Dynamic Programming

Dynamic Programming

"Dynamic programming algorithm stores the results for small sub problems and looks them up, rather than recomputing them, when it needs them later to solve larger sub problems"

Steps in developing a dynamic programming algorithm

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

Matrix chain multiplication

Problem:

We are given a sequence(chain) $<A_1, A_2, A_3, \ldots A_n>$ of n matrices to be multiplied, we wish to compute the product $A_1A_{2\ldots}A_{n.}$ using least work.

Since matrix multiplication is associative so all ways of parenthesization will yield the same answer.

The product of $A_1A_2A_3A_4$ can be parenthesized in five distinct ways

$(A_1(A_2(A_3A_4)))$
$(A_1((A_2A_3)A_4))$
$((A_1A_2)(A_3A_4))$
$((A_1(A_2A_3))A_4)$
$(((A_1A_2)A_3)A_4)$

Although answer by every one will be same, but the cost of evaluating varies drastically.

Example: $A_1 : 10 \times 100$

$A_2 : 100 \times 5$

$A_3 : 5 \times 50$

$((A_1A_2)A_3)$:total cost = 10.100 .5+10.5.50= 7500

$(A_1(A_2A_3))$:total cast 100.5.50 +10.100.50 = 75000

$P(k)P(n-k)\ if n >= 2$

Matrix chain multiplication problem:

Given a chain $<A_1, A_2, \ldots A_n>$ of n matrices where for i = 1,2…n; matrix A_i has dimension p_{i-1}x $p_{i,}$

fully parenthesize the product $A_1A_2\ldots A_n$ in a way that minimizes the number of scalar multiplications.

P(n) : no of alternative parenthesizations

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$$

Let us use Dynamic programming approach

Let us divide the problem as: for $i < j$

Ai..j denotes $A_i A_{i+1} \dots A_j$

To evaluate $A_{i..j}$ We must find an index k

$A_{i..j} = A_{i..k} A_{k+1..j}$ for some k in between i and j.

Total cost to get $A_{i..j} =$

cost of computing $A_{i..k}$ + cost of $A_{k+1..j}$ + cost of combining

Optimal cost of $A_{i..j}$ can be obtained by getting each of $A_{i..k}$ and $A_{k+1..j}$ optimally.

Thus optimal solution of the original problem can be obtained from the optimal solutions of subproblems.

Recursive solution

Let m[i ,j] = minimum no. of multiplications needed to compute the matrix $A_{i..j}$.

So answer of the given problem is m[1,n].

Now m[i,j] can be defined recursively as:

  m[i , i] = 0

Let k be the index for the optimal solution

$$m[i,j] = m[i,k] + m[k+1, j] + p_{i-1}p_kp_j$$

Recursive definition for minimum cost pf parenthesizing the product $A_1A_2\ldots.A_n$ becomes:

$m[i\ ,j] =$

0 $\qquad\qquad\qquad\qquad$ if $i=j$

$\min\{m[i\ ,k] + m[k+1\ ,j] + p_{i-1}p_kp_j$

To keep track of successive decisions let us use

S[i j] to denote the value of index k from where sequence $A_i \dots A_j$ is to be partitioned.

Optimal costs are calculated by using a tabular bottom approach.

Two auxiliary tables

m[1..n,1..n] for storing m[i ,j] costs and

s[1..n,1..n] for recording indices (k) for optimal costs are used.

for finding m[i ,j]

optimal results for $A_{i..k}$ and $A_{k+1..j}$ are used and

size of $A_{i..k}$ is k-i+1 which is less than j-i+1

Also

size of$A_{k+1..j}$ is j-k which is also less than j-i+1

So Table m[i,j] is filled by bottom up approach.

```
Matrix-chain-order (p)
  $n \leftarrow len[p] - 1$
  For $i \leftarrow 1$ to $n$
    do $m[i,i] \leftarrow 0$
  for $l \leftarrow 2$ to $n$
    do for $i \leftarrow 1$ to $n - l + 1$
        do $j \leftarrow i + l - 1$
          $m[i,j] \leftarrow \alpha$
          for $k \leftarrow I$ to $j - 1$
            do $q \leftarrow m[I, k] + m[k+1, j] + p_{i-1}p_kp_j$
```

```
10   If $q < m[I,j]$
11   then $m[I,j] \leftarrow q$
12    $s[i,j] \leftarrow k$
13    Return m , s
```

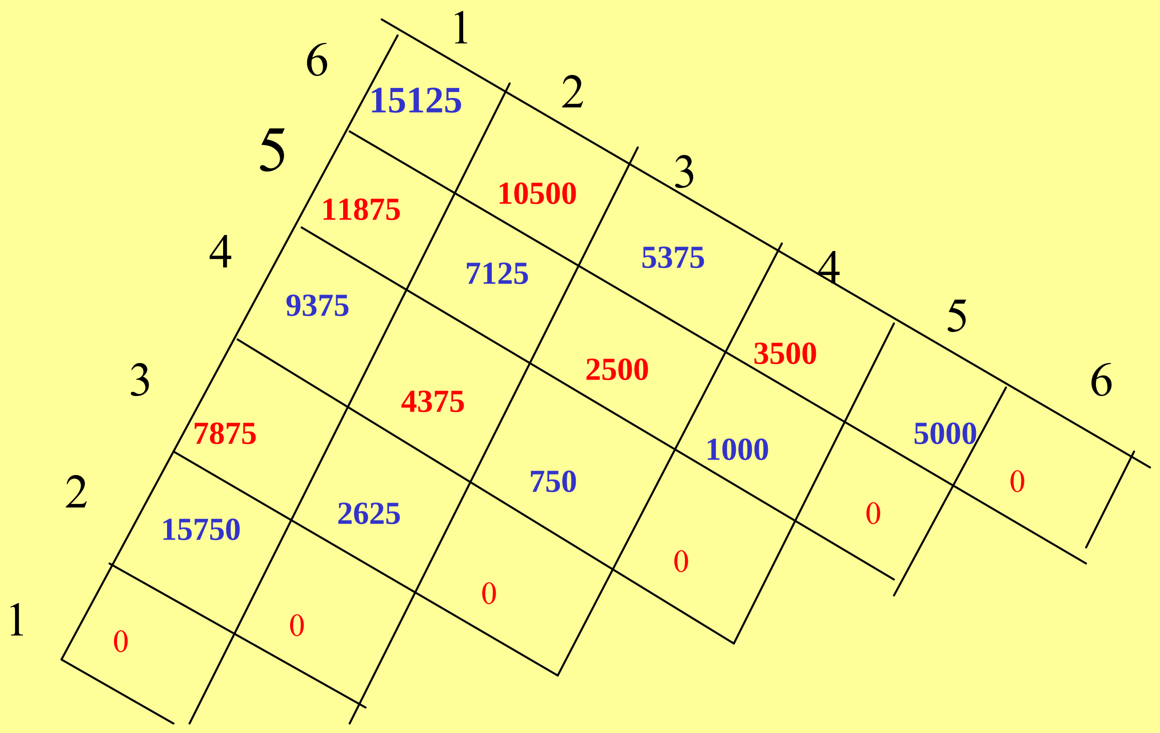
Matrix Dimension

A_1	30 x 35	p0 x p1
A_2	35 x 15	p1 x p2
A_3	15 x 5	p2 x p3
A_4	5 x 10	p3 x p4
A_5	10 x 20	p4 x p5
A_6	20 x 25	p5 x p6

$$M[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1p_2p_5 = 13000 \\ m[2,3] + m[4,5] + p_1p_3p_5 = 7125 \\ m[2,4] + m[5,5] + p_1p_4p_5 = 11375 \end{cases}$$

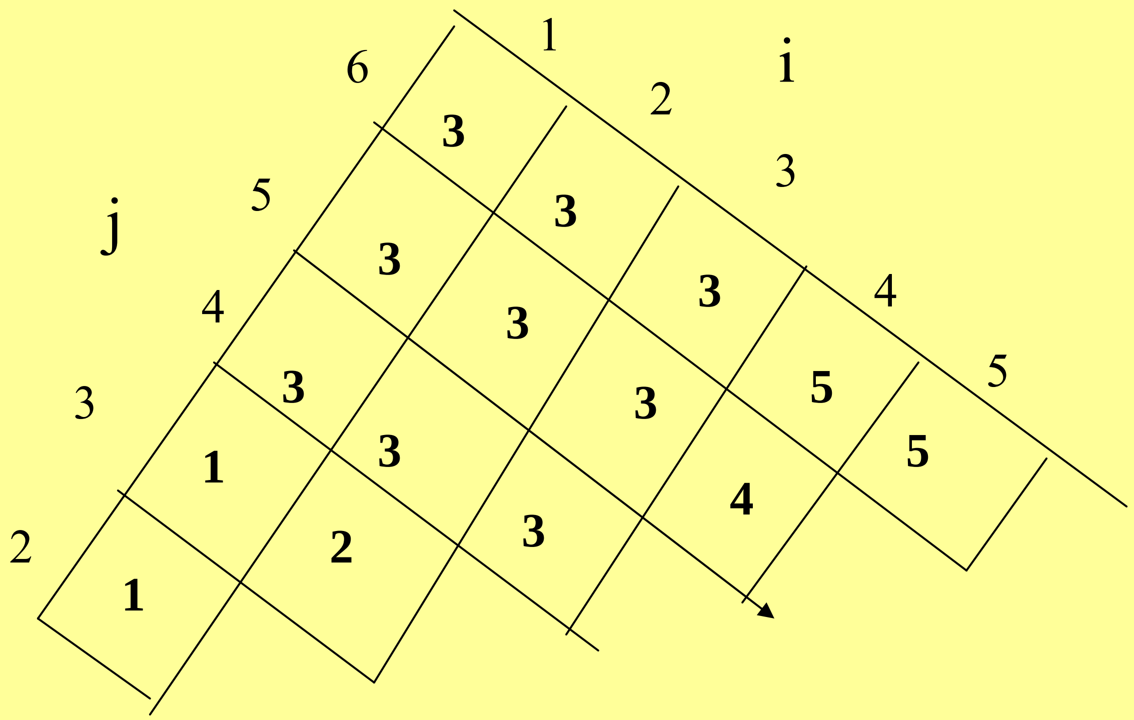
$$= 7125$$

M



	1	2	3	4	5	6
6	15125	10500	5375	3500	5000	0
5	11875	7125	2500	1000	0	
4	9375	4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

S



Optimal solution

- Optimal way to multiplication of these six matrices
- $((A_1(A_2A_3))((A_4A_5)A_6))$

Example

- Let us begin with a simple capital budgeting problem.

  A corporation has $5 million to allocate to its three plants for possible extension.

  Each plant has submitted a number of proposals .

  Each proposal gives the cost of expansion(c) annd total revenue expected(r).

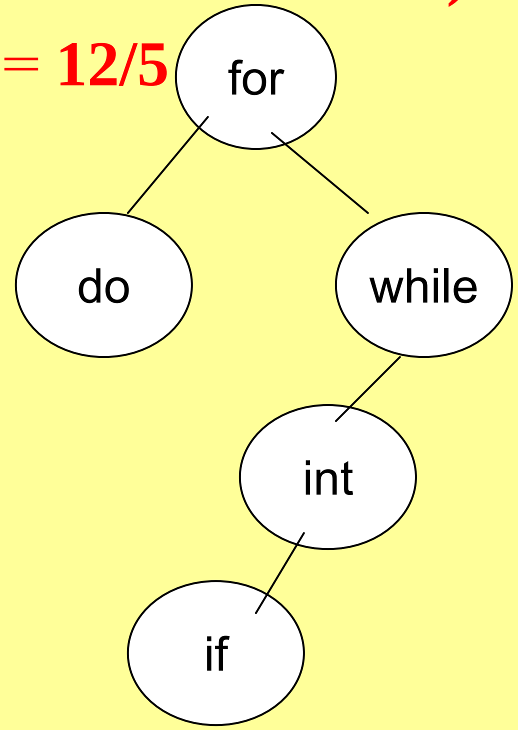
Optimal Binary search Tree

Let the set of identifiers be

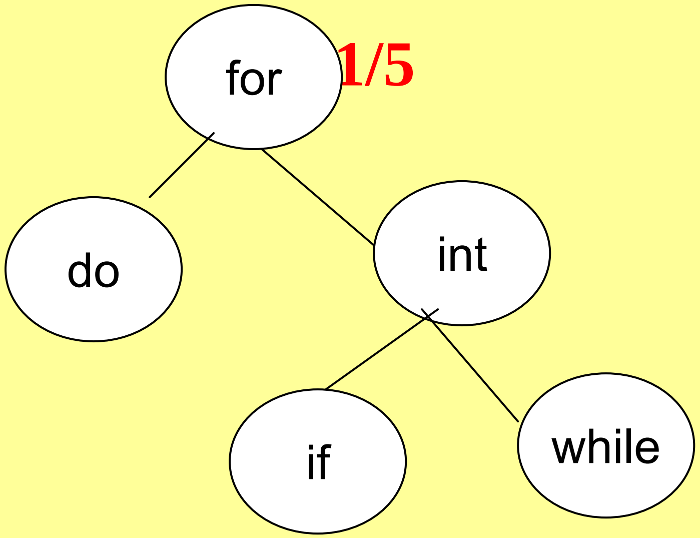
= {for , do , while. int, if} Binary search trees

Average no of searches (if assumed equal probability of occurrence)

= 12/5



1/5



General case with different probabilities

Let us take the set of keys $\{a_{1,} a_2 \ldots . a_n\}$

Where $a_1 < a_2 < \ldots < . a_n$

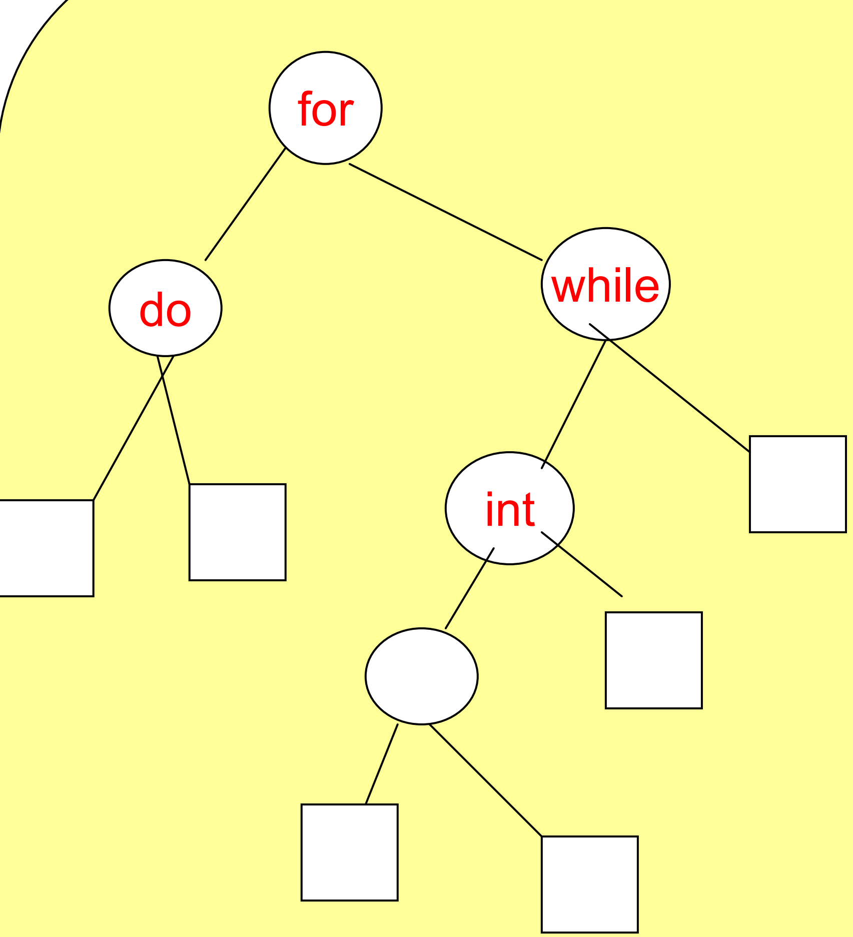
And

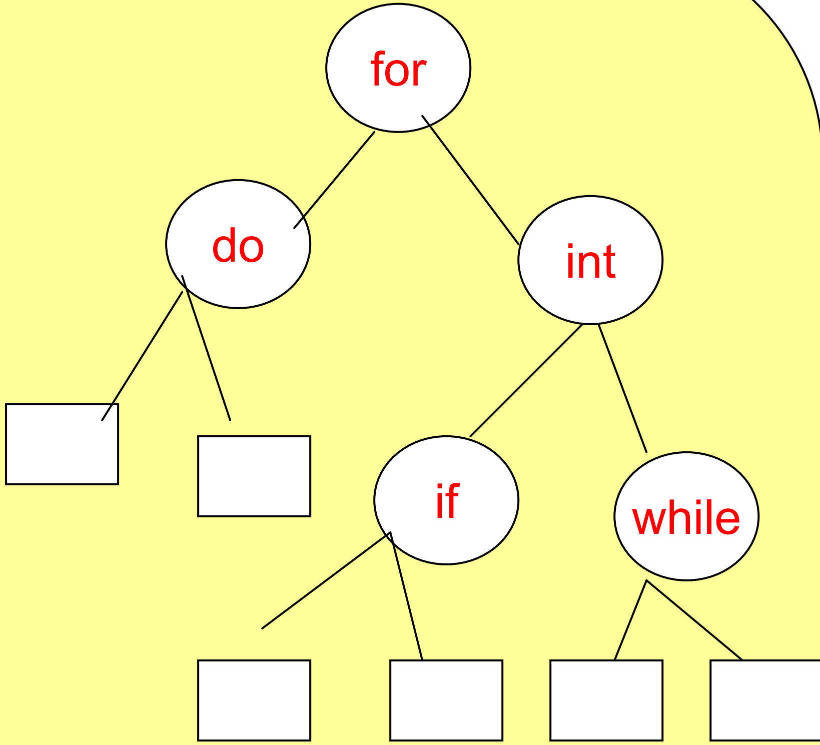
p(i)= prob. of a_i

q(i) = prob. That identifier x lies in between a_i and a_{i+1} i.e. $a_i < x < a_{i+1}$

We assume $a_0 = - \alpha$ and $a_{n+1} = \alpha$

Thus $\sum p_i + \sum q_i = 1$


for
do
while
int


for
do
int
if
while

Optimal search tree : expected cost is minimum

class E_i = set of x s.t. $a_i < x < a_{i+1}$

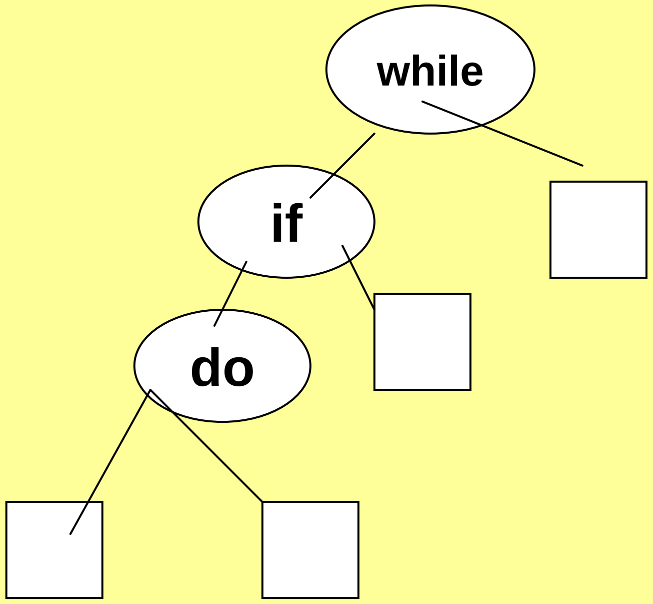
E_n = set of x s.t. $x > a_n$

E_0 = set of x s.t. $x < a_0$

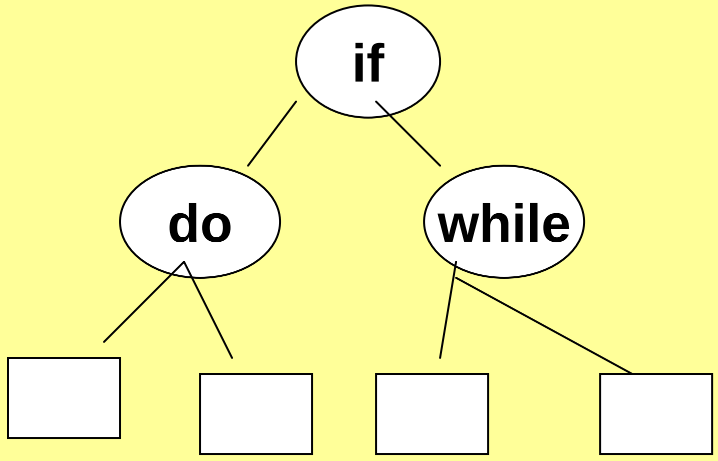
objective:

minimize $(\sum p(i) * \text{level}(a_i) + \sum q(i)(\text{level}(E_i) - 1))$

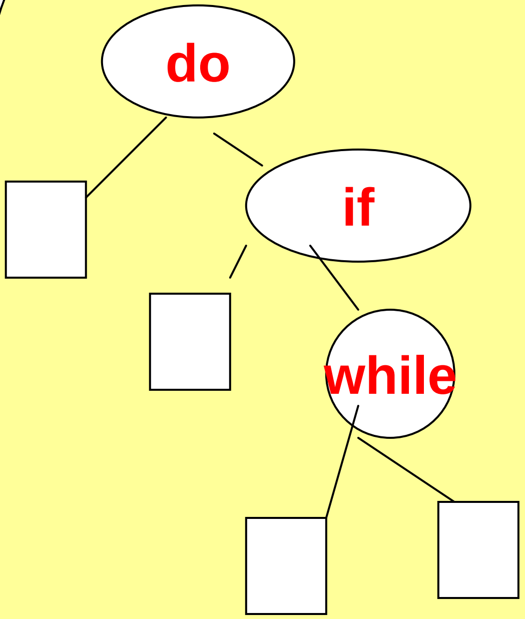
- Draw all binary search trees for set
- { while ,if and do} and find expected costs



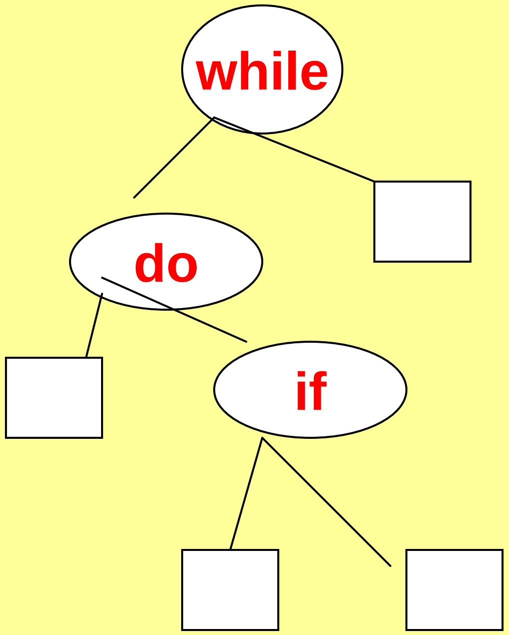
a



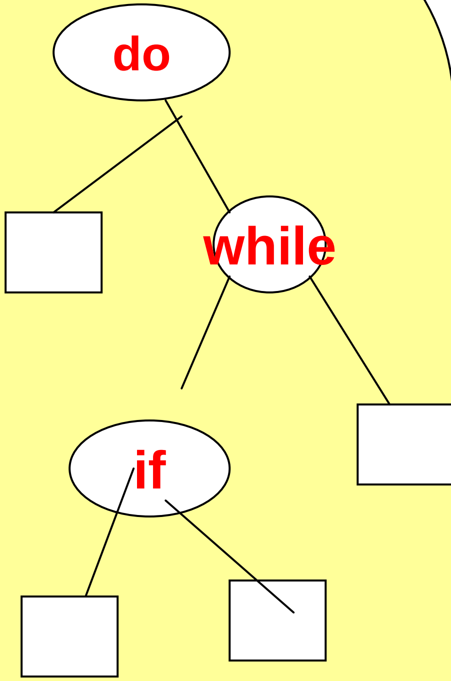
b



c



d



e

Ex: p_i = 1/7 =q_i p=(0.5, 0.1 , 0.05)

q= (0.15, 0.1, 0.05, 0.05)

A	15/7 = 2.1	2.65
B	13/7 =1.8	1.9
C	15/7 =2.1	1.5
D	15/7 =2.1	2.05
E	15/7 =2.1	1.6

To apply dynamic programming approach we need to view the construction of such a tree as the result of a sequence of decisions.

A possible approach would be to make a decision as to which of the a_i' s should be assigned to the root node of the tree.

If we choose a_k as root then internal nodes a1, a2, …ak-1 as well as external nodes for the classes E0, E1,…Ek-1 will lie in the left subtree l of the root.

Remaining will be in the right subtree r.

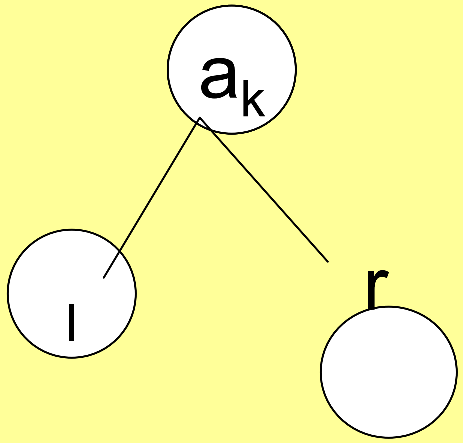
Define (l) =

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

Cost (r) =

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

Note: in both case (l) and(r) the level is measured by regarding the root of the respective subtree to be at level 1.



To consider costs from a_k

extra cost is to be added

Let $w(i,j) = q_i + \sum_{l=i+1}^{j} (q(l) + p(l))$

$w(i,j)$ is the extra cost for the tree containing nodes(a_{i+1} ,,.....a_j)

Expected cost for the tree with a_k as root

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

$W(0,\ k\text{-}1) = q_0 + \sum_{l=-1}^{k-1} (q(l) + p(l))$

$W(k\ ,\ n) = q_k + \sum_{l=k+1}^{n} (q(l) + p(l))$

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

$$p(k) + w(0,k-1) + w(k,n) = p_k + q_0 + \sum^{k-1}(p_i+q_i)$$
$$+ q_k + \sum_{k+1} (p_i + q_i)$$

$$= q_0 + \sum_1^k (p_i+q_i) + q_k + \sum_{k+1}^n (p_i+q_i)$$

so to minimize whole cost cost(l) as well as cost (r) both should individually be minimized.

Let

C(I,j) – cost of optimal binary search tree t_{ij} containing $a_{i+1} \ldots a_j$ and $E_i \ldots . E_j$

Cost (l) = C(0 , k-1)

Cost (r) = c(k , n)

Total cost of t_{0n}

$P_k + c(0, k-1) + c(k , n)\ w(0 , k-1) + w(k , n)$

So

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

$$C(I,j) = \min_{i<k \leq j}\{c(I,k-1)+c(k,j)+p_k+w(i,k-1)+w(k,j)\}$$

$$C(I,j) = \min\ \{c(I,k-1)\ ,\ c(k,\ j)\ \} + w(I,j)$$

Note:

$C(0,0) = 0$

$W(i,i) = q_i$

$w(0,1) = q(0) + p_1 + q_1$

$W(i,j) = q_i + \sum_{i+1}^{j}(p_l + q_l)$

$W(0\ ,\ 1) = q_0\ ,\ p_1 + q_1$

$W(1,2) = q_1 + p_2 + q_2$

$C(1,4) =$

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

$c(1,3)+c(4,4)\}$

Example: n=4

(a_1,a_2,a_3,a_4) =(do,if,int,while)

Let

P=(3,3,1,1)

Q=(2,3,1,1,1)

$W(I,i)= q_i$

C(i,i)= 0

And r(i,i)= 0

$W(i,j) = p_j + q_j + w(i, j-1)$

	0	1	2	3	4
0	2,0,0	3,0,0	1,0,0	1,0,0	1,0,0
1	8,8,1	7,7,2	3,3,3	3,3,4	
2	12,19,1	9,12,2	5,8,3		
3	14,25,2	22,19,2			
4	16,32,2				

$W_{i,i+j}$

$C_{i,i+j}$

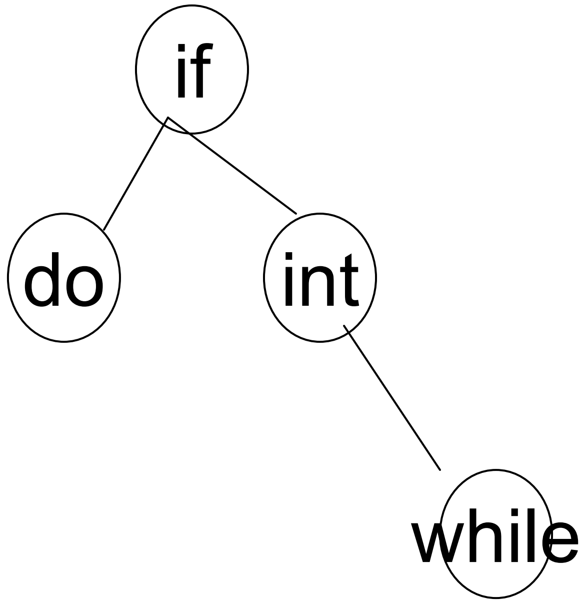
$R_{i,i+j}$

Optimal binary search tree

R(0,4)= 2 i.e. if

R(2,4)=3

i.e. int



	Plant1		Plant2		Plant 3	
Proposal	C1	r1	C2	r2	C3	r3
1	0	0	0	0	0	0
2	1	5	2	8	1	4
3	2	6	3	9	-	-
4	-	-	4	12	-	-

Each plant will only be permitted to work on one of its proposals.

The goal is to maximize the firm’s revenue resulting from the allocation of the $ 5 million.

A straight forward way to solve this is to try all possibilities and choose the best.

In this case there are only $3\times 4 \times 2 = 24$ ways of allocating the money.

Many of these are infeasible.

For example proposals 3 , 4 and 1 for the three plants

Let us break the problem into three stages:

Each stage represents the money allocated to a single plant.

So stage one represents the money allocated to plant 1

Stage 2 the money to plant 2

Stage 3 the money allocated to plant 3

Each stage is divided into states.A state encompasses the information required to go from one stage to the next

{0,1,2,3,4,5}: the amount of money spent on plant 1.

{0,1,2,3,4,5}: the amount of money spent on plants 1 and 2

{5}: the amount of money spent on plants 1,2 and 3.

Associated with each state is a revenue.

Note that to make a decision at stage 3, it is only necessary to know how much was spent on plants 1 and 2, not how it was spent.

The easy possibility is stage 1

If the available capital is	Then the optimal proposal is	the revenue for stage 1 is
0	1	0
1	2	5
2	3	6
3	3	6
4	3	6
5	3	6

We are now ready to compute for stage 2.

In this case we want to find the best solution for both plants 1 and 2.

We want to calculate the best revenue for a given amount for plants 1 and 2.

We simply go through all the plant 2 proposals, allocate the given amount of fund to it and use the above table to see how plant 1 will spend the remainder.

ppose we want to determine best allocation for state value 4.

proposal 1 of 2 gives revenue 0, leaves 4 for plant 1 which returns 6: total:6

proposal 2 of 2 gives revenue 8 leaves 2 for plant 1 which returns 6: total:14

proposal 3 of 2 gives revenue 9, leaves 1 for plant 1 which returns 5: total:6

proposal 4 of 2 gives revenue 12, leaves 0 for plant 1 which returns 0: total:12

If available capital x2	Then the optimal proposal is	And the revenue for stages 1 and 2 is
0	1	0
1	1	5
2	2	8
3	2	13
4	2 or 3	14
5	4	17

We can now go on to stage 3.

The only value we are interested in is x3 = 5.

One again we go through all the proposals for this stage, determine the amount of money remaining and use the above table to decide the value for previous stages.

Proposal 1 of plant 3 gives revenue 0 and leaves 5 to prev. stage giving revenue = 17

Proposal 2 of plant 3 gives revenue 4 leaves 4 prev stages give 14 so total revenue = 18

Here calculations are done recursively.

Stage 2 calculations are based on stage 1,

Stage 3 only on stage 2.

In fact given at a state all future decisions are made independent of how you got to the state.

This is principle of optimality.

Dynamic programming rests on this assumption.

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.*

All pair shortest paths

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

Let C be a cost adjacency matrix for G such that C(i,i)= 0 for all i.

All pair 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.

We assume that there is no cycle of negative length.

Let us examine a shortest path from i to j in G.

This path originates at vertex i and goes through some intermediate vertices and terminates at vertex j.

Let k be an intermediate vertex on this shortest path then the subpath from i to k and from k to j must be shortest paths from i to k and k to j.

So the principle of optimality holds.

If k is the intermediate vertex with highest index then the i to k path is a shortest 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 greater than k –1.

Let

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

$A(i, j)$ represent shortest distance between i and j

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

$$1 \le k \le n$$

Clearly

$$A^0(i,j) = \text{cost}(i, j)$$

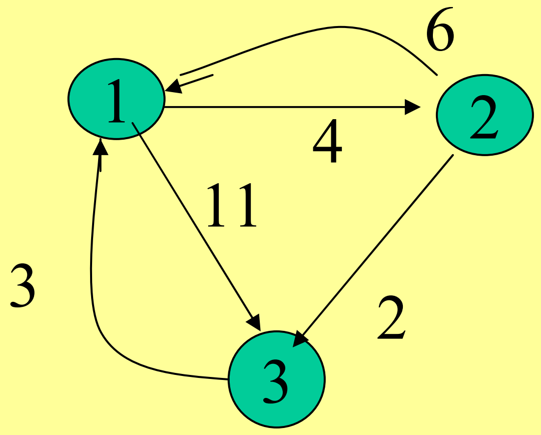
$$A^k(i, j) = \min\{A^{k-1}(i, j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$$

$$k \ge 1$$

Precedence Matrix for shortest paths

$$P_{ij}^{k} = \begin{cases} p^{k-1}_{\mathbf{ij}} & \text{if } A_{ij}^{k-1} \leq A_{ik}^{k-1} + A_{\mathbf{kj}}^{k-1} \\ p_{kj}^{k-1} & \text{if } A_{ij}^{k-1} > A_{\mathbf{ik}}^{k-1} + A_{\mathbf{kj}}^{k-1} \end{cases}$$

The Floyd Warshall Algorithm



A^0

0	4	11
6	0	2
3	∞	0

A^1

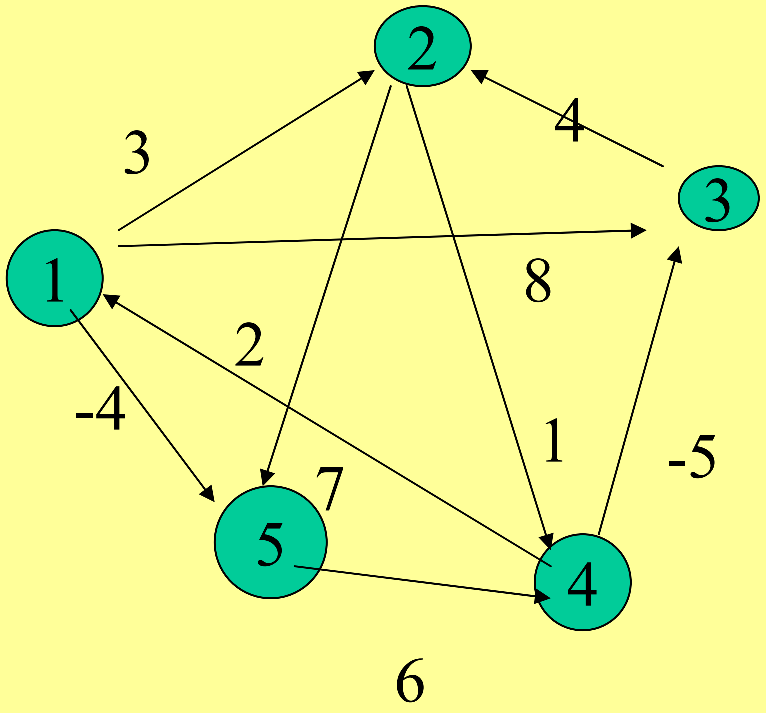
0	4	11
6	0	2
3	7	0

A^2

0	4	**6**
6	0	2
3	7	0

A^3

0	4	6
5	0	2
3	7	0


2
3
1
5
4
3
4
8
2
-4
1
-5
7
6

A^i

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

0	1	-3	2	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

P

^	1	1	^	1
^	^	^	2	2
^	3	^	^	^
4	^	4	^	^
^	^	^	5	^

^	1	1	^	1
^	^	^	2	2
^	3	^	^	^
4	1	4	^	1
^	^	^	5	^

^	1	1	2	1
^	^	^	2	2
^	3	^	2	2
4	1	4	^	1
^	^	^	5	^

^	1	1	2	1
^	^	^	2	2
^	3	^	2	2
4	3	4	^	1
^	^	^	5	^

^	1	4	2	1
4	^	4	2	1
4	3	^	2	1
4	3	4	^	^
4	3	4	5	^

^	3	4	5	1
4	^	4	2	1
4	3	^	2	1
4	3	4	^	1
4	3	4	5	^