Dynamic Programming (DP)

Dynamic Programming is a design principle which is used to solve problems with overlapping sub problems.

- It is used when the solution to a problem can be viewed as the result of a sequence of decisions. It avoids duplicate calculation in many cases by keeping a table of known results and fills up as sub instances are solved.
- In Dynamic Programming We usually start with the smallest and hence the simplest sub-instances. By combining their solutions, we obtain the answers to sub-instances of increasing size, until finally we arrive at the solution of the original instance.
- It follows a bottom-up technique by which it start with smaller and hence simplest sub instances. Combine their solutions to get answer to sub instances of bigger size until we arrive at solution for original instance.

How DP differ from Greedy and Divide & Conquer

- Dynamic programming differs from Greedy method because Greedy method makes only one decision sequence but dynamic programming considers more than one decision sequence. However, sequences containing sub-optimal sub-sequences can not be optimal and so will not be generated.

- Divide and conquer is a top-down method. When a problem is solved by divide and conquer, we immediately attack the complete instance, which we then divide into smaller and smaller sub-instances as the algorithm progresses. The difference between Dynamic Programming and Divide and Conquer is that the sub problems in Divide and Conquer are considered to be disjoint and distinct where as in Dynamic Programming they are overlapping.

Principle of Optimality

- An optimal sequence of decisions has the property that whatever the initial state and decisions are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.
- In other words, principle of optimality is satisfied when an optimal solution is found for a problem then optimal solutions are also found for its sub-problems also.

General Method

The Idea of Developing a DP Algorithm

Step1: Structure: Characterize the structure of an optimal solution.
– Decompose the problem into smaller problems, and find a relation between the structure of the optimal solution of the original problem and the solutions of the smaller problems.
Step2: Principle of Optimality: Recursively define the value of an optimal solution.
– Express the solution of the original problem in terms of optimal solutions for smaller problems.

Step 3: Bottom-up computation: Compute the value of an optimal solution in a bottom-up fashion by using a table structure.
Step 4: Construction of optimal solution: Construct an optimal solution from computed information.
Steps 3 and 4 may often be combined.
Remarks on the Dynamic Programming Approach
Steps 1-3 form the basis of a dynamic-programming solution to a problem. Step 4 can be omitted if only the value of an optimal solution is required.

All pairs Shortest Paths

Let G=(V,E) be a directed graph with n vertices.The cost of the vertices are represented by an adjacency matrix such that $1 \leq i \leq n$ and $1 \leq j \leq n$

cost(i, i) = 0,

cost(i, j) is the length of the edge <i, j> if <i, j> $\in$ E and

cost(i, j) = ∞ if <i, j> $\notin$ E.

The graph allows edges with negative cost value, but negative valued cycles are not allowed. All pairs shortest path problem is to find a matrix A such that A[i][j] is the length of the shortest path from i to j.

Consider a shortest path from i to j, $i \neq j$. The path originates at i goes through possibly many vertices and terminates at j. Assume that there is no cycles. If there is a cycle we can remove them without increase in the cost because there is no cycle with negative cost.

Initially we set A[i][j] = cost (i,j).

Algorithm makes n passes over A. Let A^0, A^1, .. A^n represent the matrix on each pass.

Let $A^{k-1}[i,j]$ represent the smallest path from i to j passing through no intermediate vertex greater than k-1. This will be the result after k-1 iterations. Hence kth iteration explores whether k lies on optimal path. A shortest path from i to j passes through no vertex greater than k, either it goes through k or does not.

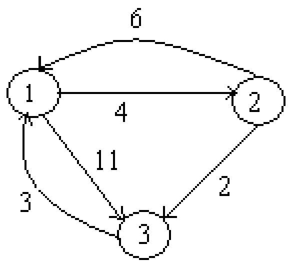
If it does, $A^k[i,j] = A^{k-1}[i,k] + A^{k-1}[k,j]$

If it does not, then no intermediate vertex has index greater than k-1. Then $A^k[i,j] = A^{k-1}[i,j]$

Combining these two conditions we get

$$A^k[i,j] = \min\{ A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j]\}, \; k>=1 \qquad (A^0[i,j]= cost(i, j))$$

Consider the directed graph given below.



Cost adjacency matrix for the graph is as given below

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

Copy the cost values to the matrix A. So we have A^0 as

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

Matrix A after each iteration is as given below.

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

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

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

Recurrence relation is:

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

Algorithm

Void All Paths (float cost[ ][ ], float A[ ][ ], int 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 (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            A[i][j] = cost[i][j];
    for (int k=1; k<=n; k++)
        for (i=1; i<=n; i++)
            for (j=1; j<=n; j++)
                A[i][j] = min{A[i][j], A[i][k] + A[k][j])};
}
```

The above is the algorithm to compute lengths of shortest paths.
Time complexity of the algorithm is O (n^3).

0/1 KNAPSACK PROBLEM (DYNAMIC KNAPSACK)

The 0/1 knapsack problem is similar to the knapsack problem as in the Greedy method except that the xi's are restricted to have a value of either 0 or 1.

We can represent the 0/1 knapsack problem as:

The 0/1 knapsack problem thus is:

Maximize $\sum p_i x_i$

subject to: $\sum w_i x_i \leq$ m(capacity) and x_i=0 or 1 where $1 \leq i \leq n$

The decisions on the x_i are made in the order $x_n, x_{n-1} \ldots x_1$. Following a decision on x_n, we may be in one of two possible states the capacity remaining in the knapsack is m and no profit has accrued or the capacity remaining is m-w_n and profit of p_n has accrued). It is clear that the remaining decisions $x_{n-1},\ldots,x_1$ must be optimal w.r.t the problem state resulting from the decision x_n. Otherwise $x_n,\ldots x_1$ will not be optimal.

Let $f_n(m)$ be the value of optimal solution to KNAP (1,n,m).

Since principle of optimality holds good, we obtain

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

in general,

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

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

Where S^i is the set of all pairs for f_i including (0,0) and f_i is completely defined by the pairs (p_i,w_i)

S^{i+1} may obtained by merging S^i and $S_1{}^i$. This merge corresponding to taking the make of two functions $f_{i-1}(x)$ and $f_{i-1}(x-w_i)+p_i$ in the object function of 0/1 Knapsack problem

So, if one of S^{i-1} and $S_1{}^i$ has a pair (p_j, w_j) and the other has a pair (p_k,w_k) and $p_j \leq p_k$while $w_j \geq w_k$. Then the pair (p_j ,w_j) is discarded this rule is called purging or dominance rule when generating S^i all the pairs (p,w)with w>m may also be purged

If we try to remove i^{th} item then profit of i^{th} item is reduced from total profit and weight of i^{th} item is reduced from total weight which belongs to the profit and weight of $(i-1)^{th}$ item

Outline to solve the 0/1 problem

Initially we start with $S^0=$ {0, 0} i.e profit =0 & weight=0 (bag is empty)

Addition

Then, S0 1 is obtained by adding P1, W1 to all pairs in S0. For general i, S I 1 is obtained by adding Pi, Wi to S i-1 The following recurrence relation defines S I 1.

$$S_1{}^i = \{(P, W) / (P-P_i, W-W_i) \in S^i\}$$

Merging or union $S^i = S^{i-1} \cup S_1{}^i$

Purging (Dominance) Rule

Take any two sets S^i and $S^i{}_1$ consisting of pairs(P_j , W_j) and (P_k ,W_k). The purging rule states that if $P_j \leq P_k$ and $W_j \geq W_k$ then (P_j, W_j) will be deleted. In other words, remove those pairs that achieve less profit using up more weight.

problem : m=6 n=3 (W1,W2,W3)= (2,3,3), and (P1,P2,P3)=(1,2,4)

In other words, the given profit, weight pairs are: (1,2) (2,3) and (4,3)

Solution: Initially take $S^0 =\{0,0\}$

Addition:

$S^0{}_1$ = Add (P_1, W_1) for all pairs in S^0

$S^0{}_1 = \{(1,2)\}$ because S^0 is (0,0)

Merging operation:

$S^i = S^{i-1} + S^{i-1}{}_1$

$S^1 = S^0 \cup S^0{}_1$ there fore $S^1 = \{(0,0) \cup \{(1,2)\} = \{(0,0)\ (1,2)\}$

$S_1{}^2 = S^1 + (P_2 , W_2)$

= {(0,0),(1,2)}+{2,3)}

S_1^2 = {(2,3),(3,5)}

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

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

S_1^3 = S^2 + (P3, W3)

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

S_1^3 = {(4,3),(5,5),(6,6),(7,8)}

S^3 = S^{3-1} U S_1^3 = S^2 U S_1^3

= {(0,0),(1,2),(2,3),(3,5)}U{(4,3),5,5),(6,6),(7,8)}

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

Tuple (7,8) is Discarded because the weight (8) exceeds max capacity of Knapsack m=6 so,
S^3 ={(0,0),(1,2),(2,3),(3,5),(4,3),(5,5),(6,6)

Applying purging rule:

(P_j,W_j) = (3,5); (P_k,W_k) = (5,3)

Here, (3, 5) will be deleted because 3≤5 and 5≥3

This purging rule has to be applied for every Si. In this example, no purging was necessary in S1 and S2.

S^3 = {(0,0),(1,2),(2,3),(4,3),(5,5),(6,6)}

(2,3) will be deleted because 2≤4 and 3≥3.

S^3 = {(0,0),(1,2),(4,3),(5,5),(6,6)}

Finding optimal Solution

The searching process is done starting with the last tuple (P1, W1) of $S^{n.}$ A set of 0/1 values for x_i 's such that $\sum p_i x_i$=P1 and $\sum w_i x_i$=W_1 . This can be determined by carrying out a search through the s^i s.

If $(P_1,W_1) \in S^{n-1}$, we can set x_n=0. Otherwise (if (P1, W1) not in S n-1), we set Xn to 1. Then, we will continue this search recursively through all Sn-1, Sn-2, …., S0 with either (P1, W1) in the case xn = 0 or (P_1-p_n,W_1-w_n) in the case xn=1.

In the above example, last tuple in S3 is (6,6) but (6, 6) $\notin S^2$. Hence x_3=1.

Now, we need to search for (6-P3, 6-W3) which is (2,3).

(2, 3) $\in S^2$ but (2,3) $\notin S^1$ So, x_2=1.

Now, we need to search for (2-P2, 3-W2) which is (0,0).

(0,0) $\in S^1$ and (0,0) $\in S^0$. Hence, X1=0.

Therefore the optimal sol is {x1=0,x2=1,x3=1} meaning weight 2 and weight 3 are taken and weight 1 is ignored.

Informal knapsack algorithm

```
1.   **Algorithm** DKP(p, w, n, m)
2.   {
3.   $S^0$ := {0,0)};
4.   **for** i :=1to n-1 **do**
5.   {
6.       $S^{i-1}$ :={(P,W)(P-pi,W-wi) $\in S^{i-1}$ and W$\leq$m};
7.       $S^i$ :=**Marge Purge**($S^{i-1}$, $S^{i-1}_1$);
8.   }
9.   (PX,WX) :=last pair in $S^{n-1}$
10.  (PY,WY) :=(P' +$p_n$,W' +$w_n$) where W' is the largest W in
11.          any pair in $S^{n-1}$ such that W+ $w_n \leq$m;
12.        //Trace back for $x_n, x_{n-1}, \ldots, x_1$.
13.        if(PX > PY) **then** $x_n$ :=0;
14.        **else** $x_n$ :=1;
15.         Trace Back For($x_{n-1}, \ldots x_1$);
16.  }
```

TRAVELLING SALESMAN PROBLEM

- Let G = (V,E) be a directed graph with edge cost c_{ij} is defined such that $c_{ij} > 0$ for all i and j and $c_{ij} = \infty$,if (i,j) $\notin$ E.
  Let | V | =n and assume n>1.

- The traveling salesman problem is to find a tour of minimum cost.
- A tour of G is a directed simple cycle that includes every vertex in V.
- The cost of the tour is the sum of cost of the edges on the tour.
- The tour is the shortest path that starts and ends at the same vertex i.e 1.
- We know that the tour of the simple graph starts and ends at vertex 1. Every tour consists of an edge <i, k> for some k$\in$ V-{1} and a path from k to 1. Path from k to 1 goes through each vertex in V- {1, k} exactly once. If tour is optimal, path from k to 1 muat be shortest k to 1 path going through all vertices in V-{1, k}. Hence the principle of optimality holds.
  Let g(i, S) be length of shortest path starting at i, going through all vertices in S ending at 1. Function g (1, V-{1}) is the length of optimal salesman tour. From the principle of optimality

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

$$\text{for } i \notin S \quad g(1, S) = \min_{j\in S} \{C_{ij} + g(j, S-\{j\})\} \quad \text{———} \quad (2)$$

$$g(i, \varphi) = C_{i1},\ 1\le i\le n$$

Use eq. (1) to get g (i, S) for |S| =1. Then find g(i, S) with |S|=2 and so on.

APPLICATION:

1. Suppose we have to route a postal van to pick up mail from the mail boxes located at 'n' different sites.
2. An n+1 vertex graph can be used to represent the situation.
3. One vertex represents the post office from which the postal van starts and return.
4. Edge <i,j> is assigned a cost equal to the distance from site 'i' to site 'j'.
5. The route taken by the postal van is a tour and we are finding a tour of minimum length.
6. Every tour consists of an edge <1,k> for some k $\in$ V-{} and a path from vertex k to vertex 1.
7. The path from vertex k to vertex 1 goes through each vertex in V-{1,k} exactly once.
8. the function which is used to find the path is
   $g(1,V-\{1\}) = \min\{ c_{ij} + g(j,S-\{j\})\}$

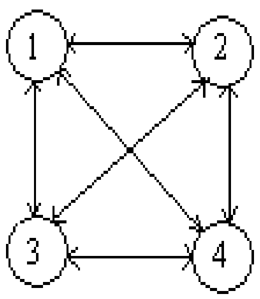
9. g(i,s) be the length of a shortest path starting at vertex i, going through all vertices in S,and terminating at vertex 1.

---

10. The function g(1,v-{1}) is the length of an optimal tour.

STEPS TO FIND THE PATH:

1. Find $g(i,\Phi) = c_{i1}$, $1\le i <n$, hence we can use equation(2) to obtain g(i,s) for all s to size 1.
2. That we have to start with s=1,(ie) there will be only one vertex in set 's'.
3. Then s=2, and we have to proceed until |s| <n-1.
4. for example consider the graph.



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

$g(i,s) \longrightarrow$ set of nodes/vertex have to visited.

$\downarrow$

starting position

From eq $g(i,s) = \min\{c_{ij} + g(j,s-\{j\})$ we obtain $g(i,S)$ for all S of size 1,2…

When $|s|<n-1$, the values of I and S for which $g(i,S)$ is needed such that

$i \neq 1$, $1 \notin s$ and $i \notin s$.

<u>|s| = 0</u>

i =1 to n. **$g(i, \Phi)=c_{i1}, 1\leq i\leq n$**

$g(1,\Phi) = c_{11} \Rightarrow 0$; $g(2,\Phi) = c_{21} \Rightarrow 5$; $g(3,\Phi) = c_{31} \Rightarrow 6$; $g(4,\Phi) = c_{41} \Rightarrow 8$

<u>|s| = 1</u>

i =2 to 4

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

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

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

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

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

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

<u>| s | = 2</u>

$i \neq 1$, $1 \notin s$ and $i \notin s$.

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

$= \min\{9+20,10+15\} = \min\{29,25\} =25$

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

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

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

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

<u>| s | = 3</u>

from eq(1) we obtain

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

$= \min\{10+25,15+25,20+23\} = \min\{35,35,43\} =35$

optimal cost is 35

the shortest path is,

$g(1,\{2,3,4\}) = c_{12} + g(2,\{3,4\})$ => 1->2

$g(2,\{3,4\}) = c_{24} + g(4,\{3\})$ => 1->2->4

$g(4,\{3\}) = c_{43} + g(3\{\Phi\})$ => 1->2->4->3->1

so, the optimal tour is **1 → 2 → 4 →3 → 1**

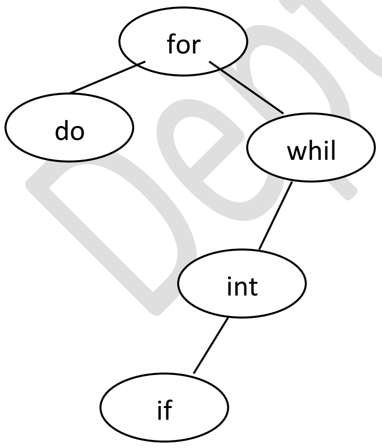
Optimal Binary Search Tree

Consider a fixed set of words and their probabilities. The problem is to arrange these words in a binary search tree

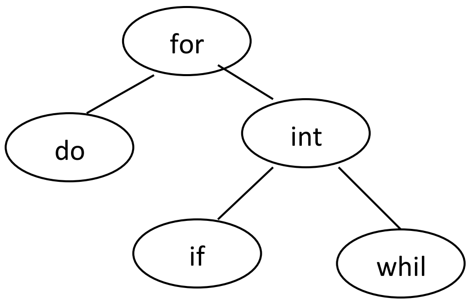
Let us consider **S** is the set of words

S= {if, for, int, while, do}

if the following are Two Binary Search Trees (B S T) For given set S. here assumption is each word has same probability no unsuccessful search



(a)



(b)

We require 4 comparisons to find out an identifier in worst case in fig(a).But in fig (b) only 3 comparisons on avg the two trees need 12/5 ,11/5 comparisons

In General, we can consider different words with different frequencies (probabilities) and un successful searches also

Let Given set of words are $\{a_1, a_2, a_3, \ldots\ldots a_n\}$ With $a_1 < a_2 < \ldots.. a_n$

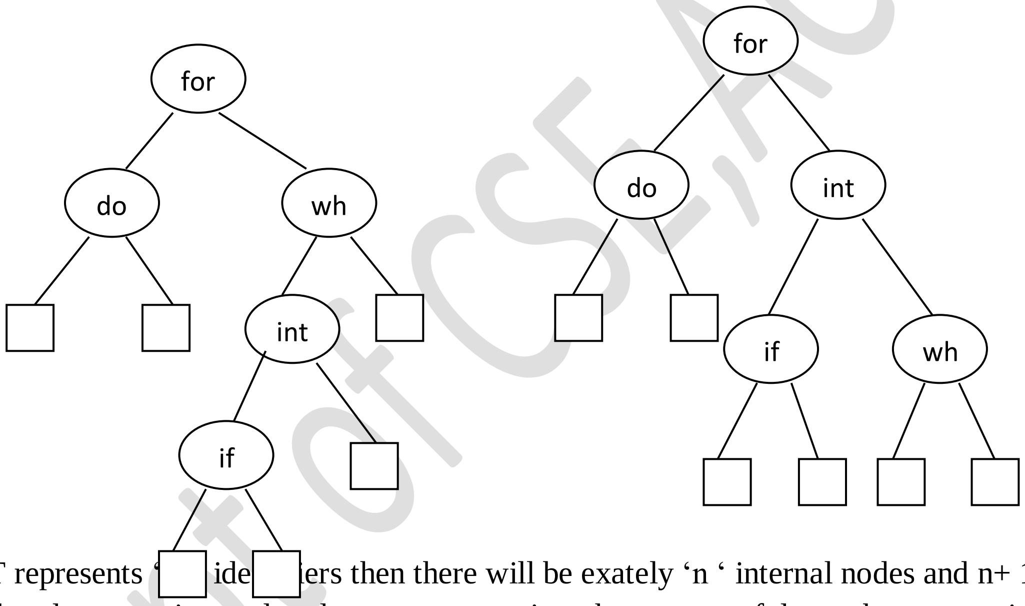
Let P(i) be the probability for searching of a_i

q(i) be the probability for unsuccessful search so clearly

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

So Let us construct optimal Bin search true

To obtain a cost function for BST add external nodes in the place of every empty sub true. These nodes called external nodes.



If BST represents ‘ ide ers then there will be exately ‘n ‘ internal nodes and n+ 1 external nodes every internal node represent a point where successful search may terminate and every external node represents a point where an unsuccessful search may terminates

If a successful search terminates at an internal node at level L, then L interations are required. So expected cost for the internal node for ai is P(i) * Level (ai)

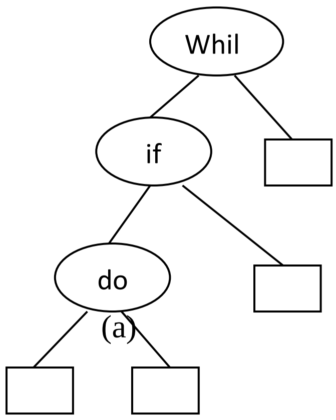
For unsuccessful search terminate at external node the words which are not in BST can be partioned into n+1 Equalnce classes. Equvalence class E_0 represents least among all the identifers in BST& The equalence classes ‘E_n’ represents greatest among all Elements in BST Hence search terment at level Ei-1

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

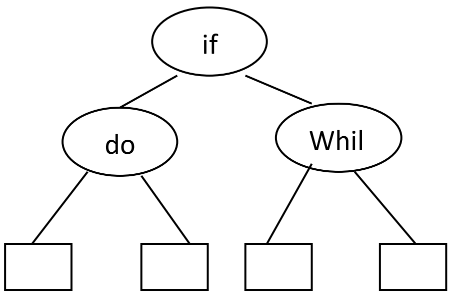
we can obtain optimal BST for the above value is min

problem:

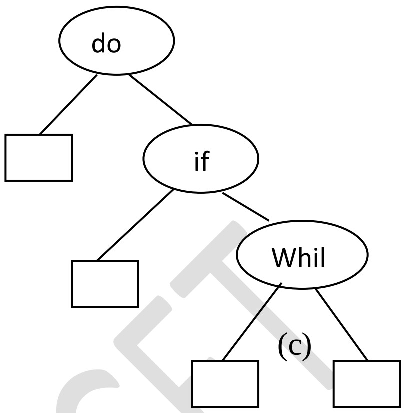
Let P(i)=q(i)=1/7, for all i in the BST The following word set(a_1,a_2,a_3)= (do,it,while)



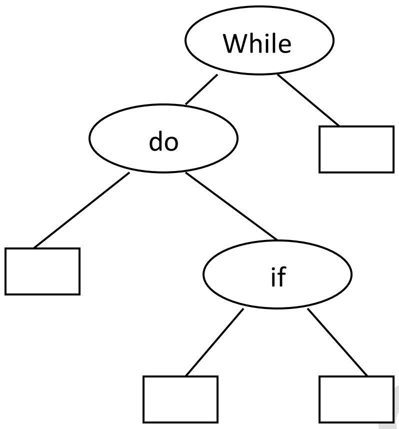
(a)



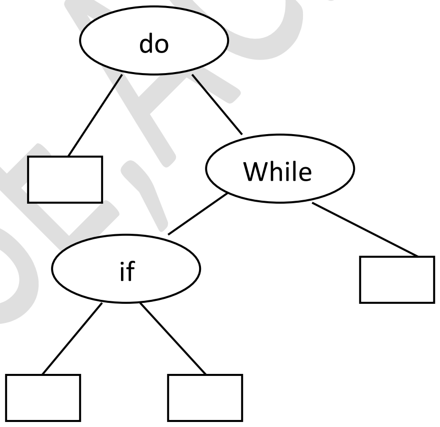
(b)



(c)



(d)



(e)

with Equal probability p(i) = q(i)=1/7 for all i

we have cost (a) = 1/7* 1+1/7*2+1/7*3+1/7*2+1/7*2+1/7*1

= 15/7

Cost (b) = 1/7*1+1/7*2+1/7*2+1/7*2+1/7*2+1/7*2+1/7*2

= 13/7

Cost (c) =cost(d)=cost(e)=15/7

Tree b is optimal

with p(1) = 0.5,p(2) = 0.1,p(3) = 0.05,q(0)=0.05,q(1)=0.1,q(2)=0.05,q(3)=0.05

cost(Tree a)=2.05

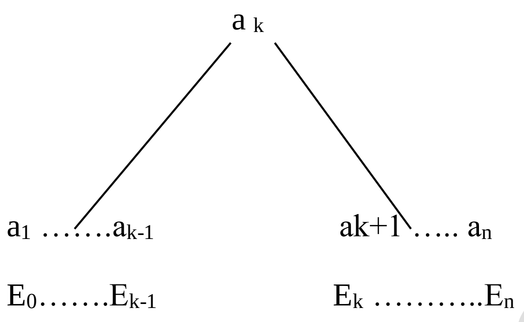
cost (Tree c)= 1*0.5 + 2*0.1 + +3*0.05+3*0.15+3*0.11+2*0.05 +1 * 0.05

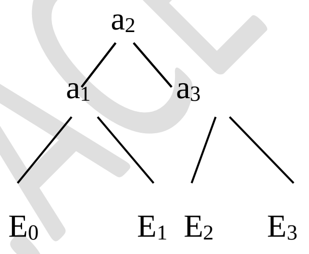
= 0.5+0.2 + 0.15 + 0.45 + 0.3 + 0.10 + 0.05

=1.75

To apply dynamic programming to the problem of obtaining an optimal BST , we need to construct the tree as the result of a sequence of decisions and observe the principal of optimality

One possible approach to this would be to make a desition as to which of the ai's should be assigned to the root node of the tree. If we choose ak then the internal nodes for $a_1, a_2, \dots\dots\dots a_{k-1}$ will lie one left sub tree of the root. The remaining will be in the right sub tree r.





Cost (l) = $\sum$ p(i) * level (ai) + $\sum$q(i)* level (E_i-1)

$1 \le i \le k-1$ $0 \le i \le k-1$

Cost (r) = $\sum$ P(i) * level (ai) + $\sum$ q(i) *(level (Ei)-1)

$k+1 \le i \le n$ $k < i \le n$

By using the forumula

w (i ,j) = q(i) + $\sum$ j((q(l) + p(l)) we obtain

l=i+1

the following as the expected cost of search tree

p(k) +cost (l) + cost (r) + w(0,k-1) + w (k , n)

For left sub tree cost is c (0,k-1) and using (0,k-1)

For right sub Tree cost is c(k,n) and using w(k,n)

cost of the tree is

c(0,n) = min∑ { c (0,k-1) + c(k,n) + w(0,k-1) + w(k,n) + p(k) }

1≤k≤n Total true weight

= min ∑ { (0,k-1) + c (k,n) + w (0,n) }

1≤k≤n

In General,

c(i,j) = min ∑{ c (i,k-1) + c(k,j) + w(i,j) + w(i,j) }

i≤k≤j

The above Eq can be solved for c (0,n) by first computing all c(i,j) such that j-1 = 1

Next we compute all c(i,j) of such that j-i=2 then c(i,j) with j-i=3 and so on.we record all the values of r(i,j) of each tree then optimal BST can be computed from these r (i,j)

<u>Note:</u> Initial values are c (i,i) = 0

w(i,i) = q(i)

r(i,i) = 0 for all 0≤i<4

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

(Problem and Solution discussed in the class room)

<u>MULTI SATAGE GRAPHS</u>

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

5.2. *MULTISTAGE GRAPHS* 259

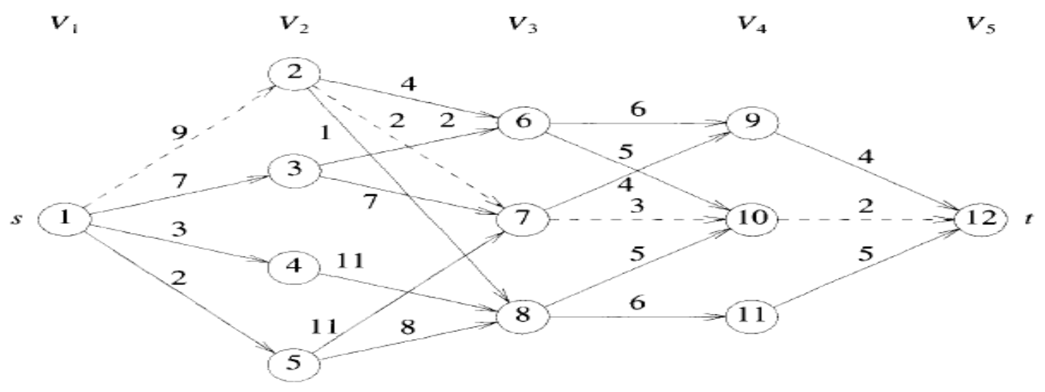


Figure 5.2 Five-stage graph

VERTEX	1	2	3	4	5	6	7	8	9	10	11	12
COST	**16**	**17**	**9**	**18**	**15**	**7**	**5**	**7**	**4**	**2**	**5**	**0**
DISTANCE	**2/3**	**7**	**6**	**8**	**8**	**10**	**10**	**10**	**12**	**12**	**12**	**12**

$$\begin{aligned} cost(3,6) &= \min \ \{6 + cost(4,9), 5 + cost(4,10)\} \\ &= 7 \\ cost(3,7) &= \min \ \{4 + cost(4,9), 3 + cost(4,10)\} \\ &= 5 \end{aligned}$$