DYNAMIC PROGRAMMING

- The General Method
- Multi stage Graphs
- All pairs shortest paths
- Optimal binary search tree
- Reliability Design
- 0/1 Knapsack Problem
- The Traveling Salesman Problem

DYNAMIC PROGRAMMING

THE GENERAL METHOD

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

Principle of optimality:

The principle of optimality 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.

1. Dynamic Programming is technique for solving problems with overlapping subproblems. ▪ In this method each sub problem is solved only once. The result of each subproblem is recorded in a table from which we can obtain a solution to the original problem. ▪ In Dynamic computing duplications in solution is avoided totally. ▪ Efficient then Divide and conquer strategy. ▪ Uses bottom up approach of problem solving.

2. Dynamic Programming ▪
   Dynamic Programming is an algorithm design technique for optimization problems: often minimizing or maximizing. ▪ Like divide and conquer, DP solves problems by combining solutions to subproblems. ▪ Unlike divide and conquer, subproblems are not independent. ▪ Subproblems may share other subproblems ▪ However, solution to one subproblem may not affect the solutions to other subproblems of the same problem.

3. Principle of Optimality ▪ Obtains the solution using principle of optimality. ▪ In an optimal sequence of decisions or choices, each subsequence must also be optimal. When it is not possible to apply the principle of optimality it is almost impossible to obtain the solution using the dynamic programming approach.
   ▪ Example: Finding of shortest path in a given graph uses the principle of optimality.

- Dynamic programming is applicable when the sub-problems are dependent.
- For generating decision sequence we use the principle of optimality i.e., output of stage-1 will be given as input stage-2, output of stage-2 will be given as input for stage-3 and so on.
- Initial conditions are given as input for stage-1.
- In greedy method only one decision sequence is generated but in dynamic programming many decision sequences may be generated.

DYNAMIC PROGRAMMING

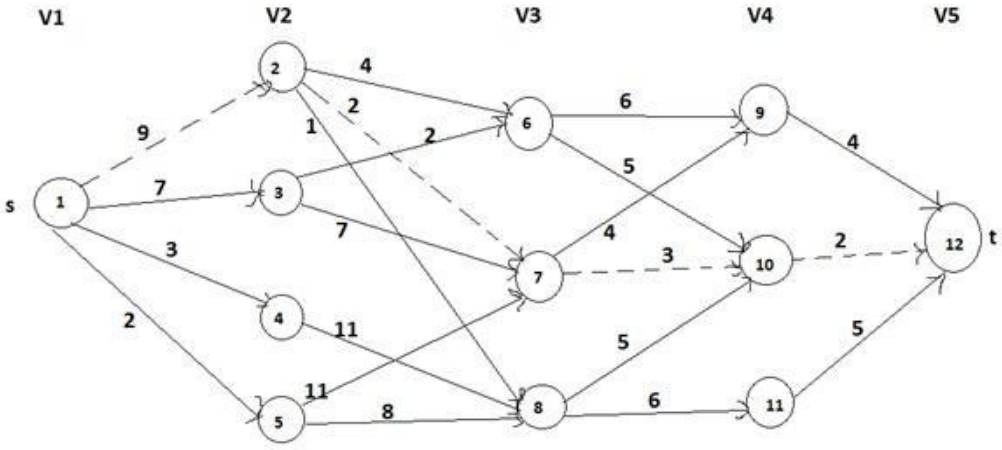
MULTI STAGE GRAPHS

- A multi stage graph G = (V, E) is a directed graph in which the vertices are partitioned into k ≥ 2 disjoint sets V_i $1 \le i \le k$.
- In addition, if (u, v) is an edge in E, then u ∈ V_i and v ∈ V_{i+1} for some $1 \le i < k$. The sets V1 and Vk are such that |V1|=1 and |Vk|=1.
- Let s and t be the vertices in V1 and Vk respectively. The vertex 's' is the source vertex and 't' is the sink vertex.
- Let C(i, j) be the cost of edge (i, j). The cost of the path from s to t is the sum of the costs of the edges on the path. The multi stage graph problem is to find the 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. It is easy to see that principle of optimality holds.

Example:

Find a minimum cost path from s to t in the given multi stage graph.



MULTI STAGE GRAPH

DYNAMIC PROGRAMMING

we solve the problem using two approaches

i. Forward approach
ii. Backward approach

Forward approach:

cost(i, j) = min { c(j, l) + cost(i+1, l) }
$l \in V_{i+1}, (j, l) \in E$

cost(1, 1) = min { 9+cost(2, 2), 7+cost(2, 3), 3+cost(2, 4), 2+cost(2, 5)}
= min { 9+7, 7+9, 3+18, 2+15 } = 16

cost(2, 2) = min { 4+cost(3, 6), 2+cost(3, 7), 1+cost(3, 8) }
= min { 4+7, 2+5, 1+7} = min {11, 7, 8} = 7

cost(2, 3) = min { 2+cost(3, 6), 7+cost(3, 7)} = min {2+7, 7+5}
= min { 9, 12} = 9

cost(2, 4) = min { 11+cost(3, 8)} = min { 11+7} = 18

cost(2, 5) = min { 11+cost(3, 7), 8+cost(3, 8)}
= min { 11+5, 8+7 } = min { 16, 15} = 15

cost(3, 6) = min { 6+cost(4, 9), 5+cost(4, 10)}
= min { 6+4, 5+2 } = min {10, 7 } = 7

cost(3, 7) = min { 4+cost(4, 9), 3+cost(4, 10)}
= min { 4+4, 3+2 } = min {8, 5} = 5

cost(3, 8) = min { 5+cost(4, 10), 6+cost(4, 11)}
= min { 5+2, 6+5 } = min {7, 11} = 7

cost(4, 9) = 4 cost(4, 10) = 2 cost(4, 11) = 5

For finding path, we obtain

d(i, j) = the value of l(l is a node) that min { c(j, l)+cost(i+1, l)}
d(1, 1) = 2
d(2, 2) = 7 d(2, 3) = 6 d(2, 4) = 8 d(2, 5) = 8
d(3, 6) = 10 d(3, 7) = 10 d(3, 8) = 10

The minimum cost path is s=v1=1, v2, v3, v4,12=v5=t where vi+1 = d(i, vi)

v2 = d(1, v1)=d(1, 1) = 2 v3 = d(2, v2) = d(2,2) = 7 v4 = d(3, v3) = d(3, 7) = 10

Therefore, **the minimum cost path is 1— 2 --- 7 ----10 ---- 12 and cost = 16.**

Algorithm:

```
Algorithm FGraph ( G, k, n, p)
{
        cost [n] = 0;
        for j = n-1 to 1 step -1 do
        {
                Let r be the vertex such that (j, r) is an edge of G and c[ j, r]+cost[r] is
                minimum;
                cost[j] = c[j, r] + cost[r];
                d[j] = r;
        }
        P[1] = 1;
        P[k] = n;
        for j = 2 to k-1 do
                p[j] = d[ p[j-1]];
}
```

Backward approach:

bcost(i, j) = min { c(l, j) + bcost(i-1, l) }
$l \in V_{i-1}, (l, j) \in E$

bcost(5, 12) = min { 4+bcost(4, 9), 2+bcost(4, 10), 5+bcost(4, 11)}
= min { 4+15, 2+14, 5+16} = 16

bcost(4, 9) = min { 6+bcost(3, 6), 4+bcost(3, 7) }
= min { 6+9, 4+11, } = min {15, 15 } = 15

bcost(4, 10) = min { 5+bcost(3, 6), 3+bcost(3, 7), 5+bcost(3, 8)}
= min {5+9, 3+11, 5+10}= min { 14, 14, 15} = 14

bcost(4, 11) = min { 6+bcost(3, 8)} = min { 6+10} = 16

bcost(3, 6) = min { 4+bcost(2, 2), 2+bcost(2, 3)}

= min { 4+9, 2+7 } = min { 13, 9} = 9

bcost(3, 7) = min { 2+bcost(2, 2), 7+bcost(2,3), 11+bcost(2, 5)}
= min { 2+9, 7+7, 11+2 } = min {11, 14, 13 } = 11

bcost(3, 8) = min { 1+bcost(2, 2), 11+bcost(2, 4), 8+bcost(2, 5)}
= min { 1+9, 11+3, 8+2 } = min {10, 14, 10 } = 10

bcost(2, 2) = 9 bcost(2, 3) = 7 bcost(2, 4) = 3 bcost(2, 5) = 2

For finding path, we obtain

d(i, j) = the value of l(l is a node) that min { c(l, j)+cost(i-1, l)}

d(5, 12) = 10
d(4, 9) = 6 or 7 d(4, 10) = 7 d(4, 11) = 8
d(3, 6) = 3 d(3, 7) = 2 d(3, 8) = 2

The minimum cost path is s=v1=1, v2, v3, v4, 12==v5=t where vi = d(i+1, vi+1)
v4 = d(5, v5) =d(5, 12)= 10 v3 = d(4, v4) = d(4, 10) = 7 v2 = d(3, v3) = d(3, 7) = 2

Therefore, **the minimum cost path is 1— 2 --- 7 ----10 ---- 12 and cost = 16.**

Algorithm:

```
Algorithm Bgraph ( G, k, n, p )
{
        bcost[1] = 0;
        for j = 2 to n do
        {
                Let r be such that (r, j) is an edge of G and bcost[j] + c[r, j] is minimum;
                bcost[j] = bcost[r]+c[r, j];
                d[j] = r;
        }
        P[1] = 1;
        P[k] = n;
        for j = k-1 to 2 step -1 do
                p[j] = d[ p[j+1]];
}
```

DYNAMIC PROGRAMMING

ALL PAIRS SHORTEST PATHS

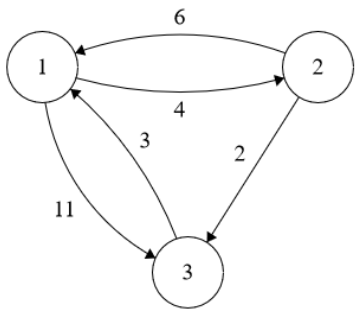
- All pairs shortest paths problem is to find the length of the shortest paths from each vertex to all other vertices.
- We record the length of shortest paths in an nXn matrix 'D' is called the distance matrix.
- We computes the distance matrix of a weighted graph with n vertices through a series of nXn matrices D^0, D^1, D^2, ……, D^n

The elements

d_{ij}^k in the matrix D^k = the length of the shortest path among all paths from the i th vertex to the j th vertex with each intermediate vertex, if any, numbered not higher than k.

therefore, $d_{ij}^k = \min \{ d_{ij}^{k-1} , d_{ik}^{k-1} + d_{kj}^{k-1} \}$ for $k \geq 1$ and $d_{ij}^0 = w_{ij}$

Example:



$$D^0 = \begin{matrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{matrix} \qquad D^1 = \begin{matrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{matrix}$$

$$D^2 = \begin{matrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{matrix} \qquad D^3 = \begin{matrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{matrix}$$

Algorithm Allpaths (cost, D, n)

DYNAMIC PROGRAMMING

```
{
        for i = 1 to n do
                for j = 1 to n do
                        D [i , j] = cost [i , j];
        for k = 1 to n do
                for i = 1 to n do
                        for j = 1 to n do
                                D [i, j] = min { D[i, j], D[i, k] + D[k, j]};
}
```

OPTIMAL BINARY SEARCH TREE(OBST)

DYNAMIC PROGRAMMING

Given a fixed set of identifiers, we wish to create a binary search tree organization. We construct different binary search trees for the same identifier set. If n identifiers in the set then we construct binary search trees are

C(n) = 2nCn 1/(n+1) called Catalan number.

In general, we can expect different identifiers to be searched for with different frequencies (probabilities). In addition, we can expect unsuccessful searches are also to be made.

Let us assume that the given set of identifiers is { a1, a2, ….., an } with a1 < a2 < …..< an. Let P(i) be the probability with which we search for ai. Let q(i) be the probability that the identifier 'x' being searched for is such that ai < x <ai+1.

We add fictitious node in place of every empty sub tree in the binary search tree, such nodes are called external nodes. If a binary search tree represents n identifiers then there will be exactly n internal nodes and n+1 external nodes. Every internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.

The identifiers not in the binary search tree can be partitioned into n+1 equivalence classes Ei, 0 ≤ i ≤ n.

The class E0 contains the identifiers x, x<a1.

The class Ei contains the identifiers x, ai < x < ai+1, 1≤i<n

The class En contains the identifiers x, x>an.

The cost of an internal node ai is p(i) * level(ai)

If the failure for Ei is at level l, then only l-1 iterations required. So the cost of an external node is q(i) * level(Ei-1).

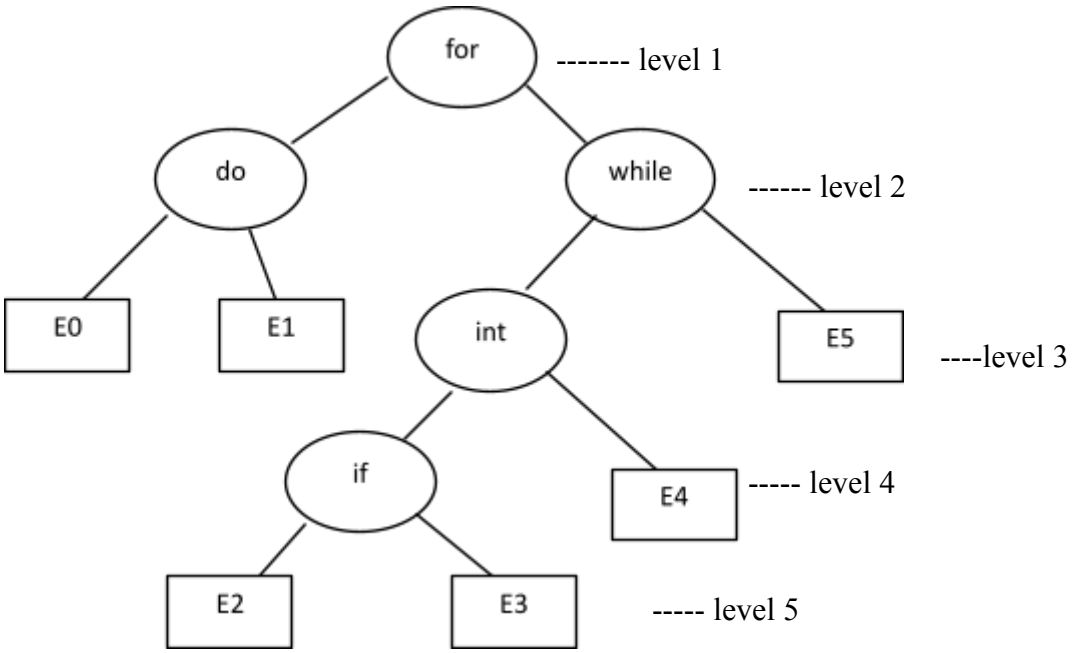
Therefore, the cost of the binary search tree is

$$\sum_{1\le i\le n} p(i) * level(ai) + \sum_{0\le i\le n} q(i) * level(Ei - 1)$$

Example:

DYNAMIC PROGRAMMING

find the cost given binary search tree, {do, for, if, int, while} (p1,p2,p3,p4,p5) = (2/20,1/20,1/20,3/20,1/20) and (q0,q1,q2,q3,q4,q5) = (1/20,2/20,3/20,2/20,3/20,1/20).



$$\text{cost} = \sum_{1 \le i \le 5} p(i) * level(ai) + \sum_{0 \le i \le 5} q(i) * level(Ei - 1)$$

= (2*2/20 + 1*1/20 + 4*1/20 + 3*3/20 + 2*1/20) + (2*1/20 + 2*2/20 + 4*3/20 + 4*2/20 + 3*3/20 + 2*1/20)

= ((4+1+4+9+2)/20) + ((2+4+12+8+9+2)/20) = (20/20) +(37/20) = 57/20 = 2. 85

To apply dynamic programming to the problem of obtaining an optimal binary search tree, we need to view construction of such a tree as the result of a sequence of decisions and then observe that the principle of optimality holds.

A possible approach to this would be to make a decision as to which of the ai's should be assigned to the root node of the tree. if we choose ak, then the internal nodes a1, a2, …, ak-1 and the external nodes for the classes E0, E1, …., Ek-1 will lie on the left sub tree(l), the internal

nodes ak+1, ak+2, …, an and the external nodes for the classes E_k, E_{k+1}, …., E_n will lie on the right sub tree(r) and the root of the tree is a_k.
Therefore the tree is

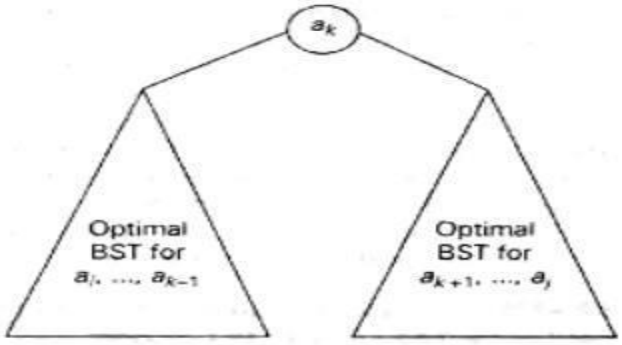


Fig: Binary search tree with root a_k and two optimal binary search subtrees and

We obtain, cost of the tree is

$$= p_k + \sum_{1\le i\le k-1} p(i) * (level(ai) + 1)) + \sum_{0\le i\le k-1} q(i) * (level(Ei) - 1 + 1)) +$$

$$\sum_{k+1\le i\le n} p(i) * (level(ai) + 1)) + \sum_{k\le i\le n} q(i) * (level(Ei) - 1 + 1))$$

$$= p_k + \sum_{1\le i\le k-1} p(i) * level(ai) + \sum_{0\le i\le k-1} q(i) * (level(Ei) - 1) + \sum_{k+1\le i\le n} p(i) * level(ai)) +$$

$$\sum_{k\le i\le n} q(i) * (level(Ei) - 1) + \sum_{1\le i\le k-1} p(i) + \sum_{0\le i\le k-1} q(i) + \sum_{k+1\le i\le n} p(i) + \sum_{k\le i\le n} q(i)$$

But, $$\text{cost(l)} = \sum_{1\le i\le k-1} p(i) * (level(ai)) + \sum_{0\le i\le k-1} q(i) * (level(Ei) - 1)$$

$$\text{cost(r)} = \sum_{k+1\le i\le n} p(i) * (level(ai)) + \sum_{k\le i\le n} q(i) * (level(Ei) - 1)$$

DYNAMIC PROGRAMMING

Let W[i, j] = $\sum_{i+1 \le k \le j} p(k) + \sum_{i \le k \le j} q(k)$

Therefore, the cost of (1) = pk + cost(l) + cost(r) + W[0, k-1] + W[k, n].

Let C[i, j] = the cost of an optimal binary search tree containing the internal nodes ai+1, ai+2, …., aj and the external nodes Ei, Ei+1, …., Ej .

If the binary search tree (1) is an optimal, then by using principle of optimality the left sub tree(l) and the right sub tree(r) are also an optimal binary search trees.
So, cost(l) = C[0, k-1] and cost(r) = C[k, n]
Therefore, cost of (1) = p_k + C[0, k-1] + C[k, n] + W[0, k-1] + W[k, n]

Take every node in a1, a2, …., an as a root and find the cost of optimal binary search tree, which tree gives the minimum value that binary search tree is the optimal binary search tree for the identifiers a1, a2, …., an.

So, C[0, n] = $min_{k=0}^{n}$ { C[0, k-1] + C[k, n] } + pk + W[0, k-1] + W[k, n]

= $min_{k=0}^{n}$ { C[0, k-1] + C[k, n] } + W[0, n]

We can generalized, we obtain,

C[i, j] = $min_{k=i+1}^{j}$ { C[i, k-1] + C[k, j] } + W[i, j]

Initially, C[i, i] = 0 and W[i, i] = q(i).

For solving C[0, n] first compute C[i, j] such that j-i = 1, j-i = 2 and so on.

Example:

Find the optimal binary search tree for the given instances n = 4 (a1, a2, a3, a4) = (count, float, if, while) , p(1, 2, 3, 4) = (2/20, 1/20, 3/20, 3/20) and (q0, q1, q2, q3, q4) = (2/20, 3/20, 2/20, 3/20, 1/20)

Solution:

DYNAMIC PROGRAMMING

$$C[i, j] = min_{k=i+1}^{j} \{ C[i, k-1] + C[k, j] \} + W[i, j]$$

$$W[i, j] = \sum_{k=i+1}^{j} p(k) + \sum_{k=i}^{j} q(k)$$

C[i, i] = 0 and W[i, i] = q(i)

	`0	1	2	3	4
0	W[0, 0]=2/20 C[0, 0]=0 r[0, 0]=0	W[1, 1]=3/20 C[1, 1]=0 r[1, 1]=0	W[2, 2]=2/20 C[2, 2]=0 r[2, 2]=0	W[3, 3]=3/20 C[3, 3]=0 r[3, 3]=0	W[4, 4]=1/20 C[4, 4]=0 r[4, 4]=0
1	W[0, 1]=7/20 C[0, 1]=7/20 r[0, 1]=1	W[1, 2]=6/20 C[1, 2]=6/20 r[1, 2]=2	W[2, 3]=8/20 C[2, 3]=8/20 r[2, 3]=3	W[3, 4]=7/20 C[3, 4]=7/20 r[3, 4]=4	
2	W[0, 2]=10/20 C[0, 2]=16/20 r[0, 2]=1	W[1, 3]=12/20 C[1, 3]=18/20 r[1, 3]=3	W[2, 4]=12/20 C[2, 4]=19/20 r[2, 4]=3		
3	W[0, 3]=16/20 C[0, 3]=31/20 r[0, 3]=2	W[1, 4]=16/20 C[1, 4]=29/20 r[1, 4]=3			
4	W[0, 4]=20/20 C[0, 4]=43/20 r[0, 4]=3				

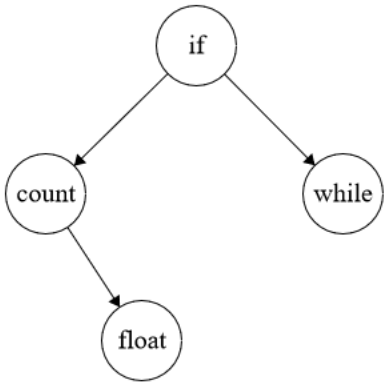
W[0, 4] = (p(1)+p(2)+p(3)+p(4)) + (q(0)+q(1)+q(2)+q(3)+q(4))
= (2/20 + 1/20 + 3/20 + 3/20) + (2/20 + 3/20 + 2/20 + 3/20 + 1/20)
= (9/20) + (11/20) = 20/20

C[0, 4] = k = 1 C[0, 0] + C[1, 4] = 0 + 29/20 = 29/20

DYNAMIC PROGRAMMING

k = 2 C[0, 1] + C[2, 4] = 7/20 +19/20 = 26/20
k = 3 C[0, 2] + C[3, 4] = 16/20 + 7/20 = 23/20
k = 4 C[0, 3] + C[4, 4] = 31/20 + 0 = 31/20

The optimal binary search tree is



Cost = 43/20 = 2.15

Algorithm:

```
Algorithm OBST (p, q, n)
{
        for i = 0 to n-1 do //Initialize
        {
                W[i, i] = q(i); r[i, i] = 0; c[i, i] = 0;
                W[i, i+1] = q(i) + q(i+1) + p(i+1);
                r[i, i+1] = i+1;
                c[i, i+1] = q(i) + q(i+1) + p(i+1);
        }
        W[n, n] = q(n); r[n, n] = 0; c[n, n] = 0;
        for m = 2 to n do
                for i = 0 to n-m do
                {
                        j = i+m;
                        W[i, j] = W[i, j-1] +p(j) +q(j);
                        k = find (c, r, i, j);
                        c[i, j] = W[i, j] + c[i, k-1] + c[k, j];
                        r[i, j] = k;
                }
```

DYNAMIC PROGRAMMING

```
        write ( c[0, n], W[0, n], r[0, n]);
}

Algorithm find (c, r, i, j)
{
        min = ∞;
        for m = r[i, j-1] to r[i+1, j] do
        {
                if ( c[i, m-1] + c[m, j] < min) then
                {
                        min = c[i, m-1] + c[m, j];
                        l = m;
                }
        }
        return l;
}
```

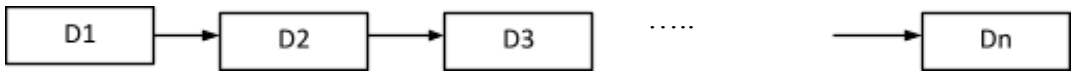
RELIABILITY DESIGN

The problem is to design a system that composed of several devices.

Devices	D1	D2	D3	D4	
Cost	C1	C2	C3	C4	
Reliability	r1	r2	r3	r4	
Reliability	0.9	0.9	0.9	0.9	(example)

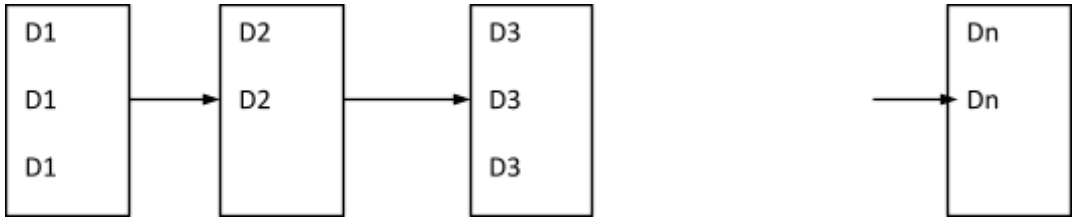
So, the reliability of the entire system = $\prod_{i=1}^{4} ri = (0.9)^4 = 0.6561$

The problem is design a system such that the reliability is maximum, so we should have more than one copy of devices.
The devices are connected serial



DYNAMIC PROGRAMMING

Suppose we have more than one device, the devices are connected as



One device is working and the remaining are back up.
The reliability of D1 have 3 copies is (r1 = 0.9)

r1 = 0.9 1-r1 = 1-0.9 = 0.1 (reliability for not working)
$(1-r1)^3 = (0.1)^3 = 0.001$ (for failed all copies)
$1-(1-r1)^3 = 0.999$ (for working all copies)

Example:
Design a three stage system with devices D1, D2, D3 and cost(C) = 105

Di	Ci	ri
D1	30	0.9
D2	15	0.8
D3	20	0.5

Solution:
We maintain multiple copies of devices. We maintain at least one copy of each device.
So, the minimum cost required = Σ Ci = 30+15+20 = 65
The remaining cost = 105 – 65 = 40

Di	Ci	ri	Ui
D1	30	0.9	2
D2	15	0.8	3
D3	20	0.5	3

DYNAMIC PROGRAMMING

We solve the problem using set method. We take an order pair (r, C) (reliability, cost)

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

Consider D1:

$S^1_1 = \{ (0.9, 30)\}$
$S^1_2 = \{(0.99, 60)\}$

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

Consider D2:

$S^2_1 = \{ (0.72, 45), (0.792, 75)\}$
$S^2_2 = \{(0.864, 60), (0.9504, 90)\}$
We eliminate (0.9504, 90), because not enough money for third device(D3)
$S^2_3 = \{ (0.8925, 75), (---, 105)\}$

So , $S^2 = \{(0.72, 45), (0.864, 60),(0.792, 75), (0.8925, 75) \}$
By using the dominance rule, if reliability increase then cost is also increase. So we eliminate the order pair (0.792, 75).
So, $S^2 = \{(0.72, 45), (0.864, 60),(0.8925, 75) \}$
Lo
Consider D3:

$S^3_1 = \{ (0.36, 65), (0.432, 80), (0.4464, 95)\}$
$S^3_2 = \{(0.54, 85), (0.648, 100), (---, 115)\}$
We eliminate (---, 115)
$S^3_2 = \{(0.54, 85), (0.648, 100),\}$
$S^3_3 = \{ (0.63, 105), (---, 120), (---, 135)\}$
We eliminate (---, 120), (---, 135)
$S^3_3 = \{ (0.63, 105)\}$

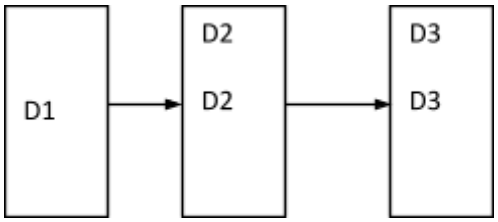
So , $S^3 = \{(0.36, 65), (0.432, 80),(0.54, 85), (0.4464, 95), (0.648, 100), (0.63, 105) \}$

DYNAMIC PROGRAMMING

By using the dominance rule, if reliability increase then cost is also increase. So we eliminate the order pair (0.4464, 95) and (0.63, 105).

So, S^3 = {(0.36, 65), (0.432, 80),(0.54, 85), (0.648, 100) }

Therefore the system is



Reliability = 0.648 cost = 100

0/1 KNAPSACK PROBLEM

We are given n objects and a knapsack or bag. Object i has a weight wi and profit pi and the knapsack has a capacity m. If a object i is selected, then weight wi is placed into the knapsack and profit pi is earned. The object is to obtain a filling the knapsack that maximize the total profit earned.

Solution to the knapsack problem can be obtained by making a sequence of decisions on variables x_1,x_2,x_3…….x_n i.e., the decision determine which of the values 0/1 is assigned to the variables.

It may be in one of the two possible states,if an object is directly placed into a knapsack then x=1 otherwise x=0.

PROCEDURE:-

Step1:- Let $f_i(m)$ represents the optimum solution.

Step2:- S[i] is a pair of (profit,weight) by using S^i. We calculate next values i.e., S^{i+1} .

DYNAMIC PROGRAMMING

Step3:- Initially $s^i=\{0,0\}$. To obtain the optimal solution we use $f_n(m)=\max\{ f_{n-1}(m), f_{n-1}(m-w_n)+f_n \}$ we need to compute the ordered set $S^i=\{ f((y_i),y_j) / 1\le j\le k \}$ to represent $f_i(y)$.

Step4:- Each member of S^i is a pair of (p,w) where $p=f_i(y_i)$, $w=y_j$.

Step5:- We can compute S^{i+1} from S^i by first computing s^i_1 .If contains two pairs (p_j, w_j), (p_k, w_k) with the property $p_j \le p_k$, $w_j \ge w_k$.then the pair (p_j , w_j) can be discarded. This is known as rule and sometimes it is known as purging rule.

EXAMPLE:- Let us consider a knapsack instance

p_i	w_i
1	2
2	3
5	4
6	5

Where n=4 and m=8.

Sol:- We have $S^0=(0,0)$

Object 1:

$S_0^1=\{ (0,0)\}$ (not include object 1)

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

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

Object 2:

$S_0^2=\{ (0,0), (1, 2)\}$ (not include object 2)

$S_1^2=\{(0+2,0+3), (1+2, 2+3)= \{ (2,3), (3, 5) \}$ (include object 2)

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

Object 3:

$S_0^3=\{(0,0), (1,2), (2,3), (3,5)\}$ (not include object 3)

$S_1^3=\{(0+5,0+4)(1+5,2+4)(2+5,3+4)(3+5,5+4)\}$ (include object 3)

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

$S^3=\{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7), (8,9)\}$

(after applying dominance rule (3,5) will be discarded and (8,9) discard because weight exceed the capacity of knapsack)

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

Object 4:

DYNAMIC PROGRAMMING

S_0^4={(0,0), (1,2), (2,3), (5,4), (6,6), (7,7)} (not include object 4)
S_1^4={(0+6,0+5)(1+6,2+5)(2+6,3+5)(5+6,4+5)(6+6,6+5),(7+6,7+5)}(include object 4)
={(6,5), p (11,9), (12,11), (13,12)}

S^4={(0,0), (1,2), (2,3), (5,4), (6,5), (6,6), (7,7), (8,8), (11,9), (12,11), (13,12), (14,14)}
(after applying dominance rule (6,6) will be discarded and (11,9), (12, 11), (13, 12),
(14, 14) discard because weight exceed the capacity of knapsack)
={(0,0), (1,2), (2,3), (5,4), (6,5), (7,7), (8,8)}

The pair (8, 8) $\in$ S^4 and (8, 8) $\notin$ S^3 so, the object 4 include the solution and $x_4 = 1$.
The pair (8, 8) came from the pair (8-p4, 8-w4) = (8-6, 8-5) = (2, 3)

The pair (2,3) $\in S^3$ and (2, 3) $\in S^2$ so, the object 3 not include the solution and $x_3 = 0$.

The pair (2,3) $\in S^2$ and (2, 3) $\notin$ S^1 so, the object 2 include the solution and $x_2 = 1$.
The pair (2, 3) came from the pair (2-p2, 3-w2) = (2-2, 3-3) = (0, 0)

The pair (0,0) $\in S^1$ and (0, 0) $\in S^0$ so, t wehe object 1 not include the solution and $x_1 = 0$.

Therefore **the optimal solution is (0, 1, 0, 1) and profit = 8.**

ALGORITHM:-

```
Algorithm DKP (p,w,n,m)
{
        S^0={(0,0)};
        for i:=1 to n-1 do{
                S_1^0 = S_1^{i-1}:={(p,w)/(p-p_i , w-w_i) ϵ S^{i-1} and w≤m;
                S^i:= merge purge(S^{i-1},S^{i-1}); □(S^0+S_1^1 = S^1)
                S^1 = (S^0,S^i)
        }
        (p_X,w_X) : = last pair in S^{n-1}
        (p_Y,w_Y) := (p^1+p_n,w^1,w_n) where w^1 is large w pair in S^{n-1} such
                        that w+w_n≤m;
        //True back for x_n,x_{n-1},…….x_1
```

If (p_x>p_y) then x_n=0;
else
x_n:=1
True back for(x_{n-1}.....x_1);
}

Time Complexity is - $O(2^{n/2})$

Travelling Salesman Problem:

- Let the 'G' be a directed graph with n vertices and edges each edge contains some cost.
- A tour of G is simple cyclic that includes every vertex in 'G'.
  The cost of the tour is the sum of the edges on the tour.
- The main objective of travelling salesman problem is to find the tour of the minimum cost.

Example:

1. Suppose we have to route a postal van to pickup mails from boxes located at n different sites.
2. And n+1 vertex graph may be used to represents the situation.
3. One vertex represents the post office from which postal van starts and to which it return.

4. The route taken by a van is atour.
5. We need to find the tour with minimum length.
    - Every tour consists of an edge <i,k> for some k belongs to v – { 1 }.
      And path from k to 1
    - A path from vertex k to vertex 1 goes through each vertex in v – {1,k} exactly once.
    - Let g(i,S) = the length of shortest path starting at vertex i, going through all vertices in S and terminating at vertex 1.
    - The function g(1,v – {1}) is the cost of the traveling salesman problem.

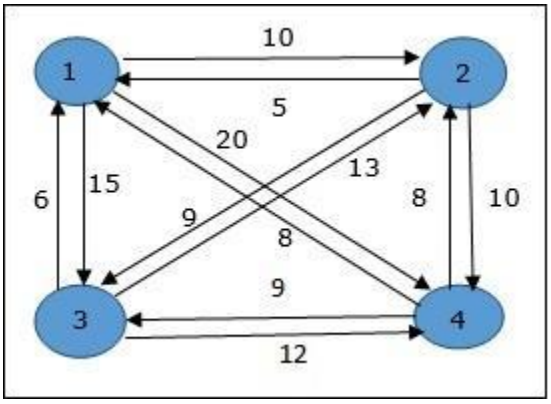
$g(i, S) = min_{jS}\{ c_{ij} + g(j , S-\{j\}) \}$

- For calculating distance from initial vertex to remaining vertices.
  $g(i,\varnothing\emptyset) = c_{i1}, 1 \leq i \leq n$

Example:

consider the following graph

DYNAMIC PROGRAMMING



Adajacency matrix for the above graph is :

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

We know that, $g(i,\emptyset)\varnothing) = c_{i1}$, $1<i\leq n$

$g(2, \varnothing) = C_{21} = 5$ $g(3, \varnothing) = C_{31} = 6$ $g(4, \varnothing) = C_{41} = 8$

Before solving this problem we make an assumption that traveling salesmanproblem starts and end at vertex 1.

$g(1 , \{2 , 3 , 4\})$

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

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

$= \min \{ 10+25, 15+ 27, 20+23 \}$

$=\min \{35,40,43\}$

$= 35$

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

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

$= \min \{ 9 + g(3,\{4\}) , 10 + g(4,\{3\}) \}$

$= \min \{ 9 + 20 ,10 + 15\}$

$= \min \{ 29 ,25 \} = 25$

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

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

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

$= \min \{ 13 + 18 ,12 + 15 \}$

$= \min\{ 31, 27 \} = 27$

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

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

$= \min \{ 8 + g(2,\{3\}) , 9 + g(3,\{2\}) \}$

$= \min \{ 8 + 15 ,9 + 18\}$

$= \min\{ 23 ,27 \} = 23$

$g(2, \{3\}) = \min \{ c_{23} + g(3 , 3 - \{ 3 \}) \}$

$= \min \{ 9 + g(3,\varnothing)\}$

$= \min \{ 9 + 6 \}$

$= \min \{15\} = 15$

$g(2, \{4\}) = \min \{ c_{24} + g(4 , 4 - \{ 4 \}) \}$

$= \min \{10 + g(4,\varnothing)\}$

$= \min \{ 10 + 8 \}$

$= \min \{18\} = 18$

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

$= \min \{ 13 + g(2,\varnothing)\}$

$= \min \{ 13 + 5)$

$= \min \{18\} = 18$

$g(3, \{4\}) = \min \{ c_{34} + g(4 , 4 - \{ 4 \}) \}$

$= \min \{12 + g(4,\varnothing)\}$

$= \min \{ 12 + 8 \}$

$= \min \{20\} = 20$

$g(4, \{2\}) = \min \{ c_{42} + g(2 , 2 - \{ 2\}) \}$

$= \min \{ 8 + g(2,\varnothing)\}$

$= \min \{ 8 + 5 \}$

$= \min \{13\} = 13$

$g(4, \{3\}) = \min \{ c_{43} + g(3 , 3 - \{ 3 \}) \}$

$= \min \{ 9 + g(3,\varnothing)\}$

$= \min \{ 9 + 6)$

= min {15} = 15

The shortest path for visiting all the vertices is **1-2-4-3-1 and cost = 35.**

Analysis:- Time complexity is $O(n^2,2^n)$

Space complexity is $O(n,2^n)$