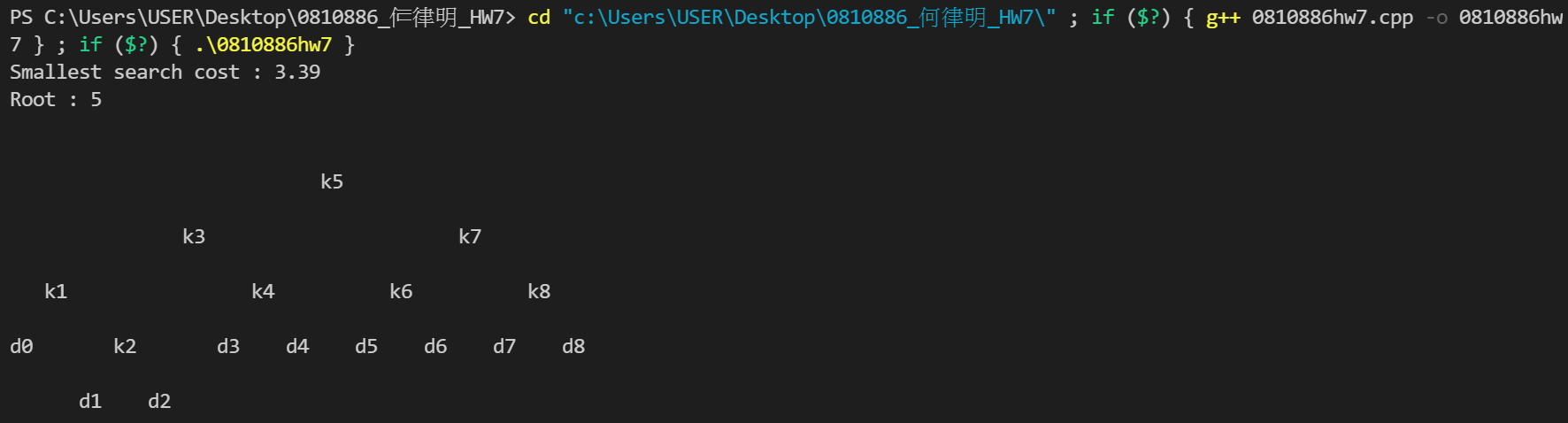
【作業要求】

1.

Result:

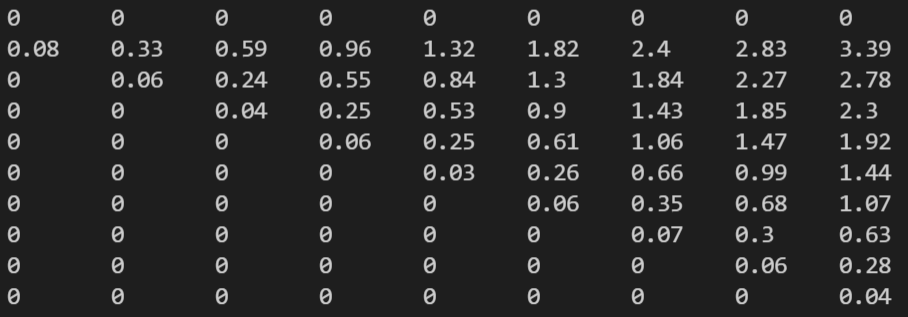


最小的Expected search cost : 3.39

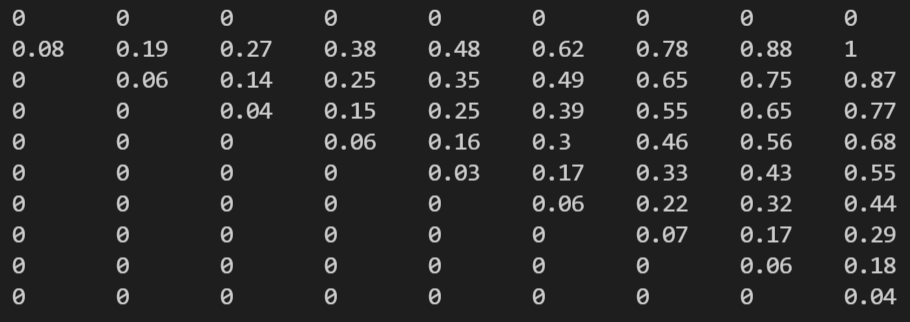
Root : 5

Table:

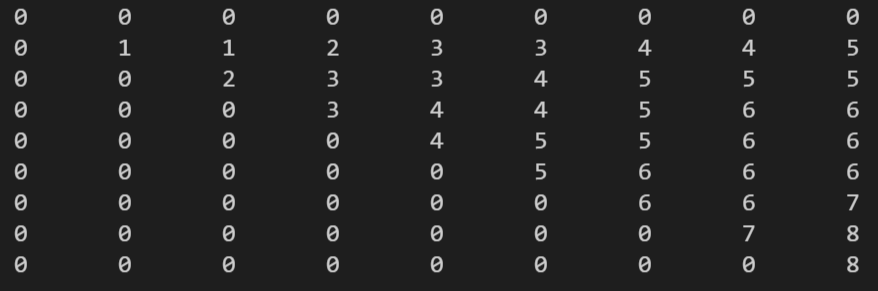
e:



w:

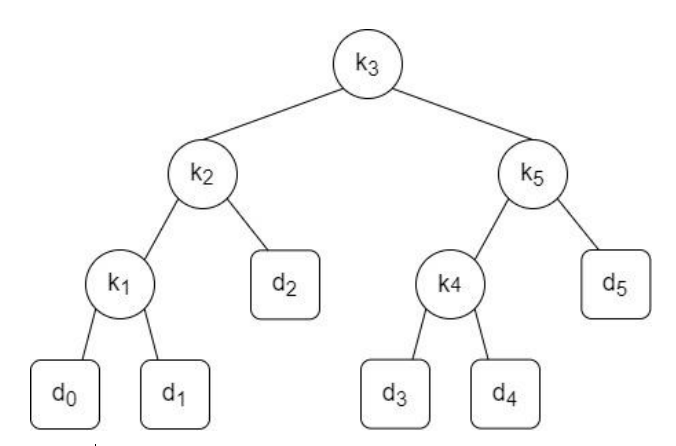


Root:

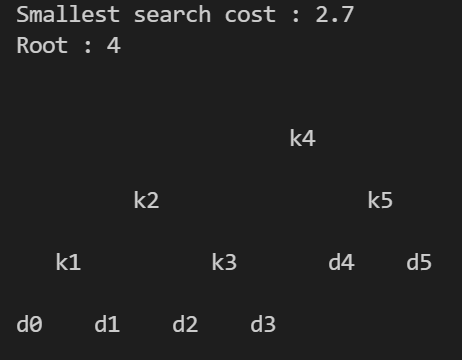


【討論】

1.

題目: 

Real result:



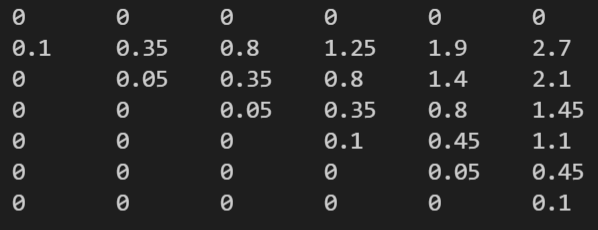
🡪此題的題目的圖並非Optimal Binary Search Tree，真正的Optimal Binary Search Tree的root為4(如上附程式結果圖)，

Smallest expected search cost: 2.7

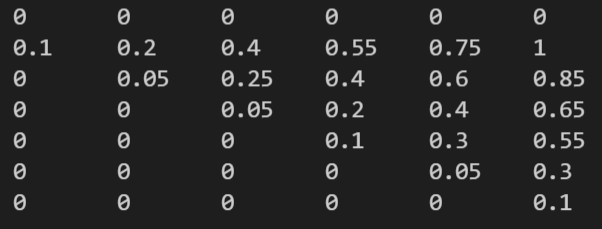
Root: 4

Table:

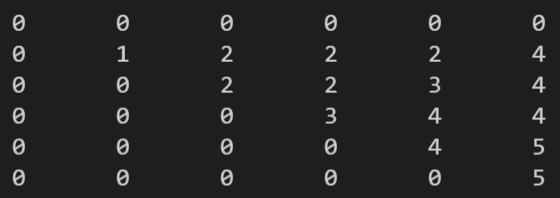
e:



w:

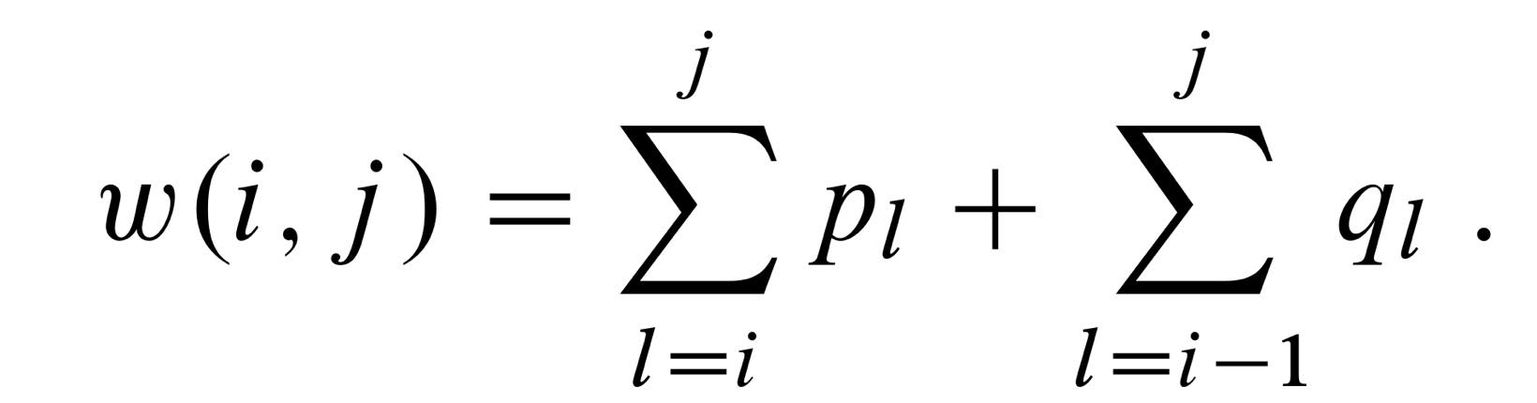


Root:

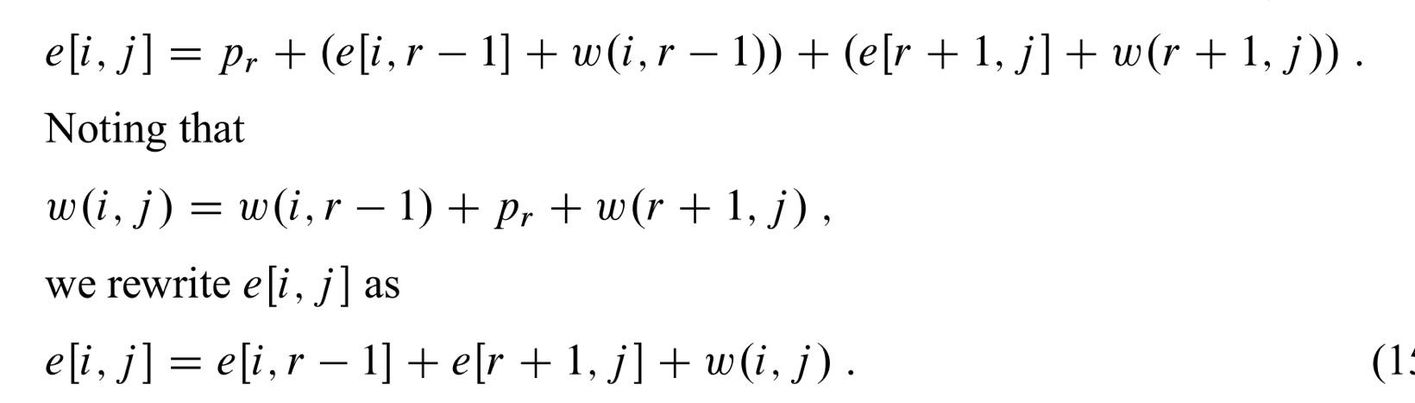


2.

W存在的目的為因為dynamic programming的方式去實踐解決這個問題，於是會用到小的subproblems 的 solutions 去堆疊產生出較大subproblem 的 solution ，所以說假設我們計算完一個高度為n的subtree之expected cost，但我們把這個subtree裝在另一個node的left 或 right上，這時候這個subtree各個node的深度+1，所以會用到w這個table (這subtree各node的機率總和為多少)來儲存當subtree深度+1時，total cost會加上多少。 課本解釋 :



代表從ki ~ kj ， di-1 ~ dj 各node查詢的機率總和。



e[i, j] 為其左subtree原本的cost (e[i, r-1]) 加上其右subtree原本的cost (e[r+1, j]) + 新root所造成的cost (Pr)+ 兩個subtree深度+1所多加的cost (w(i, j))。解釋 : 因為左subtree跟右subtree的各node深度+1，所以新的BST cost要加上w[i, r-1] (左subtree各node的機率總和) 跟 w[r+1, j] (右subtree各node的機率總和)，此外，還有這整個BST新的root所造成的cost (Pr)，根據w(i, j)的定義，w[i, r-1] 、 w[r+1, j] 、三項可以合併為 w(i, j)。這就是w存在的原因。

Code Introduction:

void new2d(double\*\* &e, double\*\* &w, double\*\* &root){

    int n = pi.size()-1; //8

    e = new double\* [n+2]; //[0~n+1, 0~n] (n+2)\*(n+1)

    w = new double\* [n+2]; //[0~n+1, 0~n] (n+2)\*(n+1)

    root = new double\*[n+1]; //[0~n, 0~n] (n+1)\*(n+1)

    for(int i=0 ; i<n+2 ; ++i){

        e[i] = new double[n+1];

        w[i] = new double[n+1];

    }

    for(int i=0 ; i<n+1 ; ++i){

        root[i] = new double[n+1];

    }

    //Init w and e

    for(int i=0 ; i<n+2 ; ++i){

        for(int j=0 ; j<n+1 ; ++j){

            e[i][j] = 0;

            w[i][j] = 0;

        }

    }

    //Init root

    for(int i=0 ; i<n+1 ; ++i){

        for(int j=0 ; j<n+1 ; ++j){

            root[i][j] = 0;

        }

    }

}

根據不同pi大小來創造出不同大小的3個所需table。

void delete2d(double\*\* &e, double\*\* &w, double\*\* &root){

    int n = pi.size()-1;

    for(int i=0 ; i<=n+1 ; ++i){

        if(i <= n){

            delete[] e[i];

            delete[] w[i];

            delete[] root[i];

        }

        else{

            delete[] e[i];

            delete[] w[i];

        }

    }

    delete[] e;

    delete[] w;

    delete[] root;

}

程式執行完後釋放3個table的空間。

int caldepth(double\*\* root, int k, int i, int j){

    int r = root[i][j];

    if(k == r){

        return 0;

    }

    else if(k < r){

        return caldepth(root, k, i, r-1) + 1;

    }

    else

        return caldepth(root, k, r+1, j) + 1;

}

給定key值找出k(key)所在tree的深度。

void Optimal\_BST(vector<double> pi, vector<double> qi, double\*\* &e, double\*\* &w, double\*\* &root){

    int n = pi.size()-1;

    for(int i=1 ; i<=n+1 ; ++i){

        e[i][i-1] = qi[i-1];

        w[i][i-1] = qi[i-1];

    }

    for(int l=1 ; l<=n ; ++l){

        for(int i=1 ; i<=n-l+1 ; ++i){

            int j = i + l - 1;

            e[i][j] = INT\_MAX;

            w[i][j] = w[i][j-1] + pi[j] + qi[j];

            for(int r=i ; r<=j ; ++r){

                double t = e[i][r-1] + e[r+1][j] + w[i][j];

                if(t <= e[i][j]){

                    e[i][j] = t;

                    root[i][j] = r;

                }

            }

        }

    }

}

根據pseudo code。

void drawBST(double\*\* &e, double\*\* &root){

    int n = pi.size()-1;

    int maxdepth = 0, tmp;

    for(int i=1 ; i<=n ; ++i){

        tmp = caldepth(root, i, 1, n);

        if(tmp > maxdepth)

            maxdepth = tmp;

    }

    // Add the depth of dummys

    ++maxdepth;

    //Init table for drawing BST

    string \*\*t = new string\* [maxdepth+1];

    for(int i=0 ; i<=maxdepth ; ++i){

        t[i] = new string[2\*n + 1];

    }

    for(int i=0 ; i<=maxdepth ; ++i){

        for(int j=0 ; j<2\*n + 1 ; ++j){

            t[i][j] = "  ";

        }

    }

    //Put ki in the table

    int dep;

    for(int i=1 ; i<=n ; ++i){

        dep = caldepth(root, i, 1, n);

        t[dep][2\*i - 1] = "k" + to\_string(i);

    }

    //Put dummy in the table

    int d0 = caldepth(root, 1, 1, n);

    t[d0+1][0] = "d" + to\_string(0);

    int d8 = caldepth(root, n, 1, n);

    t[d8+1][2\*n] = "d" + to\_string(n);

    for(int i=1 ; i<n ; ++i){

        int q1 = caldepth(root, i, 1, n);

        int q2 = caldepth(root, i+1, 1, n);

        if(q1 > q2){ //Put on the right subtree of ki

            t[q1+1][2\*i] = "d" + to\_string(i);

        }

        else if(q1 < q2){ //Put on the left subtree of ki+1

            t[q2+1][2\*i] = "d" + to\_string(i);

        }

    }

    //Print

    for(int i=0 ; i<=maxdepth ; ++i){

        for(int j=0 ; j<2\*n + 1 ; ++j){

            cout<<t[i][j]<<" ";

        }

        cout<<endl<<endl;

    }

    //Free the space

    for(int i=0 ; i<=maxdepth ; ++i){

        delete [] t[i];

    }

    delete [] t;

}

畫出最終的Optimal BST。

int main(){

    double \*\*e, \*\*w, \*\*root;

    int n = pi.size() - 1;

    new2d(e, w, root);

    Optimal\_BST(pi, qi, e, w, root);

    cout<<"Smallest search cost : "<<e[1][n]<<endl;

    cout<<"Root : "<<root[1][n]<<endl;

    cout<<endl<<endl;

    drawBST(e, root);

    delete2d(e, w, root);

    return 0;

}

main 檔。