DATE:
PAGE:

Loops :

1. Entry controlled loops : In entry controlled loops the test condition is checked before entering the main body of the loop
Ex For & while loop
2. Exit Controlled loops :- In exit controlled loops the test condition is evaluated at the end of loop body. The loop body will execute at least once, irrespective of whether the condition is true or false. do-while loop is example

For loop :- For loops in C prog. is a repetition control structure that allows programmers to write a loop that will be executed a specific num. of times

Syntax

```
for (initialize expression; test exp; update exp)
{
    body of for loop
}
```

While loop :- While loop does not depend upon the num of iteration. In for loop the num of iteration was previously known to us but in the while loop, the execution is terminated on the basic of the test condition If the test condition will become false then

it will break from the while loop else body will be executed.

Syntax :-

```
initialization _ expression;
while (test _ expression)
{
    // body of the while loop
    update _ expression;
}
```

Do - while loop :- The do - while loop is similar to a while loop but the only diff lies in the do - while loop test condition which is tested at the end of the body. In the do - while loop, the loop body will execute at least once irrespective of the test condition.

Syntax :-

```
initialization _ expression;
do
{
    // body of do - while loop
    update expression;
} while (test _ expression);
```

Break :- is used to terminate the switch and loop statement. It transfers the execution to the statement immediately following the loop or switch.

DATE
PAGE

Continue :- Continue statement skips the remainder body & immediately reevaluates its condition before reiterating it.

Goto :- Goto statements transfers the control to the labeled statement

If-else statement :- The if-else statement in C is a flow control statement used for decision-making in program. It is one of the core concepts of C prog. It is an extension of the if in C that includes an else block along with the already existing if block

C if statement :- is used to execute a block of code based on a specified condition.

Syntax

```
if (condition) {
    // Code to be executed if the condition is true
}
```

if-else statement :- That if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the specified condition if the given condition is true, then the code inside the if block is executed, otherwise the code inside the else block is executed.

DATE.
PAGE.

Syntax:

```
if (condition) {
// code executed when the condi is true
}
else {
// code executed when the condi is false
}
```

Design and Analysis of Algorithm

UNIT - 1

Graph, Set, Union
Searching → Linear
→ Binary
Sorting → Merge
→ Quick
→ Selection
Straser Matrix multiplication

Unit - 2

Greedy Method

- Knapsack problem (fractional)
- Job sequencing with deadline
- Minimum spanning tree
- Single Source shortest path

Dynamic programming

- Tree, OTT Knapsack
- Travelling Sales person

UNIT - 3

Back Tracking

- 8 Queen's problem
- Graph colouring
- Hamilton Cycle

Module-4

Branch & Bound

- OTT Knapsack
- Travelling salesperson

Module-5

- NP Hard
- NP complete problem
- Cook's theorem
- NP hard graph
- NP scheduling

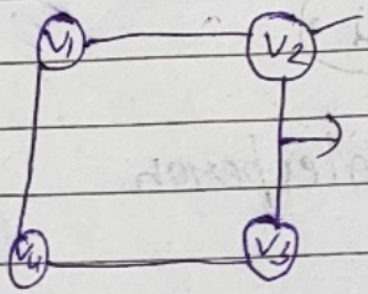
DATA STRUCTURE ⇒ A data structure is used to organize data in computer effictively
- Data structure can be linear or non linear.

TYPES

linear data structure	non linear data structure
• Array	• Tree
• list	• Graph
• Stack	• Heap
• Queue	

- Linear data str organized data in sequential order
- Non linear data structure organize data in random order

Graph ⇒ Greaphs is non L.D.S made up of vertices & edges. The edges connected any two node in the graph. That is made up of nodes and vertices.



Imp

Order of graph :- The order of graph is total no. of vertices in graph.

Size :- The size of the graph is the total no. of edges in the graph

Directed graph
Undirected graph

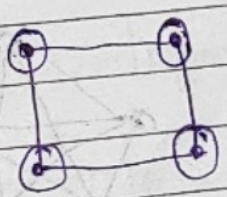
Adjacent Vertices :- Two vertices having common edge.

Adjacent edges :- Are the edges that share common vertices

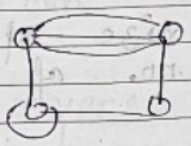
Self loop :- An edge of a graph which starts & end at the same vertex. Some types of graph allows self loops & some do not

Multi edge system :-

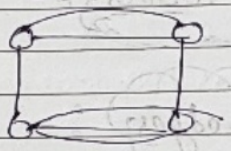
Simple Graph :- A simple graph is no self loop & no multiple edge



Seudo graph :- Seudo graph is a combination of multiple edges & self loop

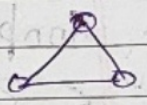


Multi graph :- A graph which has multiple edges but no self loop

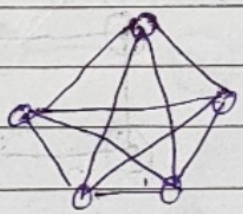


Regular graph :- A graph in which every node has same degree (no. of edges coming towards the vertices)

No. of edges = $\frac{\text{no of node} \times \text{degree}}{2}$



Complete graph :- A graph in which every pair of vertices is adjacent to each other



A set is

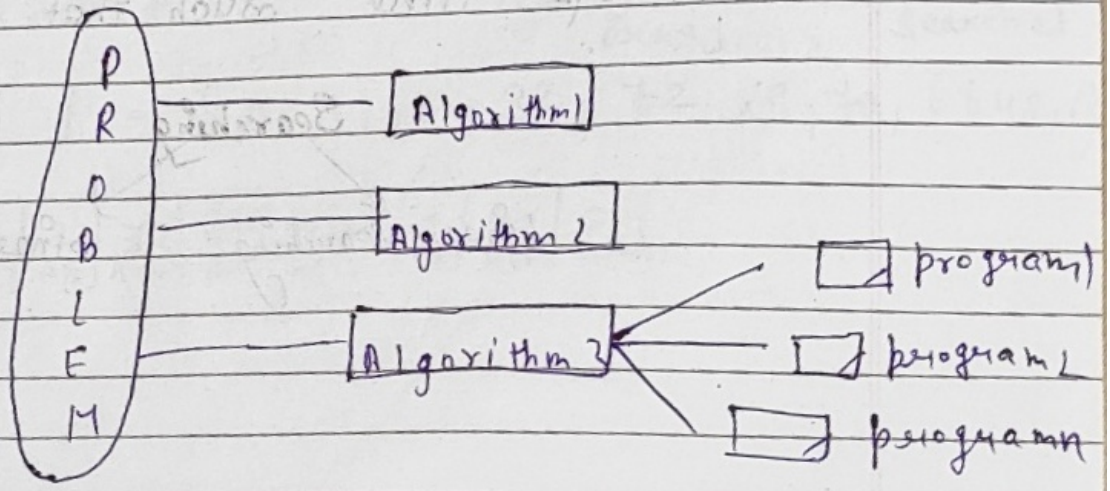
Set :- A set is a data str. then can store any num. of unique value. They only allow non repeated unique values within them

Disjoint :- A pair of set which does not have any common element are called disjoint set.

A = {1, 7, 8, 9}
B = {3, 6}

What is algorithm?

- An algorithm is a sequence of computational steps that transform the input into the output.
- An algorithm is a step by step procedure to solve problems
- An algorithm defines as a sequence of statement which are used to perform a task.



DATE:
PAGE:

An algorithm is a sequence of unambigous inst^n used for solving a problem which can be implemented on a computer.

ALGORITHM CHARACTERSTICS :-

1. INPUT :- Every algorithm must take 0 or more input value from external
2. OUTPUT :- Every algorith must produce 1 or more output as result.
3. Definitness :- Every inst^n in an algorithm must be clear & unambigous which mean their should be only 1 meaning of inst^n
4. Finitness :- The algorithm must produce result within a finite num of steps
5. EFFICTIVENESS :- Every inst^n must be basic enough to be carried out & it also must be feasible such that it can be turn by a person using a pen & paper in a finite amount of time such that it can.

Searching

linear searching

binary searching

Linear Searching :- Linear search is a very simple search algorithm. In this type of search a sequentially search is made over all items one by one. Every item is checked and if a match is found then the particular item is returned otherwise the search continues till the end of data collection.

Algorithm :-

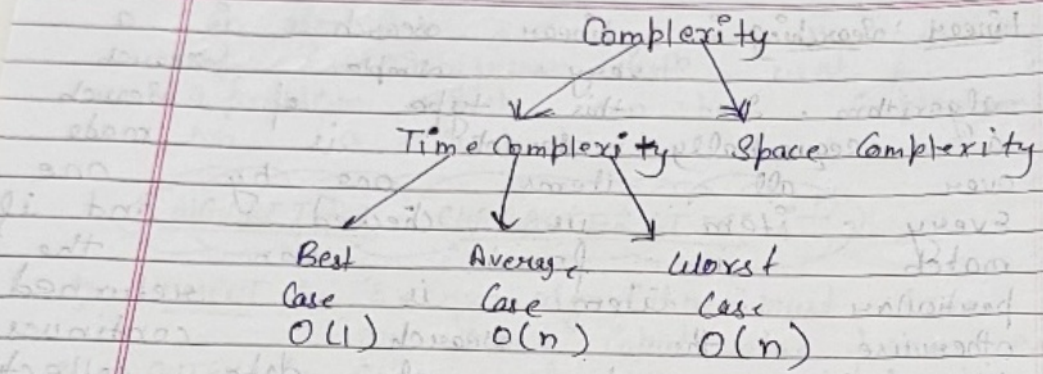
linear search (Array A, Value = x)

Step 1 : Set i to 0
Step 2 : if i > n then go to step 7
Step 3 : if A[i] = x then go to step 6
Step 4 : Set i to i+1
Step 5 : Go to step 2
Step 6 : Print Element x found at index i and
Step 7 : Print element not found
Step 8 : Exit

n = size of array x = item to be searched

int A[] = {50, 1, 5, 32, 72, 89, 70, 69, 42, 61};

50	1	5	32	72	89	70	69	42	61
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]



(1) It is used for unsorted list of element
(2) Use linear search when the num. of elements in the list are small
(3)

Binary search :- Binary search works on shorted array by repeatdly dividing the search interval of half. We begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of array, narrow the interval to lower half. Otherwise narrow it to upper half. Repeatdly check until the value is found or the interval is empty.

Algorithm

```
Binary Search (int low, int high, key)
{
    while (low ≤ high)
    {
    int mid = (low + high)/2
        if (a[mid] < key)
        {
            low = mid + 1;
        }
        else if (a[mid] > key)
        {
            high = mid - 1;
        }
        else
        {
            return mid
        }
        return -1;
```

Complexity :-

Best case O(1)
worst Case :- (log n)
Average Case : (log n)

Divide & Conquer technique based

Sorting :-

Sorting is nothing but arranging a data in ascending or decending order. their are so many thing in our daily life that we need to search for like a particular record in database, Roll no in merit list, a particular telephone no in telephone directory or a particular page in book. All of these would have been a mess if the data was kept unorder & unsorted. But fortunately the concept of sorting came into existence making it easier for everyone to arrange data in an order, hence making it easier to search

Their are many diff tech available for sorting differentiated by their efficiency & their space requirement which are given below:

1) Bubble sort 2) Selection sort 3) Merge 4) Quick sort 5) heap sort 6) Insertion sort 7) Bucket sort

Selection Sort :-

Selection Sort Algorithm :-

Algorithm

Step1 : Begin
Step2 : input a[5]
Step3 : Set i ← 0
Step4 : Repeat step 5 to 10 while (i < a)
Step5 : Set m ← a[i] , j ← i+1
Step6 : Repeat step 7 to 8 while (j < 5)
Step7 : if m > a[j] then
set m ← a[j]
set loc ← j
Step8 : j ← j+1
Step9 : if a[loc] < a[i] then
set temp ← a[loc]
a[loc] ← a[i]
a[i] ← temp
Step10 : Set i ← i+1
Step11 : print a[5]

Binary search algorithm :-

Iterative approach :-

```
binarySearch (arr, size)
loop until first is not equal to end
  midIndex = (beg + (first + end)) 2
    if (item == arr[midIndex]
      return midIndex
    else if (item > arr[midIndex])
      first = mid index + 1
```

else
end = mid index − 1

Recursive approach

1) binarySearch (arr, item, first, end)
2) if beg <= end
3) mid index = (first + end) / 2
4) if item == arr[midIndex]
5) return binarySearch (arr, item, midIndex + 1, end)
6) else
7) return binarySearch (arr, item, beg first, midindex-1)

return -1

Merge Sort

It is the sorting technique that follows the divide and conquer approach.
The merge sort - repeatdly divide the array in two half until we reach a stage where we try to perform merge sort on a sub array of size 1.

Merge Sort (A[Array], p, r), last element
if p ≥ r
return
q = (p+r) / 2

merge sort (A, P, r)
merge sort (A, q+1, r)
merge (A, P, q, r)

Algorithm :-

Merge Sort

MERGE_SORT (arr, first, last)

~~if beg <~~ If first < last
set mid = (beg + end)/2 [beg = first]
MERGE_SORT (arr, beg, mid)
MERGE_SORT (arr, mid+1, end)
MERGE (arr, beg, mid, end)

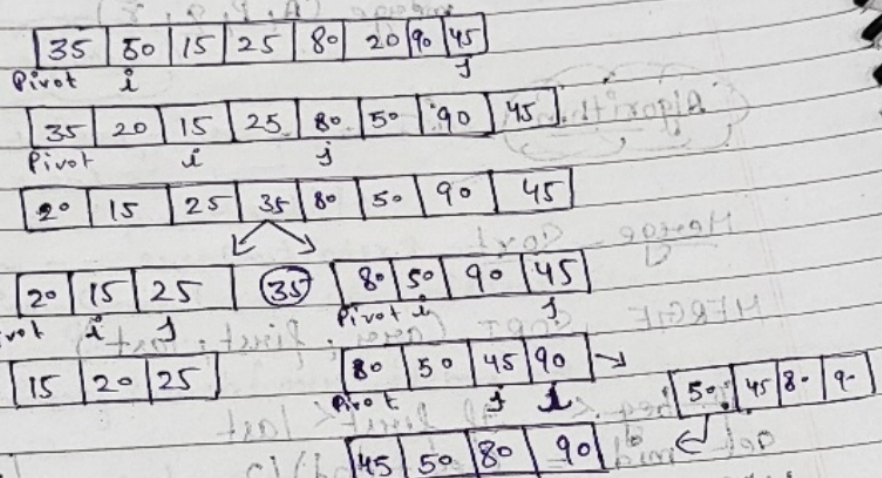
end of if

Quick Sort

Quick sort is a sorting algorithm based on the Divide & Conquer al that picks an element as a and partitions the given ar around the picked pivot by plac the pivot in its correct positio the sorted array

Quick Sort

i > Pivot
j < Pivot



Algorithm :-

```
QuickSort (array A, start, end)
{
    if (start < end)
    {
    P = partition (A, start, end)
    QuickSort (A, start, P-1)
    QuickSort (A, P+1, end)
    }
}
```

STRASSEN'S MATRIX MULTIPLICATION

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}_{2\times 2} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}_{2\times 2}$$

$C = A \times B$ ~~$O(n^3)$~~

Naive Method :-

Algorithm : Matrix multiplication (A, B, C)

```
for int i = 1 to p do
    for j = 1 or r do
        C[i, j] = 0
    for K = 1 to q do
        C[i, j] = A[i, K] X B[K, j]
```

2nd method

$$C = \begin{bmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} \times b_{12} + a_{12} \times b_{22} \\ a_{21} * b_{11} + a_{22} * b_{21} & a_{21} \times b_{12} + a_{22} * b_{22} \end{bmatrix}$$

$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$
$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$
$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$
$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$

Comp $= O(n^{2.81})$

Ex $Z = \begin{bmatrix} I & J \\ K & L \end{bmatrix} \quad X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$

$M_1 = (A+C) * (E+F)$
$M_2 = (B+D) * (G+H)$
$M_3 = (A-D) * (E+H)$
$M_4 = A * (F-H)$
$M_5 = (C+D) * E$
$M_6 = (A+B) * H$
$M_7 = D * (G-E)$

$I = M_2 + M_3 - M_6 - M_7$
$J = M_4 + M_1$
$K = M_5 + M_7$
$L = M_1 - M_3 - M_4 - M_5$

Ex $X = \begin{bmatrix} 5 & 4 \\ 2 & 1 \end{bmatrix}$ $Y = \begin{bmatrix} 4 & 6 \\ 5 & 7 \end{bmatrix}$ $Z = ?$

$M_1 = 70$
$M_2 = 60$
$M_3 = 44$
$M_4 = -5$
$M_5 = 12$
$M_6 = 63$
$M_7 = 1$

$I = 40$
$J = 58$
$K = 13$
$L = 19$

So $Z = \begin{bmatrix} 40 & 13 \\ 58 & 19 \end{bmatrix}$

Complexity $\Rightarrow O(\log^{n} 7)$.

Unit ⇒ 2

Greedy technique

A Greedy algorithm always makes the choice that looks best at that particular movement.

Properties of Greedy technique

Greedy technique works on following property:-

Greedy choice property :- Greedy choice property makes locally optimal choice in the hope that the choice tends to globally optimal soln.

Optimal Substructure :- Optimal substructure mean optimal soln contain optimal sub solution.

Knapsack problem :- In knapsack problem we want to pack n item in your luggage the i^{th} item is worth (v_i) dollar and weight (w_i) pound. We need to put these items in a knapsack of capacity (W) to get the maximum total value in the knapsack.

	0/1 Knapsack	Fractional Knapsack
1	The zero/one knapsack problem is solved used Dynamic programming approach.	The fractional knapsack problem is solved using Greedy approach
2	Do not guarantee optimal soln	Do give optimal soln
3	In 0/1 we are not break items.	In this we are break item
4	We either take whole item or do not take it.	We can take fraction of items.

Q There are 5 items with their respective weight and values.

$I = [I_1, I_2, I_3, I_4, I_5]$
$W = [5, 10, 20, 30, 40]$
$V = [30, 20, 100, 90, 60]$
The Capacity of knapsack is 60 Solve using fractional knapsack/Greedy tech.

Soln

Items	Weight	Value
I_1	5	30
I_2	10	20
I_3	20	100
I_4	30	90
I_5	40	160

2.

Step 1 :- Taking value per weight ratio.

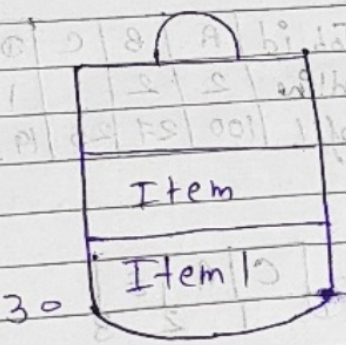
Items	weight	value	$P_i = v_i/w_i$
I_1	5	30	30/15 = 6
I_2	10	20	20/10 = 2
I_3	20	100	100/20 = 5
I_4	30	90	90/30 = 3
I_5	40	160	160/40 = 4

Step 2 :- Arrange value of P_i in decreasing order

Items	weight	value	P_i
I_1	5	30	6
I_2	20	100	5
I_3	40	160	4
I_4	30	90	3.
I_5	10	20	2

(or (1 for (2)

Step 3 :- Now fill the items in bag



If we take something in fraction then formula

$$\frac{v_i}{w_i} * \text{weight left in knapsack}$$

DATE: 5
PAGE:

Q2

Job Sequencing with Deadline

Job id	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	100	19	27	25	15

Given an array of jobs where every job has a deadline & associated profit if the job is finished before the deadline. It is also given that every job takes single unit of time. So the minimum possible deadline for any job is 1. So how to maximise total profit if one & only one job can be scheduled at a time. This problem is solved during Greedy approach

Sol^{n}) Sort all job in order of profit

Job id	A	B	C	D	E
deadline	2	2	1	1	3
Profit	100	27	25	19	15

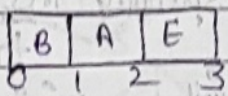
C	A	E
0	1 2	3

100 + 27 + 15 = 142

Job id	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	100	27	19	25	15

Sort in ↓ order

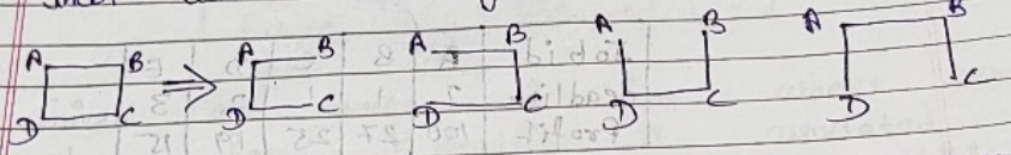
Job id	A	B	C	D	E
deadline	2	1	1	2	3
Profit	100	27	25	19	15



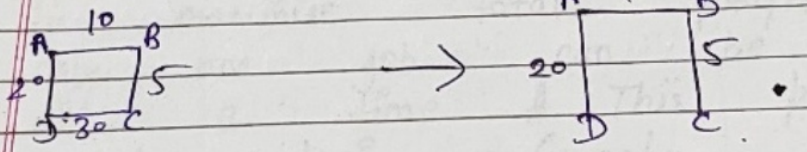
100 + 27 + 15
= 142

Minimum Spanning tree

Spanning tree :- S.T is a connected graph using all vertices in which their is no cycle.



Minimum Spanning tree :- In this we want to find a subset of edges with minimum total weight that connects all vertices into a tree



• Ans = 35

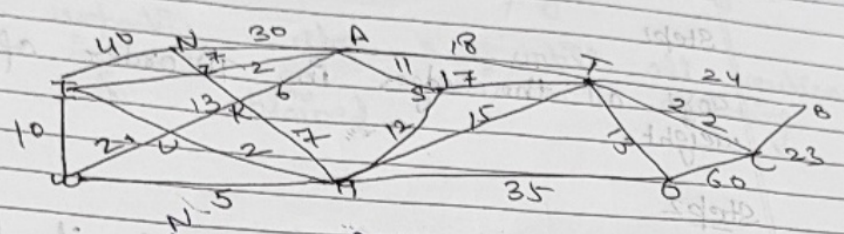
A graph may have more than one minimum spanning tree

Minimum S.T

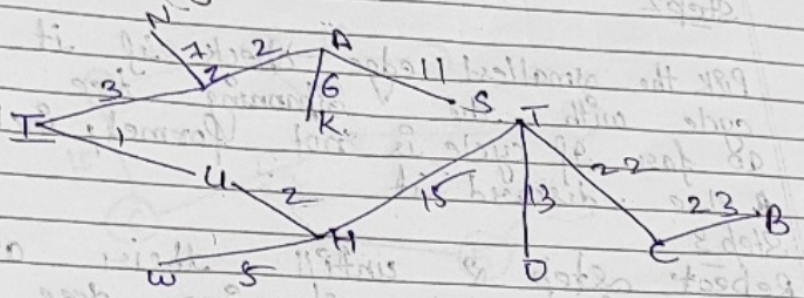
Kruskal MST Prim's MST

Kruskal Approach

Q



Ans



Edges	weight
I, U	1
U, H	2
Z, A	2
I, Z	3
W, H	5
K, A	6
N, Z	7
✗ K, H	7
✗ I, W	10
A, S	11
✗ S, H	12
T, O	13
✗ U, K	13
H, T	15

Edges	weight
✗ S, T	17
✗ A, T	18
✗ W, U	20
T, C	22
C, B	23
✗ T, B	24
✗ Z, K	27
✗ N, A	30
✗ H, O	35
✗ I, N	40
✗ O, C	60

Kruskal Algorithm :-

Step1

Sort all the edges in ↑ order of their weight.

Step2

Pick the smallest edge check if it form a cycle with the spanning tree formed so far. If cycle is not formed, include the edge & else discard it

Step 3

Repeat step 2 until there are (V-1) edges in the spanning tree.

Prim's

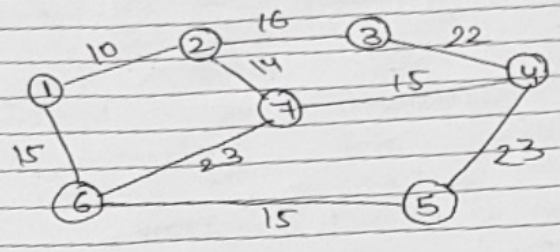
Prim's algorithm

1 Randomly choose any vertex. We usually select and start with a vertex that connects to the edge having least weight.

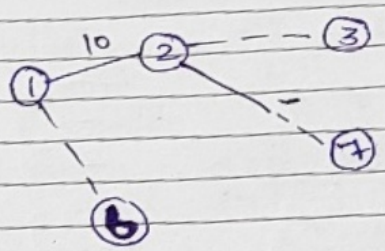
2 Find all the edges that connects the tree to new vertices, then find the least weight edge and include among those edge & included in the existing tree.

If including that edge creates a cycle then reject that edge

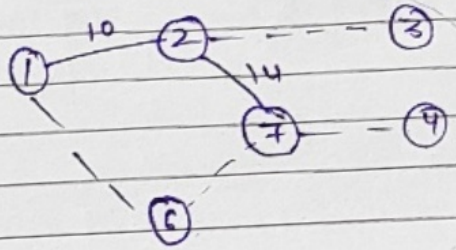
Keep repeating step 2 untill all vertices are included and minimum spanning tree is obtained.



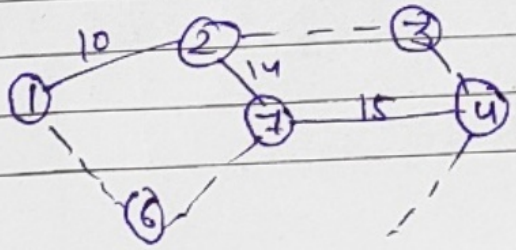
Step1



Step2



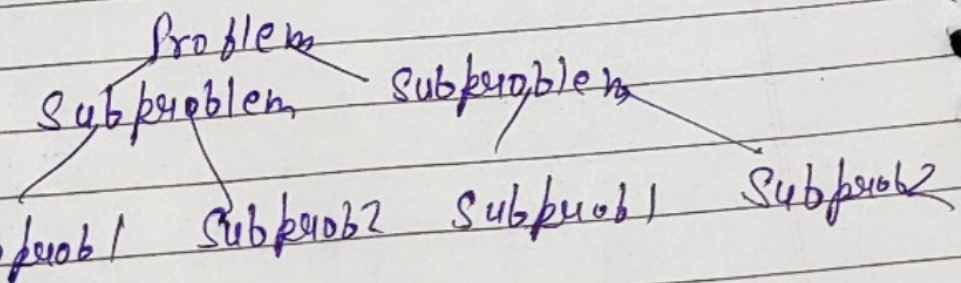
Step3



Knapsack Problem

- Fractional Knapsack (Greedy Approach)
- 0/1 Knapsack (Dynamic programming)

Dynamic Programming :- Dynamic programming is an algorithmic paradigm that solve a given complex problem by breaking it into sub problem and stores the result of sub problems to avoid computing the same result again. D.P are used where we have problems which can be divided into similar sub problems so that their results can be reused. D.P approach is used for optimisation. Before solving the in hand difficult sub problem, the D.P approach examines the result a previously solved sub problems



eg ⇒ Fibonacci Series

F_n = 0, 1, 1, 2, 3, 5

int a[10]

* 0/1 Knapsack :- We want to pack n items in your luggage. The ith item is worth v_i dollar and weight w_i pound we need to put these items in a knapsack of capacity W to get the maximum total weight in the knapsack. 0/1 knapsack problem can be solved using dynamic programming approach.
* We are not allowed to break items
* We can either take all items, or don't take it.
* Each item is taken or not taken.
* Zero represent : not taken and 1 represent taken.

Item	weight	Value
1	1	1
2	3	4
3	4	5
4	5	7

Capacity ⇒ 7

Weight of Knapsack (Capacity)

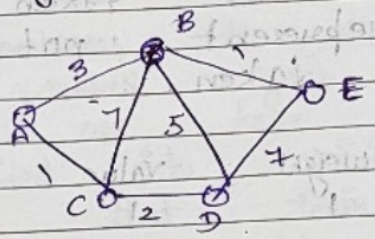
Items	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0	1	1	4	5	5	5	5
3	0	1	1	4	5	6	6	9
4	0	1	1	4	5	7	8	9

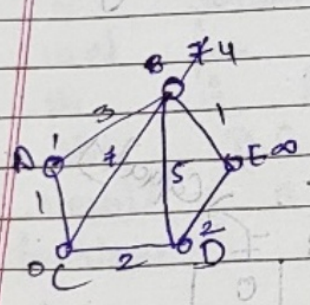
int a[10][10]

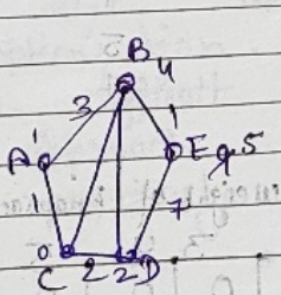
a[i][j]

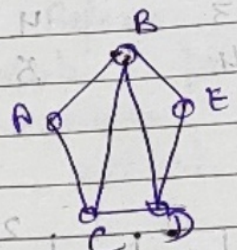
Single source shortest path problem (Dijkstra Algorithm)

Dijkstra algo. is used to provide soln to the single source shortest path problem in graph theory. * It can be applied on both directed and undirected graph all edges of graph must have non negative weight graph must be connected remove all self loops and parallel edges.









Visited vertex	A	B	D	E
	①	7	2	∞
A		4	②	∞
D		④		7
B				5
E				

C → A
C → A → B
C → D
C → A | B | E

Q

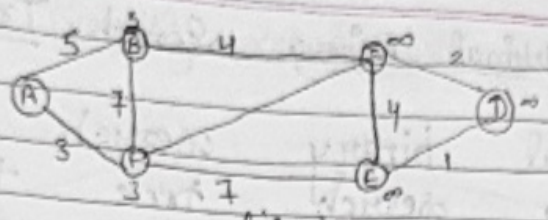


fig-1

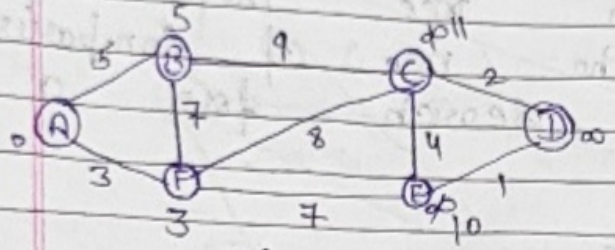


fig-2

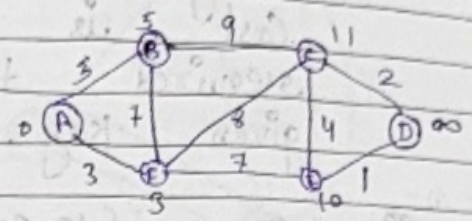
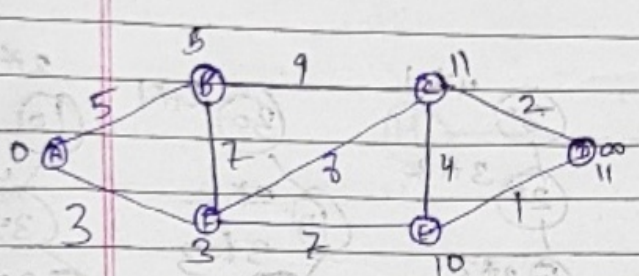


fig-3



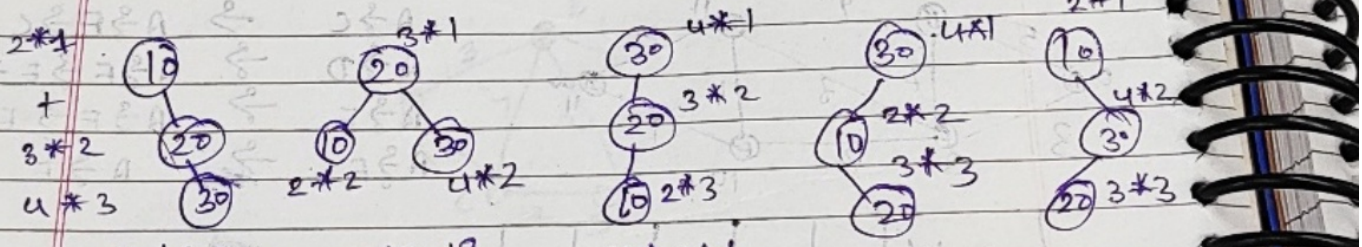
A→B → A→B
A→C → A→F→C
A→D → A→F→E→D
A→E → A→F→E
A→F → A→F

visted Vertex	B	C	D	E	F
F	5	∞	∞	∞	(3)
B	(5)	11	∞	∞	–
E	–	11	∞	(10)	–
D	–	11	(11)	–	–
C		11			

Optimal Binary Search Tree

* An optimal binary search tree is a binary search tree that minimize the expected search cost. In binary search tree the search cost is the no. of comparision required to search for a given key.

Key = {10, 20, 30}
freq = {2, 3, 4}



2+6+12 = 20

3+4+8 = 15

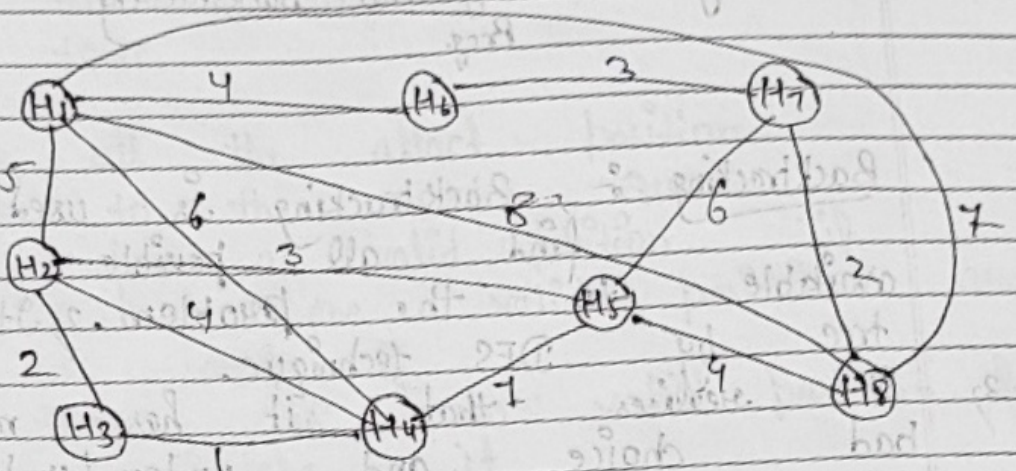
4+6+6 = 16

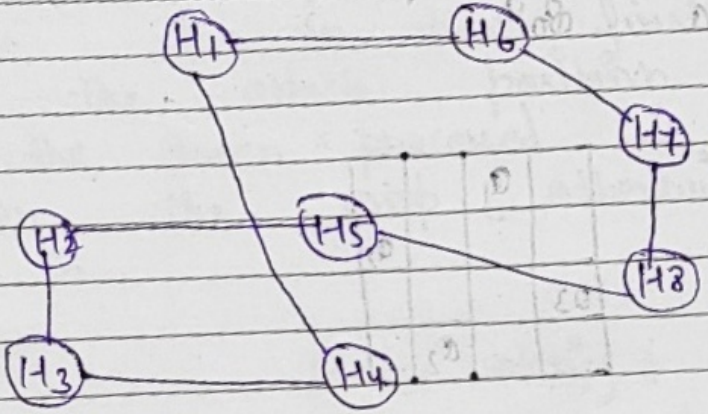
4+4+9 = 17

=2+8+9 = 19

DATE:
PAGE:

Travelling Salesperson Problem





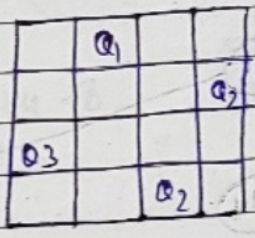
	H1	H2	H3	H4	H5	H6	H7	H8
H1	0	5	0	6	0	(4)	0	7
H2	5	6	(2)	4	3	0	0	0
H3	0	2	0	(1)	6	0	0	0
H4	(6)	4	1	0	7	0	0	0
H5	0	(3)	0	7	0	0	6	4
H6	4	0	0	0	0	0	(3)	0
H7	0	0	0	0	6	3	0	(2)
H8	7	0	0	0	(4)	0	2	0

Techniques → Greedy tech, Dynamic Prog., Backtracking, Branch & Bound

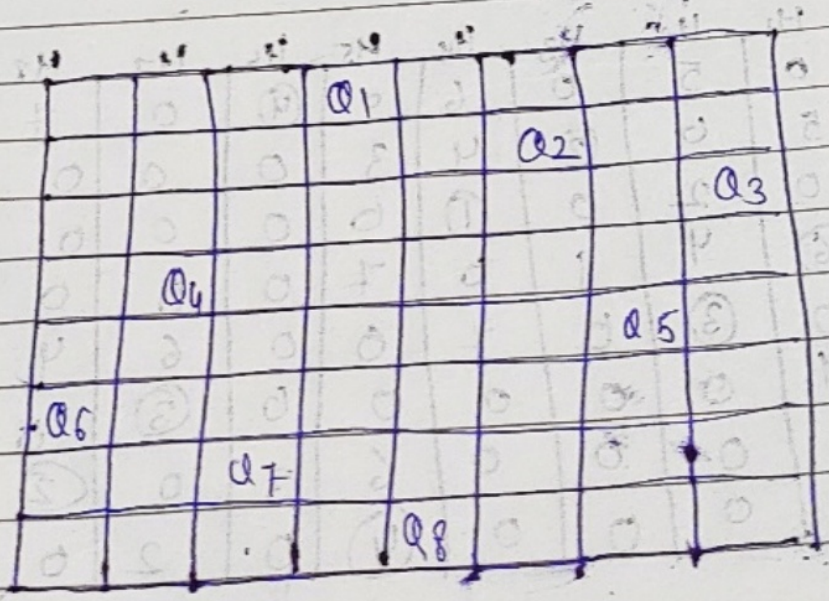
Backtracking :- Backtracking is used to find all possible solⁿ available to the problem. 2) It traverse tree by DFS technique.

3) If it realises that it has meet a bad choice and under / undo the last choice by backing up. It searches the state space tree untill it find a solⁿ

4) 4×4



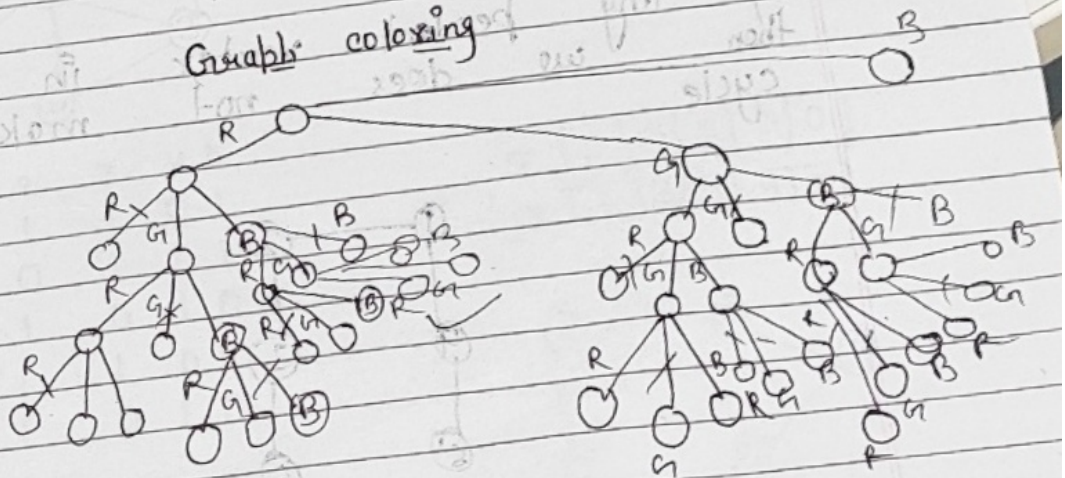
8×8



8 steps Algorithm :-

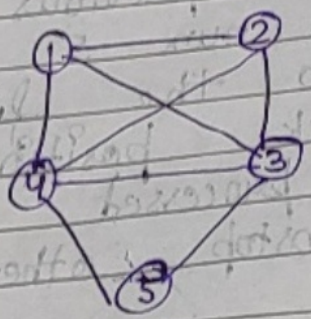
* Place the Queen in left upper corner of the table.
* Save all the attack position.
* Move to the next Queen
* Search for a valid position. if there is one go to step 9.
* if there is no valid position and if Queen Delete it.
* Move to the previous Queen.
* Go to step num 5.
* Place it to the first valid position
* Save the attack position.
* If the Queen processed is the last Queen the stop - otherwise go to step no. 3.

Graph coloring

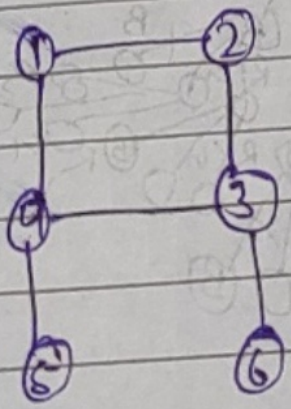


Hamiltonian Cycle

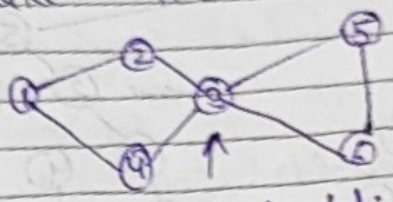
In an undirected graph a hamiltonian cycle is a graph cycle that visits each node exactly ones. A graph possing a hamiltonian cycle is said to be a hamiltonian graph. Given a graph G we have to find hamiltonian circuit using back tracking approach. We start a search from any arbitary vertex that vertex becomes the root of our implicit tree



If any pendent vertex in the graph then we does not make hamiltonian cycle



* If articulation point is present in graph we can't make H.C

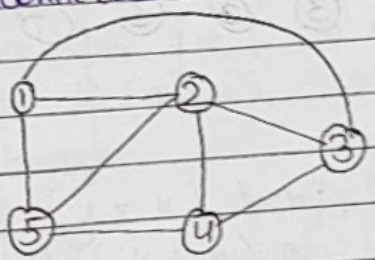


Articulation point

A graph G is called pendant Graph. If in a Graph, (G) a vertex V is called pendant vertex if and only if the degree of particular vertex is one

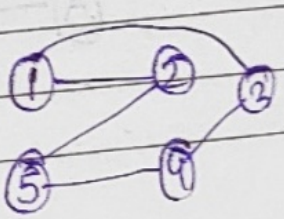
An Articulation point is defined as a vertex which when remove along with associated edges makes the graph disconnected.

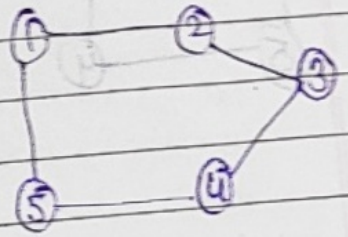
Q

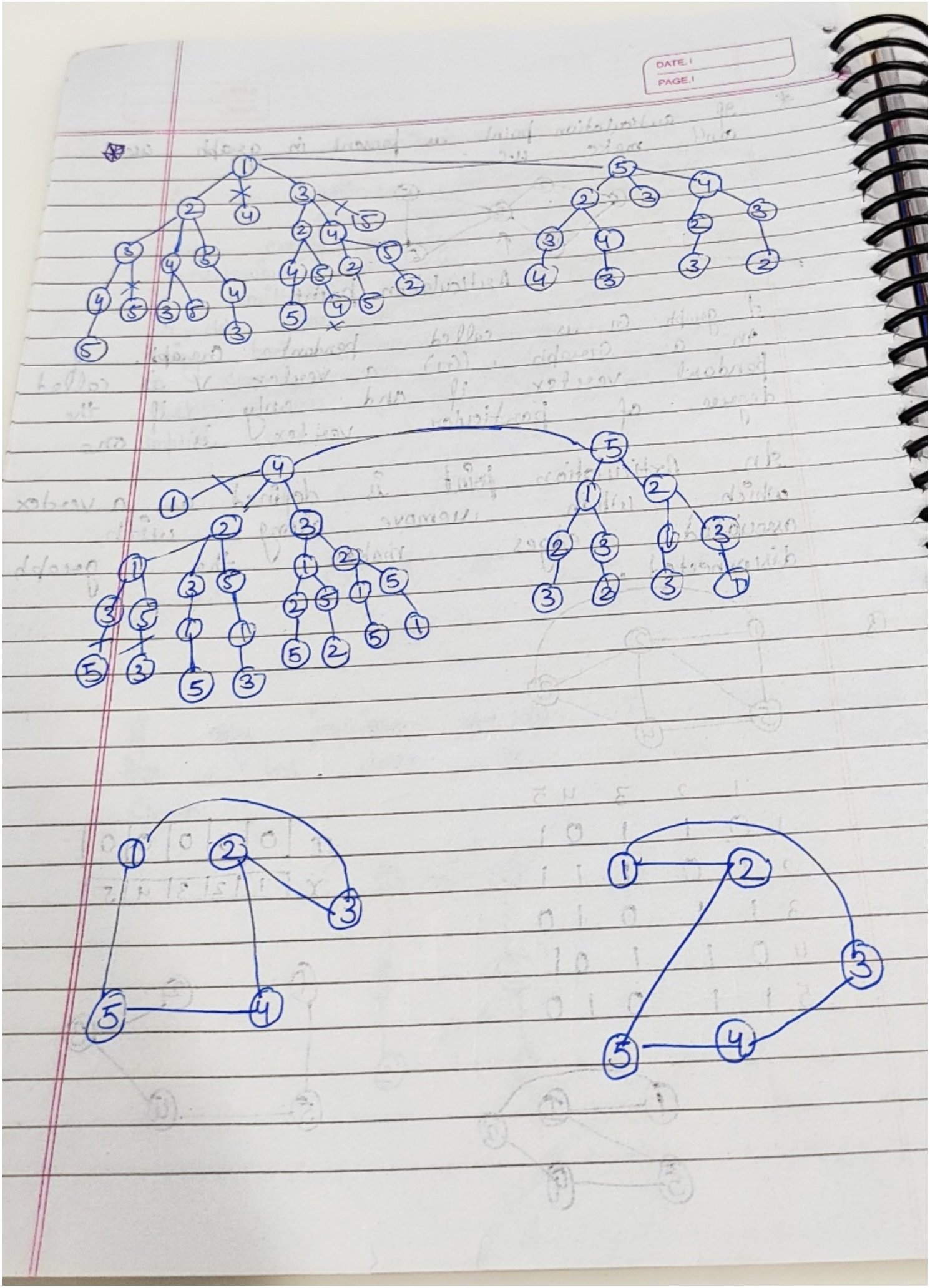


	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

x	0	0	0	0	0
x	1	2	3	4	5







HAMILTONIAN CYCLE (ALGORITHM)

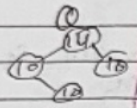
```
Hamiltonian cycle (k)
do
NEXT - VALUE (k)
if (x[k] = 0) then → checking for next value if value is not zero there is no path
return
if (k = n) then PRINT x[1 n] → if k is last place, then print
else HAMILTONIAN CYCLE (k+1) → if not last place, then find (k+1)
while (false)

NEXT - VALUE (k)
do
x[k] ← (x[k] + 1 mod n+1) → check for next possible value. The
if (x[k] = 0) then → value should not exist in Solution
return
if (G[x[k-1], x[k]] ≠ 0] then → New node must have
do for j ← 1 to k-1 → an edge to previous
do if (x[j] = x[k] then
break
if (j = k) then → If we got last place there must be an edge to first place
do if ((k < n) or ((k = n) and G[x(n), x(1)] ≠ 0)) then
return
while (false)
```

I 7, 5, 1, 6, 3, 2 G 5 : 33

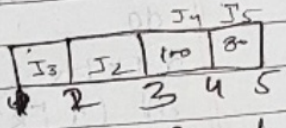
Q Find optimal sol^n of the knapsack instance $n=7$, $M=15$ $P=(10,5,15,7,6,18,13)$ and $W=(2,3,5,7,1,4,1)$ using greedy approach.

Q Construct optimal Binary search tree for Keys = {10, 12, 14, 16} and freq = {4, 2, 6, 3}

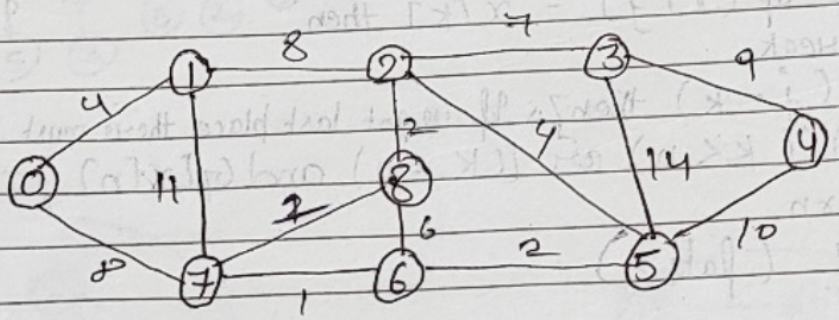


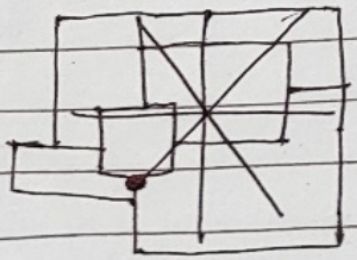
Q Consider the following task with their deadline and profit. Schedule the tasks to produce max profit.

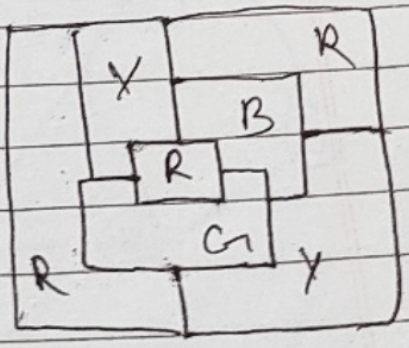
Jobs	J_1	J_2	J_3	J_4	J_5
Deadlines	2	2	1	3	4
Profit	20	60	40	100	80



Q Calculate MST for following graph using prim's 2, kruskal.







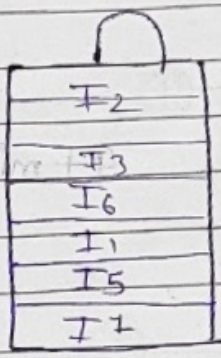
Q1 Ans

Item	Profit	weight	wi/wi
I1	10	2	5
I2	5	3	0.15
I3	15	5	3
I4	7	7	1
I5	6	1	6
I6	18	4	4.5
I7	13	1	13

Arrange value of Pi in ↓ order

Item	Profit	weight	Pi
I7	13	1	13
I5	6	1	6
I1	10	2	5
I6	18	4	4.5
I3	15	5	3
I4	7	7	1
I2	5	3	0.15

Now fill the bag



The capacity of bag is 15

Q2 Ans Keys ⇒ 10, 12, 14, 16
freq ⇒ 2, 4, 3, 1

	0	1	2	3
0	2	(8)¹	(14)¹	(18)²
1		4	(10)²	(13)²
2			3	(5)³
3				1

0	1	2	3
10	12	14	16
2	4	3	1

0	1
10	12
2	4

6 + min { 4 0 ; 2 1 }

6 + 2 = [8]¹

1	2
12	14
4	3

7 + min { 3 1 ; 4 2 } = [10]

2	3
14	16
3	1

4 + min { 1 2 ; 3 3 }

0	1	2
10	12	14
2	4	3

9 + min { 0 ; 2+3 1 ; 7 2 }

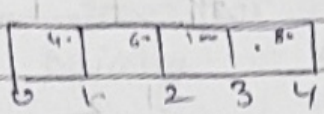
1	2	3
12	14	16
4	3	1

8 + min { 12 1 ; 5 2 ; 7 3 }

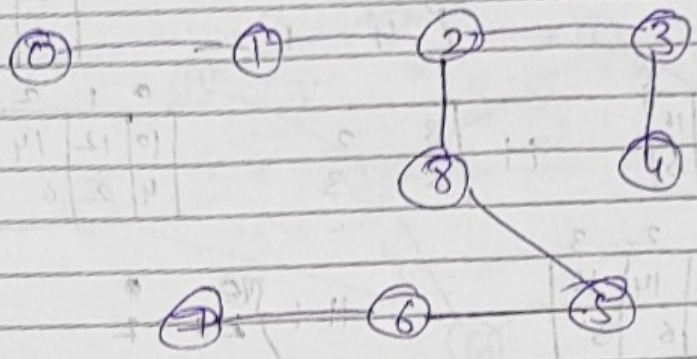
10	12	14	16
2	4	3	1

10 + min { 0, 1, 2, 3

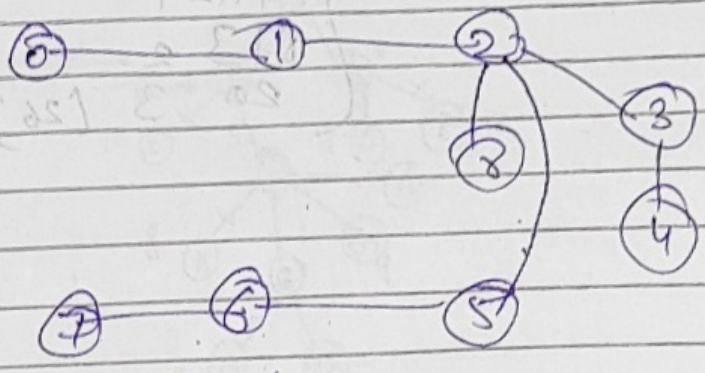
Q3



Q4

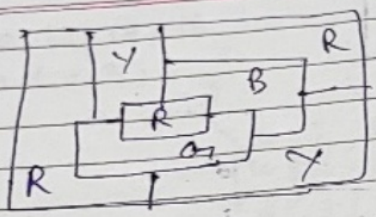


By Kruskal's method



By prims

Q5 Ans



Q Keys ⇒ 10, 12, 14, 16 ⇒
Freq ⇒ 4, 2, 6, 3

0	1
10	12
4	2

	0	1	2	3
0	4	$(8)^0$	20	26
1		2	$(10)^2$	$(16)^2$
2			6	$(12)^2$
3				3

$6 + \min \begin{cases} 2 & 0 \\ 4 & 1 \end{cases}$

1	2
12	14
2	6

$8 + \begin{cases} 6 & 1 \\ 2 & 2 \end{cases}$

2	3
14	16
6	3

$9 + \begin{bmatrix} 3 & 2 \\ 6 & 3 \end{bmatrix}$

0	1	2
10	12	14
4	2	6

$12 + \begin{cases} 8 & 0 \\ 4+6 & 1 \\ 2+6 & 2 \end{cases}$

1	2	3
12	14	16
2	6	3

$11 + \begin{cases} 10 & 0 \\ 2+3 & 1 \\ 2+6 & 2 \end{cases}$

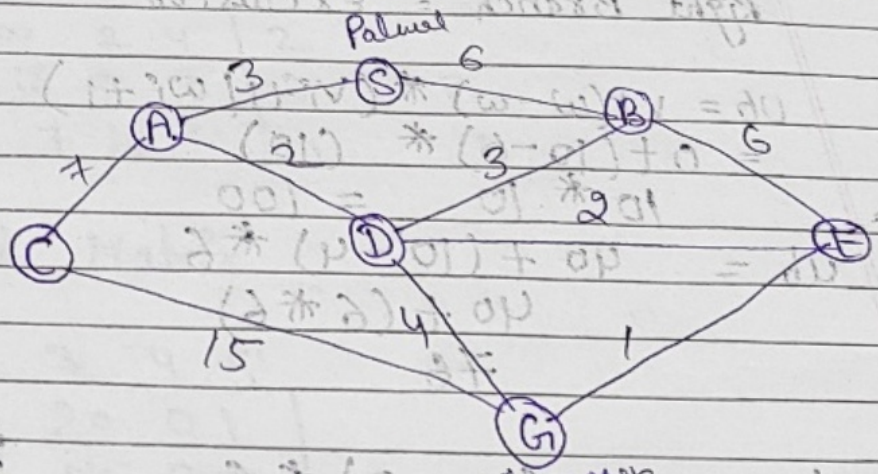
10	12	14	16
4	2	6	3

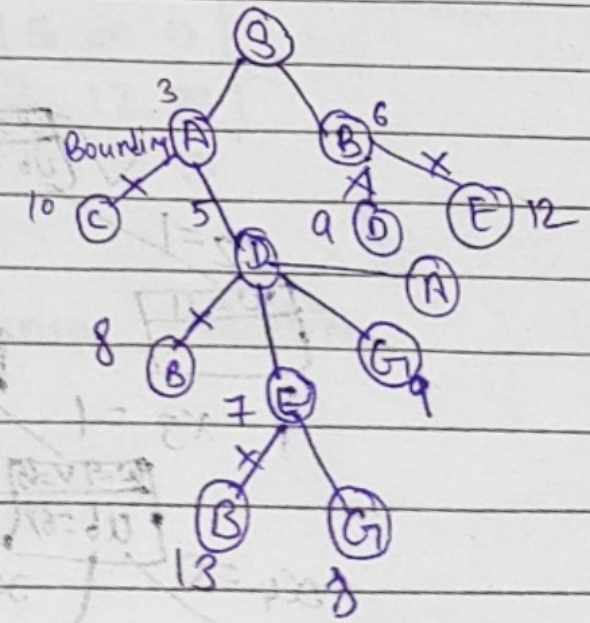
$15 + \begin{cases} 16 & 0 \\ 4+12 & 1 \\ 8+3 & 2 \\ 20 & 3 \quad (26) \end{cases}$

Unit => 4

Branch and Bound => Branching is the process of genrating subproblems.

* Bounding refers to ignoring partially solutions that can not be better than the current best solution.
* It is a search procedure to find the optimal soln
* It eliminates those path of a search space which does not contain better solution. In this method our technique we basically extend the cheapest path





DATE:
PAGE:

Q1) Knapsack using Branch and Bound

Items	VALUE	Weight
1	4	40
2	7	42
3	5	25
4	3	12

$w = 10$

upper bound $\Leftarrow$ $ub = v + (w - w) * (v_{i+1} / w_{i+1})$

We will make state space tree

Left Branch = inclusion

Right Branch = exclusion

$ub = v + (w - w) * (v_{i+1} / w_{i+1})$

$= 0 + (10 - 6) * (10)$

$10 * 10 = 100$

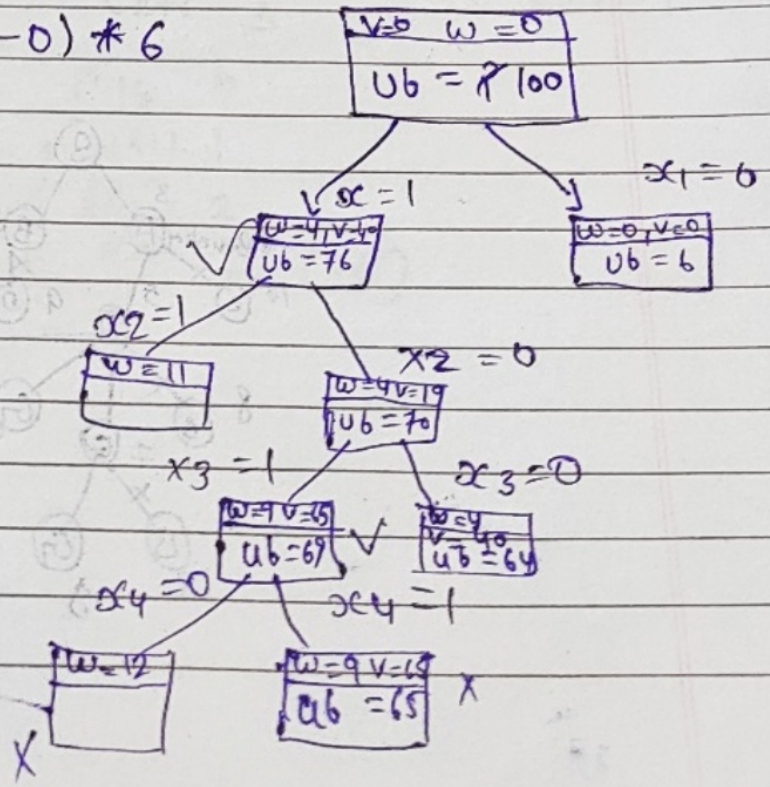
$ub = 40 + (10 - 4) * 6$

$40 + (6 * 6)$

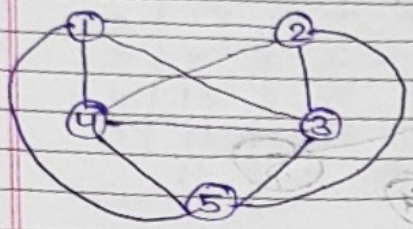
76

$ub = 0 + (10 - 0) * 6$

60



Travelling Salesperson using Branch and Bound



	1	2	3	4	5	
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

Step1 Reduced Matrix

	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞
	1	0	3	0	0

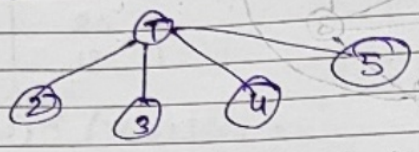
⇒ Row Reduction

Step2 Column Reduction

	1	2	3	4	5
1	∞	10	17	0	
2	12	∞	11	2	
3	0	3	∞	0	
4	15	3	12	∞	
5	11	0	0	12	

Step 3 Reduced Cost
$10+2+2+3+4 = 21$ $1+3=4$
$21+4=25$

upper bound $= \infty$
lower cost $= 25$



Step 4 $1 \rightarrow 2$
Row $= \infty$ Column ∞
$a_{21} = \infty$

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	∞	∞	11	2	0	0
3	0	∞	∞	0	12	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0

Row reduction of this II matrix
Column ⟶ II matrix
Matrix for Reduction of Row & Column is same

Step 5 $1 \rightarrow 2$
$c(1,2) + r + r' \rightarrow$ any other reduced cost
$10 + 25 + 0$
$= 35$

1→3

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	12	∞	∞	2	6	0
3	∞	3	∞	∞	0	2
4	15	3	∞	∞	0	0
5	11	0	∞	12	∞	0
	11	0	0	0	0	

for row reduction matrix is same
for colums reduction 11 — by all matrix

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	2	6
3	∞	3	∞	0	2
4	4	3	∞	∞	0
5	0	0	∞	12	∞

cu(1,3) + r + r'
17 + 25 + 11
53

1→4

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	12	∞	11	∞	0	0
3	0	3	∞	∞	2	0
4	∞	3	12	∞	0	0
5	11	0	0	∞	∞	0
	0	0	0	0	0	

0 + 25 + 0 = 25

1 → 5

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	12	∞	11	2	∞	2
3	0	3	∞	2	∞	0
4	15	3	12	∞	∞	3
5	∞	0	0	12	∞	0
	0	0	0	0		

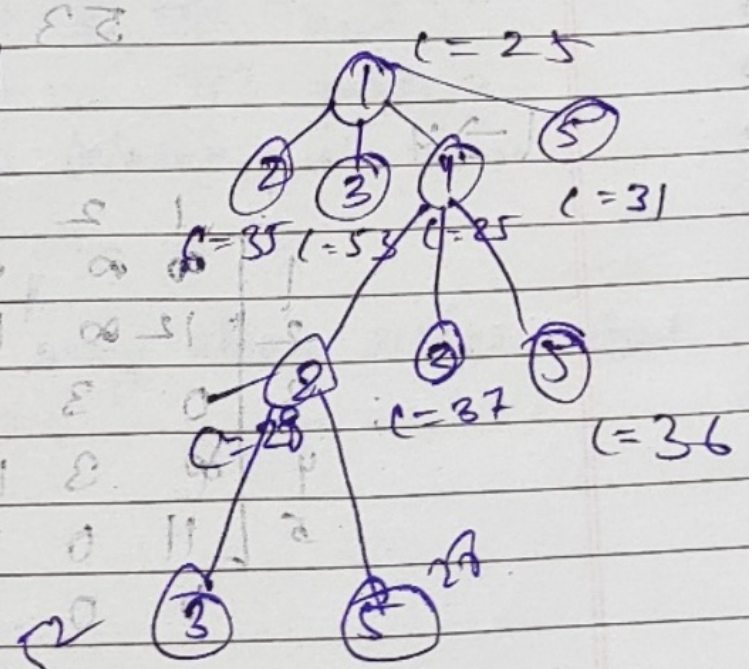
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	∞	0	0	12	∞

1 + 25 + 5

31

lowest Cost matrix

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞



4 → 2

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

No column or row reduction

3 + 25 + 0 = 28

4 → 3

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	12	∞	∞	∞	0	0
3	0	∞	∞	∞	2	0
4	∞	∞	∞	∞	∞	0
5	11	∞	∞	∞	∞	0
	0	0	0	0	0	

12 + 25 + 0 = 37

4 → 5

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	12	∞	11	∞	∞	11
3	0	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞	0
5	11	∞	0	∞	∞	0
	0	0	11	0	0	

0 + 25 + 11 = 36

DATE:
PAGE:

Unit - 5

Polynomial Time Taking problem

1) Linear Search : n
2) Binary Search : $\log n$
3) Merge Sort : $n \log n$
4) Insertion Sort : n^2

Exponential Time Taking problem

0/1 Knapsack → 2^n
TSP → 2^n
Sum of Subset → 2^n
Graph coloring → 2^n

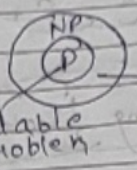
Class problem

A problem which can be solved in polynomial time E.g. Searching, Sorting, etc.

Class NP (Non-Deterministic Polynomial time :-

A problem which can not be solved in polynomial time but can be verified in polynomial time.
E.g 0/1 knapsack, Sudoku.

Diagram :-

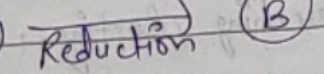


Deterministic Algo :-

The algo which we generally use where we know each & every step and how it works.
Eg : Linear Search, Binary Search

Non Deterministic Algo :-

Reduction



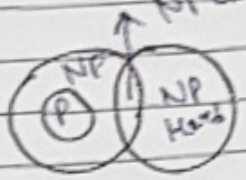
Let A and B are two problem. A reduces to B if there is a way to solve A by Deterministic algo that solve B in Polynomial time.

NP Hard Problem :- A problem is NP Hard if every problem in NP can

be polynomial reduced to it.

NP Complete problems :-

A problem is NP complete if it is NP and NP hard.



P = Searching
NP = 0/1 knapsack

NP Hard : Decision problems
NP Complete : shortest path