Unit - III

Dynamic Programming

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

Examples :-

→ The solution to the knapsack problem can be viewed as the result of a sequence of decisions. We have to decide the values of x_i, $1 \le i \le n$. First we make a decision on x_1, then on x_2, then on x_3, and so on. An optimal sequence of decisions maximizes the objective function $\sum p_i x_i$.

→ One way to find a shortest path from vertex i to vertex j in a directed graph G is to decide which vertex should be the second vertex, which the third, which the fourth, and so on, until vertex j is reached. An optimal sequence of decisions is one that results in a path of least length.

(*)

Dynamic Programming (DP) was invented by Richard Bellman, a prominent U.S. mathematician, in 1950's.

The solutions for the dynamic programming are based on multistage optimizing decisions, on a few common elements.

Here the word 'Programming' does not symbolise computer programming but "planning".

The DP is closely related to divide and conquer technique, where the problem breaks down into smaller subproblems and each subproblem is solved recursively.

The DP differs from divide and conquer in a way that instead of solving subproblems recursively, it solves each of the subproblem only once and stores the solution to the subproblems in a table.

Later on, the solution to the main problem is obtained by these subproblem's solutions.

The steps for achieving Dynamic Programming are:

⇒ Divide, Subproblems

The main problem is divided into several smaller subproblems. In this, the solution of the main problem is expressed in terms of the solution for the smaller subproblems.

Table, Storage

The solution for each subproblem is stored in a Table. So, that it can be used many times whenever required.

⇒ Combine, Bottom-up Computation

The solution to main problem is obtained by combining the solutions of smaller subproblems.

For problem to be solved through DP it must follow the following conditions:

1. Principle of optimality - The principle of optimality states that for solving the problem optimally, its subproblem should be solved optimally.

It should be noted that not all the times each subproblem is solved optimally, so, in that case we should go for the optimal majority.

2. Polynomial breakup - For solving the main problem, the problem is divided into several smaller subproblems and for efficient performance of DP the total number of subproblems to be solved should be atmost a polynomial number.)

All-Pairs Shortest Path :-

Applying single source shortest path to all n nodes is called as All pairs shortest path.

→ Let $G=(V,E)$ be a directed graph with n vertices.

→ Let cost be a cost adjacency matrix for G such that $cost(i,i)=0, 1 \le i \le n$. Then $cost(i,j)$ is the length (or) cost of edge $<i,j>$ if $<i,j> \in E(G)$ and $cost(i,j)=\infty$ if $i \neq j$ and $<i,j> \notin E(G)$.

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

→ The matrix A can be obtained by solving n single-source problems using the algorithm Shortestpath.

→ Let us examine a shortest i to j path in G, such that $i \neq j$. This path originates at vertex i and goes through some intermediate vertices and terminates at vertex j.

→ We can assume that this path contains no cycles. If there is a cycle then this can be deleted without increasing the path length.

→ If k is an intermediate vertex on this shortest path, then the sub paths i to k and from k to j must be shortest paths from i to k and k to j respectively.

→ Otherwise the i to j path is not of minimum length. Hence, the principle of optimality holds.

(*)

Optimal Binary Search Trees (OBS Trees) – (6)

⇒ Given, a fixed set of identifiers, we have to create a binary search tree organization.

→ There may be different binary search trees for the same identifier set which have different performance characteristics.

→ Let us assume that, the given set of identifiers is $\{a_1, a_2, \ldots, a_n\}$ with $a_1 < a_2 < a_3 < \ldots < a_n$.

→ Let $p(i)$ be the probability with which we search for a_i.

→ Let $q(i)$ be the probability that the identifier 'x' being searched for is such that $a_i < x < a_{i+1}$, $0 \le i \le n$.

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

⇒ $\sum_{1 \le i \le n} p(i) + \sum_{0 \le i \le n} q(i) = 1$

⇒ In obtaining a cost function for binary search trees, it is useful to add a external node in place of every empty sub tree in the binary search tree. All other nodes are internal nodes.

If a BST represents 'n' identifiers then there will be exactly 'n' internal nodes and (n+1) external nodes.

57

- Successful search terminates at the internal node at level l, then l comparisions has to be made. Hence, the expected cost contribution from the internal node for a_i is $\boxed{p(i) * level(a_i)}$.
- Unsuccessful search terminates at external nodes. The identifier not in the BST can be partitioned into $n+1$ equivalence classes E_i, $0 \le i \le n$. The class E_0 contains all identifiers x_i such that $x < a_i$.

The class E_i contains all identifiers x such that $a_i < x < a_{i+1}$, $1 \le i \le n$. The class E_n contains all identifiers x such that, $x > a_n$.

For a failure node E_i at level l, only $l-1$ comparisions are made.

$\therefore$ cost contribution is $\boxed{q(i) * (level(E_i) - 1)}$.

$$\therefore \text{Total cost} = \sum_{1 \le i \le n} p(i) * level(a_i) + \sum_{0 \le i \le n} q(i) * (level(E_i) - 1)$$

- All identifiers in the left subtree of 'T' are less than (numerically / alphabetically) the identifier in the root node T.
- All identifiers in the right subtree are greater than the identifier in the root node T.
- The left and right subtrees of 'T' are also binary search tree.

mple - (*)

The identifier set (a_1, a_2, a_3) = (do, if, while)

Probability, $p(i) = q(i) = 1/7$, $\forall i$.

(B.S.T → (n) elements
$(2^n - n)$ B.S.T for)

a) Level 1 → do

Level → E_0

while

if

E_3

Level 3 →

E_1 E_2

Level 4

n internal nodes → (n+1) external nodes

Successful Search → Level.

P(do) = 1/7 × 1 = 1/7

P(while) = 1/7 × 2 = 2/7

P(if) = 1/7 × 3) = 3/7

Total = 6/7

Unsuccessful Search (Level − 1)

E_0 = 1/7 * (2−1) = 1/7

E_1 = 1/7 * (4−1) = 3/7

E_2 = 1/7 * (4−1) = 3/7

E_3 = 1/7 * (3−1) = 2/7

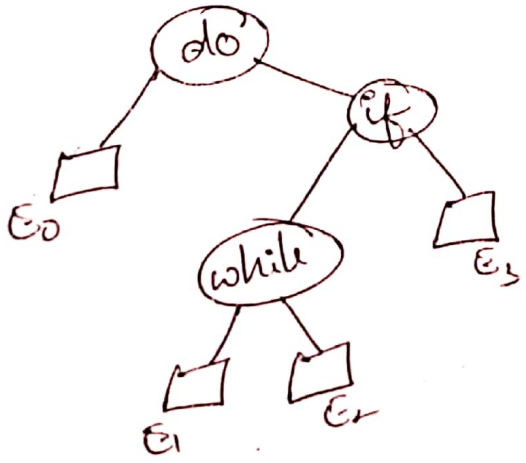
Total = 9/7.

∴ Total cost = 6/7 + 9/7 = 15/7.

No of Possible BST with n nodes

$2^n - n$

b)



iccessful Searches

$p(do) = 1/7 \times 1 = 1/7$

$p(if) = 1/7 \times 2 = 2/7$

$p(while) = 1/7 \times 3 = 3/7$

Total = 6/7

Unsuccessful Searches

$\varepsilon_0 = 1/7 * (2-1) = 1/7$

$\varepsilon_1 = 1/7 * (3-1) = 2/7$

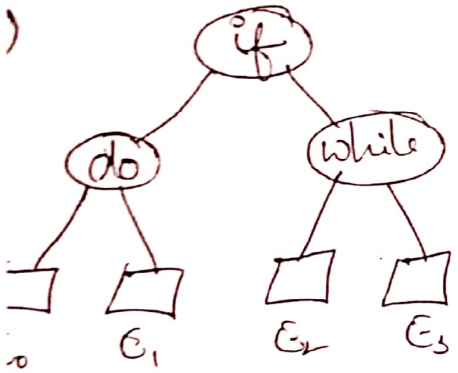
$\varepsilon_2 = 1/7 * (4-1) = 3/7$

$\varepsilon_3 = (1/7) * (4-1) = 3/7$

Total = 9/7

$\therefore$ Total cost = $6/7 + 9/7 = 15/7$ (2.14)

)



Successful Searches

$p(if) = 1/7 \times 1 = 1/7$

$p(do) = 1/7 \times 2 = 2/7$

$p(while) = 1/7 \times 2 = 2/7$

Total = 5/7

Unsuccessful Searches

$\varepsilon_0 = 1/7 \times (3-1) = 2/7$

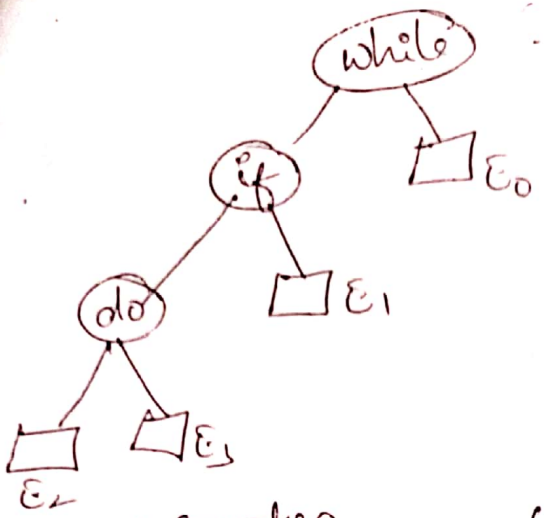
$\varepsilon_1 = 2/7$

$\varepsilon_2 = 2/7$

$\varepsilon_3 = 2/7$.

Total = 8/7.

$\therefore$ Total cost = $5/7 + 8/7 = 13/7$. (1.85)



Successful Searches

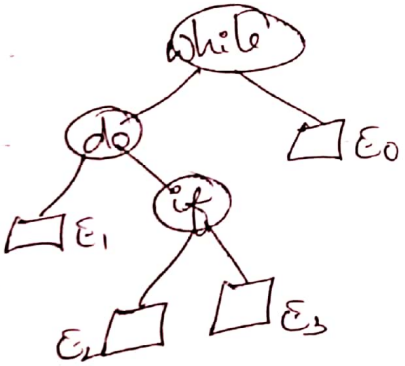
P(while) = 1/7 × 1 = 1/7
P(if) = 1/7 × 2 = 2/7
P(do) = 1/7 × 3 = 3/7
Total = 6/7

Unsuccessful Searches

ε_0 = 1/7
ε_1 = 2/7
ε_2 = 3/7 = ε_3
Total = 9/7

∴ Total cost = 15/7.

b)



Successful Searches

P(while) = 1/7
P(do) = 2/7
P(if) = 3/7
Total = 6/7

Unsuccessful Searches

ε_0 = 1/7
ε_1 = 2/7
ε_2 = ε_3 = 3/7.
Total = 9/7.

∴ Total cost = 15/7.

∴ Tree (c) is the OBS Tree with the (min.) cost (13/7).

> $n=4$ $(a_1, a_2, a_3, a_4) = (do, if, int, while)$

Successful Searches $P(i)$ $(P_1\ P_2\ P_3\ P_4) = (3\ \ 3\ \ 1\ \ 1)$

Unsuccessful Searches $q(i)$ $(q_0\ q_1\ q_2\ q_3\ q_4) = (2\ \ 3\ \ 1\ \ 1\ \ 1)$

Formulae -

if $i = j$

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

$$c(i,j) = r(i,j) = 0.$$

w → weight
c → cost
r → root.

$\neq j$

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

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

$$r(i,j) = \min k.$$

3 $(a_1\ a_2\ a_3) = (do, if, while)$ (9)

$(p_1\ p_2\ p_3) = (0.5, 0.1, 0.05)$ $(q_0, q_1, q_2, q_3) = (0.15, 0.1, 0.05, 0.05)$

$i \longrightarrow$

	0	1	2	3
$j-i=0$, $j=i$, 0	$w_{00}=0.15$ $c_{00}=0$ $r_{00}=0$	$w_{11}=0.1$ $c_{11}=0$ $r_{11}=0$	$w_{22}=0.05$ $c_{22}=0$ $r_{22}=0$	$w_{33}=0.05$ $c_{33}=0$ $r_{23}=0$
$j-i=1$, $j=i+1$, 1	$w_{01}=0.75$ $c_{01}=0.75$ $r_{01}=1$ $r=1$	$w_{12}=0.25$ $c_{12}=0.25$ $r_{12}=2$ $r=2$	$w_{23}=0.15$ $c_{23}=0.15$ $r_{23}=3$ $r=3$	
$j-i=2$, $j=i+2$, 2	$w_{02}=0.9$ $c_{02}=1.15$ $r_{02}=1$ $r=1,2$	$w_{13}=0.35$ $c_{13}=0.5$ $r_{13}=2$ $r=2,3$		
$j-i=3$, $j=i+3$, 3	$w_{03}=1$ $c_{03}=1.5$ $r_{03}=$ ~~3~~ $r=(1,2,3)$			

$c_{ij} = 1.7$ ⓒ

$c(i,j) \Rightarrow r(i,j) = 0$

W → weight
c → cost
r → root

$w(0,1) = p(1) + q(1) + w(0,0)$ ($j \neq 1$)

$= 0.5 + 0.1 + 0.15 = 0.75$ ($i \neq 1$)

$$c(0,1) = \min_{0<k\leq 1} \{c(0,0) + c(1,1)\} + w(0,1)$$

$k = \{1\}$

$= \min\{0+0\} + 0.75 = 0.75$

$r_{01} = 1$.

60

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

$= 0.1 + 0.05 + 0.1$

$= 0.25$

$c(1,2) = \min_{1<k\le j} \{c(1,1) + c(2,2)\} + w(1,2)$

$k = \{2\}$

$= 0.25$

$r(1,2) = 2$

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

$= 0.05 + 0.05 + 0.05$

$= 0.15$

$c(2,3) = \min_{1<k\le 3} \{c(2,2) + c(3,3)\} + w(2,3)$

$k = \{3\}$

$= 0 + 0.15 = 0.15$

$r(2,3) = 3.$

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

$= 0.1 + 0.05 + 0.75$

$= 0.9$

$c(0,2) = \min_{k=\{1,2\}} \begin{cases} c(0,0) + c(1,2) & k=1 \\ c(0,1) + c(2,2) & k=2 \end{cases} + w(0,2)$

$= \min \begin{cases} 0 + 0.25 \\ 0.75 + 0 \end{cases} + 0.9$

$= 0.25 + 0.9$

$= 1.15$

$r(1,2) = \min\{1,2\} = 1$

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

$= 0.05 + 0.05 + 0.25$

$= 0.35$

$c(1,3) = \min_{x=\{2,3\}} \begin{cases} c(1,1) + c(2,3) & k=2 \\ c(1,2) + c(3,3) & k=3 \end{cases} + w(1,3)$

$= \min \begin{cases} 0 + 0.15 \\ 0.25 + 0 \end{cases} + 0.35$

$= 0.15 + 0.35 = 0.5$

$r(1,3) = \{2$

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

$= 0.05 + 0.05 + 0.9$

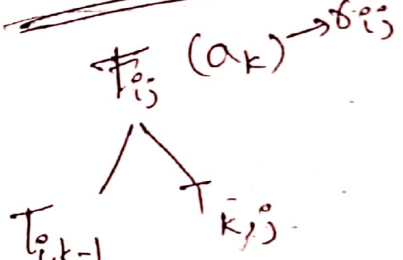
$= 1$

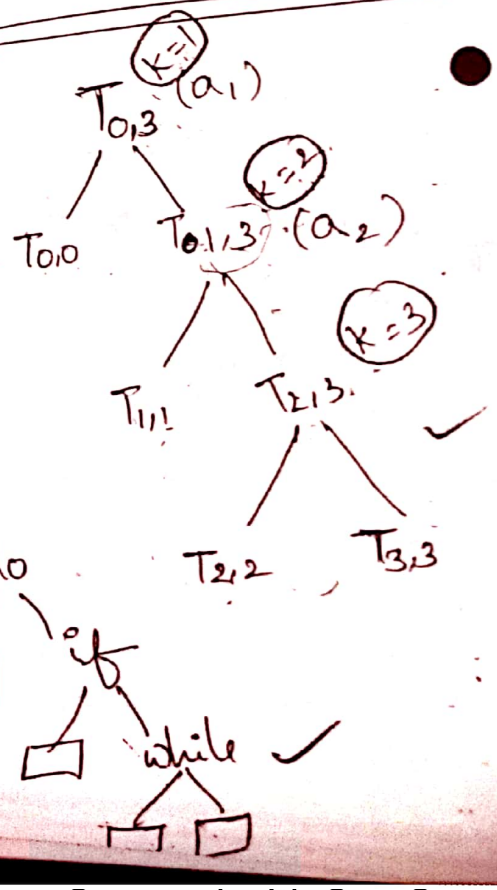
$c(0,3) = \min_{k=\{1,2,3\}} \begin{cases} c(0,0) + c(1,3) & k=1 \\ c(0,1) + c(2,3) & k=2 \\ c(0,2) + c(3,3) & k=3 \end{cases} + w(0,3)$

$= \min \begin{cases} 0 + 0.5 \\ 0.75 + 0.15 \\ 1.15 + 0 \end{cases} + 1$

$= 0.5 + 1 = 1.5$

$r(0,3) = 1$





(1) $(a_1\ a_2\ a_3\ a_4)$

$n=4$ (do if int while)

p_1	p_2	p_3	p_4	q_0	q_1	q_2	q_3	q_4
3	3	1	1	2	3	1	1	1

(18)

→ Consider 4 elements $a_1 < a_2 < a_3 < a_4$ with $q_0 = 1/8$, $q_1 = 3/16$, $q_2 = q_3 = q_4 = 1/16$ (count, float, if, while)

$p_1 = 1/4$, $p_2 = 1/8$, $p_3 = p_4 = 1/16$.

Construct the table of values of w_{ij}, c_{ij} & r_{ij} computed by the algorithm to compute the roots of the OBST.

→ $n=4$, $(p_1\ p_2\ p_3\ p_4) = (1\ 4\ 2\ 1)$

$(q_0\ q_1\ q_2\ q_3\ q_4) = (4\ 2\ 4\ 1\ 1)$

$n=7$

$(p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7) = (0.04, 0.06, 0.08, 0.02, 0.1, 0.12, 0.14)$

$q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7) = (0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05)$.

Using OBST Algorithm compute $w(i,j)$, $r(i,j)$ & $c(i,j)$ $0 \le i < j < 4$, for

$n=4$ (a_1, a_2, a_3, a_4) = (count, float, if, while). 61

$p(1) = 1/20$, $p(2) = 1/5$, $p(3) = 1/10$, $p(4) = 1/20$.

$q(0) = 1/5$, $q(1) = 1/10$, $q(2) = 1/5$, $q(3) = 1/20$, $q(4) = 1/20$.

Using $r(i,j)$'s, construct the OBST.

```
Algorithm OBST(p,q,n)
// Given n distinct identifiers a1<a2<...<an and probabilities
// p[i], 1≤i≤n, and q[i], 0≤i≤n, this algorithm computes the
// cost c[i,j] of OBST t_ij for identifiers a_{i+1}, ..., a_j. It
// also computes r[i,j], the root of t_ij. w[i,j] is the weight of t_ij.
{
  for i:=0 to n-1 do
  {
    w[i,i]:=q[i]; r[i,i]:=0; c[i,i]:=0.0;
    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]:=0.0;
  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;
    }
  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;
}
```

Knapsack - (*)

The solution to the 0/1 Knapsack problem is obtained by making a sequence of decisions on the variables $x_1, x_2, \ldots, x_n$.

Decision on variable x_i involves determining which value 0 or 1 is to be assigned to x_i.

The two possibilities are,

1. The capacity remaining in the knapsack is m & no profit has occured.
2. The capacity remaining in the knapsack is $(m-w_n)$ & the profit p_n has occured.

Solving 0/1 Knapsack problem involves the following operations.

⇒ Addition operation -

$$0 \rightarrow S_1^i = S^i + (P_{i+1}, w_{i+1})$$

⇒ Merging operation -

$$(2) \rightarrow S^{i+1} = \text{Merge}\,(S^i, S_1^i)\ (\text{or})\ (S^i \cup S_1^i)$$

65

⇒ Purging Rule (or) Dominance Rule -

- Take any two sets S^i & S_1^i consisting of pairs (P_j, w_j) & (P_k, w_k). The purging rule states that if $P_j \le P_k$ & $w_j \ge w_k$ then (P_j, w_j) will be deleted. This rule must be applied to the merging operation.

Rule - If (P,W) is the last tuple in S^n, a set of 0 & 1, values for x_i's can be determined as follows.

1. If $(P,W) \in S^{n-1}$ then set $x_n = 0$
2. If $(P,W) \notin S^{n-1}$ then $(P-P_n, W-W_n) \in S^{n-1}$ then set $x_n = 1$.

Example - Consider $n=3$, $m=6$ (m = knapsack size)

$(P_1$	P_2	$P_3)$	$(W_1$	W_2	$W_3)$
(1	2	5)	(2	3	4)

$0 \le i \le n-1 \Rightarrow 0 \le i \le 2$

$\therefore i = \{0,1,2\}$.

Let $S^0 = \{0,0\}$ Initially.

i=0

$\rightarrow S_1^0 = S^0 + (P_1, W_1) = \{(0,0)\} + (1,2) = \{1,2\}$

$\rightarrow S^1 = S^0 \cup S_1^0 = \{(0,0)\} \cup \{(1,2)\} = \{(0,0),(1,2)\}$.
(P_j W_j ; P_k W_k)

$P_j \le P_k \Rightarrow 0 \le 1$ ✓

$W_j \ge W_k \Rightarrow 0 \ge 2$ ✗

i=1

$S_1^1 = S^1 + (P_2, W_2) = \{(0,0),(1,2)\} + (2,3) = \{(2,3),(3,5)\}$ (set-Addition)

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

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

$0 \le 2$ ✓	$1 \le 2$ ✓
$0 \ge 3$ ✗	$2 \ge 3$ ✗
$0 \le 3$ ✓	$1 \le 3$ ✓
$0 \ge 5$ ✗	$2 \ge 5$ ✗

$S_1^2 = S^2 + (P_3, W_3) = \{(0,0)(1,2)(2,3)(3,5)\} + \{(5,4)\}$

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

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

$0 \le 5$ ✓	$1 \le 5$ ✓	$2 \le 5$ ✓	$3 \le 5$ ✓
$0 \ge 4$ ✗	$2 \ge 4$ ✗	$3 \ge 4$ ✗	$5 \ge 4$ ✓
$0 \le 6$ ✓	$1 \le 6$ ✓	$2 \le 6$ ✓	$3 \le 6$ ✓
$0 \ge 6$ ✗	$2 \ge 6$ ✗	$3 \ge 6$ ✗	$5 \ge 6$ ✗
$0 \le 7$ ✓	$1 \le 7$ ✓	$2 \le 7$ ✓	$3 \le 7$ ✓
$0 \ge 7$ ✗	$2 \ge 7$ ✗	$3 \ge 7$ ✗	$5 \ge 7$ ✗
$0 \le 8$ ✓	$1 \le 8$ ✓	$2 \le 8$ ✓	$3 \le 8$ ✓
$0 \ge 9$ ✗	$2 \ge 9$ ✗	$3 \ge 9$ ✗	$5 \ge 9$ ✗

Purging Rule satisfies the pair (3,5). So (3,5) is deleted from S^3.

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

(annotations: not (6) exceeds size m=6; not > 6 so delete (7,7),(8,9))

- As $m=6$, so w should not be $> m$.

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

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

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

$S^3 = \{(0,0),(1,2),(2,3)(5,4),(6,6),~~(7,7),(8,9)~~\}$ (66)

$(P,W) = (6,6) \in S^2$ ✗

$(P,W) = (P - P_n, w - w_n) = (6-5, 6-4)$

$= (1,2) \in S^2$. [Rule:]

all tuple: $x_3 = 1$

:2

$(P, w)\ \epsilon = (1,2) \in S^1$

$x_2^1 = 0.$

:1

$(P, w) = (1,2) \notin S^0$

$(P, w) = (P - P_1, w - w_1) = (1-1, 2-2)$

$= (0,0) \in S^0$ ✓

$x_1 = 1$

$\therefore (x_1\ x_2\ x_3) = (1\ 0\ 1)$

Maximum Profit is,

$\Sigma P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$

$= 1 \times 1 + 2 \times 0 + 5 \times 1$

$= 1 + 0 + 5 = 6$

P_1	P_2	P_3
1	2	5

Optimal Solution is $(x_1, x_2, x_3) = (1,0,1)$

Max. Profit is $\Sigma P_i x_i = 6.$

(10)

Examples

(16)

$n=4,\ m=40$

$(P_1\ P_2\ P_3\ P_4) = (11\ 21\ 31\ 33)$ $(w_1\ w_2\ w_3\ w_4) = (2\ 11\ 22\ 15)$

$n=3 \quad m=6$

$(P_1\ P_2\ P_3) = (1\ 2\ 4)$ $(w_1\ w_2\ w_3) = (2\ 3\ 3)$

$n=6 \quad m=165$

$(P_1\ P_2\ P_3\ P_4\ P_5\ P_6)$ $(w_1\ w_2\ w_3\ w_4\ w_5\ w_6)$

$(100\ 50\ 20\ 10\ 7\ 3)$ $(100\ 50\ 20\ 10\ 7\ 3)$

$n=3 \quad m=20$

P_1	P_2	P_3	w_1	w_2	w_3
25	24	15	18	15	10

$n=4 \quad m=21$

P_1	P_2	P_3	P_4	w_1	w_2	w_3	w_4
2	5	8	1	10	5	6	9

—— X ——

```
PW = record {float P; float w;}
Algorithm DKnap(p, w, x, n, m)
{ // pair[] is an array of PW's.
   b[0] := 1; pair[1].P := pair[1].w := 0.0; // S^0
   t := 1; h := 1; // Start and end of S^0
   b[1] := next := 2; // Next free spot in pair[]
   for i := 1 to n-1 do
   { // Generate S^i.
      k := t;
      u := Largest(pair, w, t, h, i, m);
```

```
for j:=1 to n do
{ //Generate S^{i-1} and merge
  PP:=pair[j].P+P[i]; WW:=pair[j].W+W[i];
        //(PP,WW) is the next element in S^{i-1}.
  while ((k<=h) and (pair[k].W<=WW)) do
  {
     pair[next].P:=pair[k].P;
     pair[next].W:=pair[k].W;
     next:=next+1; k:=k+1;
  }
  if ((k<=h) and (pair[k].W=WW)) then
  { if PP<pair[k].P then PP:=pair[k].P;
    k:=k+1;
  }
  if PP>pair[next-1].P then
  {
    pair[next].P:=PP; pair[next].W:=WW;
    next:=next+1;
  }
  while((k<=h) and (pair[k].P<=pair[next-1].P))
      do k:=k+1;
}
//Merging in remaining terms from S^{i-1}.
while (k<=h) do
{ pair[next].P:=pair[k].P; pair[next].W:=pair[k].W;
  next:=next+1; k:=k+1;
}
//Initialize for S^{i+1}
t:=h+1; h:=next-1; b[i+1]:=next;
}
TraceBack(n,W,pair,x,m,n);
```

Travelling Salesperson Problem :- (*)

Let $G=(V,E)$ be a directed graph with edge costs c_{ij}. The variable c_{ij} is defined such that $c_{ij}>0$ for all $i \& j$ and $c_{ij}=\infty$ if $<i,j> \notin E$.

- Let $|V|=n$ and assume $n>1$
- A tour of G is a directed simple cycle that includes every vertex in V.
- The cost of a tour is the sum of the cost of edges on the tour.
- The TSP is to find a tour of minimum cost.
- We can say the tour as a simple path that starts and ends at vertex 1.
- Every tour consists of an edge $<1,k>$ for some $k \in V-\{1\}$ and a path from vertex k to vertex 1.
- The path from vertex k to vertex 1 goes through each vertex in $V-\{1,k\}$ exactly once.
- If the tour is optimal, then the path from k to 1 must be a shortest k to 1 path going through all vertices in $V-\{1,k\}$.
- Hence, the principle of optimality holds.
- Let $g(i,S)$ be the length of a 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 length of an optimal salesperson tour.

From the principle of optimality it follows that,

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

$$g(i, S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \} \rightarrow (2)$$

» Eq^n (1) can be solved, if we know $g(k, V-\{1,k\})$ for all choices of k.

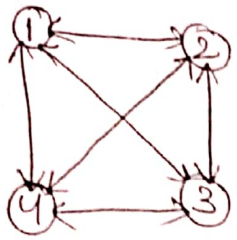
» The g values can be obtained by using eq^n (2).

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

Hence, we can use eq^n (2) to obtain $g(i, S)$ for all S of size 1. Then we can obtain $g(i, S)$ for S with $|S| = 2$ and so on.

When $|S| < n-1$, the values of i and S for which $g(i, S)$ is needed are such that $i \neq 1$, $1 \notin S$, and $i \notin S$.

Ple - Consider the directed graph and the edge length given by,



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

$n = 4$

$i = \{1, 2, 3, 4\}$

Starting Node $i = 1$

$i = \{2, 3, 4\}$

$S \rightarrow$ set of elements

$|S| \rightarrow$ length of set

$g(i, \phi) = C_{i1}$

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

$g(i, S) = \min_{j \in S} \{C_{ij} + g(j, S - \{j\})\}$ $\quad |S| < n-1 \Rightarrow |S| < 3$

$|S| = 1$

$= 2$ $\quad S = \{1, 2, 3, 4\}$ $\quad 1 \notin S,\ i \notin S,\ |S| = 1$

$1 \in S$ ✓ $\quad 2 \in S$ ✓

$S = \{3, 4\}$ But $|S| = 1$.

$\rightarrow S = \{4\}$ & $i = 2$ & $j = 4$

$\Rightarrow S = \{3\}$ & $i = 2$

$j \in S$ i.e. $j = 3$

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

$= \min_{j=4} \{10 + 8\} = 18$

$g(2, \{3\}) = \min_{j=3} \{C_{23} + g(3, \{3\} - 3)\}$

$= \min_{j=3} \{9 + g(3, \phi)\} = 15$

$= 3$ Here, we have to select the set S, that satisfies

$1 \notin S,\ i \notin S$ & $|S| = 1$

$S = \{2, 4\}$

$\rightarrow S = \{4\},\ j = 4$

$\rightarrow S = \{2\},\ j = 2$

$g(3, \{2\}) = \min_{j=2} \{C_{32} + g(2, \phi)\}$ $\quad g(3, \{4\}) = \min_{j=4} \{C_{34} + g(4, \phi)\}$

$= 18$ $\quad = 20$

4 $S = \{2,3\}$

$S = \{2\}, j = 2 \rightarrow S = \{3\}, j = 3$

$g(4,\{2\}) = \min_{j=2}\{c_{42} + g(2,\phi)\}$ $\quad g(4,\{3\}) = \min_{j=4}\{c_{43} + g(3,\phi)\}$

$= 13$ $\quad = 15$

$|S| = 2$ $\quad i = \{2,3,4\}$

$i=2$

$S = \{3,4\}$

$$g(2,\{3,4\}) = \min_{j\in\{3,4\}} \begin{cases} c_{23} + g(3,\{3,4\}-3) & j=3 \\ c_{24} + g(4,\{3,4\}-4) & j=4 \end{cases}$$

$$= \min_{j\in\{3,4\}} \begin{cases} c_{23} + g(3,\{4\}) & j=3 \\ c_{24} + g(4,\{3\}) & j=4 \end{cases}$$

$$= \min_{j\in\{3,4\}} \begin{cases} 9+20 & j=3 \\ 10+15 & j=4 \end{cases} = 25 \ (j=4)$$

$i=3$ $\quad S = \{2,4\}$

$$g(3,\{2,4\}) = \min_{j\in\{2,4\}} \begin{cases} c_{32} + g(2,\{2,4\}-2) & j=2 \\ c_{34} + g(4,\{2,4\}-4) & j=4 \end{cases}$$

$$= \min_{j\in\{2,4\}} \begin{cases} c_{32} + g(2,\{4\}) & j=2 \\ c_{34} + g(4,\{2\}) & j=4 \end{cases}$$

$$= \min_{j\in\{2,4\}} \begin{cases} 13+18 & j=2 \\ 12+13 & j=4 \end{cases} = 25 \ (j=4)$$

$s = \{2,3\}$

$$g(1,\{2,3\}) = \min_{j\in\{2,3\}} \begin{cases} c_{12} + g(2,\{3\}) & j=2 \\ c_{13} + g(3,\{2\}) & j=3 \end{cases}$$

$$= \min_{j\in\{2,3\}} \begin{cases} 8+15 & j=2 \\ 9+18 & j=3 \end{cases} = 23 \quad (j=2)$$

$$g(1, V-\{1\}) = \min_{2\le k\le 4} \{ c_{1k} + g(k, V-\{1,k\}) \}$$

$$V = \{1,2,3,4\}$$

$$g(1,\{2,3,4\}) = \min_{\substack{2\le k\le 4 \\ k=\{2,3,4\}}} \begin{cases} c_{12} + g(2,\{3,4\}) & k=2 \\ c_{13} + g(3,\{2,4\}) & k=3 \\ c_{14} + g(4,\{2,3\}) & k=4 \end{cases}$$

$$= \min \begin{cases} 35 & k=2 \\ 40 & k=3 \\ 43 & k=4 \end{cases} = 35 \quad (k=2)$$

∴ The min. cost of tour = 35 And the tour is $1 \to 2 \to 4 \to 3 \to 1$.

eliability Design :-

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

Fig: n devices, D_i, $1 \le i \le n$, connected in series.

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

It may happen that even if reliability of individual devices is very good but reliability of entire system may not be 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.

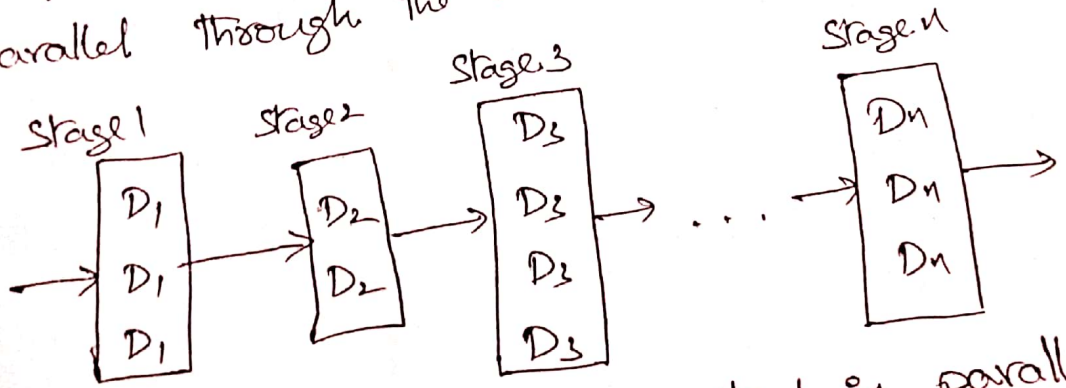


fig: Multiple devices connected in parallel in each stage

The switching circuits determine which devices in any given group are functioning properly. They then make use of one such device at each stage.