UNIT - III

Dynamic Programming

General Method

Applications of Dynamic Programming

→ optimal binary search trees

→ 0/1 Knapsack problem

→ All pairs shortest path problem

→ Travelling salesman problem

→ Reliability design.

Optimal Binary search Tree

Tree represents the nodes connected by edges. Binary tree is a special data structure used for storage purposes. A binary tree has a special condition that each node can have a maximum of two children.

Binary search tree

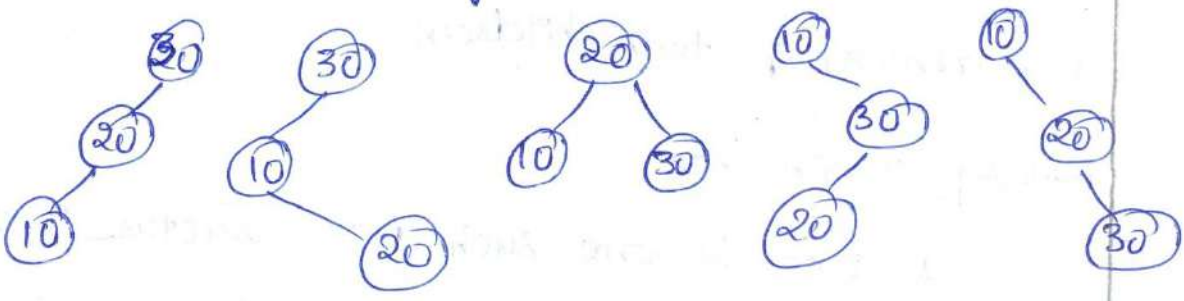
A BST is one such data structure that supports faster searching, rapid sorting and easy insertion and deletion. BST is a sorted version of a binary tree.

The ordering property of a BST is also referred to as BST property. For a node x of a BST T, the left subtree of x contains elements that are less than those stored in x. The right subtree of x contains all elements that are greater than those stored in x, with the assumption that the keys are distinct.

An optimal BST is a balanced binary search tree whose search time for finding the elements or items is optimal.

construction of Binary search trees

<u>example</u> given keys 10, 20, 30



The number of possible resulting trees is a catalan number.

Catalan numbers are a sequence of natural numbers that occurs in many interesting counting problems.

Using Binomial coefficient nth catalan number can be found as $\boxed{C_n = \frac{{}^{2n}C_n}{n+1}}$

catalan numbers satisfy the following recursive formula.

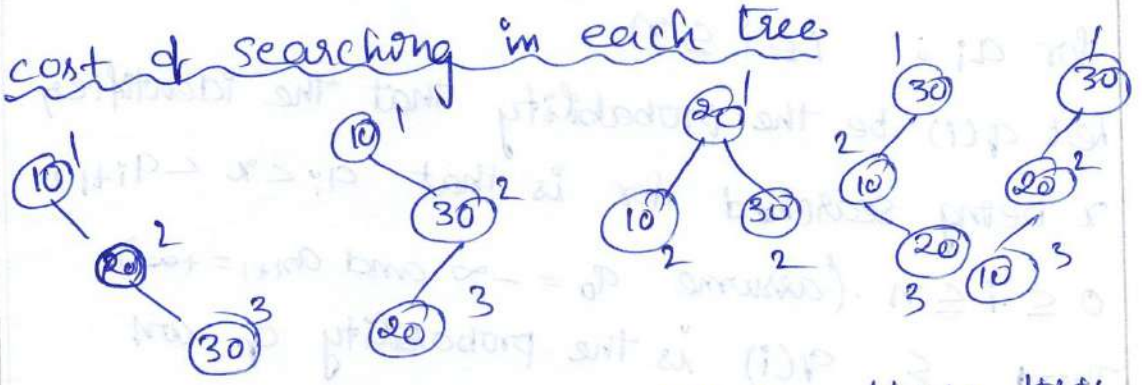
$C_0 = 1$, $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$ for $n \geq 0$

so, for our example $n = 3$

$$C_3 = \frac{{}^{2\times 3}C_3}{3+1} = \frac{{}^{6}C_3}{4} = 5$$

The total no. of BSTs with n keys is equal to nth catalan number.
The no. of comparisions in the binary tree depends on the height of the tree. so, if the height of the tree is less the no. of comparisons will be less.

cost of searching in each tree



$1+2+3=6$ $1+2+3=6$ $1+2+2=5$ $1+2+3=6$ $1+2+3=6$

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

To find whether an identifier x is present or not.

(i) x is compared with the root node.

(ii) If x is less than the value of the root node, then search continues in left subtree.

(iii) If x is greater than the value of the root node, then search continues in the right subtree.

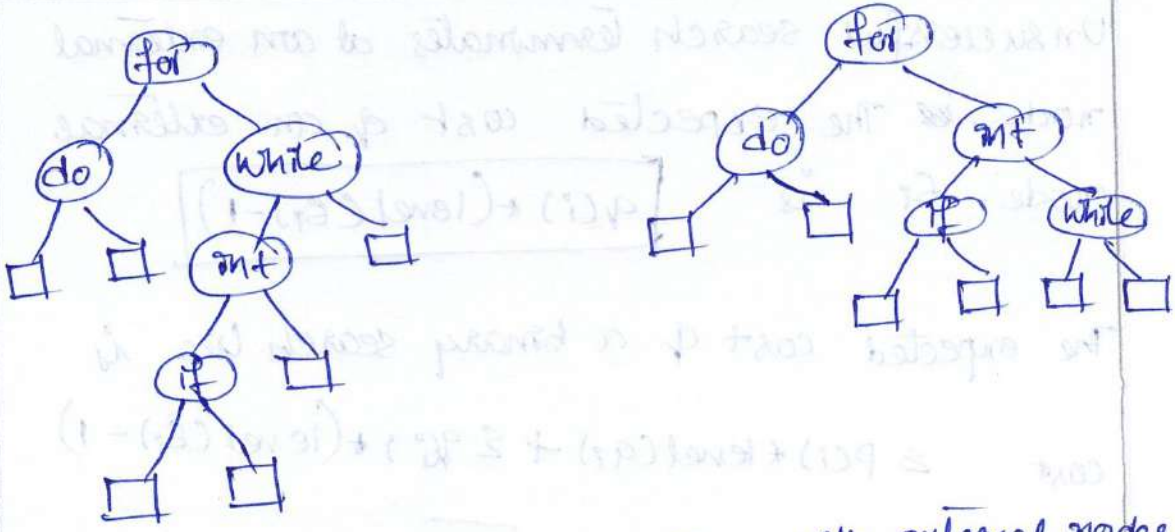
Let us assume that the given set of identifiers is $\{a_1, a_2 \ldots a_n\}$ with $a_1 < a_2 \ldots < a_n$.

Let $P(i)$ the probability with which we search for a_i. $1 \le i \le n$

Let $q(i)$ be the probability that the identifier x being searched for is that $a_i < x < a_{i+1}$, $0 \le i \le n$. (assume $a_0 = -\infty$ and $a_{n+1} = +\infty$)

Then $\sum_{0 \le i < n} q(i)$ is the probability of an unsuccessful search.

In obtaining a cost function for binary search tree, it is useful to add a fictious node in place of every empty subtree in the search tree. Such nodes are called external nodes.



Binary search trees with external nodes.

External nodes are represented by square boxes.
All other nodes are internal nodes.
If a binary search tree represents n identifiers, then there will be exactly n internal nodes and n+1 (fictious) external nodes.
Every internal node represents a point where an successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.
If a successful search terminates at an internal node at level 'L', then, the expected cost contribution from the internal node for 'a_i' is

$P(i) * level(a_i)$

Unsuccessful search terminates at an external node. The expected cost of an external node E_i is $\boxed{q(i) * (level(E_i) - 1)}$

The expected cost of a binary search tree is

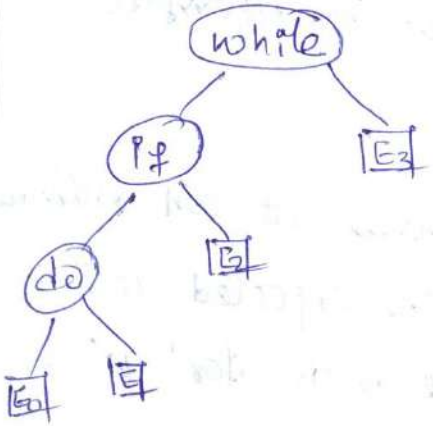
$$\text{cost} \quad \Sigma P(i) * level(a_i) + \Sigma q(i) * (level(E_i) - 1)$$

Example

Find OBST for a set of identifiers $(a_1, a_2, a_3) = (do, if, while)$. $(P_1, P_2, P_3) = (0.5, 0.1, 0.05)$ and $(q_0, q_1, q_2, q_3) = (0.15, 0.1, 0.05, 0.05)$

Sol. $n=3$ the no. of possible binary search trees are

$$\frac{{}^{2n}C_n}{n+1} = \frac{{}^{2\times3}C_3}{3+1} = \frac{{}^{6}C_3}{4} = 5$$



cost(tree a)

$= 0.05 * 1 + 0.1 * 2 + 0.5 * 3$
$0.15 * (4-1) + 0.1 * (4-1) +$
$0.05 * (3-1) + 0.05 * (2-1)$

$= 2.65$.

$$c(i,j) = \min_{i<k\le j} \{c(i,k-1) + c(k,j)\} + w(i,j)$$

$$w(i,j) = p(j) + q(j) + w(i,j-1)$$

$$r(i,j) = k$$

$c(i,j)$ can be solved for $c(0,n)$ by computing all $c(i,j)$ such that $j-i=1$.
Next we compute all $c(i,j)$ such that $j-i=2$ then all $c(i,j)$ with $j-i=3$ and so on.
If during this computation we record the root $r(i,j)$ of each tree t_{ij}, then an optimal binary search tree can be constructed from these $r(i,j)$.

Let $n=4$ and $(a_1, a_2, a_3, a_4) = (do, if, int, while)$
Let $p(1:4) = (3,3,1,1)$ and $q(0:4) = (2,3,1,1,1)$

Initially

$w(i,i) = q(i)$

$c(i,i) = 0$

$r(i,i) = 0$ $\quad 0 \le i \le 4$.

$j-i = 0$

$w(0,0) = q(0) = 2$

$c(0,0) = 0$

$r(0,0) = 0$

$w(1,1) = q(1) = 3$

$c(1,1) = 0$

$r(1,1) = 0$

$w(2,2) = q(2) = 1$

$c(2,2) = 0$

$r(2,2) = 0$

$w(3,3) = q(3) = 1$

$c(3,3) = 0$

$r(3,3) = 0$

$w(4,4) = 1$

$c(4,4) = 0$

$r(4,4) = 0$

Now $j - i = 1$

$w(0,1) = p(1) + q(1) + w(0,1-1)$

$= 3 + 3 + 2 = 8$

$c(0,1) = \min\limits_{0<k<1} \{c(i,k-1) + c(k,j)\} + w(0,1)$

$K=1$ $= \min\{c(0,0) + c(1,1)\} + w(0,1)$

$= \min\{0 + 0\} + 8 = 8$

$r(0,1) = 1$

$w(1,2) = w(1,1) + p(2) + q(2) = 3+3+1 = 7$

$c(1,2) = \min\limits_{\substack{1<k\leq 2 \\ k=2}} \{c(1,2-1) + c(2,2)\} + w(1,2)$

$= \min\{c(1,1) + c(2,2)\} + w(1,2)$

$= \min\{0+0\} + 7 = 7$

$r(1,2) = 2$

$w(2,3) = w(2,3-1) + p(3) + q(3)$

$= w(2,2) + 1 + 1 = 1+1+1 = 3$

$c(2,3) = \min\limits_{\substack{2<k\leq 3 \\ k=3}} \{c(2,3-1) + c(3,3)\} + w(2,3)$

$= \min\{0+0\} + 3 = 3$

$r(2,3) = 3$

$w(3,4) = w(3,4-1) + p(4) + q(4)$

$= w(3,3) + p(4) + q(4) = 1+1+1 = 3$

$c(3,4) = \min\limits_{\substack{3<k\leq 4 \\ k=4}} \{c(3,4-1) + c(4,4)\} + w(3,4)$

$= \min\{0+0\} + 3 = 3$

$r(3,4) = 4$

$|j-i| = 2$

$w(0,2) = w(0,2-1) + P(2) + q(2) = 8 + 3 + 1 = 12$

$$c(0,2) = \min_{\substack{0<k\le 2 \\ k=1,2}} \left\{ \begin{matrix} c(0,1-1) + c(1,2) \\ c(0,2-1) + c(2,2) \end{matrix} \right\} + w(0,2)$$

$= \min\{c(0,0) + c(1,2),\ c(0,1) + c(2,2)\} + w(0,2)$

$= \min\{0+7,\ 8+0\} + 12$

$= \min\{7, 8\} + 12$

$= 7 + 12 = 19$

$r(0,2) = 1$

$w(1,3) = w(1,3-1) + P(3) + q(3) = 7 + 1 + 1 = 9$

$$c(1,3) = \min_{\substack{1<k\le 3 \\ k=2,3}} \left\{ \begin{matrix} c(1,2-1) + c(2,3) \\ c(1,3-1) + c(3,3) \end{matrix} \right\} + w(1,3)$$

$= \min(c(1,1) + c(2,3),\ c(1,2) + c(3,3)) + w(1,3)$

$= \min(0+3,\ 7+0) + 9$

$= \min(3, 7) + 9 = 3 + 9 = 12$

$r(1,3) = 2$

$w(2,4) = w(2,4-1) + p(4) + q(4)$

$= w(2,3) + 1 + 1 = 3 + 1 + 1 = 5$

$$c(2,4) = \min_{\substack{2 < k \le 4 \\ k = 3,4}} \left\{ \begin{matrix} c(2,3-1) + c(3,4) \\ c(2,4-1) + c(4,4) \end{matrix} \right\} + w(2,4)$$

$= \min \{ c(2,2) + c(3,4),\ c(2,3) + c(4,4) \} + w(2,4)$

$= \min \{ 0+3,\ 3+0 \} + 5 = 3 + 5 = 8$

$r(2,4) = 3$ or 4

$|j-i| = 3$

$w(0,3) = w(0,3-1) + p(3) + q(3)$

$= w(0,2) + p(3) + q(3) = 12 + 1 + 1 = 14$

$$c(0,3) = \min_{\substack{0 < k \le 3 \\ k = 1,2,3}} \left\{ \begin{matrix} c(0,1-1) + c(1,3) \\ c(0,2-1) + c(2,3) \\ c(0,3-1) + c(3,3) \end{matrix} \right\} + w(0,3)$$

$= \min \{ c(0,0) + c(1,3),\ c(0,1) + c(2,3),\ c(0,2) + c(3,3) \}$
$+ w(0,3)$

$= \min \{ 0 + 12,\ 8 + 3,\ 19 + 0 \} + 14$

$= \min \{ 12, 11, 19 \} + 14$

$= 11 + 14 = 25$

$r(0,3) = 2$

$$w(1,4) = w(1,4-1) + p(4) + q(4)$$

$$= w(1,3) + p(4) + q(4) = 9+1+1 = 11$$

$$c(1,4) = \min_{\substack{1 \le k \le 4 \\ k=2,3,4}} \left\{ \begin{array}{l} c(1,2-1) + c(2,4) \\ c(1,3-1) + c(3,4) \\ c(1,4-1) + c(4,4) \end{array} \right\} + w(1,4)$$

$$= \min \{ c(1,1) + c(2,4), c(1,2) + c(3,4), c(1,3) + c(4,4) \} + w(1,4)$$

$$= \min \{ 0+8, 7+3, 12+0 \} + 11$$

$$= \min \{ 8, 10, 12 \} + 11$$

$$= 8+1 = 19$$

$$r(1,4) = 2$$

$$|j-i| = 4$$

$$w(0,4) = w(0,4-1) + p(4) + q(4)$$

$$= w(0,3) + p(4) + q(4)$$

$$= 14 + 1 + 1 = 16$$

$$c(0,4) = \min_{\substack{0 < k \le 4 \\ k=1,2,3,4}} \left\{ \begin{array}{l} c(0,1-1) + c(1,4) \\ c(0,2-1) + c(2,4) \\ c(0,3-1) + c(3,4) \\ c(0,4-1) + c(4,4) \end{array} \right\} + w(0,4)$$

$$= \min (c(0,0) + c(1,4), c(0,1) + c(2,4), c(0,2) + (3,4), c(0,3) + c(4,4) \} + w(0,4)$$

$= \min\{0+19, 8+8, 19+3, 25+0\} + 16$

$= \min\{19, 16, 22, 25\} + 16$

$= 16 + 16 = 32$

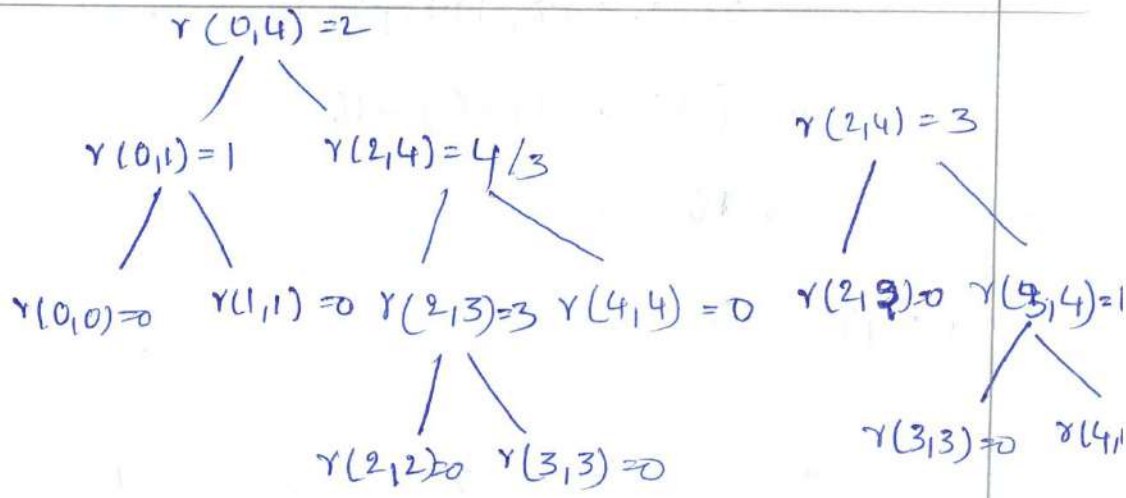
$r(0,4) = 2$.

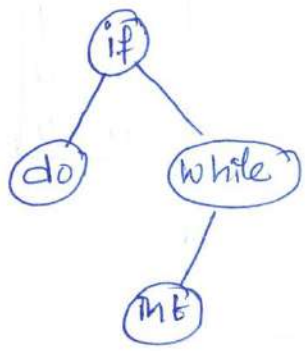
	0	1	2	3	4
0	$w_{00} =$ $c_{00} =$ $r_{00} =$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
1	$w_{01} =$ $c_{01} =$ $r_{01} =$	$w_{12} =$			
2	$w_{02} =$	w_{13}			
3	$w_{03} =$				
4	$w_{04} =$				

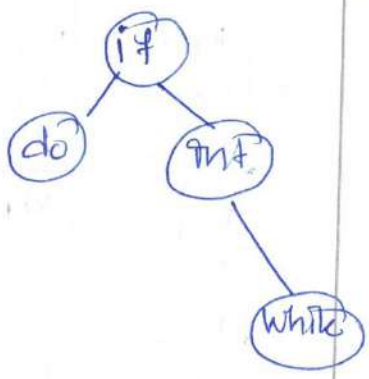
Now we need to construct the OBST.

$r(i,j) = k$

$r(i,k-1)$ $\quad$ $r(k,j)$


r(0,4) = 2
r(0,1) = 1
r(2,4) = 4/3
r(2,4) = 3
r(0,0) = 0
r(1,1) = 0
r(2,3) = 3
r(4,4) = 0
r(2,3) = 0
r(3,4) = 1
r(2,2) = 0
r(3,3) = 0
r(3,3) = 0


if
do
while
int


if
do
int
while

0/1 knapsack

The knapsack problem is one of the most popular algorithms. This can be stated formally as follows:

The knapsack problem has a knapsack of capacity K. There are n different items ~~which~~ each of which is associated with a weight w_i and a profit P_i (also called a value). The objective of this problem is to load a knapsack with as many items as possible (subjected to the capacity of the knapsack) to get the maximum profit.

Two types of knapsack problems are there

1. Fractional and
2. 0/1 knapsack.

Dynamic programming is useful in solving the 0/1 knapsack problem where either the item is selected or rejected subjected to the constraints of the knapsack capacity.

Fractional knapsack problem

Items are divisible, you can take any fraction of an item.

0/1 Knapsack Problem : Items are indivisible; we can either take an item or not.

0/1 Knapsack

We are given n objects and a knapsack or bag. Object i has a weight w_i and the knapsack has a capacity m.

A solution to the knapsack problem can be obtained by making a sequence of decisions on the variables $x_1, x_2, \ldots x_n$.

A decision on variable x_i involves determining which of the values 0 or 1 is to be assigned to it.

Let decisions on x_i are made in the order $x_n, x_{n-1} \ldots x_1$. Following a decision on x_i, we may be in one of two possible states:

→ The capacity remaining in the knapsack is m and no profit has added.

→ The capacity remaining is $m - w_i$ and a profit of p_n is added.

The remaining decisions $x_{n-1}, \ldots x_1$ must be optimal with respect to the problem state resulting from the decision on x_n. Otherwise $x_n, \ldots x_1$ will not be optimal. Hence the principle of optimality holds.

Let $f_j(y)$ be the value of an optimal solution to KNAP$(1, j, y)$. Since the principle of optimality holds, we obtain

$$f_n(m) = \max\{f_{n-1}(m), f_{n-1}(m - w_n) + p_n\} \quad \text{— (1)}$$

For $f_i(y)$, $i > 0$ equation (1) genalizes to

$$f_i(y) = \max\{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\} \quad \text{— (2)}$$

~~Equation (2) can be generalized as~~

$$\cancel{f_i(y) = \max\{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\}}$$

Equation (2) can be solved for $f_n(m)$ by beginning with the knowledge $f_0(y) = 0$ for all non-negative y and $f_i(y) = -\infty$, $y < 0$. Then $f_1, f_2 \ldots f_n$ can be successively computed using (2).

$f_i(y)$ is an ascending step function;
ie there are a finite number of y's,
$0 = y_1 < y_2 < \dots y_k$, such that $f_i(y_1) < f_i(y_2)$
$\dots < f_i(y_k)$; $f_i(y) = -\infty$. $y < y_1$;
$f_i(y) = f_i(y_k)$, $y \geq y_k$ and $f_i(y) = f_i(y_j)$
$y_j \leq y < y_{i+1}$.
So, we need to find out $f_i(y_j)$, $1 \leq j \leq k$.
we used the ordered set $S^i = \{(f(y_j), y_j) \mid 1 \leq j \leq k\}$
to represent $f_i(y)$.
each member of S^i is a pair (P, w), where
$P = f_i(y_j)$ and $w = y_j$.
Initially $S^0 = \{(0,0)\}$
we compute S^{i+1} from S^i by first computing

$$S_1^i = \{(P, w) \mid (P - P_{i+1}, w - w_{i+1}) \in S^i\}$$

Now S^{i+1} can be computed by merging
the pairs in S^i and S_1^i together.
<u>Note</u>: If S^{i+1} contains two pairs (P_j, w_j)
and (P_k, w_k) with the property that
$P_j \leq P_k$ and $w_j \geq w_k$, then the pair

(P_j, w_j) can be discarded because of eq (2)
Discarding or purging rules such as this one are also known as dominance rules. Dominated tuples get purged. In this (P_k, w_k) dominates (P_j, w_j).

We can solve the knapsack problem via a systematic examination of the up to 2^n possibilities for $x_1, x_2 \ldots x_n$.
Let S^i represent the possible states resulting from the 2^i decision sequences for $x_1, \ldots x_i$.
A state refers to a pair (P_j, w_j), w_j being the total weight of objects included in the knapsack and P_j being the corresponding profit.
To obtain S^{i+1}, we note that the possibilities for x_{i+1} are $x_{i+1} = 0$ or $x_{i+1} = 1$.
When $x_{i+1} = 0$, the resulting states are the same as for S^i.
When $x_{i+1} = 1$, the resulting states are obtained by adding (P_{i+1}, w_{i+1}) to each state in S^i.
The set of these additional states is called S_1^i

Now S^{i+1} can be computed by merging the states in S^i and S_1^i together.

eg: given $n=3$, $(w_1, w_2, w_3) = (2, 3, 4)$, $(P_1, P_2, P_3) = (1, 2, 5)$ and $m=6$.

$S^0 = \{(0,0)\}$; $S_1^0 = \{\overset{(1,2)}{(0+1, 0+2)}\} = S_1^0 \{(1,2)\}$

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

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

$S^2 = S^1 \cup S_1^1 = \{(0,0), (1,2)\} \cup \{(2,3), (3,5)\}$

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

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

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

$S^3 = S^2 \cup S_1^2 = \{(0,0), (1,2), (2,3), (3,5)\} \cup \{(5,4), (6,6), (7,7), (8,9)\}$

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

(P_j, w_j) (P_k, w_k) $(3,5)$ $(5,4)$

$P_j \leq P_k$ $w_j \geq w_k$

$3 < 5$ $5 > 4$ $\overset{(3,5)}{(P_j, w_j)}$ can be dominated

(3,5) is eliminated from S^3 as a result of the purging rule. ~~state~~

When generating the S^i's we can also purge all pairs (P, W) with $W > m$ as these pairs determine the value of $f_n(x)$ only for $x > m$. When all pairs (P_j, W_j) and $W_j > m$ are purged from the S^i's, $f_n(m)$ is given by the P value of the last pair in S^n.

By computing S^n we can find the solutions to all the knapsack problems $KNAP(1, n, x)$, $0 \le x \le m$ and not just $KNAP(1, n, m)$. Since we want only a solution to $KNAP(1, n, m)$ we can dispense with the computation of S^n. The last pair in S^n is either the last one in S^{n-1} or it is $(P_j + P_n + W_j + w_n)$, where $(P_j, W_j) \in S^{n-1}$ such that $W_j + w_n \le m$ and W_j is maximum.

If $(P1, W1)$ is the last tuple in S^n, a set of 0/1 values for the x_i's such that $\Sigma P_i x_i = P1$ and $\Sigma w_i x_i = W1$ can be determined by carrying out a search through S^i's.

we can set $x_n = 0$ if $(P1, W1) \in S^{n-1}$.
If $(P1, W1) \notin S^{n-1}$, then $(P1 - P_n, W1 - w_n) \in S^{n-1}$ and we can set $x_n = 1$. This can be done recursively.

<u>example</u>

$S^3 = \{(0,0), (1,2), (2,3), (5,4), (6,6)\}$ is the set of tuples obtained after removing $m > 6$ pairs.

$f_n(x) = f_3(6)$ is given by $(6,6)$ in S^3.

The tuple $(6,6) \notin S^2$ and so we must set $x_3 = 1$.

The pair $(6,6)$ is obtained from $(6 - P_3, 6 - w_3)$
$(6-5, 6-4) = (1,2)$.

Hence $(1,2) \in S^2$.

Since $(1,2) \in S^1$, we can set $x_2 = 0$.

Since $(1,2) \notin S^0$ we obtain $x_n = 1$.

Hence an optimal solution is $(x_1, x_2, x_3) = (1, 0, 1)$

All-pairs shortest path Problem

Let $G=(V,E)$ be a directed graph with n vertices. V - set of vertices, E - set of Edges.

Let cost be a cost adjacency matrix for G such that

$$cost[i,j] = \begin{cases} 0 & \text{if } (i==j) \\ c_{ij} & \text{if } (i,j) \in E(G) \\ \infty & \text{if } (i,j) \notin E(G). \end{cases}$$

The all-pairs shortest-path problem is to determine a matrix A such that $A(i,j)$ is the length of the shortest path from i to j.

For the shortest i to j path in G, $i \neq j$.
This path originates at vertex i and goes through some intermediate vertices and terminates at vertex j.
We assume that this path contains no cycles and negative length.
If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to j and k to j respectively. Otherwise, the i to j path

is not of minimum length. so, the principle of optimality holds.

If k is the intermediate vertex with highest index, then the i to k path is a shortest i to k path in G going through no vertex with index greater than k-1.

similarly the k to j path is a shortest k to j path in G going through no vertex of index greater than k-1.

so, in the construction of a shortest path i to j a decision need to be made regarding which is the highest indexed intermediate vertex k.

once this decision is made, we need to find two shortest paths, one from i to k and other from k to j. Neither of these may go through a vertex with index greater than k-1.

Now assume $A^k(i,j)$ represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain.

$$A^k(i,j) = \min\left\{\min_{1\le k\le n}\{A^{k-1}(i,k) + A^{k-1}(k,j)\}, \text{cost}(i,j)\right\}$$

If we assume that there is a path between i and j involving nodes $\{1, 2, \ldots k\}$. Then the nodes 1 to k are called internal or intermediate nodes.

The concept of this ~~abo~~ all pairs shortest path algorithm is to find the path from i to j such that the internal nodes belong to the subset $\{1, 2, \ldots k\}$.

Initially $A^0(i,j) = cost(i,j)$, $1 \le i \le n$, $1 \le j \le n$.
Then in subsequent iterations, the next matrix A is computed.

$A^{(1)}$ is computed with only one intermediate node
$A^{(2)}$ is computed with only two intermediate nodes
In general A^k is computed from A^{k-1} and there are only two possibilities for computing it, which are as follows:

1. The first possibility is that node k is not involved. In this case, since the node is not part of the path, only nodes $\{1, 2, \ldots k\}$ are considered.
Hence $A^k(i,j) = A^{k-1}(i,j)$.

2. The second possibility is that node k is used as an intermediate node. In this case one

can split the path i to k and k to j.

Hence $A^k(i,j) = A^{k-1}(i,k) + A^{k-1}(k,j)$.

combining we get

$$A^k(i,j) = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\} \quad k \geq 1$$

Informal Algorith ~~for~~ of Floyd-warshall.

Step 1 : Read weighted graph $G = \langle V, E \rangle$

Step 2 : Initialize $A[i,j]$ as follows:

$$cost[i,j] = \begin{cases} 0 & \text{if } i=j \\ \infty & \text{if edge } (i,j) \notin E(G) \\ w_{ij} & \text{if edge } (i,j) \in E(G) \end{cases}$$

Step 3 : For intermediate nodes k, $1 \leq k \leq n$, recursively compute A_{ij} ~~min~~ where

$$A_{ij} = \min_{1 \leq k \leq n} \{A_{ij}, A_{ik} + A_{kj}\}$$

Step 4 : Return Matrix A

Step 5 : End

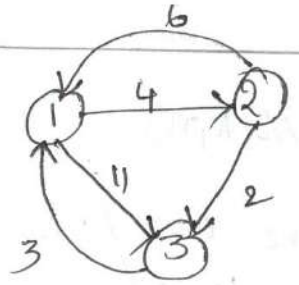
Now we can write the algorithm for all pairs shortest path as follows:

```
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 ≤ i ≤ 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]);
}
```

The time complexity of the above algorithm is $O(n^3)$ as there are 3 for loops.

Example



Cost Matrix

	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$= A^0$

Here $n=3$. for $k=1,2,3$ we have to find out the A matrix A^1, A^2, A^3.

General formula

$$A^k(i,j) = \min\{A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\}$$

$\Rightarrow$ For $k=1,\ i=1,\ j=1,2,3$

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

$= \min\{0,\ 0+0\} = \min\{0,0\}$

$= 0$

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

$= \min\{4,\ 0+4\} = \min\{4,4\}$

$= 4$

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

$= \min\{11,\ 0+11\} = \min\{11,11\}$

$= 11$

For $k=1,\ i=2,\ j=1,2,3$

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

$= \min\{6,\ 6+0\} = \min\{6,6\}$

$= 6.$

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

$= \min\{0,\ 6+4\} = \min\{0,10\}$

$= 0$

$A^1(2,3) = \min\{A^0(2,3), A^0(2,1) + A^0(1,3)\}$

$= \min\{2, 6+11\} = \min\{2, 17\}$

$= 2.$

For $k=1, i=3, j=1,2,3$

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

$= \min\{3, 3+0\} = \min(3,3\}$

$= 3$

$A^1(3,2) = \min(A^0(3,2), A^0(3,1) + A^0(1,2)\}$

$= \min\{\alpha, 3+4\} = \min(\alpha, 7)$

$= 7$

$A^1(3,3) = \min(A^0(3,3), A^0(3,1) + A^0(1,3)\}$

$= \min\{0, 3+11\} = \min\{0,14\}$

$= 0$

Now

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

$\Rightarrow$ For $k=2, i=1, j=1,2,3$

$A^2(1,1) = \min\{A^1(1,1), A^1(1,2) + A^1(2,1)\}$

$= \min\{0, 4+6\} = \min\{0,10\}$

$= 0$

$A^2(1,2) = \min\{A^1(1,2), A^1(1,2) + A^1(2,2)\}$

$= \min\{4, 4+0\} = \min\{4,4\}$

$= 4$

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

$= \min\{11, 4+2\} = \min\{11, 6\}$

$= 6$

For $k=2$, $i=2$, $j=1,2,3$

$A^2(2,1) = \min(A^1(2,1), A^1(2,2) + A^1(2,1))$

$= \min(6, 0+6) = \min(6,6)$

$= 6$

$A^2(2,2) = \min(A^1(2,2), A^1(2,2) + A^1(2,2))$

$= \min(0, 0+0)$

$= 0$

$A^2(2,3) = \min(A^1(2,3), A^1(2,2) + A^1(2,3)$

$= \min(2, 0+2) = \min(2,2)$

$= 2$

For $k=2$, $i=3$, $j=1,2,3$

$A^2(3,1) = \min(A^1(3,1), A^1(3,2) + A^1(2,1)$

$= \min(3, 7+6) = \min(3,13)$

$= 3$

$A^2(3,2) = \min(A^1(3,2), A^1(3,2) + A^1(2,2)$

$= \min(7, 7+0) = \min(7,7)$

$= 7$

$A^2(3,3) = \min(A^1(3,3), A^1(3,2) + A^1(2,3)$

$= \min(0, 7+2) = \min(0,9)$

$= 0.$

$A^3(2,3) = \min\{A^2(2,3),\ A^2(2,3) + A^2(3,3)\}$

$= \min\{2,\ 2+0\} = \min\{2,2\}$

$= 2$

$k=3,\ i=3,\ j=1,2,3$

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

$= \min\{3,\ 0+3\} = \min\{3,3\}$

$= 3$

$A^3(3,2) = \min\{A^2(3,2),\ A^2(3,3) + A^2(3,2)\}$

$= \min\{7,\ 0+7\} = \min\{7,7\}$

$= 7$

$A^3(3,3) = \min\{A^2(3,3),\ A^2(3,3) + A^2(3,3)\}$

$= \min\{0,\ 0+6\} = \min(0,6)$

$= 0.$

A^3

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

Final All pairs shortest path solution matrix.

A^2

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

$\Rightarrow$ For $k=3, i=1, j=1,2,3$

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

$A^3(1,2) = \min\{A^2(1,2), A^2(1,3)+A^2(3,2)\}$
$= \min\{4, 6+7\} = \min(4,13)$
$= 4$

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

$K=3, i=2, j=1,2,3$

$A^3(2,1) = \min\{A^3(2,1), A^2(2,3)+A^2(3,1)\}$
$= \min\{6, 2+3\} = \min\{6,5\}$
$= 5$

$A^3(2,2) = \min\{A^3(2,2), A^2(2,3)+A^2(3,2)\}$
$= \min\{0, 2+7\} = \min(0,9)$
$= 0$

Travelling salesperson problem

computer resources are limited. Hence, many problems that require a large amount of resources cannot be solved. one good example is the travelling salesperson problem (TSP).

TSP is the ~~most~~ one of the most important optimization problem. The objective is to find a tour starting from a city, visiting all other cities only once and finally returning to the city where the tour has started.

Dynamic programming is a popular approach for solving the TSP.

TSP can be modeled as a graph. A graph consists of a set of nodes and edges that interconnect the nodes. In this case, cities are the nodes and the paths that connect the cities are the edges.

Given a graph $G=(V,E)$ with weighted adjacency matrix A. Let us assume that vertex 1 is the starting node of the TSP tour. The problem involves computation of the minimum-cost path that starting ~~at~~ from vertex 1 visits all other vertices exactly once. Assume that the cost of such path is $g(i)$. The cost of the TSP cycle is $g(i)+c(i)$, where $c(i)$ is

the distance from vertex i to the starting node.

First, the distance matrix $d[i,j]$ is computed from the weighted adjacency matrix as follows:

$$d[i,j] = \begin{cases} \infty & \text{if edge } (i,j) \notin E(G) \\ 0 & \text{if } i=j \\ w_{ij} & \text{if edge}(i,j) \in E(G) \end{cases}$$

The difficult part is to find cost(i). This is computed recursively using the dynamic programming approach.

The recursion involves computation of a function $g(S,i)$ that indicates the shortest path starting from vertex 1, visiting all the vertices of set S and ending at vertex i

The function $g(S,i)$ is given as follows:

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \} \quad -(1)$$

Obviously, if S is null, the cost can be obtained directly as follows:

$$g(i,\phi) = c_{i1} \quad 1 \le i \le n.$$

When S is not null, the function $g(S,i)$ needs to be computed for every possible vertex in S. Once all these values are obtained and stored, the final

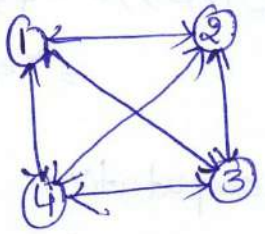
cost of the tour can be computed using dynamic programming as follows;

$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{ c_{1k} + g(k, V-\{1,k\}) \} \quad -(2)$$

Hence, we can use equation (1) to obtain $g(i,S)$ for all S of size 1. Then we can obtain $g(i,S)$ for S with $|S| = 2$ and soon.

Example:

consider the directed graph of the following figure. The edge lengths are given by matrix given below.



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Let us assume that the starting node is 1. Hence, the objective is to find the TSP tour that starts from vertex 1, visits all other vertices exactly once and returns to the destination vertex 1.

Now, we have to compute $g(i,S)$ function recursively.
When $|S| = \phi$, the cost functions for vertices 2,3 and 4 would be computed directly using base condition as follows:

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

This indicates the distance from vertices 2,3 and 4 to vertex 1. When $|S| = 1$, $g(i,S)$ can be solved using the recursive function as follows:

$g(2,\{3\}) = C_{23} + g(3,\phi)$
$= 9 + 6 = 15$ — ①

$g(2,\{4\}) = C_{24} + g(4,\phi)$
$= 10 + 8 = 18$ — ②

$g(3,\{2\}) = C_{32} + g(2,\phi)$
$= 13 + 5 = 18$ — ③

$g(3,\{4\}) = C_{34} + g(4,\phi)$
$= 12 + 8 = 20$ — ④

$g(4,\{2\}) = C_{42} + g(2,\phi)$
$= 8 + 5 = 13$ — ⑤

$g(4,\{3\}) = C_{43} + g(3,\phi)$
$= 9 + 6 = 15$ — ⑥

It can be observed that the computation involves reuse from the earlier iteration and does not involve any new computation.
Now, $g(i,S)$ is computed with $|S| = 2$ that is, set S involves two intermediate nodes.

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

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

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

$= 25$ — (7)

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

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

$= \min\{31, 25\}$

$= 25$ — (8)

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

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

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

$= 23$ — (9)

Finally, the total cost of the tour is calculated, which involves three intermediate nodes, that is $|S| = 3$. As $|S| = n-1$, where n is the number of nodes, the process terminates.

~~Finally, the total cost of the tour is calculated which involves three interme~~

Finally, using the equation,

$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{c_{1k} + g(k, V-\{1,k\})\}$$

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

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

$= \min\{35, 40, 43\} = 35.$ — (10)

Now an optimal tour of the graph given has length 35.

A tour of this length can be constructed if we retain with each $g(i, S)$ the value of j that minimizes the right-hand side of equation (1).

Let $J(i, S)$ be this value. Then $J(1, \{2,3,4\}) = 2$ because from subsolution (10), 35 is the min value and $j=2$ is giving that value. Thus the tour starts from 1 and goes to 2.

The remaining tour can be obtained from $g(2, \{3,4\})$. So, $J(2, \{3,4\}) = 4$ because from subsolution (7), 25 is the min value and $j=4$ is giving that value. Thus the next edge is $\langle 2,4 \rangle$.

The remaining tour is for $g(4, \{3\})$. So $J(4, \{3\}) = 3$.

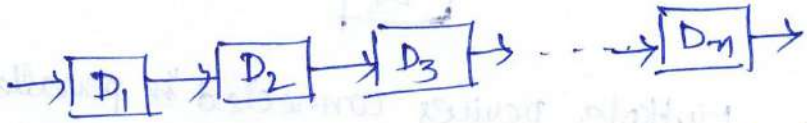
Thus the optimal tour is 1, 2, 4, 3, 1.

Reliability Design :

Reliability means the ability of an apparatus, machine or system to consistently perform its intended or required function or mission, on demand and without degradation or failure.

Reliability design using dynamic programming is used to solve a problem with a multiplicative optimization function.

The problem is to design a system which is composed of several devices connected in series. as shown below.

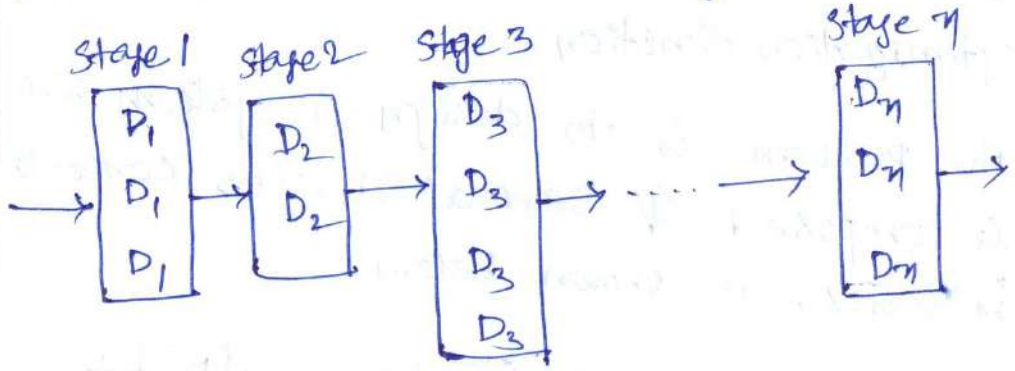


Let r_i be the reliability of device D_i (i.e., r_i is probability that device i will function properly). Then the reliability of the entire system is $\prod r_i$

Even if the individual devices are very reliable (the r_i's are very close to one) the reliability

of the system may not be very good.

Hence, it is desirable to duplicate devices. 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

If stage i contains m_i copies of device D_i, then the probability that all m_i have a malfunction is $(1-r_i)^{m_i}$. Hence the reliability of stage i becomes $1-(1-r_i)^{m_i}$.

Let us assume that the reliability of stage i is given by a function $\phi_i(m_i)$, $1 \le i \le n$. The reliability of the system of stages is $\Pi_{1 \le i \le n} \phi_i(m_i)$

Our problem is to use device duplication to maximize reliability. This maximization is to be carried out under a cost constraint.

Let c_i be the cost of each unit of device i and let c be the maximum allowable cost of the system being designed. Now

$$\text{maximize } \Pi_{1 \le i \le n} \phi_i(m_i)$$

$$\text{subject to } \sum_{1 \le i \le n} c_i m_i \le c$$

$m_i \geq 1$ and integer, $1 \le i \le n$.

$c_i > 0$, each m_i must be in the range $1 \le m_i \le u_i$ where

$$u_i = \left\lfloor \frac{(c + c_i - \sum_{1}^{n} c_j)}{c_i} \right\rfloor$$

The upper bound u_i follows from the observation that $m_j \geq 1$.

An optimal solution $m_1, m_2, \ldots m_n$ is the result of a sequence of decisions, one decision for each m_i.

Let $f_i(x)$ represent the maximum value of $\Pi_{1 \leq j \leq i} \phi(m_j)$ subject to the constraints $\Sigma_{1 \leq j \leq i} c_j m_j \leq x$ and $1 \leq m_j \leq u_j$, $1 \leq j \leq i$.

Then the value of optimal solution is $f_n(c)$.

$$f_n(c) = \max_{1 \leq m_n \leq u_n} \{ \phi_n(m_n) f_{n-1}(c - c_n m_n) \}$$

for any $f_i(x)$ $i \geq 1$ this equation generalizes to

$$f_i(x) = \max_{1 \leq m_i \leq u_i} \{ \phi_i(m_i) f_{i-1}(x - c_i m_i) \}$$

Clearly $f_0(x) = 1$ for all x, $0 \leq x \leq c$. So, this can be solved using an approach similar to that used in knapsack problem.

Let S^i consists of tuples of the form (f, x) where $f = f_i(x)$. f = reliability, x = cost.

There is atmost one tuple for each different x that results from a sequence of decisions

on $m_1, m_2, \ldots m_n$.

The dominance rule (f_1, x_1) iff $f_1 \geq f_2$ and $x_1 \leq x_2$ holds for this problem too. Hence dominated tuples can be discarded from S^i.

S^i_j represents tuples obtainable from S^{i-1} by choosing $m_i = j$.

As in the case of the knapsack problem, a complete dynamic programming algorithm for the reliability problem will use heuristics to reduce the size of the S^is.

There is no need to retain any tuple (f, x) in S^i with x value greater than $c - \sum_{i \leq j \leq n} c_j$ as such a tuple will not leave adequate funds to complete the system.

In addition, we can devise a simple heuristic to determine the best reliability obtainable by completing a tuple (f, x) in S^i. If this is less than a heuristically determined lower bound on the optimal system reliability, then (f, x) can be eliminated from S^i.

Example Problem