Lab05-DynamicProgramming

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

∗ Name: Haoyi You Student ID: 519030910193 Email: yuri-you@sjtu.edu.cn

1. *Optimal Binary Search Tree.* Given a sorted sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys, and we wish to build a binary search tree from these keys. For each key k_i, we have a probability p_i that a search will be for k_i. Some searches may be for values not in K, and so we also have $n+1$ *dummy keys* $d_0, d_1, d_2, \ldots, d_n$ representing values not in K. In particular, d_0 represents all values less than k_1, and d_n represents all values greater than k_n. For $i = 1, 2, \ldots, n-1$, the dummy key d_i represents all values between k_i and k_{i+1}. For each dummy key d_i, we have a probability q_i that a search will correspond to d_i. Each key k_i is an internal node, and each dummy key d_i is a leaf. Every search is either successful (finding some key k_i) or unsuccessful (finding some dummy key d_i), and so we have $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$.

   (a) Prove that if an optimal binary search tree T (T has the smallest expected search cost) has a subtree T' containing keys $k_i, \ldots, k_j$, then this subtree T' must be optimal as well for the subproblem with keys $k_i, \ldots, k_j$ and dummy keys $d_{i-1}, \ldots, d_j$.

   (b) We define $e[i,j]$ as the expected cost of searching an optimal binary search tree containing the keys $k_i, \ldots, k_j$. Our goal is to compute $e[1,n]$. Write the state transition equation and pseudocode using **dynamic programming** to find the minimum expected cost of a search in a given binary tree. (**Remark**: You may use $w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$).

   (c) Implement your proposed algorithm in C/C++ and analyze the time complexity. (The framework Code-OBST.cpp is attached on the course webpage). Give the minimum search cost calculated by your algorithm. The test case is given as following:

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

   (d) Please draw the structure of the optimal binary search tree in the test case, and explain the drawing process.

Solution.

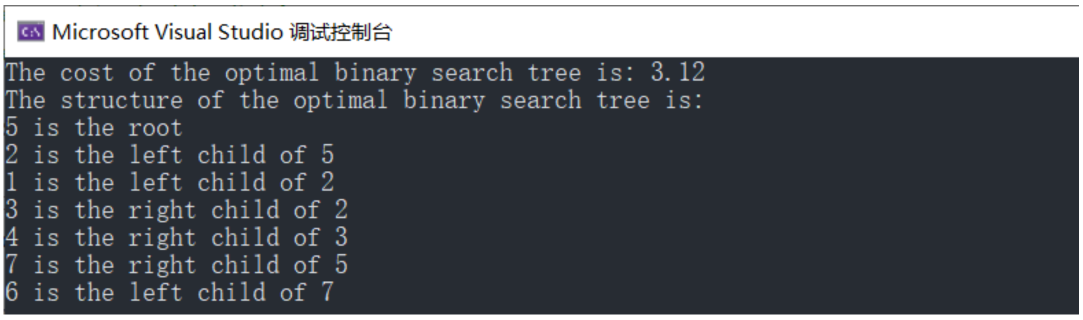
   (a) We assume the depth of k_i is h_i,and depth of d_i is g_i. Let depth of the root of T' is h. If T' is not optimal(depth of the node (in T')is $h_i - h, \ldots, h_j - h$),there exists another optimal solution T''(depth of the node is $h'_i - h, \ldots, h'_j - h$). So we build another tree S with $\forall$ node $k \in S$,if $k \in T', h(S)_k = h'_k$,else $h(S)_k = h'_i$.
   $Cost_T - Cost_S = (h_i - h'_i) * p_i \ldots (h_j - h'_j) * p_j > 0$,so T is not optimal.

(b)

```
Algorithm 1: Optimal Binary Search Tree
  Input: probability array $\{p_1, p_2 \dots p_n\}$ and probability array $\{q_0, q_1 \dots q_n\}$
  Output: minimum expected cost C
1  $e[n][n] \leftarrow 0$;
2  $w[n][n] \leftarrow 0$;
3  for $i = 0$ to $n-1$ do
4      $w[i][i] \leftarrow p_{i+1} + q_i + q_{i+1}$;//array index begins at 0
5  for $i = 1$ to $n-1$ do
6      for $j = 0$ to $n-1-i$ do
7          $w[j][j+i] \leftarrow w[j][j+i-1] + p_{j+i+1} + q_{j+i+1}$;
8  for $i = 0$ to $n-1$ do
9      for $j = 0$ to $n-i-1$ do
10         for $k = 0$ to $i$ do
11             if $k == 0$ then
12                 $left \leftarrow 2 * q_j$;//left subtree is a leaf
13             else
14                 $left \leftarrow e[j-1][j+k-1] + w[j-1][j+k-1]$;
15             if $k==i$ then
16                 $right \leftarrow 2 * q_{j+i+1}$;//right subtree is a leaf
17             else
18                 $right \leftarrow e[j+1+k][j+i-1] + w[j+1+k][j+i-1]$;
19             $sum \leftarrow left + right + p_{j+k}$;
20             if $sum > e[j][j+i]$ then
21                 $e[j][j+i] \leftarrow sum$;
22 return $e[0][n-1]$;
```

(c) the answer is 3.12,the code is in the appendix and for the result result please refer to Figure 3

(d) The structure is in Figure 3. You can also refer to Figure 2
The drawing process is a recursive process using algorithm DFS. Firstly we find the root of the whole tree,assuming as k. Then we would do this recursively to the left subtree and right subtree of this tree,that is to do this $[1, k-1]$ and $[k+1, n]$.
The whole process rely to a recursive function *void construct_optimal_bst*$(int\ i, int\ j)$



```
The cost of the optimal binary search tree is: 3.12
The structure of the optimal binary search tree is:
5 is the root
2 is the left child of 5
1 is the left child of 2
3 is the right child of 2
4 is the right child of 3
7 is the right child of 5
6 is the left child of 7
```

Figure 1: OBST result

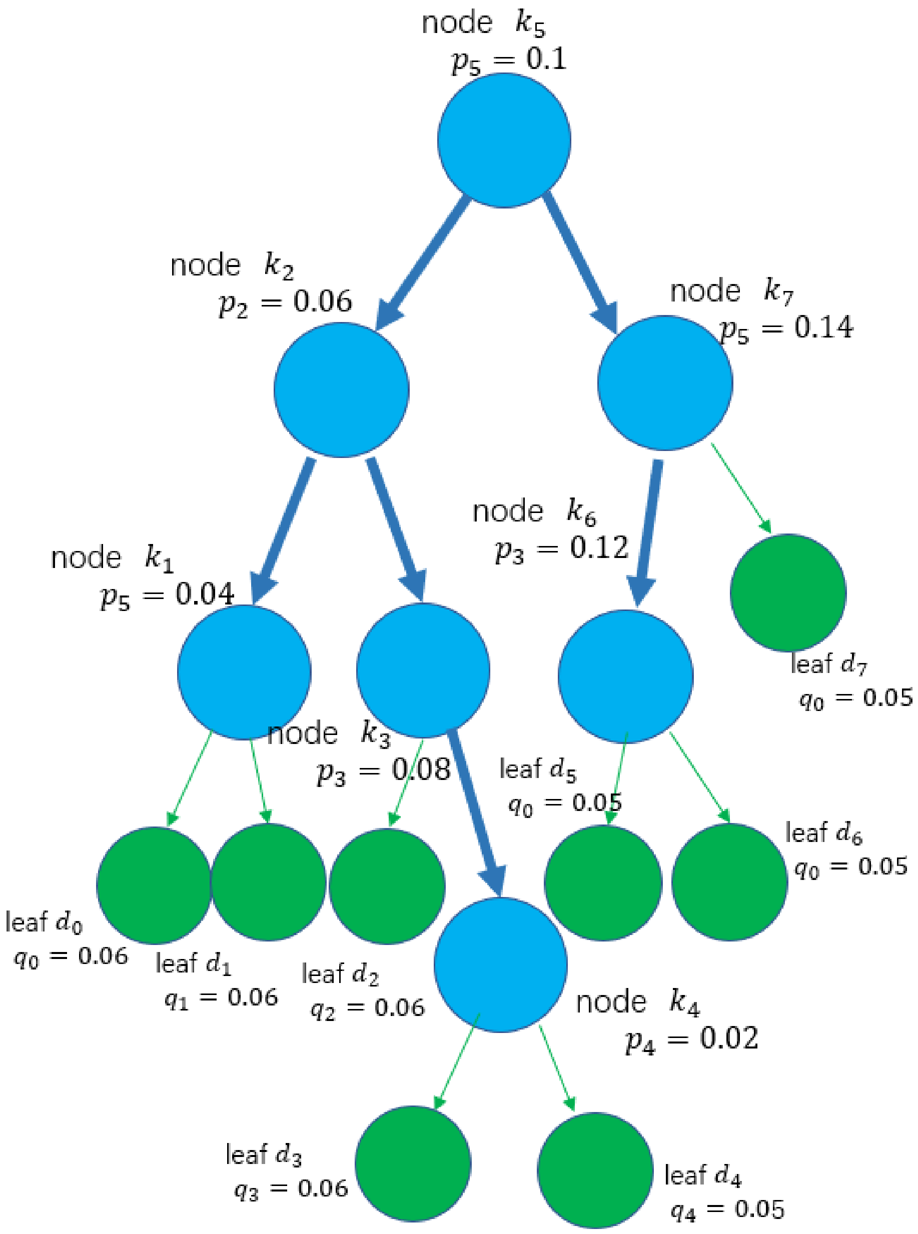


Figure 2: OBST structure

□

2. *Dynamic Time Warping Distance.* **DTW** stretches the series along the time axis in a dynamic way over different portions to enable more effective matching. Let $DTW(i,j)$ be the optimal distance between the first i and first j elements of two time series $\bar{X} = (x_1 \dots x_n)$ and $\bar{Y} = (y_1 \dots y_m)$, respectively. Note that the two time series are of lengths n and m, which may not be the same. Then, the value of $DTW(i,j)$ is defined recursively as follows:

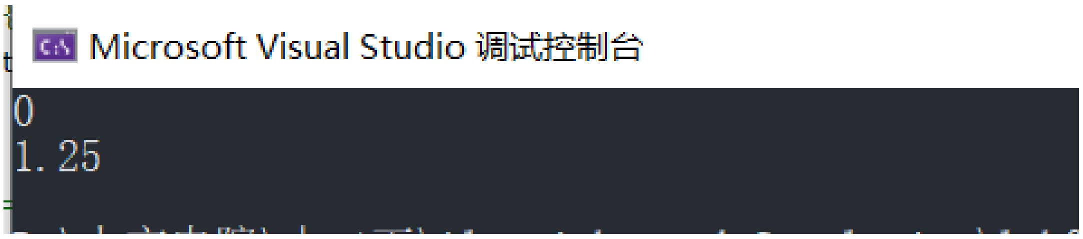
$$DTW(i,j) = |x_i - y_j| + \min(DTW(i,j-1), DTW(i-1,j), DTW(i-1,j-1))$$

   (a) Implement the proposed DTW algorithm in C/C++ and analyze the time complexity of your implementation. (The framework Code-DTW.cpp is attached on the course webpage). Two test cases have been given in the source code.

   (b) The window constraint imposes a minimum level w of positional alignment between matched elements. The window constraint requires that $DTW(i,j)$ be computed only when $|i-j| \le w$. Modify your code to add a window constraint and give the results of $w = 0$ and $w = 1$ on the two test cases.

Solution. (a) The answer is 0 and 1.25.Please refer to **??** for more details.

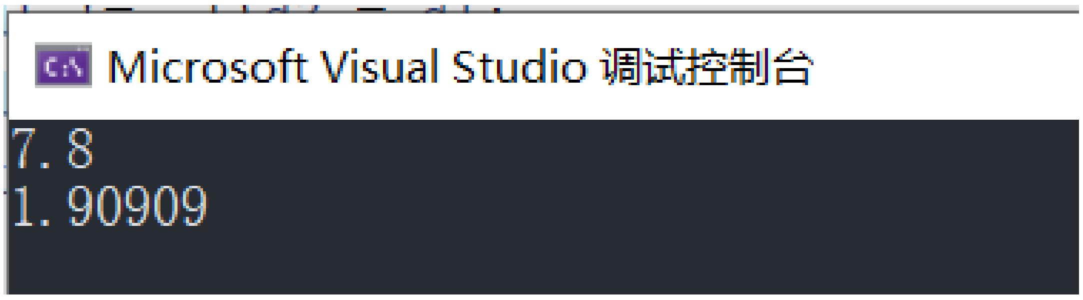
(b) The answer is 7.8 and 1.9.Please refer to 4 for more details.



Microsoft Visual Studio 调试控制台

```
0
1.25
```

Figure 3: DTW result



Microsoft Visual Studio 调试控制台

```
7.8
1.90909
```

Figure 4: DTW structure

□

A Appendix1

```
#include <iostream>
using namespace std;
#define MAX 10000
const int n = 7;
double p[n + 1] = {0,0.04,0.06,0.08,0.02,0.10,0.12,0.14};
double q[n + 1] = {0.06,0.06,0.06,0.06,0.05,0.05,0.05,0.05};
int root[n + 1][n + 1];
double e[n + 2][n + 2] = { 0 };
double w[n + 2][n + 2] = { 0 };
void optimal_binary_search_tree(double *p,double *q,int n)
{
    for (int i = 1; i <= n; ++i) {
        w[i][i] = p[i] + q[i - 1] + q[i];
    }
    for (int i = 1; i <=n; ++i) {
        for (int j = 1; j <= n - i; ++j) {
            w[j][i + j] = w[j][j + i - 1] + p[j + i] + q[j + i];
        }
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n - i+1; ++j) {
            for (int k = 1; k <= i; ++k) {
                double left, right;
                if (k == 1) {
                    left = 2 * q[j - 1];
                }
                else {
                    left = e[j][j + k-2] + w[j][j + k-2];
                }
                if (k == i) {
                    right = 2 * q[j + i -1];
                }
                else {
                    right = e[j + k][j + i - 1] + w[j + k][j + i - 1];
                }

                double sum = left + right + p[j + k - 1];
                if (e[j][i + j - 1] == 0||e[j][i + j - 1] > sum) {
                    e[j][i + j - 1] = sum;
                    root[j][i + j - 1] = j + k - 1;
                }
            }
        }
    }

}

void construct_optimal_bst(int i, int j)
{
    if (i == 1 && j == n) {
        cout << root[1][n] << ”␣is␣the␣root\n”;
    }
    int root1 = root[i][j];
    if (root1 != i) {
        cout << root[i][root1 - 1] << ”is␣the␣left␣child␣of” << root1 << endl
        construct_optimal_bst(i, root1 - 1);
    }
    if (root1 != j) {
        cout<<root[root1+1][j]<< ”␣is␣the␣right␣child␣of␣” << root1 << endl;
        construct_optimal_bst(root1 + 1,j);
    }
}
/*
Please write your code here.
*/


int main()
{
    optimal_binary_search_tree(p,q,n);
    cout<<”The␣cost␣of␣the␣optimal␣binary␣search␣tree␣is:␣”<<e[1][n]<<endl;
    cout << ”The␣structure␣of␣the␣optimal␣binary␣search␣tree␣is:␣” << endl;
    construct_optimal_bst(1,n);
    return 0;
}
```

B Appendix2

```
#include <iostream>
#include <vector>
#include <cmath>
#include <numeric>
/*
The process to calculate the dynamic can be divided into four steps:
1.Create an empty cost matrix DTW with X and Y labels as amplitudes of the tu
2.Use the given state transition function to fill in the cost matrix.
3.Identify the warping path starting from top right corner of the matrix and
i.e., When we reach the point (i, j) in the matrix, the next position is to c
For the sake of simplicity, when the cost is equal, the priority of the selec
4.Calculate th time normalized distance. We define it as the average cost of
*/
using namespace std;
void warp_path(const vector<vector<int>>& data, vector<int>& ans, int n, int
    if (n == 0) {
        if (m != 0)warp_path(data, ans, n, m - 1);
    }
    else {
        if (m == 0)warp_path(data, ans, n - 1, m);
        else {
            int min = data[n - 1][m - 1];
            if (min > data[n - 1][m])min = data[n - 1][m];
            if (min > data[n][m - 1])min = data[n][m - 1];
            if (min == data[n - 1][m - 1])warp_path(data, ans, n - 1, m - 1);
            else {
                if(min==data[n][m-1])warp_path(data, ans, n, m - 1);
                else {
                    warp_path(data, ans, n-1, m);
                }
            }
        }
    }
    ans.push_back(data[n][m]);
}
double distance(vector<int> x, vector<int> y) {
    int n = x.size();
    int m = y.size();
    vector<vector<int>> DTW;
    //Use the given state transition function to fill in the cost matrix.
    /*
    Please write your code here.

    */
    for (int i = 0; i < n; ++i) {
        DTW.push_back(vector<int>());
        for (int j = 0; j < n; ++j) {
            int d1, d2, d3;
            if (i == 0) {
                if (j == 0) {
                    d1 = d2 = d3 = 0;
                }
                else {
                    d2 = DTW[i][j - 1];
                    d1 = d3 = d2+1;
                }
            }
            else {
                if (j == 0) {
                    d1 = DTW[i - 1][j];
                    d2 = d3 = d1+1;
                }
                else {
                    d1 = DTW[i - 1][j];
                    d2 = DTW[i][j - 1];
                    d3 = DTW[i-1][j - 1];
                }
            }
            if (d2 < d1)d1 = d2;
            if (d3 < d1)d1 = d3;
            DTW[i].push_back(d1 + abs(x[i] - y[j]));
        }
    }
    vector<int> d;
    warp_path(DTW, d, n-1, m-1);
    //Identify the warping path.
    /*
    Please write your code here.
    */

    double ans = 0;
    ans = double(DTW[n-1][m-1]) / d.size();
    //Calculate th time normalized distance
    /*
    Please write your code here.
    */
    return ans;
}

int main(){
        vector<int> X,Y;
        //test case 1
        X = {37,37,38,42,25,21,22,33,27,19,31,21,44,46,28};
        Y = {37,38,42,25,21,22,33,27,19,31,21,44,46,28,28};
        cout<<distance(X,Y)<<endl;
        //test case 2
        X = {11,14,15,20,19,13,12,16,18,14};
        Y = {11,17,13,14,11,20,15,14,17,14};
        cout<<distance(X,Y)<<endl;
        //Remark: when you modify the code to add the window constraint, the
        return 0;
}
\section{Appendix1}
\begin{lstlisting}[language=C++]
#include <iostream>
using namespace std;
#define MAX 10000
const int n = 7;
double p[n + 1] = {0,0.04,0.06,0.08,0.02,0.10,0.12,0.14};
double q[n + 1] = {0.06,0.06,0.06,0.06,0.05,0.05,0.05,0.05};
int root[n + 1][n + 1];
double e[n + 2][n + 2] = { 0 };
double w[n + 2][n + 2] = { 0 };
void optimal_binary_search_tree(double *p,double *q,int n)
{
    for (int i = 1; i <= n; ++i) {
        w[i][i] = p[i] + q[i - 1] + q[i];
    }
    for (int i = 1; i <=n; ++i) {
        for (int j = 1; j <= n - i; ++j) {
            w[j][i + j] = w[j][j + i - 1] + p[j + i] + q[j + i];
        }
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n - i+1; ++j) {
            for (int k = 1; k <= i; ++k) {
                double left, right;
                if (k == 1) {
                    left = 2 * q[j - 1];
                }
                else {
                    left = e[j][j + k-2] + w[j][j + k-2];
                }
                if (k == i) {
                    right = 2 * q[j + i-1];
                }
                else {
                    right = e[j + k][j + i - 1] + w[j + k][j + i - 1];
                }

                double sum = left + right + p[j + k - 1];
                if (e[j][i + j - 1] == 0||e[j][i + j - 1] > sum) {
                    e[j][i + j - 1] = sum;
                    root[j][i + j - 1] = j + k - 1;
                }
            }
        }
    }

}

void construct_optimal_bst(int i, int j)
{
    if (i == 1 && j == n) {
        cout << root[1][n] << "␣is␣the␣root\n";
    }
    int root1 = root[i][j];
    if (root1 != i) {
        cout << root[i][root1 - 1] << "is␣the␣left␣child␣of" << root1 << endl
        construct_optimal_bst(i, root1 - 1);
    }
    if (root1 != j) {
        cout<<root[root1+1][j]<< "␣is␣the␣right␣child␣of␣" << root1 << endl;
        construct_optimal_bst(root1 + 1,j);
    }
}
/*
Please write your code here.
*/
```

```
int main()
{
    optimal_binary_search_tree(p,q,n);
    cout<<"The␣cost␣of␣the␣optimal␣binary␣search␣tree␣is:␣"<<e[1][n]<<endl;
    cout << "The␣structure␣of␣the␣optimal␣binary␣search␣tree␣is:␣" << endl;
    construct_optimal_bst(1,n);
    return 0;
}
```

C Appendix3

```
#include <iostream>
#include <vector>
#include <cmath>
#include <numeric>
/*
The process to calculate the dynamic can be divided into four steps:
1.Create an empty cost matrix DTW with X and Y labels as amplitudes of the tw
2.Use the given state transition function to fill in the cost matrix.
3.Identify the warping path starting from top right corner of the matrix and
i.e., When we reach the point (i, j) in the matrix, the next position is to c
For the sake of simplicity, when the cost is equal, the priority of the selec
4.Calculate th time normalized distance. We define it as the average cost of
*/
using namespace std;
void warp_path(const vector<vector<int>>& data, vector<int>& ans, int n, int
    if (n == 0) {
        if (m != 0)warp_path(data, ans, n, m - 1, constrain);
    }
    else {
        if (m == 0)warp_path(data, ans, n - 1, m, constrain);
        else {
            int min = data[n - 1][m - 1];
            if (min > data[n - 1][m]&&abs(n-1-m)<=constrain)min = data[n - 1]
            if (min > data[n][m - 1] && abs(m - 1 - n) <= constrain)min = dat
            if (min == data[n - 1][m - 1])warp_path(data, ans, n - 1, m - 1,
            else {
                if(min==data[n][m-1])warp_path(data, ans, n, m - 1, constrain
                else {
                    warp_path(data, ans, n-1, m, constrain);
                }
            }
        }
    }
    ans.push_back(data[n][m]);
}
double distance(vector<int> x, vector<int> y,int window_constrain) {
    int n = x.size();
    int m = y.size();
```

```
vector<vector<int>> DTW;
//Use the given state transition function to fill in the cost matrix.
/*
Please write your code here.

*/
for (int i = 0; i < n; ++i) {
    DTW.push_back(vector<int>());
    for (int j = 0; j < n; ++j) {
        if (abs(i - j) > window_constrain) {
            DTW[i].push_back(-1);
            continue;
        }
        int d1, d2, d3;
        if (i == 0) {
            if (j == 0) {
                d1 = d2 = d3 = 0;
            }
            else {
                d2 = DTW[i][j - 1];
                d1 = d3 = -1;
            }
        }
        else {
            if (j == 0) {
                d1 = DTW[i - 1][j];
                d2 = d3 = -1;
            }
            else {
                d1 = DTW[i - 1][j];
                d2 = DTW[i][j - 1];
                d3 = DTW[i - 1][j - 1];
            }
        }
        if (d3 == -1) {
            if (d2 != -1)d3 = d2 + 1;
            else d3 = d1 + 1;
        }
        if (d1 < d3 && d1 != -1)d3 = d1;
        if (d2 < d3 && d2 != -1)d3 = d2;
        DTW[i].push_back(d3 + abs(x[i] - y[j]));
    }
}
vector<int> d;
warp_path(DTW, d, n-1, m-1,window_constrain);
//Identify the warping path.
/*
Please write your code here.
*/
```

```
    double ans = 0;
    ans = double(DTW[n-1][m-1]) / d.size();
    //Calculate th time normalized distance
    /*
    Please write your code here.
    */
    return ans;
}

int main(){
        vector<int> X,Y;
        //test case 1
        X = {37,37,38,42,25,21,22,33,27,19,31,21,44,46,28};
        Y = {37,38,42,25,21,22,33,27,19,31,21,44,46,28,28};
    X1 = X, Y1 = Y;
        cout<<distance(X,Y,0)<<endl;
        //test case 2
        X = {11,14,15,20,19,13,12,16,18,14};
        Y = {11,17,13,14,11,20,15,14,17,14};
    X1 = X, Y1 = Y;
        cout<<distance(X,Y,1)<<endl;
        //Remark: when you modify the code to add the window constraint, the
        return 0;
}
```