DAG, Topological sort, SCC

Directed Acyclic graph: It is a graph with no directed cycle

meaning: A cycle is a path that starts and ends at the same vertex, with no repeated edges / vertices except the starting / Ending vertex

Example of valid cycle

Here we have a cycle & it has 3 distict nodes

Invalid cycles:

1. Repeated vertex in the middle:

   A → B → A → C → A

2. Uses the same edge twice

   A → B → A → B → A

3. Less than 3 vertex:

4. Path doesn't come back to the starting vertex

A valid cycle must have at least 3 distict vertices

Topological sort:

Steps:

1) Find indegree of all vertices.

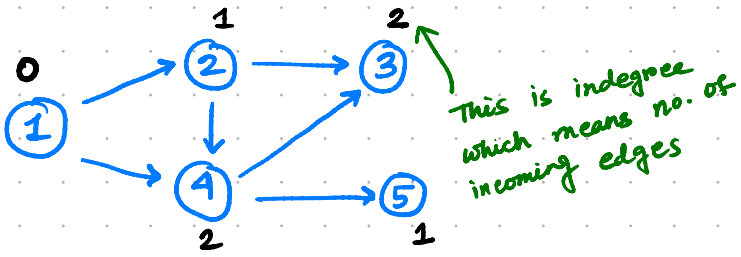
2) Delete vertex with indegree = 0 (also delete all the associated edges)

3) Repeat this process.

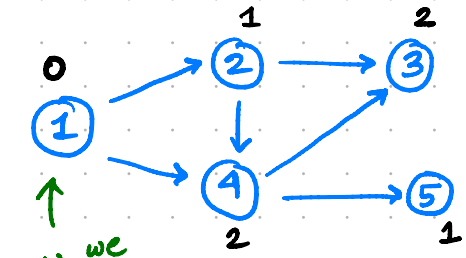
⊛ Time complexity → O(V+E)

Simulation

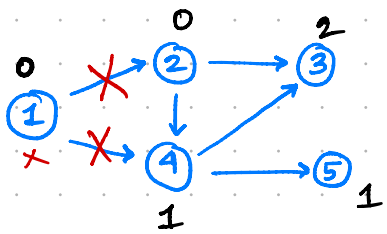
step 1



Step 2



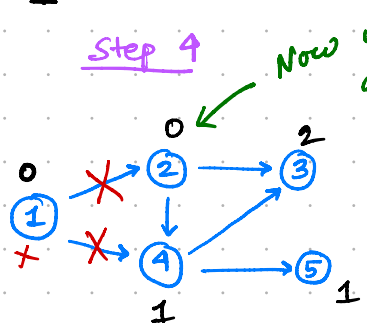
Step 3



Topological order

1

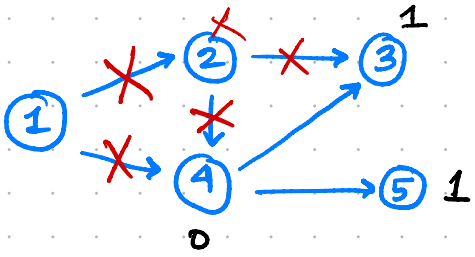
Step 4



Topological order

1

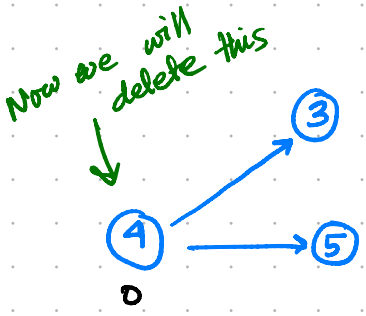
Step 5



Topological order

1 2

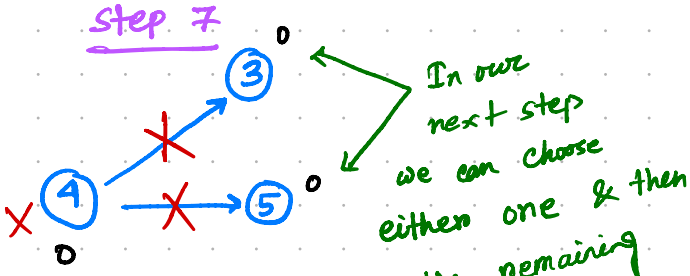
Step 6



Topological order

1 2

Step 7



Topological order

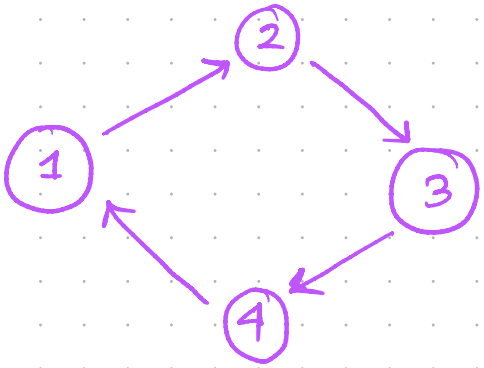
1 2 4

Step 8

Topological order → 1 2 4 3 5

SSC (Kosaraju):

* If we can reach every vertex to every vertex in a graph then it is called SCC. Example:



- A single node is always a SCC.
- All the vertices in an undirected graph are always SCC.

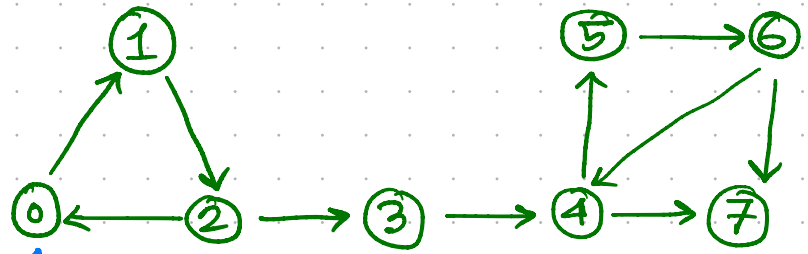
Kosaraju

* SCC will remain SCC even if we transpose the graph.

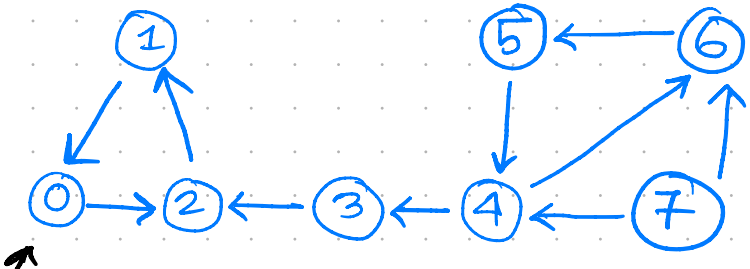
Steps

1. Perform DFS traversal of the graph. Push node to the stack before returning.
2. Find the transpose of the graph by reversing the edges.
3. Pop nodes one by one from the stack and again run DFS on the modified graph. Each successful DFS gives one SCC.

Time complexity → O(V+E)



Transpose



Stack (while returning back we will fill this stack)

7	~~0~~	~~5~~	~~4~~	~~3~~	~~2~~	~~1~~	~~0~~

We will pop one by one in this step

0, 2, 1 → 1st scc

3 → 2nd scc

4, 6, 5 → 3rd scc

7 → 4th scc

Stack for keeping a track of the path

0	1	2	3	4	5	6	7

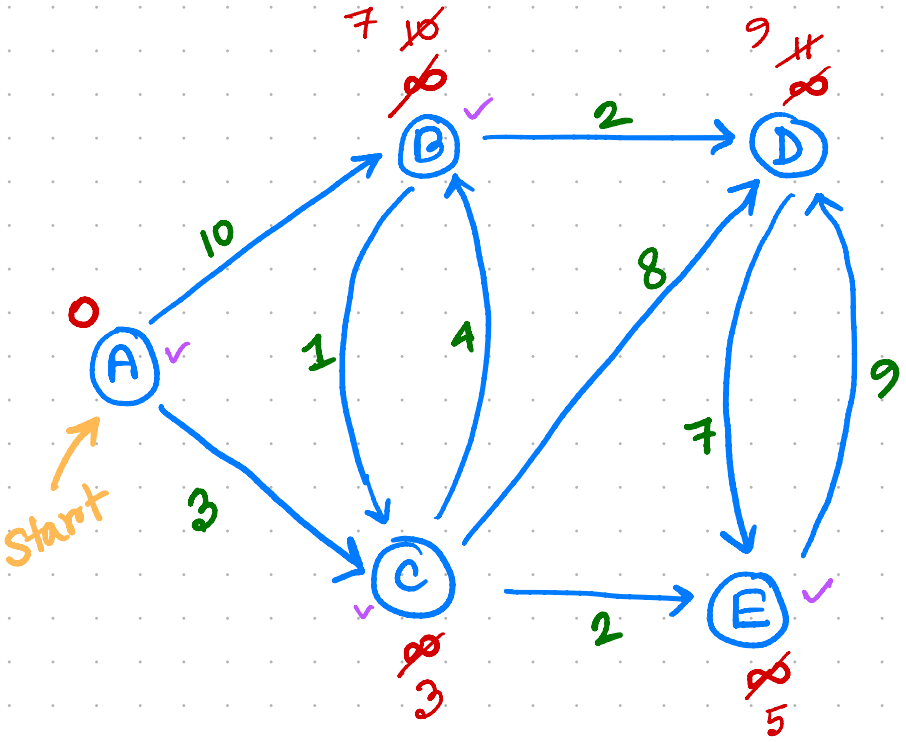
visited

T	T	T	T	T	T	T	T
~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~
0	1	2	3	4	5	6	7

index/nodes

T	T	T	T	T	T	T	T
~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~
0	1	2	3	4	5	6	7

SSSP, APSP, Dijkstra



Relaxation:

if ($w[u] + d[u,v] < w[v]$)

$w[v] = w[u] + d[u,v]$

Time complexity

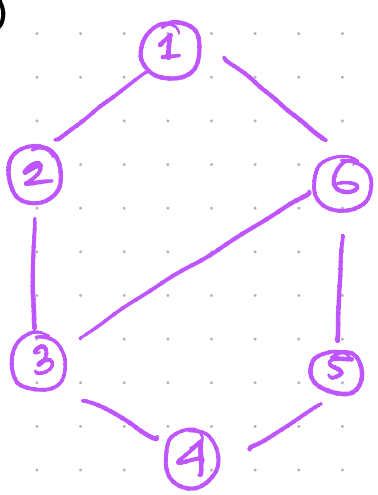
$O(n^2)$

with heap

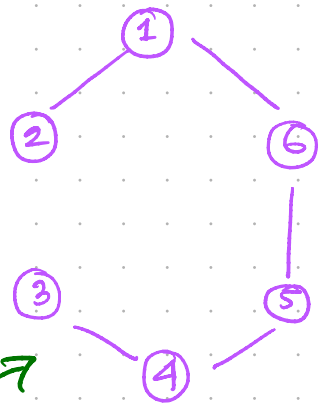
$O((v+E)\log V$

Selected vertex	A	B	C	D	E
A	0	∞	∞	∞	∞
C		10	3	∞	∞
E		7		11	5
B		7		11	
D				9	

MST (Kruskal), DSU



spanning tree is a subgraph of a graph having all vertices but (n-1) edges



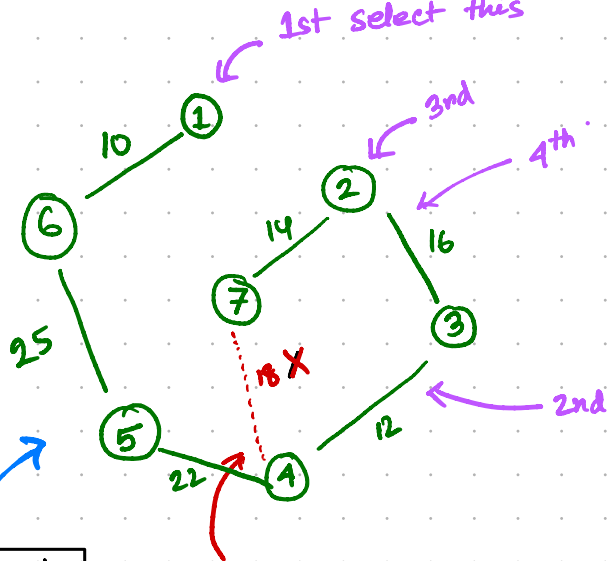
There will be no cycle because total edges are $|V|-1$

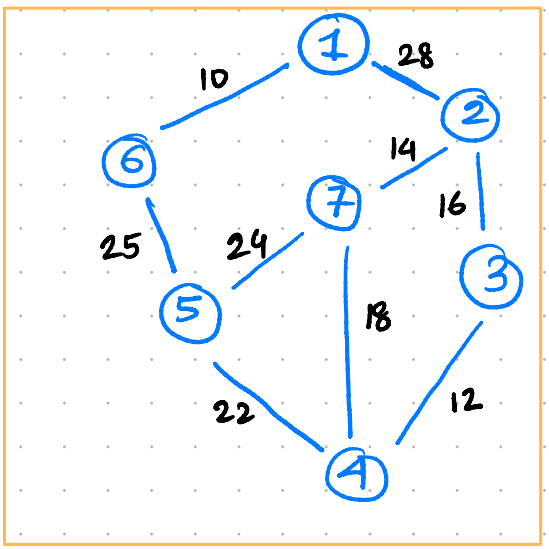
* How many spanning tree can I have from a graph?

Ans: $^{|E|}C_{|V|-1}$ − no. of cycles

Kruskal

① Always select the minimum cost edge as long as it doesn't create cycle.





Total cost is 99

This is the 5th smallest edge but if I select This then it will form a cycle. So, we have ignore it to avoid making cycle

Time complexity : $O(\underbrace{|V| * |E|}_{n^2})$; if min heap is used then $O(n \log n)$

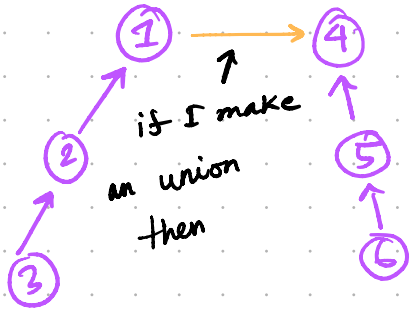
✱ Spanning tree is not possible in non-connected graph.

DSU

✱ When there is no common elements in two sets then it is called disjoint sets. Ex:

$$\{1,2,3,4\} \cap \{5,6,7\} = \emptyset$$

田 $S1 = \{1, 2, 3\}$, $S2 = \{4, 5, 6\}$



Node : 1 2 3 4 5 6

parent : -1 1 2 -1 4 5

4

✱ How to find vertex V_1 & V_2 belong to the same set?

Ans:

Find absolute root of V_1 , $AR(V_1)$

Find absolute root of V_2 , $AR(V_2)$

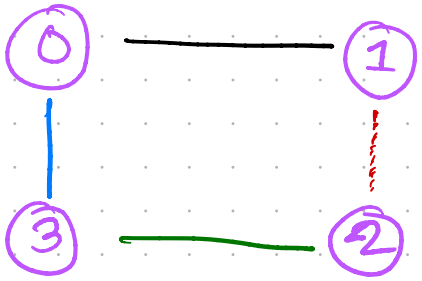
if $AR(V_1) == AR(V_2)$:

then they belong to the same set.

else :

NO

✱ Disjoint sets can't detect cycle in a directed graph. It detects in an undirected graph.

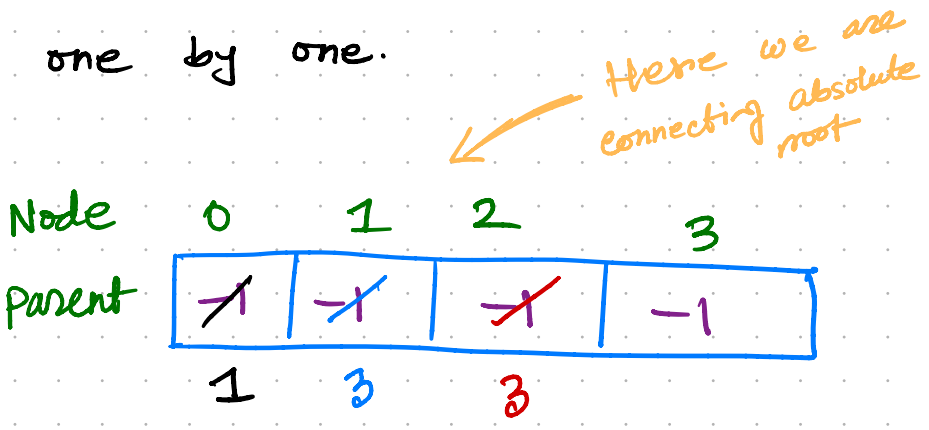


Connect these edges one by one.

(0, 1)

(0, 3)

(2, 3)



(1,2) ← Now if I add this edge then it will create cycle

let's see how to find if AR(1) is 3 & AR(2) is 3. If absolute roots are same then cycle detected

Absolute root

If AR(V_1) == AR(V_2) :
cycle detected

✳ Time complexity : O(E * V)

Huffman Encoding

- Message → BCCABBDDAECCBBAEDDCC

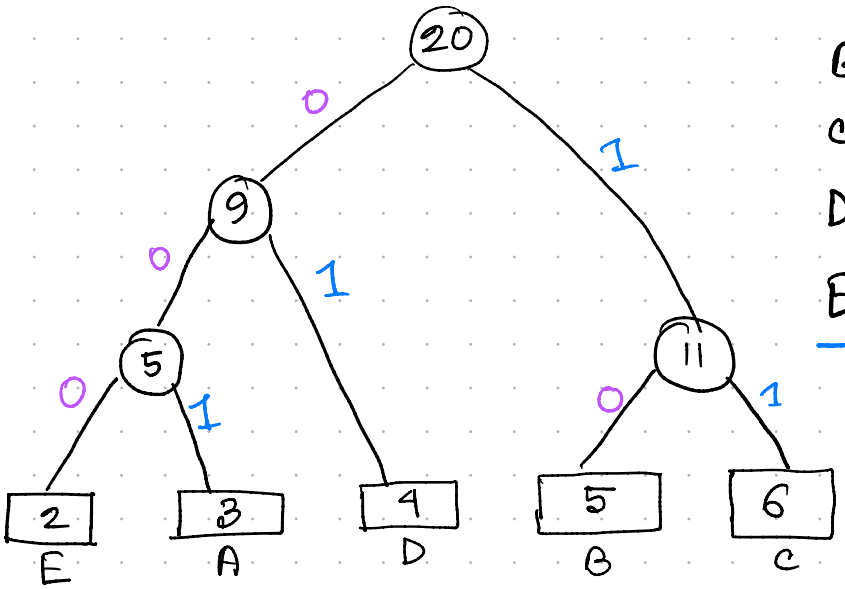
Characters	count	frequency	code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20		

ASCII is 8 bits
So total bits = 5×8 = 40

total bits = 5×3 = 15

Total message size here:
size of the table + message
(40+15) + (20×3)
= 55+60
= 115 bits

Variable sized code



A → 001
B → 10
C → 11
D → 01
E → 000

Count	Size
3	3×3 = 9
5	5×2 = 10
6	6×2 = 12
4	4×2 = 8
2	2×3 = 6
	45 bits

5×8 = 40 bits

3+2+2+2+3 = 12 bits

This is the size of the table (used for decryption)

Total message size here:
(40 + 12 + 45) bits
= 97 bits
actual message

Knapsack problem

Fractional knapsack

* This is a greedy method.

Object (O) :	1	2	3	4	5	6	7
Profits (P) :	10	5	15	7	6	18	3
weights (w) :	2	3	5	7	1	4	1
P/w :	5	1.3	3	1	6	4.5	3

we will choose the one having highest P/w first.

x :	1	2/3	1		1	1	1
	x_1	x_2	x_3	x_4	x_5	x_6	x_7

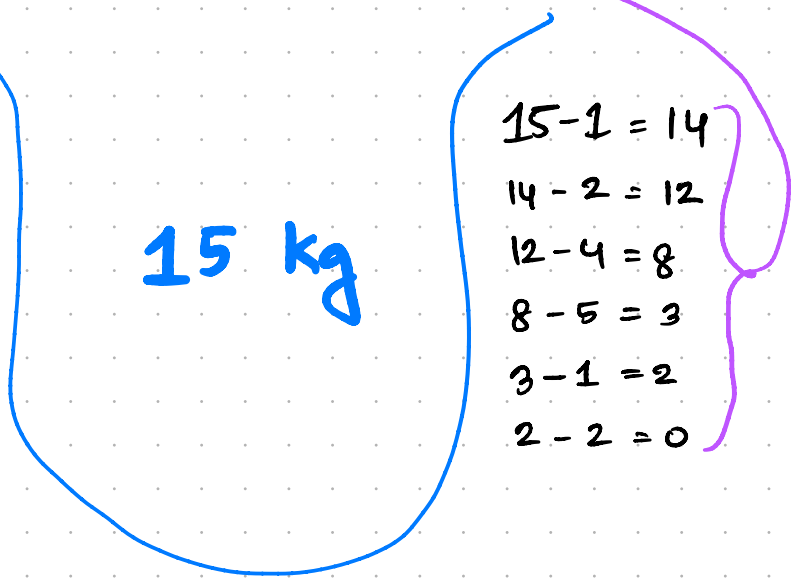
$0 \leq x \leq 1$

Total profit = $1 \times 10 + 2/3 \times 5 + 1 \times 15 + 1 \times 6 + 1 \times 18 + 1 \times 3 = 54.6$

$\sum x_i P_i$

Time Complexity

Sorting	$O(n \log n)$
Selection	$O(n)$
Total	$O(n \log n)$



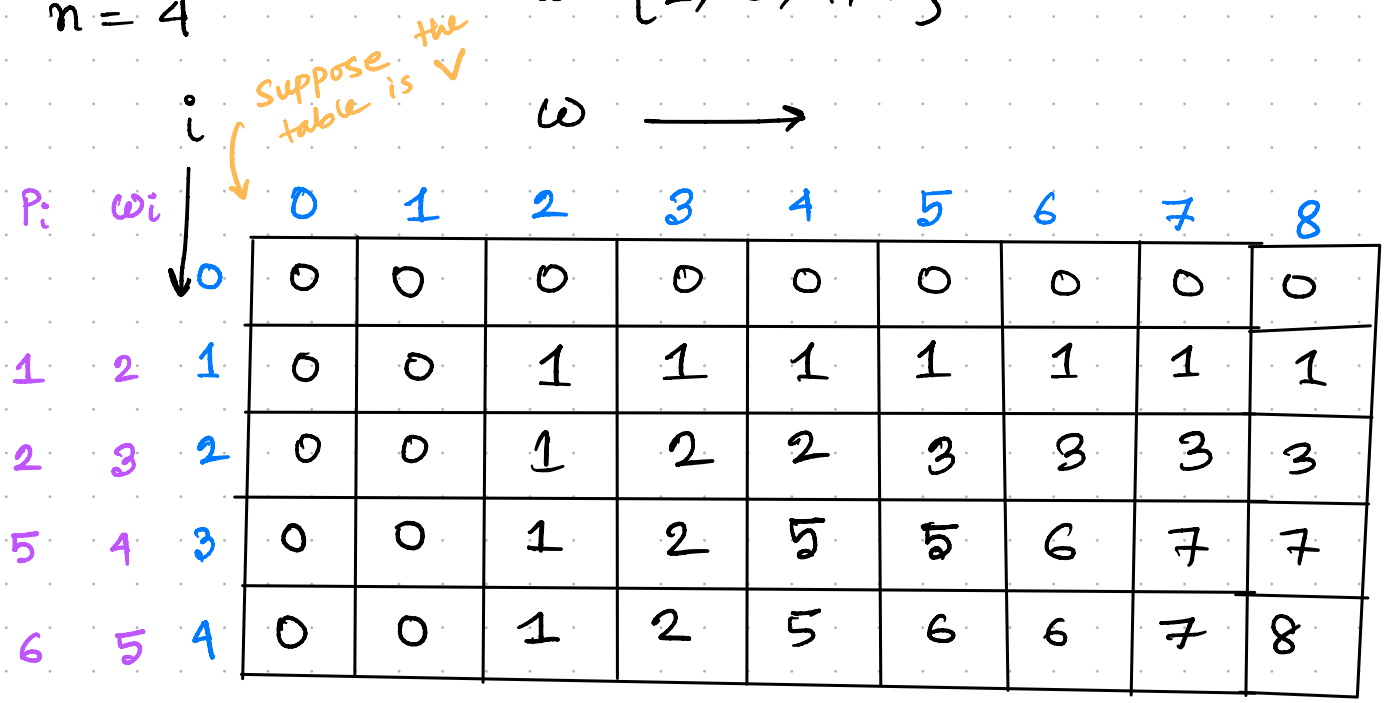
0/1 Knapsack

Capacity = 8

n = 4

P = {1, 2, 5, 6}

ω = {2, 3, 4, 5}



P_i	ω_i	i	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

Formula for filling cells

$$V[i, \omega] = \max\left(V[i-1, \omega], V[i-1, \omega-\omega_i] + P_i\right)$$

$8 - 6 = 2$

x_1	x_2	x_3	x_4
0	1	0	1

Time complexity → $O(n \times W)$

LCS - Longest common Subsequence

String 1 : a b c d e f g h

String 2 : c d f h

cdfh — it is a substring

dfh, fh, h — These are also substring but the longest is 'cdfh'

String 1 : a b c d e f g h

String 2 : c b f h

lines shouldn't intersect.

String 1 : a b c d e f g h

String 2 : c b f h

bfh is the LCS

DP

Str 1 : programming

Str 2 : gaming

Algorithm

if (A[i] == B[j]):

LCS[i,j] = 1 + LCS[i-1, j-1]

else:

LCS[i,j] = max(LCS[i-1, j], LCS[i, j-1])

		P	P	O	G	R	A	M	M	I	N	G
	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
G 1	0	0	0	0	1	1	1	1	1	1	1	1
A 2	0	0	0	0	1	1	2	2	2	2	2	2
M 3	0	0	0	0	1	1	2	3	3	3	3	3
I 4	0	0	0	0	1	1	2	3	3	4	4	4
N 5	0	0	0	0	1	1	2	3	3	4	5	5
G 6	0	0	0	0	1	1	2	3	3	4	5	6

g a m i n g

∴ LCS = g a m i n g

Time complexity

O(m x n)