Homework 7

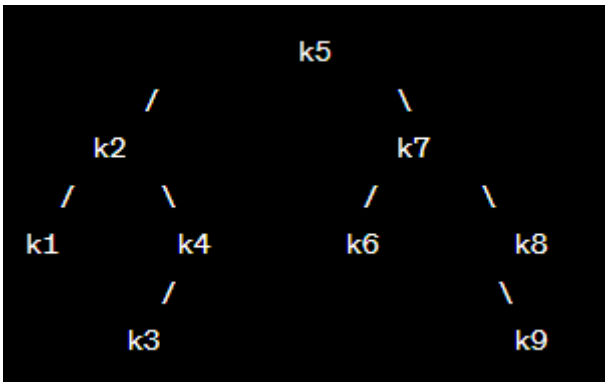
0811324 胡政煒

Case 1:

i	0	1	2	3	4	5	6	7	8	9
p_i		0.05	0.04	0.02	0.07	0.08	0.09	0.04	0.08	0.03
q_i	0.08	0.06	0.04	0.04	0.03	0.06	0.07	0.06	0.04	0.02

```
Enter the number of nodes: 9
Enter the probabilities for keys (p0 to pn): 0 0.05 0.04 0.02 0.07 0.08 0.09 0.04 0.08 0.03
Enter the probabilities for dummy keys (q0 to qn): 0.08 0.06 0.04 0.04 0.03 0.06 0.07 0.06 0.04 0.02
Smallest Search Cost: 3.45
Root: 5
Optimal BST - structure:
k5 is the root
k2 is the left child of k5
k1 is the left child of k2
k4 is the right child of k2
k3 is the left child of k4
k7 is the right child of k5
k6 is the left child of k7
k8 is the right child of k7
k9 is the right child of k8
```

Optimal Binary Search Tree:

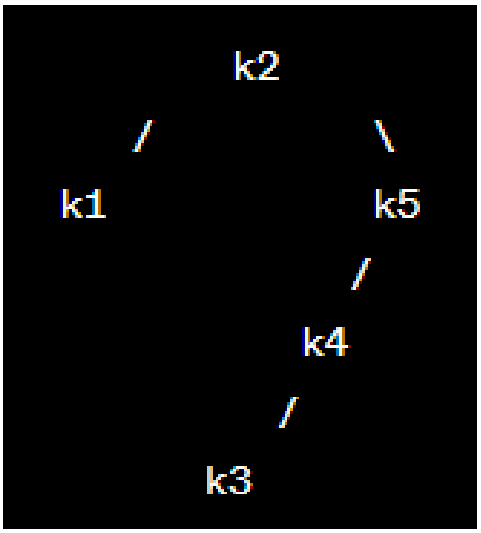


Case 2:

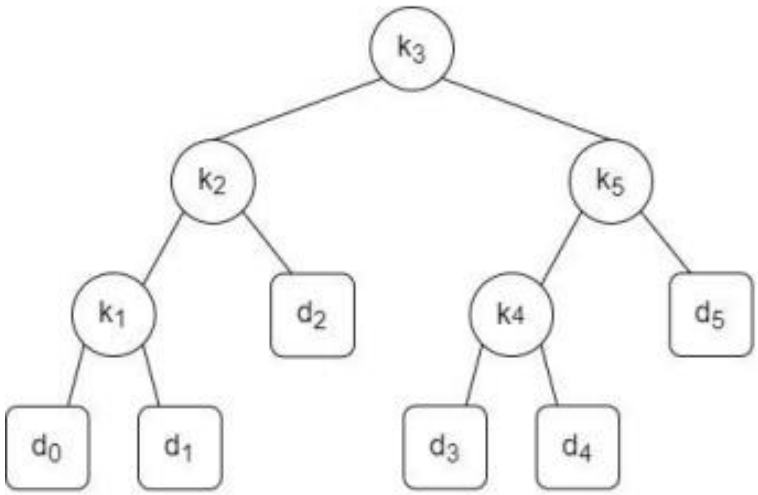
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

```
Enter the number of nodes: 5
Enter the probabilities for keys (p0 to pn): 0 0.15 0.10 0.05 0.10 0.20
Enter the probabilities for dummy keys (q0 to qn): 0.05 0.10 0.05 0.05 0.05 0.10
Smallest Search Cost: 2.75
Root: 2
Optimal BST - structure:
k2 is the root
k1 is the left child of k2
k5 is the right child of k2
k4 is the left child of k5
k3 is the left child of k4
```

Optimal Binary Search Tree:



討論:



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

$$\begin{aligned} \mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i \,, \end{aligned} \qquad (15.11)$$

根據教授上課所說，search cost 的期望值，可通過將所有 node 的 depth 與機率的乘積相加而得，因此可以透過上述公式獲得以下:

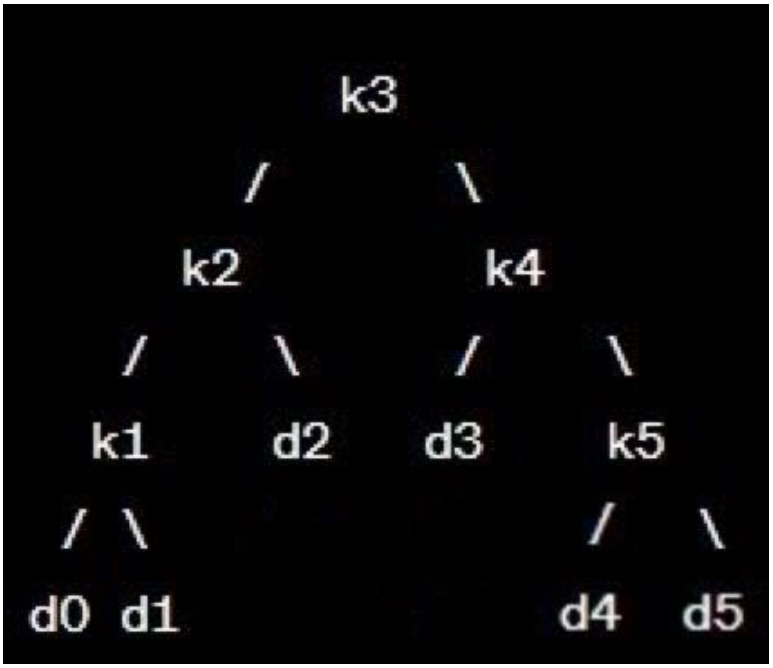
E=1*0.15 + 2 *（0.15+0.10）+ 3 *（0.05+0.05+0.10+0.10）+
4*(0.05+0.10+0.10+0.05)=2.75

透過程式碼也可以獲得相同的結果:

```
Enter the number of nodes: 5
Enter the probabilities for keys (p0 to pn): 0 0.05 0.15 0.15 0.10 0.10
Enter the probabilities for dummy keys (q0 to qn): 0.05 0.10 0.05 0.10 0.05 0.10
Smallest Search Cost: 2.75
Root: 3
Optimal BST - structure:
k3 is the root
k2 is the left child of k3
k1 is the left child of k2
k4 is the right child of k3
k5 is the right child of k4
```

因此可以得知此 Binary Search Tree 不為 Optimal Binary Search Tree，其正確樹狀圖如下:

Optimal Binary Search Tree:



Optimal Binary Search Tree 是指在給定節點頻率或概率的情況下，以最小化期望搜索成本為目標的二元搜尋樹，並且利用以下兩點:

動態規劃：Optimal Binary Search Tree 的求解可以使用動態規劃算法。通過計算子問題的最優解並遞推求解整體問題的最優解，可以找到期望搜索成本最小的樹。

結構和成本：在給定節點頻率或概率的情況下，可能存在多個最佳二元搜尋樹解。這些樹的結構可能不同，但它們的期望搜索成本是相同的，都是最小化的。

程式碼範例:

主程式:

```cpp
#include <iostream>
#include <vector>
#include <limits>

using namespace std;

void PrintOptimal_BST(const vector<vector<int>>& root, int i, int j, int level) ;
void Optimal_BST(const vector<double>& probabilities, const vector<double>& qValues, int n) ;

int main() {
    int node;
    cout << "Enter the number of nodes: ";
    cin >> node;

    vector<double> probabilities(node + 1), qValues(node + 1);

    cout << "Enter the probabilities for keys (p0 to pn): ";
    for (int i = 0; i <= node; i++)
    {
        cin >> probabilities[i];
    }

    cout << "Enter the probabilities for dummy keys (q0 to qn): ";
    for (int i = 0; i <= node; i++)
    {
        cin >> qValues[i];
    }
    // vector<double> p = {0.0, 0.15, 0.10, 0.05, 0.10, 0.20};
    // vector<double> q = {0.05, 0.10, 0.05, 0.05, 0.05, 0.10};
    // vector<double> p = {0.0, 0.05, 0.15, 0.15, 0.10, 0.10};
    // vector<double> q = {0.05, 0.10, 0.05, 0.10, 0.05, 0.10};
    // int n = 5;
    // 計算並輸出最佳二元搜尋樹
    Optimal_BST(probabilities, qValues, node);

    return 0;
}
```

印出 Optimal Binary Search Tree structure:

```
void PrintOptimal_BST(const vector<vector<int>>& root, int i, int j, int level)
{
    // 遞迴函數，用於印出最佳二元搜尋樹的結構
    // root：存儲最佳樹的根節點索引的二維向量
    // i：子樹的起始索引
    // j：子樹的結束索引
    // level：父節點的索引

    if (i > j)
    {
        // 子樹為空，返回
        return;
    }
    if (i == j)
    {
        // 子樹只有一個節點
        if (level == 0)
        {
            cout << "k" << i << " is the root" << endl;
        }
        else if (j < level)
        {
            cout << "k" << i << " is the left child of k" << level << endl;
        }
        else
        {
            cout << "k" << i << " is the right child of k" << level << endl;
        }
        return;
    }

    int r = root[i][j];
    // 獲取子樹的根節點索引
    if (level == 0)
    {
        cout << "k" << r << " is the root" << endl;
    }
    else if (r < level)
    {
        cout << "k" << r << " is the left child of k" << level << endl;
    }
    else
    {
        cout << "k" << r << " is the right child of k" << level << endl;
    }
    // 印出節點關係
    PrintOptimal_BST(root, i, r - 1, r);
    PrintOptimal_BST(root, r + 1, j, r);
    // 遞迴處理左子樹和右子樹
}
```

Optimal_BST 主程式碼:

```
void Optimal_BST(const vector<double>& probabilities, const vector<double>& qValues, int n)
{
    // 計算最佳二元搜尋樹的函數
    // probabilities：鍵的機率向量
    // qValues：虛擬鍵的機率向量
    // n：節點數量

    vector<vector<double>> cost(n + 2, vector<double>(n + 1, 0.0));
    vector<vector<double>> e(n + 2, vector<double>(n + 1, 0.0));
    vector<vector<int>> root(n + 1, vector<int>(n + 1, 0));

    // 初始化
    for (int i = 1; i <= n + 1; i++)
    {
        e[i][i - 1] = qValues[i - 1];
        cost[i][i - 1] = qValues[i - 1];
    }

    // 動態規劃求解
    for (int length = 1; length <= n; length++)
    {
        for (int i = 1; i <= n - length + 1; i++)
        {
            int j = i + length - 1;
            e[i][j] = numeric_limits<double>::max();
            cost[i][j] = cost[i][j - 1] + probabilities[j] + qValues[j];
            for (int rootIndex = i; rootIndex <= j; rootIndex++)
            {
                double temp = e[i][rootIndex - 1] + e[rootIndex + 1][j] + cost[i][j];
                if (temp < e[i][j])
                {
                    e[i][j] = temp;
                    root[i][j] = rootIndex;
                }
            }
        }
    }

    // 輸出結果
    cout << "Smallest Search Cost: " << e[1][n] << endl;
    cout << "Root: " << root[1][n] << endl;
    cout << "Optimal BST _ structure: " << endl;
    PrintOptimal_BST(root, 1, n, 0);
    // 印出最佳二元搜尋樹的結果
}
```