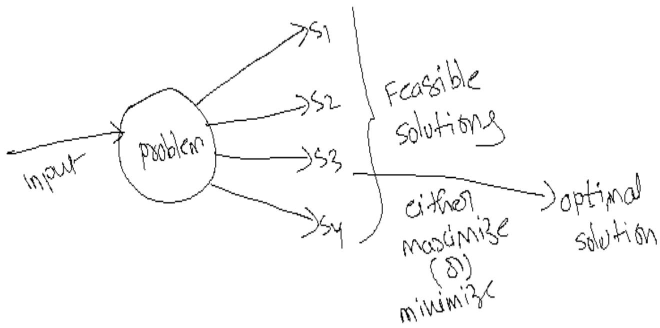
UNIT-3
GREEDY METHOD

Divide and conquer technique is only solving divisible problems, which can't be solving undivisible problems.
Ex: minimum cost spanning tree, knapsack problem

Greedy method is a straight forward method. It takes one input at a time and produce number of solutions these solutions are called feasible solutions. Out of all the feasible solutions one of the solution either maximize or minimize our constraint that is called optimal solution.



Ex: we have to find the maximum value for the problem z=3x+4y
Constraints are 0<=x<=1 , -1<=y<=1

Ex $z=3x+4y$; $0\leq x\leq 1$, $-1\leq y\leq 1$; maximum value

sol $x=0,1$; $y=-1,0,1$, (x,y)

$(0,-1)$ — $z=3*0+4*-1 \Rightarrow z=0-4=-4$
$(0,0)$ — $z=3*0+4*0 \Rightarrow z=0+0=0$
$(0,1)$ — $z=3*0+4*1 \Rightarrow z=0+4=4$
$(1,-1)$ — $z=3*1+4*-1 \Rightarrow z=3-4=-1$
$(1,0)$ — $z=3*1+4*0 \Rightarrow z=3+0=3$
$(1,1)$ — $z=3*1+4*1 \Rightarrow z=3+4=7$

Feasible solutions

maximize → optimal Solution

Feasible solutions=={(0,-1)(0,0)(0,1)(1,-1)(1,0)(1,1)}
Optimal solution={(1,1)}

Feasible solution: the problem will takes one input at a time and produce number of solutions, these solutions are called feasible solutions.

Or

The given input set that satisfies the given constraint that are called feasible solutions

Optimal Solution: the optimal solution is also a feasible solution. That maximizes or minimizes the given function that is called optimal solution.

Algorithm :

```
Algorithm greedy(a[],n,i)
{
//a[] is an input array of size n
Solution:=0
For i:=1 to n do
{
X:= select(a);
If feasible then
{
Solution:=union(solution,x)
}
Reject()
}
Return solution;
}
```

0/1 Knapsack Problem:

In olden days there was a store which contains gold items. Now a thief wants robbery the bank so that he will brought the empty bag (knapsack). But the problem is in what way he has to robbery the bank so that he will get the maximum profit.

0-not placing the items in to the knapsack
1-placing the items in to the knapsack

The gold items are n1,n2,n3,n4
Profits or each item p1,p2,p3,p4
Cost of each item c1,c,2,c3,c4
Weight of each item w1,w2,w3,w4

$$\sum \quad pi*xi= \; p1x1+p2x2+p3x3+p4x4$$

1<=i<=n

X=fractions means how much item we can placed in to the bag either complete item(1), or no item(0), or fraction of item we can placed in to the bag.

Algorithm:

```
Algorithm knapsack(i,m,n,w,x)
{
//p[i] is the profit of the item
//w[i] is the weight of the items
//x[i] is the fractions of items
//M is the size of the bag (knapsack)

For i:=1 to n do
{
If(w[i]<M)
{
X[i]:=1
M=m-w[i]
}
X[i]:=M/w[i]
}
}
```

Ex: consider the following instances of the 0/1 knapsack problem
n=3, M=20, (p1,p2,p3)=(25,24,15) , (w1,w2,w3)=(18,15,10)

Sol:

1.Maximum Profit
2.Minimum Weight
3.Maximum profit per unit weight(p/w)

1.Maximum Profit: in this case we can place the items in to the bag based on the maximum profit.
(p1,p2,p3)=(25,24,15)
P1>p2>p3

First we can place first item in to the bag. Now x1=1 means we can place complete item in to the bag.
After placing the item we can calculate the remaining bag size

remaining bag size=[Bag size- item size to be placed]=[20-18]=2

next we can place 2nd item in to the bag. Item size(15) is greater than the bag size(2) then we can break the item.

X2=remaining bag size/weight of the item to be placed

X2=2/15

After placing the 2nd item there is no space in the bag, so we con't place remaining items in to the bag, so x3=0.

X1=1, x2=2/15, x3=0

$$\sum_{1<=i<=n} pi*xi= p1*x1+p2*x2+p3*x3$$

$$=25*1+24*2/15+15*0$$
$$=25+3.2+0$$
$$=28.2$$

2.Minimum Weight: in this case we can place the items in to the based on the minimum weight.
(w1,w2,w3)=(18,15,10)

W3<w2<w1

First we can place 3rd item in to the bag. Now $x3=1$ means we can place complete item in to the bag. After placing the item we need to calculate remaining bag size

remaining bag size=[Bag size- item size to be placed]
=[20-10]=10

Next we can place next minimum weight item in to the bag. So 15 is the next minimum weight that is 2nd item. So we can place 2nd item in to the bag.

2nd item size is greater than bag size so we can break that item.

X2=remaining bag size/weight of the item to be placed
X2=10/15=2/3

So x3=1,x2=2/3,x1=0

$$\sum_{1<=i<n} pi*xi= p1*x1+p2*x2+p3*x3$$

=25*0+24*2/3+15*1
=0+16+15
=31

3.Maximum Profit per unit weight (p/w): in this case we can place the items in to the bag whose p/w ratio is maximum.

P1/w1----------25/18-----→1.4
P2/w2----------24/15-----→1.6
P3/w3----------15/10-----→1.5

P2/w2>p3/w3>p1/w1

X2>x3>x1

First we can place 2^{nd} item in to the bag. So x2=1 means we can place complete item in to the bag.

remaining bag size=[Bag size- item size to be placed]

=[20-15]=5

Next we can place 3^{rd} item in to the bag. Now item size is 10 and remaining bag size is 5 so there is no sufficient space. So we can break the 3^{rd} item.

X3=remaining bag size/weight of the item to be placed
X3=5/10=1/2

After placing the 3^{rd} item in to the bag there is no space available in the bag. So we can't place remaining items in to the bag. We can't place first item in to the bag so X1=0

X1=0,x2=1,x3=1/2

$$\sum_{1<=i<=n} pi*xi = p1*x1+p2*x2+p3*x3$$

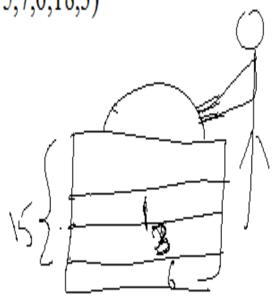
$$=25*0+24*1+15*1/2$$
$$=0+24+7.5$$
$$=31.5$$

A thief wants to robbery the bag based on p/w ratio.

Ex: find the optimal solution for the given instances of knapsack problem n=7, M=15 (p1,p2,p3,p4,p5,p6,p7)=910,5,15,7,6,18,3) (w1,w2,w3,w4,w5,w6,w7)=(2,3,5,7,1,4,1)

SOL:

find the optimal solution for the given instances of knapsack problem
n=7, M=15 (p1,p2,p3,p4,p5,p6,p7)=(10,5,15,7,6,18,3)
(w1,w2,w3,w4,w5,w6,w7)=(2,3,5,7,1,4,1)



Sol

1. maximum profit
2. minimum weight
3. P/w Ration

1. maximum profit :- in this case we can place maximum profit items into the Bag.

$p_6 > p_3 > p_1 > p_4 > p_5 > p_2 > p_7$

First we can place 6th item into the Bag.
$x_6 = 1$ means we can place complete item into the Bag

Remaining Bag Size = [Bag Size − item Size tobe placed]
= [15 − 4] = 11

next we can place 3rd item into the Bag. $x_3 = 1$
means we can place complete item into the Bag

Remaining Bag Size = [11 − 5] = 6

next we can place 1st item into the Bag. $x_1 = 1$
means we can place complete item into the Bag

Remaining Bag Size = [6 − 2] = 4

next we can place 4th item into the Bag.
4th item size is greater than Bag Size so we can Break the item.

$$x_4 = \frac{\text{Remaining Bag Size}}{\text{weight of the item tobe placed}} = \frac{4}{7}$$

After placing 4th item there is no space available in the Bag so we can't place Remaining item

So $x_6=1, x_3=1, x_1=1, x_4=\frac{4}{7}, x_5=0, x_6=0, x_7=0, x_2=0$

$$\sum_{1\le i\le n} p_i x_i = p_1x_1+p_2x_2+p_3x_3+p_4x_4+p_5x_5+p_6x_6+p_7x_7$$

$$=10*1+5*0+15*1+\cancel{7}*\frac{4}{\cancel{7}}+6*0+18*1+3*0$$

$$=10+0+15+4+0+18+0$$

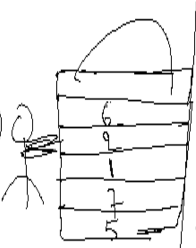
$$=10+15+4+18$$

$$=\underline{47}$$

2. minimum weight:- In this case we can place the items into the Bag whose weight is minimum

$(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$

$w_5, w_7 < w_1 < w_2 < w_6 < w_3 < w_4$



* First we can place 5^{th} item into the Bag. $x_5=1$ means we can place complete item into the Bag.

Remaining Bag size = [Bag size − item size to be placed]

$= [15-1] = 14$

* next we can place 7^{th} item into the Bag. $x_7=1$ (complete item)

$RBS = [14-1] = \underline{13}$

* next we can place 1^{st} item into the Bag. So $x_1=1$ (complete item)

$R.B.S = [13-2] = \underline{11}$

* next we can place 2^{nd} item into the Bag so $x_2=1$ means we can place complete item into the Bag.

$R.B.S = [11-3] = 8$

* next we can place 6^{th} item into the Bag. $x_6=1$ we can place complete item into the Bag.

$R.B.S = [8-4] = \underline{4}$

* next we can place 3rd item into the Bag
3rd item size is greater than Bag size
so we can Break the item.

$$x_3 = \frac{\text{Remaining Bag Size}}{\text{weight of the item tobe placed}}$$

$$= \frac{4}{5}$$

After placing 3rd item into the Bag there is no space Available in the Bag. so we can't place Remaining items into the Bag.

$x_1=1$ $x_2=1$ $x_3=\frac{4}{5}$ $x_4=0$ $x_5=1$ $x_6=1$
$x_7=1$

$$\sum_{1\le i\le n} p_i * x_i = p_1x_1+p_2x_2+p_3x_3+p_4x_4+p_5x_5+p_6x_6+p_7x_7$$

$$= 10*1+5*1+\overset{3}{\cancel{15}}*\frac{4}{\cancel{5}}+7*0+6*1+18*1+3*1$$

$$= 10+5+12+0+6+18+3$$

$$= \underline{54}$$

3. P/W Ratio:- In this we can place p/w Ratio into the Bag. whose p/w Ratio is maximum

p_1/w_1 — 10/2 — 5

p_2/w_2 — 5/3 — 1.6

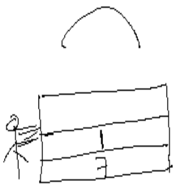
p_3/w_3 — 15/5 — 3

p_4/w_4 — 7/7 — 1

p_5/w_5 — 6/1 = 6

p_6/w_6 — 18/4 = 4.5

p_7/w_7 — 3/1 = 3



$x_7 > x_1 > x_6 > x_3 / x_7 > x_2 > x_4$

* First we can place 7th item into the Bag. $x_7=1$
we can place Complete item into the Bag.

Remaining Bag size = [Bag size − item size tobe placed]
= [15−1] = 14

* next we can place 1st item into the Bag. $x_1=1$
we can place complete item into the Bag.
R.B.S = [14−2] = 12

* next we can place 6th item into the Bag. $x_6=1$
we can place Complete item into the Bag
R.B.S = [12−4] = 8

25-01-2022
JOB SEQUENCING WITH DEADLINE

Consider there is n number of jobs are there for execution. Each job has to be executed by its deadline then only we will get the profit otherwise no profit.

Only one machine(processor) is available, only one job is executed at a time by its deadline.

Job i has integer deadline $d_i>0$ and profit is $p_i>0$

There is n number of jobs n1,n2,n3,n4 and deadline for each job is d1,d2,d3,d4 and profits of each and every job is p1,p2,p3,p4

We will obtaining the feasible solutions by the fallowing rule
1.each job takes one unit of time
2.each job has to be executed before the deadline or it deadline
Ex: p1-2days
  P2-2days
now first day p1 is executed(before it deadline)
Next day p2 is executed (at its deadline)
3.goal is to schedule the jobs to maximize the total profit.

Ex: the jobs are n=4, (p1,p2,p3,p4)=(100,10,15,27)
(d1,d2,d3,d4)=(2,1,2,1)

S.no	Feasible solution	Processing sequence	profit
1	(1)	(1)	100
2	(2)	(2)	10
3	(3)	(3)	15
4	(4)	(4)	27
5	(1,2)	(2,1)	100+10=110
6	(1,3)	(1,3) or(3,1)	100+15=115
7	(1,4)	(4,1)	100+27=127
8	(2,3)	(2,3)	10+15=25
9	(2,4)	Not a feasible solution	-
10	(3,4)	(4,3)	15+27=42

If 2nd job and 4 th job are coming for execution both are having same deadline(1,1) in this case the processor can't execute the jobs so then no profit.

1st and 2nd jobs are coming for execution with deadline(2,1) now the processor can execute the sequence(2,1) now the profit is 110

1st and 3rd jobs are coming for execution with deadline(2,2) now the processor can execute the sequence (1,3) or(3,1) niw the profit is 115

1st and 4th jobs are coming for execution with deadline(2,1) now the processor can execute the sequence (4,1) now the profit is 127.

2nd and 3rd jobs are coming for execution with deadline(1,2) now the processor can execute the sequence (2,3) now the profit is 25.

Feasible solutions are={(1)(2)(3)(4)(1,2)(1,3)(1,4)(2,3)(3,4)}

Optimal solution={(1,4)}
Not feasible feasible={(2,4)}

Ex: n=7 ,(p1,p2,p3,p4,p5,p6,p7)=(100,50,20,18,30,40,60)
(d,1,d2,d3,d4,d5,d6,d7)= (2,4,3,1,3,1,1)

Sol:

S.no	Feasible solution	Processing sequence	profit
1	(1)	(1)	100
2	(2)	(2)	50
3	(3)	(3)	20
4	(4)	(4)	18
5	(5)	(5)	30
6	(6)	(6)	40
7	(7)	(7)	60
8	(1,2)	(1,2) or(2,1)	100+50=150
9	(1,3)	(1,3)or (3,1)	100+20=120
10	(1,4)	(4,1)	100+18=118
11	(1,5)	(1,5)or(5,1)	100+30=130
12	(1,6)	(6,1)	100+40=140
13	(1,7)	(7,1)	100+60=160
14	(2,3)	(2,3)or(3,2)	50+20=70
15	(2,4)	(4,2)	50+18=68
16	(2,5)	(2,5) or(5,2)	50+30=80
17	(2,6)	(6,2)	50+40=90
18	(2,7)	(7,2)	50+60=110
19	(3,4)	(4,3)	20+18=38
20	(3,5)	(3,5) or(5,3)	20+30=50
21	(3,6)	(6,3)	20+40=60
22	(3,7)	(7,3)	20+60=80
23	(4,5)	(4,5)	18+30=48
24	(4,6)	Not a feasible solution	-
25	(4,7)	Not a feasible solution	-

26	(5,6)	(6,5)	30+40=70
27	(5,7)	(7,5)	30+60=90
28	(6,7)	Not a feasible solution	-
29	(1,2,7)	(7,1,2)	100+50+60=210
30	**(1,2,7,5)**	**(7,1,5,2)**	**100+50+60+30=240**

(4,6) is not feasible solution why because both are having highest dead line 1 so processor can't execute the job then there is no profit.

(4,7) is not feasible solution why because both are having highest dead line 1 so processor can't execute the job then there is no profit.

(6,7) is not feasible solution why because both are having highest dead line 1 so processor can't execute the job then there is no profit.

When 4 jobs are coming at the same time with the pair(1,2,7,5) we will the maximum profit 240 so that input pair is the optimal solution.

Optimal solution ={(1,2,7,5)}

Feasible solutions={(1)(2)(3)(4)(5)(6)(7)(1,2)(1,3),(1,4)
(1,5)(1,6)(1,7)(2,3)(2,4)
(2,5)(2,6)(2,7)(3,4) (3,5)(3,6)(3,7)(4,5)(5,6)(5,7)(1,2,7)(1,2,7,5)}

Not feasible
solution={(4,6)(4,7)(6,7)(1,4,6)(1,6,7)(4,6,7)(1,4,6,7)(1,2,4,6)}

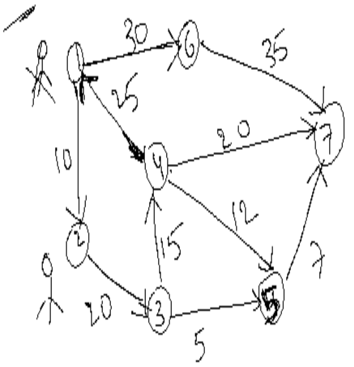
SINGLE SOURCE SHORTEST PATH PROBLEM

Graph is used to represent the distance between two cities. Everyone is moving from one city to another city with the shortest path.

G(v,e) in a graph the starting vertex (vo) is represent source vertex and the last vertex (vn)is represent as destination vertex.

We can find out the shortest path from source vertex to destination.

G1 Consider the graph G as



sol initiall $S[\,] = 0$

vortex1: $S[1]=1$, means we are visited First vortex, from First vertex we can Find out the distance to all other vortices in graph. $dist = 0$

$\{1,2\} = 10$ ✓

$\{1,3\} = \infty$

$\{1,4\} = \infty$

$\{1,5\} = \infty$

$\{1,6\} = 30$

$\{1,7\} = \infty$

Among all the possibility 10 is the shorted distance, so we can move from $1 \to 2$

path $= 1 \to 2$

distance $= 10$

vertex2: $S[2]=1$, means we are in vertex 2 from vortex-2 we can find out the distance to all other vertices in a graph

$\{1,2,3\} = 10+20 = 30$

$\{1,2,4\} = \infty$

$\{1,2,5\} = \infty$

$\{1,2,6\} = \infty$

$\{1,2,7\} = \infty$

Among all the possibility 30 is shortest distance so we can move from $2 \to 3$

path $= 1 \to 2 \to 3$

path = 1→2→3

Distance = 10+20 = 30

vertex 3: S[3]=1 means we are in vertex-3. now we can find out distance to all other vertices in a graph.

{1,2,3,4} = 30+15 = 45

{1,2,3,5} = 30+5 = 35 ✓

{1,2,3,6} = ∞

{1,2,3,7} = ∞

Among all possibilities 35 is the shortest distance so we can visit 5th vertex

path = 1→2→3→5

Distance = 10+20+5 = 35

vertex 5: S[5]=1, means we are in vertex-5, Now we can find out the distance to all other vertices in a graph.

{5,6} = ∞

{5,7} = 7

{5,4} = ∞

Among all possibilities 7 is the shortest distance so we can visit vertex 7

Now we are moving from source vertex-1 to destination vertex 7,

Shortest path = 1→2→3→5→7 =

Distance = 10+20+5+7 = 42

Algorithm:

```
Algorithm ssp(i,n,cost,dist)
{
//S is used to store all the visited vertices in a graph
//source vertex is u and destination vertex v

For i:= 1 to n do
{
S[i]:=0//initially

Dist:= cost[u,v];
}
S[i]:=1;//visited ith vertex and put it in to S
Dist[i]:=0.0
```

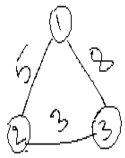
```
For i:=2 to n-2 do
{
Dist[v]:=min{dist[i]}
S[v]:=1 //put in S
//update all the distance values of the other nodes in a graph//

If(dist[u]+cost(u,v)<dist[v])
Dist[v]:=dist(u)+cost(u,v)
}
}
```

Minimum cost spanning trees

Spanning tree: a tree which includes all the vertices in a graph but without forming a cycle or circuit this is called a spanning tree. Graph contains more number of spanning trees.

Spanning trees mainly used in implementing efficient routing algorithms, and also network designing. Circuit designing.

Ex: if you consider a graph which contains n vertices then now many spanning trees are to be constructed. N pow n-2 spanning trees are to be constructed.



* If there is n vertices are there in the graph then n^{n-2} spanning Trees are to be constructed.
* $n=3$ [If 3 vertices are there] then how many spanning Trees are to be constructed
  $n^{n-2} \Rightarrow 3^{3-2} \Rightarrow 3^1 \Rightarrow 3$ spanning Trees
* consider all the vertices in a graph then we can select the edges.

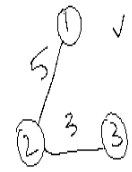


fig 1

Cost = 5+3 = 8

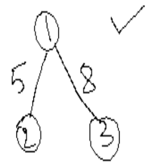


fig 2

Cost = 5+8 = 13

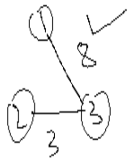


fig 3

Cost = 8+3 = 12

Among all the spanning Trees one of the spanning Tree will produce minimum cost so that spanning Tree minimum cost spanning Tree.

Suppose if there is 4 vertices are there then

* If there is 4-vertices are there then

$n^{n-2} \Rightarrow 4^{4-2} \Rightarrow 4^2 \Rightarrow 16$ spanning Trees are constructed.

* If there is 5-vertices are there then

$n=5$,

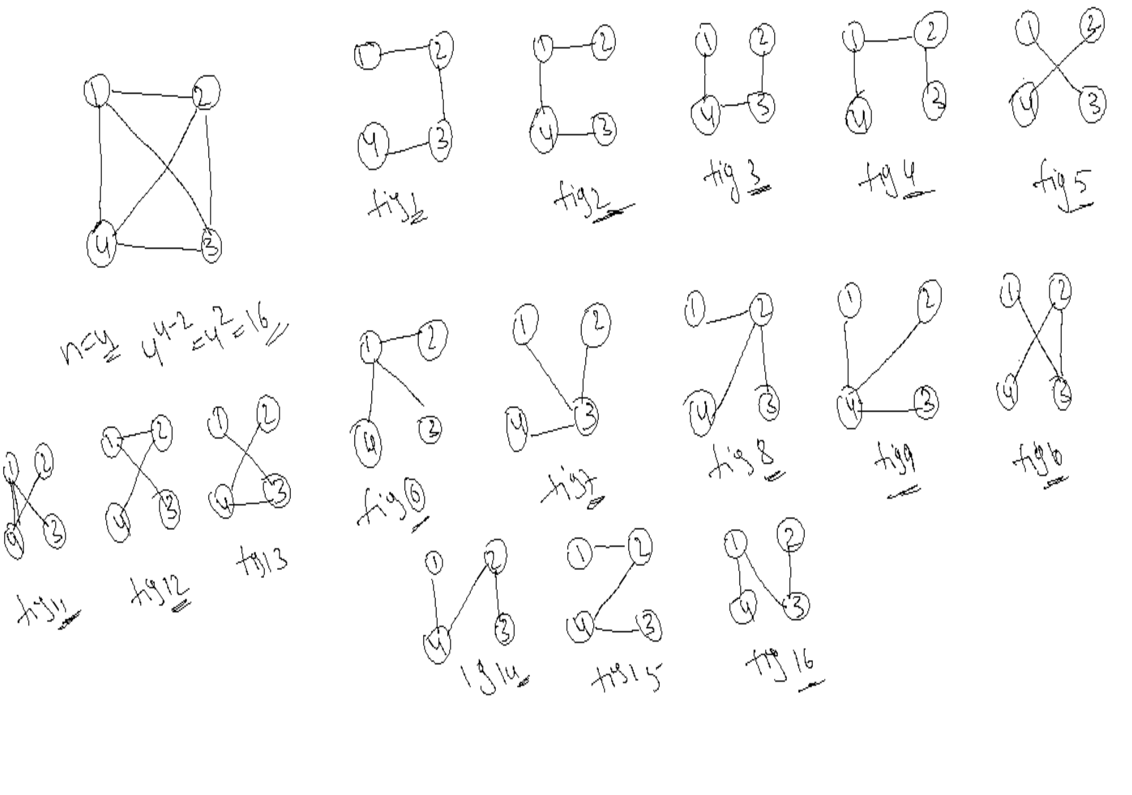
$n^{n-2} = 5^{5-2} = 5^3 \Rightarrow 125$ spanning Trees constructed.

* If $n=8$

$n^{n-2} = 8^{8-2} \Rightarrow 8^6 \Rightarrow \ldots\ldots$

So It is very Difficult to Construct and also it is a Time Taking process. To avoid these problem they are introducing Two algorithms.

1. prim's Algorithm
2. Kruskal's Algorithm.

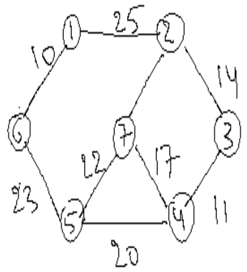


Prim's Algorithm:

In prim's algorithm we can consider all the vertices in a graph then we can select an edge with minimum cost.

The algorithm is proceeding by selecting adjacent edges with minimum cost. But that should forming a cycle.

G:- Consider the graph G.

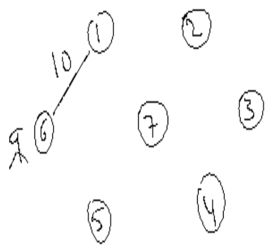


Find the minimum cost spanning Tree using prim's Algorithm

Sol Step1:- First we will consider All the vertices in a graph.

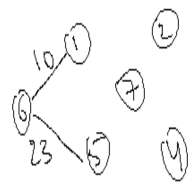
Total cost = 0

Step2:- we are in vertex-1, The adjacent vertices are 1→2, 1→6. 1→2 cost is 25, and 1→6 cost is 10. So 10 is minimum cost edge Now we can select an edge 1→6.



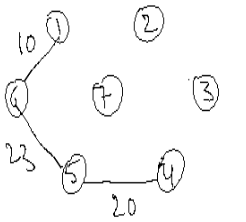
Total cost = 10

Step3:- The adjacent vertices of vertex-6 is 6→5. 6→5 cost is 23. Now we can select an edge 6→5.



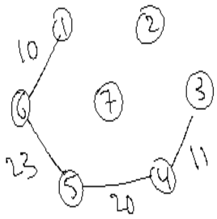
Total cost = 10+23 = 33

Step 4:- Adjacent vertices of 5th vertex are $5 \to 4$, $5 \to 7$. $5 \to 4$ cost is 20, $5 \to 7$ cost 22. Among these two 20 is minimum cost. so we can select an edge $5 \to 4$.



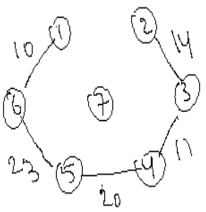
Total cost $= 10 + 23 + 20$
$= 53$

Step 5:- Adjacent edges of 4th vertex are $4 \to 7$, $4 \to 3$. $4 \to 7$ cost is 17, $4 \to 3$ cost is 11. 11 is the minimum cost so we select an edge $4 \to 3$.

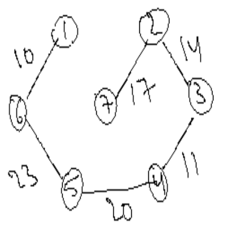


Total cost $= 10 + 23 + 20 + 11$
$= 64$

Step 6:- Adjacent edges of 3rd vertex are; $3 \to 2$
$3 \to 2$ cost is 14. so we can select an edge $3 \to 2$



Step 7:- Adjacent edges of 2nd vertex are $2 \to 1$, $2 \to 7$. If we select $2 \to 1$ edge it is forming cycle so this edge not possible. $2 \to 7$ cost is 17. so we can select an edge $2 \to 7$.

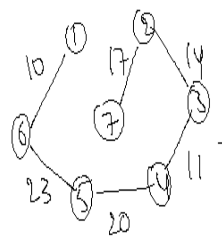


Total cost $= 10 + 23 + 20 + 11 + 14 + 17$
$= 95.$

Step 8:- Adjacent edges of 7th vertex. $7 \to 5$, $7 \to 4$
$7 \to 5$ and $7 \to 4$ is not a feasible solutions it is forming a cycle

minimum cost $= 10 + 23 + 20 + 11 + 14 + 17$
$= 95$

minimum cost spanning Tree is



$n = 7 \Rightarrow n^{n-2} = 7^{7-2} = 7^5 = 16,807$
Spanning Trees are possible

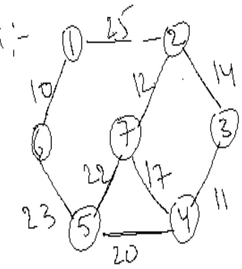
Kruskal's algorithm:

In prims algorithm we can select only adjacent edges in a graph with minimum cost.

But in kruskals algorithm we need not to fallow it adjacent edges in graph we can select any edge in a graph with minimum cost.

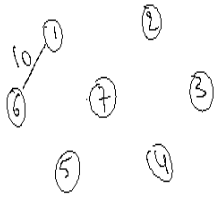
In kruskals algorithm first we can consider all the vertices in a graph, after than we can select an edge with minimum cost.

G1:-



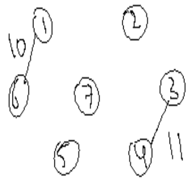
Sol step1:- Consider all the vertices in a graph

step2:- select an edge with minimum cost anywhere in the graph. 10 is the minimum cost then we can select 1-6 edge.



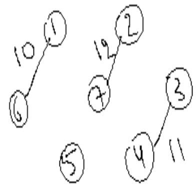
Total cost = 10

step3:- Select an edge with minimum cost, 11 is the next minimum cost edge then select 3-4 edge



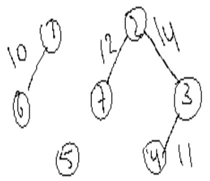
Total cost = 10+11=21

Step4:- select an edge with minimum cost, next 12 is the minimum cost edge, So we can select 2-7 edge.



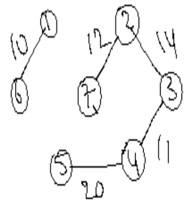
Total cost = 10+11+12
= 33

Step5:- select an edge with minimum cost next 14 is the minimum cost, So we can select 2-3 edge.



Total cost = 10+11+12+14
= 47

Step6:- select an edge with minimum cost, next 17 is the minimum cost, If we select 7-4 it is forming a cycle so it is not possible then discard.

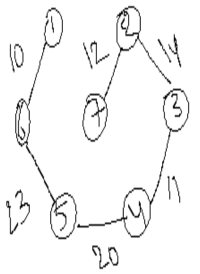


next minimum cost is 20 then we can select an edge 5-4

Total cost = 10+11+12+14+20 = 67

Step7:- select an edge with minimum cost, next 22 is the minimum cost, But if we select 5-7 it is forming a cycle so discard that edge.

next minimum cost is 23, then select an edge 6-5

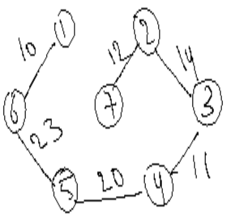


Total cost = 10+11+12+14+20+23

= 90

Step 8:- select an edge with minimum cost, next 25 is the minimum cost {1-2}, it is forming a cycle then Discard that edge

∴ Final minimum cost spanning Tree is



cost = 10+11+12+14+20+23

= 90

Difference between prims and kruskal algorithm

Prims algorithm	Kruskal algorithm
It is using vertex	It is using edges
Select any root vertex in a graph	Select shortest edge in a graph
Select next minimum cost edge which is connected to that vertex	Select the next minimum cost edge in a graph
Used to construct minimum cost spanning trees	Used to construct minimum cost spanning trees
Generate the minimum cost spanning tree starting from the root node.	Generate the minimum cost spanning tree starting from a least cost edge.

Prims algorithm:

In 1930 by vojtech jarnik , rediscoveres in 1957 by Robert c.prime

1.consider all the vertices in a given graph
2.select any root vertex
3.find out the adjacent vertices of that vertex, and find the minimum cost edges among the adjacent vertices, and select the unvisited vertex which is adjacent to visited vertices with minimum cost.
4. repeat the step-3 until all vertices are visited with MST.
5.if any minimum edges forming a cycle then discard that pair. And select next minimum cost edge in a graph.

Algorithm:

```
Algorithm prims(e,v,n,t,cost)
{
//E is the set of edges in a graph G.
//cost[1:n] is the cost of adjacent matrix. Cost[i,j] is either positive
number or infinite.

Let(k,l) be an edge of minimum cost in E
Mincost:=cost[k,l]
T[1,1]:=k;
T[1,2]:=l;
For i:=1 to n
If(cost[i,l]<cost[i,k]) then near[i]:=l
Else near[i]:=k;
Near[k]:=near[l]:=0;
For i:=2 to n-1 do
{
T[i,1]:=j;
T[i,2]:=near[j];
Mincost:=mincost+cost[j,near[j]];
Near[j]:=0;
For k:=1 to n do //update near[]
If((near[k]=0)and (cost[k,near[k]]>cost[k,j]))
Then near[k]:=j;
}
Return mincost;
}
```

Kruskal algorithm:

In 1956 written by joseph kruskal

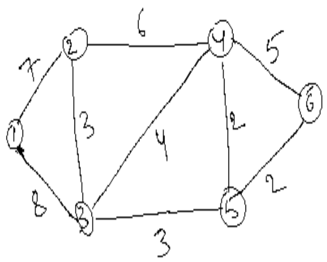
1.consider all the vertices in a graph
2.select any minimum cost edge in a graph
3.select next minimum cost edge in a graph
4.repeat step-3 until all vertices are visited in a graph
5.if any minimum cost edge forming a cycle then discard that edge, then select next minimum cost edge in a graph

Algorithm:

```
Algorithm kruskal(E,cost,n,t)
{
//E is the set of edges in graph G, graph has n vertices
//construct the heap out of the edge costs using heapify
For i:=1 to n do
Parent[i]:=-1;
//each vertex has different set
I:=0; mincost;=0.0;
While((i<n-1)and (heap not empty)) d0
{
Delete a minimum cost edge(u,v) from the heap
J:=find(u);
K:=find(v);
If(j not equal k) then
{
I:=i+`;
T[i]:=u;
T[1,2]:=v;
Mincost:=mincost+cost[u,v];
Union(j,k);
}
IF(i not eqal to n-1) then
Write("no spanning tree");
```

Else

Return mincost;
}



$G(V,E)$ V - no. of vertices in a graph

E - no. of edges "

$G'(V',E')$ V' - no. of vertices in a spanning Tree

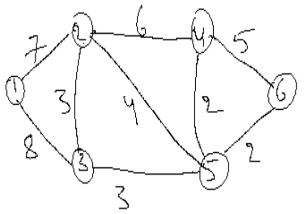
E' - no. of edges in a "

$V = V'$ [should be same vertices in both graph & spanning Tree]

$E \subset E'$

$E' = |V| - 1$

Ex



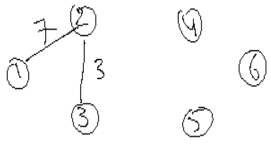
Sol **Step1:-** First consider All the vertices in a graph

Step2:- Root vertex is 1. Findout the adjacent vertices of 1. $1 \to 2$, $1 \to 3$,

$1 \to 2 = 7$, $1 \to 3 = 8$, Now 7 is the minimum cost then Slect $1 \to 2$ edge.

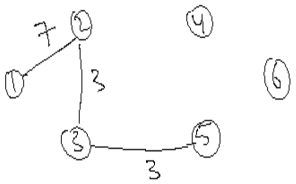
unvisited edge = $1 \to 3$

Step3: Find the adjacent vertices of 2. $2 \to 4 = 6$, $2 \to 5 = 4$, $2 \to 3 = 3$. And also we can consider unvisited vertex [Already visited vertex] $1 \to 3 = 8$
3 is the minimum cost then select $2 \to 3$ edge

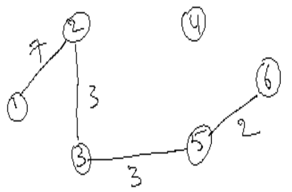


unvisited edges = $1 \to 3$
$2 \to 4$
$2 \to 5$

Step4:- Find the adjacent vertices of 3. $3 \to 5 = 3$
And also consider unvisited vertices [Already visited vertices]
$1 \to 3 = 8$, $2 \to 4 = 6$, $2 \to 5 = 4$. Among all the possibilities 3 is the minimum cost so select $3 \to 5$ edge.



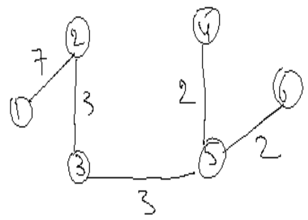
Step5: Find the adjacent vertices of 5. $5 \to 4 = 2$, $5 \to 6 = 2$.
And also consider unvisited vertices from Already visited vertices, $1 \to 3 = 8$, $2 \to 4 = 6$, $2 \to 5 = 4$,
Now 2 is the minimum cost then Select either $5 \to 4 = 2$ (or) $5 \to 6 = 2$



Step 6:- Find the adjacent vertices of 6, $6 \to 4 = 5$, and also consider unvisited vertices from already visited vertices

$1 \to 3 = 8$, $2 \to 4 = 6$, $2 \to 5 = 4$, $5 \to 4 = 2$

Among all the possibilities 2 is the minimum cost then select $5 \to 4 = 2$ edge



Total minimum cost = $7+3+3+2+2 = 17$

08-02-2022

DISJOINT SETS

Two sets are said to be disjoint sets if they have no common elements. Or a pair of sets which does not have any common elements is called disjoint sets.

Ex: A={1,2,3} B={4,5,6} these are disjoint sets, A∩B=φ null set or void set

A={1,2,3} B={3,4,5} these are not disjoint sets. A∩B=3

*when the intersection of two sets is a null or empty then they are called disjoint sets.

Ex: A={1,2,3} B={4,5,6} these are disjoint sets, A∩B=φ null set or void set

*Disjoint set data structure also called a union –find data structure (or) merge-find data structure. Data structure that stores a disjoint sets.

Suppose we can take 3 sets s1,s2,s3
S1={1,2,3,4} s2={5,6,7} s3={8,9,10}

Sn: s1={1,2,3,4} s2={5,6,7}, s3={8,9,10}

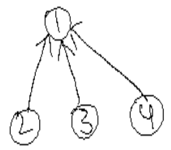
These sets are Represented in the form of Tree

each set is Represented in the form of Tree.

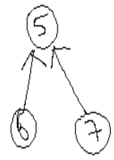
* we have linked the nodes from children node to the parent node. rather than from usual method of linking from parent to child

* we can Take one of the element is the parent node

S1={1,2,3,4} =>

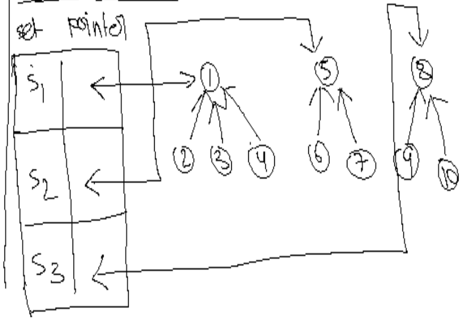


S2={5,6,7} =>



S3={8,9,10} =>

data Representation



We are performing two operations on these sets

1.disjoint set union: if si and sj are two disjoint sets then their union of siỤsj={all the elements of si and sj}

Ex: si={1,2,3,4} sj={5,6,7} then

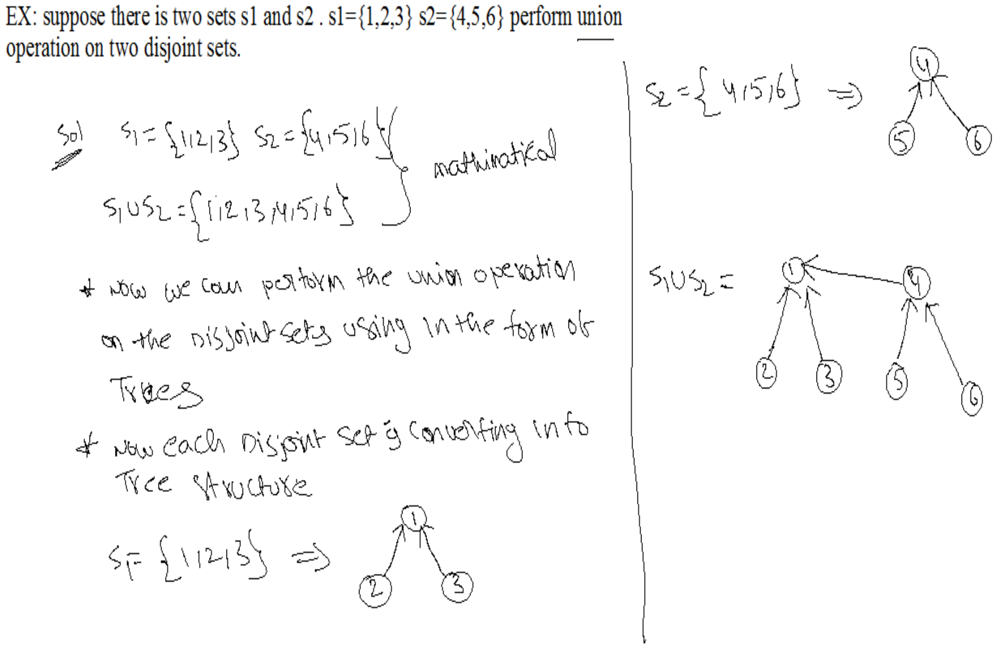
siỤsj=(1,2,3,4,5,6,7}

2.find(i): given the element i, find the set containing i. We can find the particular element ‘i’ is in the set.

Ex: s1={1,2,3,4}

Then find(4) means the element 4 is in the set.

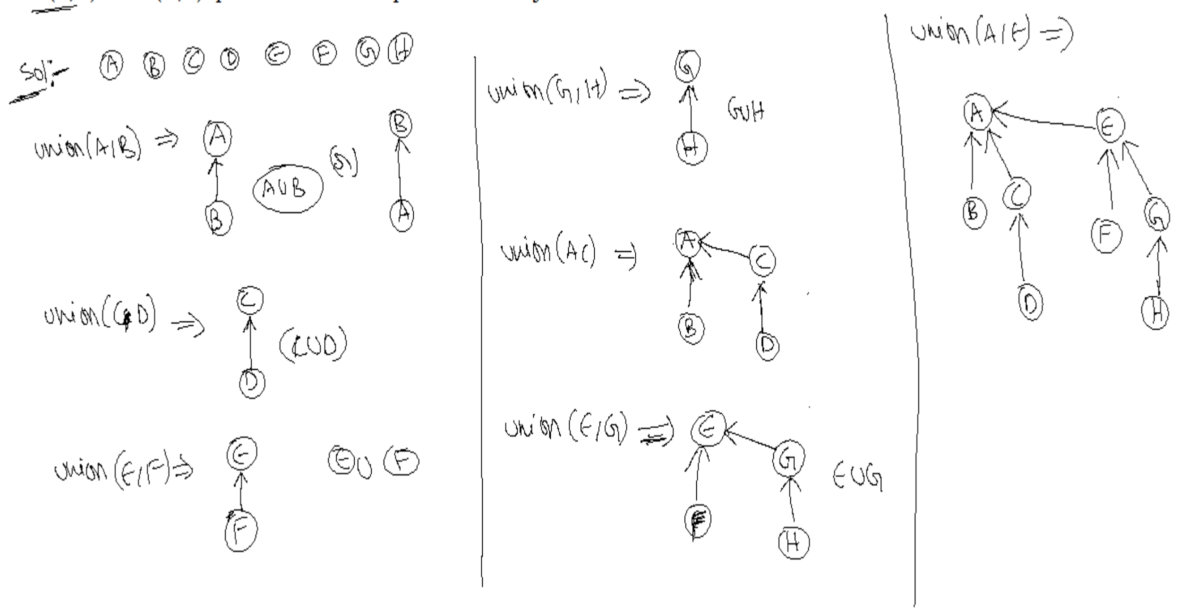
EX: suppose there is two sets s1 and s2 . s1={1,2,3} s2={4,5,6} perform union operation on two disjoint sets.



Ex2: union(A,B) union(C,D) union(E,F) union(G,H) union(A,C) union(E,G) union(A,E) perform the union operations on disjoint sets.

Sol:

Ex2: union(A,B) union(C,D) union(E,F) union(G,H) union(A,C) union(E,G) union(A,E) perform the union operations on disjoint sets.



UNION AND FIND ALGORITHMS

Union Algorithm:

Union(i,j): it means the elements or set i and the elements of set j are combined.

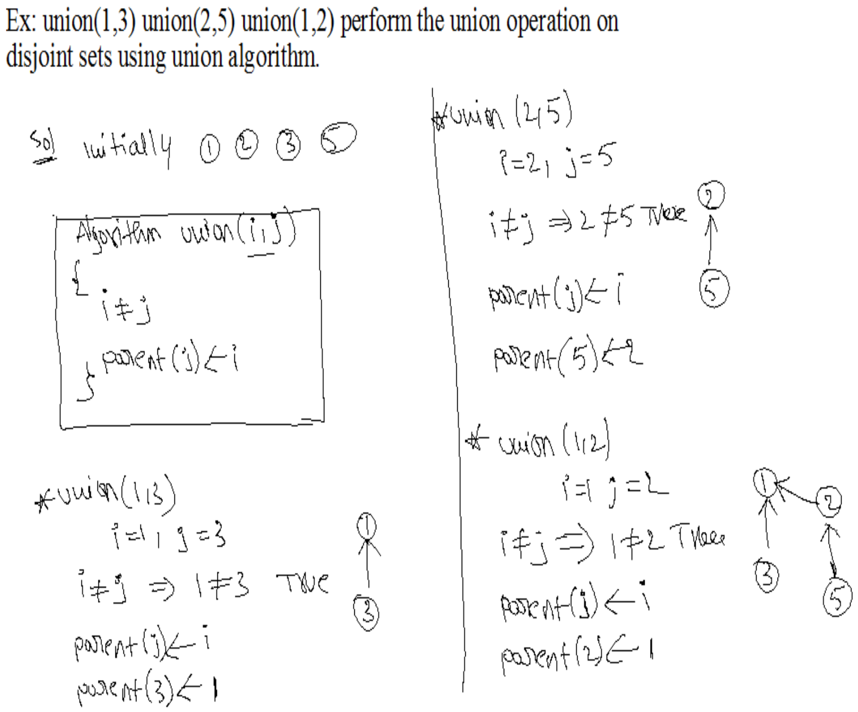
If we want to represent the union operation in the form of tree structure, we can take i is the root node and j is the children(or) j is the parent node and i is the children.

Algorithm:

Algorithm union(i,j)
{
I≠j
Parent (j)←-i
}

Algorithm union(i,j)
{
Parent(i)←--j
}

Ex: union(1,3) union(2,5) union(1,2) perform the union operation on disjoint sets using union algorithm.



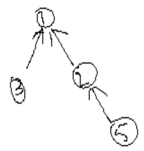
Find(i): it find the root of the ith node.

Algorithm:

Algorithm find(i)
{
// find the root for the tree containing elements
J←-i

While(parent(j)>0)
{
J<-parent(j);
}
Return j;
}

Ex



Find(5)

Sol Find(5) ⇒ i=5 we need to find out the Root of 5th node.

```
Algorithm Find(i)
{ j←i
  while(parent(j)>0)
    j←parent(j)
} return(j);
```

i=5

→ j←i ⇒ j←5

→ while(parent(5)>0)

while(2>0) True

j←parent(j)

j←parent(5)

j←2

* j=2

while(parent(j)>0)

while(parent(2)>0)

while(1>0) True

j←parent(j)

j←parent(2)

j←1

* j=1

while(parent(j)>0)

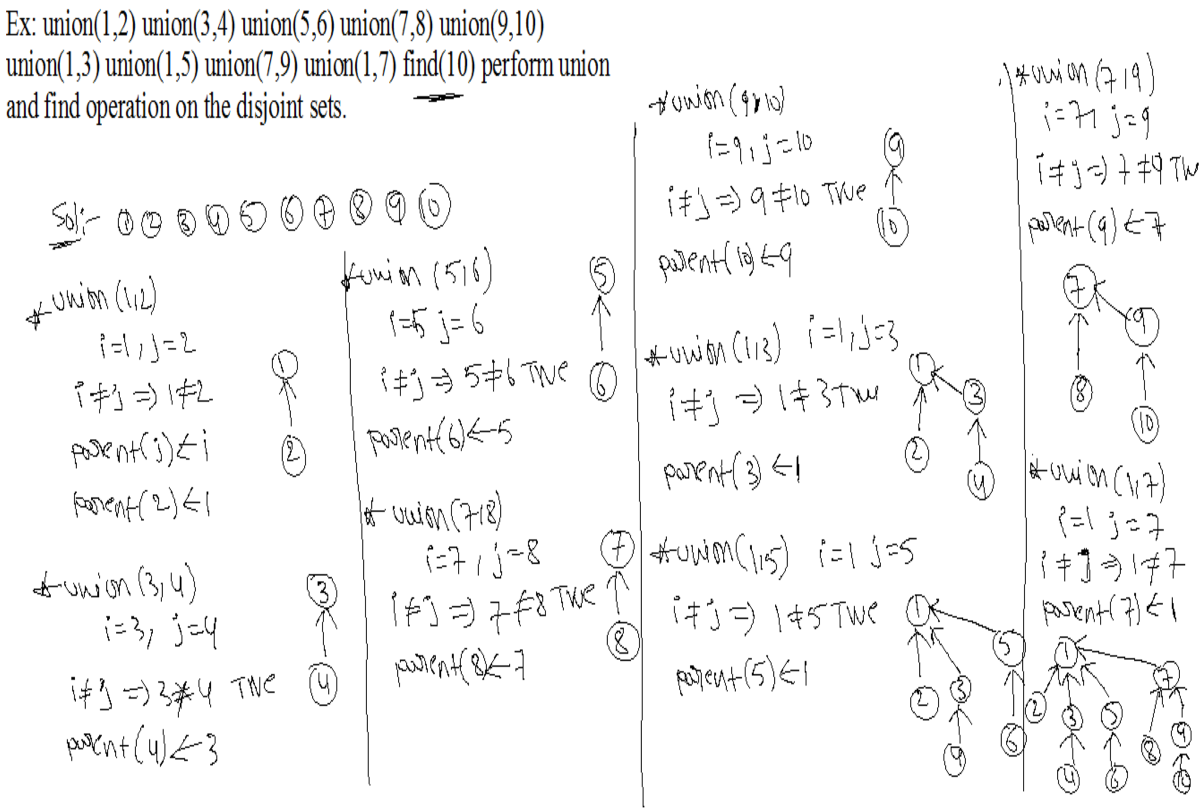
while(parent(1)>0)

while(0>0) False

return(j) ⇒ return(1)

∴ 1 is the Root of the 5th node

Ex: union(1,2) union(3,4) union(5,6) union(7,8) union(9,10) union(1,3) union(1,5) union(7,9) union(1,7) find(10) perform union and find operation on the disjoint sets.



Spanking trees:

A spanning tree is a tree which includes all the vertices in a graph but without forming a cycle is called spanning tree.

Spanning trees are very important in designing efficient routing algorithms, and circuit designing.

If there is N number of vertices are there then the possible spanning trees are n pow(n-2) spanning trees are possible.

Graph Traversals Algorithms

Inorder to visit the vertices in a graph basically there is two algorithms are there for graph search

1.Breadth First search
2.Depth First search

1.Breadth First Search(BFS): we can visit the vertices in a graph using BFS.

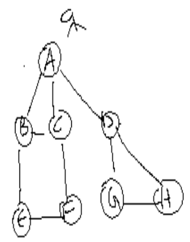
Algorithm:

1.first select an unvisited vertex in a graph, and visit it. That will be the root of the BFS tree.

2.find the unvisited neighbours of the root node, and visit all the unvisited neighbours of the root node.

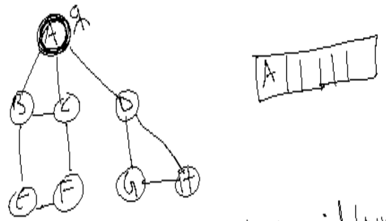
3.repeat step-2 for all the unvisited vertices

4.repeat from step 1 until no more vertices are remaining.



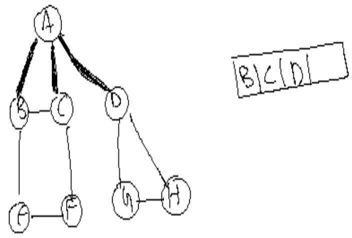
using BFS

Sol step1:- select the unvisited vertex A, and visit it. that is the Root node And placed in a queue.



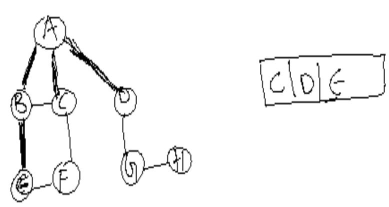
Step2:- Find the unvisited neighbours of A node. B, C, D are unvisited neighbours of A node. Then visit All the unvisited neighbours And placed in a queue. The A is deleted from the queue.

A



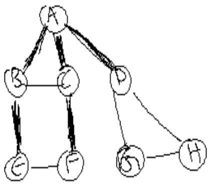
Step 3:- Find the unvisited neighbours of B node E is the unvisited neighbour, and visit it, placed in a queue, now B is Deleted from the queue

A, B



Step4:- Find the unvisited neighbours of C node.
F is the unvisited neighbors of C node. then visit it and placed in a queue. C is deleted from the queue

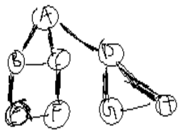
A,B,C



D	E	F

Step5:- Find the unvisited neighbours of D node.
G,H are unvisited neighbours. visit it. and placed in a queue. D is deleted from the queue

A,B,C,D



E	F	G	H

Step6:- Find the unvisited neighbours of E node.
no unvisited neighbours then Delete from the queue.

A,B,C,D,E

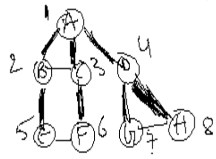
F	G	H

Step7:- Find the unvisited neighbours of F node
no unvisited neighbours then delet it from queue.

G	H

A,B,C,D,E,F

Step8:- Find the unvisited neighbours of G,H.
no unvisited neighbours then delete from que



empty queue

A,B,C,D,E,F,G,H

Depth First Search(DFS)

1.select an unvisited vertex in a graph and that is current node.

2.find the unvisited neighbours of the current node and visit. And that node will be the new current node.

3.if the current node has no unvisited neighbours then backtrack to its parent node. And find the new unvisited neighbours.

4.repeat the step-2 and 3 until no more unvisited neighbours can be visited.

5.repeat from step1 for the remaining vertices.

Algorithm:

Step-1:put the starting vertex in to the stack mark it as visited. And display it.

Step-2: if(stack[top]has unvisited neighbours)then

{

Visit the unvisited vertex and mark it as visited

Push it in to the stack

Display it.

}

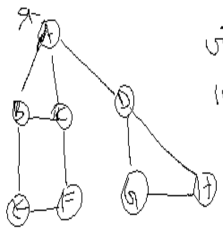
Else //if current node has no unvisited neighbours

{

Pop the top of the element from the stack

}

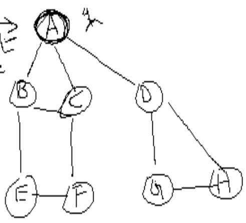
Step3: repeat step2 until stack is empty



visit the vertices in a graph using DFS

Sol step1:- select unvisited vertex A in a graph and mark it.

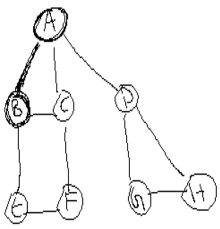
current node



A display

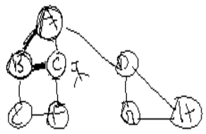
Step2:- Find the unvisited neighbors of A node. B, C, D are unvisited neighbours of A node. we can visit node B and that will be the new current node

A, B



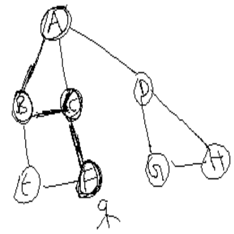
Step3:- Find the unvisited neighbours of B node. C, E are the unvisited neighbours. we can visit C node and mark C node and display it

A, B, C



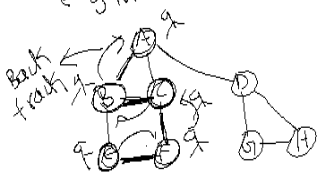
Step4:- Find the unvisited neighbours of C node. F is the unvisited neighbour of C node. Then visit it

A, B, C, F



Step5:- Find the unvisited neighbors of F node. E is the unvisited neighbour and visit it.

A, B, C, F, E



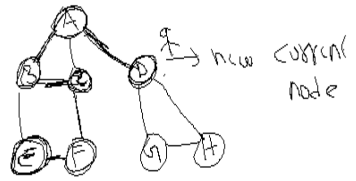
Step6:- Now we are in E node. Find the unvisited neighbours of E node. There is no unvisited neighbours of E node. Then Backtrack to its parent node and searching for new unvisited neighbours.

E parent F → no unvisited neighbours

F parent C → " "

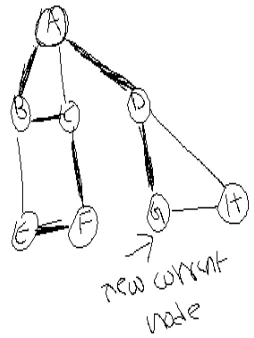
C parent B → " "

B parent A → D is the unvisited neighbour then visit it

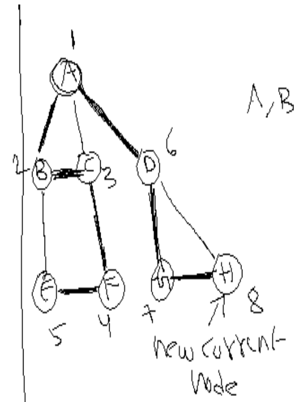


Step7:- Find the unvisited neighbors of D node. G, H are unvisited neighbours. we can visit G node that is new current node.

A, B, C, F, E, D, G



Step8:- Find the unvisited neighbors of G node. H is the unvisited neighbour then visit it that will be the new current node. And Display it



A, B, C, F, E, D, G, H

DFS number

BFS number

Step9:- There is no unvisited neighbours so we can visit all the vertices in a graph.

A, B, C, F, E, D, G, H

Connected and Bi-connected components:

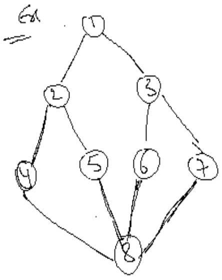
Connected components:
If a graph G is connected undirected graph then all the vertices of graph G will get connected. After converting the graph in to a spanning all the vertices are available in a spanning tree and every very vertex is connected.

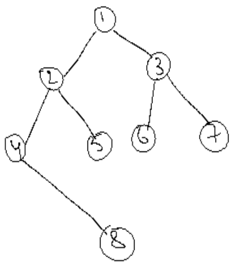
A spanning tree is obtained by using a breadth first search that tree is called breadth first(BFS) spanning tree.

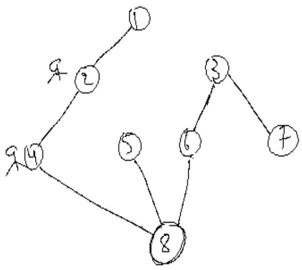
A spanning tree is constructed using Depth first search that tree is called Depth first(DFS) spanning tree.

Ex:

Ex







Sol

BFS spanning Tree

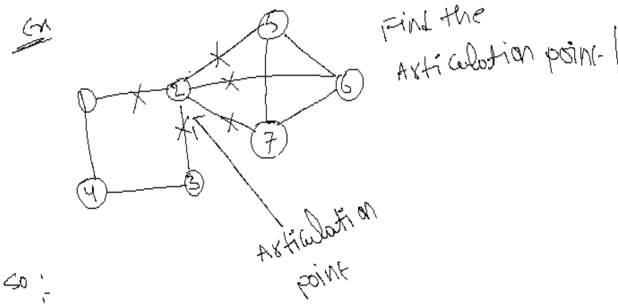
DFS spanning Tree

Bi-connected Components:
In this we need to discuss about two important concepts
1.Articulation point
2.Bi-connected graph

1.Articulation point: if you want delete any vertex in graph then the graph is dividing in to two or more number of pieces then the vertex is called articulation point.
(0r)

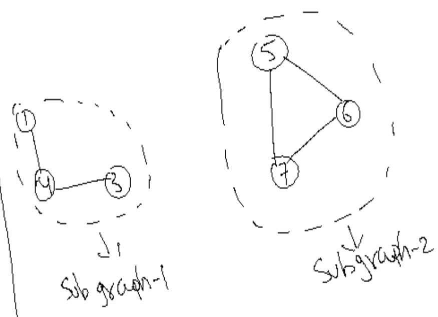
Which vertex is dividing the graph in to two or more number of pieces or parts then that vertex is called articulation point.

Ex:

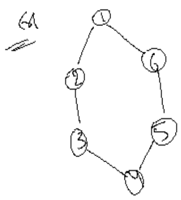


So :-

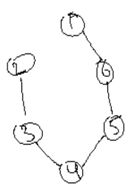
If you delete vertex 2 in a graph, then the corresponding edges are also delete.
Then graph is dividing into two parts
so. vertex-2 is the Articulation point

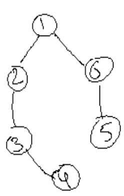


2.Bi-connected graph: if you delete any edge in a graph even thought the graph is connected that graph is called Bi-connected graph.
Ex:



Sol:- delete edge (1,2) delete edge (4,5)





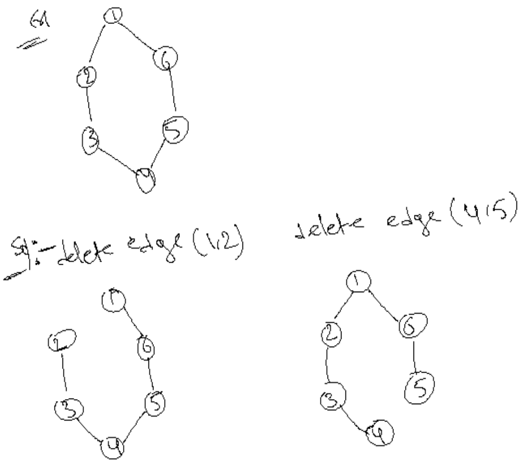
Identification of articulation point:

1.The easiest method is to remove a vertex and its corresponding edges one by one from the graph then the graph is divided or not.

2.Another method is to use DFS in order to find the articulation point. After converting the graph in to DFS that is called DFS spanning tree.

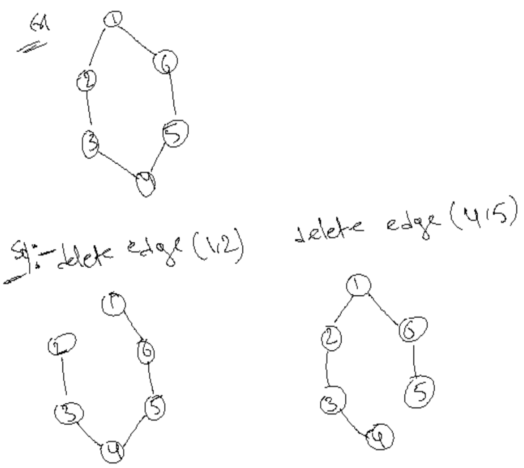
3.After converting in to DFS spanning tree. We are giving the numbers outside the every vertex. These numbers are called DFS numbers numbers(dfn). This numbers indicateds the order of visiting the vertices.

4.while building the DFS spanning tree
a.Tree edge: it is an edge in a DFS tree.



Tree edges-(1,6)(6,5)(5,4)(4,3)(3,2)

b.Back edge: it is an edge(u,v) which is not in DFS tree.



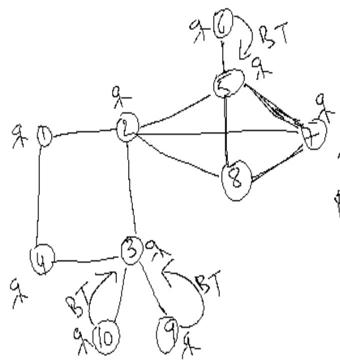
Back edges-(1,2) (4,5)

Articulation point can be identified using

Low[u]=min{dfn[u],min{low[w]/w is child} min{dfn[w]/w is back edge}}

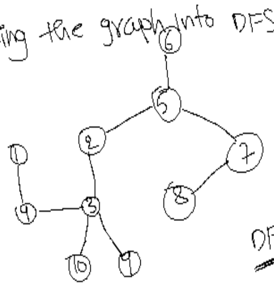
Afther than we can check

Low[w]>=dfn[u] if the condition is true then the vertex is articulation point.



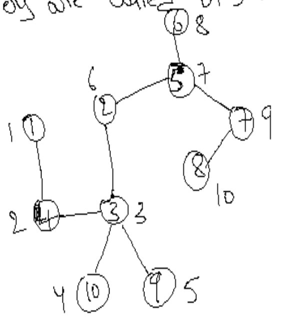
Find the Articulation point in this graph

1st step:- Converting the graph into DFS spanning Tree.



DFS spanning tree

step2:- After Converting into DFS spanning Tree. we are giving the numbers to every vertex. that is order of visiting the vertex. These numbers are called DFS number (DFN)



step3:- Find the Articulation point

$low[u] = \min\{dfn[u], \min\{low[w]\} \min\{dfn[w]\}\}$

step4:- vertex-1 $u=1$

$low[u] = \min\{dfn[u], \min\{low[w]\}, \min\{dfn[w]\}\}$

$low[1] = \min\{dfn[1], \min\{low[4]\}, \min\{dfn[2]\}\}$

$= \min\{1, low[4], 6\}$

$low[1] = 1$ [nothing is less than 1]

vertex-2 $u=2$

$low[2] = \min\{dfn[2], \min\{low[5]\}, \min\{dfn[1], dfn[7], dfn[8]\}\}$

$= \min\{6, low[5], \min\{1, 9, 10\}\}$

$= \min\{6, low[5], 1\}$

$low[2] = 1$

vertex-3 $u = 3$

$Low[3] = \min\{dfn[3], \min\{Low[10], Low[9], Low[2]\}, \text{no Backedges}\}$

$= \min\{3, \min\{Low[10], Low[9], 1\}, -\}$

$= \min\{3, 1, -\}$

$Low[3] = 1$

vertex 4 $u = 4$

$Low[4] = \min\{dfn[4], \min\{Low[3]\}, \text{no Backedges}\}$

$= \min\{2, 1, -\}$

$Low(4) = 1$

vertex-5: $u = 5$

$Low[5] = \min\{dfn[5], \min\{Low[6], Low[7]\}, \min\{dfn[8]\}\}$

$= \min\{7, \min\{Low[6], Low[7]\}, \min\{10\}\}$

$Low(5) = \min\{7, \min\{8, 6\}, 10\} \Rightarrow \min\{7, 6, 10\} = 6$

$Low[5] = 6$

vertex 6: $u = 6$

$Low[6] = \min\{dfn(6), \text{no childrens}, \text{no Backedges}\}$

$= \min\{8, -, -\}$

$Low(6) = 8$

vertex 7 $u = 7$

$Low[7] = \min\{dfn(7), \min\{Low[8]\}, \min\{dfn(2)\}\}$

$= \min\{9, Low[8], 6\} \Rightarrow \min\{9, 6, 6\}$

$Low[7] = 6$

vertex 8:- $u = 8$

$Low[8] = \min\{dfn(8), \text{no children}, \min\{dfn[2], dfn[5]\}$

$= \min\{10, -, \min\{6, 7\}\}$

$Low[8] = \min\{10, -, 6\} = 6$

vertex-9 $v=9$

$low[9] = \min\{dfn[9], -, -\}$
$= \min\{5, -, -\}$

$Low[9] = 5$

vertex-10 $v=10$

$Low[10] = \min\{dfn[10], -, -\}$
$= \min\{4, -, -\}$

$Low[10] = 4$

∴ Low values are

$Low[1:10] = \{1, 1, 1, 1, 6, 8, 6, 6, 5, 4\}$

∴ no we can find out the articulation point

$low[w] \geq dfn[v] \Rightarrow dfn[v] \leq low[w]$

Here 3 is the articulation point

$low[10] \geq dfn[3]$ $low[9] \geq dfn[3]$

$4 \geq 3$ True $5 \geq 3$ True

so 3 is the Articulation point

vertex-2 $v=2$

$Low[5] \geq dfn[2]$

$6 \geq 6$ True

2 is Articulation point

vertex 5 $v=5$

$Low[6] \geq dfn[5]$

$8 \geq 7$ True