Backtracking Previous year Question

2018-19

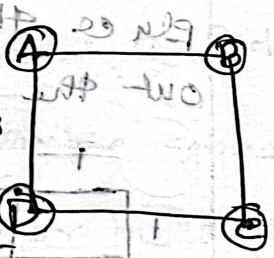
16-17

a) Graph coloring problem involves assigning colors to certain elements of a graph. In other words, the process of assigning colors to the vertices such that no two adjacent vertices have the same color is called graph coloring. This is also known as vertex coloring.

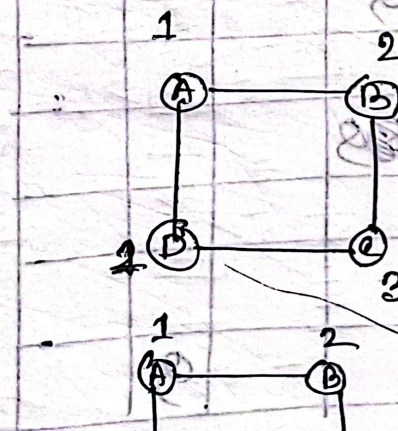
If the degree of given graph is d, then we can color it with d+1 colors. The least number of colors needed to color the graph is called chromatic number.

We can use backtracking approach for assigning different colors to adjacent vertices as follows :—

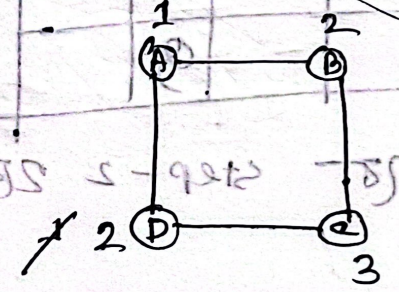
Step-1 : A graph G consists of vertices A to D. There are three colors used Red Green and Blue. We will number them out. That means 1 indicates Red, 2 indicates Green, 3 indicates Blue.



Step-2 :



Step-3 :



Thus The graph coloring problem is solved.

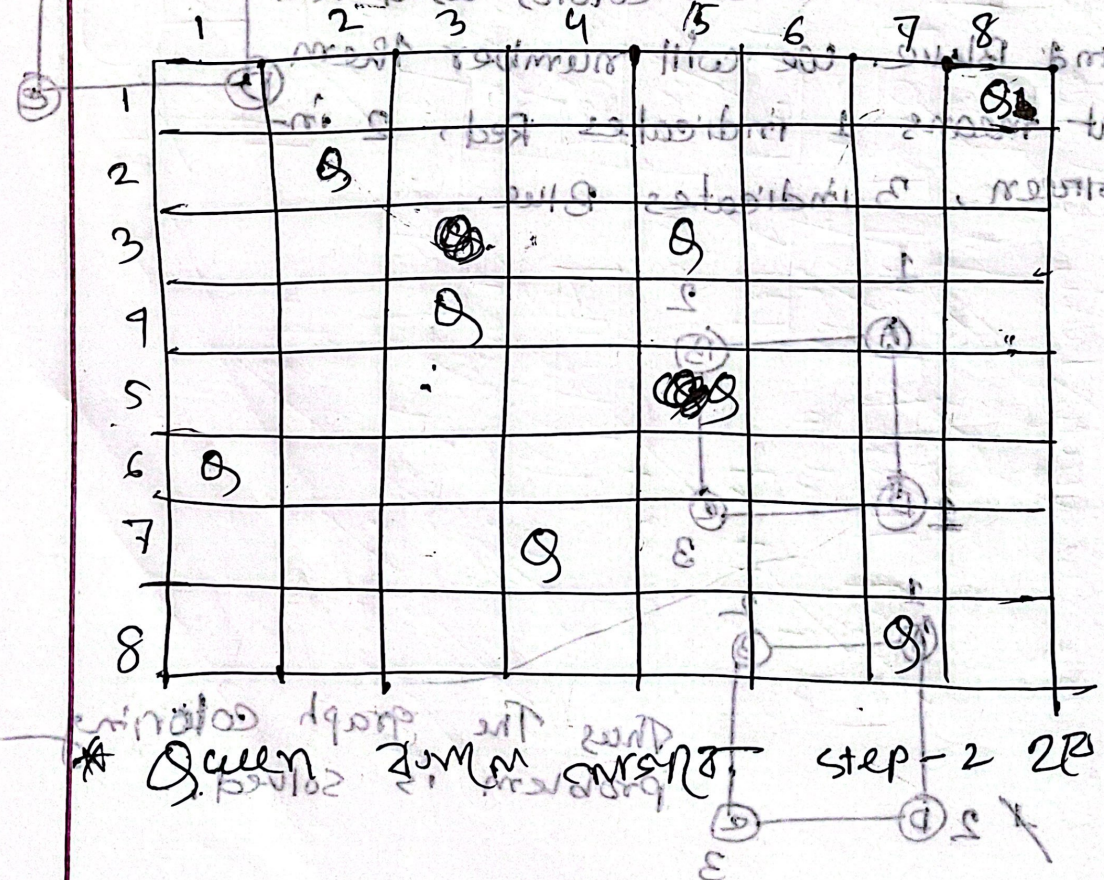
b) **Backtracking**: Backtracking is the most general technique to solve a problem or finding the set of solutions of a problem which satisfies some constraints.

Principle of Backtracking:

(i) find a set of soln from the all feasible soln.
(ii) Use recursive calling to find a soln.
(iii) Use Brute force approach.

c) 8 queens problem for a feasible sequence (8, 2, 5, 3).

As the feasible sequence is given, we will place the queens accordingly and then try out the other remaining place.



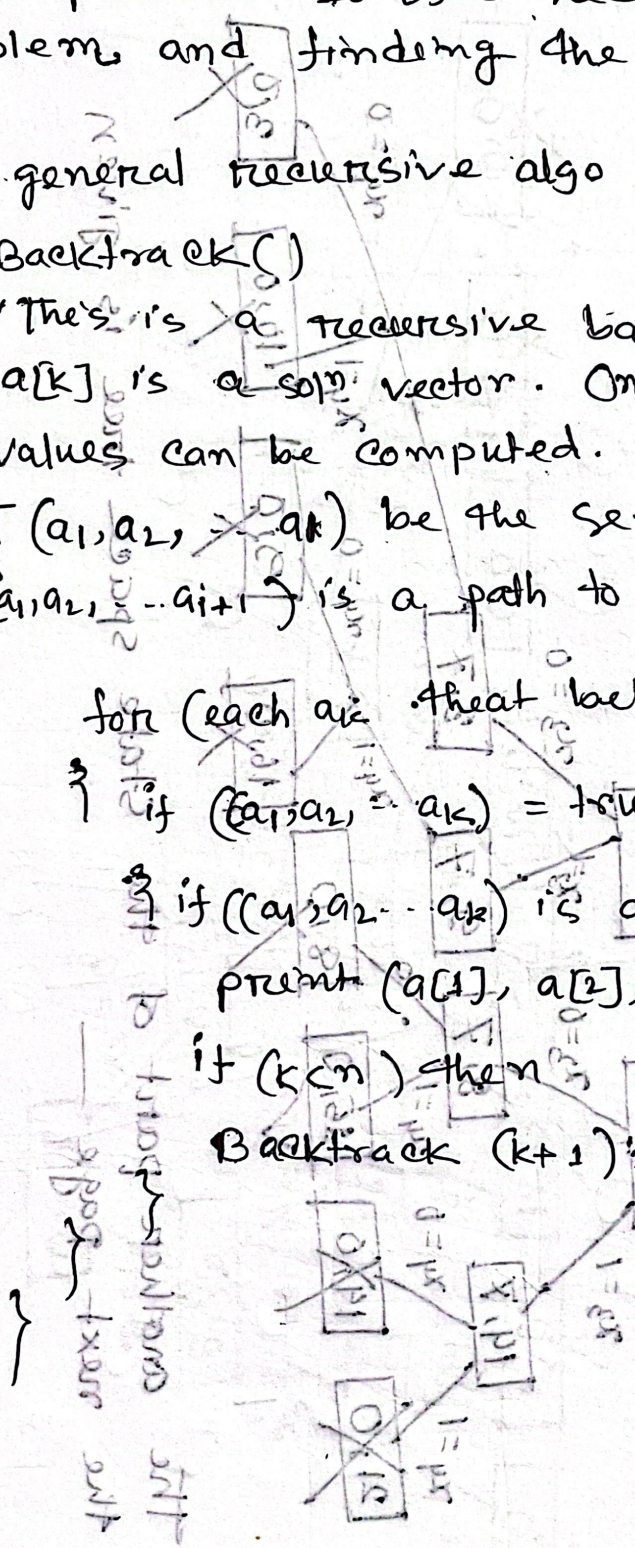
Scanned with CamScanner

2017-18

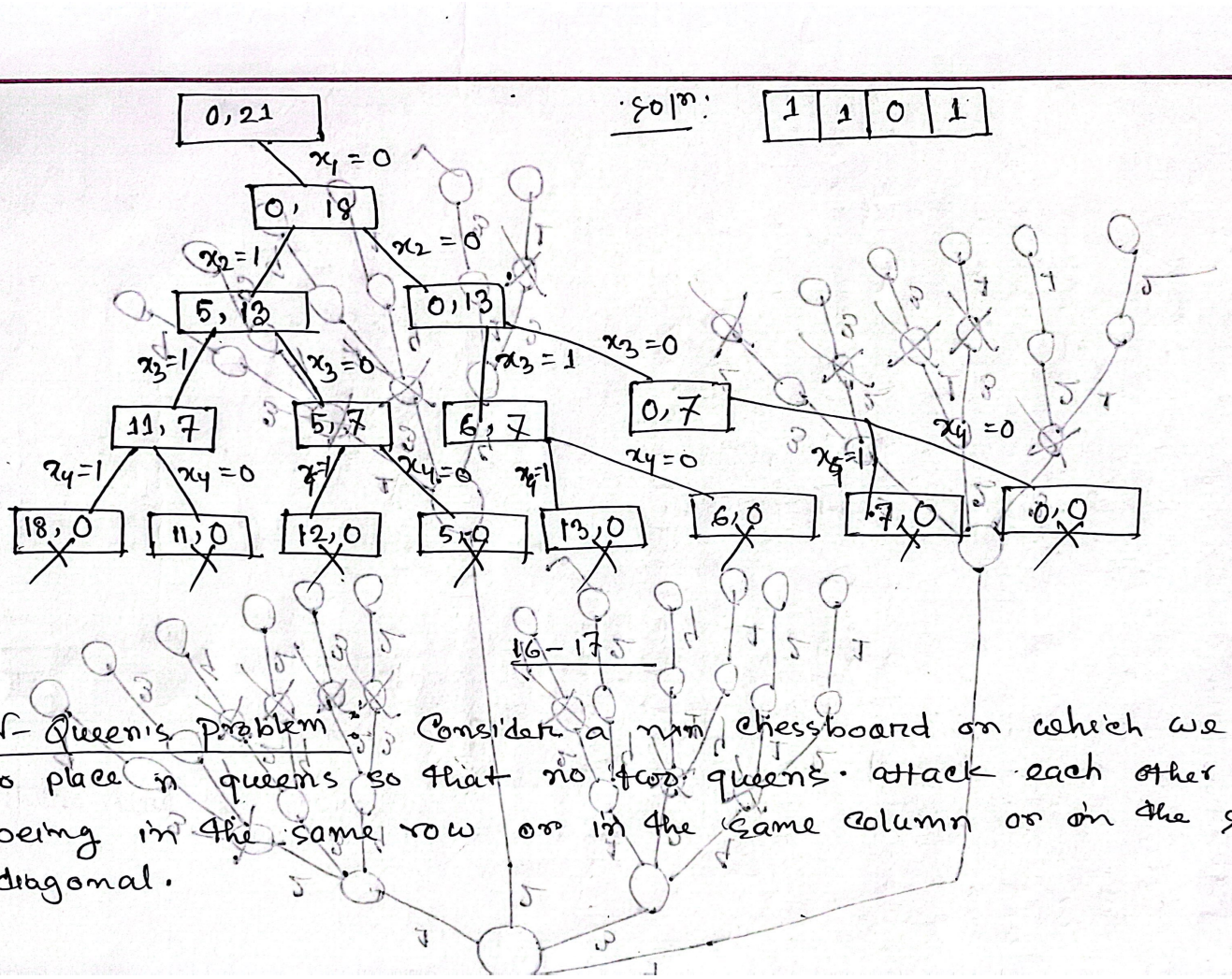
a) **Backtracking :** Backtracking is a general method to finding a set of solutions from the all possible solutions of a problem. It used recursive calling to solve a problem and finding the solutions.

The general recursive algo for backtracking :—

```
Backtrack()
// This is a recursive backtracking algorithm.
// a[k] is a soln vector. On entering (k-1) remaining next.
// values can be computed.
// T(a1, a2, ... ak) be the set of all values for a(i+1), such tha
// (a1, a2, ... ai+1) is a path to problem state.
{
    for (each ak that belong to T((a1, a2, ... ak-1) do
    {
        if (a1, a2, ... ak) = true) then // feasible sequence
        {
            if ((a1, a2 ... ak) is a path to answer node then
                print (a[1], a[2], ... a[k]);
            if (k<n) then
                Backtrack (k+1); // find the next set.
        }
    }
}
```



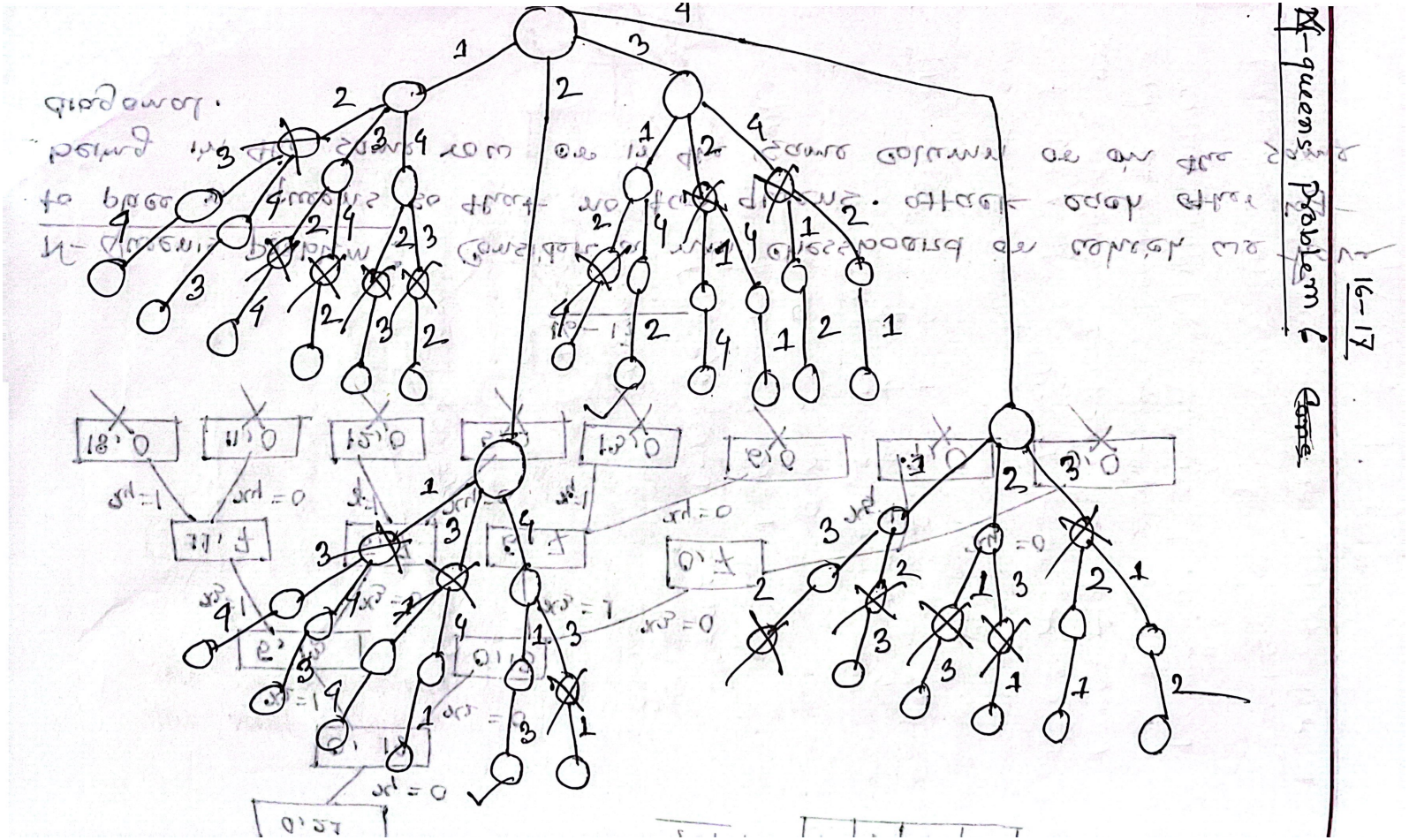
The another part of the state space tree is given the next page —



Soln: | 1 | 1 | 0 | 1 |

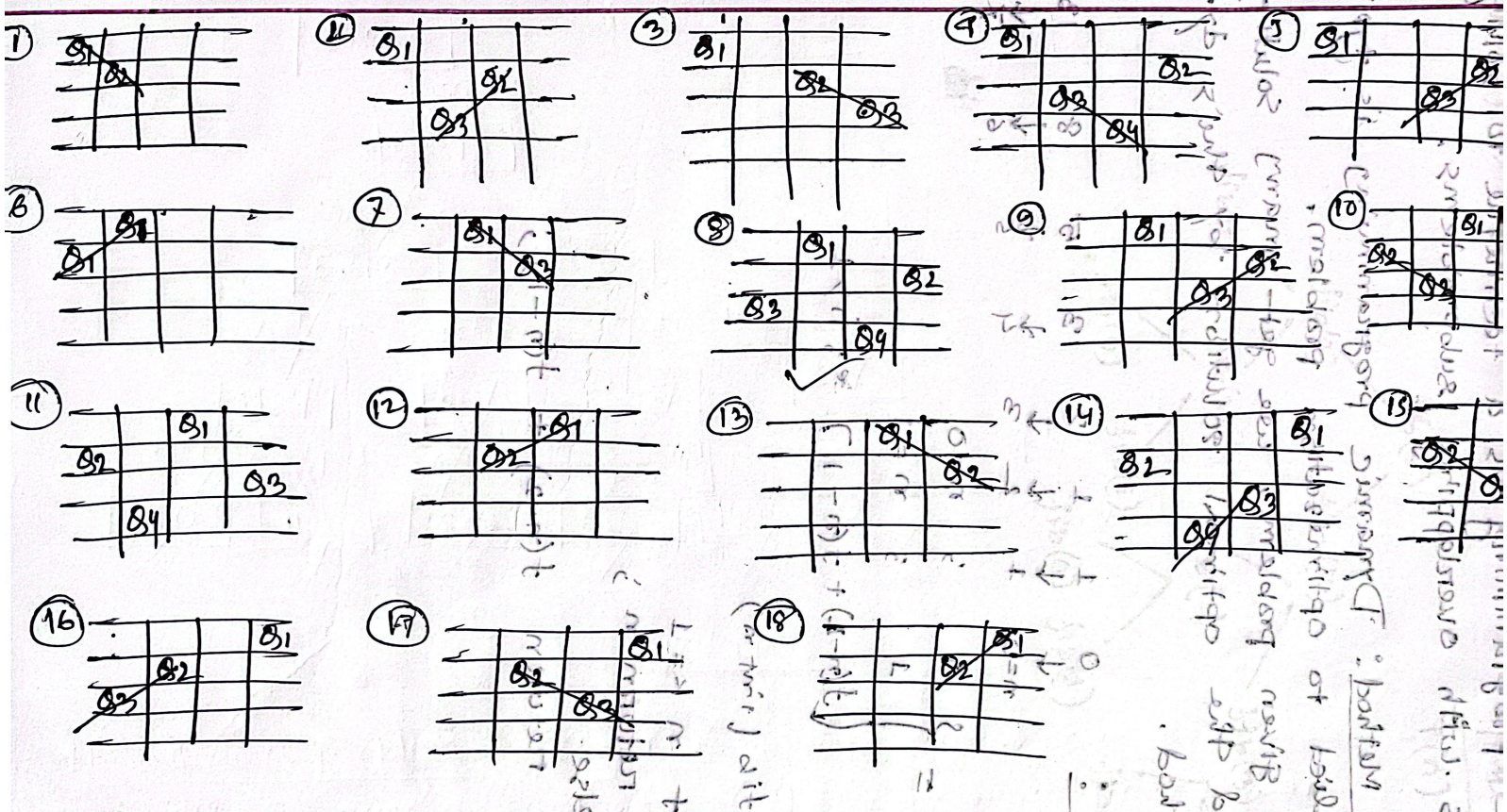
16-17

N-Queen's problem: Consider a $n \times n$ chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.



16-17

4-queens Problem :



There are two solutions:

2	4	1	3

,

3	1	4	2

Dynamic Programming

⇒ Dynamic programming is a technique to solving problems with overlapping subproblems.

⇒ **General Method:** Dynamic programming is typically applied to optimization problem. For each given problem, we get many solution and find the optimal solution by the dynamic method.

⇒ **fibonacci:**

0 1 1 2 3 5 8 13

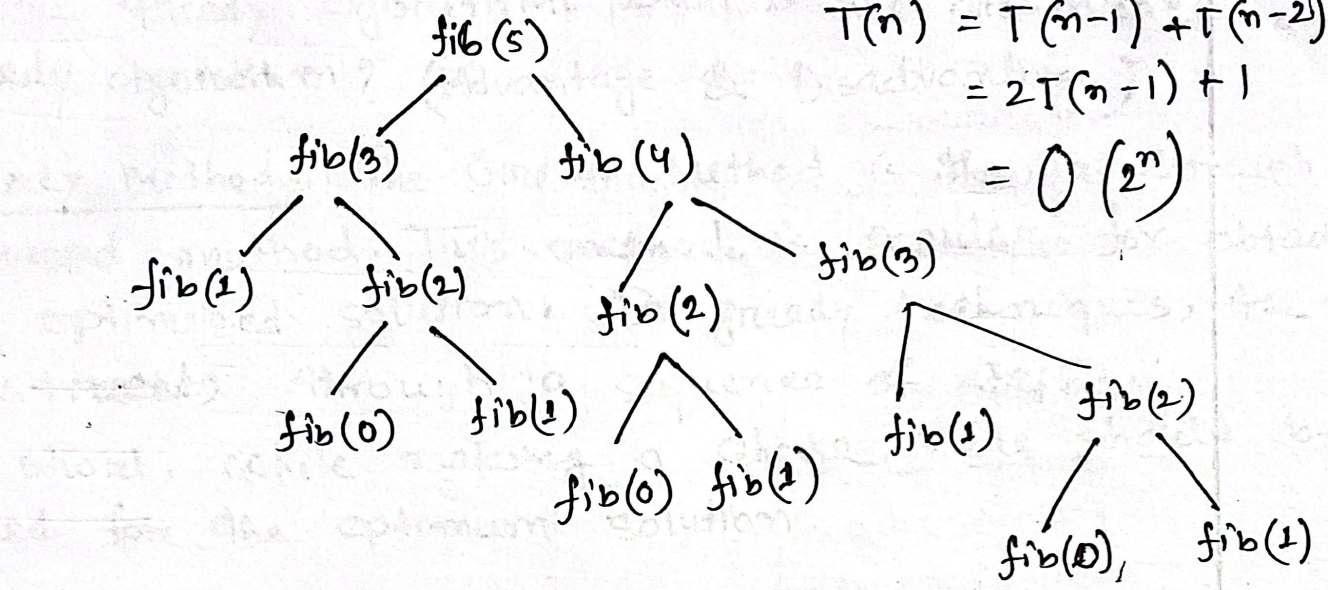
n=0 1 2 3 4 5 6 7

$$fib(n) = \begin{cases} 0 & ; n=0 \\ 1 & ; n=1 \\ f(n-2)+f(n-1) & ; n>1 \end{cases}$$

```
int fib (int n)
{
  if n <= 1
    return n;
  else
    return f(n-2) + f(n-1);
}
```

Recursive call and return

fib(5) = ?



Time complexity.

$T(n) = T(n-1) + T(n-2) + 1$

$= 2T(n-1) + 1$

$= O(2^n)$

Greedy Method

Q1. What is greedy algorithm? State the general principle of the greedy algorithm. What is the drawback of greedy algorithm? (Advantage & Disadvantage)

Greedy Method: The Greedy Method is a straight forward method. This method is popular for obtaining the optimized solution. In greedy techniques, the solution is constructed through a sequence of steps.

In short, while making a choice there should be greed for the optimum solution.

General Method: **Algorithm**

```
Greedy(D,n)
// In Greedy approach D is a domain
// from which soln is to be obtained of size n
// Initially assume
solution ← 0
for i ← 1 to n do
{
    Check if the solution is feasible or not → Make a feasible and select the optimum soln.
}
return soln.
```

General principles of greedy algorithm

- Activity selection problem
  - Optimal substructure
  - recursive soln
  - greedy choice property
  - recursive algorithm.
- Minimum spanning tree
  - Greedy Algo.
  - Prim's Algo.
  - Def: Cuts, light edg

Applications

- Knapsack Algo.
- Prim's Algo.
- Kruskal's Algo.
- Finding shortest path
- Job Sequencing with deadlines.
- optimal storage with tapes

Advantage:

- Easier to implement
- Much faster to execute
- Used to finding optimum solution.

Disadvantage / Drawback

- Not always find the optimal soln.
- It may not produce the best possible outcomes
- It is very difficult to understand.

Step's of Greedy methods :

1. Select some soln from input domain.
2. Check the soln is feasible or not.
3. From a set of feasible soln — particular soln that satisfy or nearly satisfy to the object is called the optimal soln.
4. As greedy method work in stage. At each stage only one input is considered at each time. Based on the's input — it is decided wheather particular soln gives the optimal soln or not.

Optimization problem : Optimization problem is to find out either min or max result. To solve optimization problem, there are several method, like —

(i) Greedy Method (ii) Dynamic programming (iii) Branch-N-

16.2.1 : Suppose that instead of always selecting the first activity to finish. We instead select the le activity to start that is compatible with all pre viously selected activities. Describe how that appoa is greedy algorithm and prove that is yields an optimal soln.

Knapsack problem: Suppose there are n objects from $i = 1, 2, 3, \ldots, n$. Each object have some positive weight w_i and some profit associated with the object which denotes as p_i. This knapsack carry at the most weight W.

* Chose only those object which gives maximum profit.

* Total weight of object should be $\leq W$, then we can obtained feasible soln —

maximized $\sum_{n} p_i x_i$ subject to $\sum_{n} w_i x_i \leq m$; $0 \leq i \leq n$, $0 \leq x_i \leq 1$

* **Fractional** : $n = 7$, $m = 15$

Object :	1	2	3	4	5	6	7
profit :	12	5	16	7	9	11	6
weight :	3	1	4	2	9	4	3
$\max\left(\frac{P}{W}\right)$ =	4	5	4	3.5	1	2.75	2

Object	profit	weight	remaining weight
2	5	1	$15 - 1 = 14$
1	12	3	$14 - 3 = 11$
3	16	4	$11 - 4 = 7$
4	7	2	$7 - 2 = 5$
6	11	4	$5 - 4 = 1$
7	$(6/3) = 2$	1	$1 - 1 = 0$

N = 8, M = 110

$p_i = \{11, 21, 31, 33, 43, 53, 55, 65\}$

$w_i = \{1, 11, 21, 23, 33, 43, 45, 55\}$

$p/w = \{11, 1.9, 1.47, 1.43, 1.303, 1.23, 1.22, 1.18\}$

Item	profit	weight	Remaining weight
1	11	1	110 − 1 = 109
2	21	11	109 − 11 = 98
3	31	21	98 − 21 = 77
4	33	23	77 − 23 = 54
5	43	33	54 − 33 = 21
6	$53 \times \frac{21}{43}$	21	21 − 21 = 0

43 কেজির মধ্যে
21 কেজি বসানো
যায় $\frac{21}{43}$

Object	profit	weight
1	11	1
2	21	11
3	31	21
4	33	23
5	43	33
6	$53 \times \frac{21}{43}$	21

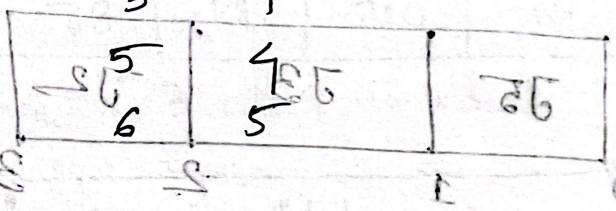
maximum profit = $11 + 21 + 31 + 33 + 43 + \left(53 \times \frac{21}{43}\right)$

State 0/1 knapsack problem:

The 0/1 knapsack problem is a classic optimization problem where a set of ~~elements~~ items with given weights and profits are to be packed in a knapsack with a limited capacity. It is called 0/1 knapsack because if the weight count then fill with 1 otherwise 0. So, it is 0/1 knapsack.

* Solve the 0/n knapsack pro. [max = 5, n = 4]

Item :	1	2	3	4
weight :	3	2	5	4
profit :	4	3	6	5



→ weight

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

Job sequencing problem

Job id : 1 2 3 4 5

profit : 10 20 15

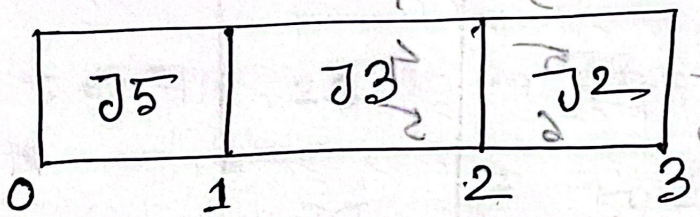
deadlines : 2 2 1

Soln :

Job id : 3 5 1 2 4

profit : 20 15 10 5 1

deadlines : 2 2 1 3 3



Job id	Profit	Deadlines	Slot assign	
J3	20	2	[1, 2]	0
J5	15	2	[0, 1]	1
J1	10	1	rejected	2
J2	5	3	[2, 3]	3
J4	1	3	rejected	

Solution : J3, J5, J2

Max profit : 20 + 15 + 5 = 40

Job id:	1	2	3	4	5	6	7	8	9	10
profit:	20	5	4	8	30	10	6	18	15	25
deadlines:	4	3	2	6	8	2	3	4	5	5

Q find the max profit from max deadlines.

Soln:

Job id:	5	10	1	8	9	6	4	7	2	3
profit:	30	25	20	18	15	10	8	6	5	4
deadline:	8	5	4	3	4	1	3	3	3	2

J6	J9	J8	J1	J10	J4		J5
0	1	2	3	4	5	6	7

Job id	profit	deadlines	Slot assign
J5	30	8	[7,8]
J10	25	5	[4,5]
J1	20	4	[3,4]
J8	18	3	[2,3]
J9	15	4	[1,2]
J6	10	1	[0,1]
J4	8	6	[5,6]
J7	6	3	rejected
J2	5	3	rejected
J3	4	2	rejected

Max profit = 126

* Explain how job sequencing with deadline problems can be solved using the greedy approach

→ Sort the jobs in decreasing order of their profit.

→ Find the highest deadlines and draw a grant chart up to that deadlines.

→ Now we need to assign time slots to individual job id.

→ Now - pick each job one by one and check if the max possible time slot for the job i.e it's deadlines is assigned to another job or not. if it is not filled yet, assign the slot to the current job.

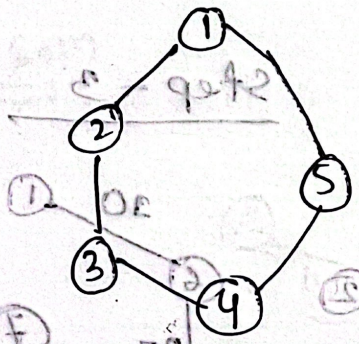
→ Otherwise, search for any empty time slot less than the deadline of the current job. if such a slot is found, assign it to the current job id and move to the next job.

→ Continue this process until all the feasible jobs are allocated to their time slot.

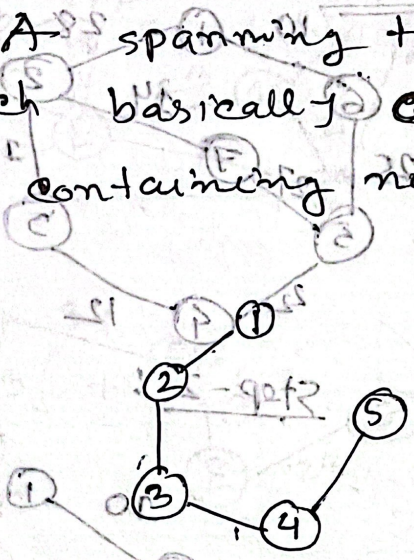
→ In the end, we can calculate the profit for all allocated feasible jobs.

Minimum cost spanning tree:

Spanning tree: A spanning tree is a subgraph of a graph G which basically a tree that contains all the vertices containing no circle.



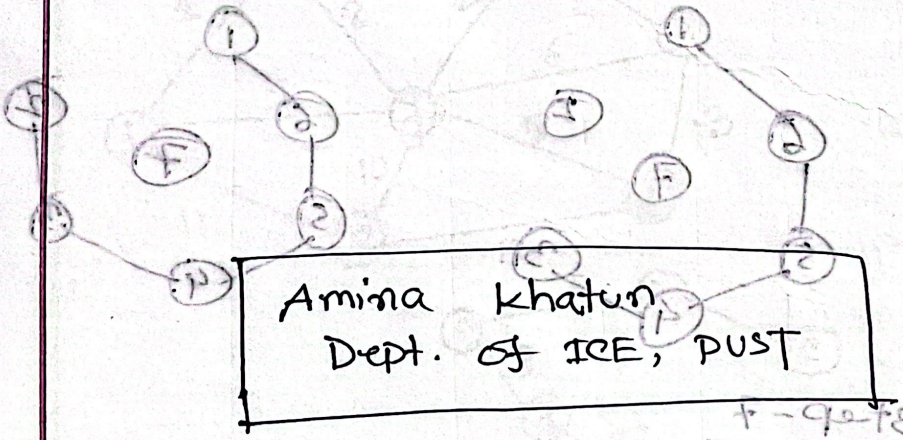
Graph



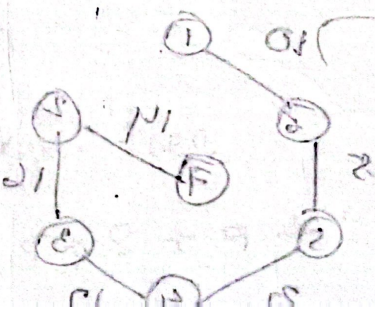
Spanning tree.

For single cycle spanning tree $= {}^{n}C_{n-1}$ Total edge.

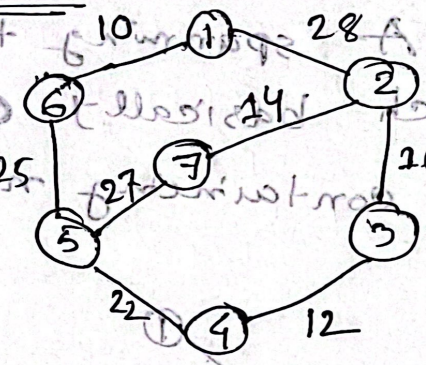
for multistage $= {}^{|e|}C_{|v-1|}$ — no of cycle



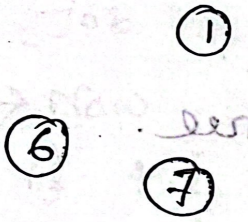
Amina Khatun,
Dept. of ICE, PUST



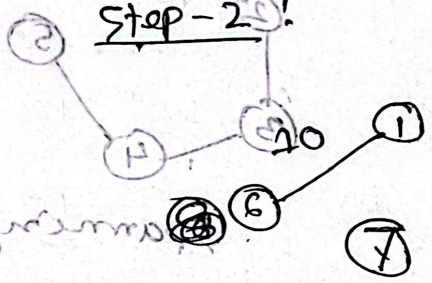
Prim's Algorithm



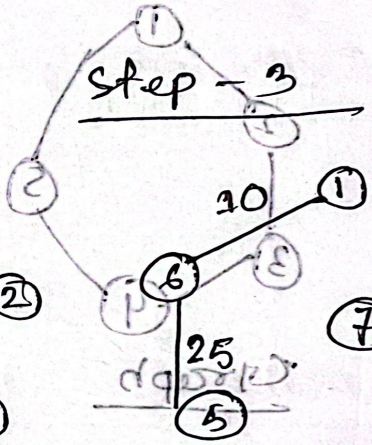
Step-1 :

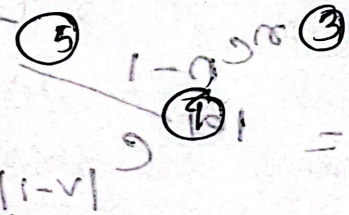


Step-2 :

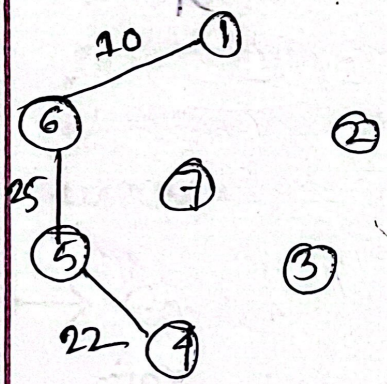


Step-3



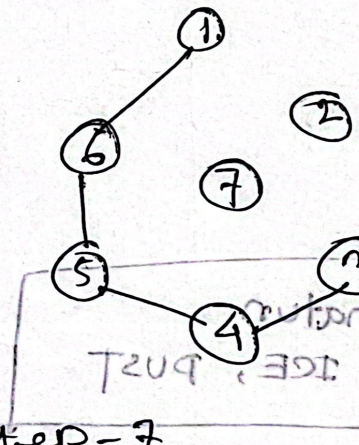


Step-4

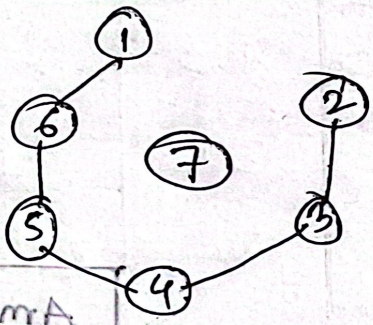


$W = 10+25+22$
$= 57$

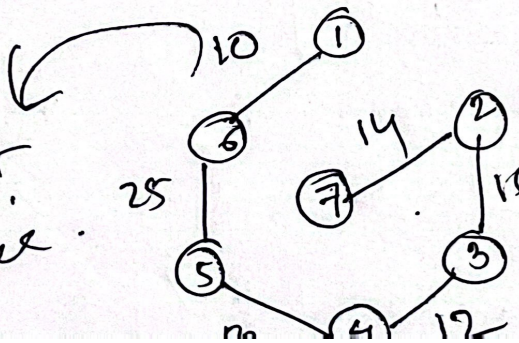
Step-5



Step-6



Step-7



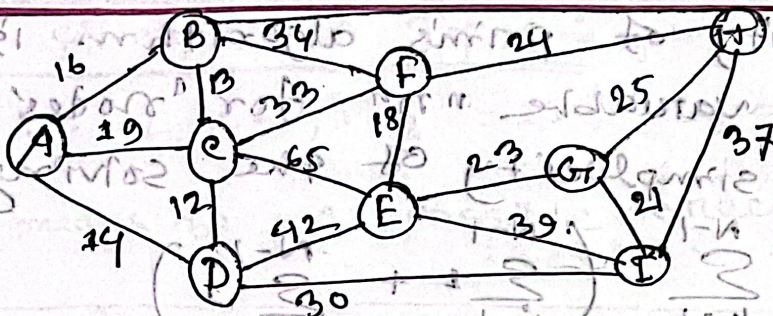
Min Cost

$= 10+25+22+12+16+14$

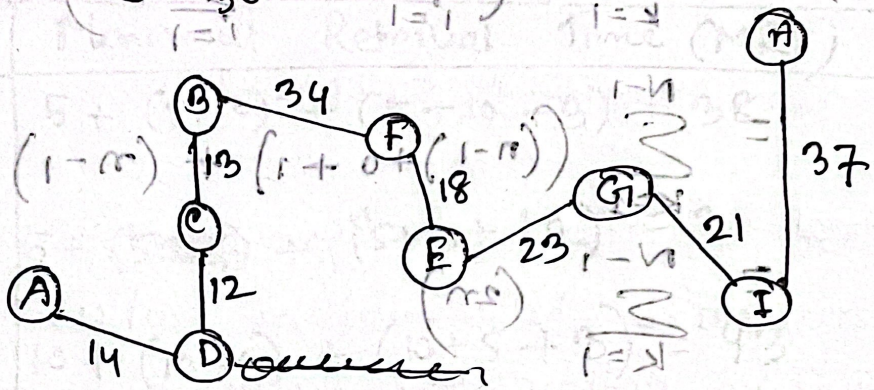
$= 99$

17-18

6.b)



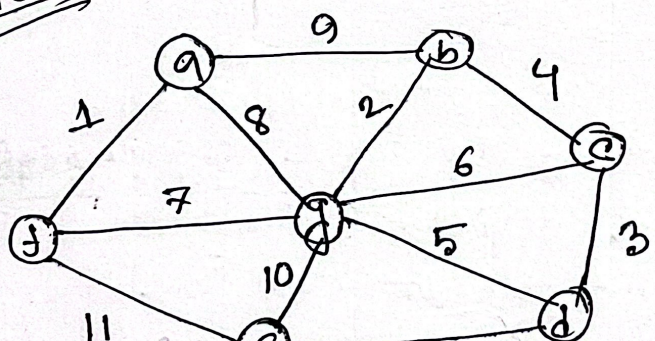
Soln

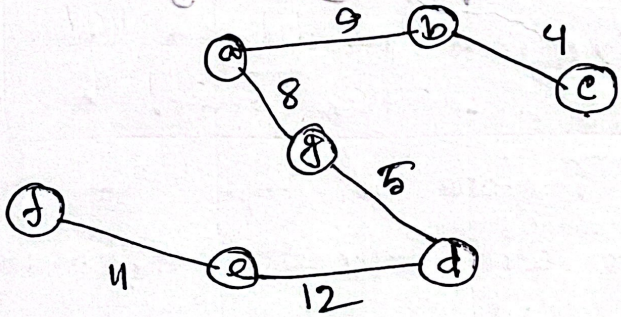


Minimum spanning tree.

Min cost = 14 + 12 + 13 + 34 + 18 + 23 + 21 + 37

18-19





Maximum spa. tree

Max cost = 11 + 12 + 5 + 8 + 9 + 4 =

* Time complexity of prim's algorithm is $\Theta(|V^2|$

⇒ We take variable 'n' for 'nodes' for the sake of simplicity of the solving eqn. The

$$T(n) = \sum_{k=1}^{N-1} \left(\sum_{i=1}^{N-1} 1 + \sum_{i=1}^{N-1} 1 \right)$$

$$= \sum_{k=1}^{N-1} \left((n-1) + 0 + 1 \right) + (n-1) + 0 + 1$$

$$= \sum_{k=1}^{N-1} (2n)$$

$$= 2n \sum_{k=1}^{N-1} 1$$

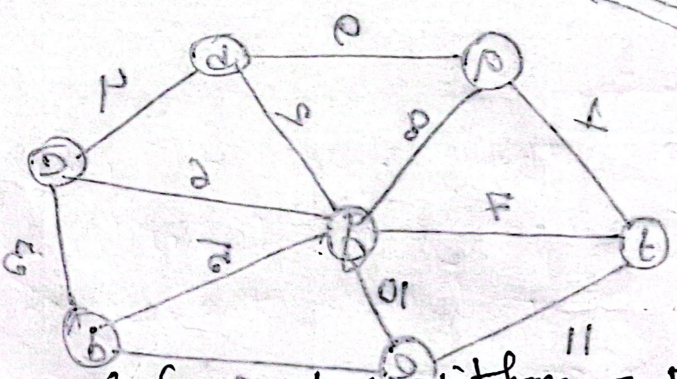
$$= 2n \{ (n-1) + 1 - 1 \}$$

$$= 2n(n-1)$$

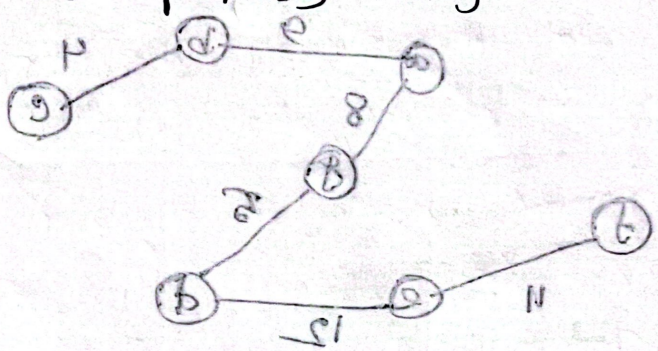
$$= 2n^2 - 2n$$

$$T(n) = n^2$$

$$T(n) = \Theta(n^2)$$



The complexity of prim's algorithm = $\Theta(|V^2|$



Optimal storage on Tapes

$n = 3, L_1 = 5, L_2 = 10, L_3 = 3$

$n = 3$ means $3! = 6$ possible ordering.

Ordering	Minimal Retrival Time (MRT)
1, 2, 3	$5 + (5+10) + (5+10+3) = 38$
1, 3, 2	$5 + (5+3) + (5+3+10) = 31$
2, 1, 3	$10 + (10+5) + (10+5+3) = 43$
2, 3, 1	
3, 1, 2	
3, 2, 1	

Computed

Tapes

Dynamic programming

⇒ Dynamic programming is a technique for solving a problem with overlapping subproblems. In this method, each subproblems is solving only once. The result of each subproblems is recorded in a table from which we can obtain the soln to original subproblems.

⇒ General procedure of dynamic programming.

(i) Characterize the structure value of optimal soln.
(ii) Recursive define the value of optimal soln.
(iii) Compute the value of optimal soln typically bottom up fashion.
(iv) Compute the optimal soln from the computed information.

⇒ Why needed DP?

The dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their result can be reused. Mostly this algorithm used for solved optimization problems. Optimization problems means that we are trying to find out the min or max soln of the problem. Before solving the inhand sub-problem, the algo will try to examine the result of the previously solved sub-problems.

* State the principle of optimality

- The dynamic programming algo obtain the soln using principle of optimality.

The principle of optimality states that, In an optimal sequence of decision or choise, each subsequence must also be optimal.

Greedy	Dynamic
→ used for obtaining optimal soln.	→ same
→ It has a set of feasible soln and picks up optimal soln.	→ No feasible set of soln.
→ There is no guarantee of getting optimal soln.	→ Guaranteed.
→ faster	→ slower
→ Ex: fractional knapsack	→ Ex: 0/1 knapsack

Fibonacci Series

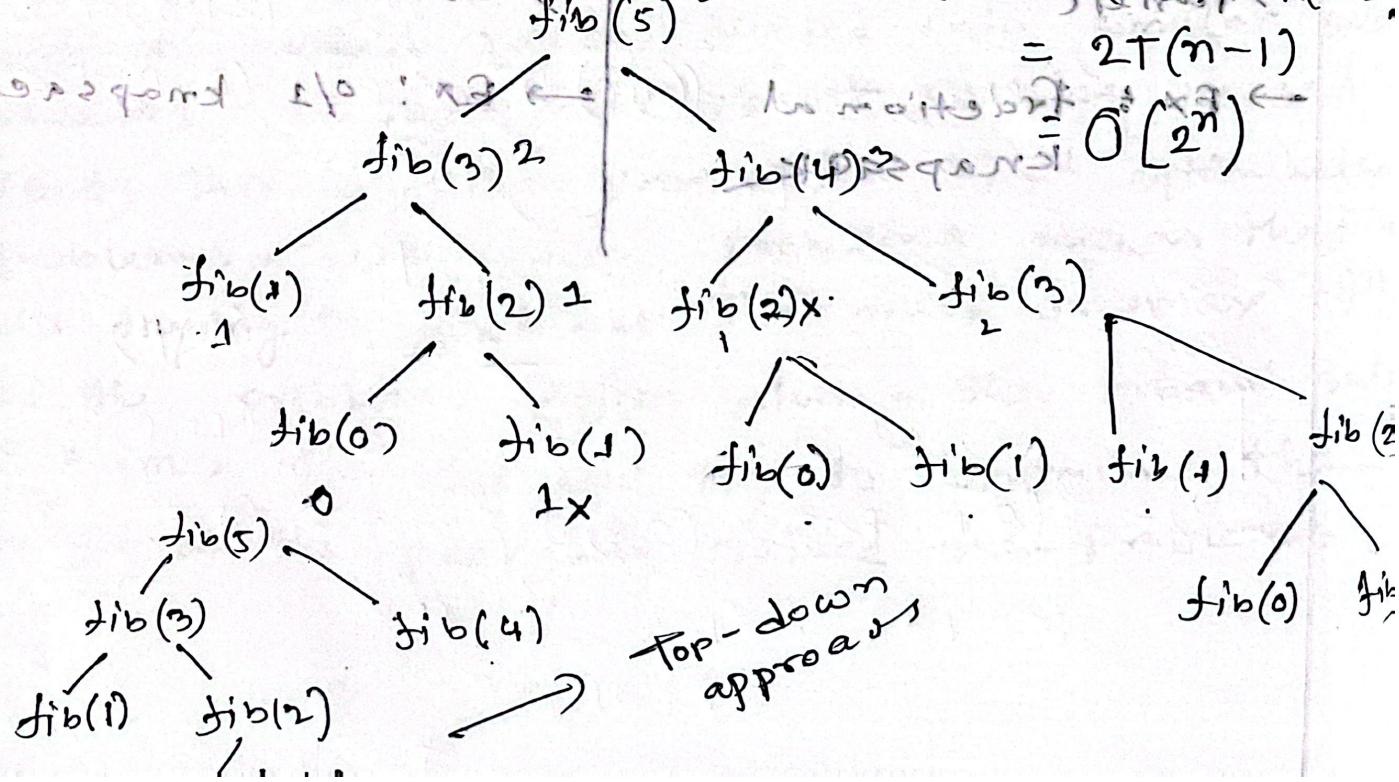
0 1 1 2 3 5 8 13 ...

n = 0 1 2 3 4 5 6 7

$$fib(n) = \begin{cases} 0 ; & n = 0 \\ 1 ; & n = 1 \\ fib(n-2) + fib(n-1) ; & n > 1 \end{cases}$$

```
int fib (int n)
{
  if (n <= 1)
    return n;
  else
    return fib(n-2) + fib(n-1);
}
```

* fib (5) = ?



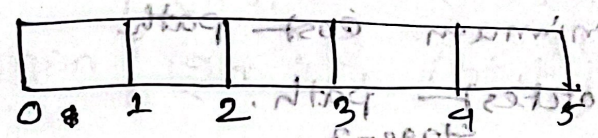
Top-down approach

Time complexity

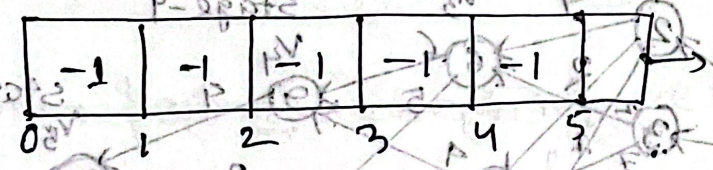
$$T(n) = T(n-1) + T(n-2)$$
$$= 2T(n-1)$$
$$\approx O(2^n)$$

Memorization Process (follow top-down approach)

Step-1:

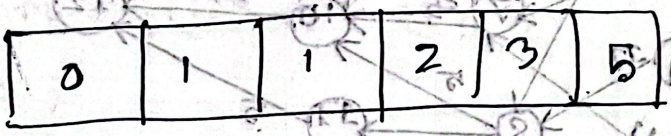


Step-2:



fib series এর প্রথম দুইটি সংখ্যা Initialize.

Step-3:



$T(n) = O(n)$

Tabulation Method: (follow bottom-up approach)

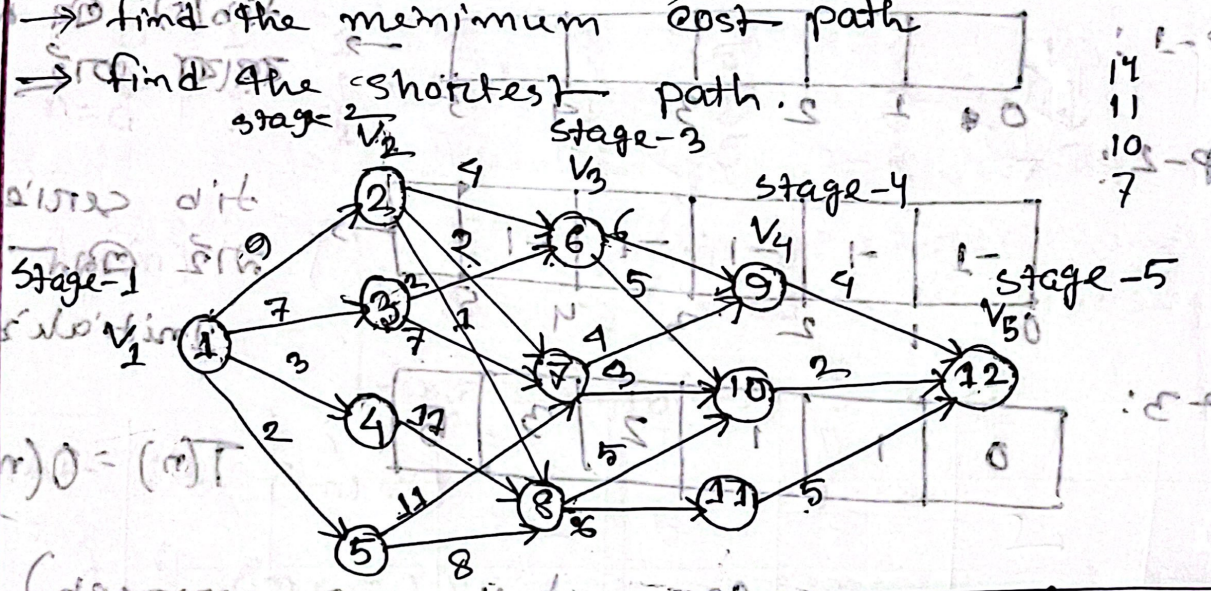
$$fib(n) = \begin{cases} 0 & \\ 1 & \\ fib(n-2) + fib(n-1) & \end{cases}$$

```
int fib(int n)
{
    F[0] = 0;  F[1] = 1;
    for (int i = 2 ; i <= n ; i++)
    {
        F[i] = F[i-2] + F[i-1]
    }
    return F(n);
}
```

* Multistage graph

→ find the minimum cost path

→ find the shortest path.



Vertex	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2 or 3	7	6	8	7	10	10	10	12	12	12	12

formula : $Cost(i,j) = \min\{(j,l) + Cost(i+1,l)\}$

- i → stage
- j → vertex
- (j,l) → j থেকে যাবে
- l → যেসব vertex এ যাবে।

$Cost(5,12) = 0$

$Cost(4,9) = \min\{c(9,12) + Cost(5,12)\}$

$= \min(4 + 0)$

$= 4$

$Cost(4,10) = 2$

$Cost(4,11) = 5$

$Cost(3,6) = \min\{c(6,9) + Cost(4,9), c(6,10) + Cost(4,10)\}$

$= \min\{(6+4), (5+2)\}$

$= \min(10, 7)$

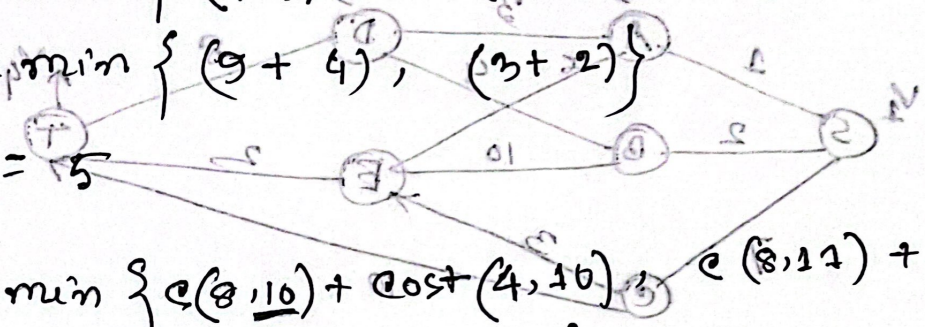
$= 7$

l এর মান 10 এর জন্য min cost পাওয়া গেছে cost = 7, d = 10

$Cost(3,7) = \min\{c(7,9) + Cost(4,9),\ c(7,10) + Cost(4,10),$

$= \min\{(9+4),\ (3+2)\}$

$= 5$



$Cost(3,8) = \min\{c(8,10) + Cost(4,10),\ c(8,11) + Cost(4,11)\}$

$= \min\{(5+2),\ (6+5)\}$

$Cost(2,2) = \min\{c(2,6) + Cost(3,6),\ c(2,7) + Cost(3,7),$

$c(2,8) + Cost(3,8)\}$

$= \min\{(4+7),\ (2+5),\ (1+7)\}$

$= \min(11, 7, 8)$

$= 7$

$Cost(2,3) = \min\{c(3,6) + Cost(3,6),\ c(3,7) + Cost(3,7)\}$

$= \min\{(2+7),\ (7+5)\}$

$= 9$

$Cost(2,4) = \min\{c(4,8) + Cost(3,8)\}$

$= 11 + 7$

$= 18$

Path: Start - 2

$d(1,1) = 2$

$d(2,2) = 7$

$d(3,7) = 10$

$d(4,10) = 12$

Start - 3

$d(1,1) = 3$

$d(2,3) = 6$

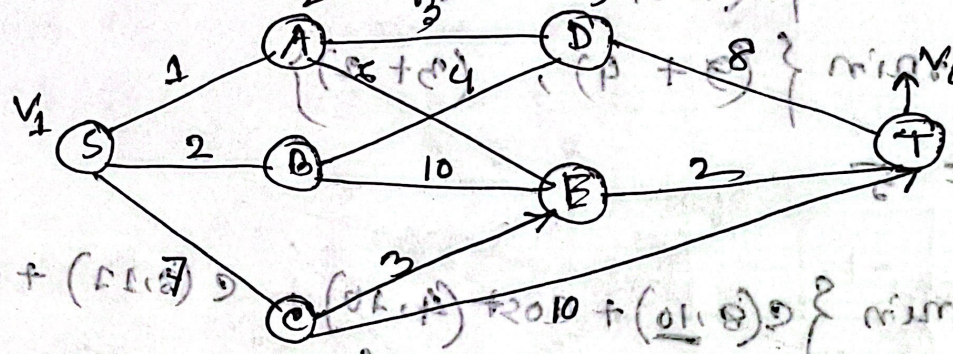
$d(3,6) = 10$

$d(4,10) = 12$

$1 \to 2 \to 7 \to 10 \to 12$

$1 \to 3 \to 6 \to 10 \to 12$

min cost = 16



Vertex	S	A	B	C	D	E	T
Cost	0	8	12	5	8	2	0
P		(S,8)	(A,12)	(S,5)	(A,8)	(C,2)	(E,0)