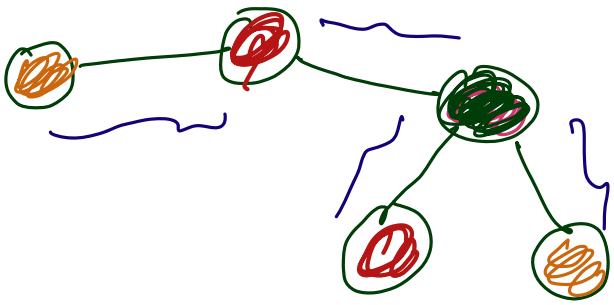
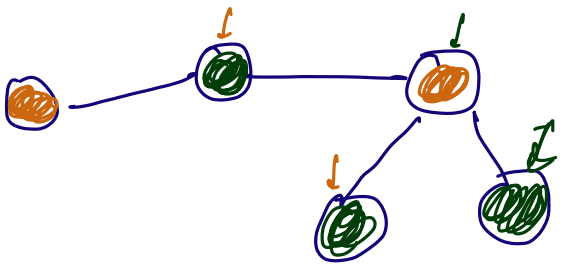
GRAPHS -3

Graph Colouring

Colouring a graph in such a way that no 2 adjacent nodes have the same colour



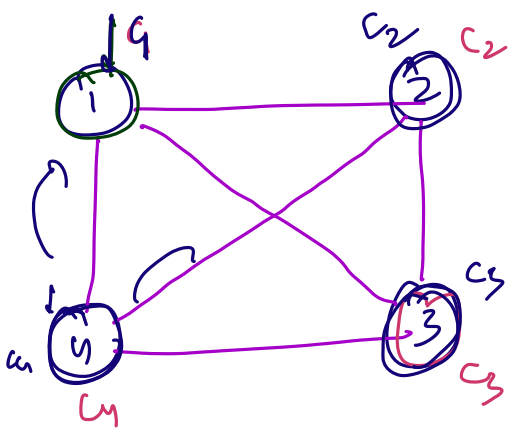
colours = 3



colours = 2

Chromatic Number: The min no. of colours to colour a graph

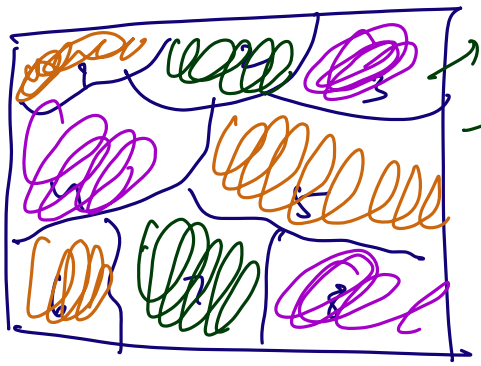
V = 4



(V-1)

V-1 = 3

Map Colouring



Map

planar graph / Map

#colors = 3

History of Graph Colouring

1) 1852 ⇒ Francis Guthrie
   We don't need more than 4 colours to colour any map

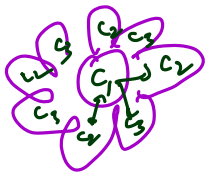
2) 1879, Alfred Kempe, he proved the 4-colour theorem [Accepted of (oyce)

3) 1890, Heawood disproved the above theorem & proposed a 5-colour theorem.

4) 1976, Kenneth & Wolfgand proved the 4-colour theorem using simulations

→ Fully connected graph of 5 vertices

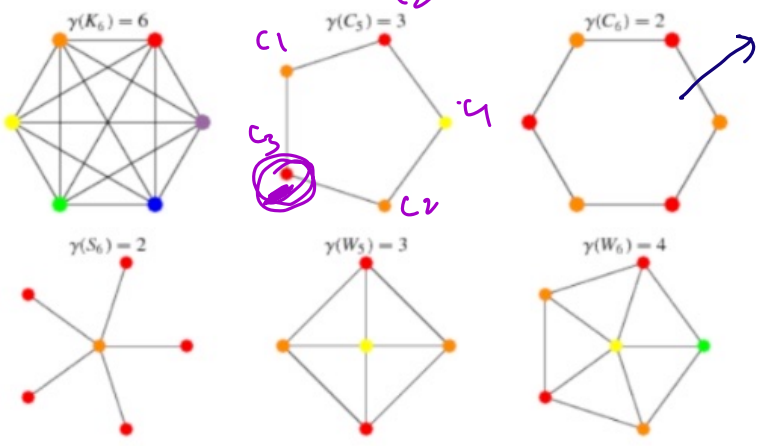
2) 5 colours

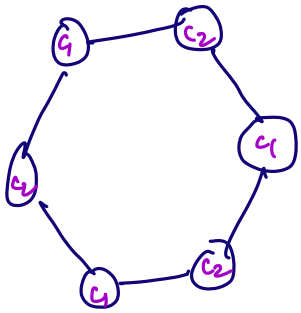


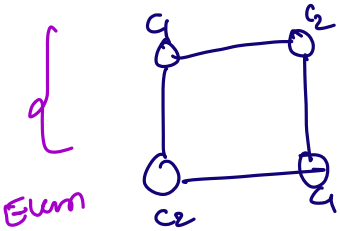
3 colour

Question: Find chromatic number of a graph

→ odd length cycle rings ⇒ 3



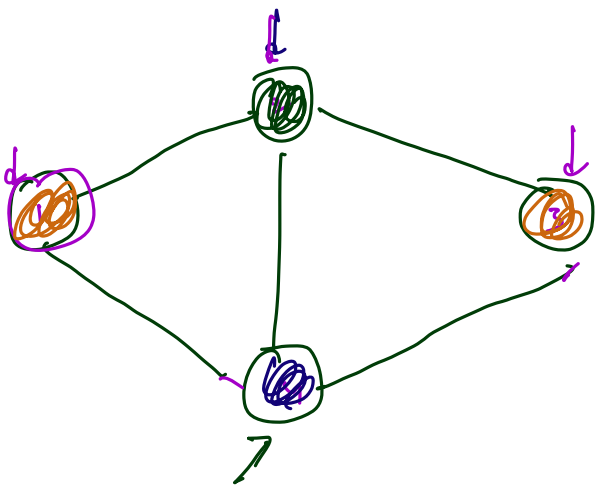


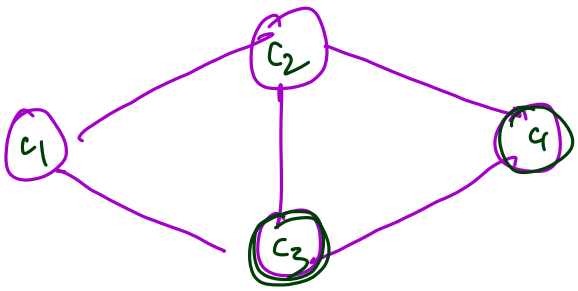


Even no of vertices

Approach1 :

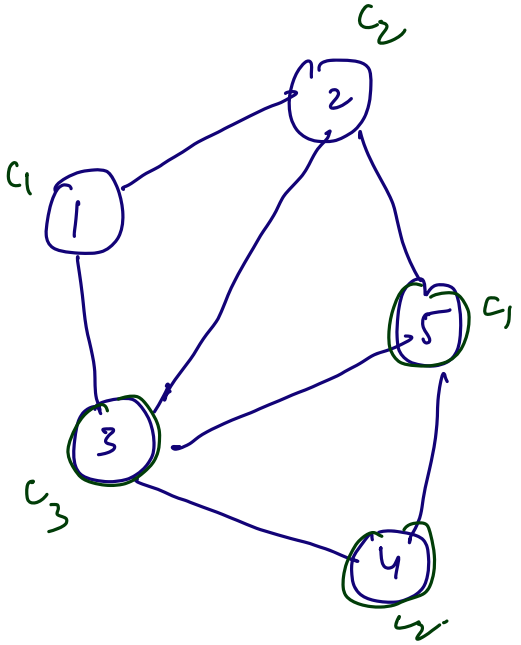
colors = 3





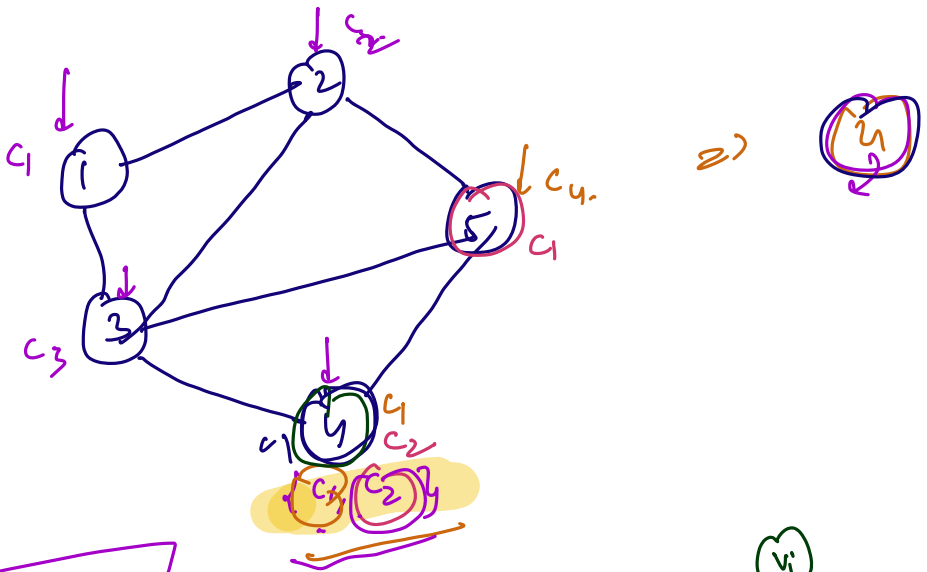
Ans = 3

Ex:



#colors = 3

$\{C_1, C_2, C_3\}$



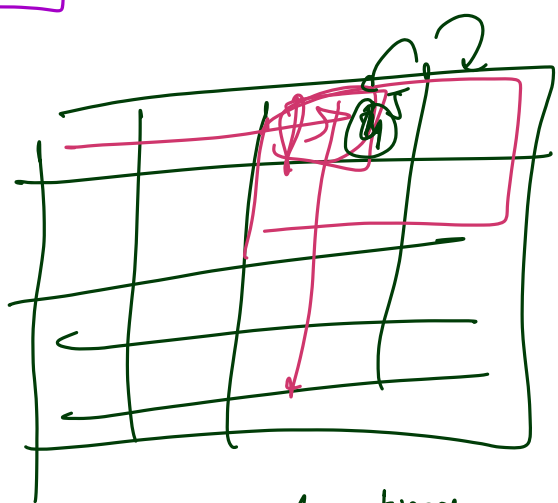
Back tracking

V_i

$\{C_1, C_2, C_3 \ldots C_k\}$

1, 2, 3 ... 9

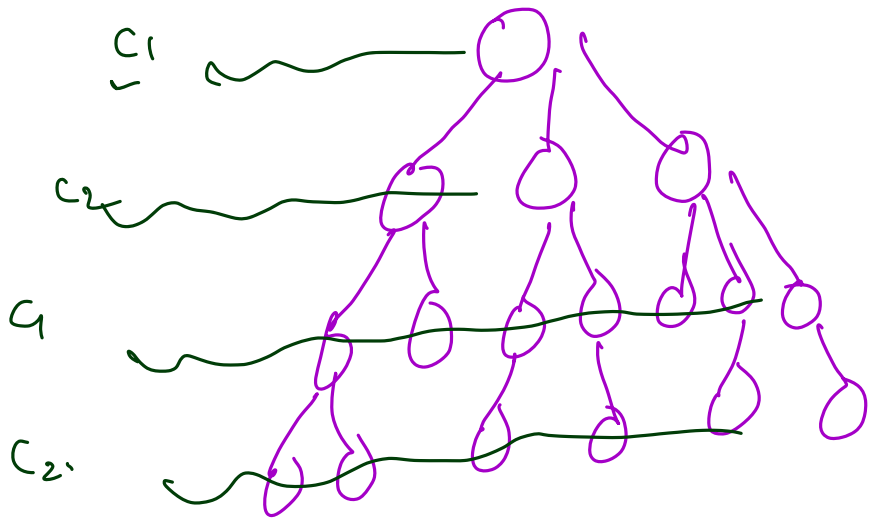
Sudoku problem



There is no polynomial time solution to find the chromatic number.

Question : Chromatic number of k ary tree

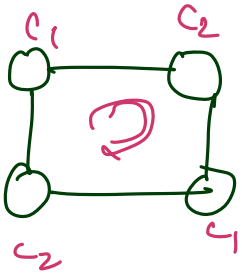
$K = 3$



graph can be coloured using 2 - colours

↓

Bipartite Graph

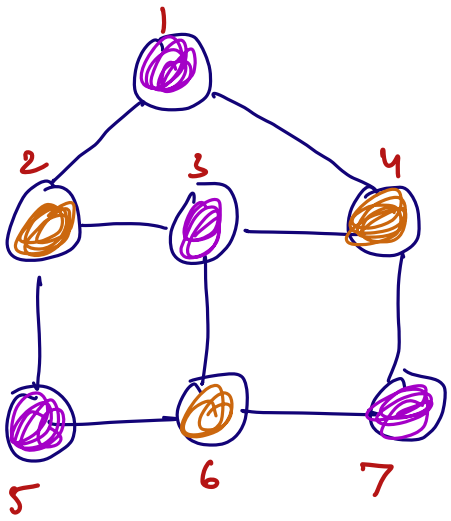


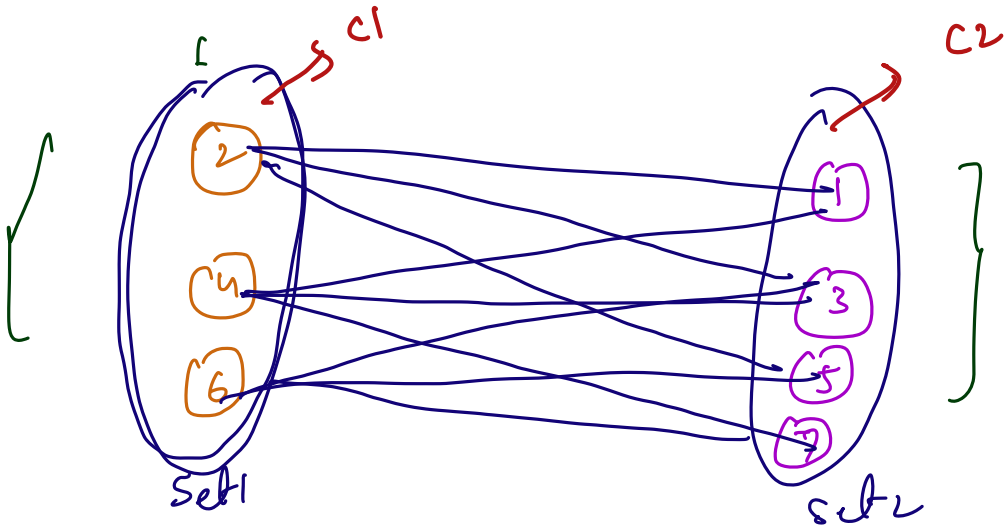
Bipartite Graphs

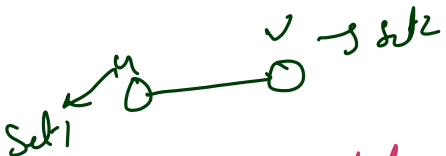
→ { Chromatic number is 2 }

→ If we are able to divide the nodes of a graph into 2 sets such that no 2 nodes of same set will have an edge between them

But, there can be an edge between nodes of different sets



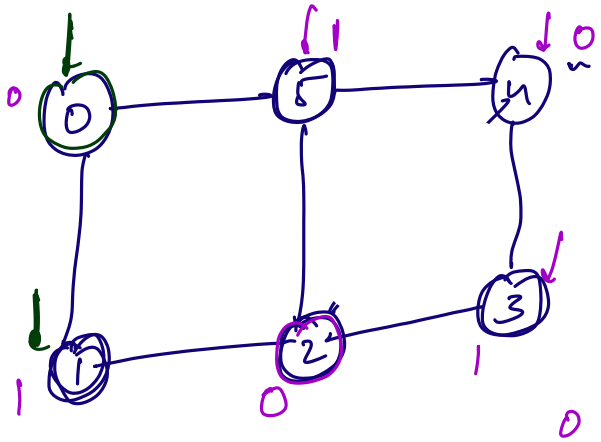




Question: Check if a graph is bipartite

Colour: {0, 1}

C1 C2



d {2, 3, 5}

return rah

	0	1	2	3	4	5
colour:	0	1	0	1	0	-1

Check if Bipartite Graph

```
color[N] = {-1}
bool checkBipartite(int v, int clr){
    color[v] = clr;
    for(i in adj[v]){
        if(color[i] == -1){
            if(!checkBipartite(i, 1 - color[v]))
                return false;
        }
        else if(color[v] == color[i])
            return false;
    }
return true;
}
ans = checkBipartite(0, 0)
```

DFS

visit all neighbours

0 → 1
1 → 0
1 − color

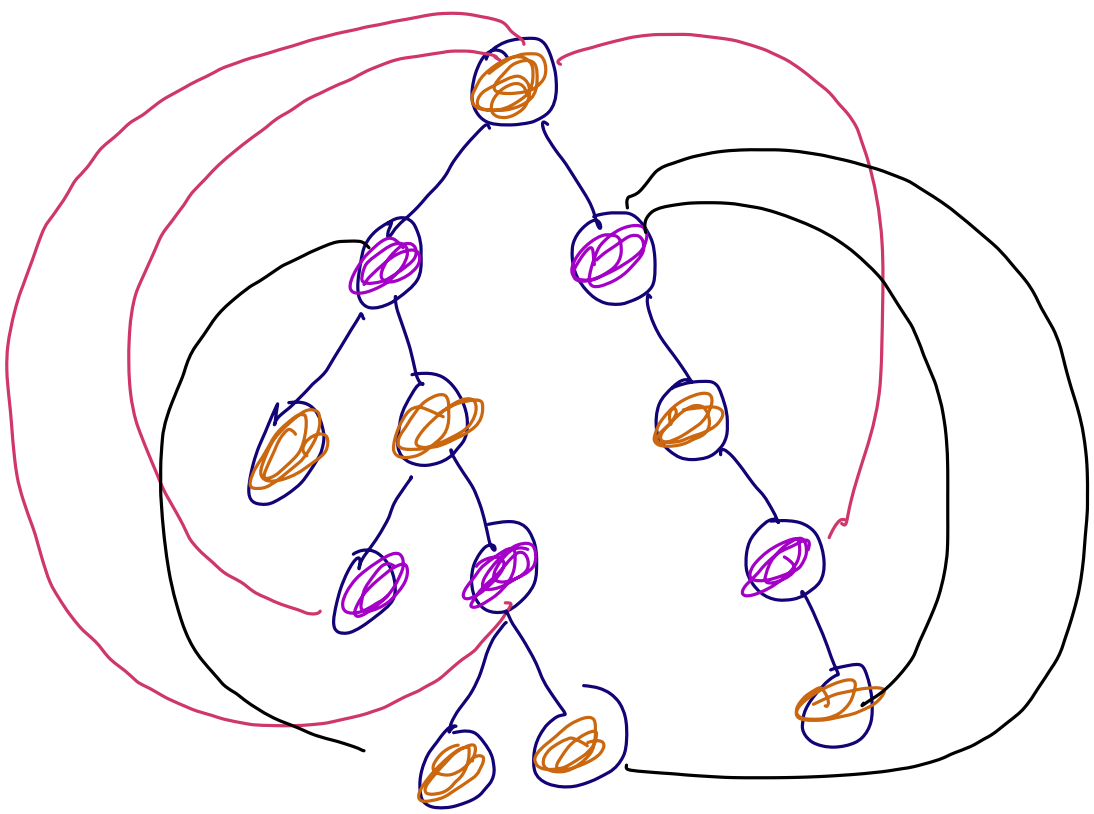
color = 0 → (1)
color = 1 → (0)

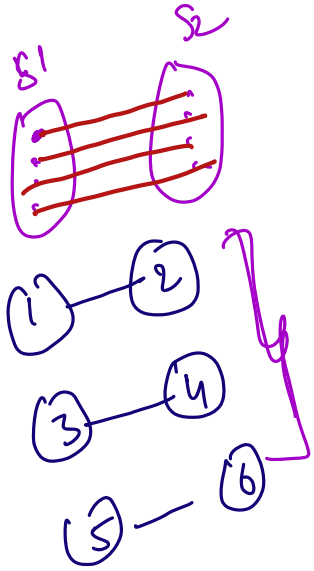
src → colour

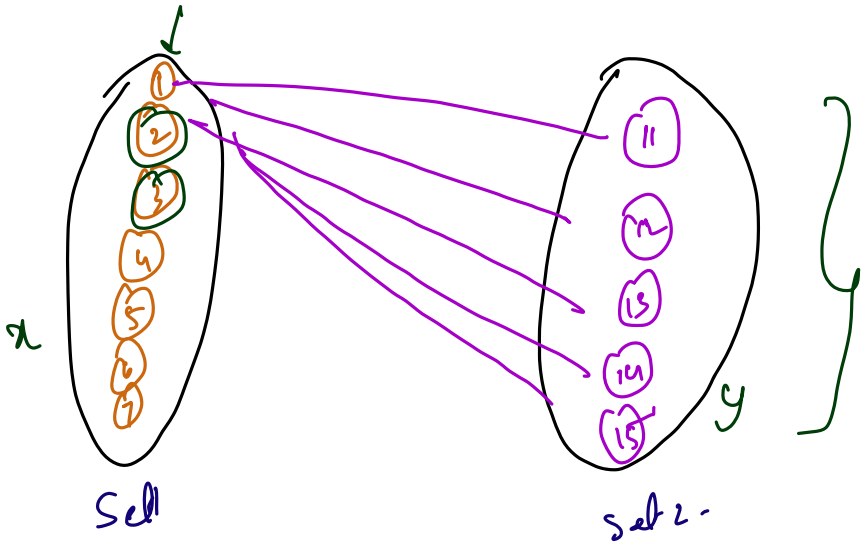
T.C: O(V+E)

BFS

Question: Map no. of edges that can be added to free such that it remains bipartite.







Max No. of edges in Bipartite graph

x * y Edges

xy - # Edges

Tree E = V - 1

Ans = x * y - (V - 1)

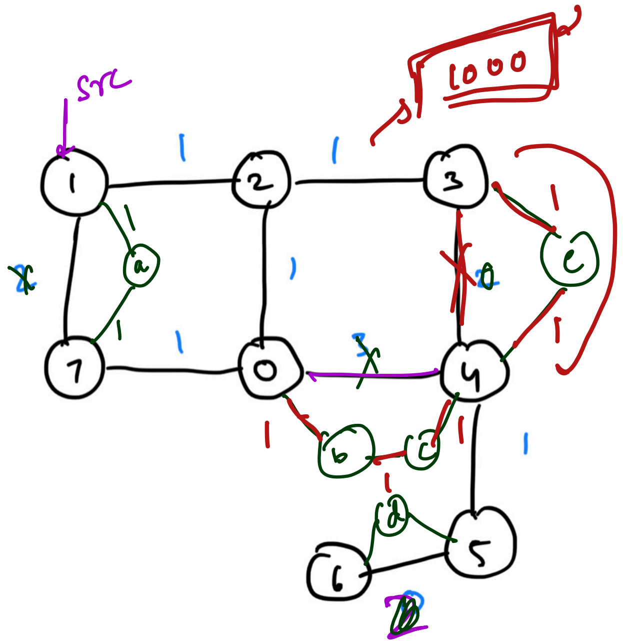
Single Source Shortest Path (SSSP)

BFS: Unweighted Graph

Weighted Graph?

Approach: Convert to BFS

W ≤ 10



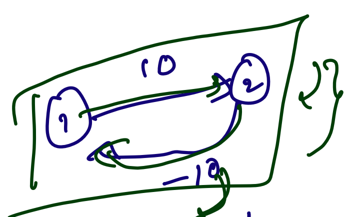
All edges as Same Weight

SSSP:

1) BFS
2) Dijkstra's Algorithm
3) Bellman Ford Algo

→ +ve edge weights
→ Weighted Graph

+ve & -ve edge weights.

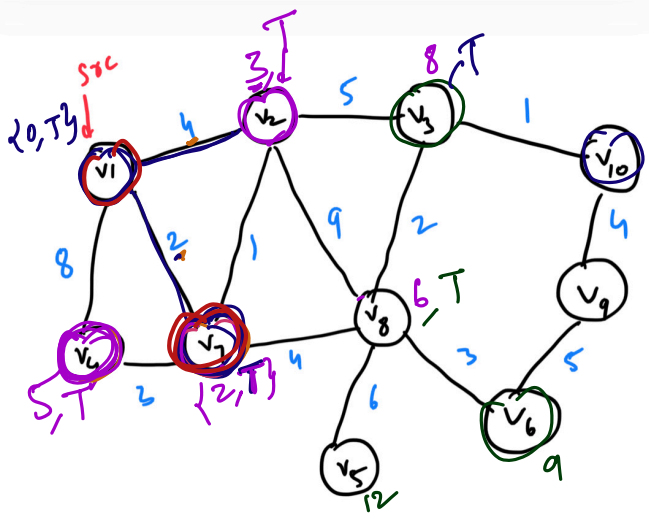


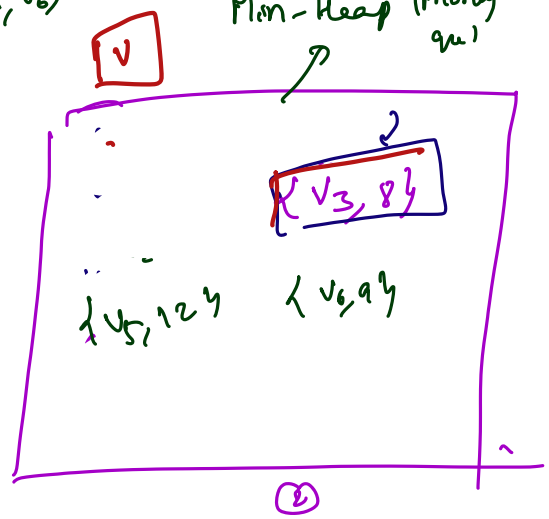
problem → +ve edge → -ve

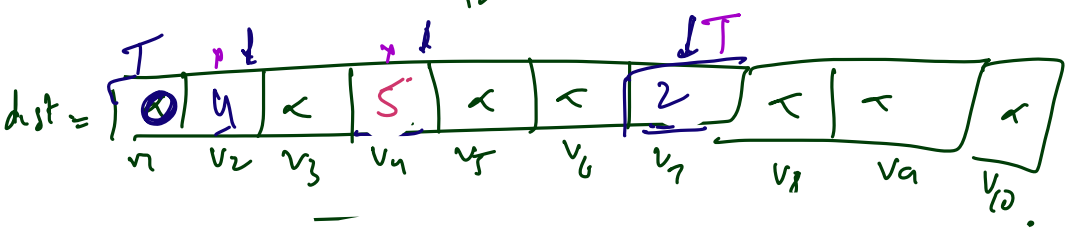
Dijkstra's Algorithm

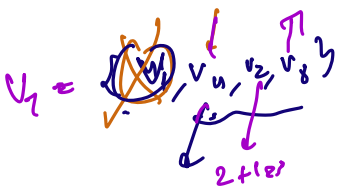
→ Works for Directed / undirected Graphs

→ Edge Weights are positive





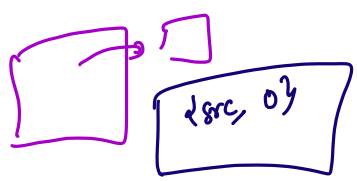


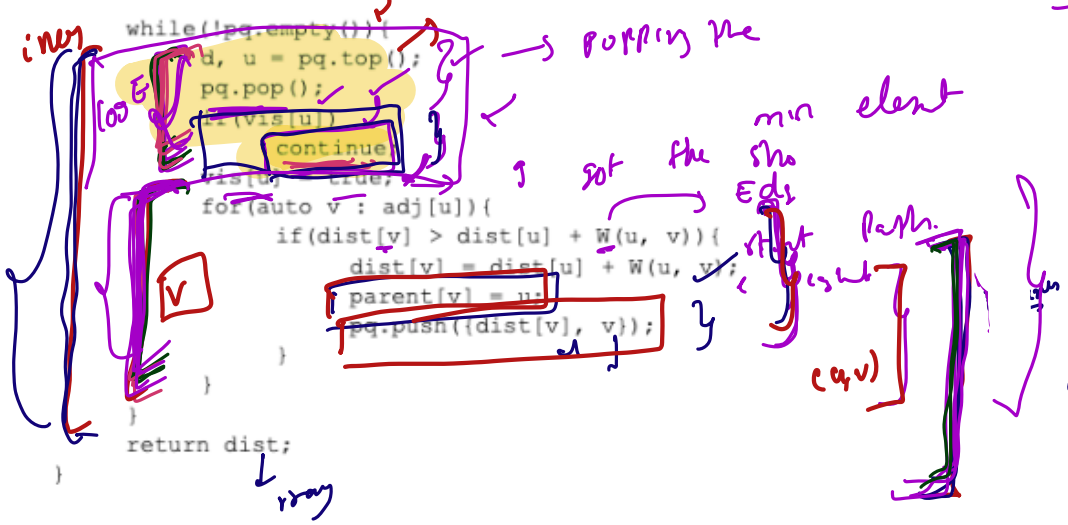


{v1, v7}

```
void dijkstra(int N, adj[], int src){
    dist[N] = INF;
    vis[N] = False;
    dist[src] = 0;
    minHeap(pair<int, int>) pq;
    pq.push({0, src});

    while(!pq.empty()){
        d, u = pq.top();
        pq.pop();
        if(vis[u])
            continue;
        vis[u] = true;
        for(auto v : adj[u]){
            if(dist[v] > dist[u] + W(u, v)){
                dist[v] = dist[u] + W(u, v);
                parent[v] = u;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}
```





Loop → E times!

V times → $\log E + (N_i)$

$N_1 + N_2 + N_3 + N_4 \cdots N_V$ = E

$V \times \log E + (N_1 + N_2 + N_3 \cdots N_V)$

E

$V \log E + E$ ①

(E-V) iteration → $\log E$

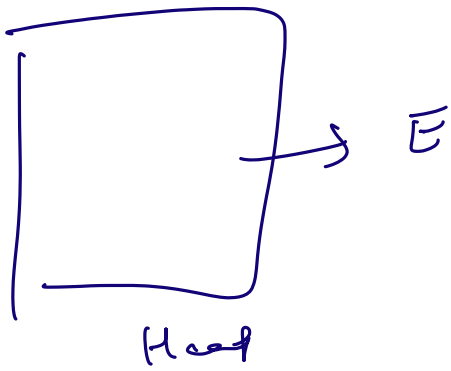
$(E-V)\log E$ → ②

$V \log E + E + E \log E - V \log E$

$E + E \log E \Rightarrow E \log E$

$O(E \log E)$

Optimization



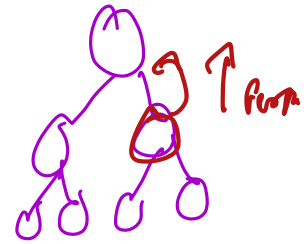
{ (v1, 10), (v1, 9), (v1, 4) }

1 entry per vertex

?

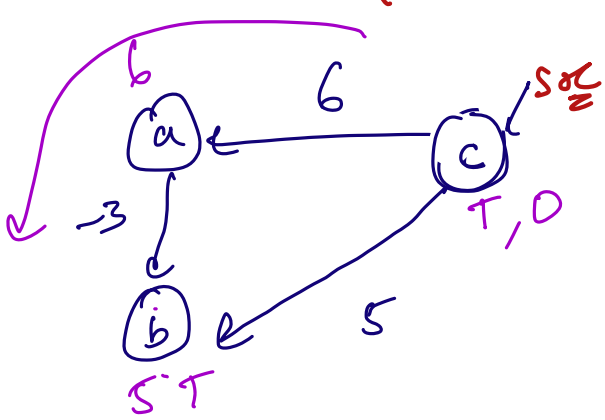
$v_1 = 10$

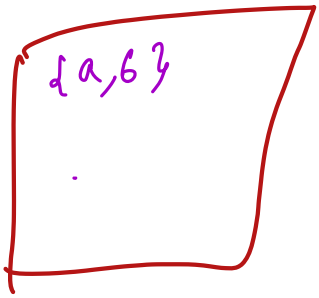
$v_1 = 5$



→ We also need to store index of every vertex in the heap array.

$O(E \log V)$



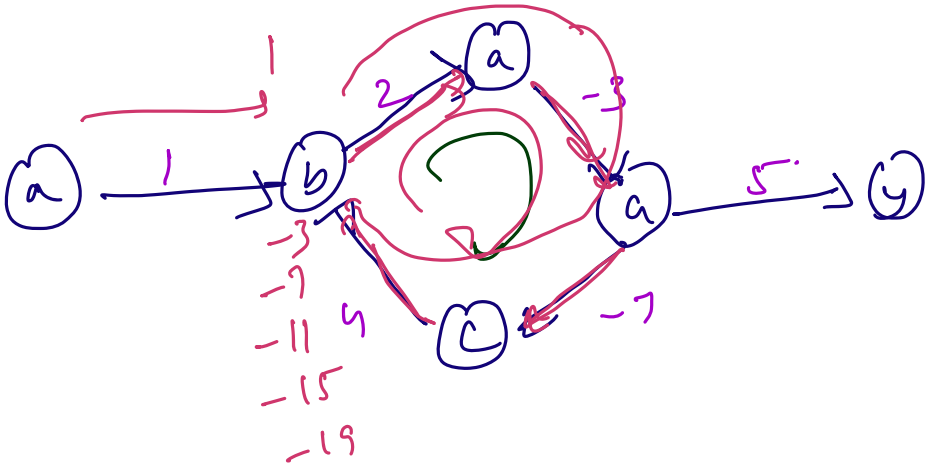


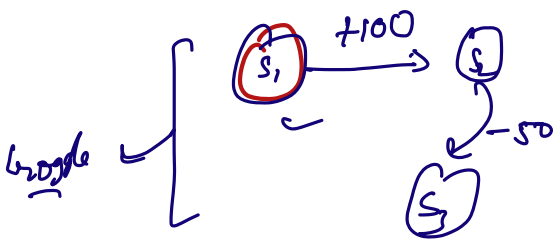
Dijkstra ✗

Negative Edge Weighted cycle [No Algo works for this Case]

there is a cycle whose sum of weights is negative.

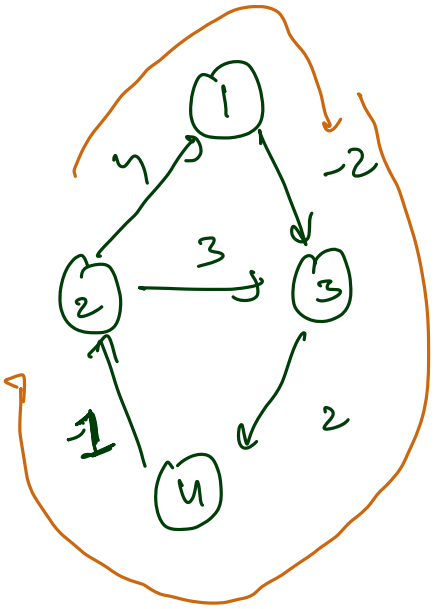
2 - 3 - 7 + 42 (-4)





Question: Floyd Warshall Algorithm [Negative Edges also]

All pair shortest Path



A_2

	1	2	3	4
1	0	-1		0
2	4	0	2	4
3			0	2
4				0

4 x 4 (V x V)

(1,4) = 0

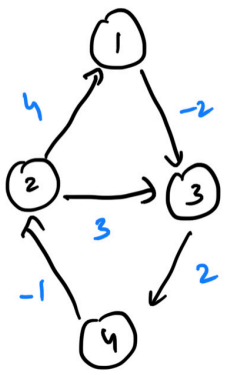
(2,3) = 2

A[i][j] = shortest path from vertex i to j.

→ Take a 2D matrix of V x V

→ Mark Diagonals as the edges

(1,4) = 2



A_2

	1	2	3	4
1	0	∞	-2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	-1	∞	0

V x V

→ Solve the problem in phases

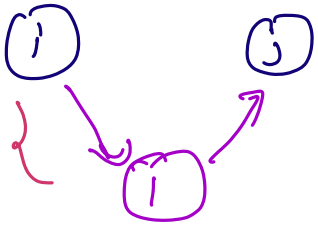
→ In phase 1,

For every pair (i, j)

A[i][1] + A[1][j] < A[i][j]

A[i][j] = A[i][1] + A[1][j]

}



→ In phase 2,
Consider Paths via 1, 2

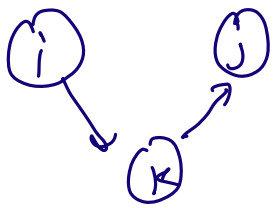
2
1
1,2

→ Phase 3
Consider Paths via 1, 2, 3

→ phase K,
Consider Paths via 1, 2, 3, ... k

→ phase V
Consider paths 1, 2, 3 - - . . V can be intermediate node

→ If k is an intermediate node in path from i to j

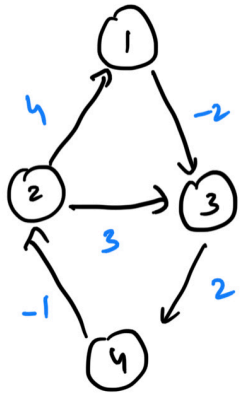


if(A[i][j] > A[i][k] + A[k][j]) {

A[i][j] = A[i][k] + A[k][j]

}

Phase 1:



	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	∞	-1	∞	0

if(A[i][i] + A[i][j] < A[i][j])

i == 1

if(A[i][i] + A[i][j] < A[i][j])

0

j == 1

A[i][i] + A[i][j] < A[i][j]

0

i == 1 && j == 1

(2,3) = (2,1) + (1,3) = 2 (4, -2)

(2,4) = (2,1) + (1,4) (4, ∞)

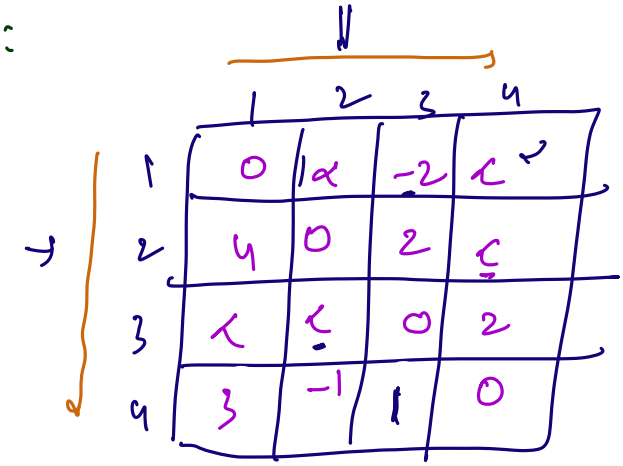
(3,2) = (3,1) + (1,2) (∞, ∞)

(3,4) = (3,1) + (1,4) (∞, ∞)

(4,2) = (4,1) + (1,2) (-1, ∞)

(4,3) = (4,1) + (1,3) (-1, 2)

Phase 2:



	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	3	-1	1	0

Consider paths via 2

k = 2

A[i][j] > A[i][k] + A[k][j]

k == i || j

(1,3) => (1,2) + (2,3) (∞)

(1,4) => (1,2) + (2,4)

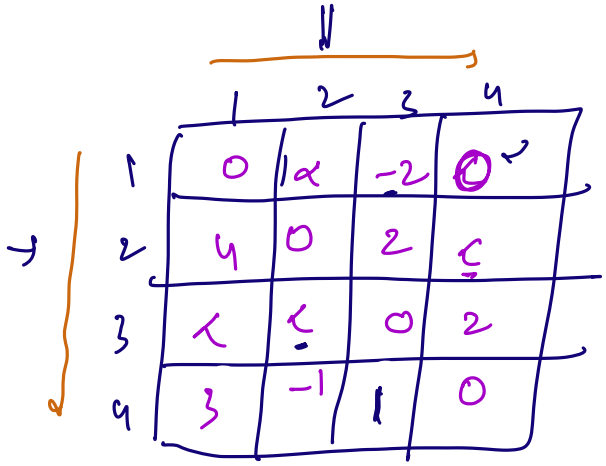
(3,1) => (3,2) + (2,1)

(3,4) => (3,2) + (2,4)

(4,1) => (4,2) + (2,1) = 3 (-1, 4)

(4,3) => (4,2) + (2,3) (-1, 2) = 1

Phase 3 :



3 as intermediate vertex

3rd row & 3rd column will not get updated

1,2 = 1,3 + 3,2
-2

1,4 = 1,3 + 3,4 = 0
-2 2

2,1 = 2,3 + 3,1

Dynamic Programming

Phase 3

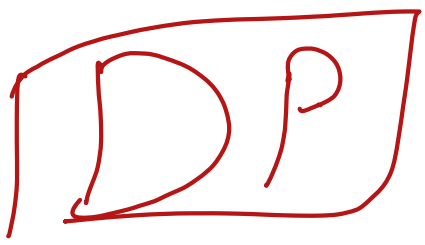
dist[1][4] = dist[1][3] + dist[3][4]

shortest dist from 1-3 in the first 2 phases

shortest dist from 3-4 in first 2 phases

dist[1][5] = dist[1][3] + dist[3][5]

Overlapping Subproblem



```
for(k = 0; k < n; k++){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(a[i][k] != INF && a[k][j] != INF
                && a[i][k] + a[k][j] < a[i][j])
                a[i][j] = a[i][k] + a[k][j]
        }
    }
}
```

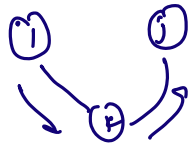
(i,j)

0,1,2 ---- N-1

Bottom-up DP

Tabulation

10^9



T-C: $O(V^3)$ ↙ V = 100

No of Vertices

S.C: $O(1)$ / $O(V^2)$

Check if Bipartite Graph

```
color[N] = {-1}
bool checkBipartite(int v, int clr){
      color[v] = clr;
      for(i in adj[v]){
            if(color[v] == -1){
                  if(!checkBipartite(i, 1 - color[v])
                        return false;
            }
            else if(color[v] == color[i])
                  return false;
      }
return true;
}

ans = checkBipartite(0, 0)
```

```
void dijkstra(int N, adj[], int src){
      dist[N] = INF;
      vis[N] = False;
      dist[src] = 0;
      minHeap(pair<int, int>) pq;
      pq.push({0, src});

      while(!pq.empty()){
            d, u = pq.top();
            pq.pop();
            if(vis[u])
                  continue;
            vis[u] = true;
            for(auto v : adj[u]){
                  if(dist[v] > dist[u] + W(u, v)){
                        dist[v] = dist[u] + W(u, v);
                        parent[v] = u;
                        pq.push({dist[v], v});
                  }
            }
      }
      return dist;
}
```

Floyd Warshall

```
for(k = 0; k < n; k++){
     for(int i = 0; i < n; i++){
          for(int j = 0; j < n; j++){
               if(a[i][k] != INF && a[k][j] != INF
                    && a[i][k] + a[k][j] < a[i][j])
                    a[i][j] = a[i][k] + a[k][j]
          }
     }
}
```