Introduction to Algorithms HW6

109511207 蔡宗儒

Pseudo Code

OPTIMAL-BST(p,q,n)

1. let e[1…n+1,0…n], w[1…n+1,0…n], and root[1…n,1…n] be new tables
2. for i = 1 to n+1
3. e[i,i-1] = qi-1
4. w[i,i-1] = qi-1
5. for l = 1 to n
6. for i = 1 to n-l+1
7. j = i+l-1
8. e[i,j] = ∞
9. w[i,j] = w[i,j-1] + pj + qj
10. for t = i to j
11. t = e[i,r-1] + e[r+1,j] + w[i,j]
12. if t < e[i,j]
13. e[i,j] = t
14. root[i,j] = r
15. return e and root

Introduction of Code

```
void Optimal_BST(vector<vector<double>> &e,vector<vector<double>> &w,vector<vector<int>> &r)
{
    for(int i=1;i<=p.size();i++)
    {
        e[i][i-1] = q[i-1];
        w[i][i-1] = q[i-1];
    }
    for(int l=1;l<=p.size()-1;l++)
    {
        for(int i=1;i<=p.size()-l;i++)
        {
            int j=i+l-1;
            e[i][j] = INT_MAX;
            w[i][j] = w[i][j-1] + p[j] + q[j];
            for(int k=i;k<=j;k++)
            {
                double t = e[i][k-1] + e[k+1][j] + w[i][j];
                if(t < e[i][j])
                {
                    e[i][j] = t;
                    r[i][j] = k;
                }
            }
        }
    }
}
```

Optimal_BST，根據課本 pseudo code 實踐計算最小 search cost

```
void printOBST(vector<vector<int>> &r)
{
    int root = r[1][p.size()-1];
    int depth = Depth(r,root,1,p.size()-1) + 1;
    vector<vector<string>> obst(depth+1,vector<string> (2*p.size(),"  "));
    fillOBST(r,obst,root,1,p.size()-1,1);
    for(int i=0;i<=depth;i++)
    {
        for(int j=1;j<=2*p.size()-1;j++)
            cout << obst[i][j]<<" ";
        cout << "\n\n";
    }
}
```

printOBST，利用自己寫的 Depth 計算 OBST 的高度，由此高度宣告出需要的 size 的二維 string vector，最後利用自己寫的 fillOBST 來把要印的 OBST 存入二維 string vector 中，最後再輸出。

```
int Depth(vector<vector<int>> &r,int root,int left,int right)
{
    int ldep, rdep;
    if(root-1 < left)
        ldep = 1;
    else
    {
        int lroot = r[left][root-1];
        ldep = Depth(r,lroot,left,root-1) + 1;
    }
    if(root+1 > right)
        rdep = 1;
    else
    {
        int rroot = r[root+1][right];
        rdep = Depth(r,rroot,root+1,right) + 1;
    }
    return max(ldep,rdep);
}
```

Depth，利用將 r table(即 pseudo code 中的 root table)結合 recursive 的方式來 back track 整個 OBST 的高度為何。

```
void fillOBST(vector<vector<int>> &r,vector<vector<string>> &obst,int root,int left,int right,int level)
{
    obst[level][2*root] = "k" + to_string(root);
    if(root-1 < left)
        obst[level+1][2*root-1] = "d" + to_string(root-1);
    else
    {
        int lroot = r[left][root-1];
        fillOBST(r,obst,lroot,left,root-1,level+1);
    }
    if(root+1 > right)
        obst[level+1][2*root+1] = "d" + to_string(root);
    else
    {
        int rroot = r[root+1][right];
        fillOBST(r,obst,rroot,root+1,right,level+1);
    }
}
```

fillOBST，同樣利用 r table 結合 recursive 的方式來 back track 整個 OBST 在不同高度所需輸出的 string，並存入二維 string vector 中。

討論

1.

實際執行我所打的程式碼結果如下，與題目之圖有所差別。

```
Smallest search cost: 2.75
Root: 3

Optimal Binary Search Tree are below:


              k3

         k2          k4

   k1        d2    d3        k5

d0    d1                  d4    d5
```

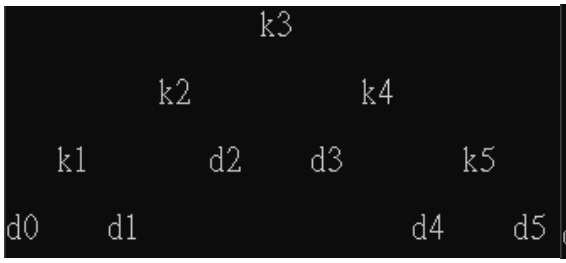
但實際計算題目之圖的 Search Cost 後可以發現，此圖的 Search Cost 和我打的程式碼的 Search Cost 一樣都是 2.75，表示此兩個 BST 都是 optimal 的 BST。這也是因為我的程式碼如下，在找到 Search Cost 比較小時才會更新 e 跟 r 的 table，然而當 Search Cost 一樣時，此寫法僅只會保留 root 較小的 optimal BST，當我將 t < e[i][j] 改為 t <= e[i][j] 後，便能產生出題目所提供之下圖

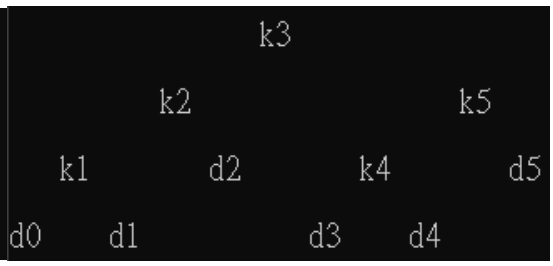
```
double t = e[i][k-1] + e[k+1][j] + w[i][j];
if(t < e[i][j])
{
    e[i][j] = t;
    r[i][j] = k;
}
```

```
Smallest search cost: 2.75
Root: 3

Optimal Binary Search Tree are below:

                k3

          k2                  k5

    k1          d2          k4          d5

d0      d1              d3      d4
```

2.

實際所能找到的 Optimal BST 如上題有兩個，分別如下圖





執行結果

根據下面 table 所產生的 OBST 如下圖

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

```
Smallest search cost: 3.45
Root: 5

Optimal Binary Search Tree are below:

                          k5

          k2                                  k7

    k1                  k4              k6              k8

d0      d1          k3          d4      d5      d6      d7          k9

              d2      d3                                          d8      d9
```

根據下面 table 所產生的 OBST 如下圖

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