

Greedy MethodGeneral Method

Greedy Method is a design-technique, applicable to problems that have 'n' inputs and needs to obtain a subset that satisfies some 'constraints'.

- Any subset that satisfies these constraints is called a feasible solution.
- Feasible soln that either maximizes or minimizes the given objective solution is called optimal solution.

Methods:-Subset paradigm:-

- Algorithms work in stages, considering one i/p at a time.

At each step, a decision is made regarding whether a particular i/p is in an optimal soln.

- I/p's are considered in the order determined by some selection procedure, where selection proc is an optimization measure.
- If the inclusion of the next i/p into the partially constructed optimal soln will result in an infeasible soln, then this i/p is not added to the partial soln, otherwise it is added.

- Selection proc's optimization measure is mostly the objective func. $\therefore$

This version of the greedy technique is called the subset paradigm.

EX:- Knapsack problem, Job sequencing with deadlines, Minimum-cost spanning trees fit in subset paradigm.

Greedy method control abstraction for the subset paradigm.

Algorithm Greedy(a, n)

// $a[1:n]$ contains the 'n' inputs.

{
 $\text{soln} := \emptyset$ // Initialize the soln

 for $i=1$ to n do

 {
 $x_i := \text{Select}(a);$

 if Feasible(soln, x_i) then

$\text{soln} := \text{Union}(\text{soln}, x_i);$

 }
 }
 return soln;

Select(): - selects an i/p func from $a[]$ and removes it.

Selected i/p value is assigned to x_i .

Feasible(): Boolean-valued func that determines whether x_i can be included into the soln vector.

Union(): - combines x_i with the soln and updates the objective func.

2. Ordering paradigm:-

- Problems that do not call for the selection of an optimal subset, make decisions by considering the i/p's in some order.
- Each decision is made using an optimization criterion that can be computed using decisions already made.

This version of greedy method is called ordering paradigm.

Ex: Single source shortest path problem.

Ex 1 change Making:

A child buys a toy for ₹232 and gives ₹500 to the cashier.

Assume cashier has unlimited 100 notes, 10 notes, 50 notes, 5 rupee coins, 2 rupee coins & 1 rupee coin.

If the cashier wishes to return change using the fewest no. of coins.

In each stage, a ^{note or} coin is added to the change. This ^{note or} coin is selected using greedy criterion: At each stage increase the total amount of change constructed by as much as possible. The selected one should not cause the total amount of change given so far to exceed the final desired amount.

First	give	2	nos of	100s notes =	₹200
Second	"	1	no of	50 "	= ₹50
Third	"	1	"	10 "	= ₹10
Fourth	"	1	"	5 coin	= ₹5
Fifth	"	1	"	2 coin	= ₹2
					₹268

knapack Problem - 2 types of Problems - Fractional knapsack

Fractional knapsack problem

- Given 'n' objects

↳ a knapsack or bag.

- Object 'i' has weight w_i

- knapsack has a capacity 'm'.

- If a fraction x_i , $0 \leq x_i \leq 1$, of object 'i' is placed into the knapsack, then profit $p_i x_i$ is earned.

The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Since the knapsack capacity is 'm', total weight chosen should be at most m.

$$\text{formally, maximize } \sum_{1 \leq i \leq n} p_i x_i \quad \text{--- (1)}$$

$$\text{subject to } \sum_{1 \leq i \leq n} w_i x_i \leq m \quad \text{--- (2)}$$

$$\text{and } 0 \leq x_i \leq 1, \quad 1 \leq i \leq n \quad \text{--- (3)}$$

The profits and weights are positive numbers.

- Any set, $(x_1, \dots, x_n)$ is feasible solution if it satisfies (2) & (3)

- An optimal soln is feasible for which (1) is maximized.

- Consider the following instance of the knapsack problem:

$n=3$, $m=20$, $(p_1, p_2, p_3) = (25, 24, 15)$ and $(w_1, w_2, w_3) = (18, 15, 10)$

feasible solns

(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1. $(\frac{1}{2}, \frac{1}{3}, \frac{1}{4})$	16.5	24.25
2. $(1, \frac{2}{15}, 0)$	20	28.2
3. $(0, \frac{2}{3}, 1)$	20	31
4. $(0, 1, \frac{1}{2})$	20	31.5

$\Rightarrow$ Optimal solutions is $(0, 1, \frac{1}{2})$

Algorithm Greedy knapsack (m,n)

(3)

// $P[1:n]$ and $w[1:n]$ contain the profits and weights respectively

// of the n objects ordered such that $\frac{P[i]}{w[i]} \geq \frac{P[i+1]}{w[i+1]}$

// m is the knapsack size and $x[1:n]$ is the solution vector

{ for $i := 1$ to n do $x[i] := 0.0$ // Initialize x .

$U := m;$

for $i := 1$ to n do

{ if $(w[i] > U)$ then break;

$x[i] := 1.0$; $U = U - w[i];$

} if $(i \leq n)$ then $x[i] := \frac{U}{w[i]};$

}

Given Item	$\frac{P_1}{w_1}$	$\frac{P_2}{w_2}$	$\frac{P_3}{w_3}$
P_i	25	24	15
w_i	18	15	10
$\frac{P_i}{w_i}$	1.38	1.6	1.5

$(x_1, x_2, x_3) = (0, 0, 0)$

$m = 20, U = 20$

$1.6 > 1.5 > 1.38$

order becomes
i.e. x_2, x_3, x_1

step 1 Consider x_2

$w_2 = 15$

$w_2 \neq U$

$$\frac{P_i}{w_i} = \frac{24}{15}$$

$\Rightarrow x_2 = 1.0, U = 20 - 15 = 5$

step 2 Consider x_3

$w_3 = 10$

$w_3 > U$

$\Rightarrow x_3 = \frac{5}{10} = 0.5$

$\Rightarrow (x_1, x_2, x_3) = (0, 1, 0.5)$

Profit
 $\Rightarrow \sum P_i x_i$

$$= 25 \times 0 + 24 \times 1 + 15 \times 0.5$$

$$= 31.5$$

2. Find an optimal solution to the knapsack instance

$n=7$, $m=15$, $(p_1, p_2, \dots, p_7) = (10, 5, 15, 7, 6, 18, 3)$ and

$(w_1, w_2, \dots, w_7) = (2, 3, 5, 7, 1, 4, 1)$

→ 0/1 knapsack problem:-

Solve

$$\max \sum_{i=1}^n p_i x_i$$

subject to $\sum_{i=1}^n w_i x_i \leq m$

and $x_i = 0$ or 1 , $1 \leq i \leq n$

→ Solve the following 0/1 knapsack problem where $P = \{10, 5, 15, 7, 6, 18, 3\}$
 $p_1, p_2, p_3, p_4, p_5, p_6, p_7$

$(w_1, w_2, \dots, w_7) = (2, 3, 5, 7, 1, 4, 1)$, $M = 15$
 $w_1, w_2, w_3, w_4, w_5, w_6, w_7$

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (0, 0, 0, 0, 0, 0, 0)$

Acc to $\frac{p_i}{w_i}$ 'i' order $x_5 + x_1, x_6, x_3, x_7, x_2, x_4$
 Consider in non increasing order

i	$\frac{p_i}{w_i}$
1	5
2	$5/3 = 1.6$
3	3
4	1
5	6
6	$9/2 = 4.5$
7	3

Consider i's in order

Step 1 Consider $i=5, x_5$

i.e $\Rightarrow U = M = 15$

$U = 15 - 1 = 14$
 $\Rightarrow x_5 = 1$

Step 2 Consider $i=1$

$\Rightarrow U = 14 - 2 = 12$
 $\Rightarrow x_1 = 1$

Step 3 Consider $i=6$

$\Rightarrow U = 12 - 4 = 8$
 $\Rightarrow x_6 = 1$

Step 4 Consider $i=3$

$15 - 2 - 1 = 12$

Step 5

Consider $i=7$

$U = 3 - 1 = 2$
 $\Rightarrow x_7 = 1$

Step 6 Consider $i=2$

$U = 2 - 3 < 0$

$\Rightarrow$ This x_2 cannot be put in knapsack

Step 7

Consider x_4

$U = 2 - 7 < 0$

$\Rightarrow x_4 = 0$

soln $(0, 1, 0, 1, 1, 1, 1)$

Job Sequencing with deadlines:-

(4)

Given a set of 'n' jobs. Associated with job 'i' is an integer deadline $d_i \geq 0$ and a profit $P_i > 0$. For any job 'i' - the profit P_i is earned iff the job is completed by its deadline.

To complete a job, one has to process the job on the machine for one unit of time. Only one machine is available for processing jobs.

A feasible soln for this problem is a subset J of jobs such that each job in this subset can be completed by its deadline. The value of a feasible soln J is the sum of the profits of the jobs in J or $\sum_{i \in J} P_i$.

An optimal soln is a feasible soln with max value.

1. Let $n=4$, $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ and

$$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$$

feasible soln	processing sequence	value
1. (1, 2)	2, 1	110
2. (1, 3)	1, 3 or 3, 1	115
3. (1, 4)	4, 1	127
4. (2, 3)	2, 3	25
5. (3, 4)	4, 3	42
6. (1)	1	100
7. (2)	2	10
8. (3)	3	15
9. (4)	4	27

optimal soln
 $J = \{1, 4\}$ with
value 127.

High-level description of job sequencing algorithm:-

→ This algo constructs an optimal set J of jobs that can be processed by their due times.

Algorithm Greedy_Job(d, J, n)

// d is a set of jobs that can be completed by their deadline

{

$J := \{1\};$

for $i := 2$ to n do

{

if (all jobs in $J \cup \{i\}$ can be completed by their deadlines) then $J := J \cup \{i\};$

}

}

→ Greedy algorithm for sequencing unit time jobs with deadlines and profits.

Algorithm JS(d, j, n)

/* $d_i \geq 1, 1 \leq i \leq n$ are the deadlines, $n \geq 1$. The jobs are ordered such that $p[1] \geq p[2] \geq \dots \geq p[n]$. $J[i]$ is the i th job in the optimal soln. $1 \leq i \leq k$.

Also at termination $d[J[i]] \leq d[J[i+1]], 1 \leq i < k$

{

$d[0] := J[0] := 0;$ // Initialize

$J[1] := 1;$ // Include job 1

$k := 1;$

for $i := 2$ to n do

{ /* considers jobs in non increasing order of $p[i]$.
Find the position for 'i' & check feasibility of insertion */

```

x := k
while ((d[J[x]] > d[i]) and (d[J[x]] ≠ x))
do x := x - 1;

if ((d[J[x]] ≤ d[i]) and (d[i] > x)) then
{
    // Insert i into J.
    for q := x to (x+1) step -1
do J[q+1] := J[q];

    J[x+1] := i; k := k+1;
}
}
return k;
}

```

- $J[i]$, $1 \leq i \leq k$, is the set already included, such that $d[J[i]] \leq d[J[i+1]]$
- while loop determines where in J this job has to be inserted.
- J_0 is a fictitious job that allows easy insertion into pos 1.
- If job i is included into J , then jobs $J[q]$, $w < q \leq k$, has to be moved one position up in J . (while loop)
- i is inserted at pos $w+1$ if $d[i] > w$. This is verified in 'for' inner loop of 'if'

Time complexity of JS $\rightarrow \Theta(n^2)$

Ex. let $n=5$, $(p_1, \dots, p_5) = (20, 15, 10, 5, 1)$ and

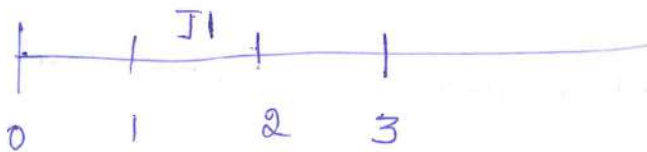
$$(d_1, \dots, d_5) = (2, 2, 1, 3, 3)$$

J	assigned slots	Job considered	action	Profit
$\emptyset$	none	J1	assign to $[1, 2]$	20
$\{1\}$	$[1, 2]$	J2	" $[0, 1]$	35
$\{1, 2\}$	$[0, 1], [1, 2]$	J3	cannot fit; reject	35
$\{1, 2, 4\}$	$[0, 1], [1, 2]$	J4	assign to $[2, 3]$	40
$\{1, 2, 4\}$	$[0, 1], [1, 2], [2, 3]$	J5	reject	40

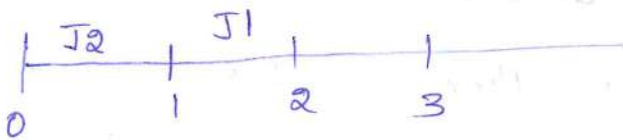
The optimal soln is $J = \{1, 2, 4\}$ with a profit of 40.

Explanation $\rightarrow$ All the profits are in nonincreasing order

consider J1, $d_1 = 2$

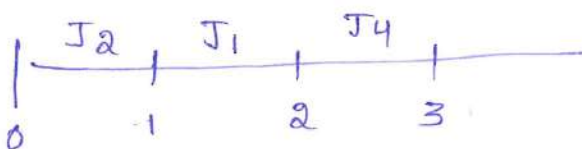


consider J2, $d_2 = 2$



consider J3 as $d_3 = 1$, it cannot be accomplished

consider J4 as $d_4 = 3$

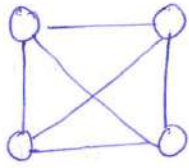


consider J5, $d_5 = 3$, so reject.

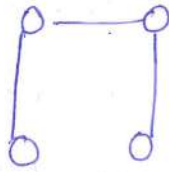
Minimum-cost spanning trees:-

(6)

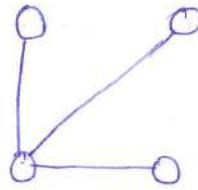
Let $G = (V, E)$ be an undirected connected graph. A subgraph $t = (V, E')$ of G is a spanning tree of G iff t is a tree.



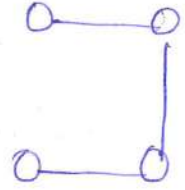
Complete graph
on 4 graphs



Spanning
tree 1



Spanning
tree 2



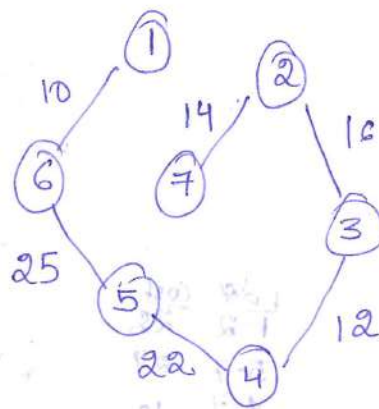
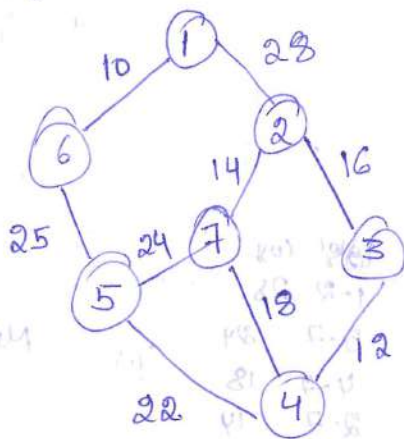
Spanning
tree 3 ...

Applications of spanning trees:-

1. They can be used to obtain an independent set of circuit equations for an electric n/w.

2. Min cost spanning tree:-
Spanning tree is a minimal subgraph G' of G such that $V(G') = V(G)$ and G' is connected. The spanning trees of G represent all feasible choices.

Assign weights to edges, weights represent the cost of construction, length of the link, & soon. Find a spanning tree of G with min cost.



1. Prim's Algorithm:-

- Builds the min cost spanning tree edge by edge. The next edge to be included must be chosen according to some optimization criterion.
- Simple criterion - choose an edge that results in a min increase in

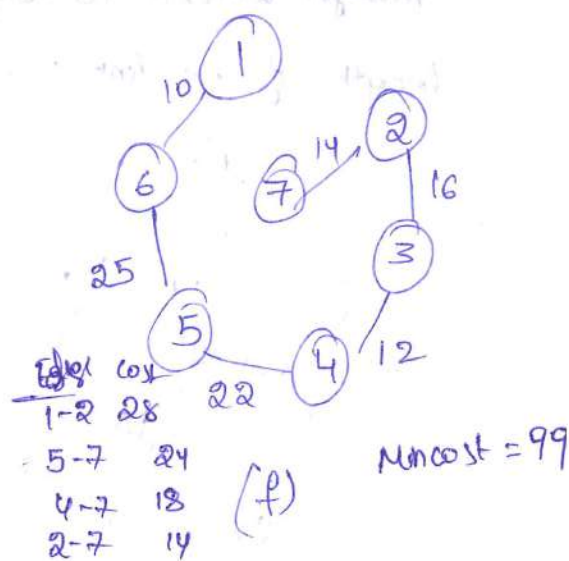
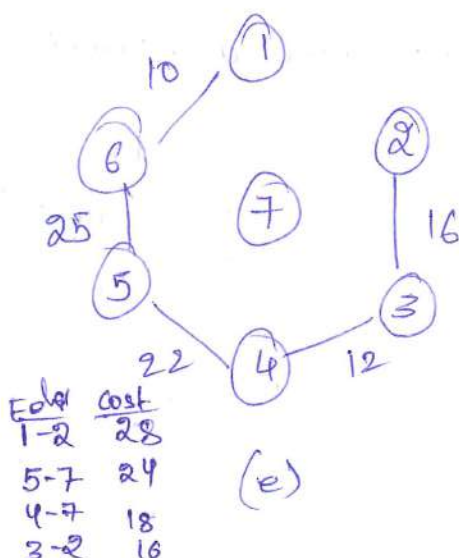
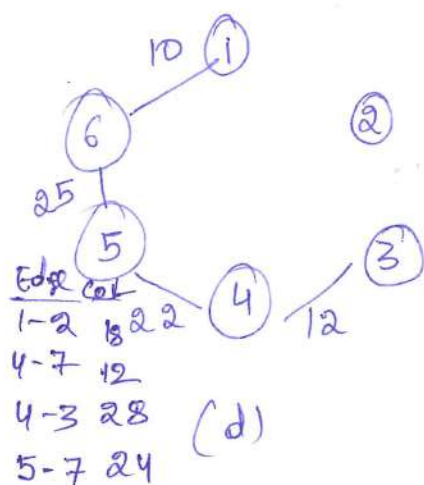
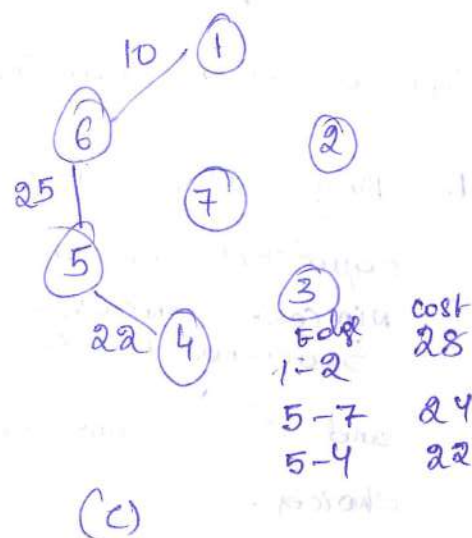
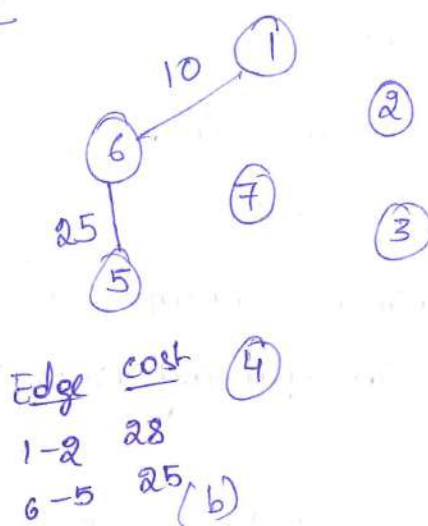
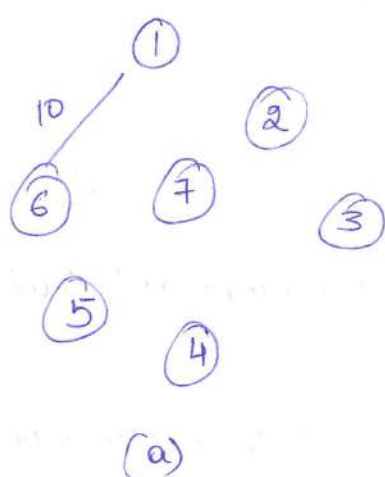
-the sum of the costs of the edges so far included.

2 possible ways to interpret the criterion.

1. using Prim's algo
2. " Kruskal's algo.

Prim's :- If A is the set of edges selected so far, then A forms a tree. The next edge (u,v) to be included in A is a min-cost edge not in A with the property that $A \cup \{(u,v)\}$ is also a tree.

Stages in Prim's algorithm:-



Min cost = 99

-Add the edges to the tree one by one.

The next edge (i,j) to be added is such that 'i' is a vertex already included in the tree, j is a vertex not yet included, and $\text{cost}[i,j]$ is the cost (i,j) is min among all edges (k,l) such that vertex 'k' is in the tree and vertex 'l' is not in the tree.

To determine this (i, j) efficiently, associate with each vertex (7)
 j not yet included in the tree a value $\text{near}[j]$.
 $\text{near}[j]$ is a vertex in the tree such that $\text{cost}[j, \text{near}[j]]$ is
 minimum among all choices for $\text{near}[j]$.

$\text{near}[j] = 0$ ~~value~~ for all vertices j that are already in the tree.

Algorithm Prim(E, cost, n, t) $\rightarrow O(n^2)$

E is the set of edges in G , $\text{cost}[1:n, 1:n]$ is the cost of
 adjacency matrix of an n vertex graph such that each entry
 is either ∞ or ∞ .

A min spanning tree is computed and stored as a set of
 edges in the array $t[1:n-1, 1:2]$.

$t[i, 1], t[i, 2]$ is an edge in the min-cost spanning tree.
 final cost is returned ∞

$\{$ Let k, l be an edge of min cost in E ; $\rightarrow O(|E|)$

$\text{mincost} := \text{cost}[k, l]; \rightarrow O(1)$

$t[k, 1] := k; t[l, 2] := l;$

for $i := 1$ to n do // Initialize near $\rightarrow O(n)$

if $(\text{cost}[i, l] < \text{cost}[i, k])$ then $\text{near}[i] := l;$

else $\text{near}[i] := k;$

$\text{near}[k] := \text{near}[l] := 0;$

for $i := 2$ to $n-1$ do

$\rightarrow O(n)$ for each iteration

$\{$ // Find $n-2$ additional edges for t . ~~repeated~~

Let j be an index such that $\text{near}[j] \neq 0$ and $\} \rightarrow O(n)$

$\text{cost}[j, \text{near}[j]]$ is min;

$t[i, 1] := j; t[i, 2] := \text{near}[j];$

$\text{mincost} := \text{mincost} + \text{cost}[j, \text{near}[j]];$

$\text{near}[j] := 0;$

for $k := 1$ to n do // update near[] $\rightarrow O(n)$

if $((\text{near}[k] \neq 0) \text{ and } (\text{cost}[k, \text{near}[k]] > \text{cost}[k, j]))$

then $\text{near}[k] := j;$

}

return mincost;

}

$O(n^2)$ but if we use red-black tree it reduces

Prims algo's time complexity is $O(n^2)$ $O((n + |E|) \log n)$

→ Given cost adjacency Matrix

	1	2	3	4	5	6	7
1	0	28	∞	∞	∞	10	∞
2	28	0	16	∞	∞	∞	14
3	∞	16	0	12	∞	∞	∞
4	∞	∞	12	0	22	∞	18
5	∞	∞	∞	22	0	25	24
6	10	∞	∞	∞	25	0	∞
7	∞	14	∞	18	24	∞	0

$k=1, l=6$
mincost = 10

t_1	1	6
2	5	6
3	4	5
4	3	4
5	2	3
6	7	2
7		

Consider $\infty = 1,00,000$
 $\Rightarrow 28 > \infty$ is True $\& \infty < \infty$ is false
 ~~$\infty \neq \infty$~~

(70)

Select j such that
 $near[j] \neq 0$ and
 $cost(i, near[j])$ is min

min cost

Update near
 $cost(i, near[k]) > cost(i, j)$
then $near[k] := j$

$k=1$ $l=6$

near
[2] [3] [4] [5] [6] [7]

$1,6 = 10 \neq 1,1 = 0$
 $2,6 = \infty \neq 2,1 = 28$
 $3,6 = \infty \neq 3,1 = \infty$
 $4,6 = \infty \neq 4,1 = \infty$
 $5,6 = 25 < 5,1 = \infty$
 $6,6 = 0 < 6,1 = 10$
 $7,6 = \infty < 7,1 = \infty$

10

$2,1 = 28 \neq 2,5 = \infty$
 $3,1 = \infty \neq 3,5 = \infty$
 $4,1 = \infty > 4,5 = 22$
 $7,1 = \infty > 7,5 = 24$

10+25

$j=5$
 $5,6 = 25$

$2,1 = 28 \neq 2,4 = \infty$
 $3,1 = \infty > 3,4 = 2$
 $7,5 = 24 > 7,4 = 18$

10+25+22

$j=4$
 $4,5 = 22$

10+25+22+12

$j=3$
 $3,4 = 12$

10+25+22+12+16

$j=2$
 $2,3 = 16$

10+25+22+12+16+14

$j=7$
 $7,2 = 14$

return mincost = 99

Kruskal's algorithm:

Every spanning tree of n vertices will have $n-1$ edges. The problem here is to select $n-1$ edges such that the cost of spanning tree is min.

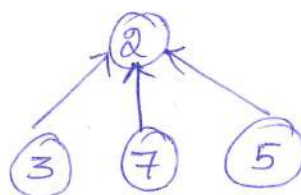
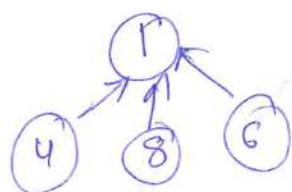
Let $G = (V, E)$ be a undirected graph. Algo selects $n-1$ edges one at a time. Algo selects least cost edge (u, v) from E and delete the edge (u, v) from E . check whether selected edge results in a cycle when added to already selected edges to form min spanning tree (MST). If no cycle is formed add the edge to tree. Otherwise neglect the edge.

Repeat the same proc till all $n-1$ edges are selected

Two functions used by Kruskal's algo are

1. $\text{find}(u)$: If u is a node in a tree, this returns root of the corresponding tree.

Ex: consider $\{1, 4, 8, 6\} \subseteq \{2, 3, 7, 5\}$



$$\text{find}(4) = \text{find}(8) = \text{find}(6) = 1$$

$$\text{find}(3) = \text{find}(7) = \text{find}(5) = 2$$

2. $\text{Union}(i, j)$: Let G_1 and G_2 be two trees. If i is a vertex in the tree G_1 , and j is a vertex in tree G_2 , this func merges G_1 & G_2 into a single tree.

$$\text{Ex: } S_1 = \{1\} \quad S_2 = \{2\} \quad S_3 = \{3\} \quad S_4 = \{4\} \quad S_5 = \{5\} \quad S_6 = \{6\} \\ S_7 = \{7\} \quad , \quad S_8 = \{8\}$$

Represent sets using array

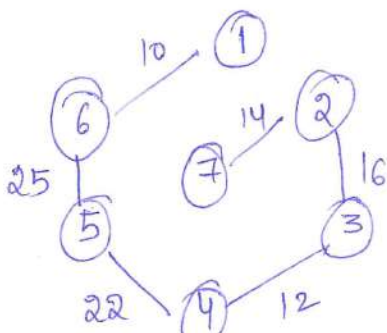
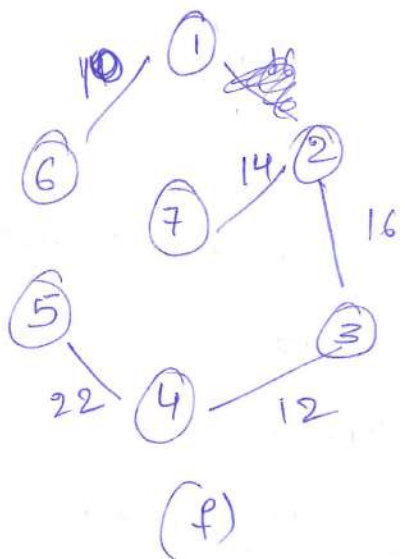
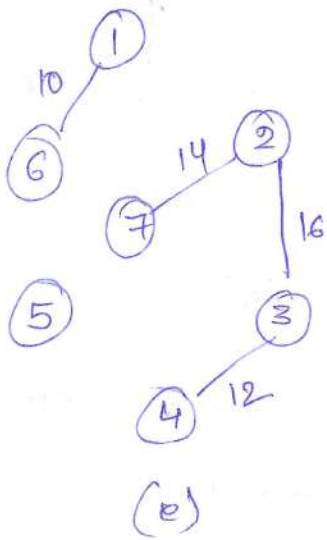
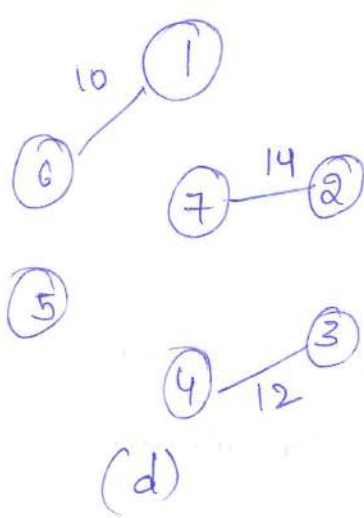
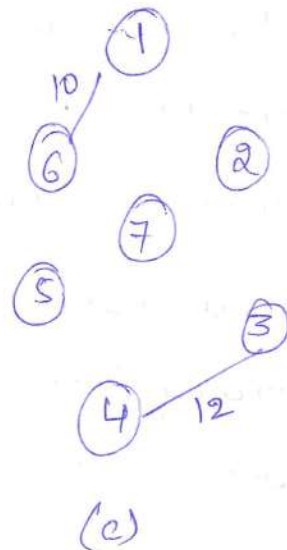
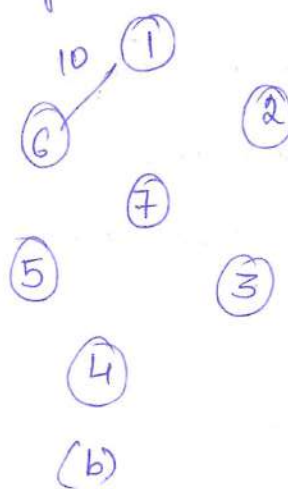
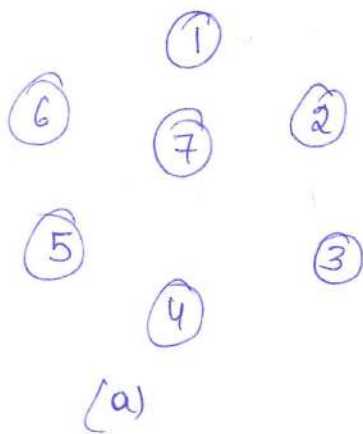
$s[1]$	$s[2]$	$s[3]$	$s[4]$	$s[5]$	$s[6]$	$s[7]$	$s[8]$
1	2	3	4	5	6	7	8

But $S_2 \cup S_5$

then $s[1]$	$s[2]$	$s[3]$	$s[4]$	$s[5]$	$s[6]$	$s[7]$	$s[8]$
1	2	3	4	2	6	7	8

— * — * —

Stages in Kruskal's algorithm



Early form of Kruskal's algo:-

(9)

$t = \emptyset$

while (t has less than $n-1$ edges) and ($E \neq \emptyset$) do

Σ choose an edge (v, w) from E of lowest cost;

 Delete (v, w) from E ;

 if (v, w) edges does not create a cycle in t then

 add (v, w) to t ;

 else discard (v, w) ;

 }

Kruskal's algo:-

Algorithm Kruskal (E, cost, n, t)

/* E is the set of edges in G . G has n vertices, $\text{cost}[u, v]$ is the cost of edge (u, v) . t is the set of edges in the min cost spanning tree. The final cost is returned.

Σ construct a heap out of the edge costs using Heapify;

 for $i := 1$ to n do $\text{parent}[i] := -1$;

 // Each vertex is in a different set.

$i := 0$; $\text{mincost} := 0.0$;

 while ($i < n-1$) and (heap not empty) do

Σ Delete a min cost edge (u, v) from the heap and

 reheapify using Adjust

$j := \text{Find}(u)$; $k := \text{Find}(v)$;

 if ($j \neq k$) then

$\Sigma i := i + 1$;

$t[i, 1] := u$;

6

$t[i, 2] := v;$

$\text{mincost} := \text{mincost} + \text{cost}[u, v];$

$\text{Union}(j, k);$

}

}

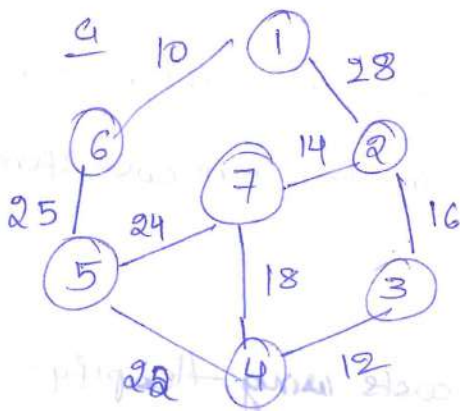
if $(i \neq n-1)$ then write ("No spanning tree");

else return mincost;

}

Computing time of Kruskal's algo is $O(|E| \log |E|)$

→ Find the min cost spanning tree for G



Arrange the edges in ~~non~~ increasing order of weights

9.	1-2	28
8	5-6	25
7	5-7	24
6	4-5	22
5	4-7	18
4	2-3	16
3	2-7	14
2	3-4	12
1.	1-6	10

order



Single-Source Shortest-Paths

Graphs represent highway structure of a state or country

vertices " cities

edges " sections of highway.

weights " distance b/w two cities connected by the edge
or
avg time to drive along that section of highway.

Is there a path from A to B?

If there is more than one path from A to B, which is the

shortest path?

length of a path is sum of the weights of the edges on that path.

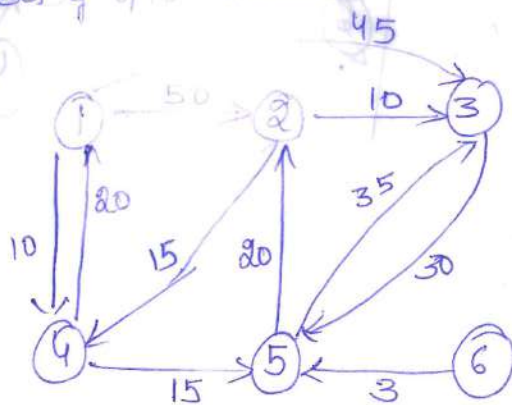
The starting vertex of the path is referred to as the source

destination

The graphs are digraphs to allow for one-way streets.

Given a directed graph $G = (V, E)$, a weighting function 'cost' for the edges $c: E \rightarrow \mathbb{R}$ and a source vertex v_0 . The problem is to determine the shortest paths from v_0 to all the remaining

vertices of G . Assume all weights are +ve.



Shortest path from vertex ①

Path	length
1) 1, 4	10
2) 1, 4, 5	25
3) 1, 4, 5, 2	45
4) 1, 3	45

Greedy-based algo: -

(11)

needs a multistage soln to the problem and also an optimization measure. Build shortest path one by one.

" " is sum of the lengths of all paths so far generated.

To minimize sum of lengths, each individual path must be of min length.

→ Shortest path to the nearest vertex is generated. Then a shortest path to the second nearest vertex is generated & soon.

→ Let S denote the set of vertices (including v_0) to which the shortest paths have already been generated.

For w not in S , let $\text{dist}[w]$ be the length of the shortest path starting from v_0 , going through only those vertices that are in S , and ending at w .

Algorithm Shortest paths ($v, \text{cost}, \text{dist}, n$)

* $\text{dist}[j]$, $1 \leq j \leq n$, is set to length of the shortest path from vertex v to vertex j in a digraph G with n vertices.

$\text{dist}[v]$ is set to zero.

G is represented by its cost adjacency matrix $\text{cost}[1:n, 1:n]$

for $i := 1$ to n do

 // Initialize S .

$S[i] := \text{false}$; $\text{dist}[i] := \text{cost}[v, i]$;

$S[v] := \text{true}$; $\text{dist}[v] := 0.0$; // put v in S .

for $\text{num} := 2$ to n do

 // Determine $n-1$ paths from v .

 choose u among those vertices not in S such that $\text{dist}[u]$ is min.

$s[u] := \text{true};$ // Put u in S .

for (each w adjacent to u with $s[w] = \text{false}$) do

// Update distances.

if ($\text{dist}[w] > (\text{dist}[u] + \text{cost}[u, w])$) then

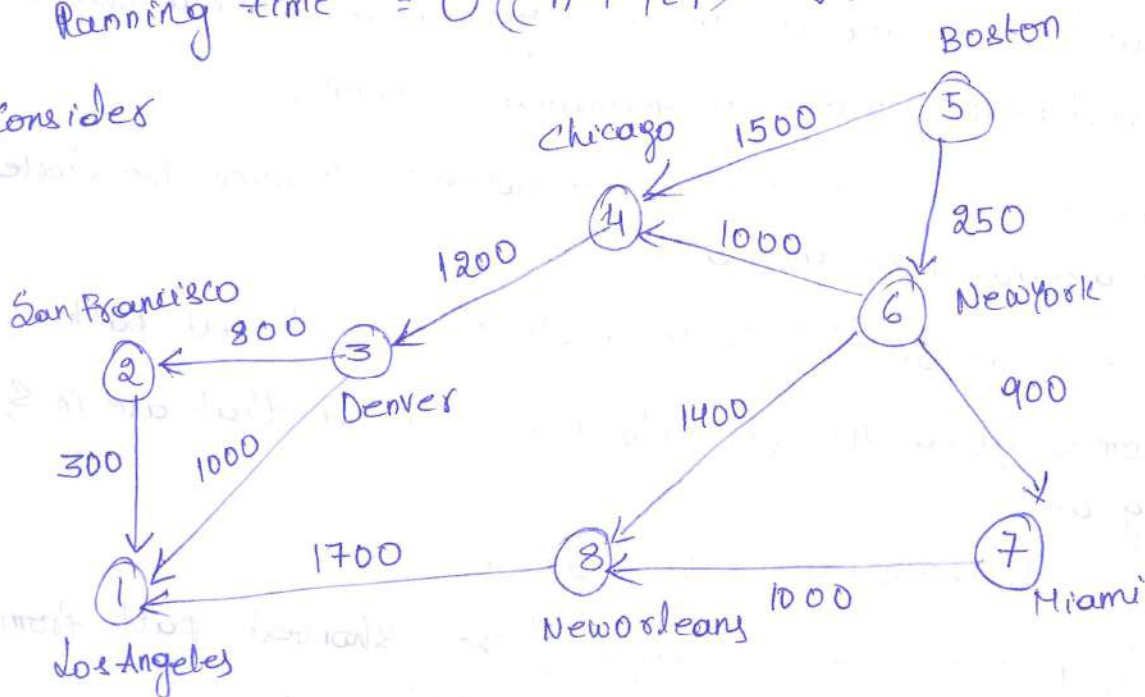
$\text{dist}[w] := \text{dist}[u] + \text{cost}[u, w];$

}

}

Running time = $O((n + |E|) \log n)$

→ Consider



	1	2	3	4	5	6	7	8
1	0							
2	300	0						
3	100	800	0					
4			1200	0				
5				1500	0	250		
6				1000		0	900	1400
7							0	1000
8	1700							0

Source :- BOSTON

Distance

(12)

Iteration	S	Vertex selected 'u'	LA [1]	SF [2]	DEN [3]	CHI [4]	BOST [5]	NY [6]	MIA [7]	NO [8]
Initial	-	-	$+\infty$	$+\infty$	$+\infty$	1500	0	250	$+\infty$	$+\infty$
1	{5}	6	$+\infty$	$+\infty$	$+\infty$	1250	0	250	1150	1650
2	{5,6}	7	$+\infty$	$+\infty$	$+\infty$	1250	0	250	1150	1650
3	{5,6,7}	4	$+\infty$	$+\infty$	2450	1250	0	250	1150	1650
4	{5,6,7,4}	8	3350	$+\infty$	2450	1250	0	250	1150	1650
5	{5,6,7,4,8}	3	3350	3250	2450	1250	0	250	1150	1650
6	{5,6,7,4,8,3}	2	3350	3250	2450	1250	0	250	1150	1650

{5,6,7,4,8,3,2}

if ($\text{dist}[w] > (\text{dist}[u] + \text{cost}[u,w])$) then
 $\text{dist}[w] := \text{dist}[u] + \text{cost}[u,w];$

Iteration 1

$u=6$

$w=4$

$\text{dist}[u] > (\text{dist}[6] + \text{cost}[6,4])$

$1500 > 250 + 100$

$\text{dist}[4] = 1250$

$w=7$

$w=8$

Iteration 2

$u=7$

$w=8$

$\text{dist}[7] > (\text{dist}[8] + \text{cost}[7,8])$

$1150 > (1650 + 100)$

False

