Module 3. - Greedy Algorithms and Dynamic Programming

* The Greedy Approach.

→ In the greedy approach, the solution to the problem is built incrementally by choosing the locally optimal choice at each step.

→ The optimal choice is made based on the information available at that moment.

* Kruskal's algorithm for Minimum Spanning Tree.

→ Step 1 - Construct a minimum heap of e-edges.

→ Step 2 - Take the edges one-by-one starting from the least weight. (cycle/loop is not created)

Best case → (n-1) edges
Worst case → e edges

* Spanning Tree.

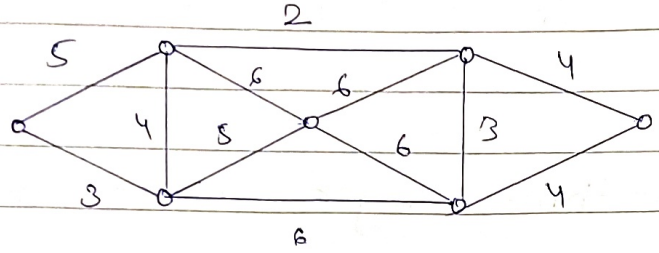
→ A spanning tree (S) should be a sub-graph of the graph G(V, E)

① S should contain all the vertices of G.

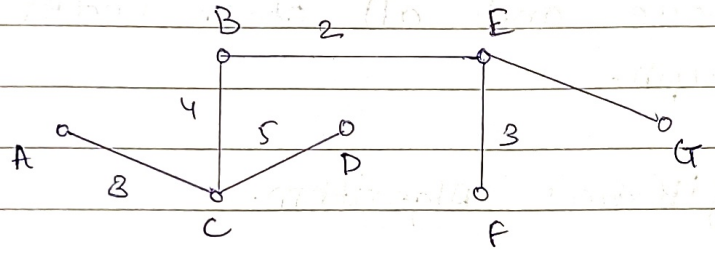
② S should contain (|V|-1) edges.

③ S should have no cycles or loops.

Example ①



→ Step ① → Arrange all edges to be in ascending order.



Consider minimum weight first

① → 2 (BE)

② → 3 (AC, EF)

③ → 4 (BC, FG, GE) {EFG form a loop so one out of FG and EG is not drawn}

④ → ⑤ 5 (CD, AB) { ABC form a cycle so AB will not be drawn }

⑤ ∴ Total weight = 3+4+5+2+3+4 = 21

So edges = (n-1) = 7-1 = 6

* Djikstra's Algorithm. Single Source Shortest Path.

→ Djikstra's Algorithm is a popular algorithm used for finding the shortest path between a source node and all other nodes in a weighted graph.

Steps in Djikstra's Algorithm.

① Initialize all nodes with a distance value of ∞ and the source node to 0.

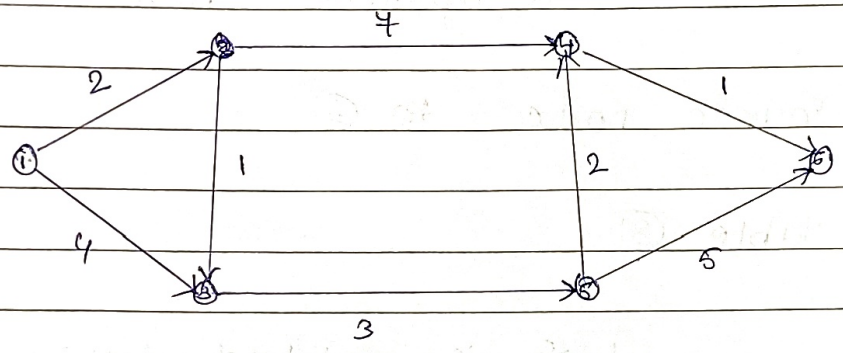
② Create a set of unvisited nodes and all all nodes to the graph.

③ While there are unvisited nodes, select the node with the smallest distance value and mark it visited

④ For each neighbour of the selected node that is still unvisited, calculate the distance to that neighbour by adding the weight of the edge b/w two nodes to the distance value of selected nodes.

⑤ Repeat this for all nodes

Example ①



→ Let the source node be ①

So Table 1

1	0
2	2
3	4
4	∞
5	∞
6	∞

→ Change the source node to 2.

So Table 2

Use formula for Relaxation

2	0
3	∞ 1
4	∞ 7
5	∞
6	∞

if (d[u] + c(u,v) < d[v])
d[v] = d[u] + c(u,v)

As (2+1) ≤ 4 & 2+7 < ∞

∴ 1 → 3 is updated from 4 to 3
1 → 4 is updated from ∞ to 9.

→ Move source node to ③

So Table ③

3	
4	∞
5	3
6	∞

1→5 is updated from ∞ to 6.

As 2+1+3 < ∞

2 → 5 is updated from ∞ to 4

→ Move source node to 5.

So Table 5

5	0
1	6
2	4
3	3
4	2
6	5

2→6 updated from ∞ to 9.
2→4 updated from 7 to 6
(1→3→5)
1→4 updated from 9 to 8
1→6 updated from ∞ to 11.
3→6 updated from ∞ to 8.
3→4 updated from ∞ to 5

Move selected node to (4)

Table 4

4		
1	8	$1 \to 6$ updated from $11 \to 9$
2	6	$2 \to 6$ updated from 9 to 7
3	8	$3 \to 6$ updated from 8 to 6.
5	2	$5 \to 6$ updated from 5 to 3
6	1	

Move to node 6.

Full scanned.

So minimum distance of 1 to 2, 3, 4, 5, 6.

Final Table

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

* Knapsack Problem.

→ The Knapsack problem is a classic optimization problem which involves packing a knapsack with a set of items, each with a weight and a value. in such a way that we can maximise the value while staying within the weight capacity.

There are two types of knapsack problems

① 0/1 Knapsack - Each item can either be included or excluded in the knapsack. We cannot include the fraction of an item.

The Recurrence Relation is:-

→ If $i=0$ and $j=0$ then max value is 0.

→ If $w_i > j$ then item cannot be included in the knapsack, the max value is reached by packing the first $i-1$ items.

→ If $w_i <= j$ - Then item can be included or excluded.

Here i is the items
j is the knapsack capacity.

② Fractional Knapsack - In this problem, fractions of weights and profits can be taken.

① 0/1 Knapsack Problem.

Example

$m = 8$ $\quad P = \{1, 2, 5, 6\}$

$n = 4$ $\quad W = \{2, 3, 4, 6\}$

Method ① Table form.

Pi	Wi	0	1	2	3	4	5	6	7	8
1	2	0	0	1	1	1	1	1	1	1
2	3	0	0	1	2	2	3	3	3	3
5	4	0	0	1	2	5	5	6	7	7
6	5	0	0	1	2	5	6	6	7	8

Formula.

$$V[i, w] = \max \{ V[i-1, w], V[i-1, w - W[i]] + P[i] \}$$

Method ② - Sets Method.

→ Form sets in form (P, W)

$S^0 = \{(0,0)\}$

$S^0_1 = \{(1,2)\}$

$\therefore S^1 = S^0 \cup S^0_1 = \{(0,0), (1,2)\}$

$S^1_1 = \{(2,3), (3,5)\}$

$S^2 = \{(0,0), (1,2), (2,3), (3,5)\}$

$S^2_1 = \{(5,4), (6,6), (7,7), (8,9)\}$ (Weight can't be 9)

$S^3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7)\}$

$S^3_1 = \{(6,5), (7,7), (8,8), (11,9), (12,11), (13,12)\}$

$S^4 = \{(0,0), (1,2), (2,3), (5,4), (6,6), (7,7), (6,5), (7,7), (8,8)\}$

(8,8) ∉ S_4
but (8,8) ∉ S_3 → (1)

(8-6, 8-5) = (2,3)

(2,3) ∉ S_3
but (2,3) ∉ S_2 — (2) 0.

(2,3) ∉ S_2
but (2,3) ∉ S_1 — (1)

(2-2), (3-3) = (0,0)

---

* Job sequencing. (with deadlines)

→ Job sequencing is an optimisation problem that involves scheduling a set of jobs on a single machine with a fixed deadline for each job.

→ Algorithm for Job sequencing

(1) Sort all jobs in ~~non~~ decreasing order based on profit.

(2) Initialize a schedule, all slots empty.

a) Starting from a deadline find the first available slot.

b) If slot exists, assign the job to that slot.

c) If not, skip the job.

④ Compute total profit

Example

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profits	35	30	25	20	15	12	5
Deadlines	3	4	4	2	3	1	

Job	slots assigned	profit
J_1	[2,3]	35
J_2	[2,3] [3,4]	35+30
J_3	[2,3] [3,4] [1,2]	35+30+25
J_4	[2,3] [3,4] [1,2]	35+30+25
J_5	[2,3] [3,4] [1,2]	35+30+25
J_6	[2,3] [3,4] [1,2] [0,1]	35+30+25+
J_7	[2,3] [3,4] [1,2] [0,1]	35+30+25+

100.