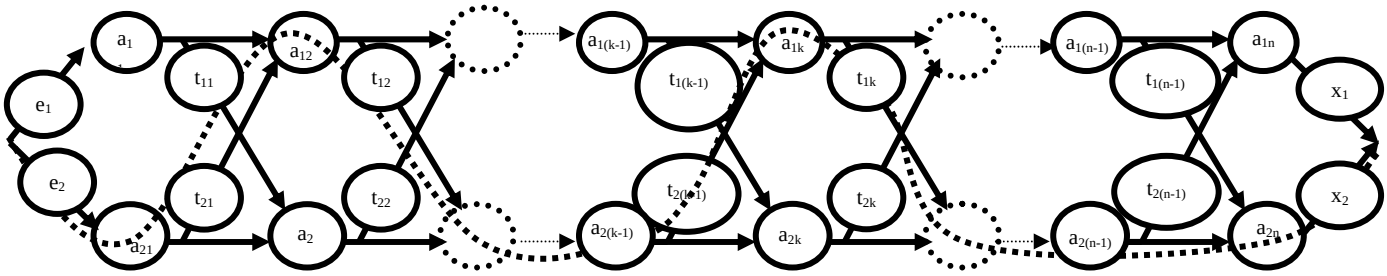
# *Dynamic Programming:*

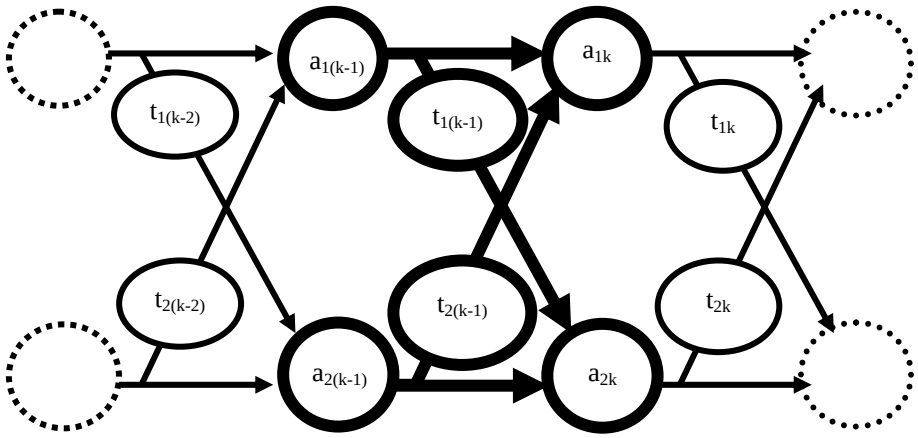
## *Example 1:* Assembly line scheduling. *Instance:*




***Task:*** Find the fastest way through the factory?

***Solution 1:*** brute force search calculates times for $2^n$ many possible paths, and compares them.

***Solution 2:*** recursion:




Denote the fastest way through the node $a_{j,k}$ by $m(j,k)$, $j=1,2$. The fastest way through $a_{1k}$ is either

through $a_{1(k-1)}$ or through $a_{2(k-1)}$ via $t_{2(k-1)}$. Similarly, the fastest way through $a_{2k}$ is either through $a_{2(k-1)}$ or through $a_{1(k-1)}$ via $t_{1(k-1)}$. Thus:

$$m(1, k) = \min\{\mathbf{m(1,k-1)} + a_{1k},\ \mathbf{m(2,k-1)} + t_{2(k-1)} + a_{1k}\}$$

$$m(2, k) = \min\{\mathbf{m(2, k-1)} + a_{2k},\ \mathbf{m(1, k-1)} + t_{1(k-1)} + a_{2k}\}$$

***Fastest-Way(a,t,e,x,n)***

1. $f_1[1] \leftarrow e_1 + a_{1,1}$
2. $f_2[1] \leftarrow e_2 + a_{2,1}$
3. **for** $j \leftarrow 2$ *to* $n$
4. **do if** $f_1[j-1] + a_{1,j} \le f_2[j-1] + t_{2,j-1} + a_{1,j}$
5. **then** $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$
6. **else** $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
7. **if** $f_2[j-1] + a_{2,j} \le f_1[j-1] + t_{1,j-1} + a_{2,j}$
8. **then** $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$
9. **else** $f_1[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
10. **if** $f_1[n] + x_1 \le f_2[n] + x_2$
11. **then** $f^* = f_1[n] + x_1$
12. **else** $f^* = f_2[n] + x_2$

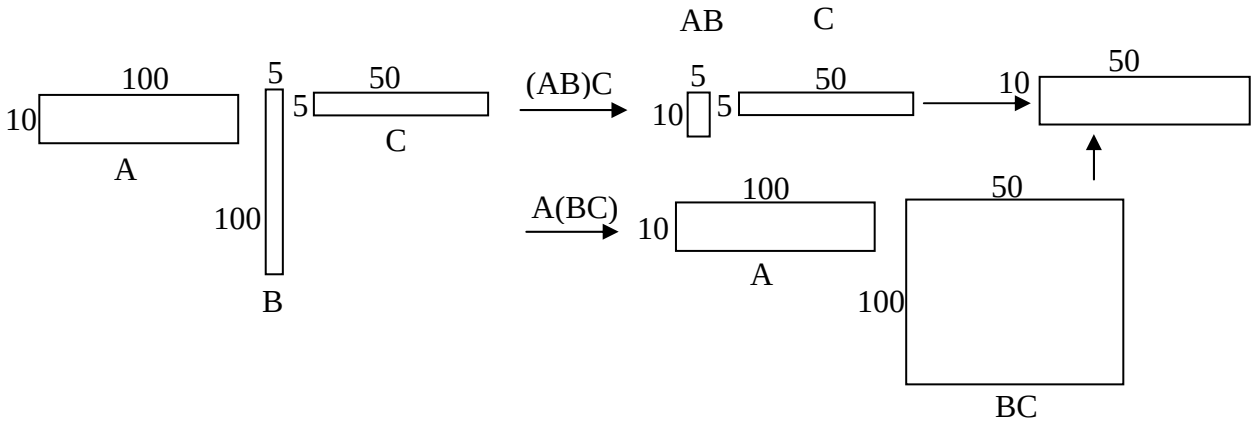
***Example 2:*** **Matrix chain multiplication**.

***Instance:*** A sequence of matrices $A_1A_2 \ldots A_n$.

***Task:*** Group them in such a way as to minimize the number of multiplications.

Assume for example that the matrices have the following dimensions:

*A = 10 x 100*
*B = 100 x 5*
*C = 5 x 50*




*(AB)C: 10 x 5 x 100+10 x 50 x 5=5000 + 2500=**7500***

*A(BC): 100 x 5 x 50+10 x 100 x 50=*
*25000 + 50000=**75000***

Thus, order of matrix multiplication can change the number of multiplications many times.

How many possible distribution of brackets are there to try out?

$$T(n) = \sum_{i=1}^{n-1} T(i)T(n-i)$$

The solution is a huge number (exponential in terms of the number of matrices: $T(n) = \Omega(2^n)$; it is equal to the number of binary trees with $n$ leaves, which is easy to see that it is exponential).

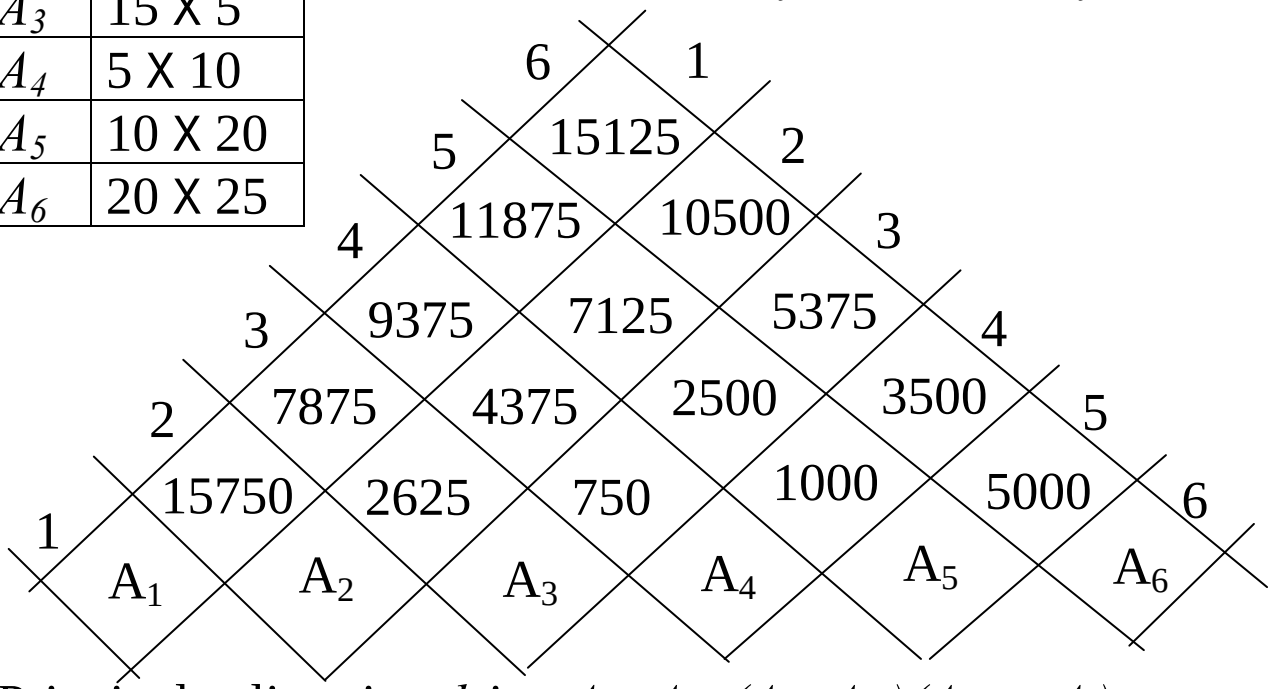
Thus, we cannot check them all. So what do we do? Eliminate repetitions in the calculation, and avoid calculations!!!! These different distributions of brackets significantly overlap!

We apply the same technique as in the previous example, building the table of optimal distribution of brackets for all sequences $A_i...A_j$ , such that in the $m^{th}$ raw we have all optimal distribution of brackets for all sequences $A_i...A_j$ for $j-i = m \geq 2$. In order to find optimal distribution for sequences $A_i...A_j$ of length $j-i = m+1$ we consider all splits $(A_i...A_k)(A_{k+1}...A_j)$ for all $k \leq i < j$ and use the table for $k-i$ and $j-(k+1)$ which have already been calculated:

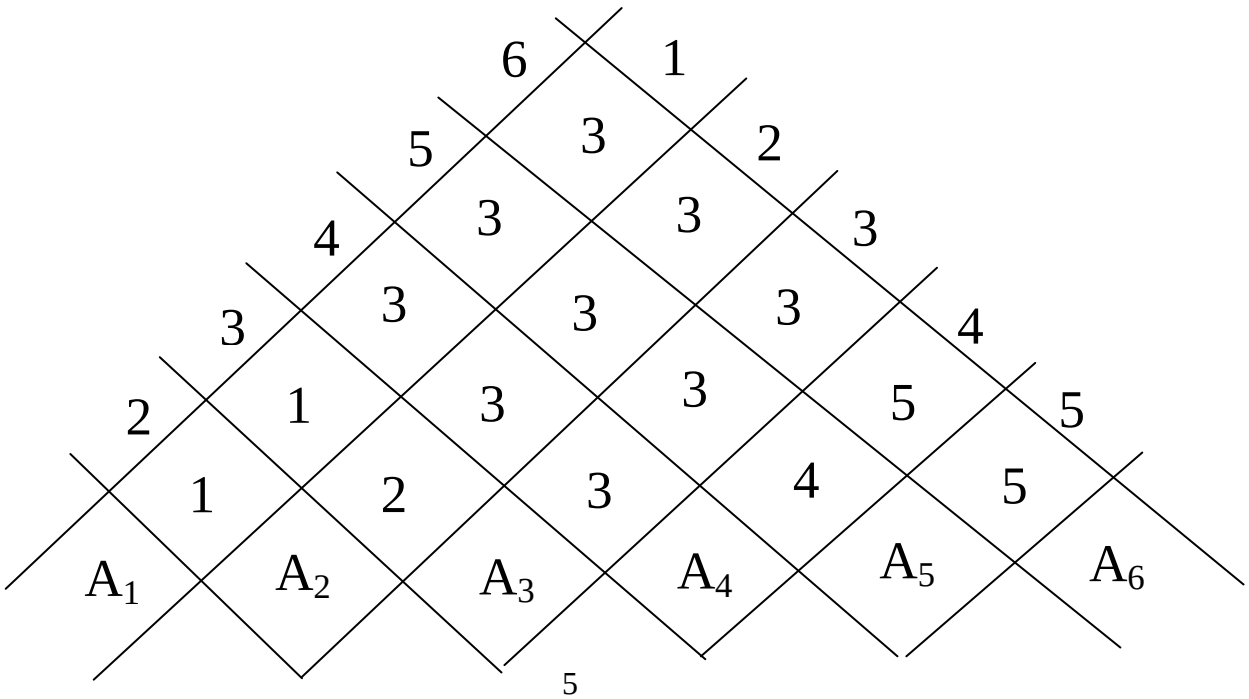
$m[i,j] = min\{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\ ,\ \ i \le k< j\}$

| mat | dim |
|---|---|
| $A_1$ | 30 X 35 |
| $A_2$ | 35 X 15 |
| $A_3$ | 15 X 5 |
| $A_4$ | 5 X 10 |
| $A_5$ | 10 X 20 |
| $A_6$ | 20 X 25 |

Here $A_i$ is of size $p_{i-1}p_i$ the product $A_i \ldots A_k$ is of size $p_{i-1}p_k$ and the product $A_{k+1} \ldots A_j$ of size $p_{k+1}p_j$




Principal split points $k$ i.e. $A_i \ldots A_j = (A_i \ldots A_k)(A_{k+1} \ldots A_j)$ :



## *Example 3:* Longest Common Subsequence

Assume we want to compare two DNA sequences A and B, to see if two viruses are related. Then this notion of similarity must be robust with respect to both

- short insertions and
- short deletions.

Thus, the best way is to see if two sequences share a long subsequence. Longer the joint subsequence – more similar A and B are.

***Instance:*** Two sequence $X$ and $Y$ over certain alphabet.

***Problem:*** Given $X$ and $Y$ find a ***longest common subsequence (LCS)*** for X and Y.

***Proposition:*** If $X=< x_1, x_2,...., x_m>$ and $Y=< y_1, y_2,...., y_n>$ are two sequences and $Z=< z_1, z_2,...., z_k>$ is a *LCS* of $X$ and $Y$, then

1. **If** $x_m = y_n$ **then** $z_k = x_m = y_n$ and $Z^*=< z_1, z_2,...., z_{k-1}>$ is an *LCS* for $X^*=< x_1, x_2,...., x_{m-1}>$ and $Y^*=< y_1, y_2,...., y_{n-1}>$.

2. **If** $x_m \neq y_n$ **and** $z_k \neq x_m$ then $Z$ is a *LCS* of $X^*=< x_1, x_2,...., x_{m-1}>$ and $Y=< y_1, y_2,...., y_n>$.

3. **If** $x_m \neq y_n$ **and** $z_k \neq y_n$ then $Z$ is a *LCS* of $X=< x_1, x_2,...., x_m>$ and $Y^*=< y_1, y_2,...., y_{n-1}>$.

Thus, the following recursion formula can holds:

$$c[i,j]=\begin{cases} 0 & \text{if i=0 or j=0} \\ c[i-1, j-1] +1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

If $c[i, j-1] = c[i-1, j]$ we chose (arbitrarily) which one we take.
Also, instead of a recursive procedure which would be exponential, we use dynamic programming to avoid repetitions of computations. The arrows encode which subsequences are taken, as it is clear from below:

```
LCS-LENGTH(X, Y)
 1  m ← length[X]
 2  n ← length[Y]
 3  for i ← 1 to m
 4       do c[i, 0] ← 0
 5  for j ← 0 to n
 6       do c[0, j] ← 0
 7  for i ← 1 to m
 8       do for j ← 1 to n
 9            do if x_i = y_j
10                 then c[i, j] ← c[i − 1, j − 1] + 1
11                      b[i, j] ← "↖"
12                 else if c[i − 1, j] ≥ c[i, j − 1]
13                      then c[i, j] ← c[i − 1, j]
14                           b[i, j] ← "↑"
15                      else c[i, j] ← c[i, j − 1]
16                           b[i, j] ← "←"
17  return c and b
```

| $i$ | | $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| | | $y_j$ | | B | D | C | A | B | A |
| 0 | $x_i$ | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | | 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↖ 1 | ← 1 | ↖ 1 |
| 2 | B | | 0 | ↖ 1 | ← 1 | ← 1 | ↑ 1 | ↖ 2 | ← 2 |
| 3 | C | | 0 | ↑ 1 | ↑ 1 | ↖ 2 | ← 2 | ↑ 2 | ↑ 2 |
| 4 | B | | 0 | ↖ 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ 3 | ← 3 |
| 5 | D | | 0 | ↑ 1 | ↖ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 |
| 6 | A | | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ 3 | ↑ 3 | ↖ 4 |
| 7 | B | | 0 | ↖ 1 | ↑ 2 | ↑ 2 | ↑ 3 | ↖ 4 | ↑ 4 |

## *Example 4:* Optimal binary search trees

***Instance***: Given are: ***(1)*** A sequence of $n$ distinct keys $K = <k_1, k_2, \ldots, k_n>$ in sorted order. For each key $k_i$ we have the probability $p_i$ that a search will be for $k_i$; ***(2)*** A sequence of "dummy keys" $D= <d_0, d_1, \ldots, d_n>$ with the corresponding probabilities $q_i$. The key $d_i$ represent values which would be between $k_i$ and $k_{i+1}$, but are not in $K$. Thus, searches for values between $k_i$ and $k_{i+1}$ terminate by reaching $d_i$, which happens with probability $q_i$.

***Problem:*** Find a binary search tree with ***minimal expected value*** of the search time.

Note that $q_0$ is the probability of a search for keys smaller than $k_1$, and $q_n$ is the probability of a search for keys larger than $k_n$.

Thus, the probabilities should satisfy $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$

*Recall: a* ***binary search tree*** *is a* binary tree with keys as nodes such that every node (key) in the left sub-tree of any key $k$ is smaller (or equal) than $k$ and every key in the right sub-tree of $k$ is larger (or equal) than $k$. (In our case all keys are assumed to be different).

*What is the expected search time for a binary search tree?*

$$\begin{aligned} \mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i \end{aligned}$$

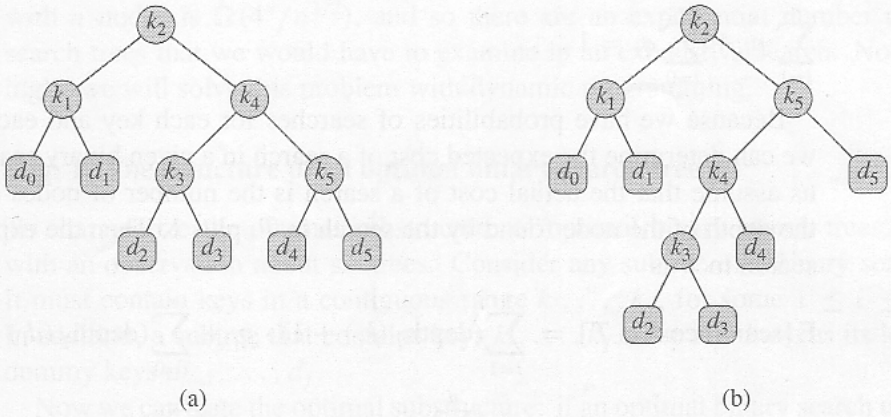



**Figure 15.7** Two binary search trees for a set of $n = 5$ keys with the following probabilities:

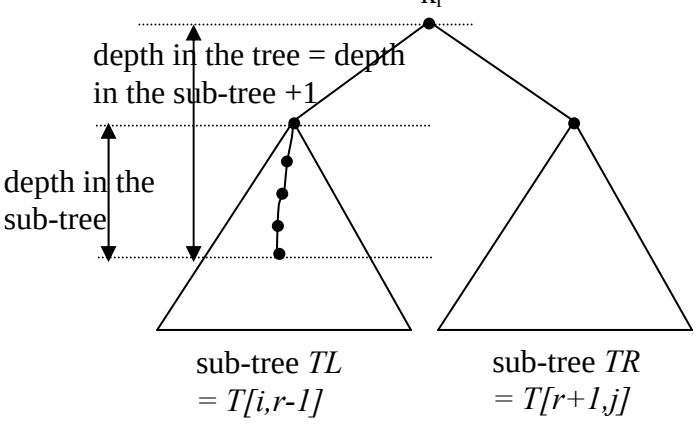
| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $p_i$ | | 0.15 | 0.10 | 0.05 | 0.10 | 0.20 |
| $q_i$ | 0.05 | 0.10 | 0.05 | 0.05 | 0.05 | 0.10 |

(**a**) A binary search tree with expected search cost 2.80. (**b**) A binary search tree with expected search cost 2.75. This tree is optimal.

Notice that the optimal tree (b) is less balanced than the tree (a)!

How to get optimal binary search tree? We use the (obvious) fact that if $r$ is the root of an optimal binary search tree, then both of its sub-trees are optimal binary search trees for the corresponding subsets of keys. Let $w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$ and let us denote the expected search time for the optimal binary search tree for keys $k_i, k_{i+1}, \ldots, k_j$ by $e[i, j]$.

The depth of each element in the entire tree is 1 plus its depth in the sub-tree. Thus, if the root is $k_r$ we get, by multiplying this 1 by the corresponding probability and taking all of these probabilities together:+ ($P_k$ stands for either $p_k$ or $q_k$)




$$e[i,j] = p_r + \sum_{k\in TL}(d_k+1)P_k + \sum_{k\in TR}(d_k+1)P_k =$$

$$p_r + \sum_{k\in TL} d_k P_k + \sum_{k\in TL} P_k + \sum_{k\in TR} d_k P_k + \sum_{k\in TR} P_k =$$

$$\sum_{k\in TL} d_k P_k + \sum_{k\in TR} d_k P_k + p_r + \sum_{k\in TL} P_k + \sum_{k\in TR} P_k =$$

$$e[i,r-1] + e[r+1,j] + (p_r + \sum_{k\in TL} P_k + \sum_{k\in TR} P_k) =$$

$$e[i,r-1] + e[r+1,j] + w[i,j]$$

We choose the root that gives the lowest expected search cost, giving us our final recursive formulation:

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j \end{cases}$$

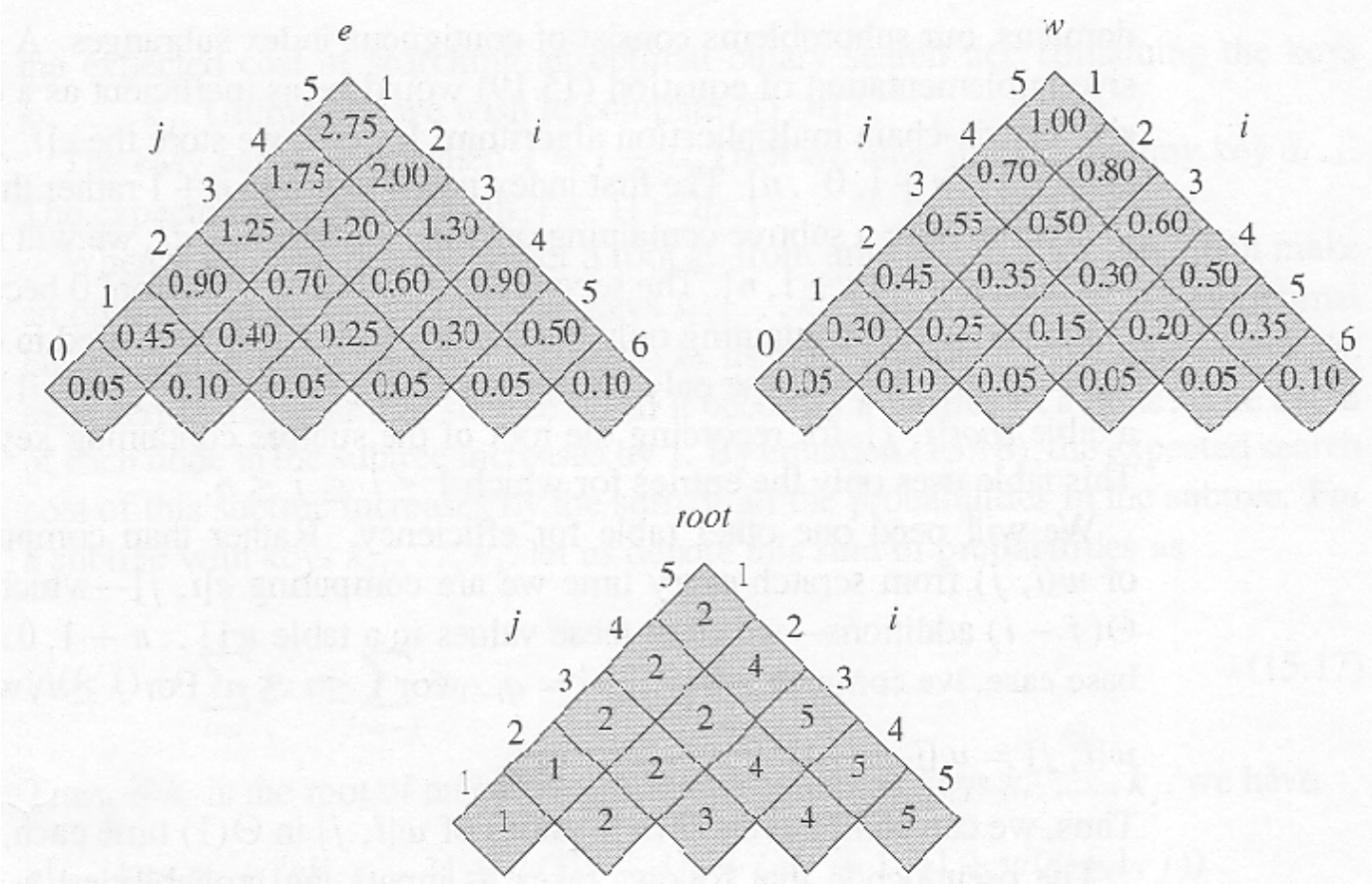
We now proceed as before by building the table for all sub-trees over $m$ consecutive keys $k_i$, $k_j$, $j$-$i = m$.

```
OPTIMAL-BST(p, q, n)
 1  for i ← 1 to n + 1
 2       do e[i, i − 1] ← q_{i−1}
 3          w[i, i − 1] ← q_{i−1}
 4  for l ← 1 to n
 5       do for i ← 1 to n − l + 1
 6             do j ← i + l − 1
 7                e[i, j] ← ∞
 8                w[i, j] ← w[i, j − 1] + p_j + q_j
 9                for r ← i to j
10                   do t ← e[i, r − 1] + e[r + 1, j] + w[i, j]
11                      if t < e[i, j]
12                         then e[i, j] ← t
13                              root[i, j] ← r
14  return e and root
```

Example: for

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $p_i$ | | 0.15 | 0.10 | 0.05 | 0.10 | 0.20 |
| $q_i$ | 0.05 | 0.10 | 0.05 | 0.05 | 0.05 | 0.10 |

we get

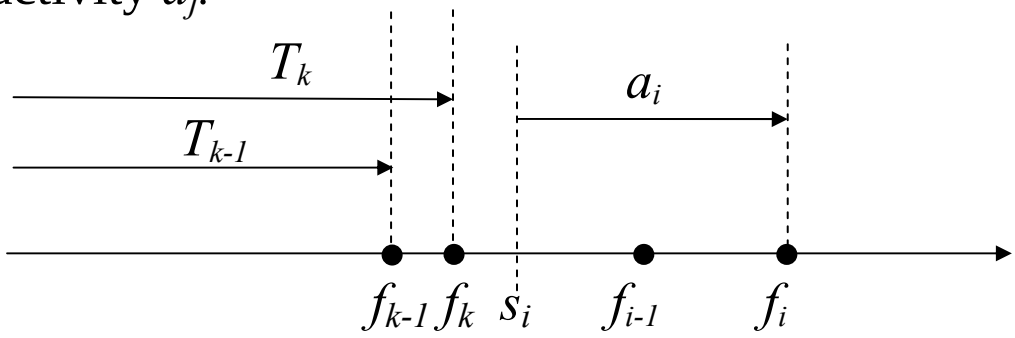


## *Example 5:* **Optimal resource use**

***Instance:*** A list of activities $a_i$, $1 \le i \le m$ with starting times $s_i$ and finishing times $f_i$. No two activities can take place simultaneously.

***Task:*** Find a subset of compatible activities of maximal total duration.

Order activities according to non decreasing finishing times $f_1 \ldots f_m$. For each $i$ we find optimal choice among all sequences which end with activity $a_i$. For $i=1$ simply take $a_1$. Assume that we have solved the problem for all $j<i$, obtaining maximal duration time $T_j$ of all sequences of activities which end with activity $a_j$.




$T_i = max\{T_j + f_i - s_i, j < i$ and activity $a_j$ finishes before the starting time of activity $a_i\}$

Finally let $T_{max}=max\{T_k : k \le n\}$.