Dynamic Programming: is an algorithm design method that can be used when the solution to a problem may be viewed as the result of a sequence of decisions.

- One way to solve problems for which it is not possible to make a sequence of stepwise decisions leading to an optimal decision sequence is to try out all possible decision sequences.
- We could enumerate all decision sequences and then pick out the best.
- In dynamic programming an optimal sequence of decisions is arrived at by making explicit appeal to the Principle of Optimality.
- This principle states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.
- **Dynamic programming is typically applied to optimization problems. In such problem there can be many solutions. Each solution has a value, and we wish to find a solution with the optimal value.**

Principle of Optimality

- **Definition: A problem is said to satisfy the Principle of Optimality if the subsolutions of an optimal solution of the problem are themesleves optimal solutions for their subproblems.**
- **Examples:**
  - **The shortest path problem satisfies the Principle of Optimality.**
  - **This is because if a,x1,x2,...,xn,b is a shortest path from node a to node b in a graph, then the portion of xi to xj on that path is a shortest path from xi to xj**

Forward approach and backward approach

Note that if the recurrence relations are formulated using the forward approach then the relations are solved backwards. i.e., beginning with the last decision

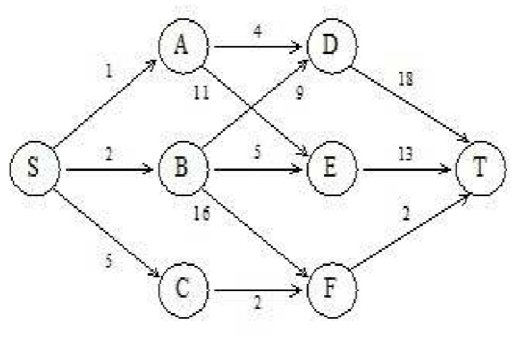
On the other hand if the relations are formulated using the backward approach, they are solved forwards.

To solve a problem by using dynamic programming

- Find out the recurrence relations.
- Represent the problem by a multistage graph.

Multistage Graphs

Multistage graph problem is to determine shortest path from source to destination. A multistage graph G=(V,E) is a directed graph in which the vertices are partitioned into k>=2 disjoint sets Vi, 1<=i<=k.

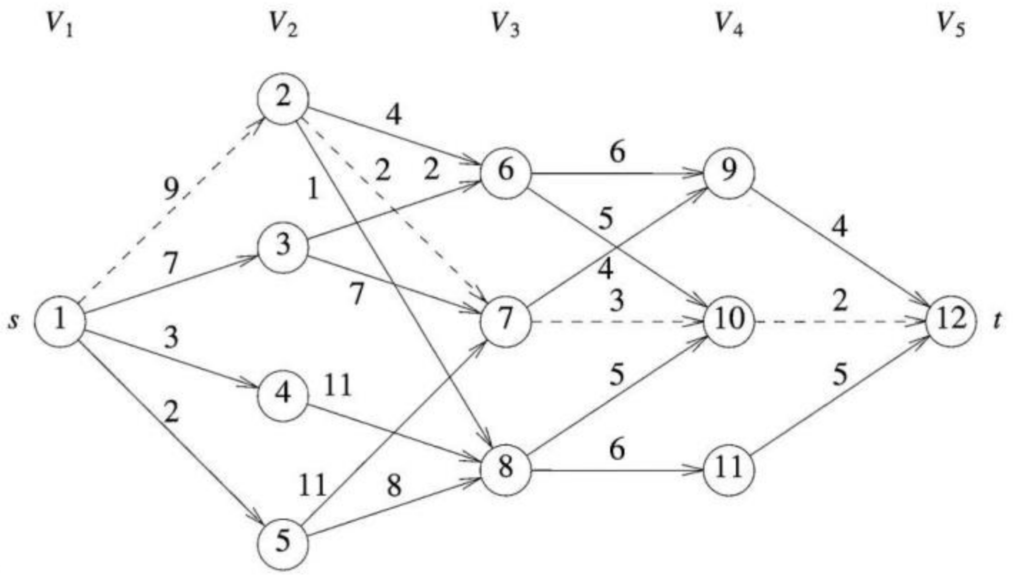


The vertex s is source and t is the sink. Let c(i,j) be the cost of edge <i, j>. The cost of a path from s to t is the sum of costs of the edges on the path. The multistage graph problem is to find a minimum-cost path from s to t.

A dynamic programming formulation for a k-stage graph problem is obtained by first noticing that every s to t path is the result of a sequence of k-2 decisions.

The ith decision involve determining which vertex in Vi+1, 1<=i<=k-2, is on the path. It is easy to see that principal of optimality holds.
Let p(i,j) be a minimum-cost path from vertex j in Vi to vertex t. Let cost(i,j) be the cost of this path.

$$cost(i,j) = \min_{\substack{l \in V_{i+1} \\ \langle j,l \rangle \in E}} \{c(j,l) + cost(i+1,l)\}$$



Cost(5,12)=0;

• Cost(4,9)=4+ Cost(5,12)=4;

• Cost(4,10)=2+ Cost(5,12)=2;

• Cost(4,11)=5+ Cost(5,12)=5;

- Cost(3,6)=min{6+cost(4,9), 5+cost(4,10)}=min{10,7}=7
- Cost(3,7)=min{4+cost(4,9), 3+cost(4,10)}=min{8,5}=5
- Cost(3,8)=min{5+cost(4,10),6+cost(4,11)}=min{7,11}=7
- Cost(2,2)=min{4+cost(3,6),2+cost(3,7),1+cost(3,8)}=min{11,7,8}=7
- Cost(2,3)=min{2+cost(3,6),7+cost(3,7)}=min{9,12}=9
- Cost(2,4)=11+cost(3,8)=18
- Cost(2,5)=min{11+cost(3,7), 8+cost(3,8)}=min{16,15}=15
- Cost(1,1)=min{9+cost(2,2),7+cost(2,3),3+cost(2,4),2+cost(2,5}

=min{16,16,21,17}=16

Shortest path 1-2-7-10-12

All Pairs Shortest Path problem

1. 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.
2. We will apply dynamic programming to solve the all pairs shortest path.
3. In all pair shortest path algorithm, we first decomposed the given problem into sub problems.
4. In this principle of optimally is used for solving the problem.
5. It means any sub path of shortest path is a shortest path between the end nodes.

Steps

i. Let $A^k_{i,j}$ be the length of shortest path from node i to node j such that the label for every intermediate node will be ≤ k.

ii. Now, divide the path from i node to j node for every intermediate node, say 'k' then there arises two case.

a. Path going from i to j via k. b. Path which is not going via k.

iii. Select only shortest path from two cases.

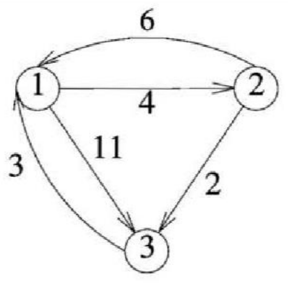
iv. Using recursive method we compute shortest path.

v. Initially: $A^0 = W[i, j]$

vi. Next computations: $A^k_{i,j} = min A^{k-1}_{i,j}, A^{k-1}_{i,k}, A^{k-1}_{k,j}$

Problem-

Consider the following directed weighted graph-



(a) Example digraph

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

(b) A^0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

(c) A^1

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

(d) A^2

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

(e) A^3

Calculations

At d=1

$d^1{}_{(1,1)} = \min\{d^0 (1,1), d^0 (1,1,)+ d^0 (1,1,)\} = 0$

$d^1{}_{(1,2)} = \min\{d^0 (1,2), d^0 (1,1)+d^0 (1,2)\} = \min\{4, 0+4\}=4$

$d^1{}_{(1,3)} = \min \{d^0 (1,3), d^0 (1,1)+d^0 (1,3)\} = \min\{11,0+11\} =11$

$d^1{}_{(2,1)} = \min\{d^0 (2,1), d^0 (2,1)+ d^0 (1,1)\}$

$d^1{}_{(2,2)} = \min\{d^0 (2,2), d^0 (2,1)+d^0 (1,2)\} = 0$

$d^1{}_{(2,3)} = \min\{d^0 (2,3), d^0 (2,1)+d^0 (1,3)\}= \min\{2,6+11\}=2$

$d^1{}_{(3,1)} = \min\{d^0 (3,1), d^0 (3,1) + d^0 (1,1)\} = \min\{3, 3+0\} = 3$

$d^1{}_{(3,2)} = \min\{d^0 (3,2), d^0 (3,1)+d^0 (1,2)\}= \min \{\infty, 3+4\}=7$

$d1_{(3,3)} = 0$

At d=2

$d^2{}_{(1,1)} = \min\{d^1 (1,1), d^1 (1,2)+ d^1 (2,1,)\} = \min\{ 0, \infty\} = 0$

$d^2{}_{(1,2)} = \min\{d^1 (1,2), d^1 (1,2)+d^1 (2,2)\} =4$

$d^2{}_{(1,3)} = \min \{d^1 (1,3), d^1 (1,2)+d^1 (2,3)\} = 6$

$d^2{}_{(2,1)} = \min\{d^1 (2,1), d^1 (2,2)+ d^1 (2,1)\} =6$

$d^2{}_{(2,2)} = \min\{d^1 (2,2), d^1 (2,1)+d^1 (2,2)\} = \min\{0, 0+0\} = 0$

$d^2{}_{(2,3)} = \min\{d^1 (2,3), d^1 (2,1)+d^1(2,3)\}= 2$

$d^2{}_{(3,1)} = \min\{d^1 (3,1), d^1 (3,2) + d^1 (2,1)\} = \min\{3, 7+6\} = 3$

$d^2{}_{(3,2)} = \min \{7, 7+0\}=7$

$d2_{(3,3)} = 0$

At d=3

$d^3_{(1,1)} =$ min{ 0, somevalue} = 0

$d^3_{(1,2)} =$ min{4,6+0}=4

$d^3_{(1,3)}$ = min {6,6+0)} = 6

$d^3_{(2,1)}$ = min{6, 2+3}=5

$d^3_{(2,2)}$ = 0

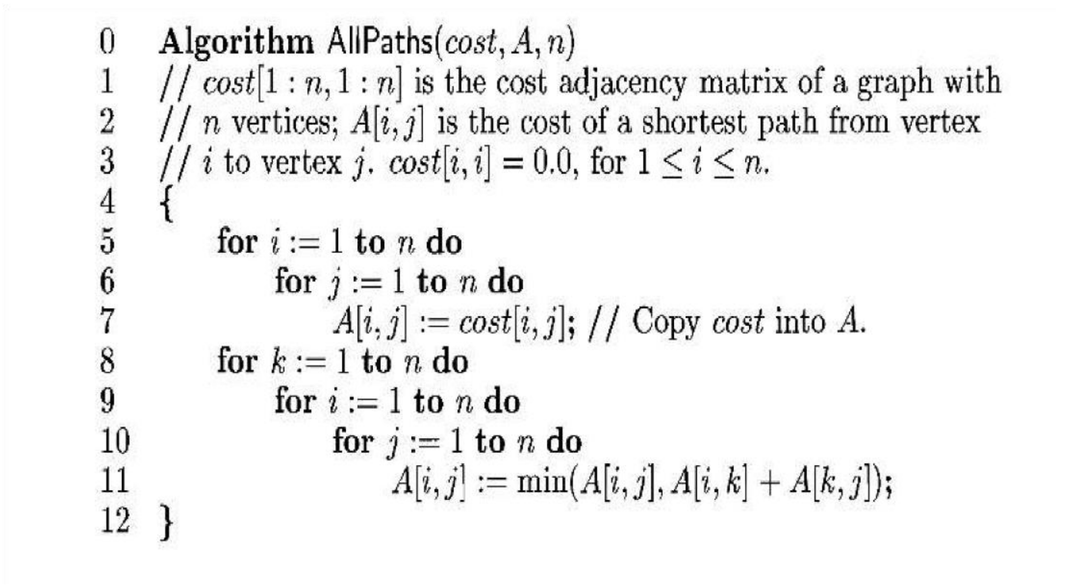
$d^3_{(2,3)}$ = min{2,2}=2

$d^3_{(3,1)}$ = min{3, 3+0} = 3

$d^3_{(3,2)}$ = min {7, 7+0}=7

$d^3_{(3,3)} =$ 0

A^3 gives all pairs shortest path



```
Algorithm AllPaths($cost, A, n$)
// $cost[1:n, 1:n]$ is the cost adjacency matrix of a graph with
// $n$ vertices; $A[i,j]$ is the cost of a shortest path from vertex
// $i$ to vertex $j$. $cost[i,i] = 0.0$, for $1 \le i \le n$.
{
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            $A[i,j] := cost[i,j]$; // Copy $cost$ into $A$.
    for $k := 1$ to $n$ do
        for $i := 1$ to $n$ do
            for $j := 1$ to $n$ do
                $A[i,j] := \min(A[i,j], A[i,k] + A[k,j])$;
}
```

Travelling salesman Problem

In the TSP, given a set of cities and the distance between each pair of cities, a salesman needs to choose the shortest path to visit every city exactly once and return to the city from where he started.

Step 1: Let d[i, j] indicates the distance between cities i and j. Function C[x, V – { x }]is the cost of the path starting from city x. V is the set of cities/vertices in given graph. The aim of TSP is to minimize the cost function.

Step 2: Assume that graph contains n vertices V1, V2, ..., Vn. TSP finds a path covering all vertices exactly once, and the same time it tries to minimize the overall traveling distance.

Step 3: Mathematical formula to find minimum distance is stated below:

$C(i, V) = \min \{ d[i, j] + C(j, V - \{ j \}) \}, j \in V$ and $i \notin V$.

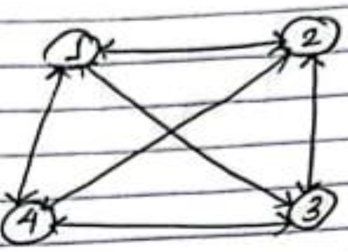
TSP problem possesses the principle of optimality, i.e. for d[V1, Vn] to be minimum, any intermediate path (Vi, Vj) must be minimum.

Complexity Analysis of Traveling salesman problem

Dynamic programming creates $n.2^n$ subproblems for n cities. Each sub-problem can be solved in linear time. Thus the time complexity of TSP using dynamic programming would be $O(n^2 2^n)$. It is much less than n! but still, it is an exponent. Space complexity is also exponential.

Travelling Salesman problem:

$$A = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 5 & 0 & 9 & 10 \\ 3 & 6 & 13 & 0 & 12 \\ 4 & 8 & 8 & 9 & 0 \end{array}$$



$g(i, s) = \min\limits_{k \in s} (C_{ik} + g(k, s - \{k\}))$

$g(i,s)$ Cost of starting vertex i to set of vertox s.

C_{ik}: cost of starting vertex i to any vertex (k)

$g(k, s-\{k\})$: Cost of vertex k to other remaining vertex.

$g(2,\phi)$ = cost of vertex 2 to 1 itself.
$= 5$

$g(3,\phi) = 6$

$g(4,\phi) = 8$

$g(2, \{3\}) =$ ~~$\min\limits_{2 \in S} (C_{12} + g(2,\{3\})$~~

$= C_{23} + g(3,\phi)$

$= 15$

~~$g(3, \{2\}$~~

$g(2, \{4\}) = C_{24} + g(4,\phi)$

$= 10 + 8 = 18$

$g(3, \{2\}) = 18$, $g(3, \{4\}) = 20$

$g(4, \{2\} = 13$, $g(4, \{3\}) = 15$

Now,

$g(2, \{3, 4\}) = \min\{c_{23} + g(3, \{4\}),$
$c_{24} + g(4, \{3\})\} = 25$

$g(3, \{2, 4\}) = \min\{c_{32} + g(2, \{4\}), c_{34} + g(4, \{2\})\} = 25$

$g(4, \{2, 3\}) = \min\{c_{42} + g(2, \{3\}), c_{43} + g(3, \{2\})\}$
$= 23.$

At last,

$g(1, \{2, 3, 4\}) = \min\{c_{12} + g(2, \{3, 4\}), c_{13}$
$+ g(3, \{2, 4\}), c_{14} + g(4, \{2, 3\})\}$

$= \min(35, 40, 43)$
$= 35$ //

Reliability-design

problem:- To setup a system that consists of devices $d_1, d_2, d_3 \ldots d_n$ and each devices has a cost and reliability

i.e.	D_1	D_2	D_3	D_4
	c_1	c_2	c_3	c_4
	.9	.9	.9	.9.

∴ Then reliability of system $= \pi r_i$
$= 0.9^4$
$= 0.6561.$

∴ So our task is to optimize reliability.

e.g.

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

$C = 105$

$\Sigma C_i = C_1 + C_2 + C_3$
$= 30 + 15 + 20 = 65$. (Taking one Copy of each device)

Now remaining Cost $C - \Sigma C_i = 105 - 65 = 90$

$\therefore u_i$ (upper bound) $= \left\lfloor \dfrac{C - \Sigma C_i}{C_i} \right\rfloor + 1$

<u>Set method:</u>
(R, C)

$= \frac{40}{30} + 1 = 2$

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

Consider D_1,

$S_1^1 = \{(0.9, 30)\}$ ($S_1^1 \to$ first copy first device)

1 ← device
Copy ↑

$S_2^1 = (0.99, 60)$

$1 - (1 - 0.9)^2$
$= 1 - (0.1)^2$
$= 1 - 0.01$
$= 0.99$

(II) $S^1 = \{(0.9, 30)(0.99, 60)\}$

Consider θ_2:

$$S_1^2 = \{(\overset{\overset{0.9\times0.8}{\swarrow}}{0.72}, 45), (\overset{\overset{0.99\times0.8}{\swarrow}}{0.792}, 75)\}$$

$$S_2^2 = \{(0.864, 60), \underset{\times}{\underline{(0.9504, 90)}}\}$$

↑ Infeasible because we can't use third device (90+20 des $C=105$)

$$S_3^2 = \{(0.8928, 75), (-,-)\}$$

$1-(1-r_2)^3$ ← device.
$= 1-(1-0.8)^3$
$= 1-0.2^3$
$= 1-0.008$
$= 0.992$

$$\therefore S^2 = \{(0.72, 45), \cancel{(0.792, 75)}, (0.864, 60), (0.8928, 75)\}$$

→ [Cut this one because reliability increased, cost increase, & in next set, reliability increased, but cost decreased]

At θ_3: $S_1^3 = \{(0.36, 65), (0.432, 80), (0.4464, 95)\}$

$S_2^3 = \{(0.54, 85), (0.648, 100), (-, 125)\}$ ↑ cost exceed.

$S_3^3 = \{(0.63, 105), \underset{\times}{(-, 120)}, \underset{\times}{(-, 135)}\}$

$\therefore S^3 = \{(0.36, 65), (0.432, 80), (0.4464, 95), (0.54, 85), (0.648, 100), (0.63, \cancel{1}05)\}$

$\therefore$ Reliability $= 0.648$
cost $= 100$.