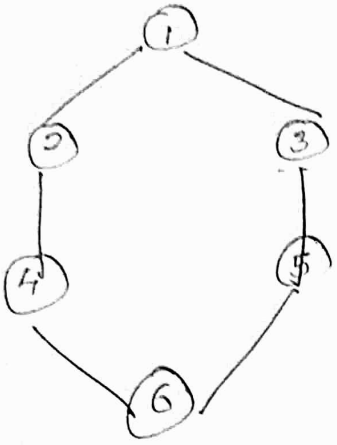
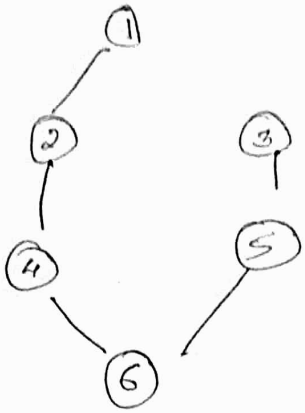
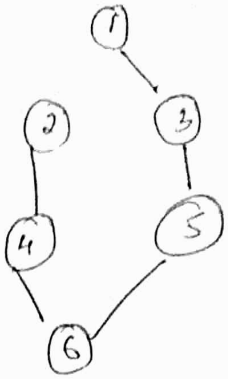
Prim's

Minimum Cost Spanning Tree







$G = (V, E)$

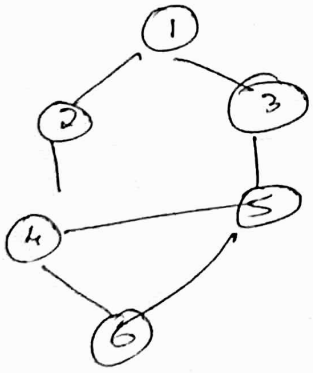
$V = \{1, 2, 3, 4, 5, 6\}$

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

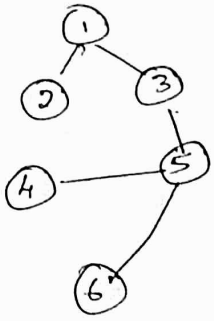
$|V| = N = 6$

$|V| - 1 = 5$

How many diff Spanning Tree is possible? ${}^{6}C_5 = 6$



${}^{7}C_5$

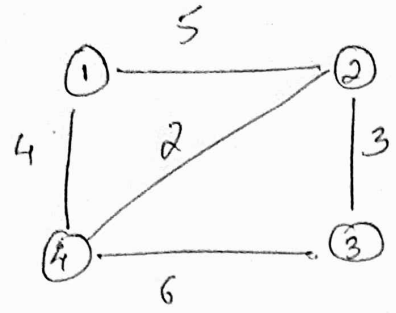


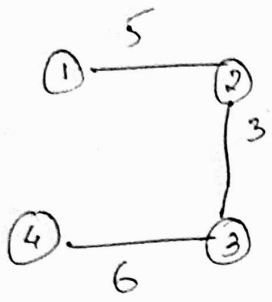
${}^{7}C_5 - 2$

$|E|$ — No of Cycles.

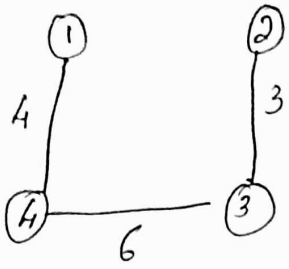
$C_{|V-1|}$

Ex:

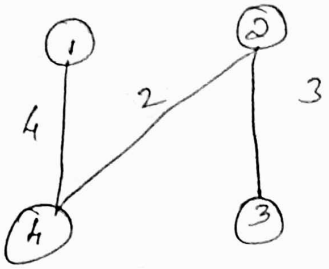




Cost = 14

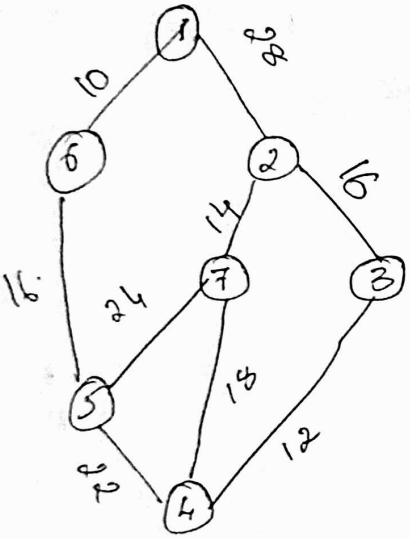


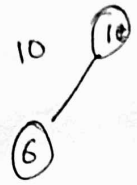
Cost = 13

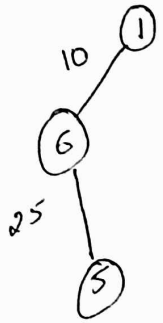


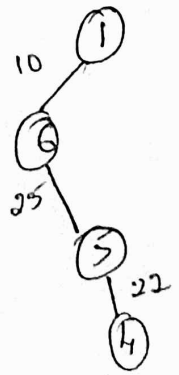
Cost = 9

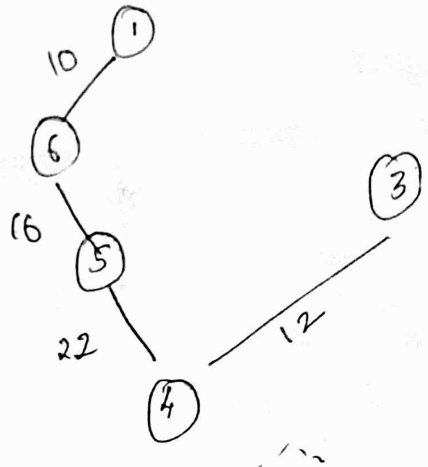
1. Prims
2. Kruskal's Algm.

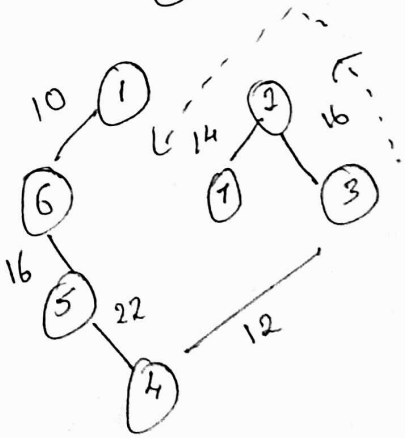






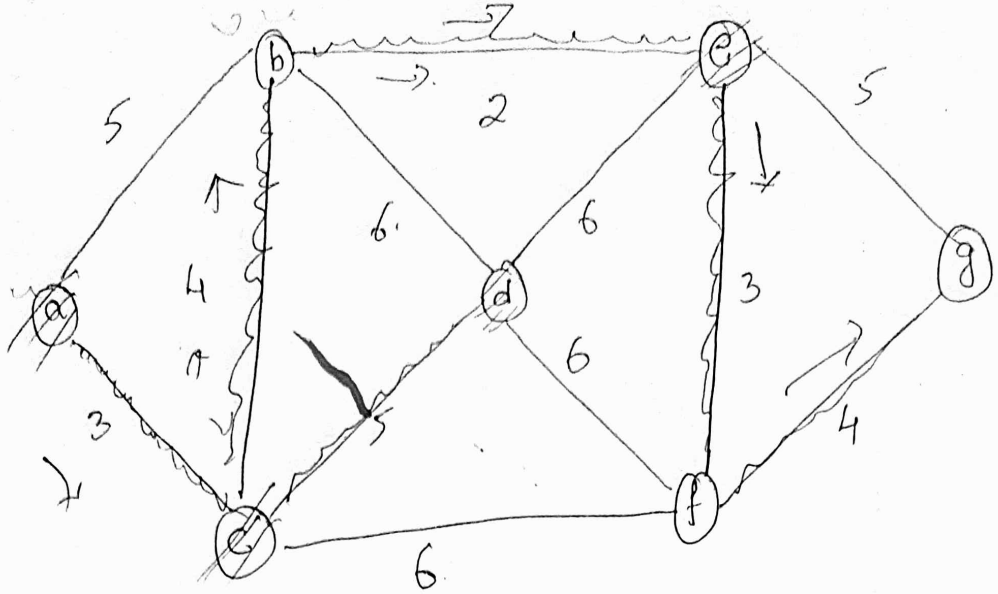






Cost = 49

```
for each u ∈ V(G)
{
    key[u] ← ∞
    π[u] ← NIL
}
key[r] ← 0
Q ← V[G]
while (Q != ϕ)
{
    u ← Extract-Min(Q)
    for each v ∈ adj[u]
    {
        if (v ∈ Q & w(u,v) < key[v])
        {
            π[v] ← u
            key[u] ← w(u,v)
        }
    }
}
```



	a	b	c	d	e	f	g
key	0	4	3	5	2	3	4
find	—	min ②	min ①	—	min ③	min ④	
Π[u]	—	a	a	c	b	e	f

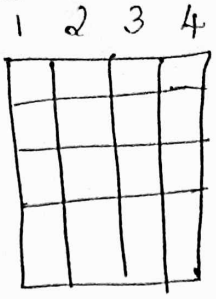
N- Queens

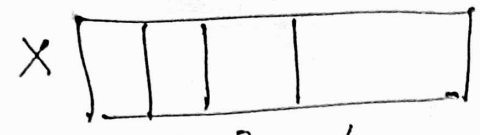
$^{16}C_4$

* Back Tracking.

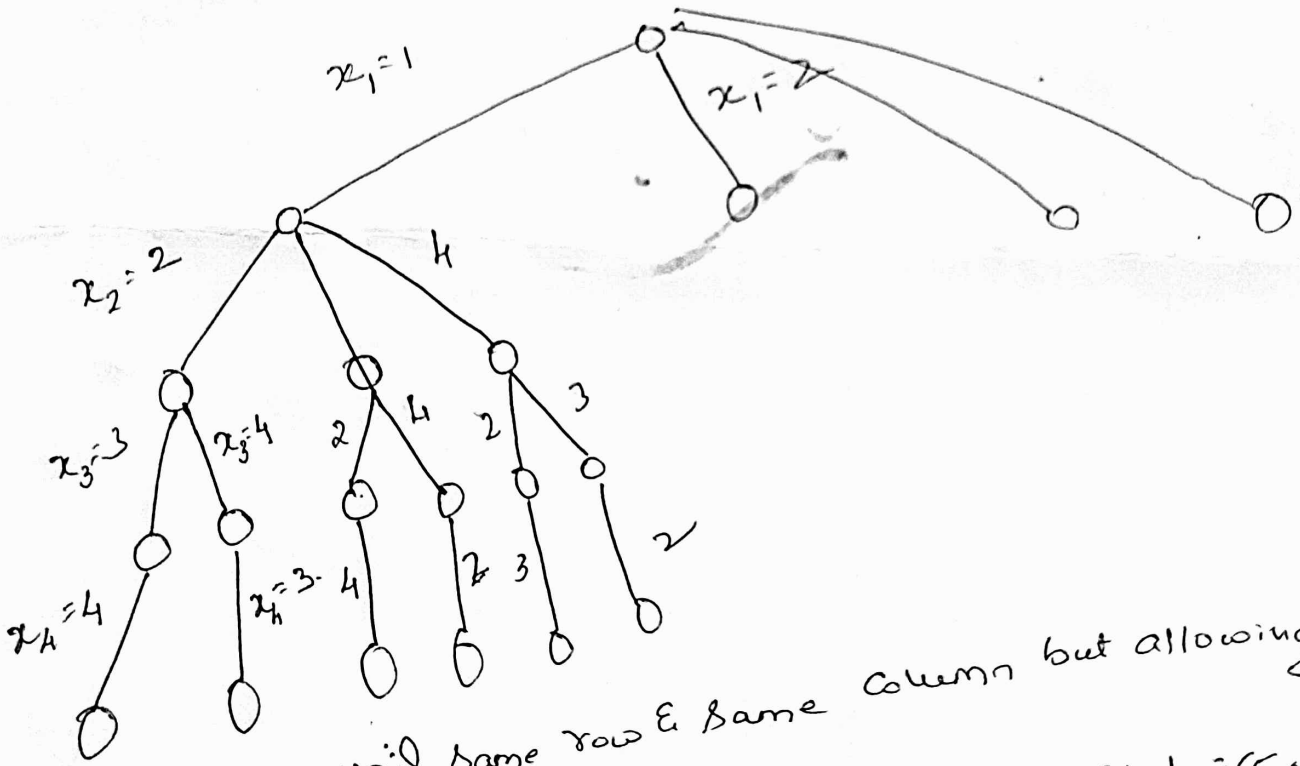
Queens Under Attack

* Same Row
* Column
* Diagonal





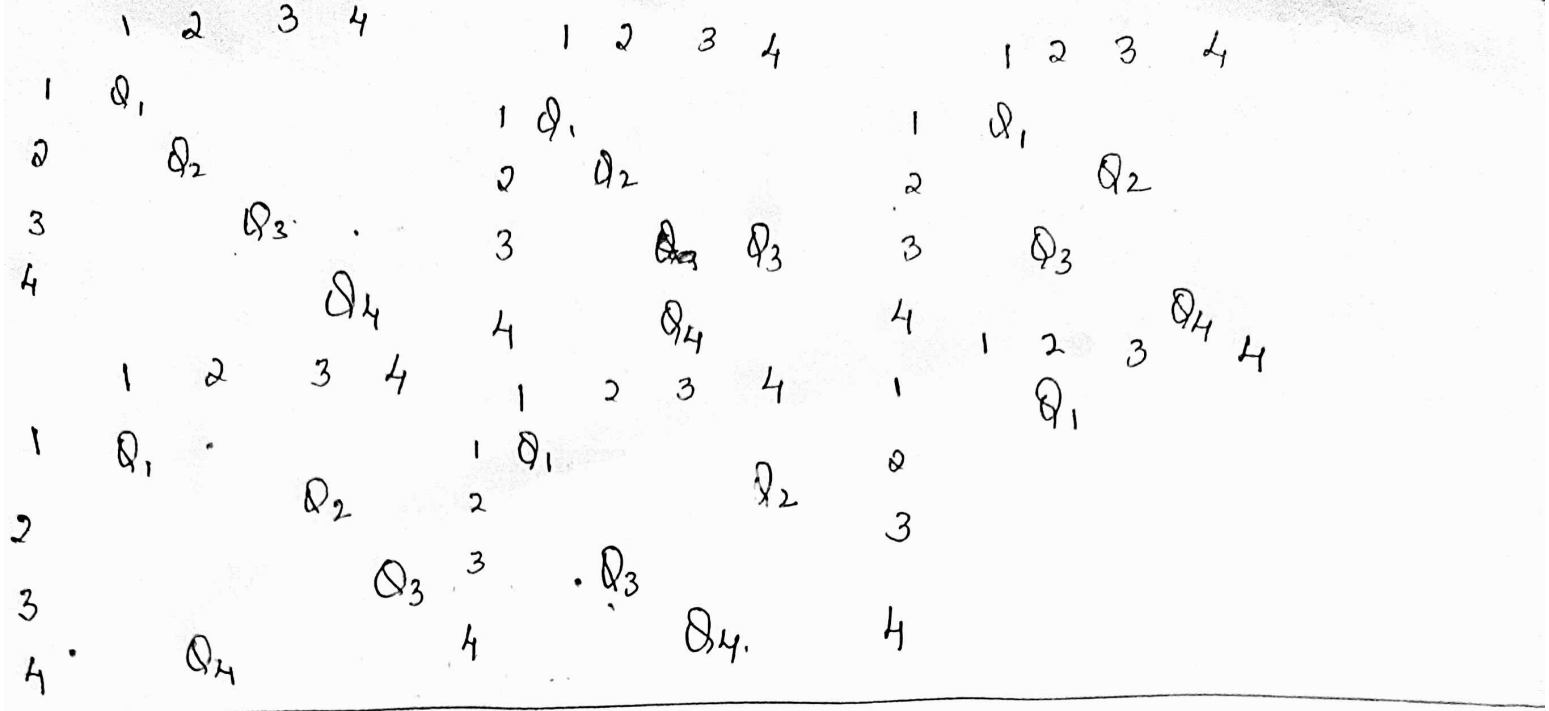
State Space Tree (without Attacks we prefer State Space Tree)

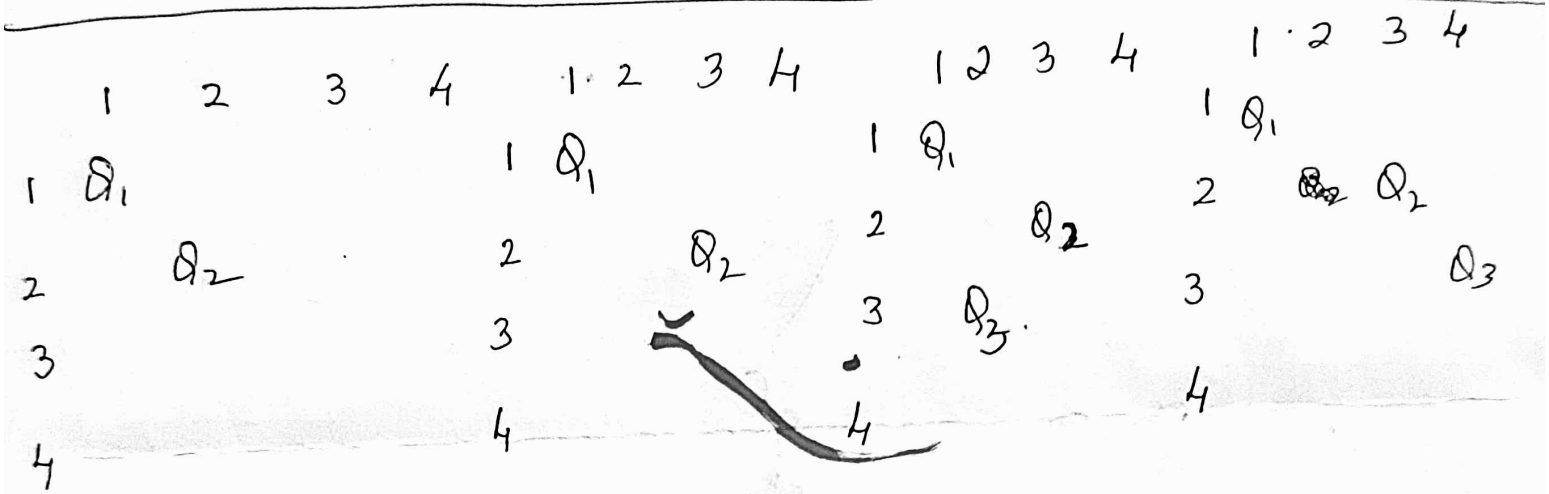


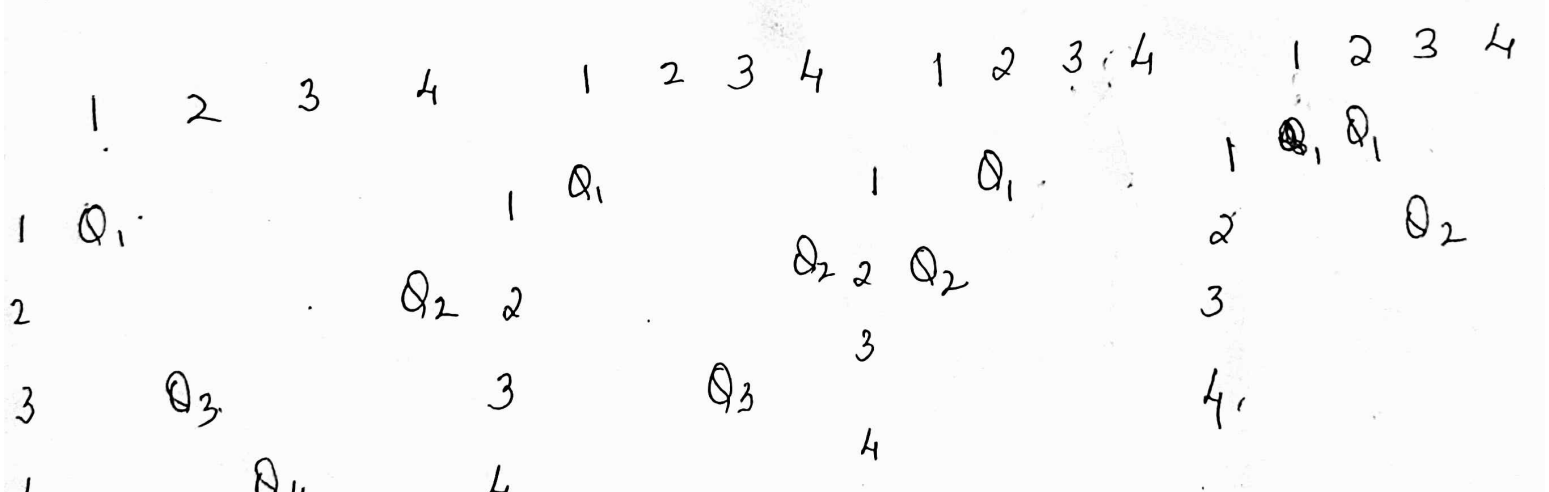
When we avoid same row & same column but allowing diagonal

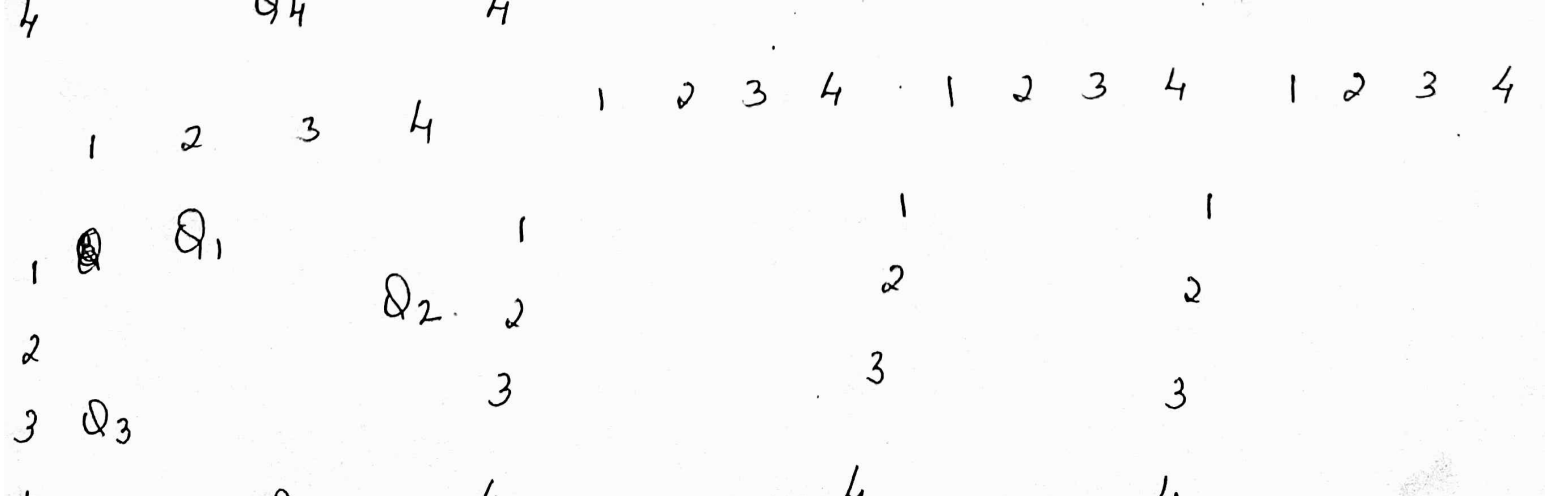
$$1+4+4*3+4*3*2+4*3*2*1 = 65 \text{ Nodes.}$$

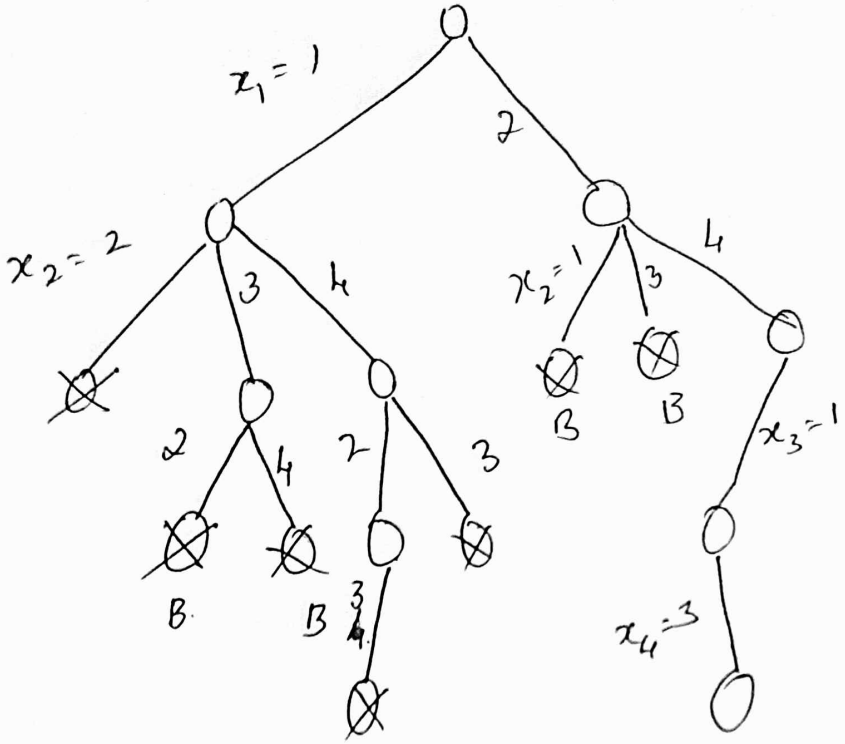
$$1+\sum_{i=0}^{3}\left[\prod_{j=0}^{i}(4-j)\right] = 65 \text{ Nodes.}$$

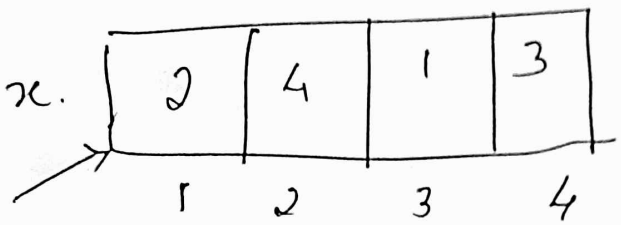

1 2 3 4
1 Q1
2 Q2
3 Q3
4 Q4


1 2 3 4
1 Q1
2 Q2
3 Q3
4 Q4


1 2 3 4
1 Q1
2 Q2
3 Q3
4 Q4

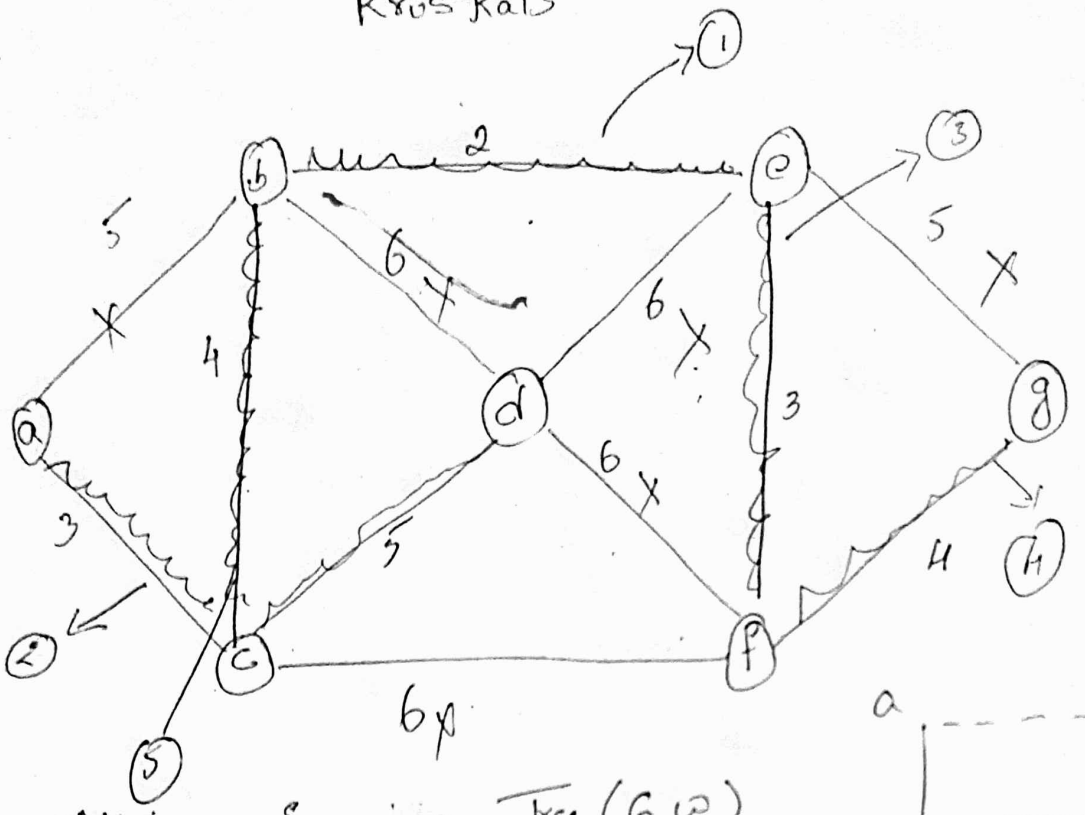

1 2 3 4
1 Q1
2 Q2
3 Q3
4 Q4





Column

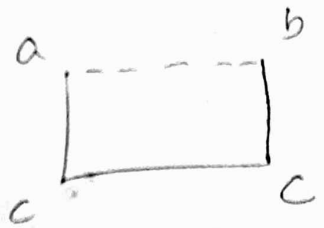
Kruskal's



Minimum Spanning Tree (G, w)

```
{
    A ← ϕ
    for each vertex v ∈ V(G)
    {
        do make-set(v)
    }
Sort the Edges of E into non-decreasing order by weight w.
for each Edge (u, v) ∈ E, then in Non-decreasing order
by weights.
{
    if [find-set[u] != find-set(v)]
    {
        A ← A ∪ {(u, v)}
        union(u, v)
    }
}
Return A
}
```

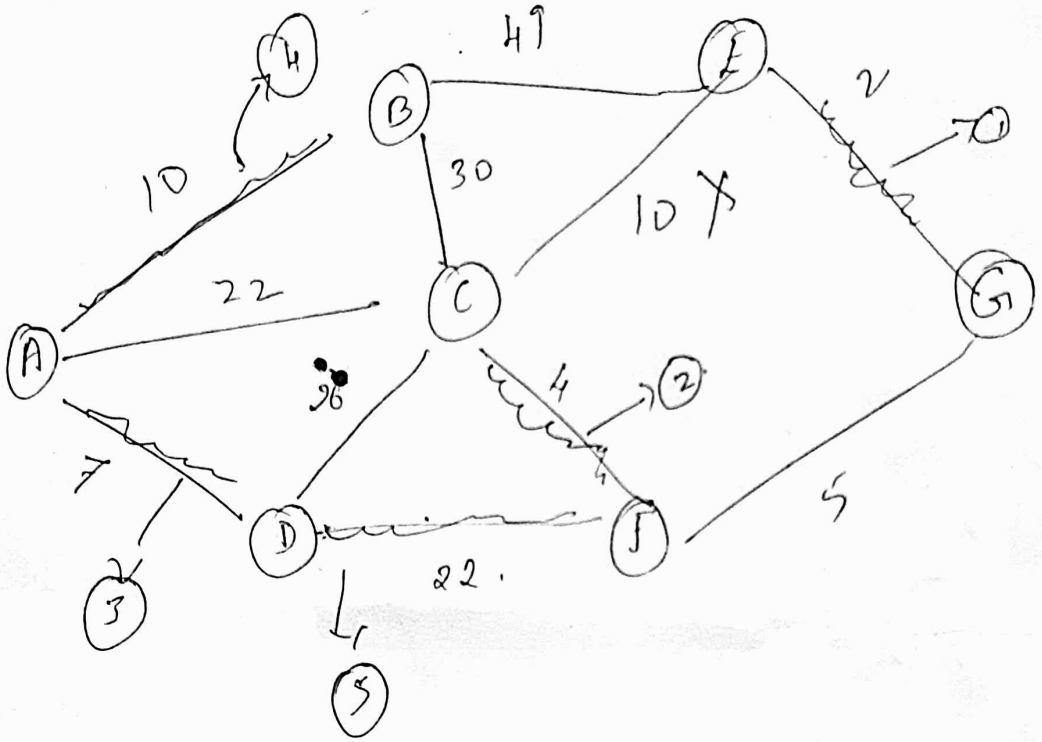
find set (b) = b
(e) = e
find set (a) = a
(c) = c
findset (f) = b e f
find set (g) = g
find set (c) = a c
b = b e f g
findset (a) = a b c e f g } will form cycle
find set (b) = b e a e f g }



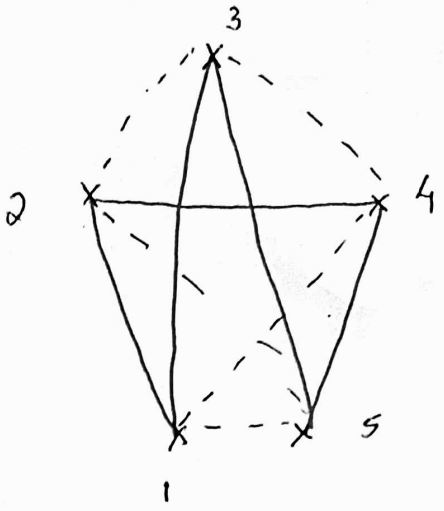
* Pick Shortest Edge

* If Cycle then avoid such Edges.

* until graph is not Connected keep on moving



Travelling Salesman Problem.



5-City TSP

Feasible Solution:

1-2-4-5-3-1

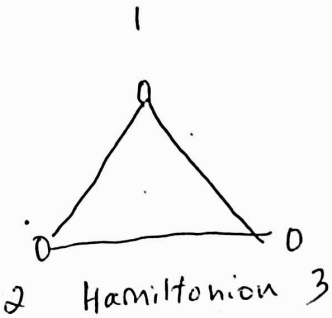
1-4-3-2-5-1

for n-Nodes we have $(n-1)!$ Solution

1-5-2-3-4-1

2-3-4-1-5-2

"Hamiltonian"

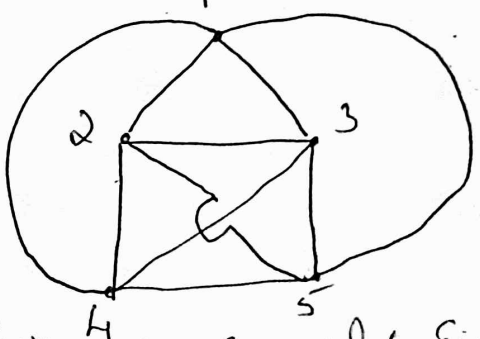


Hamiltonian

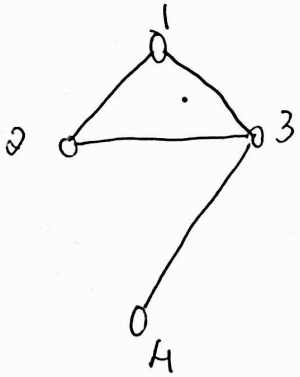
1-3-2-1

Hamiltonian

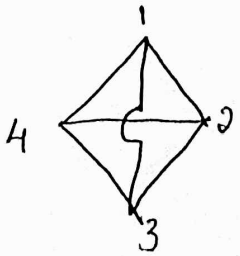
We need to find least cost Hamiltonian CKt.



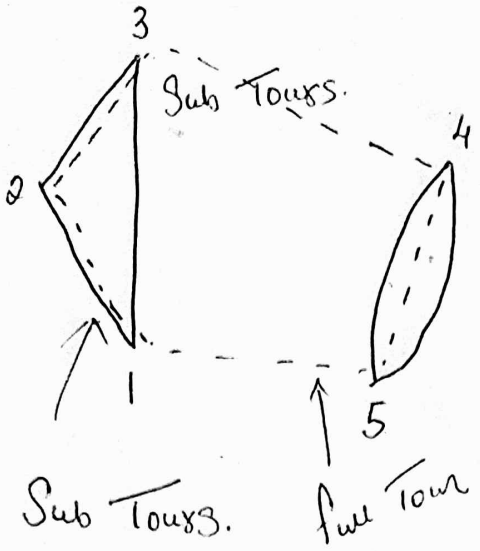
Every vertex connected to Every other Vertex desasion problem moves from desasion to optimization problem



Not Hamiltonian.



If a Graph is Hamiltonian a Graph may have more than 1 Circuit



	1	2	3	4	5
1	–	10	8	9	7
2	10	–	10	5	6
3	8	10	–	8	9
4	9	5	8	–	6
5	7	6	9	6	–

Feasible Solun should Comprise of Tours but not Sub Tour

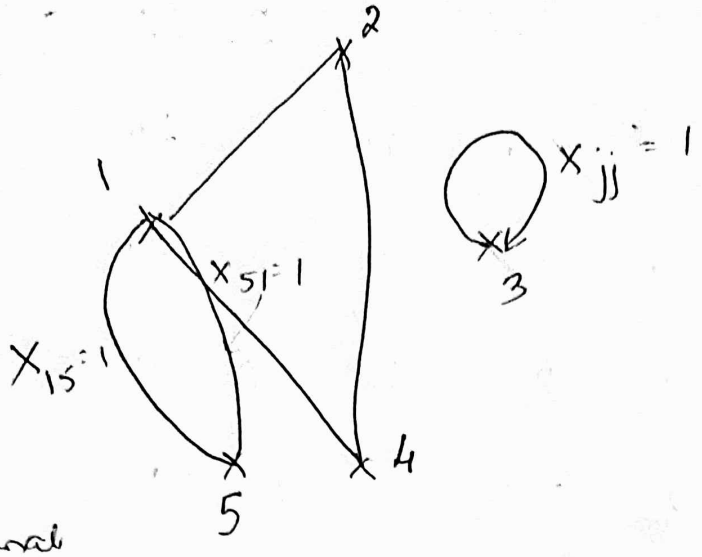
$X_{ij} = 1$ if the person Goes immediately from i to j

$$\sum \sum C_{ij} X_{ij}$$

$$\sum_{j=1}^{n} X_{ij} = 1 \quad \forall i$$

$$\sum_{i=1}^{n} X_{ij} = 1 \quad \forall j \;, \quad X_{ij} = 0/1$$

We need to add some more problems in TSP & those Constraints are Called Sub Tour Elimination Constraint



Elimination

length 1 $X_{jj} = 0$ $C_{jj} = \infty$ $n = 5$ Any $^nC_8$ is

2 $X_{ij} + X_{ji} \leq 1 \rightarrow {}^nC_2$ $^nC_2 = 10$ Exponential

$X_{15} = 1 + X_{51} = 1$ (If X_{15} is in Solution then X_{51} cannot be in the Solution.

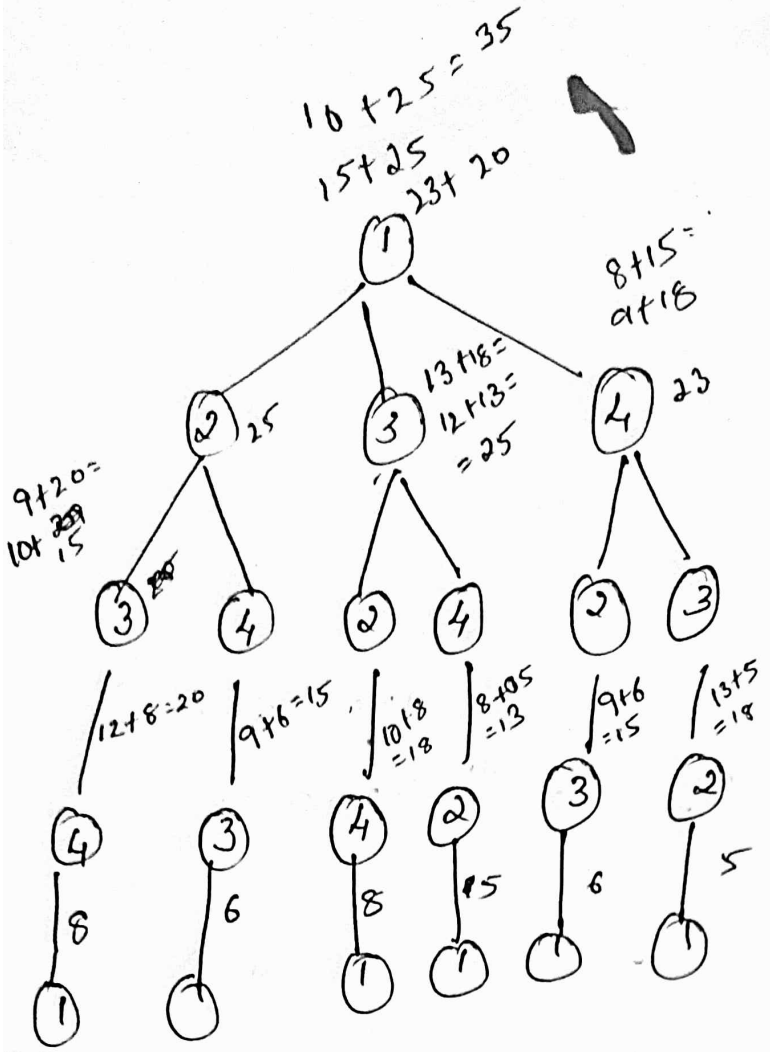
$1 + 1 \leq 1 \rightarrow {}^nC_3$ $n = 5$ $^nC_3 = 10$

3. $X_{ij} + X_{jk} + X_{ki} \leq 2 \rightarrow {}^nC_3$.

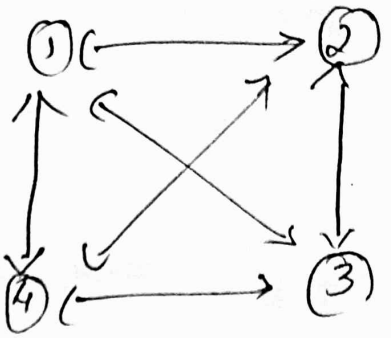
Sub Tour Elimination constraints are large & many.

Five City TSP.

Sub Tours of length		need to be Eliminated
	① ② | 3 4	
	$C_{jj} = \infty$ $^nC_2$.	$^nC_2 + {}^nC_3 + {}^nC_{\frac{n-1}{2}}$; N is odd
		; N is Even.
	$^nC_2 +$	
	1 2 3 | 4 5	$^nC_2 + {}^nC_3 + {}^nC_{\frac{n}{2}}$



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



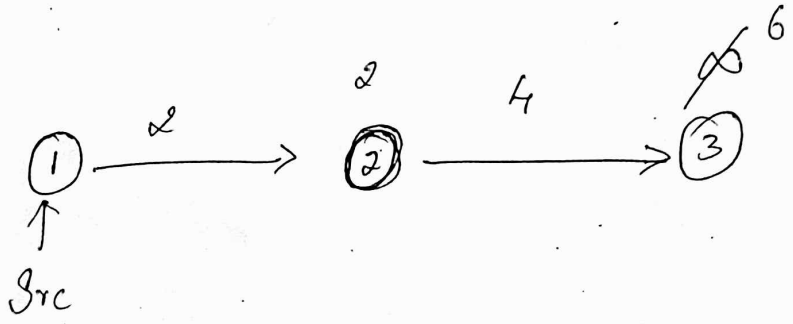
$$g(1, \{2,3,4\}) = \min \{ C_{1k} + g(k, \{2,3,4\} - \{1k\}) \}$$

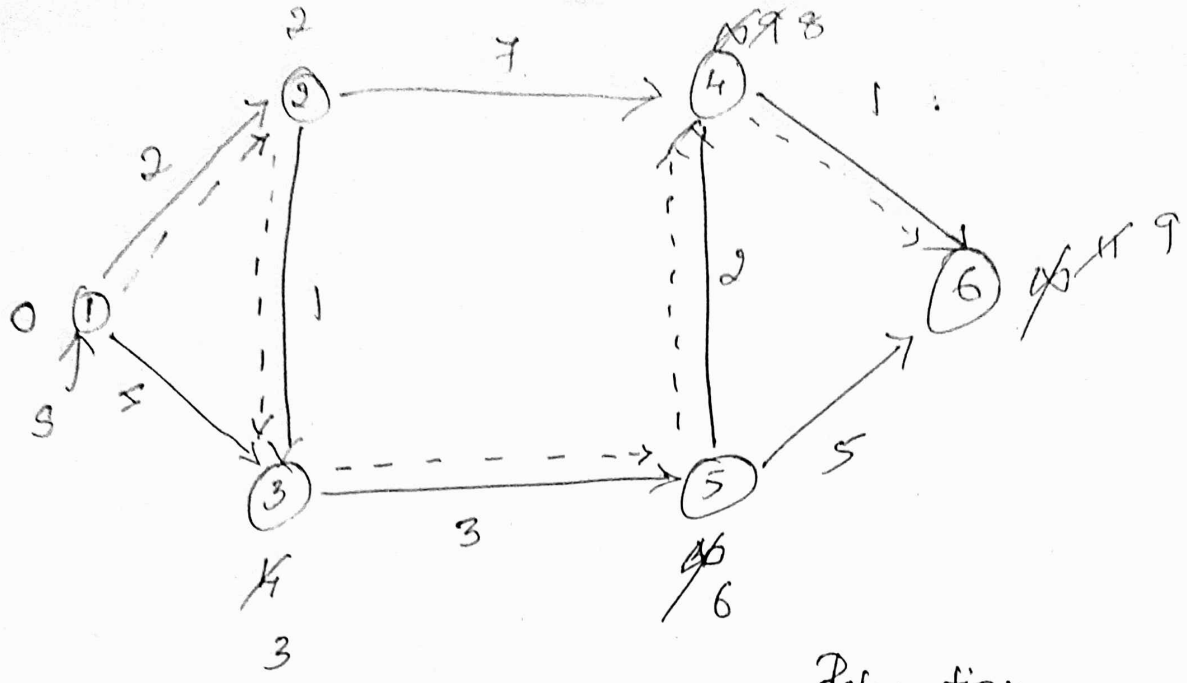
$$k \in \{2,3,4\}.$$

$$g(i, S) = \min_{k \in S} \{ C_{i,k} + g(k, S - \{k\}) \}$$

Dijikstras Algorithm.

* Single Source Shortest path.
* Greedy Method.
* For any selected source vertex we find shortest path to all other vertices via directed Edges or via undirected Edges.
* As we need to find a shortest path it is a minimization problem. Minimization problem is a optimization problem & can be solved using a Greedy method.
* Greedy method says that problem to be solved in stages by taking one ~~step~~ i/p at a time. ie (Shortest Path)
* Dijikstras algo gives an optimal solution
  ↳ works on directed & non directed Graphs





V	d[v]
2	2
3	3
4	8
5	6
6	9

Relaxation

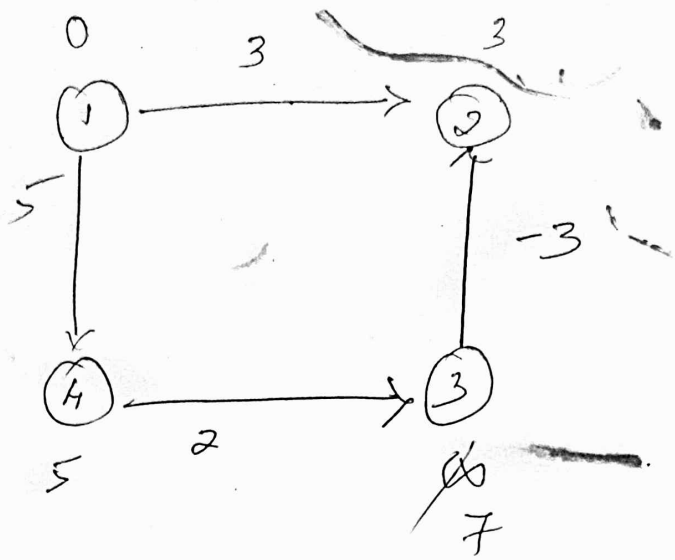
if $(d[u] + c(u,v) < d[v])$

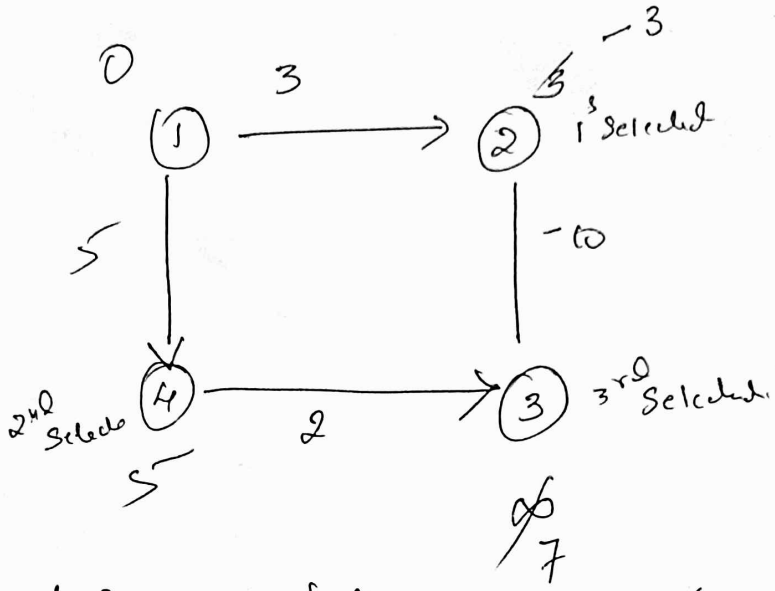
$d[v] = d[u] + c(u,v)$

$n = |V|$

$n \times n = n^2$

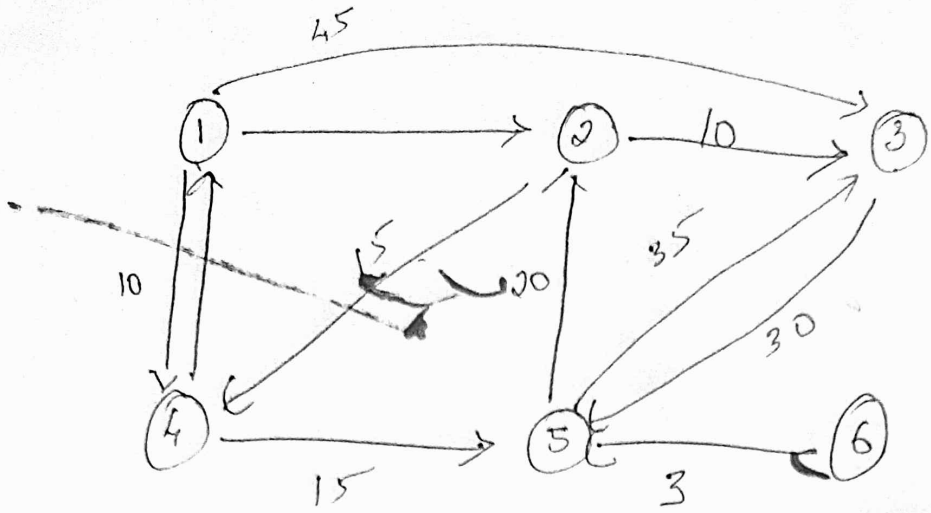
Draw Back of Dijkstras algm.





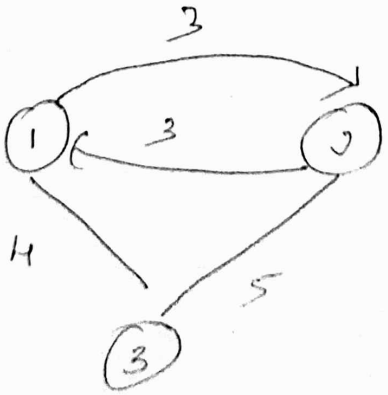
we are finding a better answer for already visited Vertex due to -ve Edge

Dijkstras algo may work or may not work on -ve Edges.



Starting Vertex 1

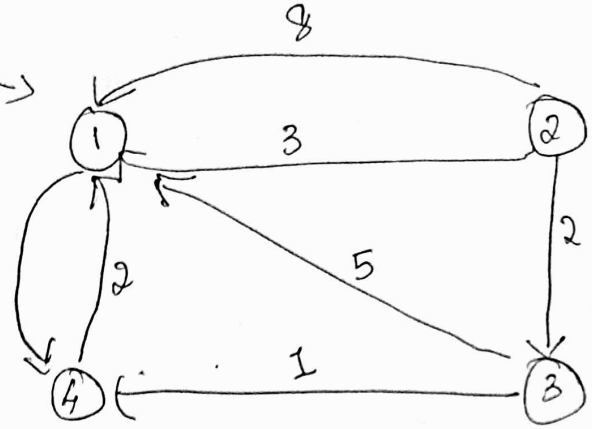
Selected Vertex	2	3	4	5	6
4	50	45	(10)	∞	∞
5	50	45	(10)	(25)	∞
2	(45)	45	(10)	(25)	∞
3	(45)	(45)	(10)	(25)	∞
6	(45)	(45)	(10)	(25)	(∞)



All Pair Shortest Path Problem

* Floyd-warshalls
* Dynamic Approach.
* Problem is to find shortest Path b/w Every pair of vertices

directed weighted Graph.

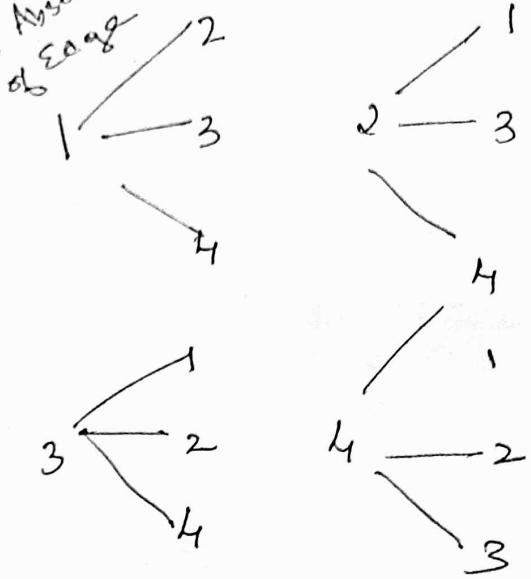


* Dijkstra but diff is Dijkstra is Single Source shortest path & warshall - floyd is all pair shortest path problem.

Adjacency matrix
self loop.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

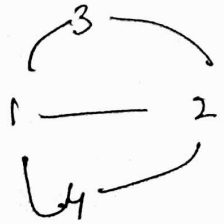
Absence of Edge



* All pair shortest path problem can be solved using greedy approach also.

$n^2 \times n = O(n^3)$

* In Dynamic programming problem is solved using Sequence of stages or taking



* Shortest path via some other vertex. via 1, 2, 3 --- N

A^1

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	8	∞	0

$A^0[2,3]$ $A^0[2,1] + A^0[1,3]$

$2 < 8 + \infty$

$A^0[2,4]$ $A^0[2,1] + A^0[1,4]$

$\infty > 8 + 7$

$\infty > 15$

$A^0[3,2]$ $A^0[3,1] + A^0[1,2]$

∞ $5 + 3$

$\infty > 8$

A^2

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3		8	0	
4		8		0

$A^1[1,3] = A^1[1,2] + A^1[2,3]$

$\infty > 3 + 2$

$8 > 5$

$A^1[1,4]$ $A^1[1,2] + A^1[2,4]$

$7 < 3 + 15$

$A^2 =$

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

$A^3 =$

	1	2	3	4
1	0		5	
2		0	2	
3	5	8	0	1
4			7	0

$A^2[1,2] = A^2[1,3] + A^2[3,2]$

$3 < 5 + 8$

$A^4 =$

	1	2	3	4
1	0	3	5	6
2	5	0	2	3
3	3	6	0	1
4	2	5	7	0

$$A^{k}[i,j] = \min\left\{A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j]\right\}$$

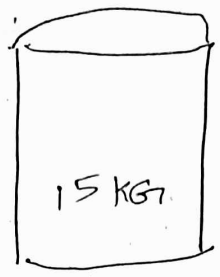
```
for ( k=1; k<=n; k++ )
{
    for( i=1; i<=n; i++ )
    {
      for (j=1; j<=n; j++ )
      {
A[i,j] = min(A[i,j], A[i,k] + A[k, j] )
      }
    }
}
```

Knapsack Problem.

$n=7$ $\quad m=15$

Objects	1	2	3	4	5	6	7
Profits	10	5	15	7	6	18	3
weights	2	3	5	7	1	4	1
$\frac{P}{w}$	5	1.3	3	1	6	4.5	3.
x	$(1$	$2/3$	1	0	1	1	$1)$
	x_1	x_2	x_3	x_4	x_5	x_6	x_7

$0 \le x \le 1$



$15-1=14$
$14-2=12$
$12-4=8$
$8-5=3$
$3-1=2$
$2-2=0$

$$\sum x_i w_i = 1*2 + \frac{2}{3}*3 + 1*5 + 0*7 + 1*1 + 1*4 + 1*1$$

$$= 2+2+5+0+1+4+1 = 15$$

$$\sum x_i P_i = 1*10 + \frac{2}{3}*5 + 1*15 + 1*6 + 1*18 + 1*3$$

$$= 10 + 2*1.3 + 15 + 6 + 18 + 3 = 54.6.$$

Constraint

$$\sum x_i w_i \le m$$

Objective: $\max \sum x_i P_i$

```
Knapsack (w[1....n], p[1....n], W)

for i=1 to n
    do x[i]=0
weight = 0
for i=1 to n.
    if weight + w[i] ≤ W then.
        x[i] = 1
        weight = weight + w[i].
    else
        x[i] = (W - weight) / w[i]
        weight = W.
        break
return x
```

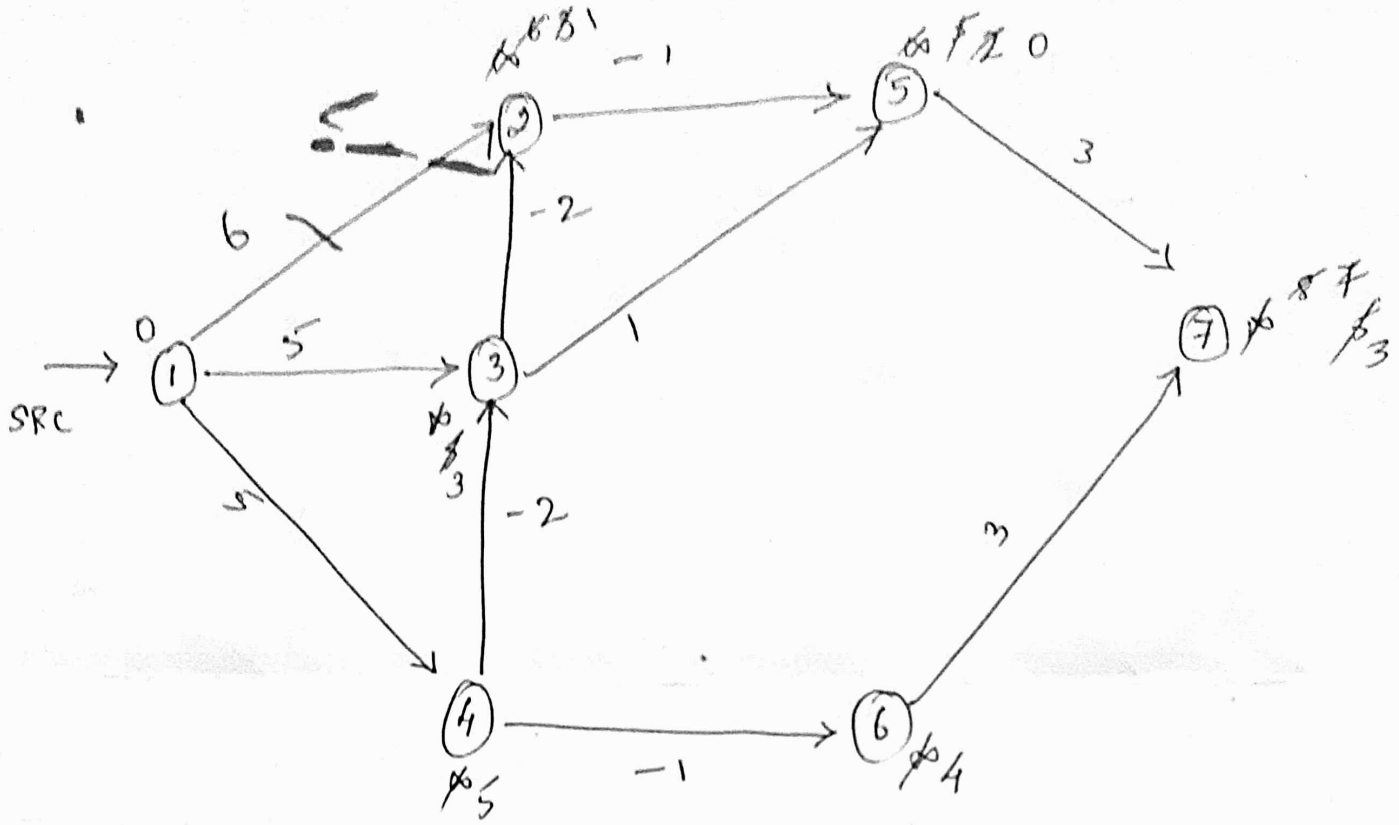
Bellman - Ford Algm.

* Dynamic Programming
* Negative weight Cycles.
* Single Source Shortest Path algm from single src vertex to all other vertices in weighted diagraph.
* It is slower than Dijkstras but more versatile as it can handle -ve weights.
* Dijkstras may/may not work with Negative Edges.

```
Bellman-Ford (G, w, s)
{
 Initialize Single Source (G,S)
 for i ← 1 to |V(G)| - 1
 {
  for Each Edge (u,v) ∈ E(G)
   {
    Relax (u,v,w)
   }
 }
 for Each Edge (u,v) ∈ E(G)
 {
  if (d[v] > d[u] + w(u,v))
   {
    Return false
   }
 }
}
```

```
Relax (u, v, w)
{
if (d[v] > d[u] + w(u,v))
{
d[v] ← d[u] + w(u,v)
π[v] ← u
}
}
```

* Dynamic : Try all Possible Solutions & select the best one
* Keep on relaxing the Edges (N-1) times



I Edge List; (1,2) (1,3) (1,4) (2,5) (3,2) (3,5) (4,3) (4,6) (5,7) (6,7)

II : (2,5) (3,2)

(5,7)

III : (2,5)

(5,7)

IV : No change

Result

1 - 0
2 - 1
3 - 3
4 - 5
5 - 0
6 - 4
7 - 3