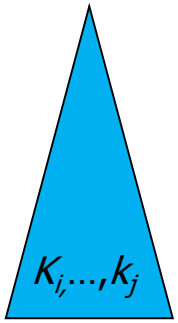
A Recursive Solution

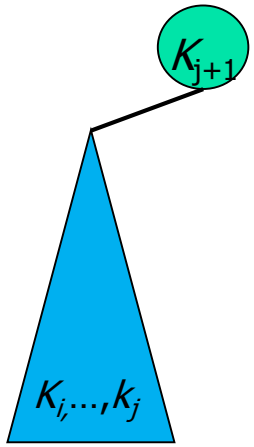
- Let us define e[i, j] as the expected cost of searching an optimal binary search tree containing the keys $k_i,\ldots,k_j$. Ultimately, we wish to compute e[1, n].
  - When j=i-1, we have just the dummy key d_{i-1} .

    The expected search cost is $e[i, i-1] = q_{i-1}$.
  - When $j \geq i$, we need to select a root k_r from among $k_i,\ldots, k_j$ and then make an optimal binary search tree with keys $k_i,\ldots,k_{r-1}$ as its left subtree and an optimal binary search tree with keys $k_{r+1},\ldots,k_j$ as its right subtree.

A Recursive Solution



$$e[i, j] = \sum_{l=i}^{j} (depth_T(k_l) + 1) \cdot p_l + \sum_{l=i-1}^{j} (depth_T(d_l) + 1) \cdot q_l$$



$$e'[i, j] = \sum_{l=i}^{j} (depth_T(k_l) + 1 + 1) \cdot p_l + \sum_{l=i-1}^{j} (depth_T(d_l) + 1 + 1) \cdot q_l$$

$$= e[i, j] + \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

A Recursive Solution

- What happens to the expected search cost of a subtree when it becomes a subtree of a node?
  - The depth of each node in the subtree increases by 1.
  - The expected search cost of this subtree

$$e[i, j] = \sum_{l=i}^{j} (depth_T(k_l) + 1) \cdot p_l + \sum_{l=i-1}^{j} (depth_T(d_l) + 1) \cdot q_l$$

    increases by the sum of all the probabilities in the subtree.

$$e'[i, j] = \sum_{l=i}^{j} (depth_T(k_l) + 1 + 1) \cdot p_l + \sum_{l=i-1}^{j} (depth_T(d_l) + 1 + 1) \cdot q_l$$

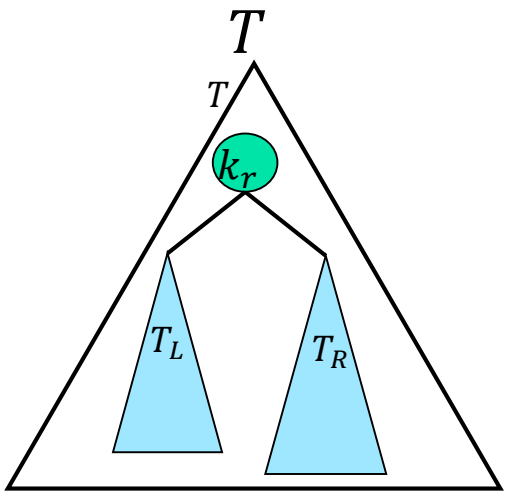
$$= e[i, j] + \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

  - For a subtree with keys $k_i, \ldots, k_j$, let us denote this sum of probabilities as

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

Recursive Solution

- Assume that k_r is the root of an optimal subtree T containing keys $k_i, \ldots, k_j$
- Let T_L be the left subtree of T
- Let T_R be the right subtree of T

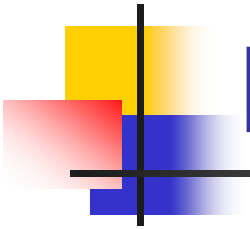


A Recursive Solution

- If k_r is the root of an optimal subtree containing keys $k_i,\ldots,k_j$, we have
  - $e[i, j] = p_r + e'[i, r-1]+e'[r+1, j]$
    $= p_r +(e[i, r-1]+w(i, r-1))+(e[r+1, j]+w(r+1, j))$

- Since $w(i, j) = w(i, r-1)+p_r+w(r+1, j)$,
  - $e[i, j] = e[i, r-1]+e[r+1, j]+ w(i, j)$

- We choose the root that gives the lowest expected search cost, giving us our final recursive formulation:
  - $e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1]+e[r+1, j]+w(i, j)\} & \text{if } i \le j \end{cases}$



Dynamic Programming

- A direct, recursive implementation of the recursive equation would be as inefficient as a direct, recursive matrix chain multiplication algorithm.
- Instead, we store the e[i, j] values in a table e[1..n+1, 0..n].
  - The first index needs to run to n+1 rather than n because in order to have a subtree containing only the dummy key d_n, we need to compute and store e[n+1,n].
  - The second index needs to start from 0 because in order to have a subtree containing only the dummy key d_0, we need to compute and store e[1,0].
  - We use only the entries e[i, j] for which $j \geq i - 1$.
- We also use a table root[i, j], for recording the root of the subtree containing keys $k_i,\ldots,k_j$.
  - This table uses only the entries for which $1 \leq i \leq j \leq n$.

Improving Efficiency

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

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

- Rather than compute the value of w(i, j) from scratch every time we are computing e[i, j] with $\Theta(j - i)$ additions, we store these values in a table w[1..n+1, 0..n].
  - For the base case, $w[i, i-1] = q_{i-1}$ for $1 \le i \le n$.
  - For $j \ge i$, $w[i, j] = w[i, j-1] + p_j + q_j$.
- We can compute the $\Theta(n^2)$ values of w[i, j] in $\Theta(1)$ time each.

Dynamic Programming

```
OPTIMAL-BST(p, q, n)
1.    for i = 1 to n+1
2.        e[i, i-1] = q_{i-1}
3.        w[i, i-1] = q_{i-1}
4.    for l = 1 to n
5.        for i = 1 to n-l+1
6.            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.               t = e[i, r-1]+e[r+1, j]+w[i, j]
11.               if t < e[i, j]
12.                   e[i, j] = t
13.                   root[i, j] = r
14.   return e and root
```

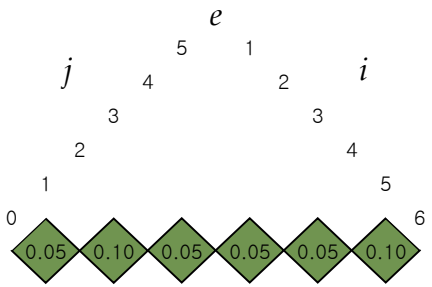
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1]+e[r+1, j]+w(i, j)\} & \text{if } i \le j \end{cases}$$

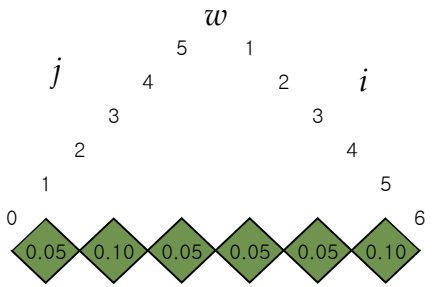
$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

- The OPTIMAL-BST procedure takes $\Omega(n^3)$ time.

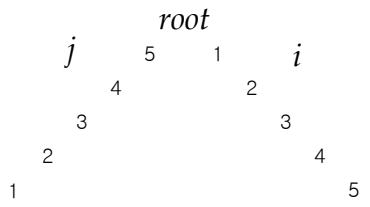
Optimal Binary Search Trees - How Does the Algorithm Work?

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





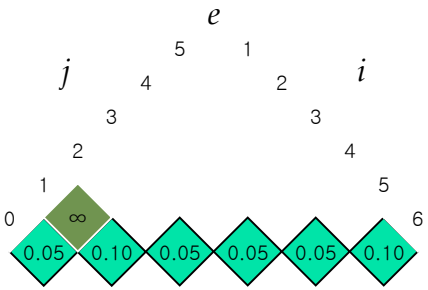
$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

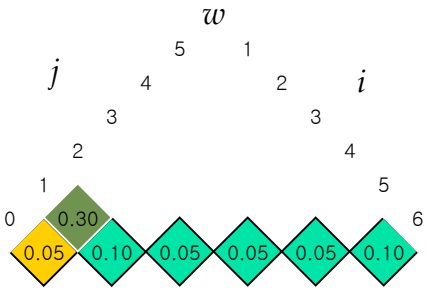


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

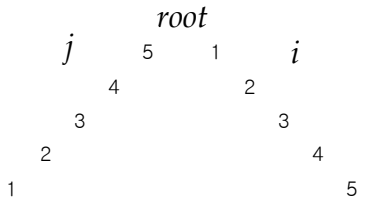
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

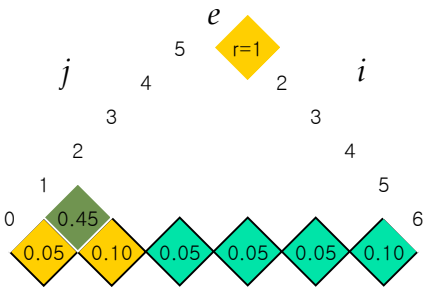
$$w[1,1] = w[1,0] + p_1 + q_1 = 0.30$$

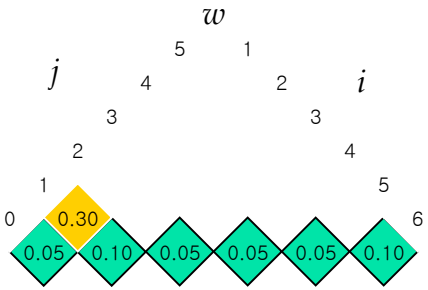


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

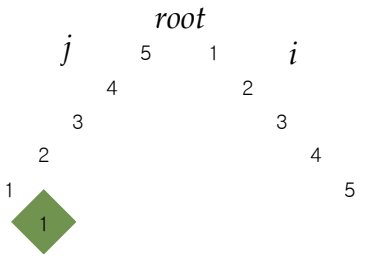
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

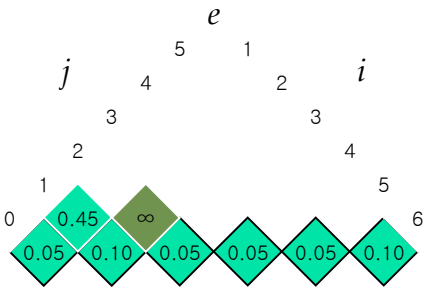
$$e[1,1] = e[1,0] + e[2,1] + w[1,1] = 0.45$$

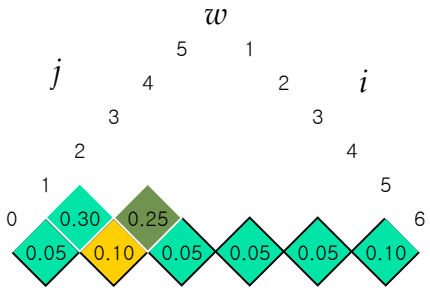


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

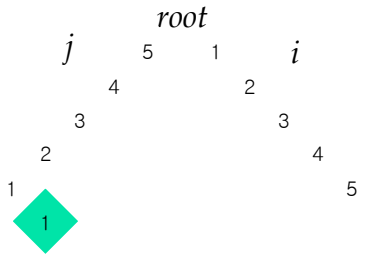
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

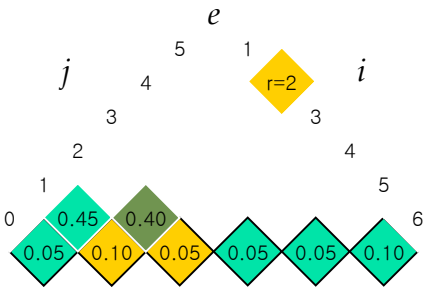
$$w[2,2] = w[2,1] + p_2 + q_2 = 0.25$$

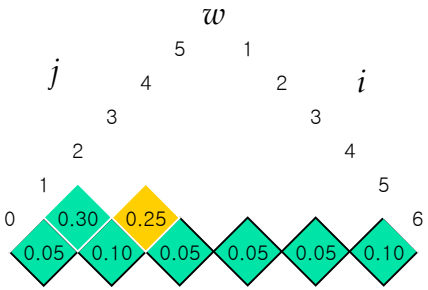


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

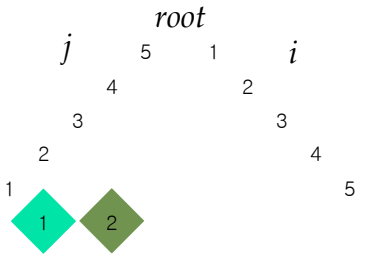
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

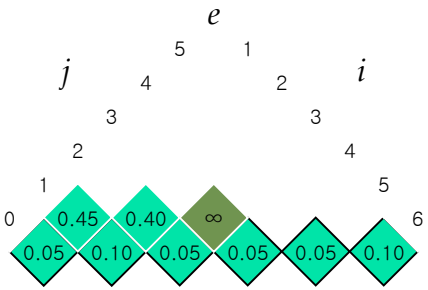
$$e[2,2] = e[2,1] + e[3,2] + w[2,2] = 0.40$$

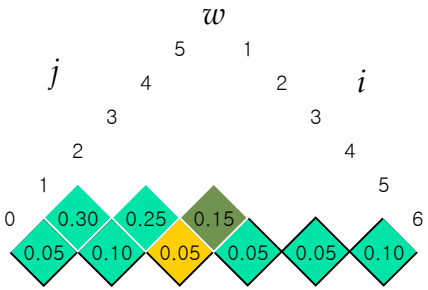


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

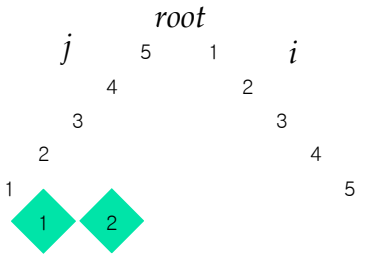
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

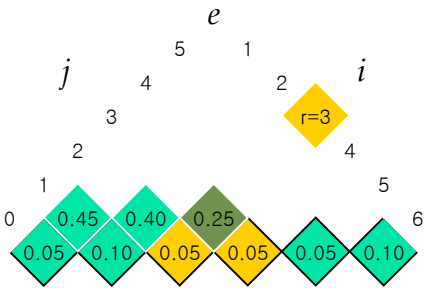
$$w[3,3] = w[3,2] + p_3 + q_3 = 0.15$$

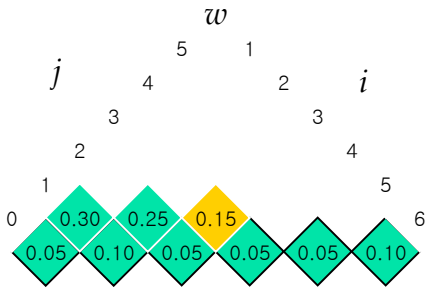


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

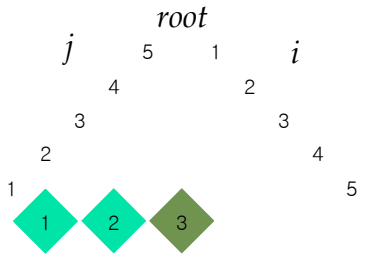
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

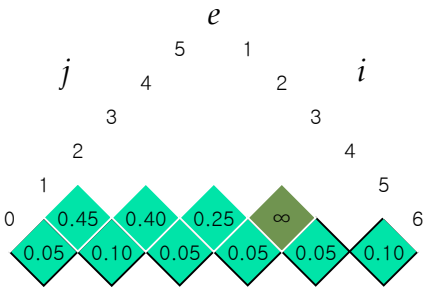
$$e[3,3] = e[3,2] + e[4,3] + w[3,3] = 0.25$$

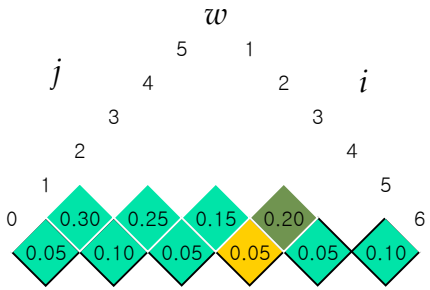


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

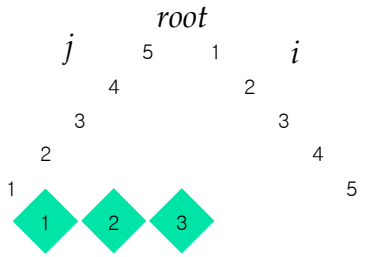
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

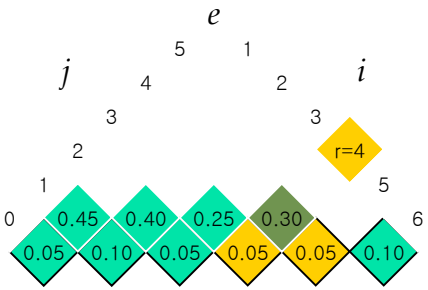
$$w[4,4] = w[4,3] + p_4 + q_4 = 0.20$$

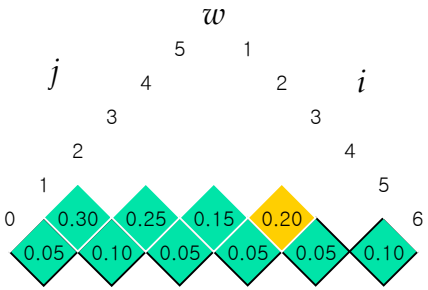


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

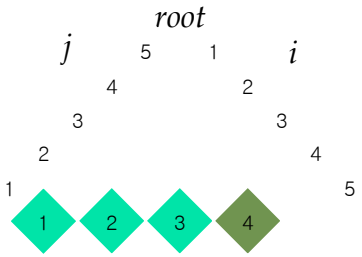
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

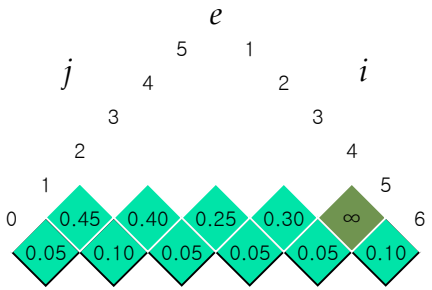
$$e[4,4] = e[4,3] + e[5,4] + w[4,4] = 0.30$$

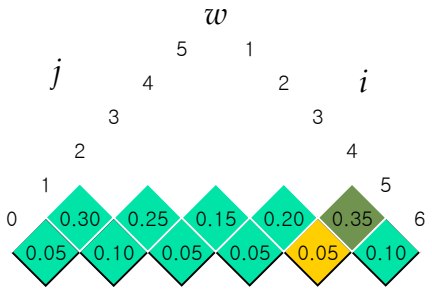


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

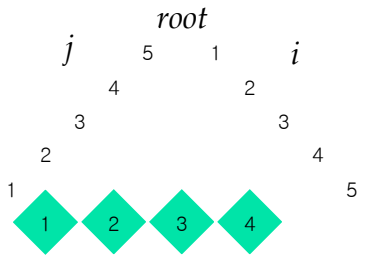
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

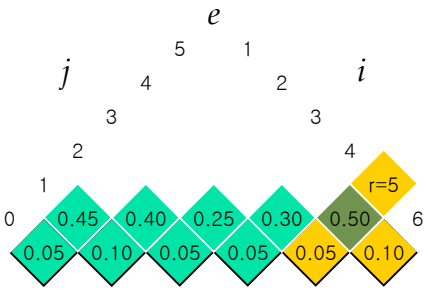
$$w[5,5] = w[5,4] + p_5 + q_5 = 0.35$$

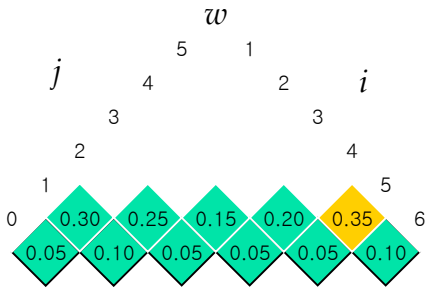


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

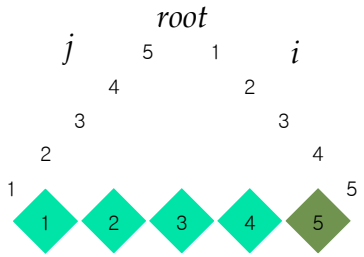
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

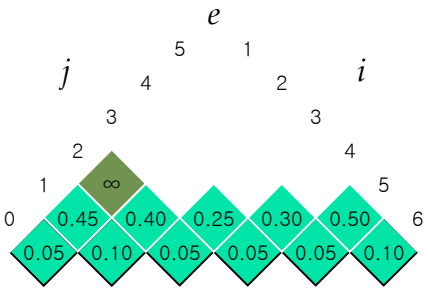
$$e[5,5] = e[5,4] + e[6,5] + w[5,5] = 0.50$$

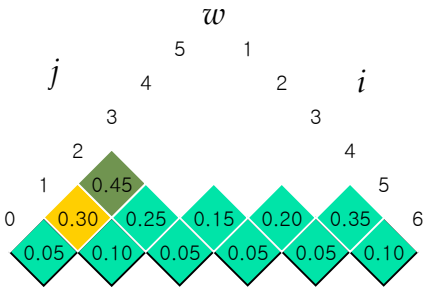


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

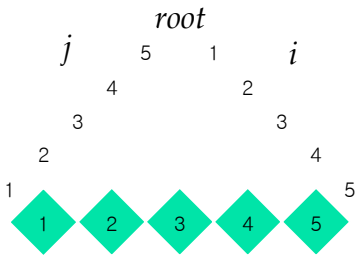
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

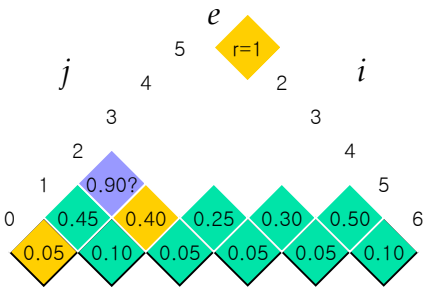
$$w[1,2] = w[1,1] + p_2 + q_2 = 0.45$$

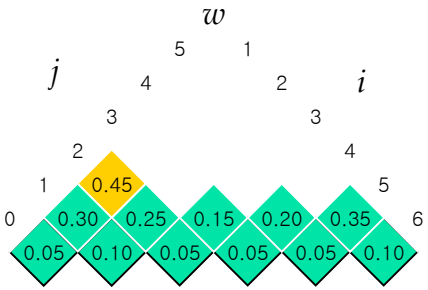


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

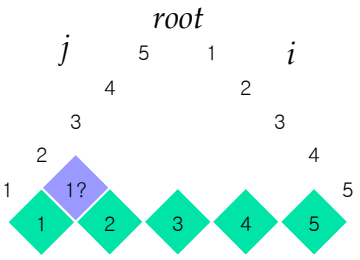
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

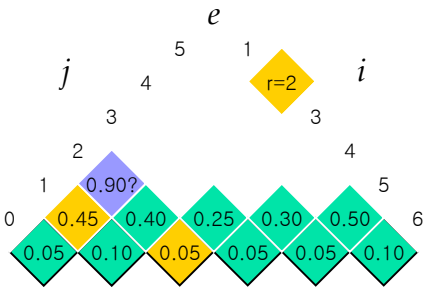
$$e[1,2] = min \begin{cases} e[1,0] + e[2,2] + w[1,2] = 0.90 \\ e[1,1] + e[3,2] + w[1,2] = ?? \end{cases}$$

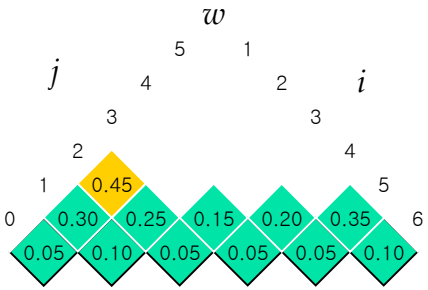


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

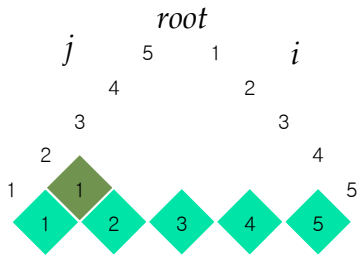
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

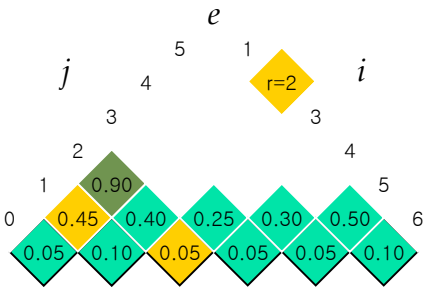
$$e[1,2] = min \begin{cases} e[1,0] + e[2,2] + w[1,2] = 0.90 \\ e[1,1] + e[3,2] + w[1,2] = 0.95 \end{cases}$$

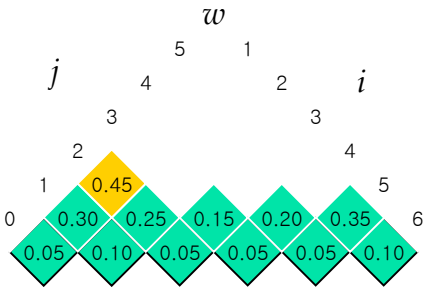


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

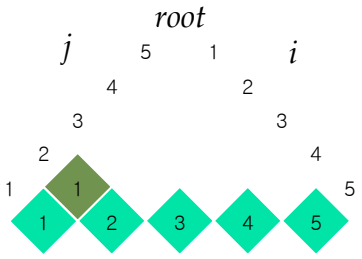
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

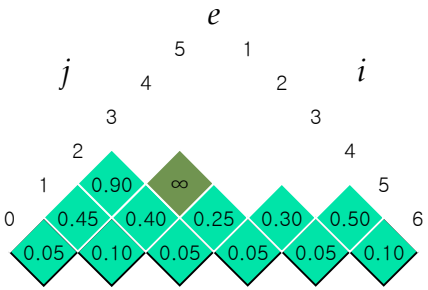
$$e[1,2] = min \begin{cases} \boldsymbol{e[1,0] + e[2,2] + w[1,2] = 0.90} \\ e[1,1] + e[3,2] + w[1,2] = 0.95 \end{cases}$$

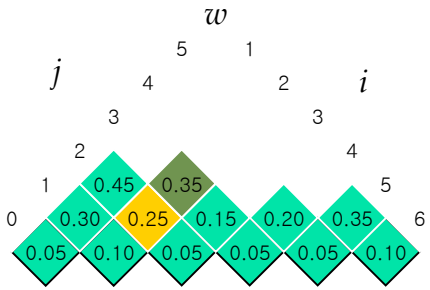


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

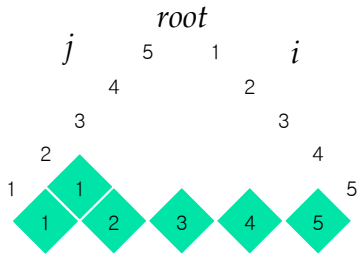
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

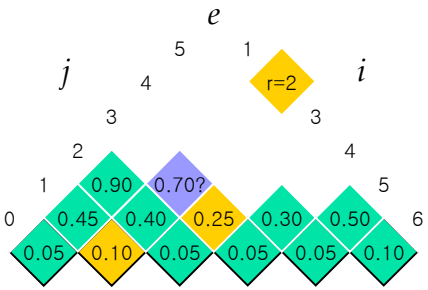
$$w[2,3] = w[2,2] + p_3 + q_3 = 0.35$$

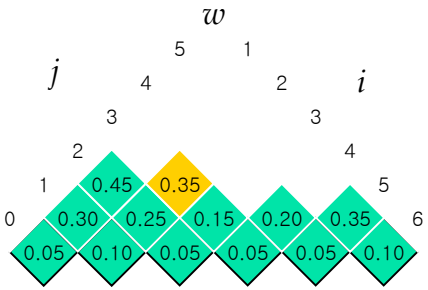


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

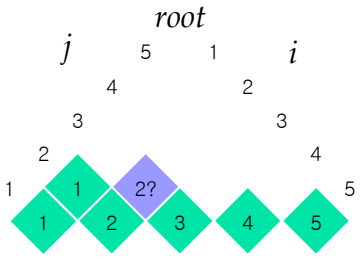
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

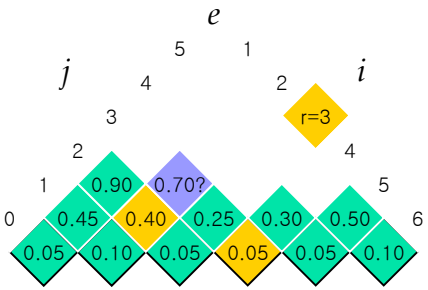
$$e[2,3] = min \begin{cases} e[2,1] + e[3,3] + w[2,3] = 0.70 \\ e[2,2] + e[4,3] + w[2,3] = ?? \end{cases}$$

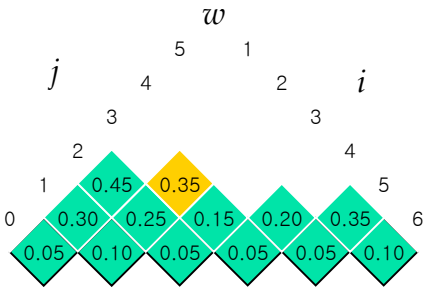


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

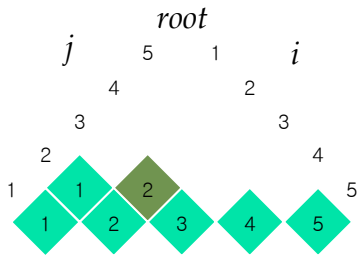
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

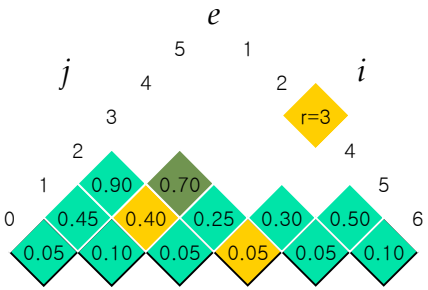
$$e[2,3] = min \begin{cases} e[2,1] + e[3,3] + w[2,3] = 0.70 \\ e[2,2] + e[4,3] + w[2,3] = 0.80 \end{cases}$$

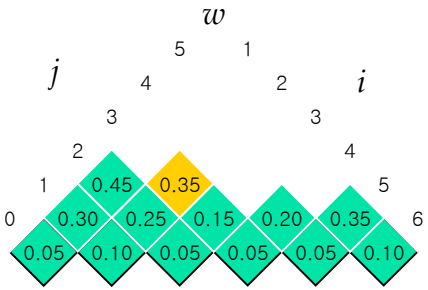


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

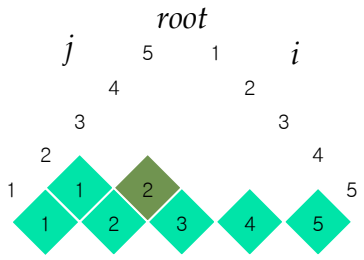
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

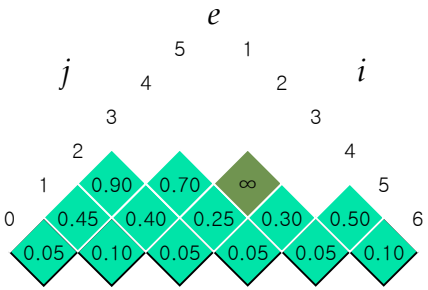
$$e[2,3] = min \begin{cases} \boldsymbol{e[2,1] + e[3,3] + w[2,3] = 0.70} \\ e[2,2] + e[4,3] + w[2,3] = 0.80 \end{cases}$$

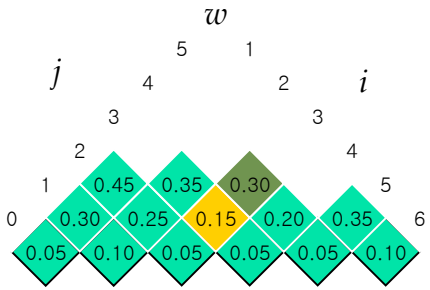


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

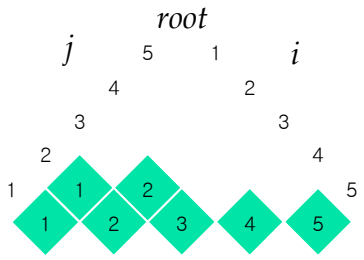
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

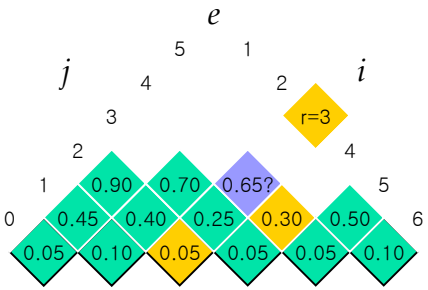
$$w[3,4] = w[3,3] + p_4 + q_4 = 0.30$$

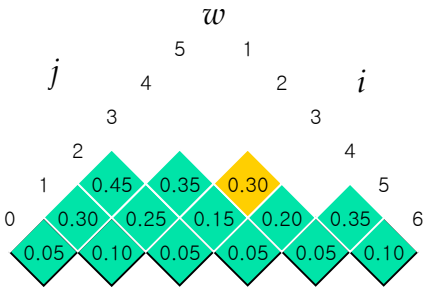


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

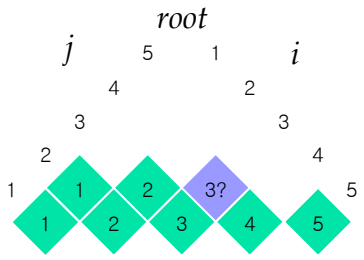
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

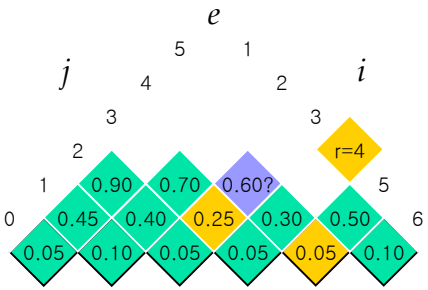
$$e[3,4] = min \begin{cases} e[3,2] + e[4,4] + w[3,4] = 0.65 \\ e[3,3] + e[5,4] + w[3,4] = ?? \end{cases}$$

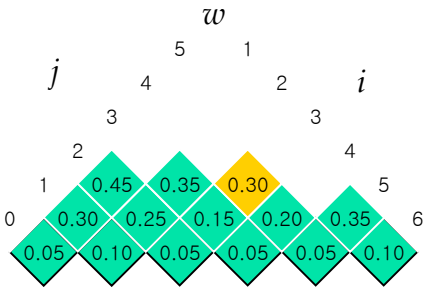


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

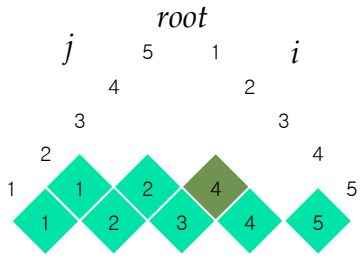
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

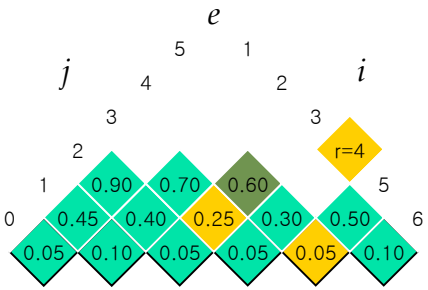
$$e[3,4] = min \begin{cases} e[3,2] + e[4,4] + w[3,4] = 0.65 \\ e[3,3] + e[5,4] + w[3,4] = 0.60 \end{cases}$$

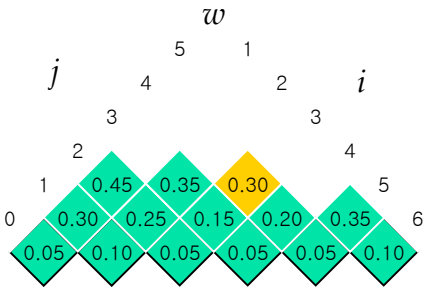


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

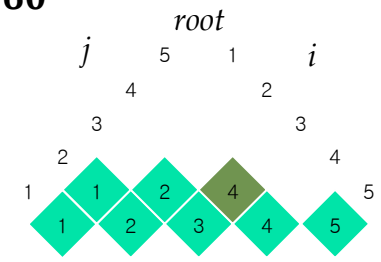
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

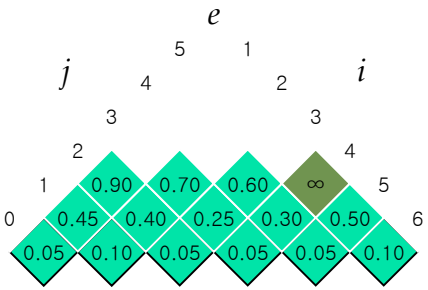
$$e[3,4] = min\begin{cases} e[3,2] + e[4,4] + w[3,4] = 0.65 \\ \boldsymbol{e[3,3] + e[5,4] + w[3,4] = 0.60} \end{cases}$$

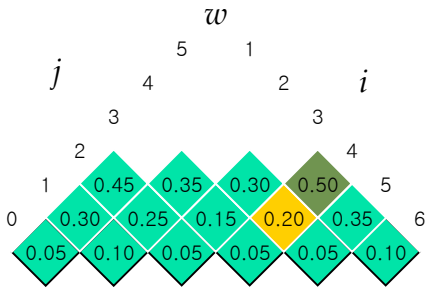


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

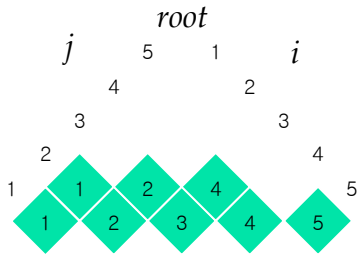
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

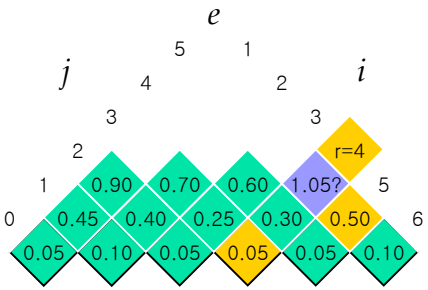
$$w[4,5] = w[4,4] + p_5 + q_5 = 0.50$$

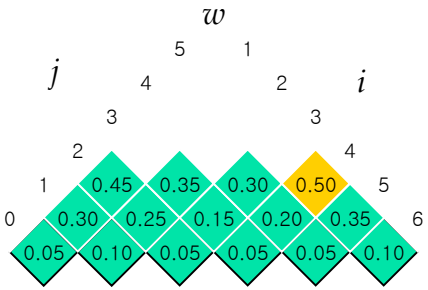


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

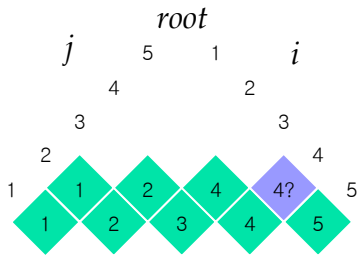
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

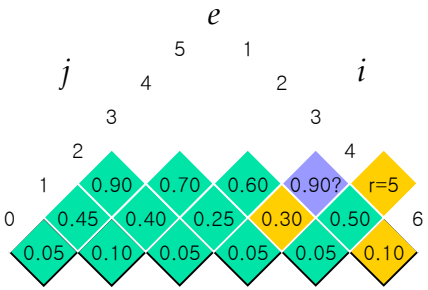
$$e[4,5] = min \begin{cases} e[4,3] + e[5,5] + w[4,5] = 1.05 \\ e[4,4] + e[6,5] + w[4,5] = ?? \end{cases}$$

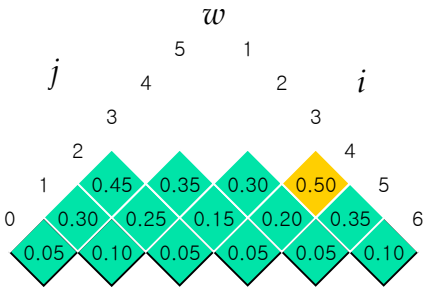


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

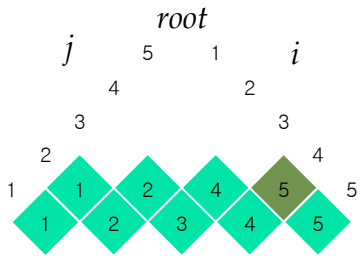
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

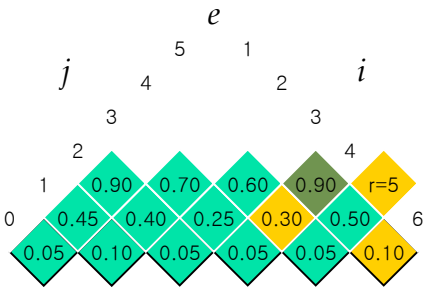
$$e[4,5] = min \begin{cases} e[4,3] + e[5,5] + w[4,5] = 1.05 \\ e[4,4] + e[6,5] + w[4,5] = 0.90 \end{cases}$$

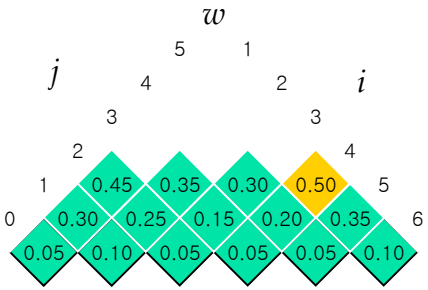


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

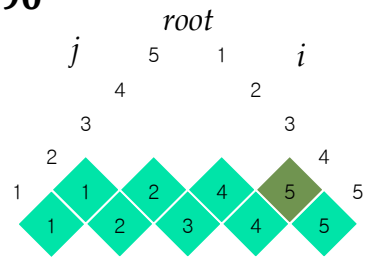
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

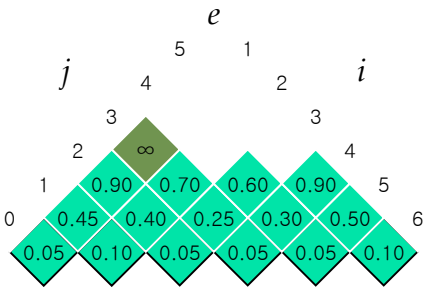
$$e[4,5] = min \begin{cases} e[4,3] + e[5,5] + w[4,5] = 1.05 \\ \boldsymbol{e[4,4] + e[6,5] + w[4,5] = 0.90} \end{cases}$$

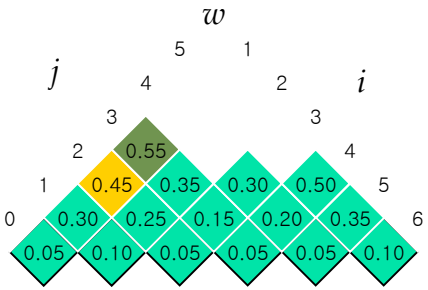


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

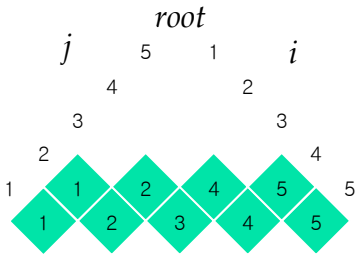
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

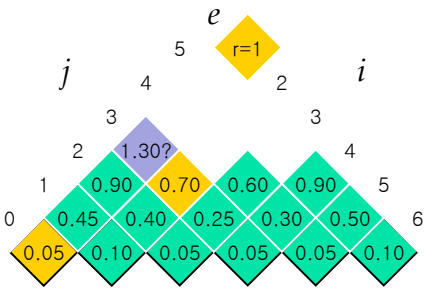
$$w[1,3] = w[1,2] + p_3 + q_3 = 0.55$$

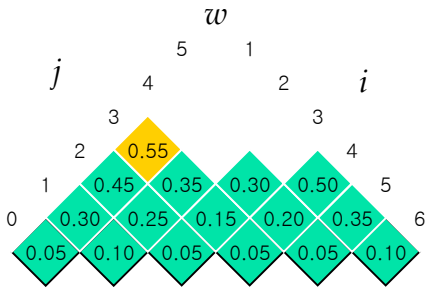


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

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

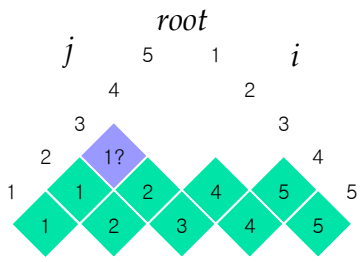




$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

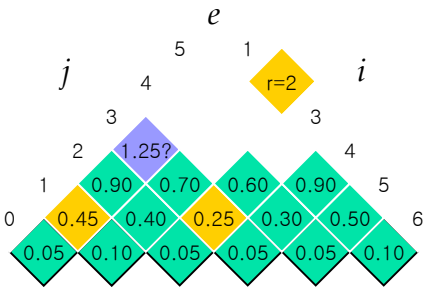
$$e[1,3] = min \begin{cases} e[1,0] + e[2,3] + w[1,3] = 1.30 \\ e[1,1] + e[3,3] + w[1,3] = ?? \\ e[1,2] + e[4,3] + w[1,3] = ?? \end{cases}$$

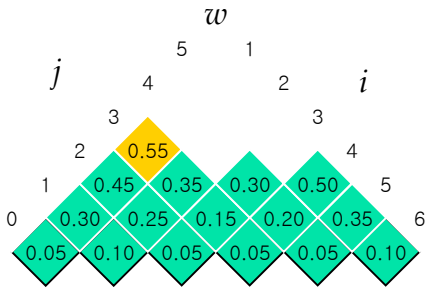
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10



Optimal Binary Search Trees - How Does the Algorithm Work?

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

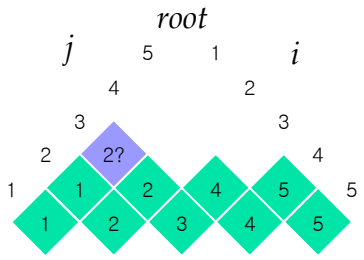




$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

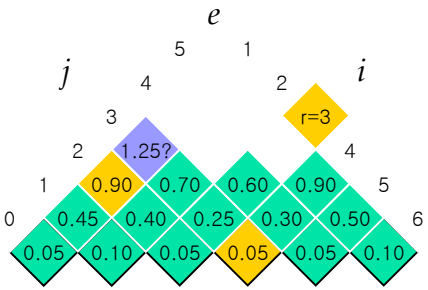
$$e[1,3] = min \begin{cases} e[1,0] + e[2,3] + w[1,3] = 1.30 \\ e[1,1] + e[3,3] + w[1,3] = 1.25 \\ e[1,2] + e[4,3] + w[1,3] = ?? \end{cases}$$

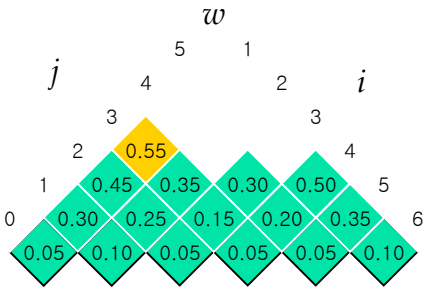
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10



Optimal Binary Search Trees - How Does the Algorithm Work?

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

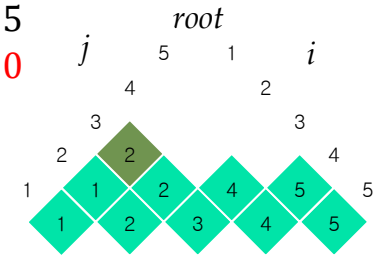




$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

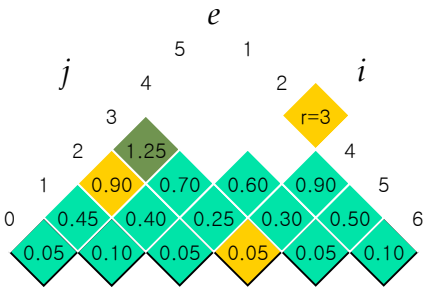
$$e[1,3] = min \begin{cases} e[1,0] + e[2,3] + w[1,3] = 1.30 \\ e[1,1] + e[3,3] + w[1,3] = 1.25 \\ e[1,2] + e[4,3] + w[1,3] = 1.50 \end{cases}$$

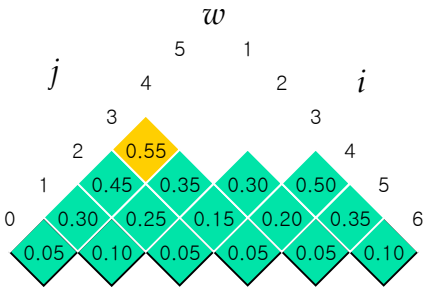
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10



Optimal Binary Search Trees - How Does the Algorithm Work?

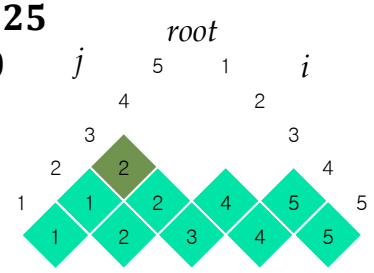
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

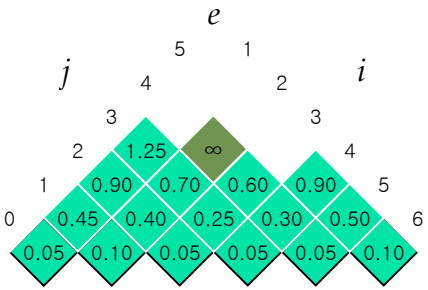
$$e[1,3] = min \begin{cases} e[1,0] + e[2,3] + w[1,3] = 1.30 \\ \boldsymbol{e[1,1] + e[3,3] + w[1,3] = 1.25} \\ e[1,2] + e[4,3] + w[1,3] = 1.50 \end{cases}$$

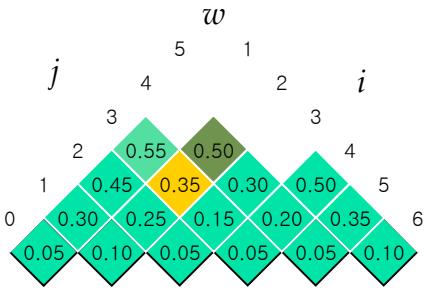


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

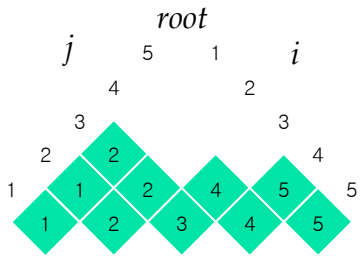
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

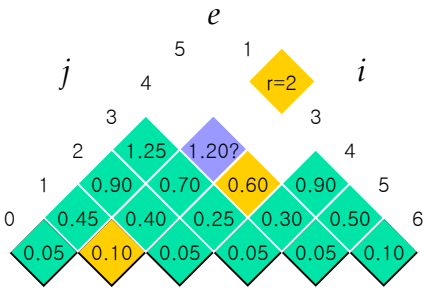
$$w[2,4] = w[2,3] + p_4 + q_4 = 0.50$$

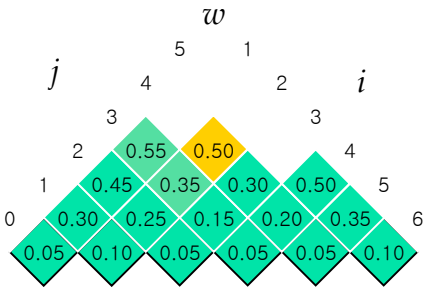


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

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

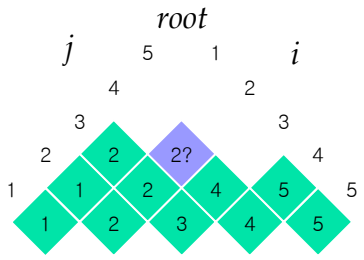




$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

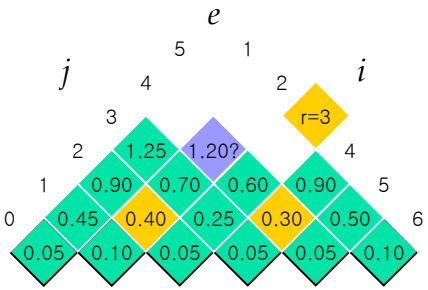
$$e[2,4] = min \begin{cases} e[2,1] + e[3,4] + w[2,4] = 1.20 \\ e[2,2] + e[4,4] + w[2,4] = ?? \\ e[2,3] + e[5,4] + w[2,4] = ?? \end{cases}$$

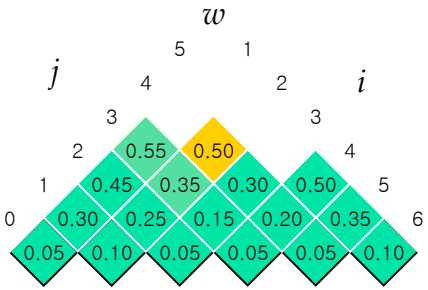
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10



Optimal Binary Search Trees - How Does the Algorithm Work?

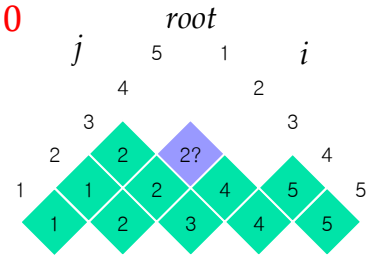
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

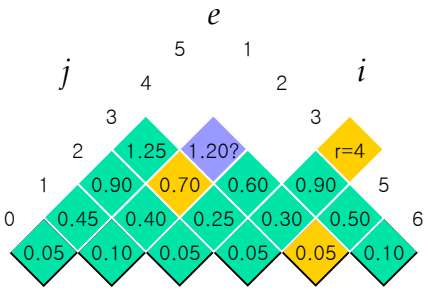
$$e[2,4] = min \begin{cases} e[2,1] + e[3,4] + w[2,4] = 1.20 \\ e[2,2] + e[4,4] + w[2,4] = 1.20 \\ e[2,3] + e[5,4] + w[2,4] = ?? \end{cases}$$

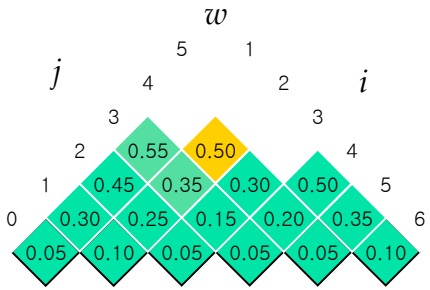


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

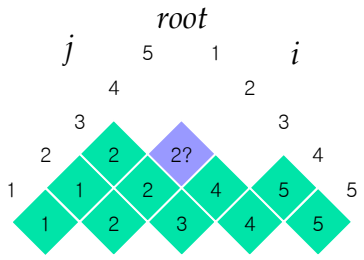
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

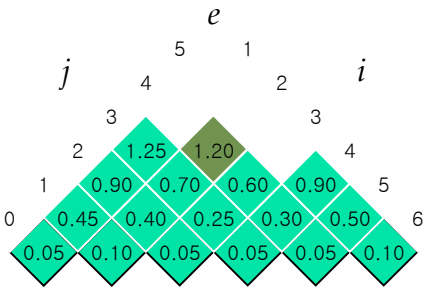
$$e[2,4] = min \begin{cases} e[2,1] + e[3,4] + w[2,4] = 1.20 \\ e[2,2] + e[4,4] + w[2,4] = 1.20 \\ e[2,3] + e[5,4] + w[2,4] = 1.25 \end{cases}$$

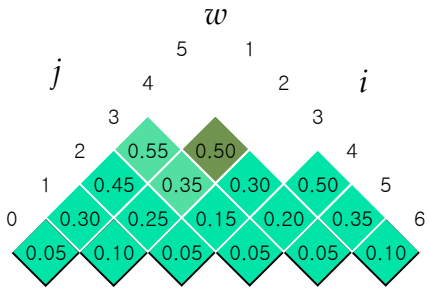


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

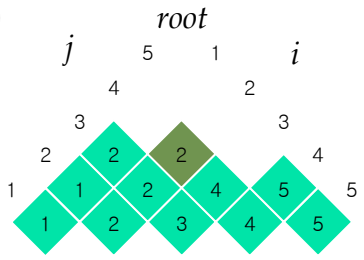
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

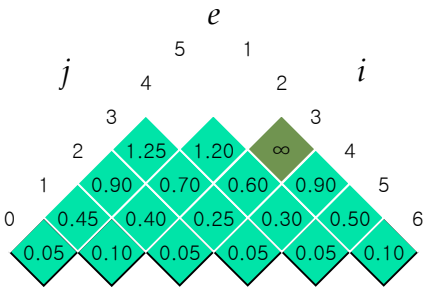
$$e[2,4] = min \begin{cases} \boldsymbol{e[2,1] + e[3,4] + w[2,4] = 1.20} \\ e[2,2] + e[4,4] + w[2,4] = 1.20 \\ e[2,3] + e[5,4] + w[2,4] = 1.25 \end{cases}$$

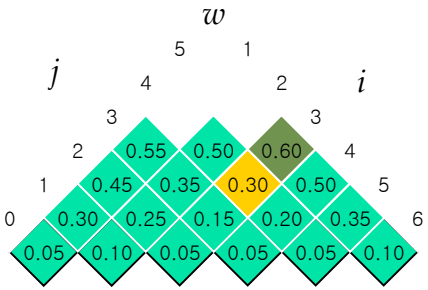


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

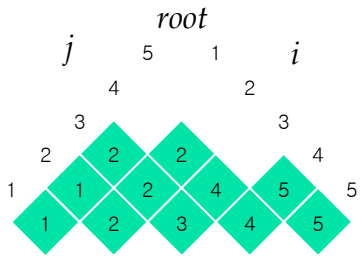
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

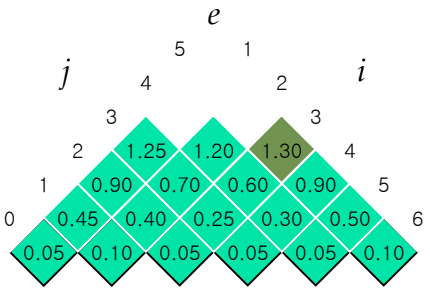
$$w[3,5] = w[3,4] + p_5 + q_5 = 0.45$$

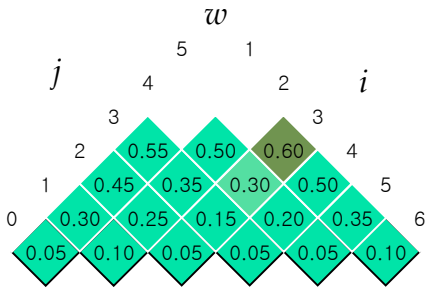


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

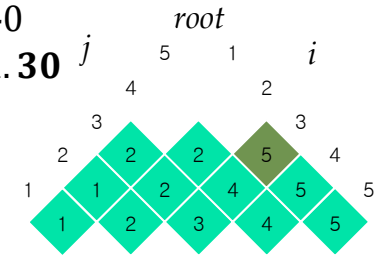
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

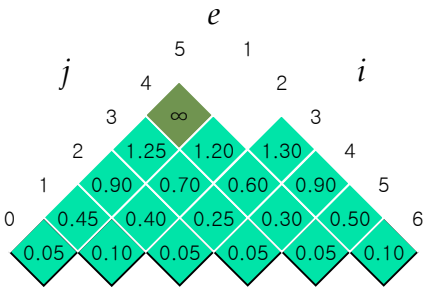
$$e[3,5] = min \begin{cases} e[3,2] + e[4,5] + w[3,5] = 1.60 \\ e[3,3] + e[5,5] + w[3,5] = 1.40 \\ \boldsymbol{e[3,4] + e[6,5] + w[3,5] = 1.30} \end{cases}$$

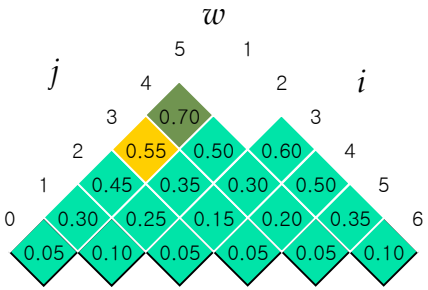


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

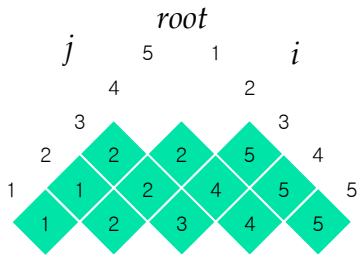
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

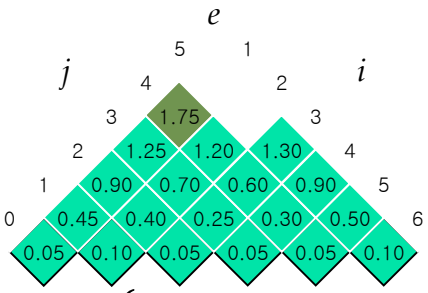
$$w[1,4] = w[1,3] + p_4 + q_4 = 0.70$$

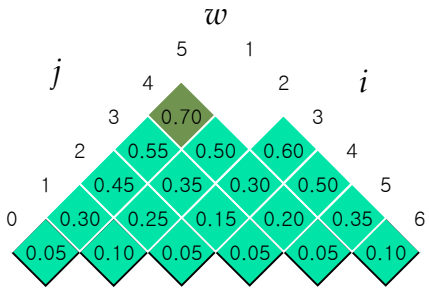


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

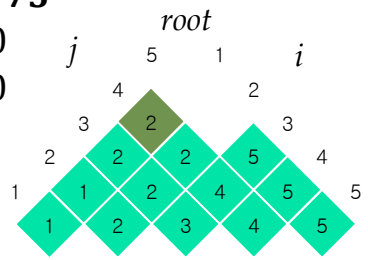
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

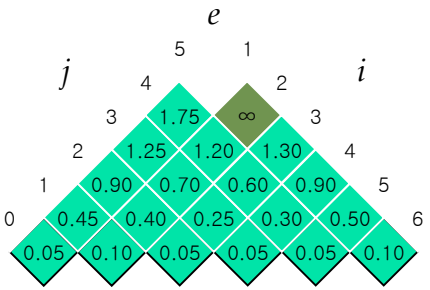
$$e[1,4] = min \begin{cases} e[1,0] + e[2,4] + w[1,4] = 1.95 \\ \boldsymbol{e[1,1] + e[3,4] + w[1,4] = 1.75} \\ e[1,2] + e[4,4] + w[1,4] = 1.90 \\ e[1,3] + e[5,4] + w[1,4] = 2.00 \end{cases}$$

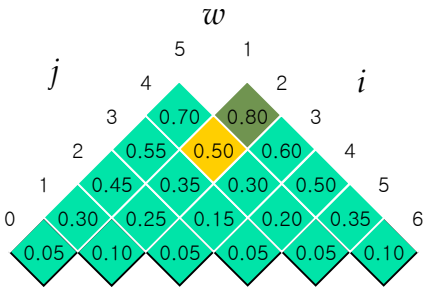


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

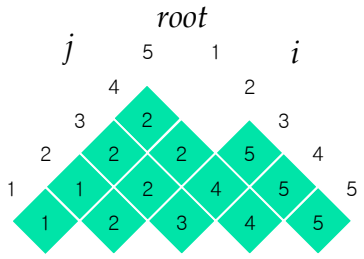
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

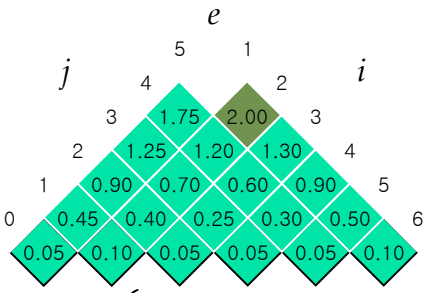
$$w[2,5] = w[2,4] + p_5 + q_5 = 0.80$$

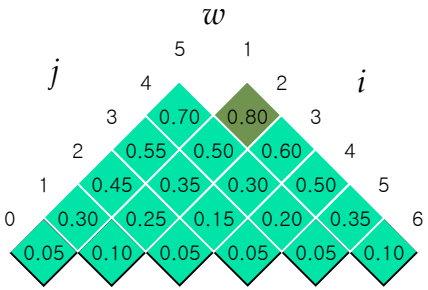


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

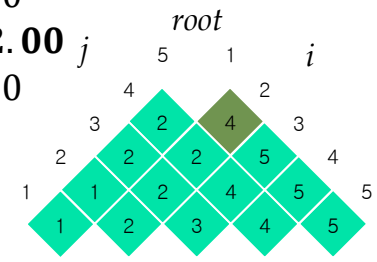
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

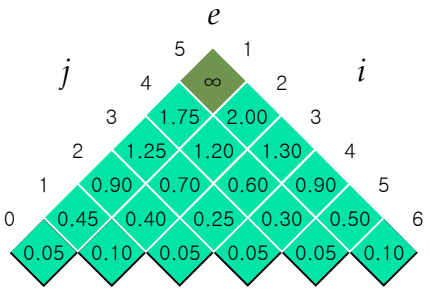
$$e[2,5] = min \begin{cases} e[2,1] + e[3,5] + w[2,5] = 2.20 \\ e[2,2] + e[4,5] + w[2,5] = 2.10 \\ \boldsymbol{e[2,3] + e[5,5] + w[2,5] = 2.00} \\ e[2,4] + e[6,5] + w[2,5] = 2.10 \end{cases}$$

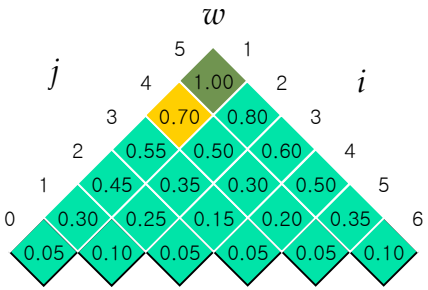


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

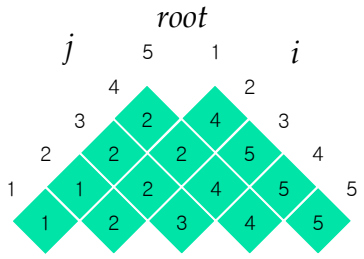
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

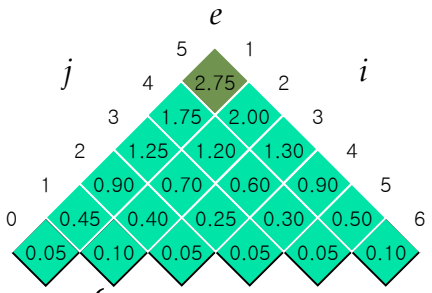
$$w[1,5] = w[1,4] + p_5 + q_5 = 1.00$$

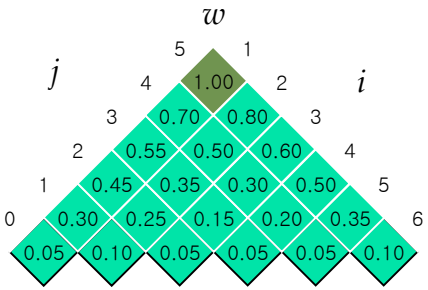


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - How Does the Algorithm Work?

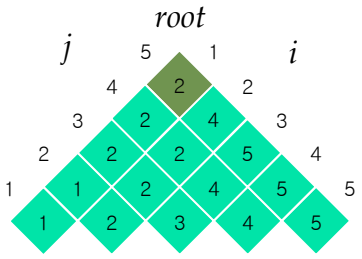
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$





$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

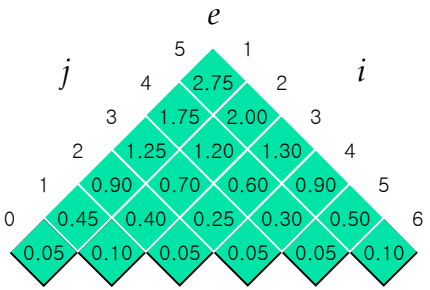
$$e[1,5] = min \begin{cases} e[1,0] + e[2,5] + w[1,5] = 3.05 \\ \boldsymbol{e[1,1] + e[3,5] + w[1,5] = 2.75} \\ e[1,2] + e[4,5] + w[1,5] = 2.80 \\ e[1,3] + e[5,5] + w[1,5] = 2.75 \\ e[1,4] + e[6,5] + w[1,5] = 2.85 \end{cases}$$

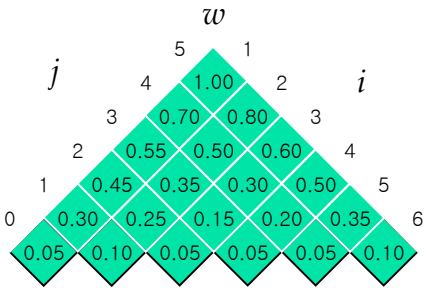


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

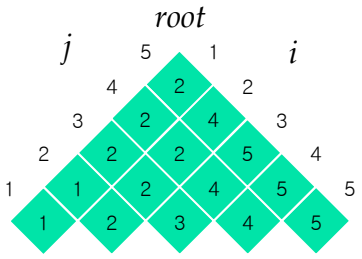
Optimal Binary Search Trees - Result

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



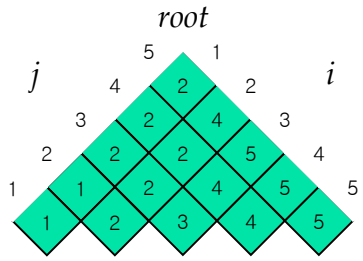


$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

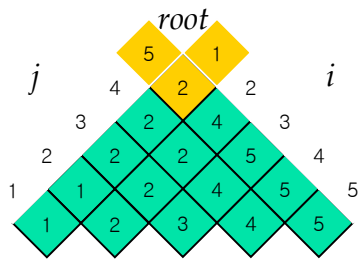


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees - Construction



Optimal Binary Search Trees - Construction

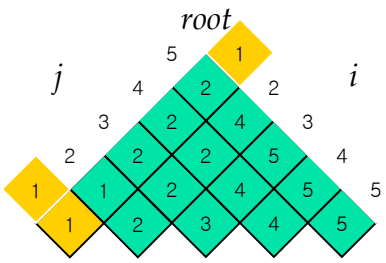


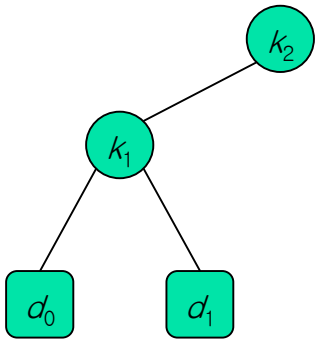
k_2

$root[1,5] = 2$

➡ split to [1,1] and [3,5]

Optimal Binary Search Trees - Construction

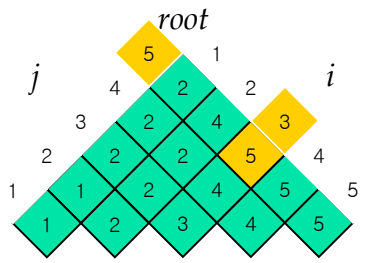


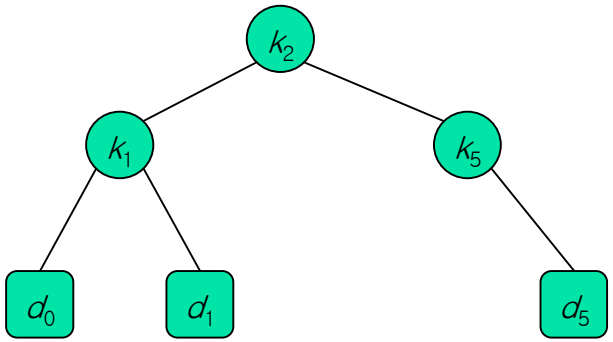


$$root[1,1] = 1$$

➡ split to [1,0] and [2,1] : dummies

Optimal Binary Search Trees - Construction

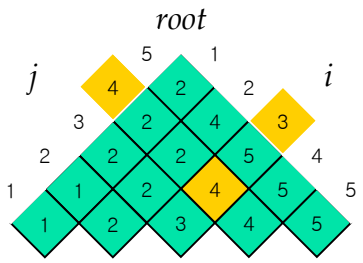


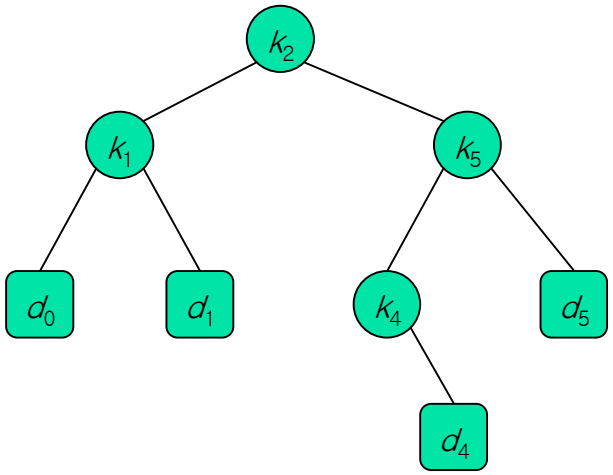


$root[3,5] = 5$

➡ split to [3,4] and [6,5]:
[6,5] is dummy

Optimal Binary Search Trees - Construction

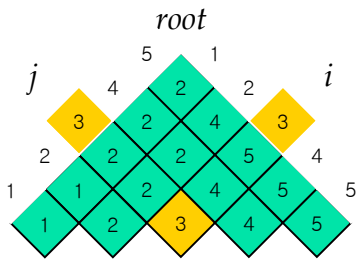




$$root[3,4] = 4$$

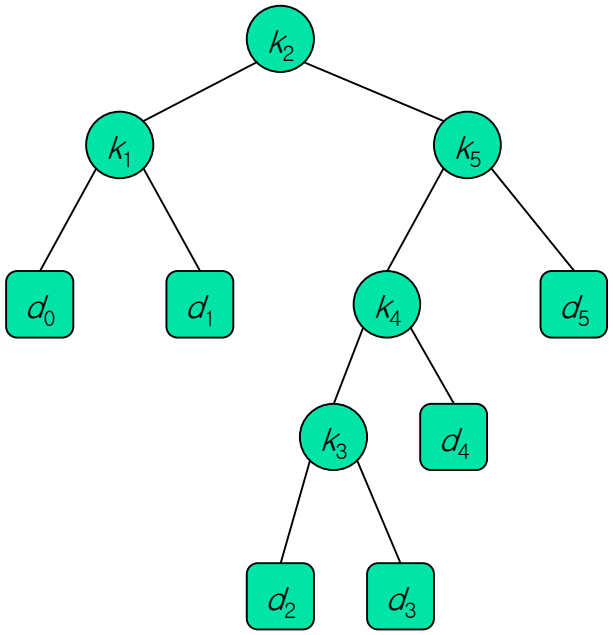
➡ split to [3,3] and [5,4]:
[5,4] is dummy

Optimal Binary Search Trees - Construction

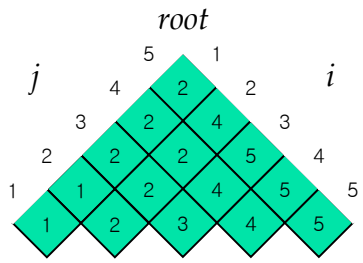


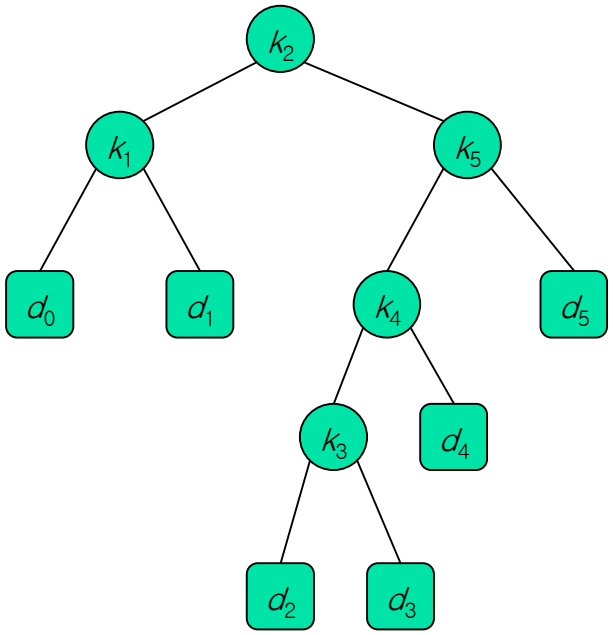
$root[3,3] = 3$

split to [3,2] and [4,3]: dummies



Optimal Binary Search Trees - Construction



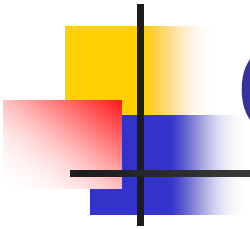


Any Question?

Introduction to Algorithms (Chapter 16: Greedy Algorithms)

Kyuseok Shim

Electrical and Computer Engineering
Seoul National University



Outline

- This chapter covers an important technique called greedy algorithms used in designing and analyzing efficient algorithms.

Greedy Algorithms

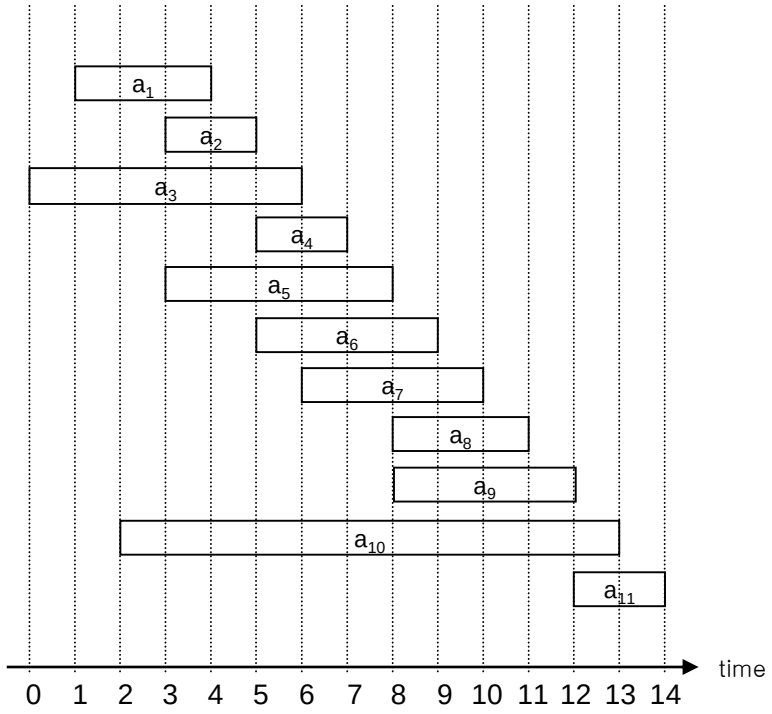
- For many optimization problems, using dynamic programming to determine the best choices is overkill and more efficient algorithms will do.
- A greedy algorithm always makes the choice that looks best at the moment.
- That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- This chapter explores optimization problems for which greedy algorithms provide optimal solutions.

An Activity Selection Problem

- The problem of scheduling several competing activities requiring exclusive use of a common resource to select a maximum-size set of mutually compatible activities.
- Select a maximum-size subset of mutually compatible activities.
  - An activity set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed activities that wish to use a resource
  - Each activity a_i has a start time s_i and a finish time f_i, where $0 \le s_i < f_i < \infty$
  - If selected, activity a_i takes place during the half-open time interval $[s_i, f_i)$.
  - Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap
  - We assume that the activities are sorted in monotonically increasing order of finish time: $f_1 \le f_2 \le \ldots \le f_n$

An Example of Activities

k	s_k	f_k
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



Dynamic Programming

- We consider several choices when determining which subproblems to use in an optimal solution.

Optimal Substructure

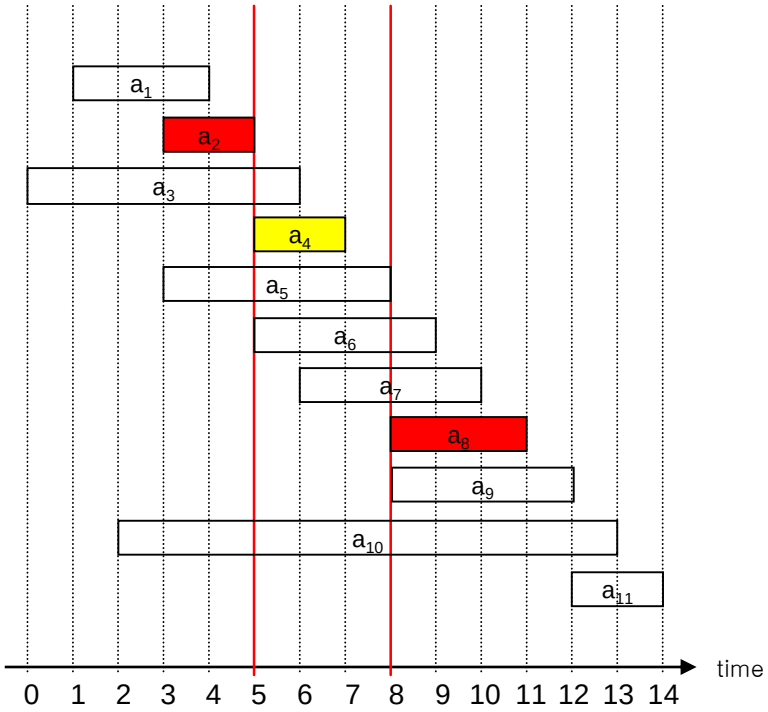
- Let us denote by S_{ij} the set of activities that start after activity a_i finishes and that finish before activity a_j starts.
- Suppose that we wish to find a maximum set of mutually compatible activities in S_{ij}
- Suppose further that such a maximum set is A_{ij}, which includes some activity a_k.
- By including a_k in an optimal solution, we are left with two subproblems:
  - finding mutually compatible activities in the set S_{ik}
  - finding mutually compatible activities in the set S_{kj}

Optimal Substructure

- Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$.
  - A_{ik} contains the activities in A_{ij} that finish before a_k starts.
  - A_{kj} contains the activities in A_{ij} that start after a_k finishes.
- We have $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.
  - The maximum-size set A_{ij} of mutually compatible activities in S_{ij} consists of $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ activities.
- The optimal solution A_{ij} must also include optimal solutions to the two subproblems for S_{ik} and S_{kj}.
  - If we find a set A'_{kj} of mutually compatible activities in S_{kj} where $|A'_{kj}| > |A_{kj}|$, then we could use A'_{kj} , rather than A_{kj} , in a solution to the subproblem for S_{ij}.
  - We would have constructed a set of $|A_{ik}| + |A'_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1 = |A_{ij}|$ mutually compatible activities, which contradicts the assumption that A_{ij} is an optimal solution.
  - A symmetric argument applies to the activities in S_{ik}.

Examples of S_{ij}

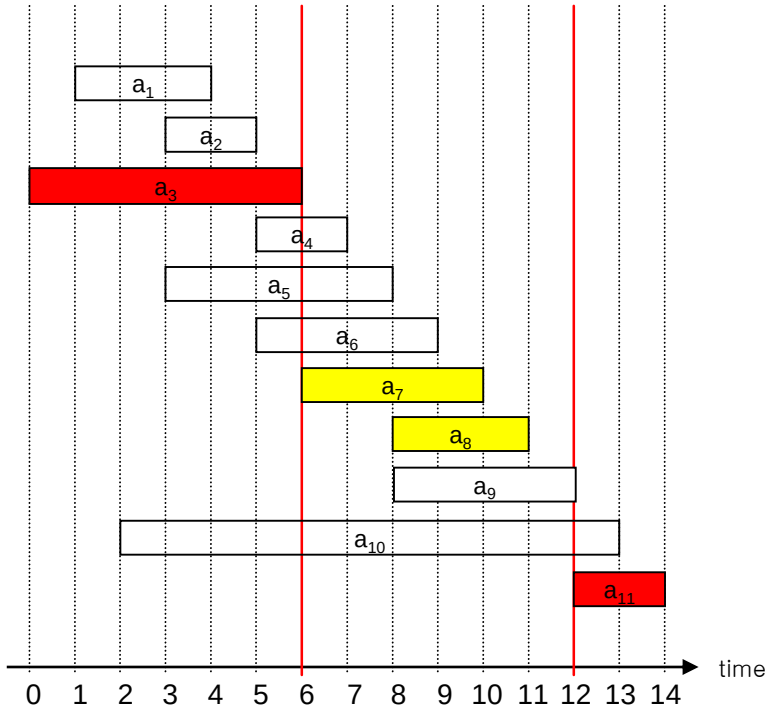
$S_{2,8}$



Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Examples of S_{ij}

$S_{3,11}$



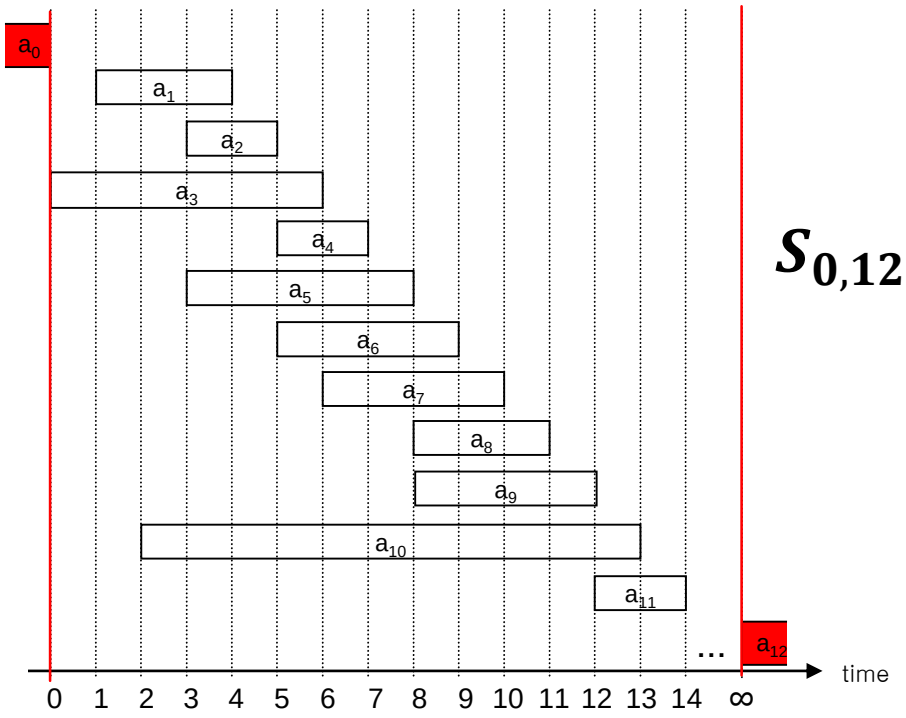
Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Dynamic Programming

- $S_{ij} = \{ a_k \in S : f_i \le s_k < f_k \le s_j \}$
- a_0: $f_0 = 0$, a_{n+1}: $s_{n+1} = \infty$, $S = S_{0,n+1}$
- Assume that the activities are sorted in increasing order of finish time
  - $f_0 \le f_1 \le f_2 \le \ldots \le f_n < f_{n+1}$
- $S_{ij} = \emptyset$, whenever $i \ge j$
- Let an optimal solution A_{ij} to S_{ij} and $a_k \in A_{ij}$
  - $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.
- $A_{0,n+1}$: an optimal solution to the entire problem

An Example of Activities

k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-



$S_{0,12}$

Dynamic Programming

- Recursive definition
  - c[i,j]: the number of activities in a maximum-size subset of mutually compatible activities in S_{ij}
  - C[i, j] = c[i, k] + c[k, j] +1

$$C[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \varnothing \\ \max_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \varnothing \end{cases}$$

Making the Greedy Choice

- What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems?
- That could save us from having to consider all the choices inherent in recurrence.
- In fact, for the activity selection problem, we need to consider only one choice: the greedy choice.
- Intuition suggests that we should choose an activity that leaves the resource available for as many other activities as possible.
- Choose the activity in S with the earliest finish time, since that would leave the resource available for as many of the activities that follow it as possible.

Making the Greedy Choice

- Since the activities are sorted in monotonically increasing order by finish time, the greedy choice is activity a_1.
- If we make the greedy choice, we have only one remaining subproblem to solve: finding activities that start after a_1 finishes.
- Why don't we have to consider activities that finish before a_1 starts?
- We have that $s_1 < f_1$, and f_1 is the earliest finish time of any activity, and therefore no activity can have a finish time less than or equal to s_1.
- Thus, all activities that are compatible with activity a_1 must start after a_1 finishes.

Making the Greedy Choice

- Let $S_k = \{a_i \in S : s_i \geq f_k\}$ be the set of activities that start after activity a_k finishes.
- If we make the greedy choice of activity a_1, then S_1 remains as the only subproblem to solve.
- Optimal substructure tells us that
  - If a_1 is in the optimal solution,
  - An optimal solution to the original problem consists of activity a_1 and all the activities in an optimal solution to the subproblem S_1.
- Proof
  - Assume that A_1 is an optimal solution of S_1 and there is an optimal solution O such that $|\{a_1\} \cup A_1| < |O|$ where O has a_1.
  - If we replace $(O - \{a_1\})$ by A_1, we have $|O| \leq |\{a_1\} \cup A_1|$.
  - Thus, $|\{a_1\} \cup A_1| < |\{a_1\} \cup A_1|$which contradicts.

Greedy Solution

- Theorem 16.1
  - Consider any nonempty subproblem S_k, and let a_m be the activity in S_k with the earliest finish time.
  - Then, activity a_m isincluded in some maximum-size subset of mutually compatible activities of S_k.
- Proof
  - Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the earliest finish time.
  - If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k.
  - If $a_j \neq a_m$, let the set $A'_k = A_k - \{ a_j \} \cup \{ a_m \}$.
  - The activities in A'_k are disjoint, because the activities in A_k are disjoint, a_j is the first activity in A_k to finish, and $f_m \leq f_j$.
  - Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m.

Greedy Solution

- Theorem 16.1
  - Consider any nonempty subproblem S_k, and let a_m be the activity in S_k with the earliest finish time.
  - Then, activity a_m isincluded in some maximum-size subset of mutually compatible activities of S_k.
- Although we might be able to solve the activity-selection problem with dynamic programming, we don't need to.
- Instead, we can repeatedly choose the activity that finishes first, keep only the activities compatible with this activity, and repeat until no activities remain.
- Because we always choose the activity with the earliest finish time, the finish times of the activities we choose must strictly increase.
- We can consider each activity just once overall, in monotonically increasing order of finish times.

Greedy Solution

- An algorithm to solve the activity-selection problem does not need to work bottom-up, like a table-based dynamic-programming algorithm.
- Instead, it can work top-down, choosing an activity to put into the optimal solution and then solving the subproblem of choosing activities from those that are compatible with those already chosen.
- Greedy algorithms typically have this top-down design.
  - Make a choice and then solve a subproblem, rather than the bottom-up technique of solving subproblems before making a choice.

Greedy Solution

- Recursive definition revisted
  - c[i,j]: the number of activities in a maximum-size subset of mutually compatible activities in S_{ij}
  - C[i, j] = c[i, k] + c[k, j] +1

$$C[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \varnothing \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \varnothing \end{cases}$$

A Recursive Greedy Algorithm

- The procedure RECURSIVE-ACTIVITY-SELECTOR takes
  - the start and finish times of the activities, represented as arrays s and f
  - the index k that defines the subproblem S_k it is to solve
  - the size n of the original problem
- We assume that the n input activities are already ordered by monotonically
- In order to start, we add the fictitious activity a_0 with $f_0 = 0$, so that subproblem S_0 is the entire set of activities S. increasing finish time.
- The initial call, which solves the entire problem, is RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, n).

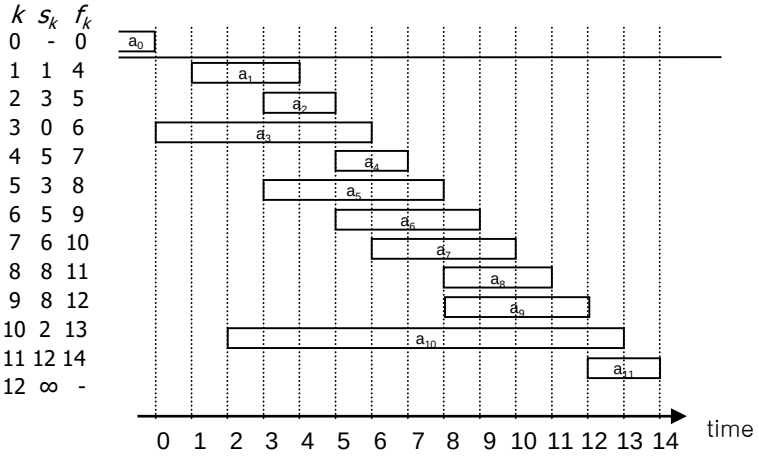
A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

The running is $\theta(n)$.

A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

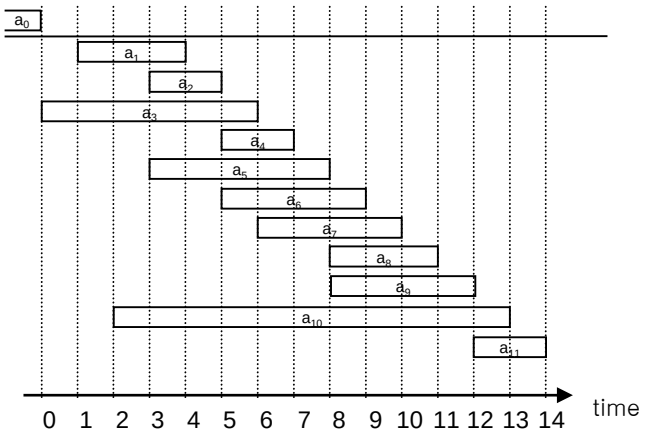


A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

n = 11
k = 0
m = 1

k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-

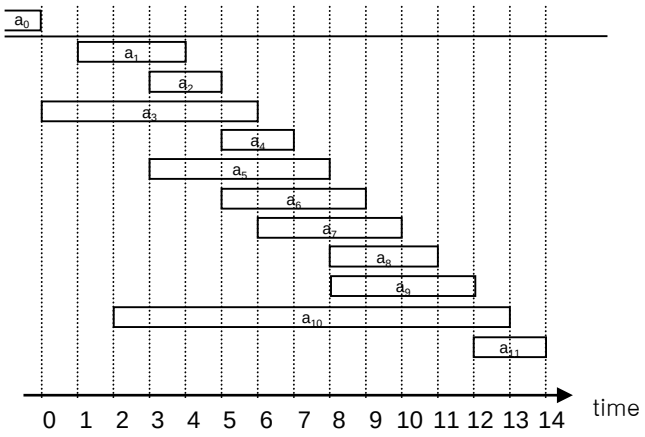


A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

n = 11
k = 0
m = 1

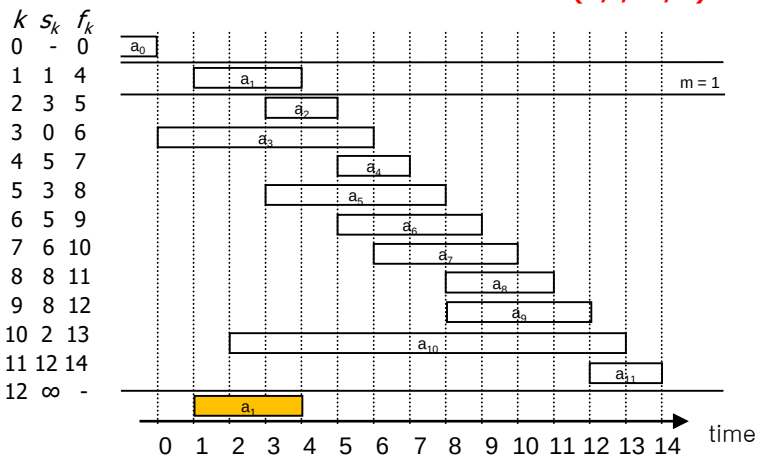
k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

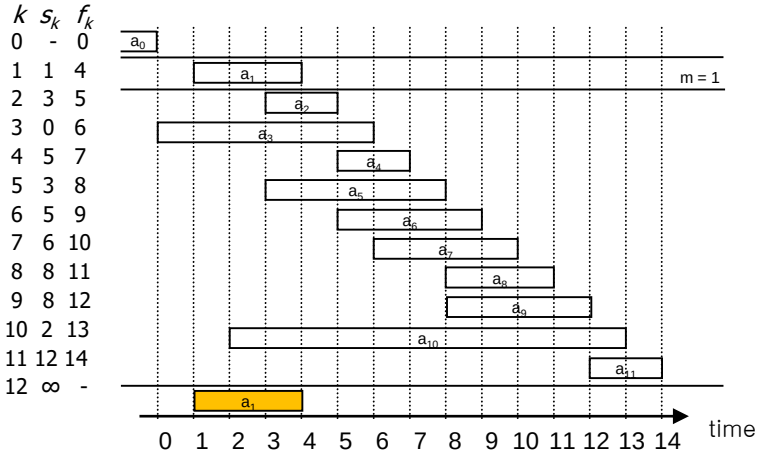
n = 11
k = 0
m = 1



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

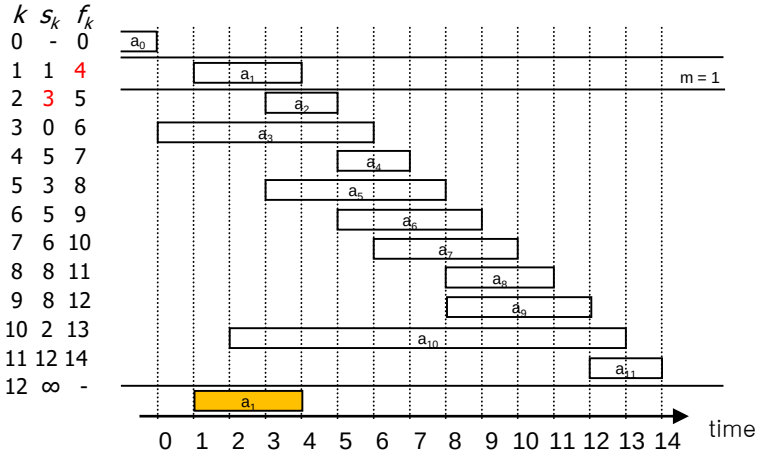
n = 11
k = 1
m = 2



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

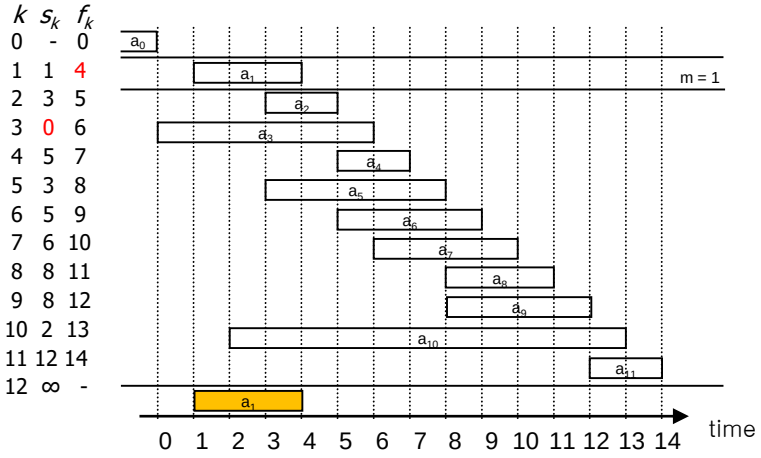
n = 11
k = 1
m = 2



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

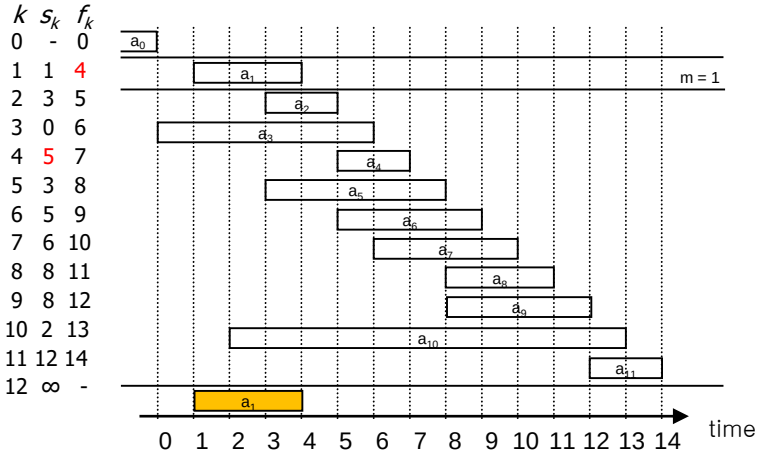
n = 11
k = 1
m = 3



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

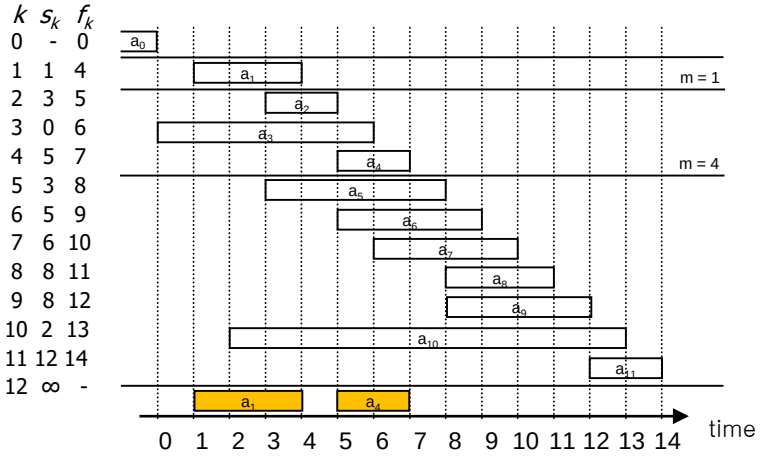
n = 11
k = 1
m = 4



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

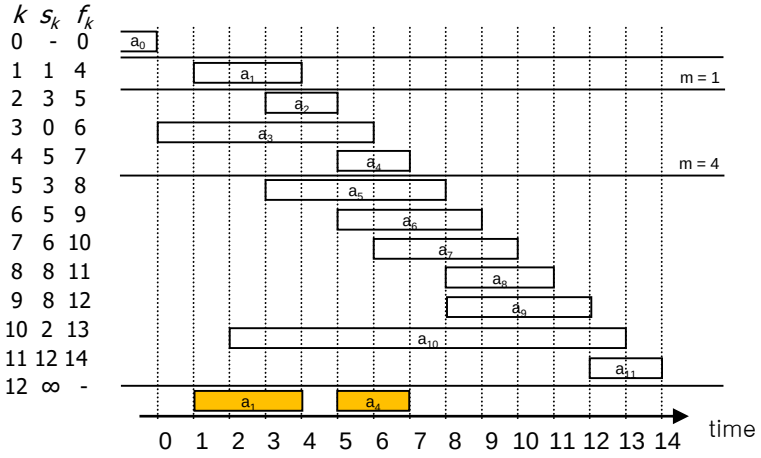
n = 11
k = 1
m = 4



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

n = 11
k = 4
m = 5

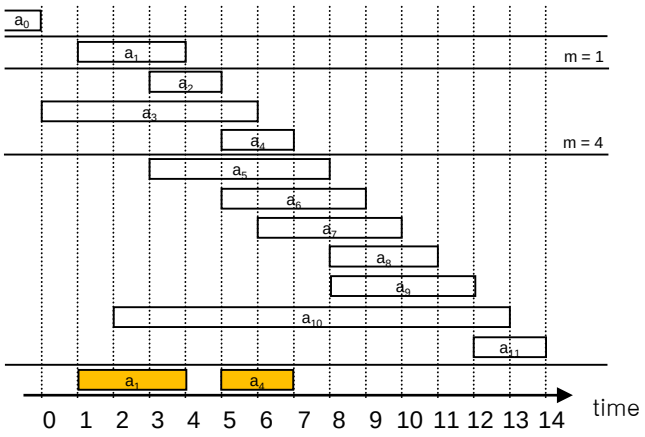


A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

n = 11
k = 4
m = 5

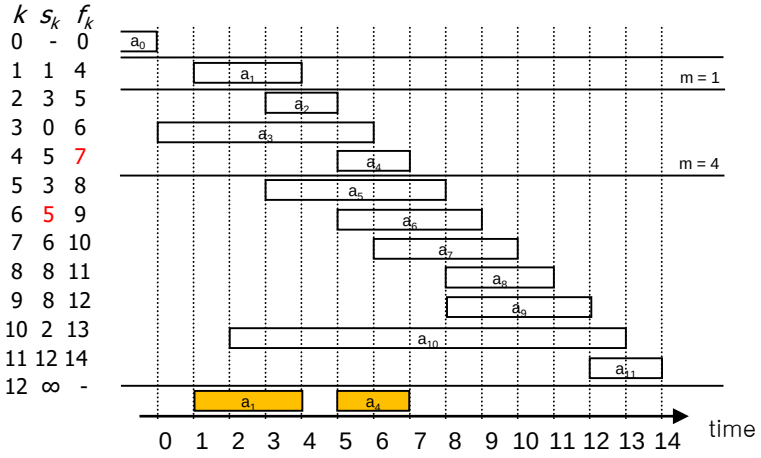
k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

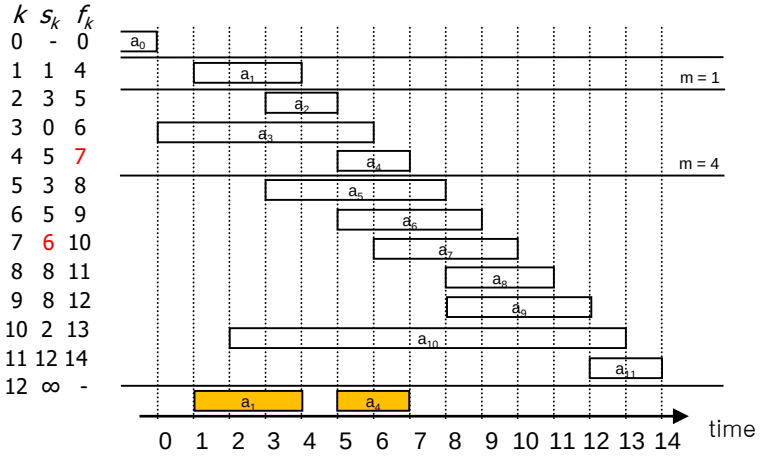
n = 11
k = 4
m = 6



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

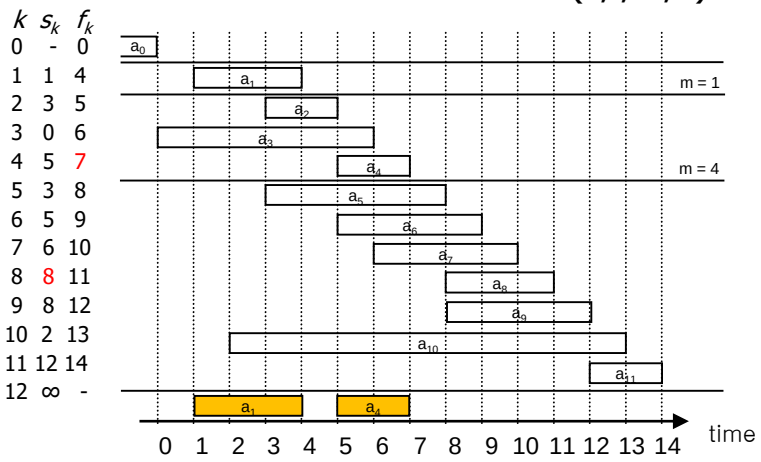
n = 11
k = 4
m = 7



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

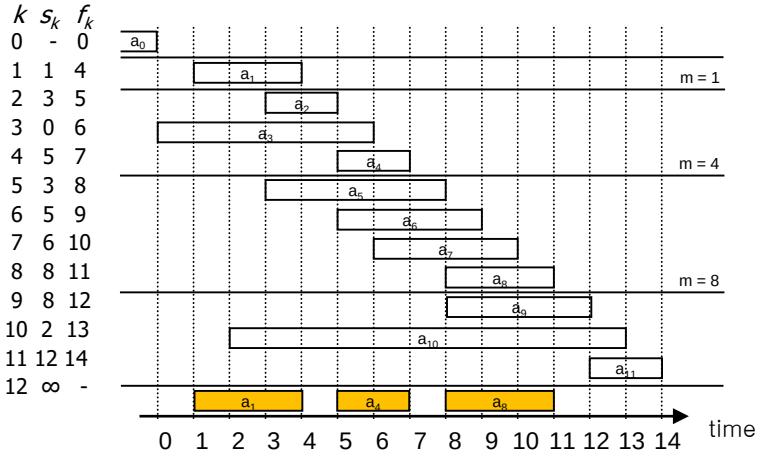
n = 11
k = 4
m = 8



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

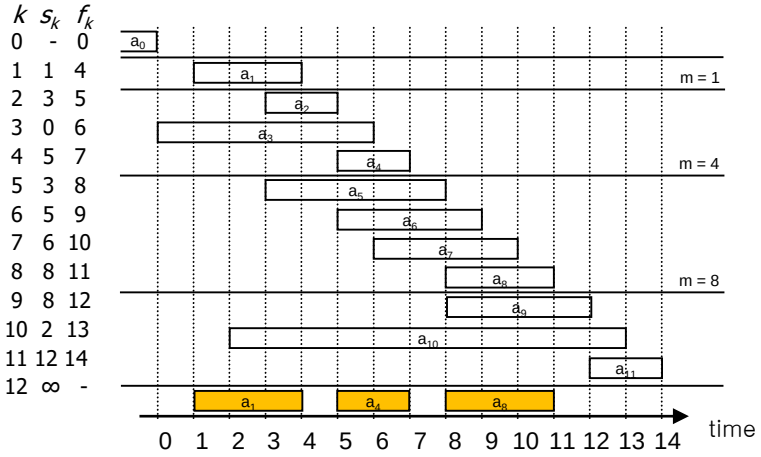
n = 11
k = 4
m = 8



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

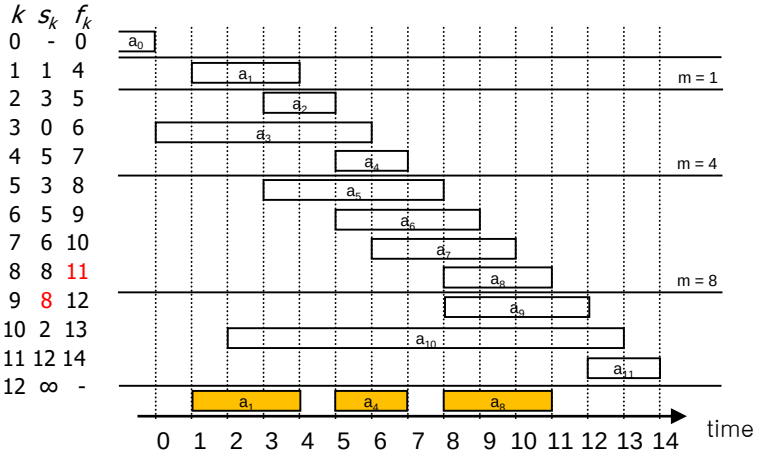
n = 11
k = 8
m = 9



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

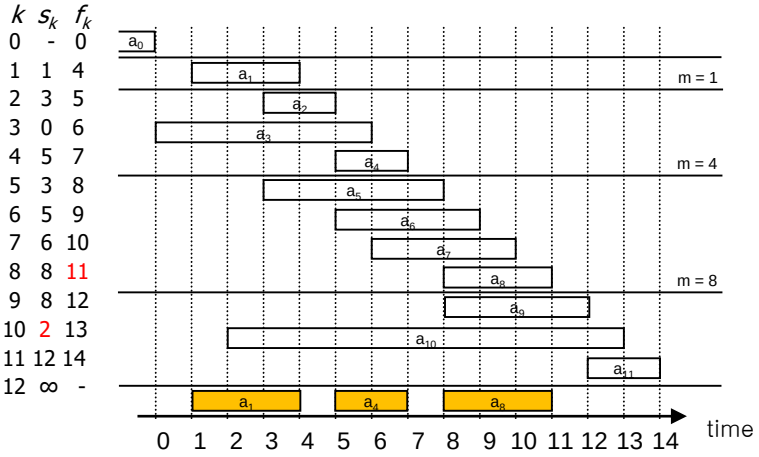
n = 11
k = 8
m = 9



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

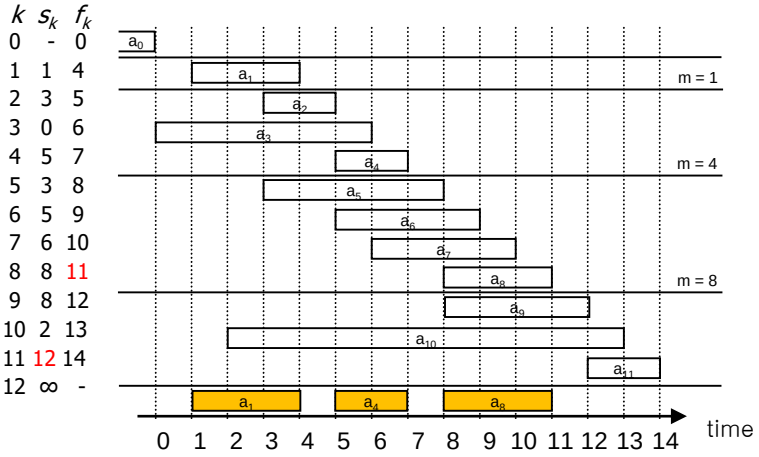
n = 11
k = 8
m = 10



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

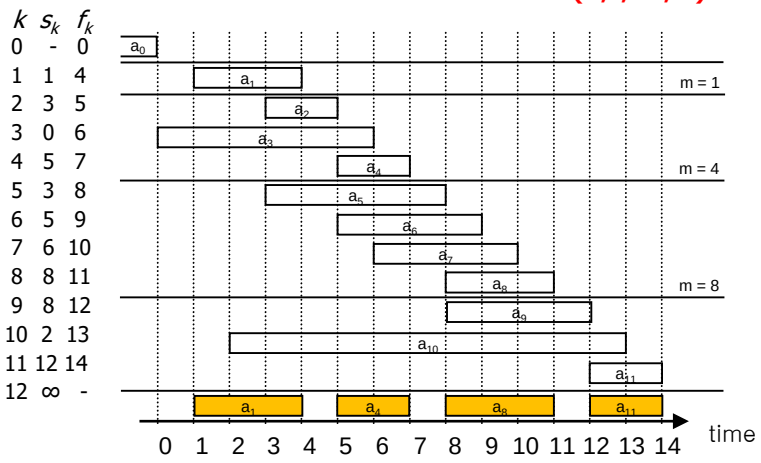
n = 11
k = 8
m = 11



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

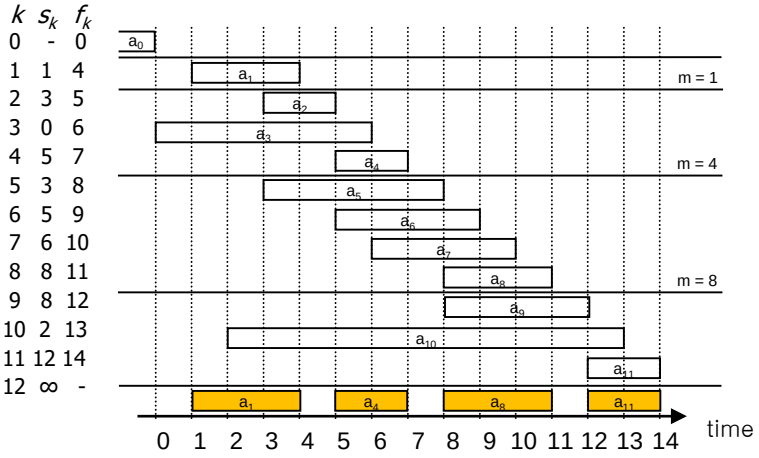
n = 11
k = 8
m = 11



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1    m = k+1
2    while m ≤ n and s[m] < f[k]
3        m = m+1
4    if m ≤ n
5        return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6    else
7        return ∅
```

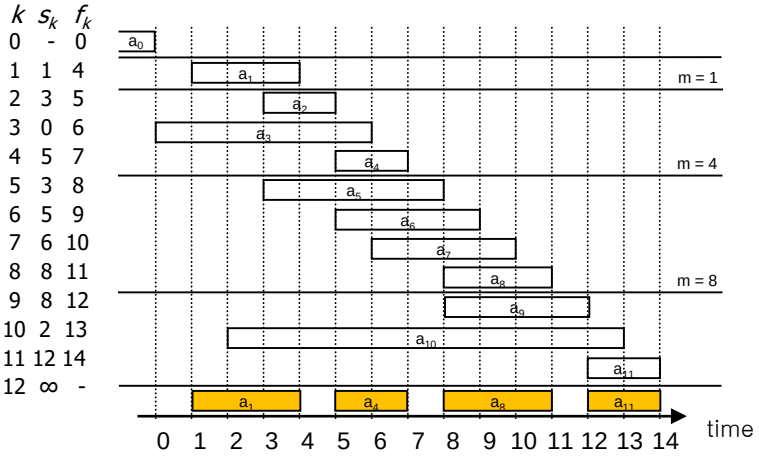
n = 11
k = 11
m = 12



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

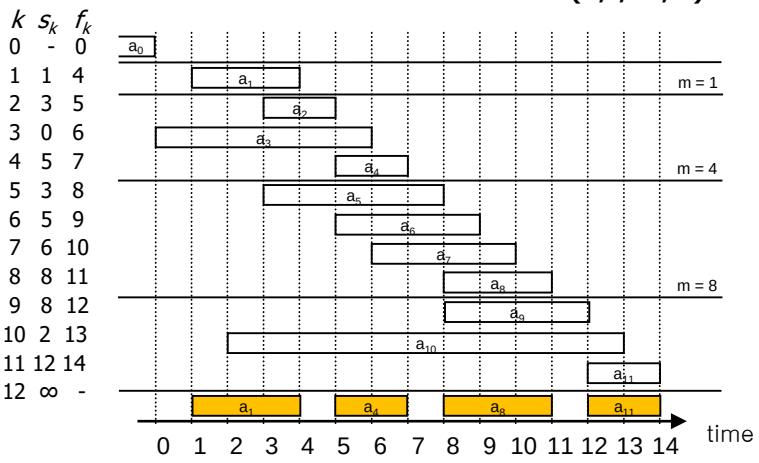
n = 11
k = 11
m = 12



A Recursive Greedy Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

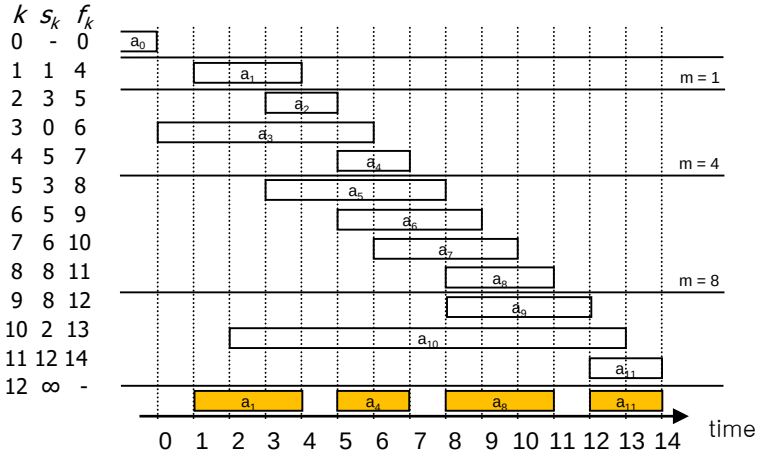
n = 11
k = 11
m = 12



A Recursive Greedy Algorithm

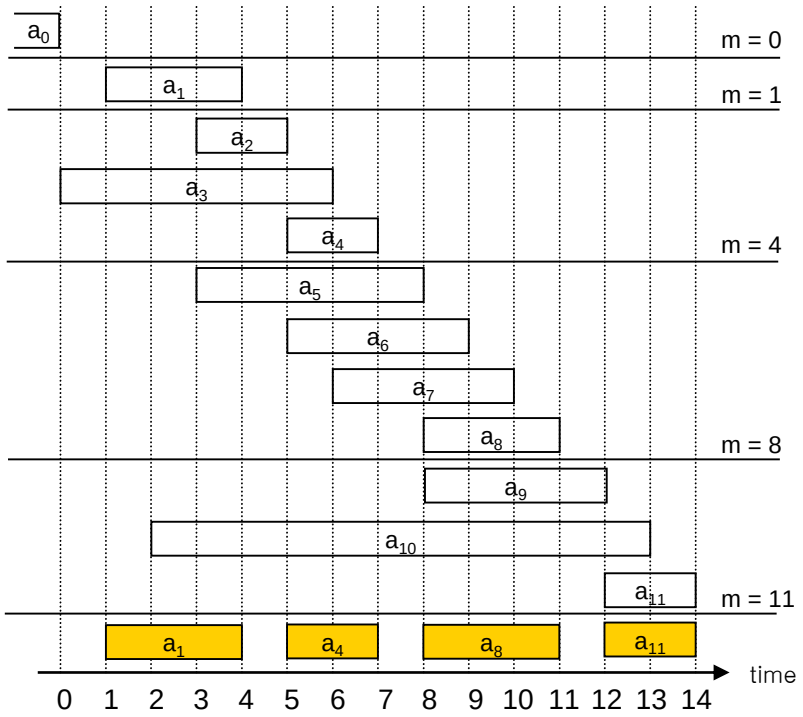
```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1     m = k+1
2     while m ≤ n and s[m] < f[k]
3         m = m+1
4     if m ≤ n
5         return {a_m} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
6     else
7         return ∅
```

n = 11
k = 11
m = 12



Operation of RECURSIVE-ACTIVITY-SELECTOR

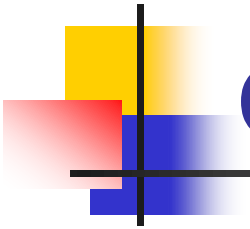
k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-



An Iterative Greedy Algorithm

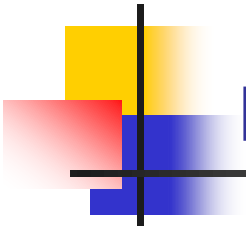
```
GREEDY-ACTIVITY-SELECTOR(s, f)
1    n = s.length
2    A = {a_1}
3    k = 1
4    for m = 2 to n
5        if s[m] ≥ f[k]
6            A = A U {a_m}
7            k = m
8    return A
```

Elements of Greedy Strategy



Greedy Algorithms

- A greedy algorithm obtains an optimal solution to a problem by making a sequence of choices.
- At each decision point, the algorithm makes choice that seems the best at the moment.
- This heuristic strategy does not always produce an optimal solution.
- We next discusses some of the general properties of greedy methods.



Previous Process to Develop a Greedy Algorithm

- Determine the optimal substructure of the problem.
- Develop a recursive solution.
- Show that if we make the greedy choice, then only one subproblem remains.
- Prove that it is always safe to make the greedy choice.
- Develop a recursive algorithm that implements the greedy strategy.
- Convert the recursive algorithm to an iterative algorithm.

Previous Process to Develop a Greedy Algorithm

- Dynamic-programming underpins a greedy algorithm.
- In the activity-selection problem, we first defined the subproblems S_{ij}, where both i and j varied.
- We then found that if we always made the greedy choice, we could restrict the subproblems to be of the form S_k.
- Alternatively, we could have fashioned our optimal substructure with a greedy choice in mind, so that the choice leaves just one subproblem to solve.
- We could have started by dropping the second subscript and defining subproblems of the form S_k.
- Then, we could have proven that a greedy choice (the first activity a_m to finish in S_k), combined with an optimal solution to the remaining set S_m of compatible activities, yields an optimal solution to S_k.

Steps of Designing Greedy Algorithms

- Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
- Demonstrate optimal substructure by showing that
  - Having made the greedy choice, show that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

Steps of Designing Greedy Algorithms

- How can one tell if a greedy algorithm will solve a particular optimization problem?
  - There is no way in general.
  - But the greedy-choice property and optimal substructure are the two key ingredients.
  - If we can demonstrate that the problem has these properties, then we are well on the way to developing a greedy algorithm.

Greedy-choice property

- A globally optimal solution can be arrived at by making a locally optimal greedy choice.
- Make the choice that looks best in the current problem, without considering results from subproblems.

Difference from Dynamic Programming

- A Dynamic Programming Algorithm
  - The choice at each step usually depends on the solutions to subproblems.
  - Consequently, typically solves dynamic programming problems in a bottom-up manner.
  - Progresses from smaller subproblems to larger subproblems.
- Alternatively, we can solve them top down, but memoizing.
- Of course, even though the code works top down, we still must solve the subproblems before making a choice.

Difference from Dynamic Programming

- A Greedy Algorithm
  - Makes whatever choice seems the best at the moment.
  - Solves the subproblem arising after the choice is made.
  - The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to subproblems.
  - Unlike dynamic programming, which solves the subproblems before making the first choice, a greedy algorithm makes its first choice before solving any subproblems.
  - A dynamic programming algorithm proceeds bottom up, whereas a greedy strategy usually progresses in a top-down fashion.

Difference from Dynamic Programming

- We can usually make the greedy choice more efficiently than when we have to consider a wider set of choices.
- In the activity-selection problem, assuming that we had already sorted the activities in monotonically increasing order of finish times, we needed to examine each activity just once.
- By preprocessing the input or by using an appropriate data structure, we often can make greedy choices quickly, thus yielding an efficient algorithm.

Optimal Substructure

- A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.
- A key ingredient of assessing the applicability of dynamic programming as well as greedy algorithms.
- e.g.) If an optimal solution to subproblem S_{ij} has an activity a_k, then it must also contain optimal solutions to the subproblems S_{ik} and S_{kj}.
- We usually use a more direct approach regarding optimal substructure when applying it to greedy algorithms.
- We have the luxury of assuming that we arrived at a subproblem by having made the greedy choice in the original problem.
- All we really need to do is argue that an optimal solution to the subproblem, combined with the greedy choice already made, yields an optimal solution to the original problem.

Greedy vs. Dynamic Programming

- The optimal substructure property is exploited by both the greedy and dynamic programming.
- Because of this, there might be a mistake to decide which approach is proper for a given problem.