Q $A = qpqrr$
$B = pqprqrp$

$x \rightarrow$ length of subsequence
$y \rightarrow$ no. subsequence exist
find $x + 10y$

		A	q	p	q	r	r
	B	0	0	0	0	0	0
p		0	0	1	1	1	1
q		0	1	1	2	2	2
p		0	1	2	2	2	2
r		0	1	2	3	3	3
q		0	1	2	3	3	3
r		0	1	2	2	4	4
p		0	1	2	3	4	4

$x = 4$

$y = 3 + 1$

34

p q r r
q p q r
q p q r
q p r r

Same

Q Gate-2009 Solve or solution is explained in video (example 2009)

Subset Sum

Given a set of 'n' numbers then, is there any subset whose sum is w.

No. of subset possible $= 2^n$

So, we can use Brute force method

So, algorithm can be written as-

SS (i, s) =

① false ; i = 0, s ≠ 0
② True ; i = 0, s = 0] Base Conditions

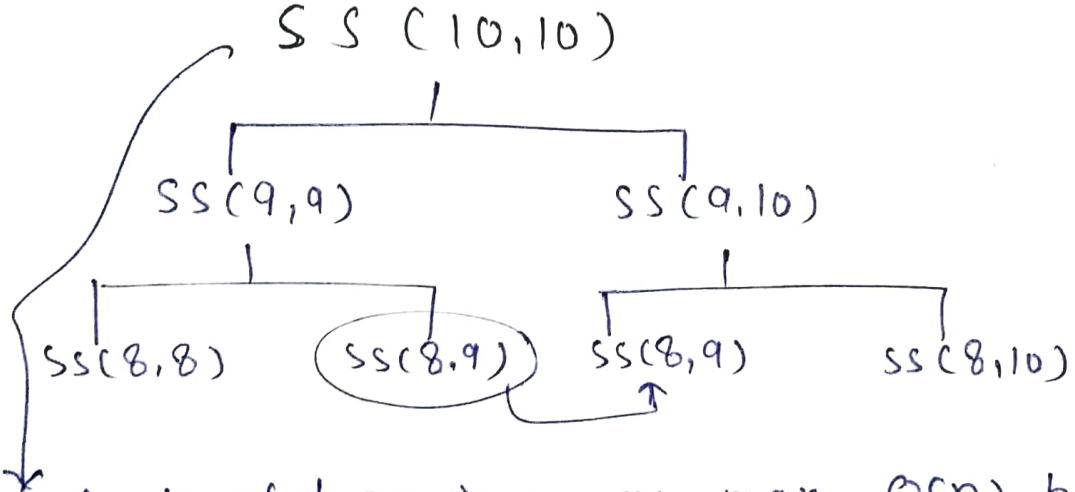
③ SS(i-1, s-ai) (U) SS(i-1, s) ; s ≥ ai

SS(i-1, s-ai) → on including (ai)
(U) (OR)
SS(i-1, s) → if you don't include (ai)

SS(i-1, s) ; s < ai

no. of elements

desired sum



depth of tree is go as max O(n) bcoz at each level one level is approached completely and not growing further.

no. of nodes as max = (2^n)

It may take $O(2^n)$ but by reducing the problem which are repeating, we can repeat it. So if we can find the unique subproblem we can achieve it in $\boxed{O(nw)}$

Algorithm and approach is similar to Knapsack not same.

Example → It is a decision problem.

Let S = {6, 3, 2, 1}

W = 5

We are interested in value not in solution.

no of element in (set)	0	1	2	3	4	5 (Sum)
0	T	F	F	F	F	F
1	T	F	F	F	F	F
2	T	F	F	T	F	F
3	T	F	F	T	F	T
4	T	T	T	T	T	T

So we can say that in the given set, subset of element is possible whose sum is equal to 5.

To compute a value of cell we are taking Or of two values as -

$$SS(4,5) = SS(3,4) \cup SS(3,5)$$

So, Time complexity - $O(nw) \times O(1)$

$= \boxed{O(nw)}$

If "w is very large" then we have no way to compute $< O(2^n)$ then best way is in $\boxed{O(2^n)}$. It is as better as brute force Algorithm.

Points to remember -

-# Brute force Algorithm (To read and write an abstract)

Multi stage Graph

Within one stage there is no edge between them. And only the edges are from stage $i \rightarrow i+1$ not from $i+1 \rightarrow i$.

Let us say there are n stages than a single node at stage 1 → source and a single node at stage n is → sink

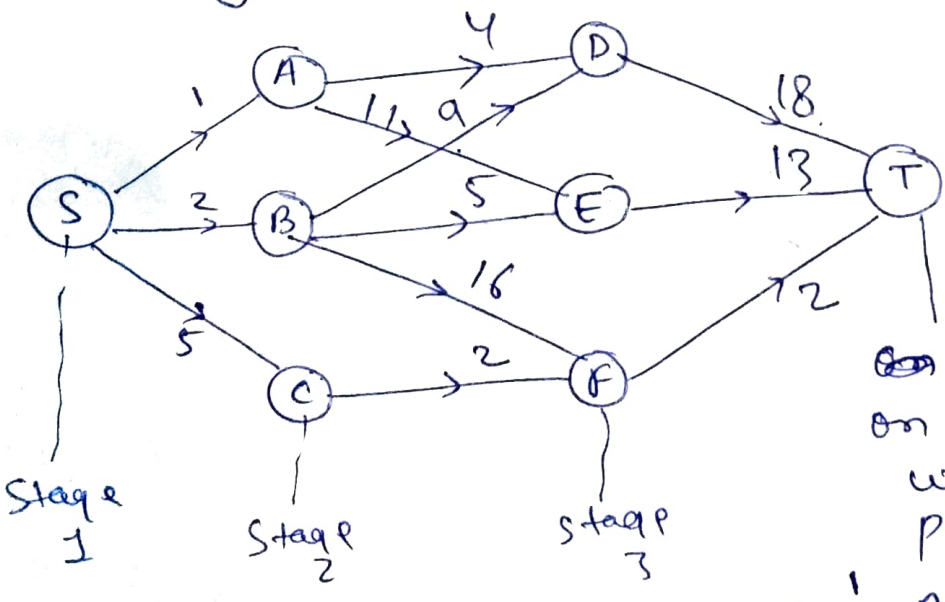
So, we have to find shortest path from source to destination.

We are not interested in shortest path from source to all nodes as happen in Dijkstra but we are interested in shortest path among source and destination or sink only.

So we interest in $T(n) < O(E \log V)$

On apply Greedy method we will opted the path with min. cost only at every nodes

Greedy is faster than dynamic approach



On apply Greedy we will get the shortest path as—

$S \xrightarrow{1} A \xrightarrow{4} D \xrightarrow{18} T$ (23)

But the shortest path is

S → C → F → T (9)

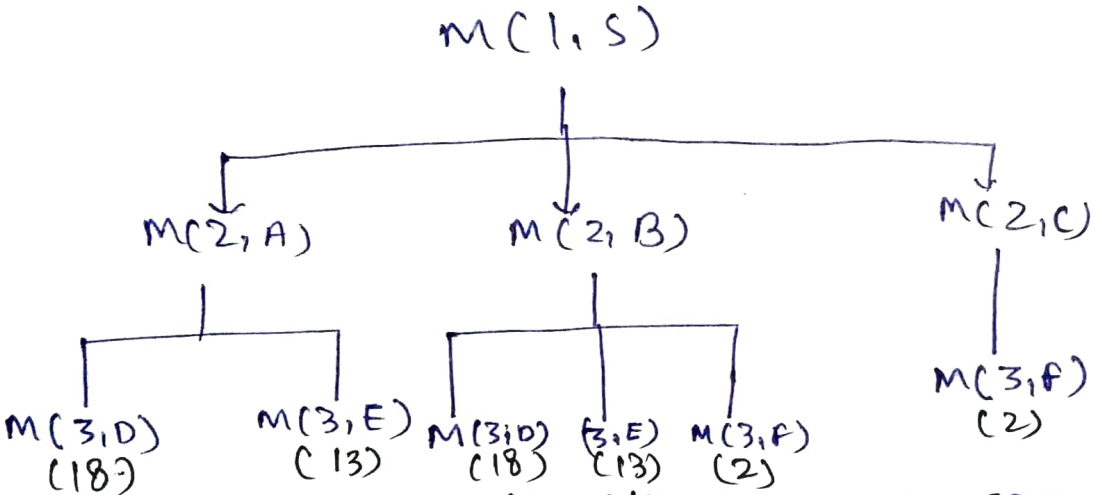
So, in this problem Greedy method fails. It generates a need to go for dynamic method

Substructure and Recursive Approach

Take recursive function as Min. cost (M)

$$M(1,S) = \min \begin{cases} S \to A + M(2,A) \\ S \to B + M(2,B) \\ S \to C + M(2,C) \end{cases}$$

stage ← 1 → node



On reaching the $(N-1)^{th}$ stage we can start the recursion as

$M(N-1, i)$ = min cost from edges to sink.

If no. of stages are k then tree will take depth of O(k) and space complexity is also O(k)

Time Complexity

At every level there could be 'n' nodes –

$$T(n) = O(k^n)$$

Since there are repeated sub-problems so we can apply dynamic programming

So, we need to find no. of distinct problems

So, no. of subproblems = $n-1$ (leave last one)

$n \rightarrow$ node or vertex.

Bottom Up - dynamic Programming Algorithm

S	A	B	C	D	E	F	T
1	2	3	4	5	6	7	8

1	2	3	4	5	6	7	8
9	22	18	4	18	13	2	0

(T') — Values are evaluated below

Logic

$T'[i] = \min$ for all $j=(i+1)$ to n { cost (i,j) + $T[j]$

for $i=7$ $T'[7] = 2+0 = 2$

$T'[6] = \min \begin{cases} \text{cost}(6,7) + T'[7] = \infty \\ \text{cost}(6,8) + T'[8] = 13 \end{cases}$

$T'[5] = 18$

$T'[4] = 2+2 = 4$

$T'[3] = \min \begin{cases} \text{cost}(3,4) + T'[4] = 60 \\ \text{cost}(3,5) + T[5] = 9+18 = 27 \\ \text{cost}(3,6) + T'[6] = 5+13 = 8 \\ \text{cost}(3,7) + T'[7] = 16+2 = 18 \end{cases}$

$T'[2] = \begin{cases} \text{cost}(2,5) + T'[5] = 4+18 = 22 \\ \text{cost}(2,6) + T[6] = 11+13 = 24 \end{cases}$

$T'[1] = \begin{cases} \text{cost}(1,2) + T'[2] = 1+22 = 23 \\ \text{cost}(1,3) + T'[3] = 2+18 = 20 \\ \text{cost}(1,4) + T'[4] = 5+4 = 9 \end{cases}$

for $i = n-1$ to 1 {

$T'[i] = \min$ for all $j=(i+1)$ to n { cost $(i,j) + T[j]$ }

Time complexity = Order of sub problem × Order of time taken by each problem

= work done by each vertices.

1	2	3	4	5	6	7	8
9	22	18	4	18	13	2	0
(7)	(6)	(5)	(4)	(3)	(2)	(1)	(1)

(1) → compare with itself and adjacent

Since the value of j is (i+1) to n so each vertices will compute to all nodes it can visit in next stage

So, all the vertices can visit total no. of times = $1+2+3+\cdots+n-1$

$= O(n^2) = O(v^2) = O(E)$

So, network done perform by all the nodes

$= \boxed{O(E)}$.

Knapsack

Difference between fractional and 0/1 Knapsack

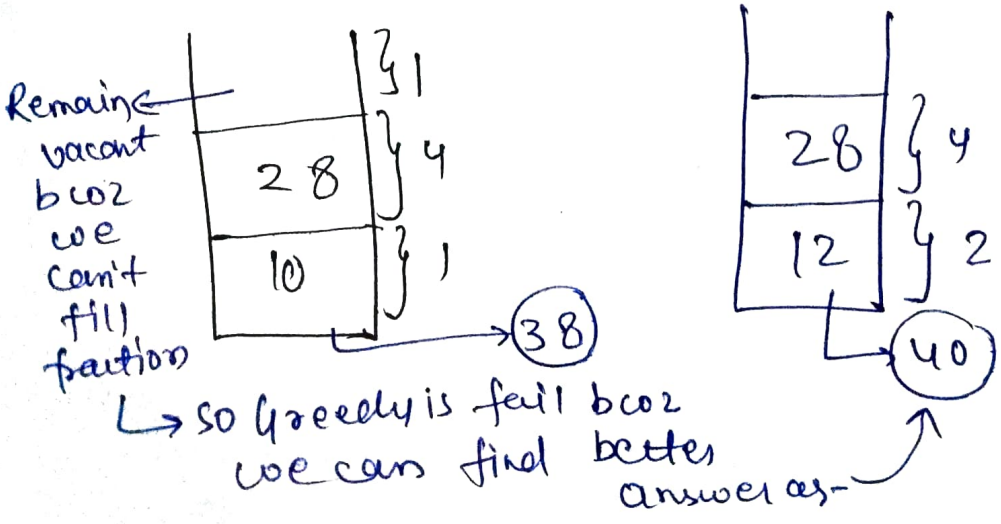
Ex- Capacity = 6

Object	1	2	3
wt	1	2	4
Profit	10	12	28
p/w	10	6	7

fractional → You can fill the object in fraction

0/1 → object are not allowed to cut into fraction

Greedy method fails in 0/1 Knapsack.



↳ so Greedy is fail bcoz we can find better answer as-

Why greedy method fails in 0/1 Knapsack?

Capacity = ω

	Ob1	Ob2
Profit	2	ω
Weight	1	ω
P/w	2	1

Here ob1 will get higher priority over ob2. So we will put 1 object and $(\omega-1)$ remain unfilled

because we can't fill fraction of obj in Knapsack

Substructure and recursive equation

Assume we have n objects.

1 2 3 . . . n

Every element have two choise either filled or not.

Total choices = 2^n

$KS(i, \omega)$ — i → no. of elements to consider. ω → capacity

So,

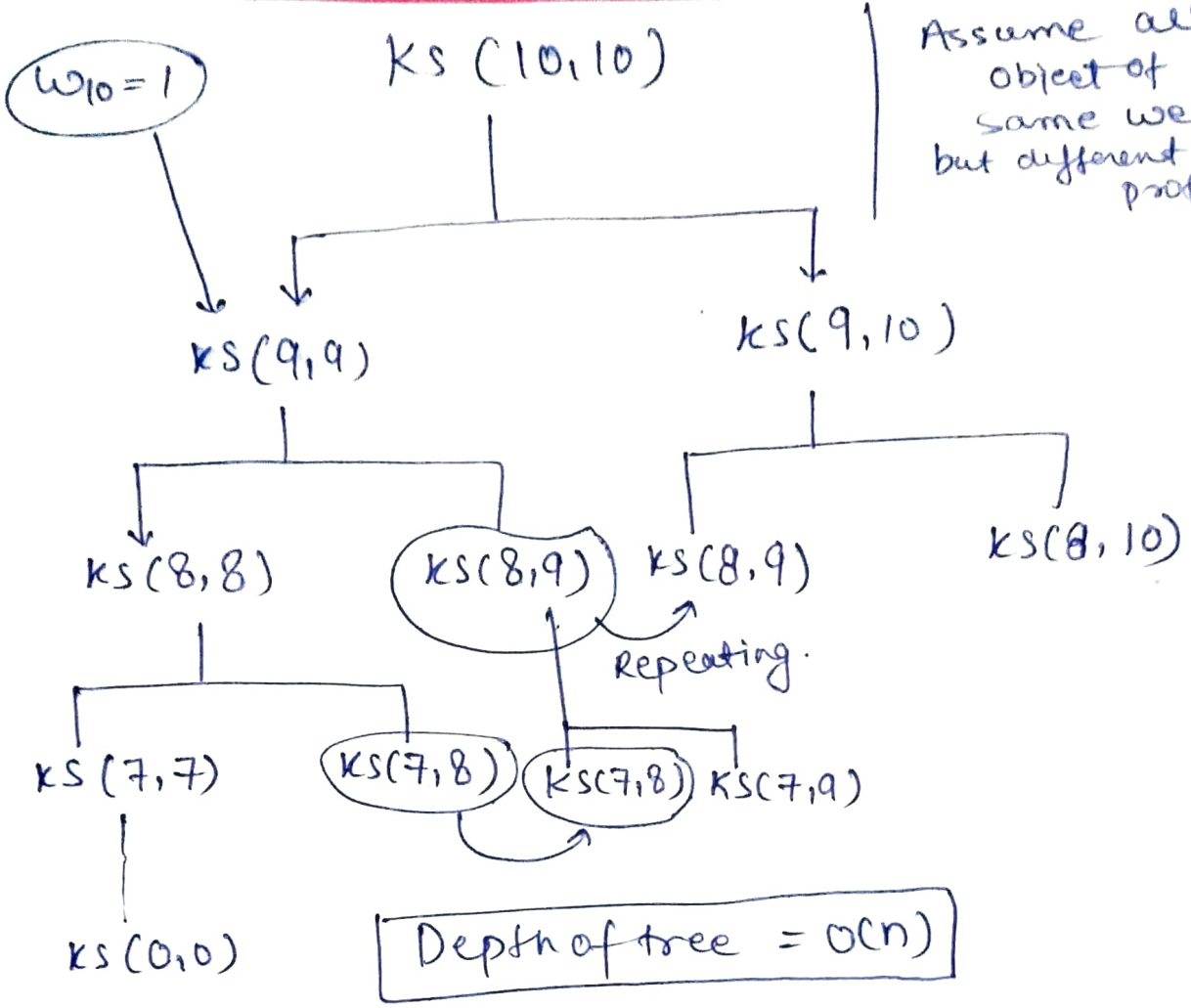
$$KS(i,\omega) = \begin{cases} \max\left(p_i + KS(i-1, \omega-\omega_i),\ KS(i-1,\omega)\right) & \text{(maximum)} \\ 0 \ ; \ i=0 \text{ or } \omega=0 \\ KS(i-1,\omega) \ ; \ \omega_i > \omega \end{cases}$$

for ith element Profit = p_i, Weight = ω_i

$KS(i-1, \omega-\omega_i)$: If ith element included

$KS(i-1, \omega)$: If ith element is not included

So, we have to call $KS(n, C)$ — n: no. of elements; C: capacity



So even the tree is half full then no. of nodes = $O(2^n)$.

Space complexity $S(n) = O(n)$

But by luck here some problems are repeating, so need to compute no. of unique sub-problems.

No. of unique sub-problems = $O(n \times w)$

So, we can solve the question by filling the table of $(n \times w)$

we will take the size $(n+1) \times (w+1)$ to fill first row and first column by 0's

$T(n) = O(2^n)$ → Don't have Bottom up Algo.

$O(nw)$ → with bottom up Algo. for small w

Example tracing out bottom up Dynamic Programming Algorithm

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0	10	12	22	22	22	22
3	0	10	12	22	28	38	40

→ max profit get

$C = 6$

	1	2	3
w	1	2	4
p	10	12	28

Here both the row major and column major order works.

$$K(2,1) = \{ K(1,1) = 10$$

$$K(2,2) = \max \begin{cases} 12 + K(1,0) = 12 \\ K(1,2) = 10 \end{cases}$$

$$K(2,3) = \max \begin{cases} p_2 + K(1,1) = 22 \\ K(1,3) = 10 \end{cases}$$

$$K(3,5) = \max \begin{cases} p_3 + K(2,1) = 38 \\ K(2,5) = 22 \end{cases}$$

$$\boxed{T(n) = S(n) = O(n \times w)}$$

And this may violate if $w = O(2^n)$ then brute force method is better than bottom up approach.

Time complexity is $\min(O(2^n), O(nw))$ depends on value of w.

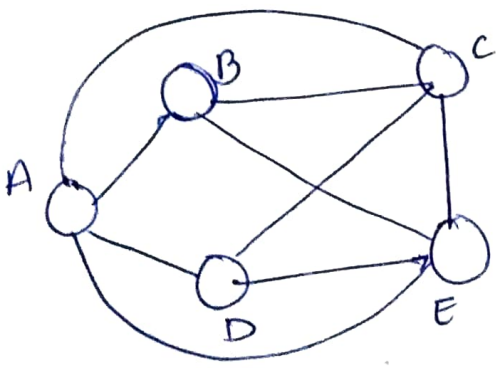
Algorithm for Knapsack problem.

Input : {$w_0, w_2, w_3 \ldots w_n$}, c, {$p_1, p_2 \ldots p_n$}

output T[n, c]

```
for i = 0 to c do
    T[0, i] = 0
for i = 1 to n {
    T[i, 0] = 0
 for (j = 1 to c) do
  if (w_i <= j) and T[i-1, j-w_i] + p_i > T[i-1, j]
 then T[i, j] = T[i-1, j-w_i] + p_i
  else T[i, j] = T[i-1, j]
 }
```

Travelling Salesman Problem



We want to visit each node exactly once with shortest path.
Starting and End point is same.

Brute force method says find all possible path, calculate cost and then find min.

Total path possible → $(n-1)!$ → $O(n!)$

$(n! >>> 2^n)$

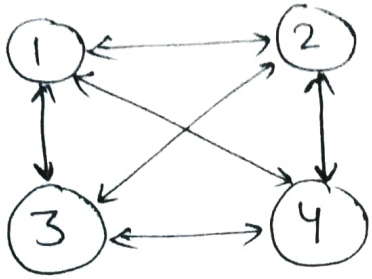
A B C D E A

A C B D E A

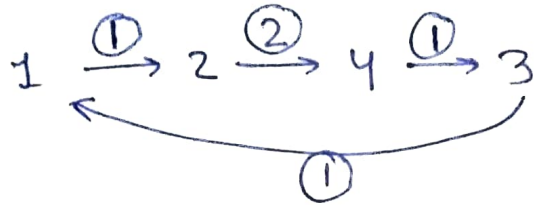
much greater so brute force is not suitable

Optimal Substructure

Ex



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



$T(1,\{2,3,4\}) = (\min)$ { $(1,2)+T(2,\{3,4\}) = 5$; $(1,3)+T(3,\{2,4\}) = 9$; $(1,4)+T(4,\{2,3\}) = 7$ }

↓↓

Tour start from 1 then visit the nodes (2,3,4) and comes back to 1

$T(2,\{3,4\}) = \min$ { $(2,3)+T(3,\{4\}) = 12$; $(2,4)+T(4,\{3\}) = 4$ }

$T(3,\{2,4\}) = \min$ { $(3,2)+T(2,\{4\}) = 7$; $(3,4)+T(4,\{2\}) = 10$ }

$T(4,\{2,3\}) = \min$ { $(4,2)+T\{2,\{3\}) = 8$; $(4,3)+T(3,\{2\}) = 4$ }

$T(3,\{4\}) = (3,4)+T(4,\phi) \rightarrow 8$

$T(4,\{3\}) = (4,3)+T(3,\phi) \rightarrow 2$

$T(2,\{4\}) = (2,4)+T(4,\phi) \rightarrow 5$

$T(4,\{2\}) = (4,2)+T(2,\phi) \rightarrow 5$

$T(2,\{3\}) = (2,3)+T(3,\phi) \rightarrow 5$

$T(3,\{2\}) = (3,2)+T(2,\phi) \rightarrow 3$

$T(2,\phi) = (2,1) = 1$

$T(3,\phi) = (3,1) = 1$

$T(4,\phi) = (4,1) = 3$

Bottom Up Dynamic Programming Approach

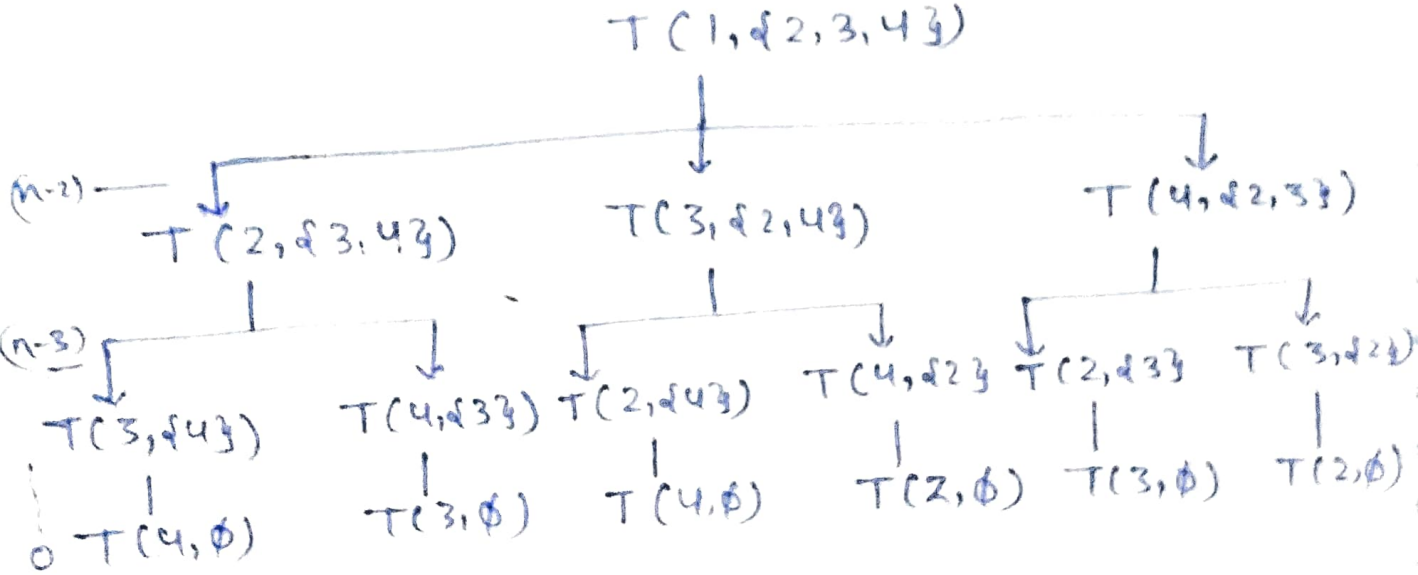
Recursive equation

$$T(i,s) = \min_{(j \in s)} \{ (i,j) + T(j, s-\{j\}) ; s \neq \phi$$

i → start node = $(i,1)$; $s = \phi$

s → set of non-visited nodes # If (i,j) are not adjacent node then cost $(i,j) = \infty$. Otherwise you will get the definite cost

Recursion tree : Vertices = (1, 2, 3, 4)



Subproblems (Number of unique recursion)

$T(i, S)$

- At each level I have (n-1) values
- started from (n-1) and down to φ.

Total subproblem = 15
Unique = 12 + 3(Repeated)

Ex-2

V = n

Total no. of subproblem = $O(n2^n)$

Time required to solve each problem = $O(n)$

So, T(n) equal to

(1, {n-1})

$(n-1) \rightarrow T(i, \{n-2\})$

$(n-2)\ C_{(n-3)}$

$(n-2)\ C_{n-4}$

$(n-1) + (n-1)^{n-2} C_{n-3} + (n-1)^{n-2} C_{n-4} + \cdots$

$\cdots + (n-1)^{(n-2)} C_0$

$\Rightarrow (n-1) \sum_{k=0}^{n-2} (n-2) C_k = (n-1) 2^{n-2}$

$= O(n2^n) < O(n!)$

So, $T(n) = O(n^2 2^n)$. It is exponential but it is better not polynomial the $O(n!)$.

So, it is also example of **NP Complete**.

Total no. of distinct sub problems

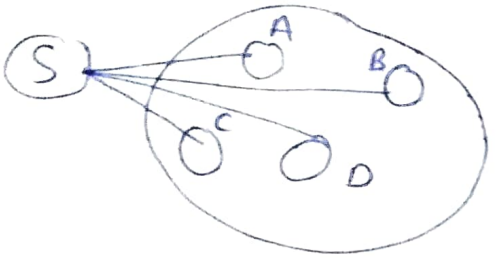
$= (n-1) 2^{n-2}$

→ This size table we can't compute manually.

Floyd Warshall
(All pairs shortest path)

① Difference btw single source shortest path and all pairs shortest path.

Single source shortest path



on allpair shortest path

S, A, B, C, D all the nodes act as source and destination both.

We can apply single source shortest path at all nodes to get all pairs shortest path but its quite slow.

On applying Dijikstra Algo at each vertex then

$$T(n) = O(E \log V) * O(V)$$

$$= O(VE \log V) = O(V^3 \log V)$$

and using Bellman ford → $T(n) = O(VE) \times V$

$$= O(V^4)$$

And both the time complexity are very much. so we apply dynamic programming to get minimum Time

Optimal Substructure

Let set of vertices $(V) = \{1, 2, \ldots n\}$

and $i, j \in V$

(dij^k) → distance of shortest path from vertex $i \to j$ using vertex from (1 to K)

$(dij^x) =$ "p"

Path 'p' may contains the vertex k or not

"p" = $d_{ij}^{(k-1)}$ (if k not included) ; $k>0$

if k included break p into two paths p_1 and p_2

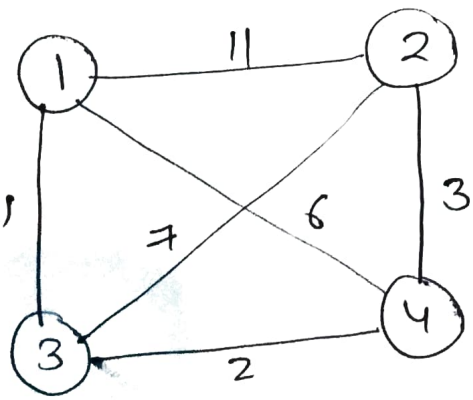
$p_1 = d_{ik}^{(k-1)}$ and $p_2 = d_{kj}^{(k-1)}$; $k>0$

$p = p_1 + p_2$ $\{ d_{ij}^{(k-1)} + d_{jk}^{(k-1)}$

$p = w_{ij}$; $k=0$ (Base Condn)

Example. Graph could be directed or undirected and with positive and negative weights. → Floyd Warshall.

But this Algo. is not capable to detect the negative weight cycles.



$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array} \quad D_0 = (D_0)^T$$

Set of all distances contains atmost 1 edge.

Matrix is symmetric bcoz graph is undirected.

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Shortest path between 2 nodes passing through node 1 is allowed

$D_1 = (D_1)^T$

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Min. distance btw two vertices including the vertex 1 and 2 both

$D_2 = (D_2)^T$

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 1 & 3 \\ 8 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 3 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Here we can consider the vertices $\{1, 2, 3\}$

$D_3 = (D_3)^T$

$$D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 6 & 1 & 3 \\ 6 & 0 & 5 & 3 \\ 1 & 5 & 0 & 2 \\ 3 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Here we can involve $\{1, 2, 3, 4\}$

$D_4 = (D_4)^T$

$\boxed{D = D^T}$ bcoz graph is undirected.

and we need to verify the same in case of directed graph also

Time and space Complexity :-

In every matrix we solve $O(V^2)$ problems.

and total no. of matrix $= O(V)$

So total values solve $= O(V^3)$

Time taken to solve each value $= O(1)$

So, Total time taken $= O(1) \times O(V^3)$

$= \boxed{O(V^3)} = O(n^3)$

Space Complexity

We need only two matrices to compute the values.

Size of matrix should be $O(V^2)$

We can use the previous to previous matrix to store new values. So.

Space Complexity = $O(V^2) \rightarrow O(n^2)$

Algorithm

```
f_w(w){
1. n = w.rows
2. D^0 = w
3. for k = 1 to n
4. Let D^k = (dij^k) be a new nxn matrix
5. for i = 1 to n
6. for j = 1 to n
7. dij^(k) = min (dij^(k-1), dik^(k-1) + dkj^(k-1))
8. return D^n.
```

Points to remember

1. In all the Algorithms discussed remember only method and recursive equation.
2. There is no need to remember the Algorithm.
3. Space Complexity and Time Complexity is need to remember.

NP Completeness (NOT in Gate) But discussed after 9mp. points further.

(2) Unbalanced : $S(n) = \Theta(n)$, $T(n) = O(n)$

Quick Sort : Worst case : $T(n) = T(n-1) + n = O(n^2)$

Best case : $T(n) = 2T(n/2) + n = \Theta(n \log n)$

Worst case sequences : Ascending / Descending order, All same.

If no. is devided into some ratio, as —

$$T(n) = T(n/3) + T(2n/3) + n$$

then, $T(n) = \Theta(n \log n)$

Heaps : Index of L child = $2i$ | start with 1
R child = $2i+1$ | $2(i+1)$ | start with 0

Max no. of nodes in n-ary tree of height 'h' = $\left[\frac{n^{h+1} - 1}{n-1}\right]$

Height of binary heap = $\Theta(\log_2 n)$.

In heap non-leaves present $\lfloor n/2 \rfloor$ to 1.

In heap leaves present $\lfloor n/2 \rfloor + 1$ to n.

If left and right tree of root are already heap then to heapify root called MAX_HEAPIFY. $\Theta(\log n) = S(n) = T(n)$

Build_MAX_HEAP from given nodes, $T(n) = O(n)$, $S(n) = O(\log n)$

Delete or extract MAX from heap : $T(n) = S(n) = O(\log n)$

Linear search : $T(n) = T(n-1) + 1$: $T(n) = O(n)$

Binary Search : $T(n) = T(n/2) + 1$: $T(n) = O(\log_2 n)$

Ternary Search : $T(n) = T(n/3) + 1$: $T(n) = O(\log_3 n)$

for Max heap : (1) find Max : $O(1)$ (2) Delete MAX : $O(\log n)$

(3) Insert / Increase key / Decrease key : $O(\log n)$

(4) find min. / search / deletion : $O(n)$

Stirling Approximation : $\lfloor n = O(n^n) = \omega(2^n)$

and $(\log \lfloor n) = \Theta(n \log n)$

HeapSort : $T(n) = O(n \log n)$

Time complexity of computing transitive closure of binary relations on a set of 'n' elements = $O(n^3)$: Using warshall Algo.

Bubble sort : $T(n) = O(n^2)$ worst case, $T(n) = O(n)$ best case ← no. of comparisons

Bucket sort
for n insertion takes $O(n)$
$S(n) = O(n+k)$
for no. ↓ for indexes
$T(n) = O(n^2)$
→ for floating point numbers.

Counting sort
→ for the no. in given range
$T(n) = O(n+k)$
$S(n) = O(k)$

Radix sort
$T(n) = O(n \log_b l)$
$S(n) = O(b)$
b → base of no. system
l = largest no. $(d \cdot \log_b l)$

Selection sort
$T(n) = O(n^2)$
$S(n) = O(1)$

In quick sort if we do partioning in both balanced & unbalanced then, $T(n) = n \log n$

Greedy Algo:

Time complexity can be computed as : $H(t) \times$ work done
$H(t)$ → height of tree, WD → work done at each level.

① **Knapsack** : find (p/w) ratio and fill in sack. $T(n) = O(n \log n)$

② **Huffman code** : $T(n) = O(n \log n)$ → Tree उल्टा बनाओ using Min. heap concept

③ **Job Sequence** : $T(n) = O(n^2)$: Job की sequence तोड़ो end time पर।

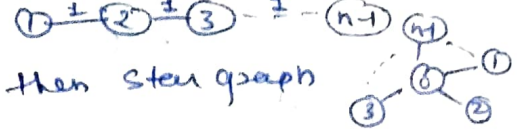
→ we find external path weight in Huffman coding using non-uniform coding.

Spanning Tree : No. of edges = $(n-1)$
No. of Spanning tree in $K_n = (n^{n-2})$
No. of spanning tree for non-complete graph using kirchoff's Algo.
In kirchoffs Algo. form an Adjacency matrix → Replace all diagonal zeroes from degree of node → find cofactor.

Prims / Kruskals (P/K) : → fibonacci heap $(E + V \log V)$

Prims : Start from min. cost edge
$T(n) = O(E \log V)$ min heap
$O(V^2)$ without min heap

Kruskals : It can form a forest
$T(n) = (E \log V)$
Process start same as Prim's, but we can choose any edges

In a min. spanning tree if weight of edge (u,v) is |u-v|
then Line graph with wt:1 
if weight of edge (u,v) is |u+v| then star graph

Dijkstra Algorithm: It is a single source shortest path algo.

→ T(n) = O(E log V), but it is not useful in case of (-ve) edge cycle, either we can use it in case of (-ve) edge without cycle, it has no capabilities to detect whether there is a cycle or not. (Using binary heap (E log v + V log v))

→ If graph is disconnected then there is no way to find the shortest paths.

Note: Do the relaxation of edge very carefully. Also, T(n) = O(V²)

Bellman Ford Algo.: If value is changing after (n-1) attempts, it means there is a (-ve) edge cycle.

→ This algo. performs (n-1) attempts of operations where n = no. of edges.

→ T(n) = O(VE) : More than Dijkstra Algo.

Matrix Multiplication: We calculate Total scalar multiplication.

→ Generally $((A_{2\times1}\ B_{1\times2})\ C_{2\times4})$
$\rightarrow 2\times1\times2=4$
$(AB_{2\times2}\ C_{2\times4})$
$\rightarrow 2\times2\times4=16$
TSM = 4+16 = 20

No. of paranthesis combination possible over n-matrices
$= \frac{2nC_n}{n+1}$ but put n-1 for n matrices

* for an order

A_1	A_2	A_3	A_4
1×2	2×1	1×4	4×1
p_0 p_1	p_1 p_2	p_2 p_3	p_3 p_4

* T(n) = O(n³)
S(n) = O(n²)

* $m[i,j] = \begin{cases} 0 & \text{if } i=j; \\ \min\{m[i,k]+m[k+1,j] + p_{i-1}\,p_k\,p_j\} & \text{otherwise} \end{cases}$

Longest Common Subsequence:

No. of subsequence possible = 2^m (for m characters) = $O(2^m)$

* $C[i,j] = \begin{cases} 0 & ; i=j=0 \\ 1+C[i-1,j-1] & ; i,j>0 \ \&\ x_i=y_j \\ \max(C[i-1,j], C[i,j-1]) & : i,j>0 \ \&\ x_i \neq y_j \end{cases}$

Best case: T(n) = s(n) = O(n) if both x and y are same as x = ABCD y = ABCD

Worst case : $x = AAAA$ (n) $\quad y = BBBB$ (m)

Height of tree = $O(n+m)$

no. of nodes in tree = $O(2^{n+m})$

But in dynamic approach some cells are repeated so,

final $T(n) = S(n) = O(m \times n)$

Subset sum : $T(n) = S(n) = O(n \times w)$ → required w to no. of elements

If 'w' is very large then there is no ways to compute $T(n) < O(2^n)$

Multi-stage Graph:- ① We have a source at level 1 and sink at level k.

② There can be an edge from level $i \rightarrow i+1$ only, no edges between the nodes of same level.

No. of comparison : → $O(n^2)$ So, $T(n) = O(n^2) = O(V^2) = O(E)$

0/1 Knapsack : Everything is same as Subset sum problem.

All pair shortest path :

① Dijkstra takes $T(n) = O(V^3 \log V) = (VE \log V)$

② Bellman ford takes $T(n) = O(V^4) = (V \cdot E) \cdot V$

③ floyd warshall takes $T(n) = O(V^3) = O(n^3)$

$S(n) = O(V^2)$

Travelling salesman Problem : $T(n) = O(n^2 2^n)$] NP complete

Total no. of distinct sub problems = $(n-1)2^{n-2}$]

Subsetsum =

$SS(i, s)$ = (i: no. of element, s: desired sum)

$$SS(i,s) = \begin{cases} F & i=0, s \neq 0 \\ T & i=0, s=0 \\ SS(i-1, s) & s < a_i \\ SS(i-1, s-a_i) \text{ U } SS(i-1, s) & s \geq a_i \end{cases}$$

Knapsack :

$$ks(i,w) = \begin{cases} \max(p_i + ks(i-1, w-w_i), ks(i-1, w)) \\ 0 \text{ ; } i=0 \text{ or } w \leq 0 \\ ks(i-1, w) \text{ ; } w_i > w \end{cases}$$

floyd warshall: $T(n) = O(n^3)$: Dynamic Prog.

$= O(V^4)$: Bellman ford

$= O(V^3 \log V)$: Dijkstra.

$S(n) = O(V^2)$

$$d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$$

Important points in PYQ

① Asymptotic Notation

① If $f(n) = O(g(n)) \Rightarrow f(n) \leq c \cdot g(n)$, $f(n) = \Omega(g(n)) \Rightarrow f(n) \geq c \cdot g(n)$

② If you can verify via option do it first.

③ Time Complexity of computing transitive closure for a Binary relation $= O(n^3)$ using Floyd Warshall Algo.

④ In an array min. no. of comparison needed to find min. and max. value is $\lceil \frac{3n}{2} \rceil - 2$.

⑤ In selection sort each time 'n' comparisor occurs but only 1 swap occurs.

⑥ Best searching algo. is binary search.

Operation Table to remember

	find	insert	Delete	Decrease key
Unsorted Array	$O(N)$	$O(1)$	$O(N)$	$O(1)$
Min heap	$O(N)$	$O(\log N)$	$O(N)$	$O(\log N)$
Sorted Array	$O(\log N)$	$O(N)$	$O(N)$	$O(N)$
Sorted DLL	$O(N)$	$O(N)$	$O(1)$	$O(N)$

Imp. Question to look at : 1.5, 1.10, 1.17, 1.25, 1.32, 1.34, 1.35, 1.45, 1.49, 1.52, 1.58 only.

② Devide and Conquer

→ BFS and DFS both used to find connected graph.

→ No. of entries in hash table ↑ ⇒ No. of collision ↑

→ In a sorted array time requires to find two elements whose sum is less than K is $O(1)$.

→ In BST both insertion and deletion takes $O(n)$.

Imp. Question: 2.6, 2.9, 2.13, 2.17, 2.20, 2.22, 2.24, 2.29, 2.34, 2.35 only

→ If we choose median as pivot then balanced partitioning and Avg $O(n) = n \log n$.
Tight upper bound $O(n) = (n^2)$

In 2.17 $O(n^2)$ bcoz $T(n) = n \times (n+n) \rightarrow$ for linear search; $T(n) = n \times (\log n + n) \rightarrow$ for binary search

③ Greedy Approach

→
```
int gcd(m,n){
  if (m%m==0) return m;
  n = n%m;
  return gcd(n,m); }
```

It is an Euclidian method to calculate GCD.

$T(n) = O(\log_2 n)$

Test Series

* Time complexity of a complete graph is $O(n!)$
* All pairs shortest path : floyd warshall.
* Insertion, Bubble, Merge and Binary tree sort are stable but Quick, heap and selection sort are not stable. (3^{rd} Jan)
* floyd warshall : $O(V^3)$, Dijkstra : $O(V^2)$
  Kruskal's : $O(E \log E)$, Topological : $O(V+E)$
* Recurrence relation for strassen Matrix Multiplication is $T(n) = 7T(n/2) + n^2$ $O(n^{2.81})$
* To multiply two long integers we have, DFC, Toom's Algo. + Karatsuba Method.
* Space Complexity of Travelling salesman problem with n vertices is $\rightarrow O(n.2^n)$.
* Time Complexity of 0/1 knapsack for weight $\underline{w}$ = $O(n.w)$ and $\underline{n}$ items
* Problem of subset sum can be solved using both dynamic and backtracking.
* Out of place Algo: means algo. is taking extra space.
* Travelling Salesman problem using Brute force method takes $T(n) = O(n^n)$
* Prims = $O(E \log V + V \log V + E) = O(E \log V)$ min heap
  $= O(E + V \log V)$ = fibonacci heap.

→ Any decision tree that sorts n-elements has height : $\underline{\Omega(n \log n)}$.

→ If b is branching factor and m is max. depth of search tree, what is the S(n) for greedy search = $\underline{O(b^m)}$

→ Non-recursive is overall better than recursive in case of Space and Time.

→ $T(n) = n * T(n-1) \Rightarrow T(n) = O(n!)$

→ Problem solved by DP can be solved by backtracking too but not vice versa.

→ Hamiltonian cycle, n-queen : Backtracking.

→ Only (O) notation is symmetric.

→ Shortest path and longest path computation in directed graph possible using DP with presence of (-ve) edge wts.

→ Ten) for Havel- Hakimi Theorem : $O(n^2 \log n)$

→ If no. of edges = no. of vertices − 1 in undired graph then most efficient algorithm to find G is connected or not is $O(m+n)$

→ Ten) for Fibonacci computation — $O(2^n)$ otherwise, $O(n)$ using DP.

→ Prim's and Dijkstra uses the idea of ~~Prims~~ BFS.

→ for a directed graph with no back edges then DFS tree means no cycles.

→ No. of comparison binary search = $2\log_2 n + 1$: Better
Ternary —— = $4\log_2 n + 1$

→ In merge sort, merging of two list of size, n, m takes $(n+m-1)$ comparisons.

→ If a graph with n vertices and $n-1 = m$ edges then it should be disconnected graph and to find graph is connected or not takes $O(n+m)$

→ If a directed graph has no back edges in DFS tree, it means it has no cycles.

→ A Queue can be used to implement Radix sort.

→ In Floyd Warshall graph can be directed or undirected and (+ve) or (-ve) edges, but it can't detect the (-ve) cut cycles

→ To compute transitive closure : $d_{ij}^{k+1} \;||\; d_{ik}^{k+1} \;\&\&\; d_{kj}^{k+1}$

→ Evaluation of $(n^{1/3})$ takes $(\log n)^k$ times.