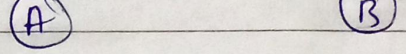
Unit-5 Graph

Types of Graph

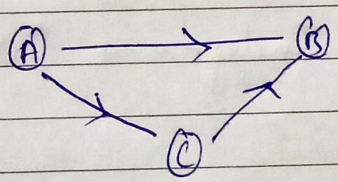
1) NULL Graph : Graph with no edges



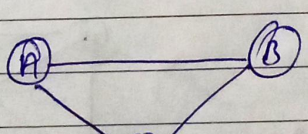
2) Empty Graph: Graph with no vertices & edges

3) Singleton Graph : Graph with only one vertex

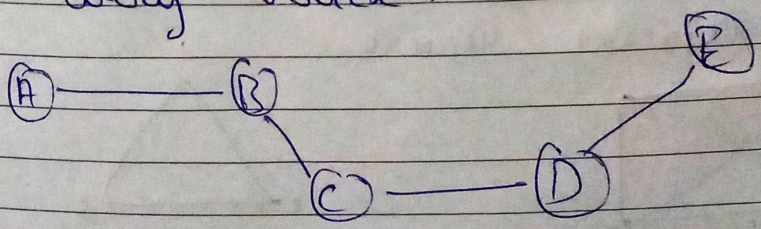
4) Directed Graph : Graph with directions



5) Un-directed Graph : Graph with no direction

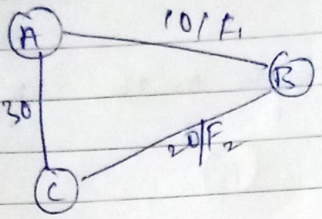


6) Connected Graph : A Graph which consist atleast one path that connects every vertex.



Labled

7) Weighted Graph: A Graph whose edges have some weight



(Undirected)

8) Degree: No. of edges connected to a particular vertex

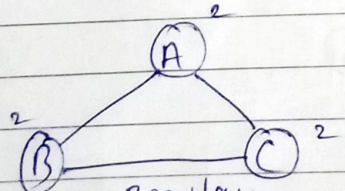
For directed

Indegree: No. of edges coming towards the node

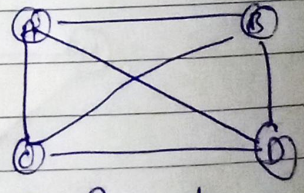
Outdegree: No. of edges moving away from node

=) Node degree = Indegree + Outdegree

9) Regular Graph: A Graph in which every vertex's degree is same.

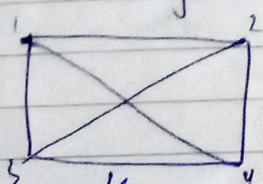


2-Regular

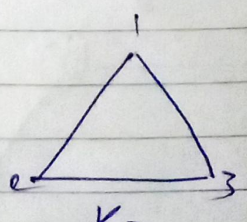


3-Regular

10) Complete Graph $^{(K)}$: A Graph in which all the vertex are connected to every adjacent vertex



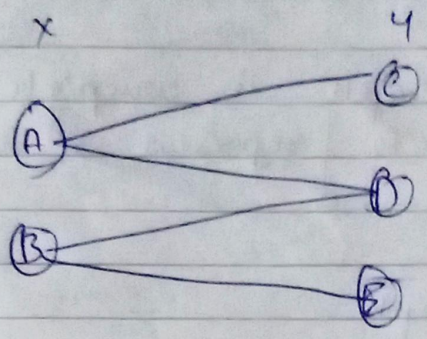
K_4



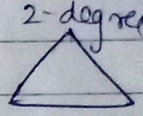
K_3

⇒ No. of edges = $\frac{n(n-1)}{2}$, where n = no. of vertex

11) Bi-Partite

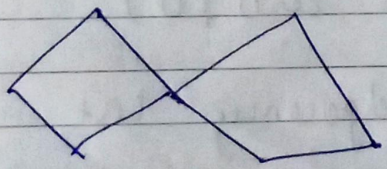


12) Cycle Graph : A Graph ~~is called~~ is called cycle Graph if:



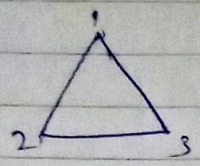
i) $n \geq 3$
ii) Each vertex have ~~degree~~ degree = 2
iii) Should have one cycle

13) Cyclic Graph: A Graph which has atleast one cycle. (no restriction on degree)



14) Acyclic Graph : A Graph with no cycle.

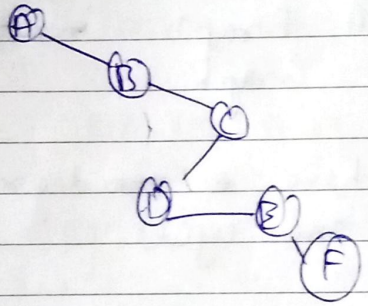
15) ~~Adjacency~~ Adjacency Matrix



	1	2	3
1	0	1	1
2	1	0	1
3	1	1	0

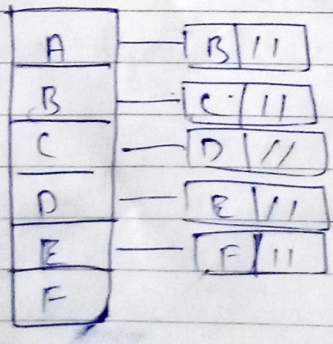
→ If the given graph is complete or Regular then best way to represent is Adjency Matrix

For sparsh Graph



→ We cannot use adjacency because max entries will be zero (0)

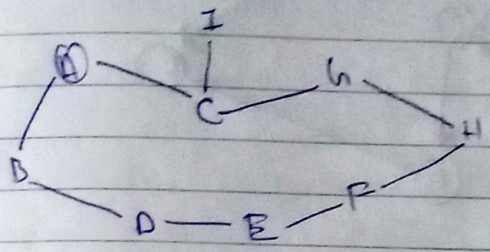
→ So, we used adjacency list



* Representation of Graph

1) Breadth first search

→ In BFS, we do searching label by label
→ It uses Queue



BFS : A, B, C, D, I, G, E, H, F
Q: A, B, C, D, I, G, E, H, F

→ Time complexity : O(V+E)

2) Depth First Search

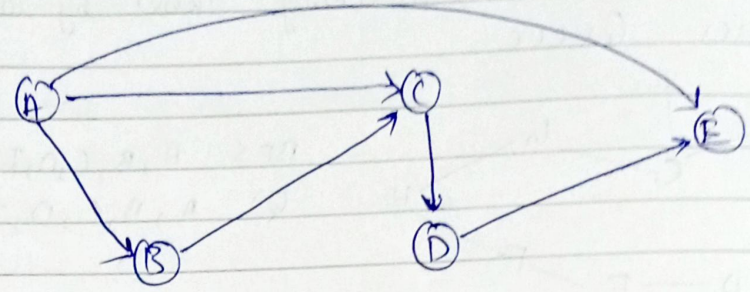
→ It follows the principle of Back-Tracking,
Back-Tracking uses Recursion,
And Recursion uses Stack

→ It uses stack data structure

→ Time Complexity : O(V+E)

✱ Topological Sorting / Ordering

→ Topological sorting can be applicable on directed acyclic graph (DAG)

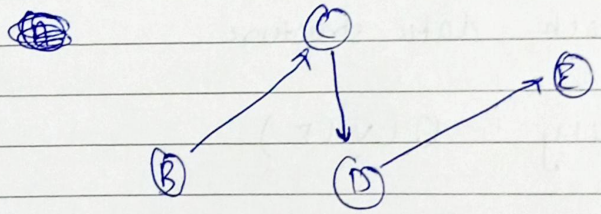


Step-1: Find out the indegree of each and every vertex of the graph

A: 0 , B: 1 , C: 2 , D: 1 , E: 2

Step-2: Remove the vertex whose indegree is minimum, along with its transitions

Topological order: A

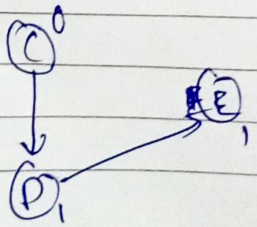


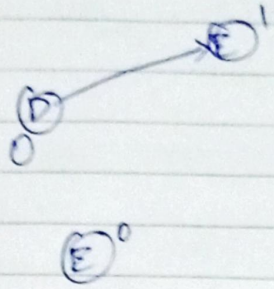
Now, again find indegree

B: 0, C: 1, D: 1, E: 1

Topological order: A, B, C

→



→ 

→ Topological order : A, B, C, D, E

→ When we encounter same indegree, its our own choice (prefer alphabetical order)

Back - Tracking

→ Back - Tracking is generally used in tactical problems

→ Sometimes it is Dynamic programming

→ Disadvantage : Time complexity

→ It is very slow.

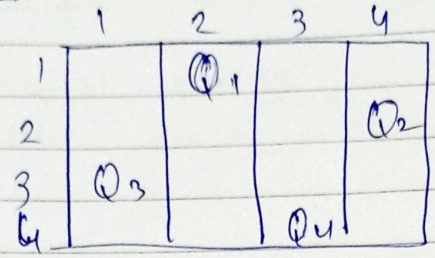
→ We cannot use backtracking in case of strategy problems.

Applications

1) N-Queen problem:

n=1, n=2, n=3 not possible

4×4



→

Q_1	Q_2	Q_3	Q_4
2	4	1	3

Total possibilities : $^{16}C_4$

Another | 3 | 1 | 4 | 2 |

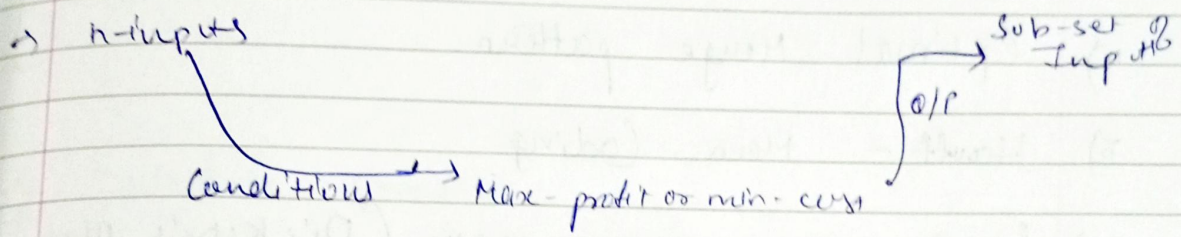
Similarly solve 8×8

8×8

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
4	6	8	2	7	1	3	5

Ch- Greedy Techniques

→ In Greed Technique, we have n-inputs ~~our~~ our object is to find the ~~subset~~ subset of given input, which will satisfy our requirement (max-profit or min-cost)



→ Four parts :-

1) Problem defn :- Understanding the problem clearly

2) Finding soln space : All possible soln for the given n-inputs

3) Feasible soln space : It is one of the soln in the soln space which will satisfy our condition

Optimal soln : It is one of feasible soln which will give max-profit or min-cost

→ Applications

1) Greedy Knapsack / ~~Optional~~ Fractional knapsack
2) Job - sequence with deadline
3) Min cost spanning Tree
4) Optimal Merge pattern
5) Hauff - Man Coding
6) Single source ~~shortest~~ person (Dijkstra's Algo)

---

1) Greedy Knapsack

Q.) There are 3 objects & capacity of knapsack is 20

n = 3, M = 20

obj	obj1	obj2	obj 3
profit	25	24	15
Weight	18	15	10

$$\text{Profit} = \sum_{i=1}^{n} x_i * P_i$$, where x_i = Ratio

Step-1 Calculate the Pi/wi ratio for all the objects

Step-2 Arrange the data in descending order

obj 2	obj 3	obj 1
1.6	1.5	1.3

Step-3 $\left(1 \quad \frac{5}{10} \quad 0\right)$

Step-4 Calculate Profit.

⇒ Greedy knapsack will give optimal solution everytime over the priority of profit & weight.

(1) $n = 7$, $M = 15$

$(P_1 \text{ to } P_7) = \{10, 5, 15, 7, 6, 18, 3\}$

$(w_1 \text{ to } w_7) = \{2, 3, 5, 7, 1, 4, 1\}$

⇒

obj 1	obj 2	obj 3	obj 4	obj 5	obj 6	obj 7
10/2	5/3	15/5	7/7	6/1	18/4	3/1
= 5	= 1.6	= 3	= 1	= 6	= 4.5	= 3

~~Sort obj 5 obj 6 obj~~

~~5~~ 4.5

Desc → ~~Objs~~

6 5 4.5 3 3 1.6 1

$(1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 2/3 \quad 0)$

$\text{Profit} = (1\times6 + 1\times10 + 1\times18 + 1\times15 + 1\times3 + \frac{2}{3}\times5)$

$= 6 + 10 + 18 + 15 + 3 + \frac{10}{3}$

≈ 55.33

Proper way

Obj	Profit	Weight	P/w
1	10	2	5
2	5	3	1.6
3	15	5	3
4	7	7	1
5	16	1	6
6	18	4	4.5
7	3	1	3

⇒ $X_i = \begin{cases} \text{obj 5} & \text{obj 1} & \text{obj 6} & \text{obj 3} & \text{obj 7} & \text{obj 2} & \text{obj 4} \\ 1 & 1 & 1 & 1 & 1 & 2/3 & 0 \end{cases}$

(6) (5) (4.5) (3.3) (3) (1.6) (1)

⇒ $\text{Profit} = \sum_{n=1}^{\infty} x_i * P_i$

Algo

Step-1 Calculate Profit / Weight Ratio for each object.

```
for (i=1 to n)
    a[i] = Pi/Wi  ⇒ O(n)
```

Step-2 Arrange objects, according to Descending order of Pi/Wi Ratio

↳ ⇒ $O(n \log n)$

Step-3 Take one by one objects until (M==0)

```
while (M ≠ 0)
{
    M = M - Wi * xi   ⇒ O(n)
    P = P + Pi * xi
}
```

⇒ Time Complexity of Greedy knapsack

↳ $O(n) + O(n \log n) + O(n)$

$= 2O(n) + O(n \log n)$

$= O(n \log n)$

2) Job - Sequence / Job Ordering with Deadline

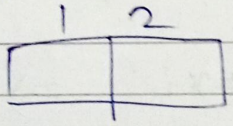
- Single Processor is available
- Interleaving is not allowed
- One Unit of Time for each job execution
- Arrival time of each job is same

Q) n = 4

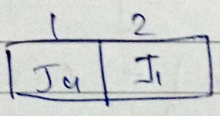
Job	J_1	J_2	J_3	J_4
Profit	100	15	10	27
Deadline	2	1	2	1

=>

Step-1 Find the max deadline value & create a array of that size



Step-2 Now start from the right hand side of the array & write according to the max profit



Total Profit = 127

Sequence = J_4, J_1

(Q) $n = 5$

$(P_1 \text{ to } P_5) = \{10, 20, 15, 5, 80\}$

$(D_1 \text{ to } D_5) = \{3, 3, 3, 4, 4\}$

What is the maximum profit ?

What is the sequence of Job for max profit ?

How many jobs are left out / not executed ?

⇒

Job	Profit	Deadline
J_1	10	3
J_2	20	3
J_3	15	3
J_4	5	4
J_5	80	4

⇒

1	2	3	4
J_1	J_3	J_2	J_5

⇒ Max profit = ~~53~~ 125

⇒ Sequence = J_1, J_3, J_2, J_5

⇒ Not executed = J_4

Algo

Step-1 Arrange the Jobs according to descending order of there profits
↳ ⇒ $O(n \log n)$

Step-2 Take the array of size → max-deadline
↳ ⇒ $O(n)$

Step-3 Start from the right end of the array & search for slot d_i, using linear search to find the job contain >= d_i
Continuous until array slots are completed
↳ ⇒ $O(n^2)$

Time Complexity ⇒ $O(n \log n) + O(n) + O(n^2)$
$= O(n^2)$

✱ Spanning Tree

→ ~~A Spanning Tr.~~

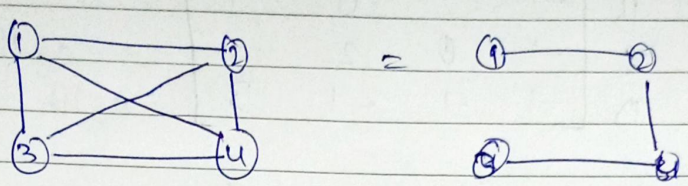
→ A Connected Sub-Graph, H of a Graph G(V, E) is said spanning Tree, if and only if:

- It should have (V) Vertices
- It should contain (V-1) Edges

→ No. of spanning Tree possibilities (For Complete Graph)

$= n^{n-2}$

Example



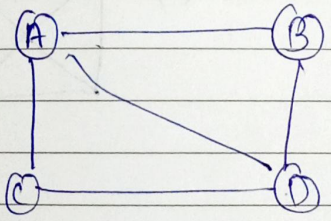
$= n^{n-2} = 4^{4-2} = 16$ possible Spanning Tree

For Non-Complete Graph

1) Brute-Force Method

2) Kirchoff's Theorem

Step-1 → Find the adjacency matrix of the given graph



→

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

Step-2 Replace all the diagonal elements in adjacency matrix which are zero with there order

	A	B	C	D
A	3	1	1	1
B	1	2	1	1
C	1	1	2	1
D	1	1	1	3

non-diagonal

Step-3 Replace all the values which are 1 with (-1)

	A	B	C	D
A	3	-1	-1	-1
B	-1	2	0	-1
C	-1	0	2	-1
D	-1	-1	-1	3

Step-4 Find the (n-1) co-factor of matrix

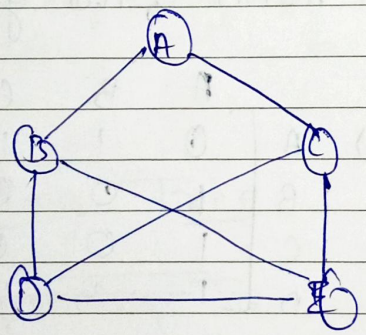
$$\begin{vmatrix} 3 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{vmatrix}$$

$= 3(4-0) + 1(-2-0) - 1(0+2)$

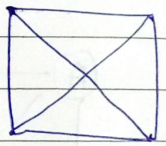
$= 12 - 2 - 2$

$= 8$

Q)



Q)



Minimum Cost Spanning Tree

→ Prism's Algo
→ kruskal's Algo

1) Prism's Algo