Dynamic programming

Dynamic Programming is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub instances.

Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems .

Main idea:

- set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
- solve smaller instances once
- record solutions in a table
- extract solution to the initial instance from that table

- Applicable when sub problems are **not** independent
  - Sub problems share sub problems

E.g.: Combinations:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\binom{n}{1} = 1 \qquad \binom{n}{n} = 1$$

A divide and conquer approach would repeatedly solve the common sub problems

Dynamic programming solves every sub problem just once and stores the answer in a table

- Used for **optimization problems**
  - A set of choices must be made to get an optimal solution
  - Find a solution with the optimal value (minimum or maximum)
  - There may be many solutions that lead to an optimal value
  - Our goal: **find an optimal solution**

Dynamic Programming Algorithm

1. **Characterize** the structure of an optimal solution
2. **Recursively** define the value of an optimal solution
3. **Compute** the value of an optimal solution in a bottom-up fashion
4. **Construct** an optimal solution from computed information (not always necessary)

- An algorithm design method that can be used when the solution to a problem may be viewed as the result of a sequence of decision.
- Dynamic Programming algorithm store results, or solutions, for small subproblems and looks them up, rather than recomputing them, when it needs later to solve larger subproblems
- Typically applied to optimiation problems

Principle of Optimalilty

- An optimal sequence of decisions has the property that whatever the initial state an decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.
- Essentially, this principles states that the optimal solution for a larger subproblem contains an optimal solution for a smaller subproblem.

Dynamic Programming VS. Greedy Method

Greedy Method

- only one decision sequence is ever genertated.

Dynamic Programming

- Many decision sequences may be genertated.

Dynamic Programming VS. Divide-and Conquer

Divide-and-Conquer

- partition the problem into independent subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem

Dynamic Programming

- applicable when the subproblems are not independent, that is, when subproblems share subsubproblems.
- Solves every subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time the subsubproblem is encountered

0/1 Knapsack:

- We have n items each with an associated weight and value (profit). The objective is to fill the knapsack with items such that we have a maximum profit without crossing the weight limit of the knapsack.
- Since this is a 0/1 knapsack problem, we can either take an entire item or reject it completely.
- To solve this, we have 2 methods:
  1) Set Method
  2) Table Method

*Table Method (or) Tabulation:

Problem Statement:

The maximum weight the knapsack can hold is m = 11. There are 5 items to choose from. The weights and profits are

$P_i = \{1, 6, 18, 22, 28\}$

$w_i = \{1, 2, 5, 6, 7\}$

Tabulation, the name itself indicates we need to create a table with 5 rows (no. of objects) including 0th object 6 rows and with 12 columns from 0 to 11 because Here 11 is the maximum capacity of the knapsack and 0 is the minimum capacity of the knapsack.

→ initially, 0th object means no weight hence place all zero's in both 0th row & 0th column as from given table

Objects	01	02	03	04	05
weights	1	2	5	6	7
profits	1	6	18	22	28

	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
1	0											
2	0											
3	0											
4	0											
5	0											

for 1st object,

	wt	pf
01	1	1

at weight = 1 and 01 = 1 we need to place value 1 at wt = 1

0 1 — Sequentially increasing

remaining values are ≥ 1, Hence we need to place 1 at every cell at object 1

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

For object 2

objects	01	02	03	04	05
weights	1	2	5	6	7
profits	1	6	18	22	28

	wt	pf
01	1	1
02	2	6

For that, at 02, wt = 2 we need to place valu 6

and at wt = 1, we need to place previous value

now,

	wt	pf
01	1	1
02	2	6

wt	pf
1+2	1+6
3	7

at wt = 3, we need to place value 7

Hence, we get at 02

0 1 6 7 — 7 = 7

Sequentially increasing manner there we need to place 7.

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

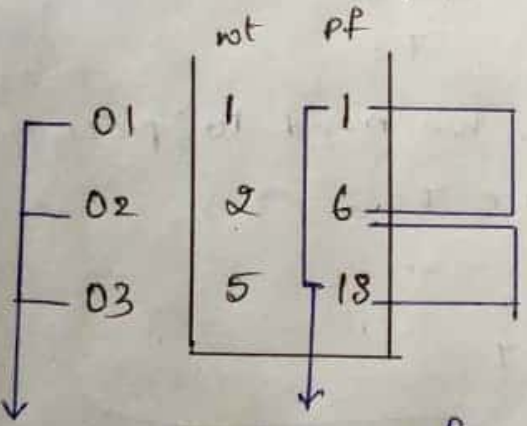
for object 3,

objects :	01	02	03	04	05
weights :	1	2	5	6	7
profits :	1	6	18	22	28

	wt	pf
01	1	1
02	2	6
03	5	18

at wt = 5, we need to place 18

and upto 5th weight, we need to place previous values, now coming to all possible combinations



wt: 1+2 = 3
pf: 1+6 = 7 → already placed

wt: 2+5 = 7
pf: 6+18 = 24 → at wt 7 profit is 24

at wt: 1+2+5
pf: 1+6+18

wt: 1+5 = 6
pf: 1+18 = 19

8 25 at weight 6, profit = 19

at weight 8,
profit 25

place 25

0 1 6 7 7 18 19 24 25

	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
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0											
5	0											

In the same manner, we need to find values of entire table

	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
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

Formula:

We have formula also to calculate profits with respective their weights of objects

$$v[i, w] = \max\{v[i-1, w],\ v[i-1, w - w[i]] + p[i]\}$$

(i: object, w: weight)

eg: $v[5, 1] = \max\{v[4, 1],\ v[4, 1-7] + 28\}$

$= \max\{v[4, 1],\ v[4, -6] + 28\}$

$= 1$

$v[5,11] = \max\{v[4,11], v[4,11-7]+28\}$

$= \max\{v[4,11], v[4,4]+28\}$

$= \max\{40, 7+28\}$

$= 40$

calculating $x1, x2, x3, x4, x5$:

→ Max value = 40 present in 4th row first so include 4th object $x5 = 0, x4 = 1$

→ subtract 40 & corresponding profit i.e $22 = 18$

→ 18 is initially present in 3rd row so include 3rd object $x3 = 1$

→ subtract 18 with corresponding profit i.e $18 = 0$

→ since we can't take any more object because profit is 0, so we don't select 2nd and 1st objects

$x2 = 0, x1 = 1$

→ so, finally objects selected are

$\{x1, x2, x3, x4, x5\}$

$\{0, 0, 1, 1, 0\}$

means

wt		pf
5+6		18+22
11	→	40

2. Set method

Set method is used to find all possibilities and become the best solution.

Problem Statement :

1) Find the optimal solution for the 0/1 knapsack problem making use of dynamic programming approach.

Sol:-

$m = 8$

$n = 4$

$P = \{1, 2, 5, 6\}$

$W = \{2, 3, 4, 5\}$

P- profit, W- weight

Let us start with set zero, (S^0)

$S^0 = \{(0, 0)\}$ in this we say no profit & no weight.

Consider 1st object,

$S_1^0 = \{(1, 2)\}$

Now prepare set one, merge the above two S^0 and S_1^0

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

consider 2nd object, i.e, S_1^1 add (2, 3) to these order pairs

$S_1^1 = \{(2,3), (3, 5)\}$

this is 2nd set for the second object.

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

prepare S_1^2 by considering (5,4) for 3rd object.

$$S_1^2 = \{(5,4)(6,6)(7,7)(8,9)\}$$

here profit is 8 and the weight is 9 it is exceeding the capacity of bag so, exclude it.

merge all these then we get.

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

here profit is increased but weight is decreased

we have to discard any one. so we will discard any one with smaller profit.

i.e, (3,5).

Consider 4th object (6,5),

$$S_1^3 = \{(6,5)(7,7)(8,8)(11,9)(12,11)(13,12)\}$$

These 3 are exceeding the capacity of bag so, exclude it.

merge these order pairs shown above.

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

here the profit is remaining same but weight is reduced.

so, according to dominance rule we should discard any one with smaller profit i.e, (6,6).

here the making order pair is (8,8)

① $(8,8) \in S^4$

but $(8,8) \notin S^3$ $\therefore x_4 = 1$

(8,8) → subract it from 4th object

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

② check whether $(2, 3)$ is in S^3. so,

$(2, 3) \in S^3$

$(2, 3) \in S^2$ this is not because of third object

therefore $x_3 = 0$.

③ $(2, 3) \in S^2$ $\therefore x_2 = 1$.

but $(2, 3) \in S^2$

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

④ $(0, 0) \in S^1$

and also, $x_1 = 0$

$(0, 0) \in S^0$

From bottom if we check $(0, 1, 0, 1)$

So, the the answer is $(0, 1, 0, 1)$

$P = \{1, 2, 5, 6\}$

$W = \{2, 3, 4, 5\}$

$\{0, 1, 0, 1\}$ //

Matrix chain Multiplication:

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C and D, we would have

$$(ABC)D = (AB)(CD) = A(BCD) = \ldots$$

Matrix chain multiplication is an optimization problem that to find the most efficient way to multiply given sequence of matrices. The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved. The number of multiplications that needed varies based on the paranthesization.

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

If given 4 matrices A_1, A_2, A_3 and A_4 the combination of paranthesization is equal to the catalog number

$$\text{catalog number} = \frac{{}^{2n}C_n}{n+1}$$

Here n = number of matrices − 1

In the above $A_1 \times A_2 \times A_3 \times A_4$ $n = 4-1 = 3$

$$\text{catalog number} = \frac{{}^{2\times3}C_3}{3+1} = \frac{{}^{6}C_3}{4}$$

$$= \frac{\frac{6\times5\times4}{3\times2}}{4} = 5$$

Ex: 1. $A_1((A_2 \times A_3) \times A_4)$

2. $A_1(A_2 \times (A_3 \times A_4))$

3. $(A_1 \times A_2) \times (A_3 \times A_4)$

4. $((A_1 \times A_2) \times A_3) \times A_4$

5. $(A_1 \times (A_2 \times A_3)) \times A_4$.

Ex: Calculate cost of each sub matrix and stored them in a table. and find minimum of $c[i,j]$ for A_1, A_2, A_3 and A_4 of size 4×10, 10×3, 3×12 and 12×20.

Sol:

All the diagnals cost in the table = 0.

So, $c[1,1]$, $c[2,2]$, $c[3,3]$, $c[4,4] = 0$.

c	1	2	3	4
1	0			
2		0		
3			0	
4				0

(i ↓, j →)

A_1	A_2	A_3	A_4
4×10	10×3	3×12	12×20
d_0 d_1	d_1 d_2	d_2 d_3	d_3 d_4

$$c[1,2] = \min_{1\le k<2}\{c[1,1] + c[2,2] + d_0 \times d_1 \times d_2\}$$

$$= 0+0+4\times10\times3$$

$$= 120.$$

We need another table to store the value of k so that the cost is minimum.

cost table

C	1	2	3	4
1	0	120		
2		0		
3			0	
4				0

K	1	2	3	4
1		1		
2				
3				
4				

$c[2,3] = \min_{2 \le k < 3} \{ k=2 \{ c[2,2] + c[3,3] + d_1 \times d_2 \times d_3 \} \}$

$= 0 + 0 + 10 \times 3 \times 12$

$= 360.$

C	1	2	3	4
1	0	120		
2		0	360	
3			0	
4				0

K	1	2	3	4
1		1		
2			2	
3				
4				

$c[3,4] = \min_{3 \le k < 4} \{ k=3 \{ c[3,3] + c[4,4] + d_2 \times d_3 \times d_4 \} \}$

$= 0 + 0 + 3 \times 12 \times 20$

$= 720$

C	1	2	3	4
1	0	120		
2		0	360	
3			0	720
4				0

K	1	2	3	4
1		1		
2			2	
3				3
4				

$$c[1,3] = \min_{1 \le k < 3} \begin{cases} k=1 & c[1,1] + c[2,3] + d_0 \times d_1 \times d_3 \\ k=2 & c[1,2] + c[3,3] + d_0 \times d_2 \times d_3 \end{cases}$$

$$= \min \begin{cases} 0 + 360 + 4 \times 10 \times 12 \\ 120 + 0 + 4 \times 3 \times 12 \end{cases}$$

$$= \min(840, 264)$$

$$= 264.$$

c	1	2	3	4
1	0	120	264	
2		0	360	
3			0	720
4				0

(i ↓, j →)

k	1	2	3	4
1		1	2	
2			2	
3				3
4				

(i ↓, j →)

$$c[2,4] = \min_{2 \le k < 4} \begin{cases} k=2 & c[2,2] + c[3,4] + d_1 \times d_2 \times d_4 \\ k=3 & c[2,3] + c[4,4] + d_1 \times d_3 \times d_4 \end{cases}$$

$$= \min \begin{cases} 0 + 720 + 10 \times 3 \times 20 \\ 360 + 0 + 10 \times 12 \times 20 \end{cases}$$

$$= \min(1320, 2760)$$

$$= 1320.$$

c	1	2	3	4
1	0	120	264	
2		0	360	1320
3			0	720
4				0

	1	2	
		2	2
			3

$$c[1,4] = \min_{1 \le k < 4} \begin{cases} k=1 & c[1,1] + c[2,4] + d_0 \times d_1 \times d_4 \\ k=2 & c[1,2] + c[3,4] + d_0 \times d_2 \times d_4 \\ k=3 & c[1,3] + c[4,4] + d_0 \times d_3 \times d_4 \end{cases}$$

$$= \min \begin{cases} 0 + 1320 + 4 \times 10 \times 20 \\ 1200 + 720 + 4 \times 3 \times 20 \\ 264 + 0 + 4 \times 12 \times 20 \end{cases}$$

$$= \min (2120 , 1080 , 1224)$$

$$= 1080 .$$

c	1	2	3	4
1	0	120	264	1080
2		0	360	1320
3			0	720
4				0

(i ↓, j →)

k	1	2	3	4
1		1	2	2
2			2	2
3				3
4				

(i ↓, j →)

Paranthesization:

$A_1 \quad A_2 \quad A_3 \quad A_4$.

$k[1,4] = 2$, so

$(A_1 \times A_2) \times (A_3 \times A_4)$

$k[1,2] = 1$ and $k[3,4] = 3$.

$$\boxed{((A_1 \times A_2) \times (A_3 \times A_4))}$$

Travelling Salesman Problem

Introduction:- Travelling sales man problem can be solved by using 3 techniques.

1) Brute force Algorithm.
2) Dyanamic Programming
3) Branch and bound.

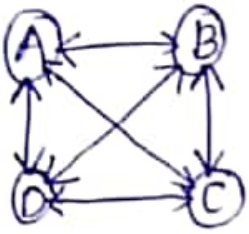
Now, in this we will solve the problem using Dyanamic Programming method.

Travelling Salesman Problem (TSP):- Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Difference b/w Hamiltonean cycle and TSP:-

The Hamiltonean cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamilton tour exists in TSP because it's a complete graph and infact many such tours exists and the problem is to find minimum weight hamilton cycle.

Let us take an example and the given adjacent matrix is



Adjacency matrix

	A	B	C	D
A	0	16	11	6
B	8	0	13	16
C	4	7	0	9
D	5	12	2	0

Let the given set of variables (vertices) be $\{A, B, C, D\}$.

Let us consider 'A' as starting and ending point of output for every other vertex j other than starting vertex, We find the minimum cost path with 'A' as starting point, j as ending point and all vertices appearing exactly once.

Let the cost of this path be cost (j) the cost of corresponding cycle would be cost(j) + dist(j, A) where dist(j, A) is the distance from j to A. Finally we return the minimum of all (cost(j) + dist(j, A)) values.

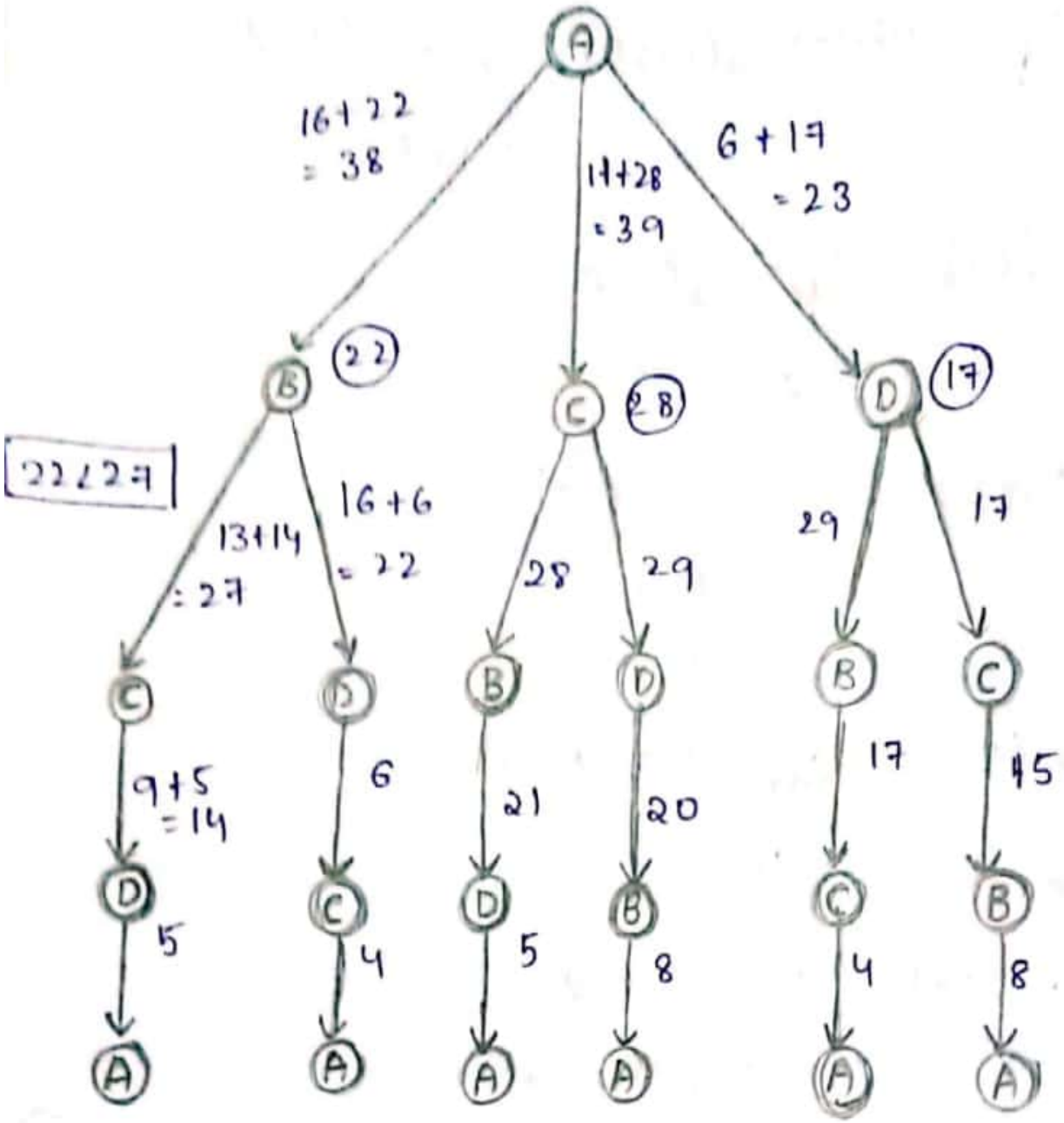
Formulae:-

$$g(i, s) = \min [w(i, j) + g(j, (s - j)]$$

where i is the starting vertex.,

j is another vertex. $\in S$

s is set of vertices. except starting vertex.



Thus from the above diagram, we obtain $g(A, \phi)$ means from D there is no otherway to move to any other vertex except starting vertex. Now, from the adjacency matrix we have

$g(B, \phi) = 8$

$g(C, \phi) = 4$

$g(D, \phi) = 5$

Now, we obtain $g(C,\{D\}) = C_{cd} + g(D,\phi)$

$= 9 + 5 = 14$

$g(D,\{C\}) = C_{dc} + g(C,\phi)$

$= 9 + 4 = 6$

$g(B,\{D\}) = C_{bd} + g(D,\phi)$

$= 16 + 5$

$= 21$

$g(D,\{B\}) = C_{db} + g(B,\phi)$

$= 12 + 8 = 20$

$g(B,\{C\}) = C_{bc} + g(C,\phi)$

$= 13 + 4 = 17$

$g(C,\{B\}) = C_{cb} + g(B,\phi)$

$= 7 + 8 = 15$

Next we compute $g(i,S)$ with $mod S = 2$

$g(B,\{C,D\}) = \min(C_{BC} + g(C,\{D\}), C_{BD} + g(D,\{C\}))$

$= \min(13+14, 16+6)$

$= \min(27, 22)$

$\boxed{g(B,\{C,D\}) = 22}$

$g(C, \{B,D\}) = \min(C_{CB} + g(B,\{D\}), C_{CD} + g(D,\{B\}))$

$= \min(7+21, 9+20)$

$= \min(28, 29)$

$\boxed{g(C,\{B,D\}) = 28}$

$g(D, \{B,C\}) = \min(C_{DB} + g(B,\{C\}), C_{DC} + g(C,\{B\}))$

$= \min(12+17, 2+15)$

$= \min(29, 17)$

$\boxed{g(D,\{B,C\}) = 17}$

Finally from above we obtain g(

$g(A,\{B,C,D\}) = \min(C_{AB} + g(B,\{C,D\}), C_{AC} + g(C,\{B,D\}$

$C_{AD} + g(D,\{B,C\}))$

$= \min(16+22, 11+28, 6+17)$

$= \min(38, 39, 23)$

$\boxed{g(A,\{B,C,D\}) = 23}$

The minimum cost for Hamiltonian cycle is 23.

All pair shortest path

→ In all pair shortest path, when a weighted graph is represented by its weight matrix W then objective is to find the distance between every pair of nodes.

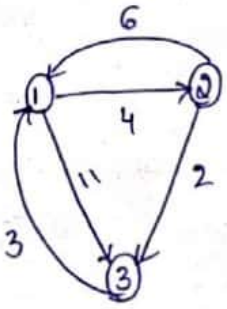
→ We will apply dynamic programming to solve all pairs shortest path.

→ In all pair shortest path algorithm, we first decompose the given problem into sub problems.

→ In this principle of optimally is used for solving the problem.

→ It means any sub path of shortest path is a shortest path between the end nodes.

Example: Compute all pair shortest path for following figure.



Sol: From the above directed graph we will write the Adjacency matrix by considering the vertices and weighted edges.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

In the matrix A^0 all diagnoal elements are zero because there is no self loops in the given graph.

The i^{th} row and j^{th} column should be the weight of the path from each vertix. Suppose if we take 1^{st} row and 2^{nd} column in the graph there exist path between those vertices and the weight is 4.

If there is no path between the vertices directly we represent that weight as ∞.

Similarly we have to write all the weights.

Now taking 1 as an intermediate vertex. we will write adjacency matrix, using A^0 matrix.

For this we have a formula i.e.,

$$A^k[i,j] = \min \{ A^{k-1}[i,j] , A^{k-1}[i,k] + A^{k-1}[k,j] \}$$

$$A' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

As we are taking 1st vertex as an intermediate vertex we should not keep 1st row and 1st column unchange.

Now $A'[2,3]$

$$A'[2,3] = \min\{ A^{\phi-1}[2,3],\ A^{1-1}[2,1] + A^{1-1}[1,3] \}$$

$$= \min\{ A^{0}[2,3],\ A^{0}[2,1] + A^{0}[1,3] \}$$

$$= \min\{ 2,\ 6+11 \}$$

$$= \min\{ 2,\ 17 \}$$

$$= 2$$

$$A'[3,2] = \min\{ A^{1-1}[3,2],\ A^{1-1}[3,1] + A^{1-1}[1,2] \}$$

$$= \min\{ A^{0}[3,2],\ A^{0}[3,1] + A^{0}[1,2] \}$$

$$= \min\{ \infty,\ 3+4 \}$$

$$= \min\{ \infty,\ 7 \}$$

$$= 7$$

Now taking 2 as an intermediate vertex, we will write adjacency matrix using A' matrix.

Scanned with CamScanner

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

we should consider 2nd row and 2nd column as unchange, because, we are considering 2nd vertex as intermediate vertex.

Now $A^2[1,3]$

$$A^2[1,3] = \min\{A^{2-1}[1,3], A^{2-1}[1,2] + A^{2-1}[2,3]\}$$
$$= \min\{A^1[1,3], A^1[1,2] + A^1[2,3]\}$$
$$= \min\{11, 4+2\}$$
$$= \min\{11, 6\}$$
$$= 6$$

$$A^2[3,1] = \min\{A^{2-1}[3,1], A^{2-1}[3,2] + A^{2-1}[2,1]\}$$
$$= \min\{A^1[3,1], A^1[3,2] + A^1[2,1]\}$$
$$= \min\{3, 7+6\}$$
$$= \min\{3, 13\}$$
$$= 3$$

Similarly to considering 3 as an intermediate vertex write adjacent matrix. using A^2 matrix.

Scanned with CamScanner

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

3rd row and 3rd column as unchange because we are considering 3rd matrix vertex as intermediate vertex.

$$A^3[1,2] = \min\{ A^{3-1}[1,2],\ A^{3-1}[1,3] + A^{3-1}[3,2] \}$$

$$= \min\{ A^2[1,2],\ A^2[1,3] + A^2[3,2] \}$$

$$= \min\{ 4,\ 6+7 \}$$

$$= \min\{ 4, 13 \}$$

$$= 4$$

$$A^3[2,1] = \min\{ A^{3-1}[2,1],\ A^{3-1}[2,3] + A^{3-1}[3,1] \}$$

$$= \min\{ A^2[2,1],\ A^2[2,3] + A^2[3,1] \}$$

$$= \min\{ 6,\ 2+3 \}$$

$$= \min\{ 6, 5 \}$$

$$= 5$$

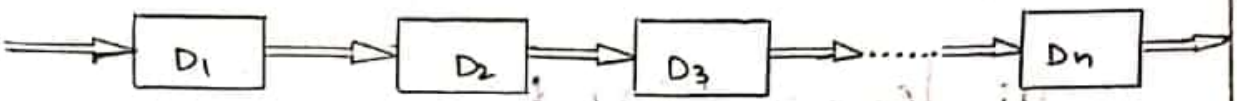
Algorithm:

```
for k=1 to n do
    for i=1 to n do
        for j=1 to n do
            A^k[i,j] = min{A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j]}
```

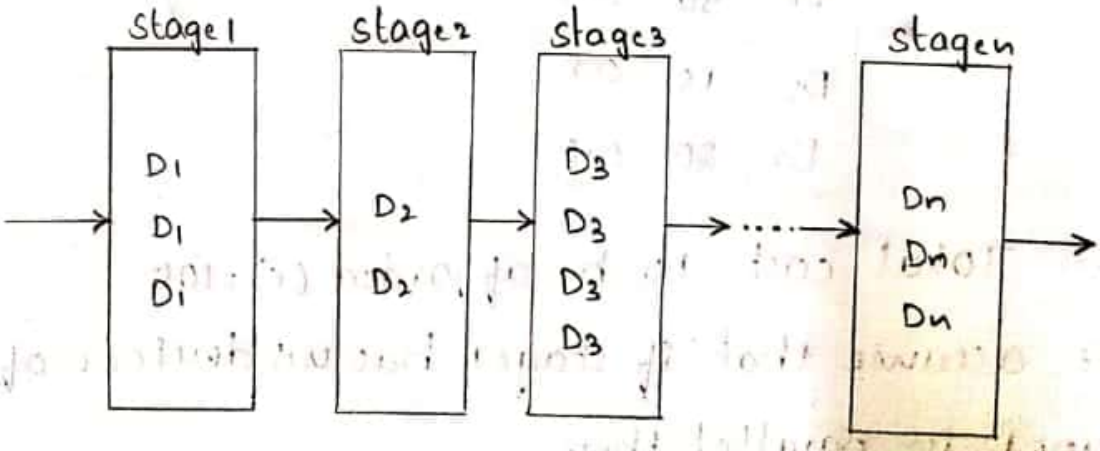

Scanned with CamScanner

Reliability Design:-

In reliability design, the problem is to design a system that is composed of several devices connected in series.



If we duplicate the devices at each stage then the reliability of the system can be increased. Multiple copies of the same device type are connected in parallel through the use of switching circuits. The switching circuits determine which devices in any given group are functioning properly. They then make use of one such device at each stage.



Multiple devices connected in parallel in each stage.

A dynamic programming solution can be obtained in a manner similar to that used for the knapsack problem. since, we can assume each $c_i > 0$, each m_i must be the range $1 \le m_i \le u_i$, where

$$u_i = \left[(c + c_i - \sum_1^n c_j) / c_i \right]$$

problem:-

Let us design a three stage system with devices D_1, D_2, D_3. the costs are Rs.30, Rs.15 and Rs.20 respectively. the cost constraint to the system is Rs.105. Reliability of devices are 0.9, 0.8 & 0.5.

Tabular form:-

Di	Ci	ri
D_1	30	0.9
D_2	15	0.8
D_3	20	0.5

Total cost to be afforded (c) = 105

we assume that if stage i has m_i devices of type i in parallel then

$$\phi(m_i) = 1 - (1 - r_i)^{m_i}.$$

In a general form:

$$\text{reliability} = 1 - (1-r)^{\text{no. of devices}}$$

we assume $c_i > 0$, each m_i must be range

$$1 \le m_i \le u_i$$

$$u_i = \lceil (c + c_i - \sum c_j) / c_i \rceil$$

using above formula, compute u_1, u_2, u_3 for devices D_1, D_2, D_3

$$u_1 = \left\lceil \frac{105+30-(30+15+20)}{30} \right\rceil = \frac{70}{30} = 2$$

$$u_2 = \left\lceil \frac{105+15-(30+15+20)}{15} \right\rceil = \frac{55}{15} = 3$$

$$u_3 = \left\lceil \frac{105+20-(30+15+20)}{20} \right\rceil = \frac{60}{20} = 3$$

Tabular form:

D_i	c_i	r_i	u_i
D_1	30	0.9	2
D_2	15	0.8	3
D_3	20	0.5	3

Maximum No. of extra copies obtained for each device

$$D_1 = \left| \frac{C - \Sigma C_i}{C_1} \right| = \frac{40}{30} = 1$$

total = 1 original copy + 1 extra copy

= 2 copies.

$$D_2 = \left| \frac{C - \Sigma C_i}{C_i} \right| = \frac{40}{15} = 2$$

total = 1 + 2 = 3

$$D_3 = \left| \frac{C - \Sigma C_i}{C_1} \right| = \frac{40}{20} = 2$$

total = 1 + 2 = 3

$S_{copy}^{device} \Rightarrow (r, c)$

for each copy, reliability is multiplied,

$r_t = r_1 \times r_2 \times r_3$, cost is added, $c_t = c_1 + c_2 + c_3$

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

Device 1:

for devices having 1 copy

reliability: $1 - (1 - r)^1$

$= 1-(1-0.9)^1$

$= 1-0.1$

$= 0.9$

cost = 30

$S_1' = \{(0.9, 30)\}$

for device 1, having 2 copies

reliability $= 1-(1-r)^2$

$= 1-(1-0.9)^2$

$= 1-0.01$

$= 0.99$

cost $= 30+30 = 60$

$S_2' = \{(0.99, 60)\}$

$S' = \{(0.9, 30), (0.99, 60)\}$

Dominance Rule:-

If S' contains two pairs (f_1, x_1) and (f_2, x_2) with the property that $f_1 \geq f_2$ and $x_1 \leq x_2$, then (f_1, x_1) dominates (f_2, x_2) can be discarded. Discarding & pruning rules such as the one above is known as dominance rule. Dominating tuples will be present in S^1 and dominated tuples has to be discarded from S'.

case1 : If $f_1 \leq f_2$ and $x_1 > x_2$ then discard (f_1, x_1)

case2: If $f_1 \geq f_2$ and $x_1 < x_2$ the discard (f_2, x_2)

case3: otherwise simply write (f_1, x_1)

Reliability (1) $= 1 - (1-r_1)^{m_1} = 1 - (1-0.5)^1$

$= 1 - 0.5 = 0.5$

cost = 20

$S_1^3 = \{(0.5\,(0.72), 45+20), (0.5\,(0.864), 60+20), (0.5\,(0.8928), 75+20)\}$

$S_1^3 = \{(0.36, 65), (0.437, 80), (0.4464, 95)\}$

Reliability (2) $= 1 - (1-0.5)^2$

$= 0.75$

cost $= 20+20 = 40$

$S_2^3 = \{(0.75\,(0.72), 45+20+20), (0.75\,(0.864), 60+20+20),$

$(0.75\,(0.8928), 75+20+20)\}$

$= \{(0.54, 85), (0.648, 100), (0.6696, 115)\}$

Reliability (3) $= 1 - (1-0.5)^3 = 0.875$

cost $= 20+20+20$

$S_3^3 = \{(0.875\,(0.72), 45+20+20+20), (0.875\,(0.864), 60+20+20+20)$

$(0.875\,(0.8928), 75+20+20+20)\}$

$S_3^3 = \{(0.63, 105), (1.756, 120), (0.7812, 135)\}$

If cost exceeds 105, remove that tuples.

$S^3 = \{(0.36,65),(0.439,80),(0.54,85),(0.648,100)\}$

Devices	No.of copies
D_1	1
D_2	2
D_3	2

final result: (0.64,100)-from s_3 which is taken from s_3^2

So we take 2 copies of devices 3

$2 \times 20 = 40$

$100 - 40 = 60$

$C = 60$

Cost 60 from s_3 - (0.864, 60) is taken from s_2^2 so we need 2 copies of devices

$2 \times 15 = 30$

$60 - 30 = 30$

$C = 30$

Cost 30 from $s_1 = (0.9, 30)$ is taken from Si so we need 1 copy of devices

$1 \times 30 = 30$.

$30 - 30 = 0$

∴ The best design has a reliability of 0.648 and a cost of 100.