/problem/498/C
5. https://cses.fi/problemset/task/1696

Minimum Vertex Cover for general graph

A Node cover (Vertex Cover) of a graph is a minimum set of nodes such that each edge of the graph has at least one endpoint in the set.

Example:
See the below graph



Here, minimum Vertex Cover = {b, e, d}

Brute Force Approach:
Iterate through all subsets of the vertices V and check, if that subset is a vertex cover (by iterating through all edges).

Time complexity: O (2^V * E)

An approximation algorithm:

Ans = {}
Repeat until E is not empty:
    Take any arbitrary edge (u,v) in set E
    Insert u, v in Ans
    Remove all edges from E, which have u or v as one of its endpoint

This is an approximate algorithm. Its answer would be not more than 2 times the correct answer.

A better algorithm for exact answer:

We can use binary search on the size of vertex cover.

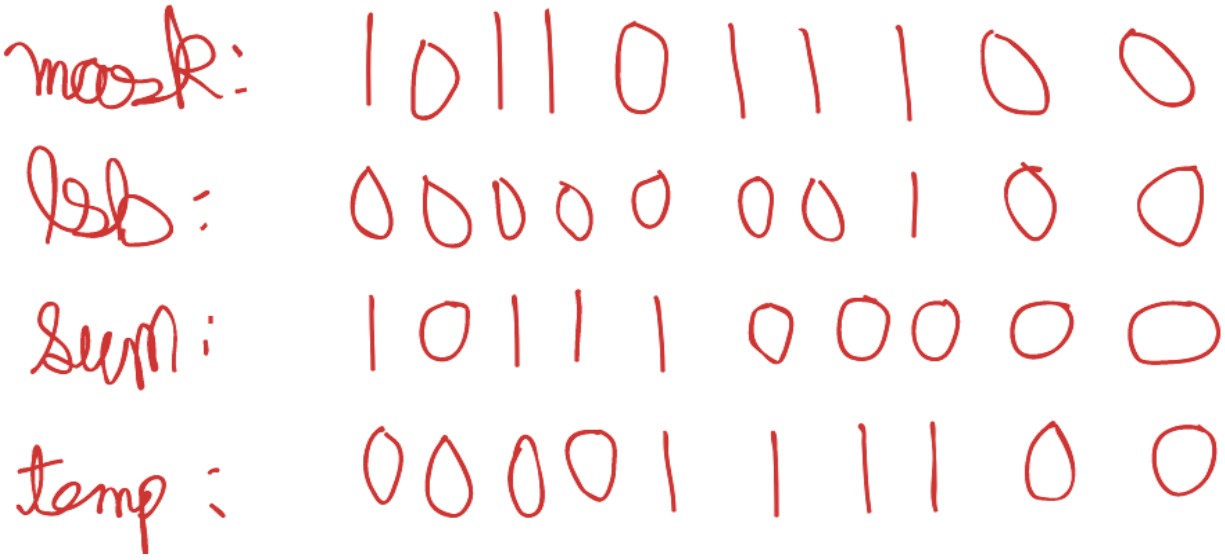
Let us say, mid = k

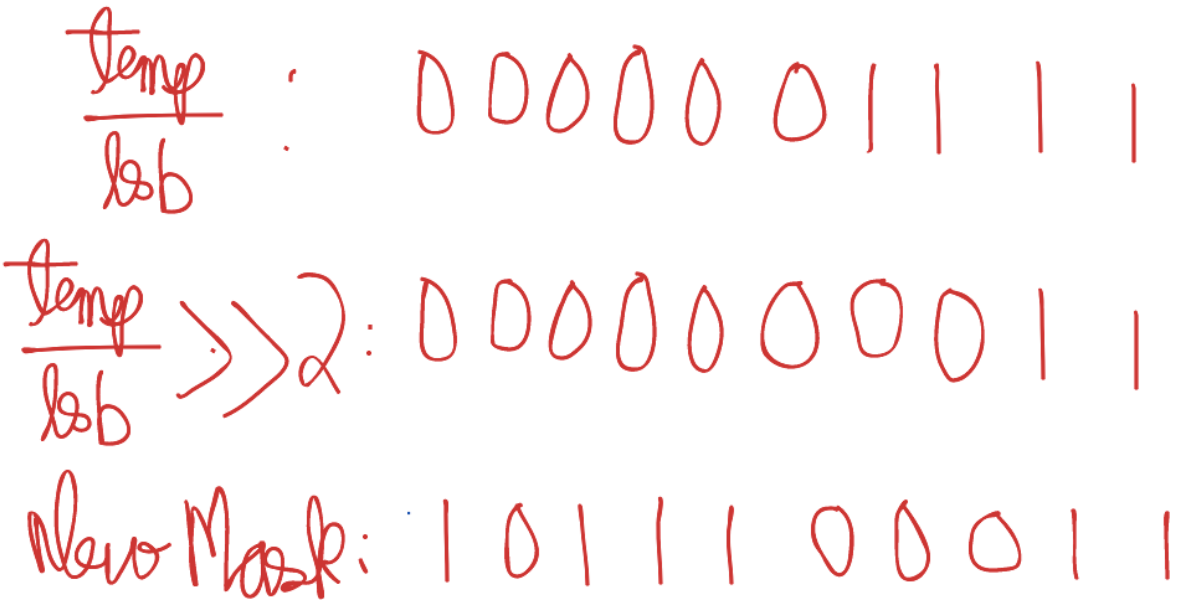
We need to iterate through all the subsets of vertices of size k.

```
int mask=(1<<k)-1;

while(mask < (1<<n) )
{
   int lsb = mask & (-mask);
   int sum = lsb + mask;
// move the leftmost bit of rightmost cluster of 1's , 1 place left
   int temp = mask ^ sum;
   mask =(( temp / lsb ) >> 2) | sum
// move remaining 1's of the rightmost cluster of 1's to the rightmost
// position
}
```

Example of how to move to next greater submask of size 6, starting with a given mask:





Time complexity of the binary search algorithm for finding minimum Vertex Cover:

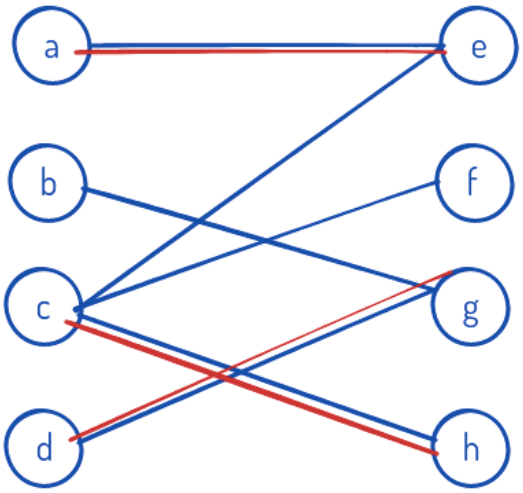
$O (E * ({}^{V}C_{V/2} + {}^{V}C_{V/4} + {}^{V}C_{V/8} + \ldots.))$

Maximum Independent Set

Maximum set of vertices such that no 2 vertices in the set are connected by an edge.

Maximum Independent Set = Complement of Vertex Cover

For a bipartite graph, we know that size of Vertex Cover = size of Maximum Matching. (Konig's Theorem). Using this, we can obtain MIS (Maximum Independent Set)



vertex cover = 3

MIS = 5

{b, e, f, h, d}

Further Resources (Optional)

If you want to understand the various algorithms in more detail and know the various proofs for time complexity, etc. and knowing more of their applications , you may refer some of the below slides: (Optional)

1. https://cseweb.ucsd.edu/classes/sp11/cse202-a/lecture8-final.pdf
2. https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI.pdf
3. https://www.cs.cmu.edu/~avrim/451f13/lectures/lect1010.pdf
4. https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowII.pdf