UNIT – III

DESIGN AND ANALYSIS OF ALGORITHMS

DYNAMIC PROGRAMMING

BASIC METHOD

- Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub-problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.
- Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler sub-problems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its sub-problems.
- Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems
- Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution

BASIC METHOD

So we can say that –

- The problem should be able to be divided into smaller overlapping sub-problem.
- An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Dynamic algorithms use Memorization.

- A given problem has Optimal Substructure Property, if the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.
- For example, the Shortest Path problem has the following optimal substructure property –
- If a node **x** lies in the shortest path from a source node **u** to destination node **v**, then the shortest path from **u** to **v** is the combination of the shortest path from **u** to **x**, and the shortest path from **x** to **v**.

Steps of Dynamic Programming Approach

Dynamic Programming algorithm is designed using the following four steps –

- Characterize the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution, typically in a bottom-up fashion.
- Construct an optimal solution from the computed information.

APPLICATIONS OF DYNAMIC PROGRAMMING

0/1 KNAPSACK PROBLEM

- We are given n objects and a knapsack(bag). Object i has a weight w_i and capacity m.
- If a solution $x_i, x_i = 0 \; or \; 1$, of object i is placed into the knapsack, then the profit $p_i x_i$ is earned.
- The objective is to obtain a filling of the knapsack that maximizes the total profit earned.
- We require the total weight of the chosen objects to be at most m.
- Hence, the objective of this algorithm is to

$$\text{Maximize } \sum_{1 \le i \le n} p_i x_i$$

$$\text{Subject to } \sum_{1 \le i \le n} w_i x_i <= \text{m}$$

$$\text{and } x_i = 0 \text{ or } 1,\ 1 \le i \le n$$

The profits and weights are positive numbers.

0/1 KnapSack Problem

Ex :1- n=4,m=8 P={1,2,5,6} and W={2,3,4,5}

Tabular Method:

		Capacity / Item	0	1	2	3	4	5	6	7	8
P	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

0/1 KnapSack Problem

- For the zero row, no item is selected and so no weight is included into knapsack. So fill first row with all 0's and first column with all 0's
- For the first row consider first element. First element weight is 2. so fill second row second column with profit of the first item i.e 1. Only first element can be selected. So all the remaining columns values are 1 only, and left side column with previous value.
- For the second row select second element. Second element weight is 3. so fill 3rd column in the 3 row with profit of the second item i.e 2. fill all the left side colmns with previous values. Here we must select both the items. So first two items weight is 5. total profit of first two item is 3. so fill column 5 with 3. fillleft side columns with previous valus. And right side columns with 3, because we can select first two items only.
- For the third row, select 3rd item. Its weight is 4. so fill 4th column with its profit i.e 5. here we select first 3 items. But we may not select all three. But we must identify the combinations with 3rd item.

If we select 3rd and 1st items, total weight is 6, so fill 6th column with total profit of 1st and 3rd items i.e 6 , in the same way select 3rd and 2nd items, total weight is 7 , so fill 7th column with total profit of 2nd and 3rd items, i.e 7.

0/1 Knapsack Problem

- Fill the last column with 7 only, because we cannot select remaining items.
- For the 4^{th} row, select 4^{th} item , the weight of the item is 5, so fill 5^{th} column in 4^{th} row with the profit of the 4^{th} item, i.e 6. all the previous columns with the old values. Now select remaining items alongwith 4^{th} itam.

  - select 4^{th} item with 1^{st} item , the total weight of these two items is 7, so fill 7^{th} column with the total profit of these two items i.e 7.

  - select item 4 with second item, the total weight of these two items is 8, so fillthe 8^{th} column with these two items profit, i.e 8.

  - 6^{th} column with 5^{th} column value.
- After filling the entire row, now construct the solutionx1, x2, x3 and x4.

0/1 Knapsack Problem

- Formula for filling all the rows:

  V[i, w] =max { V[i-1, w], V[i-1,w-w[i]]+p[i] }

- V[4, 1] = max{ V[3, 1], V[3, 1-5] +6 }

  = max{ 0, v[3, -4] + 6}

  undefined..

So upto w=4 take the same values as previous row.

- V[4, 5] = max{ V[3, 5], V[3, 5-5] +6 }

  = max{ 5, v[3, 0] + 6}

  = max{ 5, 0 + 6} = max{ 5, 6} = 6

- V[4, 6] = max{ V[3, 6], V[3, 6-5] +6 }

  = max{ 6, v[3, 1] + 6}

  = max{ 6, 0 + 6} = max{ 6, 6} = 6

0/1 Knapsack Problem

- V[4, 7] = max{ V[3, 7], V[3, 7-5] +6 }
  = max{ 7, v[3, 2] + 6}
  = max{ 7, 1 + 6} = max{ 7, 7} = 7
- V[4, 8] = max{ V[3, 8], V[3, 8-5] +6 }
  = max{ 7, v[3, 3] + 6}
  = max{ 6, 2 + 6} = max{ 6, 8} = 8

0/1 Knapsack Problem

- Select the maximum profit value , i.e. 8, which is there in 4th row, check whether 8 is there in 3rd row or not. Value is not there, means 4th row is included, **so x4=1**.

  - 4th row profit is 6. remaining profit is 8-6=2.

- 2 is there in row 3, check whether 2 is there in row 2 or not. Value is there in 2nd row also , means 3rd item is not included. **X3=0**.
- So 2 is there in row 2, check whether 2 is there in row 1 or not. No, value 2 is not there in row 1. so second item is included, **x2=1**.

  - so the remaining profit is 2-2=0

- 0 is there in row 1, check whether 0 is there in row 0 or not, yes row zero contain 0, so item 1 is not included, **x1=0.**
- So the solution is : **x1=0, x2=1, x3=0, x4=1.**
- Total profit obtained is **= p1 * x1+ p2 * x2 + p3 * x3 + p4 * x4**

  = 1 *0 + 2 * 1 + 5 * 0 + 6 * 1= 8

Knapsack filled with weight = 2*0 + 3*1 + 4*0 + 5*1 = 8

0/1 Knapsack Problem

Ex : 2 – n=3,m=6 P={1,2,5} and W={2,3,4}

Sets Method: Notice that $S^0 = \{(0,0)\}$. We can compute S^{i+1} from S^i by first computing $S^i_1 = \{(P,W) \mid (P-p_i, W-w_i) \in S^i\}$

- Now S^{i+1} can be obtained by merging S^i and S^i_1.
- If S^{i+1} is containing two pairs (p_j, w_j) and (p_k, w_k) with the property of $p_j \leq p_k$ and $wj \geq w_k$ then the pair (p_j, w_j) can be discarded according to purging or discarding rule. ((p_j, w_j) is dominating (p_k, w_k)). ---- > Dominance rule.

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

$S^0_1 = \{ (0 + 1, 0 + 2) \} = \{(1, 2)\}$ --- { add P_1, w_1 to all the pairs in S^0 }

$S^1 = S^0 \cup S^0_1 = \{(0,0)\} \cup \{(1, 2)\} = = \{ (0,0) , (1, 2) \}$ -No pair is dominating other pairs.

$S^1_1 = \{ (0+2, 0+3) , (1+2, 2+3) \}$ ---- { add P_2, w_2 to all the pairs in S^1 }

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

$S^2 = S^1 \cup S^1_1 = \{ (0,0) , (1, 2) \} \cup \{ (2,3) , (3,5) \}$

$S^2 = \{ (0,0) , (1, 2), (2,3) , (3,5) \}$ – No pair is dominating other pairs

0/1 Knapsack Problem

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

  S^2_1 = { (0+5, 0+4) , (1+5 , 2+4), (2+5, 3+4), (3+5, 5+4) }

  = { (5, 4), (6, 6), (7, 7), (8, 9) } — { add P_3, w_3 to all the pairs in S^2 }

$S^3 = S^2 \cup S^2_1$ = { (0,0) , (1, 2), (2,3) , (3,5) } U { (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) }

- In the above set (3,5) is dominating (5, 4) , so according to purging or dominance rule (3, 5) can be purged or deleted
- Also (7,7) and (8,9) pairs are also deleted because the weights are exceeding the knapsack capacity.

The resultant Set is : S^3 = {(0,0) , (1, 2), (2,3) , (5, 4), (6, 6) }

0/1 Knapsack Problem

- **Constructing Solution :**

- select the last pair in the last set i.e. (6, 6) $\in S^3$, check if this pair is there in S^2 . If it is there in S^2 , third element is not included.

No, (6, 6) does not belonging to S^2 . So 3rd item is included. $x_3=1$. for the next solution subtract P_3, w_3 from the last pair i.e. (6, 6).

The resultant pair is (6 – 5, 6 – 4) , i.e. (1,2)

- (1,2) is there in S^2 , check if this pair is there in S^1 on not.

Yes, (1, 2) is there in S^1 , so 2nd item is not included, $x_2=0$.

- (1,2) is there in S^1 , check if this pair is there in S^0 , No this pair is not there in S^0 , so first item is included, $x_1=1$.

- The solution is : $x_1=1, x_2=0, x_3=1$.
- Total profit gained is : **1*1 + 2*0 + 5*1 = 6**
- Total weight included is : **2*1 + 3*0 + 4*1 = 6**

0/1 Knapsack Problem

Algorithm DKP (P, w, n, m)

{

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

for i := 1 to n-1 do

{

$S_1^{i-1} = \{(P,W) \mid (P-P_i, W-W_i) \in S^{i-1} \text{ and } W \le m\}$

$S^i = \text{MergePurge}(S^{i-1}, S_1^{i-1})$;

}

$(P_x, W_x) :=$ last pair in S^{n-1};

$(P_y, W_y) := (P' + P_n, W' + W_n)$ where W' is the largest W in any pair in S^{n-1} such that $W + W_n \le m$.

if ($P_x > P_y$) then $x_n := 0$;

else

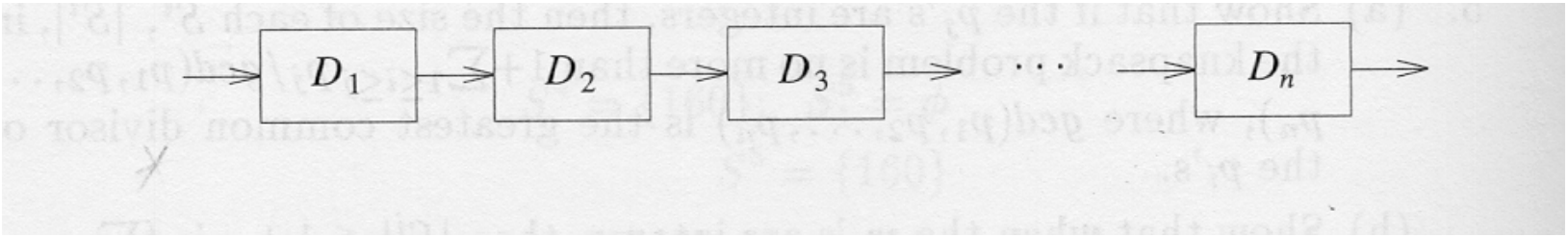
$x_n := 1$.

TraceBack ($x_{n-1}, x_{n-2}, \ldots, x_1$);

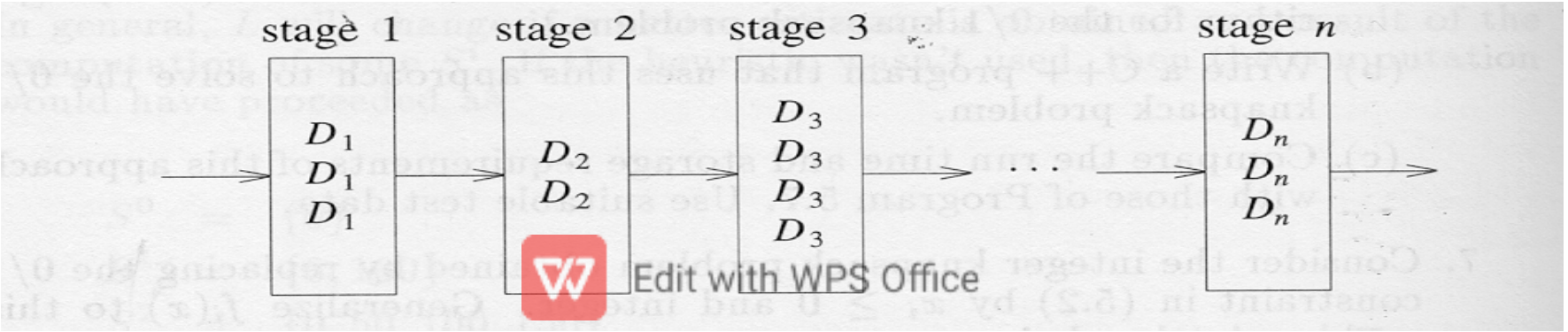
}

RELIABILITY DESIGN

- 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.



- Let ri; be the reliability of device Di; (i.e. ri; is the probability that device i will function properly). Then, the reliability of the entire system is πri . Even if the individual devices are very reliable (the ri 's are very close to one), the reliability of the system may not be very good.
- Ex : if n=10 and ri=0.99, 1<=i<=10, πri = 0.904.
- Hence it is desirable to duplicate the devices. Multiple copies of the same device type are connected in parallel.



RELIABILITY DESIGN

- If stage i contains m_i copies of devise D_i , then the probability that all m_i have a malfunction is $(i- r_i)^{mi}$. Hence the reliability of stage i becomes $1- (1- r_i)^{mi}$. Thus if $r_i= 0.99$ and $m_i=2$, then the stage reliability becomes $1-(1-0.99)^2 = 0.9999$.
- In any practical situation the stage reliability is little less than $1- (1- r_i)^{mi}$. because the switching circuits themselves are not fully reliable. Also the failure of copies of the same device may not be fully independent.
- Let us assume that the reliability of the stage i is given by the function $\varphi_i (m_i)$, $i \leq n$. The reliability of the system of stages is given by $\pi\ \varphi_i (m_i)$, $1 \leq i \leq n$.
- Our problem is to use device duplication to maximize reliability. the maximization is carried out under a cost constraint. Let c_i be the cost of each unit of device type I and let C be the maximum allowable cost of the system being designed.
- The problem statement is :

$$\text{Maximize } \pi\ \varphi_i (m_i),\ 1 \leq i \leq n$$

$$\text{Subject to } \sum_{i=0}^{n} c_i\, mi \leq C$$

$$m_i \geq 1 \text{ and } 1 \leq i \leq n$$

RELIABILITY DESIGN

Ex: Design a 3-stage system with device types D_1, D_2, D_3. The costs are \$30, \$15 and \$20 respectively. The cost of the system is to be no more than \$105. The reliability of each device type is 0.9, 0.8 and 0.5.

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

Ans. $c_1 = 30,\ c_2 = 15,\ c_3 = 20$

$r_1 = 0.9,\ r_2 = 0.8\quad r_3 = 0.5$

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

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

$$u_1 = \lfloor (105 + 30 - 65) / 30 \rfloor$$

$$= \lfloor (70/30) \rfloor$$

$$= 2$$

$$u_2 = \lfloor (105 + 15 - 65) / 15 \rfloor = \lfloor (120 - 65) / 15 \rfloor$$

$$= \left\lfloor \frac{55}{15} \right\rfloor = 3$$

$$u_3 = \lfloor (105 + 20 - 65) / 20 \rfloor = \left\lfloor \frac{125 - 65}{20} \right\rfloor$$

$$= \left\lfloor \frac{60}{20} \right\rfloor = 3$$

RELIABILITY DESIGN

using s^i_j to represent all tuples obtain from s^{i-1}

by choosing $m_i = j$; $s^i_j = \prod 1-(1-r_i)^{m_i}$

$s^1_1 = 1-(1-r_i)^1$ $\quad m_i = 1 \rightarrow \phi_1(m_1)$

$= 1-(1-0.9) = 1-0.1 = \underline{0.9}$ | $c = c_i * 1 = 30 \times 1 = \underline{30}$

$s^1_1 = \{(0.9, 30)\}$

$s^1_2 = 1-(1-r_1)^2 = 1-(1-0.9)^2$

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

$= \underline{0.99}$

$c = c_i * 2 = 30 \times 2 = 60$

$s^1_2 = \{(0.99, 60)\}$

RELIABILITY DESIGN

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

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

- $\phi_2(m_1) = 1 - (1-0.8)^1 = 1 - 0.2 = \underline{0.8}$

  $c = c_2 \times m_1 = 15 \times 1 = \underline{15}$

- $S^2_1 = \{(0.9 \times 0.8, 30+15), (0.99 \times 0.8, 60+15)\}$

  $= \{(0.72, 45), (0.792, 75)\}$

- $\phi_2(m_2) = 1 - (1-0.8)^2 = 1 - (0.2)^2 = 1 - 0.04 = 0.96$

  $c = c_2 \times m_2 = 15 \times 2 = \underline{30}$

$S^2_2 = \{(0.9 \times 0.96, 30+30), (0.99 \times 0.96, 60+30)\}$

$= \{(0.864, 60), (0.9504, 90)\}$

RELIABILITY DESIGN

- $\phi_2(m_3) = 1-(1-0.8)^3 = 1-(0.2)^3$

$= 1-0.008 = 0.992,$

$c = c_2 \times 3 = 15 \times 3 = 45$

$S_3^2 = \{(0.9 \times 0.992,\ 30+45),\ (0.99 * 0.992,\ 60+45)\}$

$= \{(0.8928,\ 75),\ (0.98208,\ 105)\}$

$S^2 = S_1^2 \cup S_2^2 \cup S_3^2$

$= \{(0.72, 45),\ (0.792, 75)\} \cup \{(0.864, 60),\ (0.9504, 90)\} \cup \{(0.8928, 75),\ (0.98208, 105)\}$

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

$(0.792, 75)$ is dominating $(0.864, 60)$ and $(0.9504, 90)$ cost is only $15 less, but the next device cost is 20, so $(0.9504, 90)$ will be also discarded.

$(0.98208, 105)$ will be also discarded.

RELIABILITY DESIGN

- $\phi_3(m_1) = 1-(1-r_3)^{m_1} = 1-(1-0.5)^1 = 1-0.5 = 0.5$

$c = c_3 * m_1 = 20 * 1 = 20$

$S^3_1 = \{(0.72*0.5,\ 45+20),\ (0.864*0.5,\ 60+20),\ (0.8928*0.5,\ 75+20)\}$

$= \{(0.36, 65),\ (0.432, 80),\ (0.4464, 95)\}$

- $\phi_3(m_2) = 1-(1-r_3)^2 = 1-(1-0.5)^2 = 1-0.25 = 0.75$

$c = c_3 * m_2 = 20 * 2 = 40$

$S^3_2 = \{(0.72*0.75,\ 45+40),\ (0.864*0.75,\ 60+40),\ (0.8928*0.75,\ 75+40)\}$

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

- $\phi_3(m_3) = 1-(1-r_3)^3 = 1-(1-0.5)^3 = 1-0.5^3 = 1-0.125$

$= 0.875$

$c = c_3 * m_3 = 20 * 3 = 60$

$S^3_3 = \{(0.792*0.875,\ 75+60),\ (0.864*0.875,\ 60+60),\ (0.8928*0.875,\ 75+60)\}$

$= \{(0.63, 105)\}$

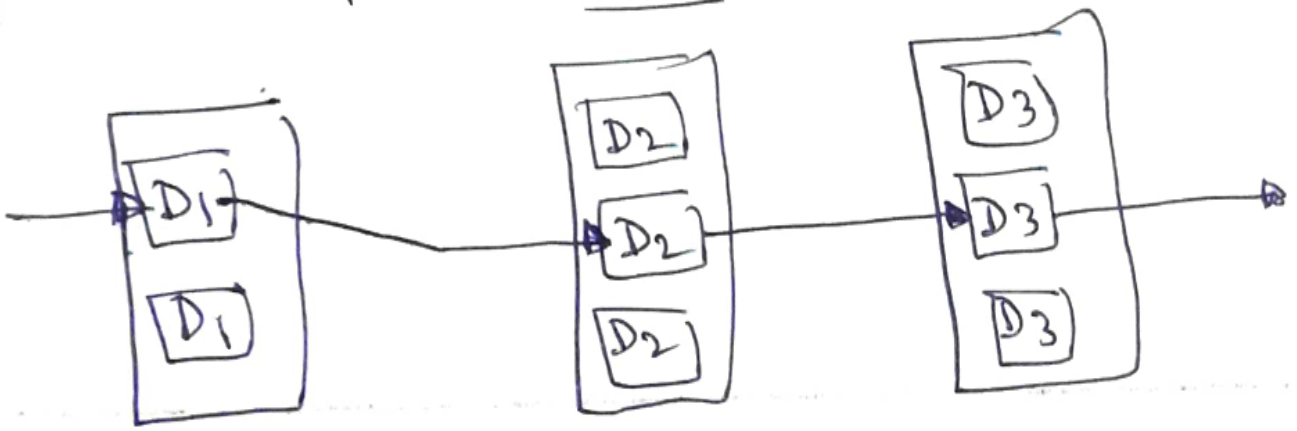
$S^3 = \{(0.36, 65),\ (0.432, 80),\ (0.54, 85),\ (0.648, 100)\}$

RELIABILITY DESIGN

The best design has a reliability of 0.648 and a cost of 100. which is there in S_2^3. So select device 2 in stage 3.

(0.648, 100) is obtained from (0.864, 80) in S^2. which is then in S_2^2. So $m_2 = 2$

(0.864, 80) is obtained from (0.9, 30) from S^1 which is there in S_1^1. So $m_1 = 1$.



Reliable System

Travelling Salesperson Problem

Travelling Salesperson Problem

Travelling salesperson problem is used to find the optimal tour. A tour to be 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 $V - \{1, k\}$ exactly once.

It is easy to see that if the tour is optimal, then the path from k to 1 must be shortest k to 1 path going through all vertices in $V - \{1, k\}$. Hence principle of optimality holds.

Let $g(i, S)$ be the length of the shortest path starting from vertex i going through all vertices in S, and terminating at vertex 1.

Travelling Salesperson Problem

$g(1, V-\{1\})$ is the length of the optimal salesperson tour.

From the principle of optimality it follows that

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

Generalizing the above eq-(1), we obtain (for $i \notin S$)

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

eq-(1) can be solved for $g(1, V-\{1\})$. If we know $g(k, V-\{1,k\})$ for all choices of k, the g value can be obtained by using eq-(2).

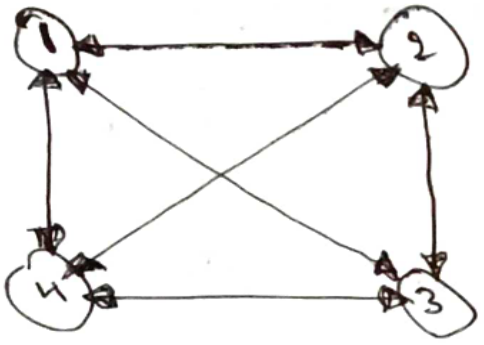
clearly $g(i, \phi) = c_{i1}, 1 \le i \le n$.

Travelling Salesperson Problem

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

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

Ex: Consider the directed graph of fig (a), the edge lengths are given by matrix in fig (b)



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

* $|S| = \phi$

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

Travelling Salesperson Problem

* $|S| = 1$

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

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

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

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

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

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

Travelling Salesperson Problem

* $|S| = 2$

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

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

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

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

$$= \min\{13+18,\ 12+13\} = \min\{31, 25\}$$

$$= \underline{25}$$

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

$$= \min\{8+15,\ 9+18\} = \min\{23, 27\}$$

$$= \underline{23}$$

Travelling Salesperson Problem

* $|S| = 3$

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

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

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

$$= 35$$

So optimal tour cost is 35

Let $J(i,s)$ be the value of j that minimizes length

$J(1,\{2,3,4\}) = 2$ - Then the tour starts with 1 and goes to 2.

The remaining tour can be obtained from $g(2,\{3,4\})$. So $J(2,\{3,4\}) = 4$

$J(4,\{3\}) = 3$

The optimal tour is : $1 \xrightarrow{10} 2 \xrightarrow{10} 4 \xrightarrow{9} 3 \xrightarrow{6} 1$

35

Travelling Salesperson Problem

for each value of $|S|$, there are $n-1$ choices for i. The number of distinct sets S of size k not including 1 and i is $\binom{n-2}{k}$

Hence, the total number of $g(i,S)$ to be computed before computing $g(1, V-\{1\})$ is

$$\sum_{k=0}^{n-1} (n-1)\binom{n-2}{k}$$

$$n-1\left[\binom{n-2}{0}+\binom{n-2}{1}+\binom{n-2}{2}\cdots+\binom{n-2}{n-1}\right]$$

$\rightarrow 2^{n-2}$

So $\sum_{k=0}^{n-1} (n-1)\binom{n-2}{k} = (n-1)\cdot 2^{n-2}$

All Pairs Shortest Path Problem

All-Pairs Shortest Paths

Let $G = (V, E)$ be a directed graph with n vertices. Let cost be a cost adjacency matrix for G such that $cost(i,i) = 0$, $1 \le i \le n$. The $cost(i,j)$ is a length (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 pair 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.

The matrix A can be obtained by solving n Single-Source problems using the algorithm Shortest path. Each application of the procedure requires $O(n^2)$ time, the matrix A can be obtained in $O(n^3)$ time

All Pairs Shortest Path Problem

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

clearly $A^0(i,j) = cost(i,j)$, $1 \le i \le n$, $1 \le j \le n$. we can obtain recurrence for $A^k(i,j)$ using an argument similar to that used before. A shortest path from i to j going through no vertex higher than k either goes through vertex k or it does not. If it does, $A^k(i,j) = A^{k-1}(i,k) + A^{k-1}(k,j)$. If it does not, then no intermediate node has index greater than $k-1$. Hence $A^k(i,j) = A^{k-1}(i,j)$

Combining we get:

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

All Pairs Shortest Path Problem

The graph of below fig (a) has the cost matrix of fig (b) - the initial matrix A^0.

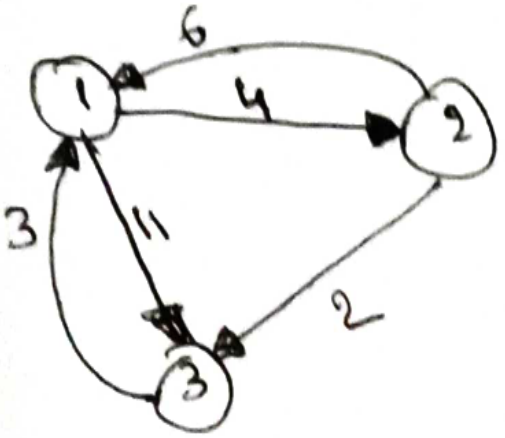


fig (a)

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

fig (b)

We apply 3 - iterations and we can calculate A^1, A^2 & A^3.

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

All Pairs Shortest Path Problem

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

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

$= 0$

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

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

$= 4$

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

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

$= 11$

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

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

$= 6$

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

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

$= 0$

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

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

$= 2$

All Pairs Shortest Path Problem

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

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

$= \underline{3}$

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

$= \min\{\infty, 3+4\} = \min\{\infty, 7\}$

$= \underline{7}$

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

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

$= \underline{0}$

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

All Pairs Shortest Path Problem

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

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

$= \underline{0}$

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

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

$= \underline{4}$

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

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

$= \underline{6}$

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

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

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

$= \underline{6}$

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

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

$= \underline{0}$

All Pairs Shortest Path Problem

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

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

$= 2$

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

$= \min\{3,\ 7+6\} = \min\{3,13\}$

$= 3$

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

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

$= 7$

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

$= \min\{0,\ 7+2\} = \min\{0,9\}$

$= 0$

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

All Pairs Shortest Path Problem

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

$= \min\{0,\ 6+3\} = \min\{0, 9\}$

$= \underline{0}$

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

$= \min\{4,\ 6+7\} = \min\{4, 13\}$

$= \underline{4}$

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

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

$= \underline{6}$

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

$= \min\{6,\ 2+3\} = \min\{6, 5\}$

$= \underline{5}$

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

$= \min\{0,\ 2+7\} = \min\{0, 9\}$

$= \underline{0}$

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

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

All Pairs Shortest Path Problem

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

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

$= \underline{3}$

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

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

$= \underline{7}$

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

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

$= \underline{0}$

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

All Pairs Shortest Path Problem

* Path: 1 to 3 via 2 is minimum

$1 \xrightarrow{4} 2 \xrightarrow{2} 3 = 6$ ✓

Direct edge cost $1 \longrightarrow 3 = 11$

* Path: 2 to 1 via 3 is minimum

$2 \xrightarrow{2} 3 \xrightarrow{3} 1 = 5$ ✓

Direct edge cost $2 \longrightarrow 1 = 6$

* Path: 3 to 2 via 1 is minimum

$3 \xrightarrow{3} 1 \xrightarrow{4} 2 = 7$ ✓

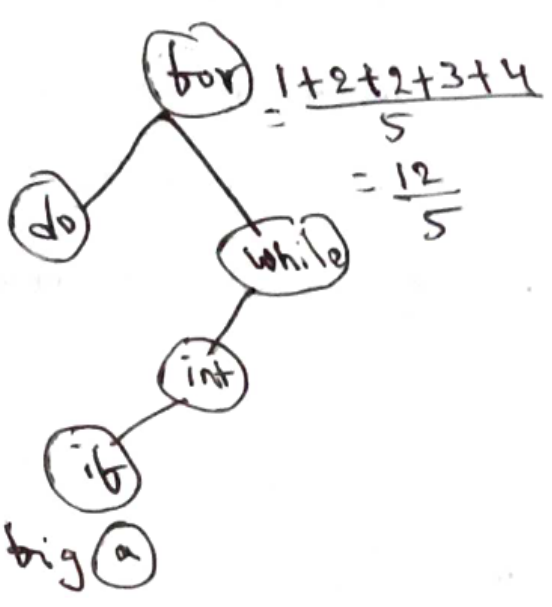
There is no direct edge so $3 \longrightarrow 2 = \infty$

All the remaining are direct edge costs only

Optimal Binary Search Trees

Optimal Binary Search Trees (OBST)

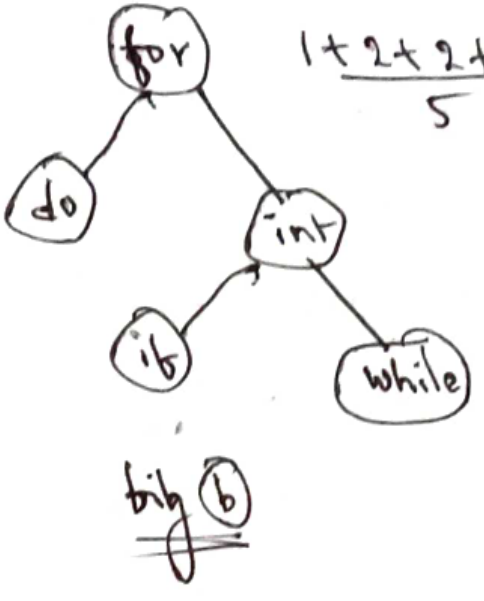
Given a set of identifiers, we wish to create a binary search tree organization. we may expect different binary search trees for the same identifiers set to have different performance characteristics



$= \frac{1+2+2+3+4}{5}$

$= \frac{12}{5}$

fig (a)



$\frac{1+2+2+3+3}{5} = \frac{11}{5}$

fig (b)

Fig (a) requiring 4 comparisons in the worst case, and fig (b) is requiring only 3 comparisons in worst case.

The average number of comparisons required by fig (a) are $\frac{12}{5}$ and fig (b) are $\frac{11}{5}$.

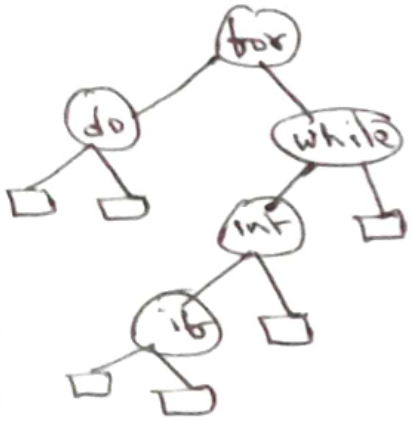
Optimal Binary Search Trees

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

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 $p(i)$ be the probability with which we can search for a_i. Let $q(i)$ be the probability that the identifier x being searched for is such that $a_i < x < a_{i+1}$, $0 \le i \le n$.

clearly $\sum_{i=1}^{n} p(i) + \sum_{i=0}^{n} q(i) = 1$

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



Optimal Binary Search Trees

If a binary search tree represents n identifiers, Then there will be exactly n internal nodes and (n+1) external nodes (fictitious nodes) Every internal node represents a point where a successful search may terminate, Every external node represents a point where an ~~unsuccessful~~ unsuccessful search may terminate.

If a successful search terminates at an internal node at level 'l', Then l iterations of the while loop is required by the algorithm. Hence the expected Cost Contribution from internal node for a_i is $\boxed{P(i) * Level(a_i)}$

The identifiers not in the binary search trees can be partitioned into n+1 equivalent classes E_i, $0 \le i \le n$. The class E_0 contains all the identifiers x such that $x < a_1$. The class E_i contains all the identifiers x such that $a_i < x < a_{i+1}$ $1 \le i \le n$. The class E_n contains all the identifiers x, $x > a_n$

Optimal Binary Search Trees

If the failure node is their at level l, then only $l-1$ iterations of the while loop are made. Hence the cost contribution of this node is $q(i) * (level(E_i) - 1)$.

The expected cost of the binary Search Tree is

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

```
Algorithm Search(x)
{
found := false; t := true;
while ((t ≠ 0) and not found) do
  {
if (x = (t → data)) then found := true;
else
  if (x < (t → data)) then t := (t → lchild);
else
    t := (t → rchild);
  }
if (not found) then return 0;
else
  return t;
}
```

Optimal Binary Search Trees

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

Make a decision that which of the a_i's should be assigned root node of the Tree. If we choose a_k, then it is clear that the internal nodes of $a_1, a_2, \ldots a_{k-1}$ as well as external nodes for the classes $E_0, E_1, \ldots E_{k-1}$ will lie in the left subtree of the root, the remaining nodes will be in the right subtree

$$Cost(\lambda) = \sum_{1 \le i \le k} p(i) * level(a_i) + \sum_{0 \le i \le k} q(i) * (level(E_i) - 1)$$

and

$$Cost(r) = \sum_{k \le i \le n} p(i) * level(a_i) + \sum_{k \le i \le n} q(i) * (level(E_i) - 1)$$

Optimal Binary Search Trees

$$P(k) + \text{cost}(L) + \text{cost}(r) + w(0, k-1) + w(k, n)$$

$\text{Cost}(L) = c(0, k-1)$ $\quad$ $\text{Cost}(R) = c(k, n)$

Hence for $c(0,n)$ we obtain

$$c(0,n) = \min_{1 \le k \le n} \{ c(0,k-1) + c(k,n) + P(k) + w(0,k-1) + w(k,n) \}$$

is minimum.

we can generalize the above eq to obtain for any $c(i,j)$

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

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

the above equation can be solved for $c(0,n)$ by first Computing all $c(i,j)$ such that $j-i=1$ [$c(i,i)=0$, $w(i,i) = q(i)$ $0 \le i \le n$] we can compute all $c(i,j)$ such that $j-i=2$, then compute all $c(i,j)$ with $j-i=3$ and so on. we also compute $r(i,j)$ $r(i,j)$ is the value k, that minimizes the above equation

Optimal Binary Search Trees

Ex: Let $n = 4$, $(a_1, a_2, a_3, a_4) = (do, if, int, while)$.
Let $p(1:4) = (3, 3, 1, 1)$ and $q(0:4) = (2, 3, 1, 1, 1)$. The p's and q's have been multiplied by 16 for convenience. Initially we have

$w(i,i) = q(i)$, $c(i,i) = 0$, $r(i,i) = 0$ $\quad 0 \le i \le 4$

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

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

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

$c(0,1) = \min_{0<k\le 1} \{c(0,0) + c(1,1)\} + w(0,1)$
$= \min\{0+0\} + 8 = 0 + 8 = 8$ $\quad r(0,1) = 1$

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

$c(1,2) = \min_{1<k\le 2} \{c(1,1) + c(2,2)\} + w(1,2)$
$= \min\{0+0\} + 7 = 0 + 7 = 7$ $\quad r(1,2) = 2$

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

$c(2,3) = \min_{2<k\le 3} \{c(2,2) + c(3,3)\} + w(2,3)$.
$= \min\{0+0\} + 3 = 0 + 3 = 3$ $\quad r(2,3) = 3$

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

$c(3,4) = \min_{3<k\le 4} \{c(3,3) + c(4,4)\} + w(3,4)$
$= \min\{0+0\} + 3 = 0 + 3 = 3$ $\quad r(3,4) = 4$

Optimal Binary Search Trees

* $j-i=2$

- $w(0,2) = p(2)+q(2)+w(0,1)$
  $= 3+1+8 = 4+8 = 12$

$c(0,2) = \min_{0<k\le 2,\ k=1,2} \{c(0,0)+c(1,2),\ c(0,1)+c(2,2)\} + w(0,2)$

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

$= 7+12 = 19$

$r(0,2) = 1$

- $w(1,3) = p(3)+q(3)+w(1,2)$
  $= 1+1+7 = 9$

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

$= \min\{0+3,\ 7+0\} + 9 = \min\{3,7\}+9 = 3+9 = 12$

$r(1,3) = 2$

- $w(2,4) = p(4)+q(4)+w(2,3) = 1+1+3 = 5$

$c(2,4) = \min_{2<k\le 4,\ k=3,4} \{c(2,2)+c(3,4),\ c(2,3)+c(4,4)\} + w(2,4)$

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

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

Optimal Binary Search Trees

* $j - i = 3$

• $w(0,3) = p(3) + q(3) + w(0,2) = 1 + 1 + 12 = \underline{14}$

$$c(0,3) = \min_{0<k\leq 3,\ k=1,2,3} \{c(0,0) + c(1,3),\ c(0,1) + c(2,3),\ c(0,2) + c(3,3)\} + w(0,3)$$

$$= \min\{0+12,\ 8+3,\ 19+0\} + 14 = \min\{12, 11, 19\} + 14$$

$$= 11 + 14 = 25$$

$r(0,3) = \underline{2}$

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

$$c(1,4) = \min_{1<k\leq 4,\ k=2,3,4} \{c(1,1) + c(2,4),\ c(1,2) + c(3,4),\ c(1,3) + c(4,4)\} + w(1,4)$$

$$= \min\{0+8,\ 7+3,\ 12+0\} + 11 = \min\{8, 10, 12\} + 11$$

$$= 8 + 11 = \underline{19}$$

$r(1,4) = \underline{2}$

Optimal Binary Search Trees

$j - i = 4$

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

$= 1 + 1 + 14 = \underline{16}$

$$c(0,4) = \min_{0<k\leq 4,\ k=1,2,3,4} \{c(0,0) + c(1,4),\ c(0,1) + c(2,4),\ c(0,2) + c(3,4),\ c(0,3) + c(4,4)\} + w(0,4)$$

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

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

$= 16 + 16 = \underline{32}$

$r(0,4) = \underline{2}$

Optimal Binary Search Trees

- Start with $j-i=4$

  $r(0,4)=2$ Consider second identifier "a_2" as root node of the OBST. a_2 = "if"

- go to $j-i=3$ and select minimum cost value. $c(1,4)$ is minimum

  $r(1,4)=2$. a_2 is already selected. (if)

- goto $j-i=2$, select minimum cost value. i.e $c(2,4)$ is minimum

  $r(2,4)=3$ or 4 a_3 = int, a_4 = while

- goto $j-i=1$. select minimum cost value - $c(2,3)$ and $c(3,4)$ are minimum

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

Only one identifier is left, i.e, a_1 = do add "do" as left child of root node "if". So the resultant OBST are