Algorithm

for n ← rows(W) ← No of vertices

D(0) ← W # Input

for k ← 1 to n

  for i ← 1 to n

    for j ← 1 to n

      $D_{ij}^{k} \leftarrow \min(D_{ij}^{k-1}, D_{ik}^{k-1} + D_{kj}^{k-1})$

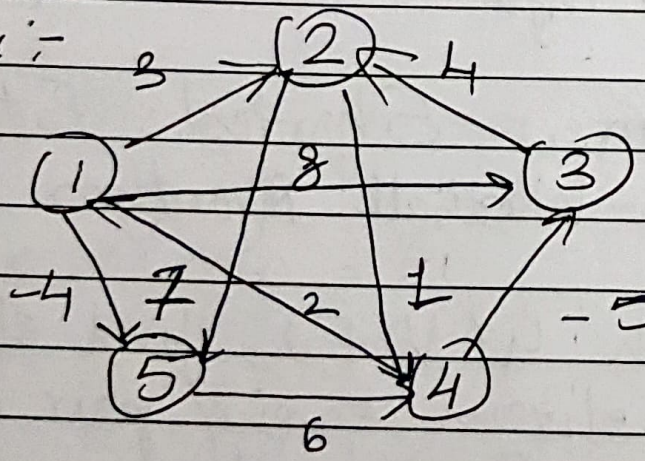
return D

Ex:-

$D_0 = W_0$

Time complexity = $O(n^3)$

Ex:-



Input- $D_0 = W_0$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

Shortest

Iteration 1 :- D_1 → Distance from to all nodes via node 1

$D^{(1)}$ =

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

Expl:-

i = 4, j = 2 To go from node 4 to 2 via 1, total cost
= min(∞, 2 + 3 = 5) = 5

Wardy i = 4, j = 3
Wt = min(-5, 2+8=10) = -5

Iteration k=2

$D^{(2)}$ =

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

k = 3

$D_3 =$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

k = 4

$D_4 =$

	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

k = 5

Output = D_5

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

(2) Longest Common Subsequence

Substring v/s subsequence

Substring :- Substring of a string S is another string S' that occur in S and all the letters are contiguous in S

Eg S :- Helloworld

S' :- Woold.

Subsequence :- Subsequence of string S is another string S' that occurs in S and all letters need not be contiguous

Eg S :- Helloworld ; S' = Herd

Longest Common Subsequence :-

Given two strings A of length x and B of length y, find the longest common subsequence between both strings from left to right

Brute Force Method :-

S1 :- Find all subsequences of A, (2^x)

S2 :- Find each subsequence find whether it is subsequence of B, $(yx2^x)$

S3 :- Choose the longest common subsequence

Time complexity: $yx2^x$ or $O(n2^m)$

Dynamic Programming Method :-

Two Strings:

$X = \{ x_1, x_2, x_3 \dots x_m \}$

$Y = \{ y_1, y_2, y_3 \dots y_n \}$

First compare x_m and y_n, if matched, find subsequence in remaining string & append x_m.

If $x_m \neq y_n$,

Remove x_m from X & find LCS from x_1 to x_{m-1} and y_1 to y_n

Remove y_n from Y & find LCS from x_1 to x_m and y_1 to y_{n-1}

$c[i,y]$ = length of LCS of x_i & y_y

$$c[i,y] = \begin{cases} 0 & \text{if } i=0 \text{ or } y=0 \\ c[i-1, y-1]+1 & \text{if } i,y>0 \text{ \& } x_i = y_i \\ \max(c[i,y-1], c[i-1,y]) & \text{if } i,y>0 \text{ \& } x_i \neq y_i \end{cases}$$

```
LCS-Length(X, Y)
m ← length(X)
n ← length(Y)
for i ← 1 to m
    do c[i,0] ← 0
for y ← 0 to n
    do c[0,y] ← 0
for i ← 1 to m
    for J ← 1 to n
        if xi = yJ
            c[i,y] ← c[i-1, y-1] + 1
        else if b[i,y] ← "↖"
        else if c[i-1, y] >= c[i, y-1]
            c[i,y] ← c[i-1, y]
            b[i,y] ← "↑"
        else
            c[i,y] ← c[i, y-1]
            b[i,y] ← "←"
return c & b
```

Initial call i, j is x.length, y.len

```
PRINT_LCS (b, X, i, y)
if i=0 & y=0
    return
if b[i,y] = "↖"
    PRINT_LCS (b, X, i-1, y-1)
    print xi
else if b[i,y] = "↑"
    PRINT_LCS(b, X, i-1, y)
else PRINT_LCS(b, X, i, y-1)
```

PAGE No.
DATE / /

Total Runtime = O(mn)

Ex:- LCS of ACBCF & ABCDAF

			A	C	B	C	F
	A	0	0	0	0	0	0
A	B	0	1↖	1←	1←	1←	1←
B	C	0	1↑	1↑	2↖	2←	2←
C	D	0	1↑	2↖	2↑	3↖	3←
D	A	0	1↑	2↑	2↑	3↑	3↑
A	F	0	1↖	2↑	2↑	3↑	3↑
F		0	1↑	2↑	2↑	3↑	4↖

Follow as per arrows, print on (↖)

F C B A → Reverse ⇒ ABCF

PAGE NO.
DATE / /

(3) Single Source Shortest Path

For graph $G = (V, E)$, to find the shortest path from given source vertex $s \in V$ to every other vertex $v \in V$

→ Bellman-Ford Algorithm

Given a weighted, directed graph $G = (V, E)$ with source s, the algo returns a boolean value indicating whether or not there is a -ve weight cycle reachable from the source.

```
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]
        d[v] > d[u] + w(u, v)
            return FALSE
return TRUE
```

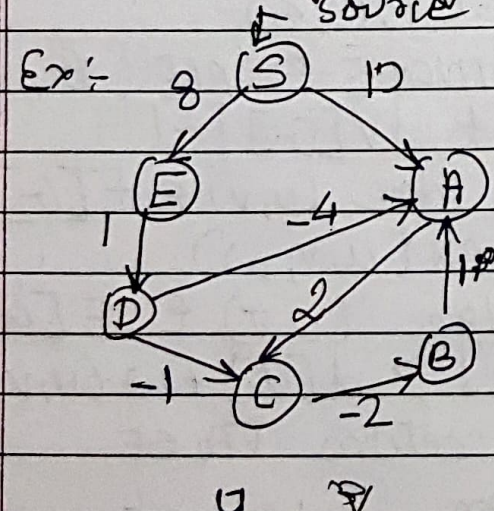
PAGE No.
DATE / /

INITIALIZE-SINGLE-SOURCE (G, S)
for each vertex $v \in V[G]$
do $d[v] \leftarrow \infty$
$\pi[v] \leftarrow NIL$
$d[s] \leftarrow 0$

RELAX (u, v, w)
if $d[v] > d[u] + w(u, v)$
$d[v] \leftarrow d[u] + w(u, v)$
$\pi[v] \leftarrow u$

Time complexity :- For V vertices & E ~~edge~~ edge $O(VE)$

Ex :-



u	v	w
S	A	10
S	E	8
A	C	2
E	D	1
B	A	1
D	A	-4
D	C	-1
C	B	-2

Initialization

	S	A	B	C	D	E
d	0	∞	∞	∞	∞	∞
π	N	N	N	N	N	N

PAGE No.
DATE / /

I st Iteration. (Go through u, v, w list)

	S	A	B	C	D	E
d	0	10 5	6	8 9 10 12 12	8+1=9	8
Π	N	S D	C	A D	E	S

Keep looping through u, v, w list till n-1 or two iterations give same result.
While creating u, v, w list always start with S & go to next connected vertex.

II

	S	A	B	C	D	E
d	0	5	5	7	9	8
Π	N	D	C	A	E	S

III

	S	A	B	C	D	E
d	0	5	5	7	9	8
Π	N	D	C	A	E	S

Output	
S → A	S $\xrightarrow{8}$ E $\xrightarrow{1}$ D $\xrightarrow{-4}$ A
S → B	S $\xrightarrow{8}$ E $\xrightarrow{1}$ D $\xrightarrow{-4}$ A $\xrightarrow{2}$ C $\xrightarrow{-1}$ B
S → C	S → E → D → A → C
S → D	S → E → D
S → E	S → E

PAGE No.
DATE / /

(4) Multistage graph
Can use Single source shortest path.

(5) 0/1 Knapsack Problem

```
DP_Knapsack (V, p, w, M, n)
for i ← 1 to n
    V[i, 0] ← 0
for y ← 0 to M
    V[0, y] ← 0
for i ← 1 to n
    for y ← 1 to M
        if w[i] ≤ y
            V[i, y] ← max{V[i-1, y],
                          p[i] + V[i-1, y - w[i]]}
        else
            V[i, y] ← V[i-1, y]
```

Complexity = O(n × M)
n ← No. of elts
M ← Capacity of knapsack
wi - Weight of item
Pi - Price of items

Maximize price within given capacity

Ex :- $M = 7$, $n = 4$

$W = \{1, 3, 4, 5\}$, $P = \{1, 4, 5, 7\}$

Item		1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0
$W_1 = 1, P_1 = 1$	0	1	1	1	1	1	1	1
$W_2 = 3, P_2 = 4$	0	1	1	[4]	5	5	5	5
$W_3 = 4, P_3 = 5$	0	1	1	4	5	6	6	[9]
$W_4 = 5, P_4 = 7$	0	1	1	4	5	7	8	[9]

Trace_Knapsack (W, V, M)

```
SW ← ϕ
SP ← ϕ
i ← n      j ← M
while (j > 0)
    if (V[i,j] == V[i-1,j])
        i ← i - 1
    else
        SW ← SW + W[i]
        SP ← SP + P[i]
        j ← j - W[i]
        i ← i - 1
```

For previous problem

SW	SP
—	—
4	5
3	4

$j = 7 - 4 = 3$

PAGE No.
DATE / /

7. Greedy Method

Make choice that looks best at that moment.

Sometimes works, sometimes doesn't

Properties :-

- Globally optimal soln is obtained from locally optimal soln
- Best choice in current situation selected.
- Optimal Substructure :- Problem has optimal substructure if an optimal soln to the problems is composed of optimal solns to subproblems.

(1) Single Source Shortest path
- Dijkstra's algorithm

All weights must be non-negative

S :- Set of vertices whose final shortest path weights have already been determined.

Q :- A min priority queue keyed by their distance values.

Repeatedly select vertex $u \in V - S$ (kept in Q, with minimum shortest

path and add u to S, & relax all edges leaving u.

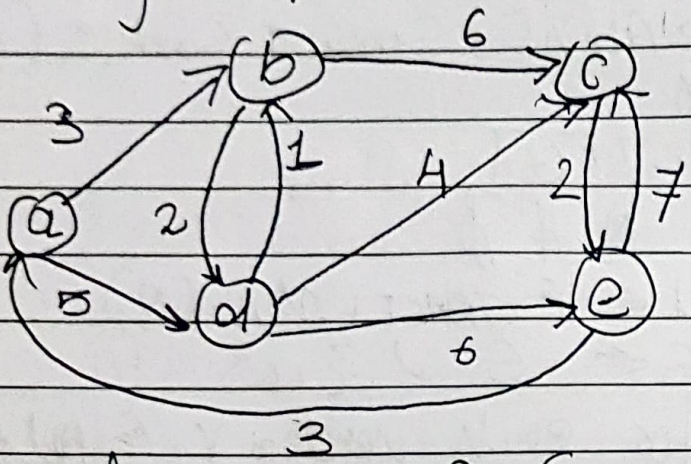
```
Dijkstra(G, w, s)
INITIALIZE_SINGLE_SOURCE(G, s)
S ← ∅
Q ← V[G]
while Q ≠ ∅
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u]
        RELAX(u, v, w)

INITIALIZE_SINGLE_SOURCE(G, s)
for each vertex v ∈ V[G]
    d[v] ← ∞
    π[v] ← NIL
d[s] ← 0

RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
    d[v] ← d[u] + w(u, v)
    π[v] ← u
```

Time complexity $O(V^2)$

Priority Queue Impl.	Time Complexity
Array	$O(V^2)$
Priority Queue	$O(V \log V)$
Binary Heap	$O(E \log V)$



$S = \phi$ $\quad Q = \{a, b, c, d, e\}$

	a	b	c	d	e
d	0	∞	∞	∞	∞
π	N	N	N	N	N

$u = a$ $\quad Q = \{b, c, d, e\}$ $\quad S = \{a\}$ $\quad V = b, d$

	a	b	c	d	e
d	0	3	∞	5	∞
π	N	a	N	a	N

$u = b$ $\quad Q = \{c, d, e\}$ $\quad S = \{a, b\}$ $\quad V = \{c, d\}$

	a	b	c	d	e
d	0	3	9	5	∞
π	N	a	b	a	N

$u = d$; $Q = \{c, e\}$, $S = \{a, b, d\}$; $V = \{b, c, e\}$

	a	b	c	d	e
d	0	3	9	5	11
π	N	a	b	a	d

$u = c$, $Q = \{e\}$, $S = \{a, b, c, d\}$; $V = \{e\}$

	a	b	c	d	e
d	0	3	9	5	11
π	N	a	b	a	d

$u = e$, $Q = \phi$, $S = \{a, b, c, d, e\}$

PAGE No.
DATE / /

② Knapsack (Fractional)

- Sort items in decreasing c_i / w_i
- Add items till no more items or sack is full (exceeds W)
- If sack is not full, fill with fraction of next unselected item.

```
Knapsack(C, W, M, X, n)
for i ← 1 to n
    X[i] ← 0
RC ← M
for i ← 1 to n
    if W[i] > RC
        break
    X[i] ← 1
    RC ← RC - W[i]

if i ≤ n then
    X[i] ← RC / W[i]
```

M - Knapsack capacity
W[i] - weight of item i
X[i] - 0/1 array telling if item i is selected or not.

Time Complexity :- O(nlogn)

PAGE NO.
DATE / /

(3) Spanning Tree

- For an connected undirected graph, spanning tree of the graph is subgraph which is a tree and connects all vertices.

for $G(V, E)$, $T = (V, E')$ is a spanning tree iff T is a tree.

Minimum Spanning Tree (MST) is a spanning tree with weight less than or equal to every other spanning tree of that graph.

Algorithms

3.1 Kruskal's Algorithm

- Set A is ~~post~~
- Edge A is added to A if it is always a least weighted edge in the graph that connects two distinct components.

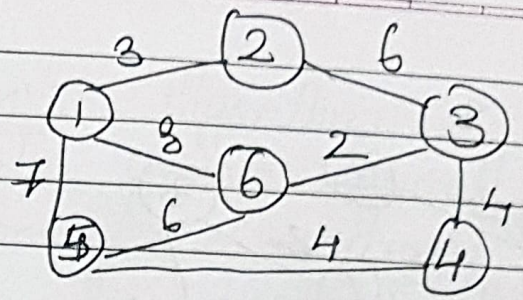
MST_Kruskal (G, w)
A ← ϕ
for each vertex v ∈ V[G]
    Make_Set (v)
Sort edges of E by increasing wt w
for each edge (u,v) ∈ E
    if Find_Set(u) ≠ Find_Set(v)
        A ← A ∪ {(u,v)}
        Union (u,v)
Return A

Make_Set (x) :- Creates set of single elt x

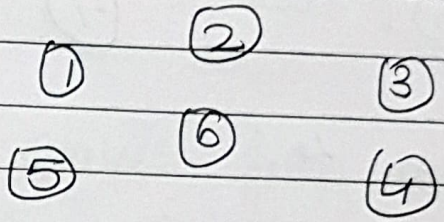
Find_Set (x) - determine if two vertices u & v belong to same tree by checking if Find_Set (u) ≠ Find_Set (v)

Union (x,y) - Unites the sets that contain x & y say Sx & Sy into a new set that is union of the two sets.

Analysis :- Avg Case :- O(E log E)



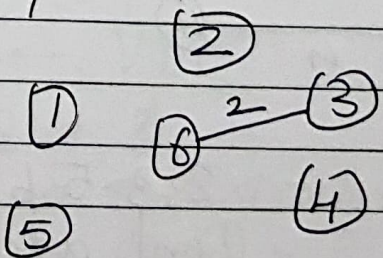
Disjoint Forest



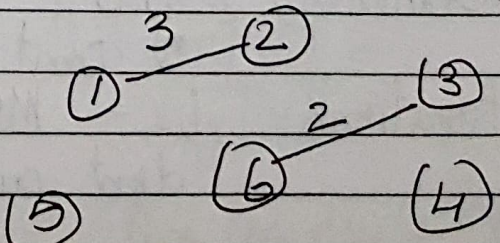
Sort Edges

Edge	wt
(3,6)	2
1,2	3
3,4	4
4,5	4
2,3	6
5,6	6
1,5	7
1,6	8

Iteration 1

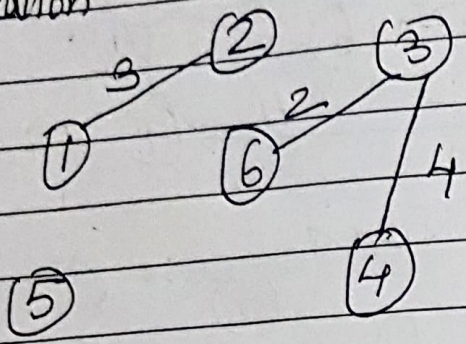


Iteration 2

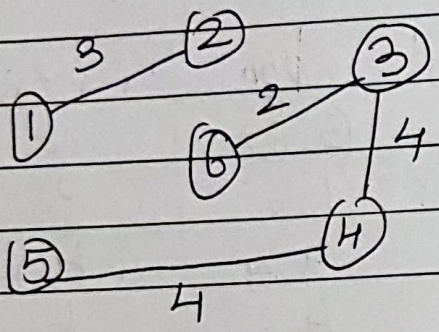


DATE / /

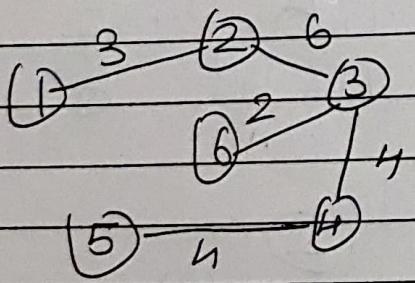
Iteration 3



Iteration 4



Iteration 5 :- (2,3) forms cycle? No, add



Iteration 6 :- 2,3 - 5,6 makes cycle? Yes
Dont add

Iteration 7 :- 1,5 - Ye makes cycle? Yes
Dont add

Iteration 8 1,6 - Makes cycle, ? Yes
dont add.

3.2 Prim's Algorithm

- MST grows "naturally" starting from an arbitrary root.
- Has the property that the edges in the set always form a single tree

Logic

- Tree starts from arbitrary root vertex r.
- Grow the tree until it spans all vertices in set V

Data Structure

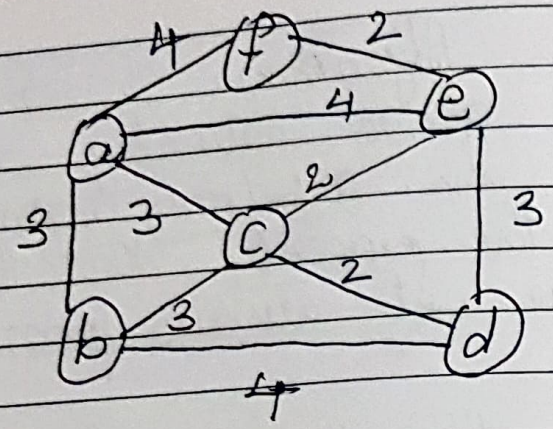
- A min. priority queue keyed by edge values.

```
                          ↓ - root vertex
MST_PRIM(G, w, r)
for each u ∈ V[G]
        key[u] ← ∞
        π[u] ← NIL
key[r] ← 0
Q      ← V[G]
while Q ≠ ∅
    do u ← Extract-Min(Q)
       for each v ∈ Adj[u]
           if v ∈ Q and w(u,v) < key[v]
                π[v] ← u
                key[v] ← w(u,v)
```

Analysis :- Avg complexity $O(E \log V)$

DATE / /

Ex



	a	b	c	d	e	f
key	0	∞	∞	∞	∞	∞
π	N	N	N	N	N	N

① $u = a$ $Q = \{b, c, d, e, f\}$

$v = b, c, e, f$

	a	b	c	d	e	f
key	0	3	3	∞	4	4
π	N	a	a	N	a	a

② $u = b$ $Q = \{c, d, e, f\}$ $v = c, d, e$ (nothing)

	a	b	c	d	e	f
key	0	3	3	4	4	4
π	N	a	a	b	a	a

③ $u = c$ $Q = \{d, e, f\}$ $v = a, b, e, d$

	a	b	c	d	e	f
key	0	3	3	2	2	4
π	N	a	a	c	c	a

④ $u = d \quad Q = \{e, f\} \quad V = b, e$

	a	b	c	d	e	f
key	∞	3	3	2	2	4
π	N	a	a	c	c	a

⑤ $u = e \quad Q = f \quad V = a, f$

	a	b	c	d	e	f
key	∞	3	3	2	2	2
π	N	a	a	c	c	e

⑥ $u = f \quad Q = \phi \quad V = a, e$