Lab05-DynamicProgramming

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

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name:Zirui Liu Student ID:519021910343 Email: L.Prime@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 can use contradiction. We assume that there exists a subtree T_2 which has lower cost than T_1, then we could replace this T_1 with T_2 without moving other parts of the tree, so a contradiction is resulted. So the original conclusion is proved.

(b):

Algorithm 1:

Input: Three arrays $e[]$,w,$root$,p,q
Output: void

```
for i ← 1 to n + 1 do
    e[i][i − 1] = q[i − 1];
    w[i][i − 1] = q[i − 1];
for m ← 1 to n do
    for i ← 1 to n − m + 1 do
        j = i + m − 1;
        e[i][j] = ∞;
        w[i][j] = w[i][j − 1] + p[j] + q[j];
        for root1 ← i to j do
            value = e[i][root1 − 1] + e[root1 + 1][j] + w[i][j] ;
            if value < w[i][j] then
                e[i][j] = value;
                root[i][j] = root1;
return ;
```

(c):
The minimum search cost calculated by your algorithm is 3.12 . The code can be found in appendix 1, and the pictures of results can be found in appendix 3.

(d):

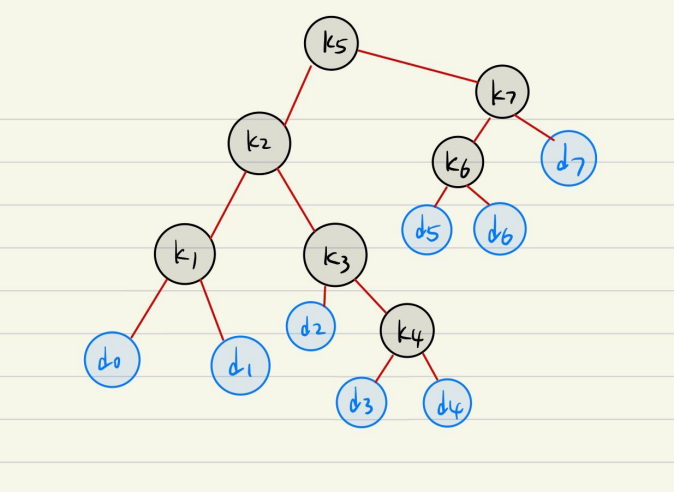

Figure 1: tree

□

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 $|i-j| \leq w$. Modify your code to add a window constraint and give the results of $w = 0$ and $w = 1$ on the two test cases.

Solution. (1):
The code can be found in appendix 2, and the pictures of results can be found in appendix 3.
Here are the time complexity analysis:
The time complexity of the DTW algorithm is $O(NM)$, where N and M are the lengths of the two input sequences. When we assume that $N \geq M$, the time complexity can be up to N^2.

(2):
The code can be found in appendix 2, and the pictures of results can be found in appendix 3.

□

Remark: You need to include your .pdf and .tex and 2 source code files in your uploaded .rar or .zip file. Screenshots of test case results are acceptable.

1 First appendix: Code-OBST.cpp

Input C++ source1:

```
#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};

// const int n = 5;
// double p[n + 1] = {0,0.15,0.10,0.05,0.10,0.20};
// double q[n + 1] = {0.05,0.10,0.05,0.05,0.05,0.10};

int root[n + 1][n + 1];//Record the root node of the optimal subtree
double e[n + 2][n + 2];//Record the expected cost of the subtree
double w[n + 2][n + 2];//Record the probability sum of the subtree

void optimal_binary_search_tree(double *p,double *q,int n)
{
//The result is stored in e.

for (int i=1;i<=(n+1);++i){
    e[i][i-1]=q[i-1];
    w[i][i-1]=q[i-1];
}
for (int m=1;m<=n;++m){
    for(int i=1;i<=n-m+1;++i){
        int j=i+m-1;
        e[i][j]=MAX;
        w[i][j]=w[i][j-1]+p[j]+q[j];
        for(int root_=i;root_<=j;++root_){
            double value=e[i][root_-1]+e[root_+1][j]+w[i][j];
            if(value<e[i][j]){
                e[i][j]=value;
                root[i][j]=root_;
            }
        }
    }
}



}

/*
You can print the structure of the optimal binary search tree based on root[
```

```
The format of printing is as follows:
k2 is the root
k1 is the left child of k2
d0 is the left child of k1
d1 is the right child of k1
k5 is the right child of k2
k4 is the left child of k5
k3 is the left child of k4
d2 is the left child of k3
d3 is the right child of k3
d4 is the right child of k4
d5 is the right child of k5
*/

int leave=0;
void construct_optimal_bst(int i,int j,bool dir)
{
//You can adjust the number of input parameters
//  dir==0 left,  dir==1 right

if(i==j){
    if(dir==0){
    cout<<"k"<<root[i][j]<<" is the left child of k"<<j+1<<endl;
    }else if(dir==1){
        cout<<"k"<<root[i][j]<<" is the right child of k"<<i-1<<endl;
    }
    cout<<"d"<<leave<<" is the left child of k"<<root[i][j]<<endl;
    leave++;
    cout<<"d"<<leave<<" is the right child of k"<<root[i][j]<<endl;
    leave++;
    return;
}

if((i==1) && (j==n)){
    cout<<"k"<<root[1][n]<<" is the root"<<endl;
}else if(dir==0){
    cout<<"k"<<root[i][j]<<" is the left child of k"<<j+1<<endl;
}else if(dir==1){
    cout<<"k"<<root[i][j]<<" is the right child of k"<<i-1<<endl;
}
int r=root[i][j];
//cout<<"ROOT: "<<r<<endl;
if(i<=(r-1)){
    construct_optimal_bst(i,r-1,0);
}else{
    cout<<"d"<<leave<<" is the left child of k"<<r<<endl;
    leave++;
}
if((r+1)<=j){
    construct_optimal_bst(r+1,j,1);
```

```
}else{
    cout<<"d"<<leave<<" is the right child of k"<<r<<endl;
    leave++;
}

return;

}

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,0);
}
```

2 Second appendix: Code-DTW.cpp

Input C++ source1:

```
#include <iostream>
#include <vector>
#include <cmath>
#include <numeric>
#define MAX 100000
/*
Dear TA:
please see this explanation:  on line 43~45
  this place i don't fully understand what the question means by " average co
     did it in my way (which is adding all the |x_i-y_j| on the path). If my
     it should also be quite easy to change to another explanation(adding all
     since i have already printed out the path and the dtw(n,m).
*/

/*
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 t
bottom left. The traversal path is identified based on the neighbor with minim
i.e., When we reach the point (i, j) in the matrix, the next position is to c
point with the smallest cost among (i-1,j-1), (i,j-1), and (i-1,j),
For the sake of simplicity, when the cost is equal, the priority of the
selection is (i-1,j-1), (i,j-1), and (i-1,j) in order.

4. Calculate th time normalized distance. We define it as the average cost of
*/
using namespace std;
vector<vector<pair<int,int>> walk;
vector<vector<int>> d;

double nodes=0,cost=0;
double walker(int n,int m){
    if(n<1 || m<1)
        return 0.0;
    int n_=walk[n][m].first;
    int m_=walk[n][m].second;
    walker(n_,m_);
    cout<<n<<" "<<m<<endl;
    cost+=d[n][m];  //  the place where i got confused about the question
    nodes++;
    return cost/nodes;   //  this place i don't fully understand what the que
    //so i did it in my way. If my understanding is wrong, it should be quite
    // since i have already printed out the path and the dtw(n,m).
}
```

```
int distance(vector<int> x, vector<int> y) {
    int n = x.size();
    int m = y.size();
    x.resize(30);
    y.resize(30);
    for(int i=n;i>0;--i){
        x[i]=x[i-1];
    }
    x[0]=0;
    for(int i=m;i>0;--i){
        y[i]=y[i-1];
    }
    y[0]=0;

    d.clear();
    walk.clear();
    vector<vector<int>> DTW;
    //vector<vector<int>> d;
    DTW.resize(30);
    d.resize(30);
    walk.resize(30);
    for(int i=0;i<30;++i){
        DTW[i].resize(30);
        d[i].resize(30);
        walk[i].resize(30);
    }

    int ans = 0;
    for(int i=0;i<=n;++i){
        for(int j=0;j<=m;++j){
            DTW[i][j]=MAX;
            d[i][j]=abs(x[i]-y[j]);
        }
    }
    DTW[0][0]=0;
    int value=0;
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            value=d[i][j];
            DTW[i][j]=value+min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1]);
            //    (i-1,j-1), (i,j-1), and (i-1,j)
            if(min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1])==DTW[i-1][j-1
                walk[i][j]=make_pair(i-1,j-1);
            }else if(min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1])==DTW[i][
                walk[i][j]=make_pair(i,j-1);
            }else if(min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1])==DTW[i-
                walk[i][j]=make_pair(i-1,j);
            }
        }
    }
```

```
    ans=DTW[n][m];
    cout<<"Path: "<<endl;
    cost=0;
    nodes=0;
    double avg=walker(n,m);
    cout<<"Average: "<<avg<<endl;
    //cout<<n<<"  "<<m<<endl;
    cout<<"Total: ";
    return ans;
}

int distance_w(vector<int> x, vector<int> y,int w) {
    int n = x.size();
    int m = y.size();
    x.resize(30);
    y.resize(30);
    for(int i=n;i>0;--i){
        x[i]=x[i-1];
    }
    x[0]=0;
    for(int i=m;i>0;--i){
        y[i]=y[i-1];
    }
    y[0]=0;

    w=max(w,abs(n-m));

    d.clear();
    walk.clear();
    vector<vector<int>> DTW;
    //vector<vector<int>> d;
    DTW.resize(30);
    d.resize(30);
    walk.resize(30);
    for(int i=0;i<30;++i){
        DTW[i].resize(30);
        d[i].resize(30);
        walk[i].resize(30);
    }

    int ans = 0;
    for(int i=0;i<=n;++i){
        for(int j=0;j<=m;++j){
            DTW[i][j]=MAX;
            d[i][j]=abs(x[i]-y[j]);
        }
    }
    DTW[0][0]=0;
    for(int i=1;i<=n;++i){
        for(int j=max(1,i-w);j<=min(m,i+w);++j){
```

```
            DTW[i][j]=0;
        }
    }

    int value=0;
    for(int i=1;i<=n;++i){
        for(int j=max(1,i-w);j<=min(m,i+w);++j){
            value=d[i][j];
            DTW[i][j]=value+min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1]);
                        //   (i-1,j-1), (i,j-1), and (i-1,j)
            if(min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1])==DTW[i-1][j-1
                walk[i][j]=make_pair(i-1,j-1);
            }else if(min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1])==DTW[i][
                walk[i][j]=make_pair(i,j-1);
            }else if(min(min(DTW[i-1][j],DTW[i][j-1]),DTW[i-1][j-1])==DTW[i-
                walk[i][j]=make_pair(i-1,j);
            }
        }
    }

    ans=DTW[n][m];
    cout<<"Path: ";
    cost=0;
    nodes=0;
    double avg=walker(n,m);
    cout<<"Average: "<<avg<<endl;
    //cout<<n<<"  "<<m<<endl;
    cout<<"Total: ";
    return ans;
}


int main(){
        vector<int> X,Y;
        cout<<"test case 1"<<endl;
    //ini();
        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;
        ///clean();
    cout<<"test case 2"<<endl;
        //ini();
    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;
   // clean();

    // vector<int> a,b;
    // cout<<"test case 3"<<endl;
    // //ini();
```

```
    // a={8, 9, 1 ,9, 6 ,1 ,3, 5};
    // b={2, 5, 4 ,6 ,7 ,8 ,3 ,7, 7, 2};
    // cout<<"Total: "<<distance(a,b)<<endl;
    // //clean();
    // cout<<"test case 4"<<endl;
    // //ini();
    // a={2, 0, 1, 1, 2, 4, 2, 1, 2, 0};
    // b={1, 1, 2, 4, 2, 1, 2, 0};
    // cout<<"Total: "<<distance(a,b)<<endl;
        //clean();
    //Remark: when you modify the code to add the window constraint, the dist


    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_w(X,Y,1)<<endl;
    cout<<distance_w(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};
        cout<<distance_w(X,Y,1)<<endl;
    cout<<distance_w(X,Y,0)<<endl;

    // a={8, 9, 1 ,9, 6 ,1 ,3, 5};
    // b={2, 5, 4 ,6 ,7 ,8 ,3 ,7, 7, 2};
    // cout<<distance_w(a,b,2)<<endl;

    // a={2, 0, 1, 1, 2, 4, 2, 1, 2, 0};
    // b={1, 1, 2, 4, 2, 1, 2, 0};
    // cout<<distance_w(a,b,2)<<endl;



    return 0;
}
```

3 Third appendix: Outcome pictures

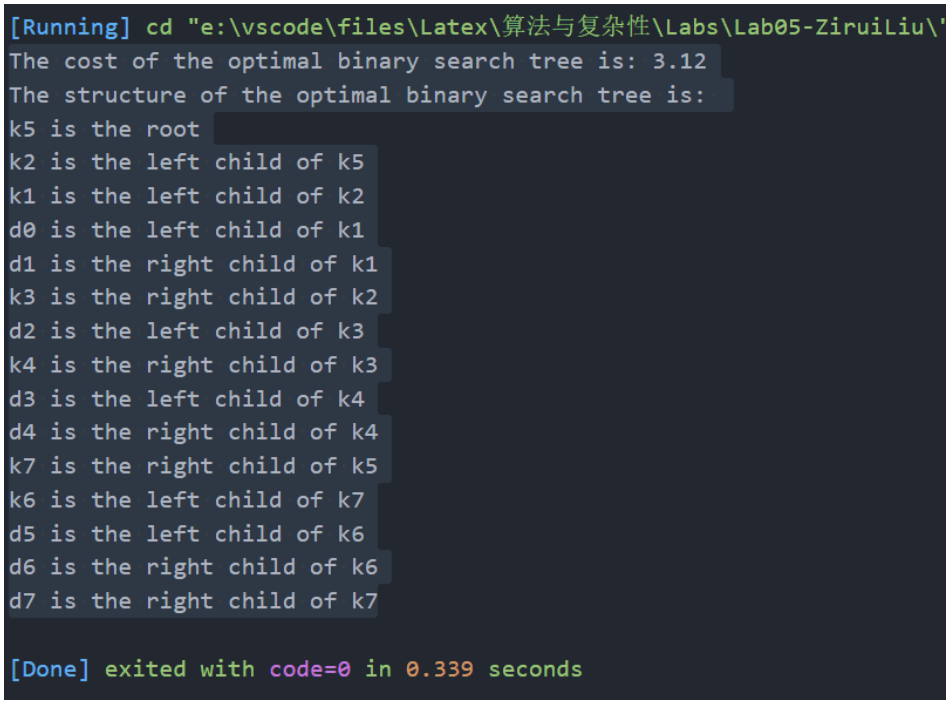

Figure 2: OBST

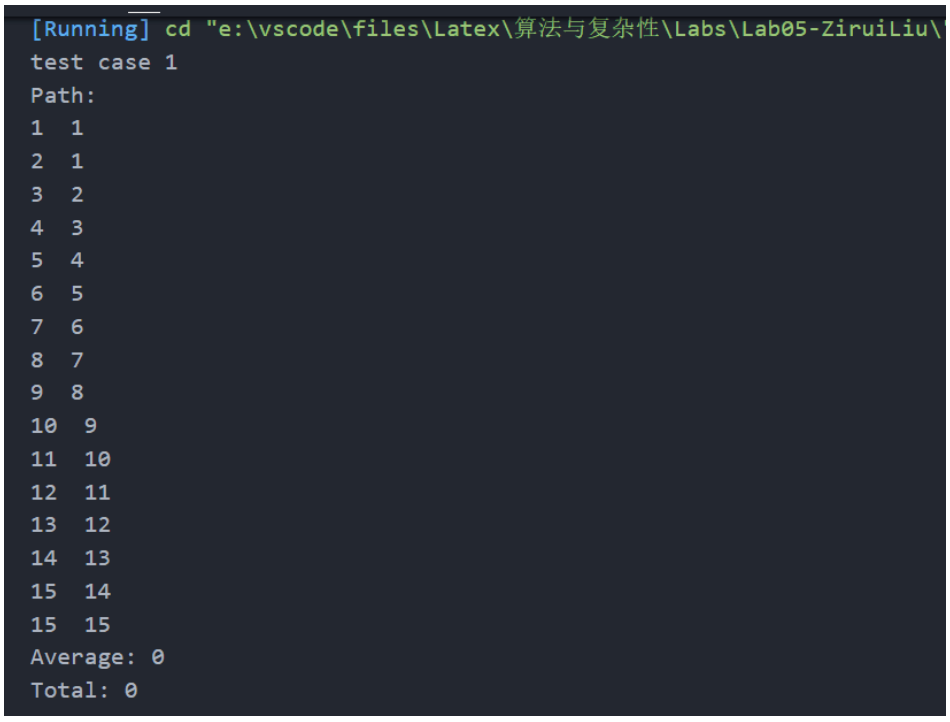

Figure 3: DTW1

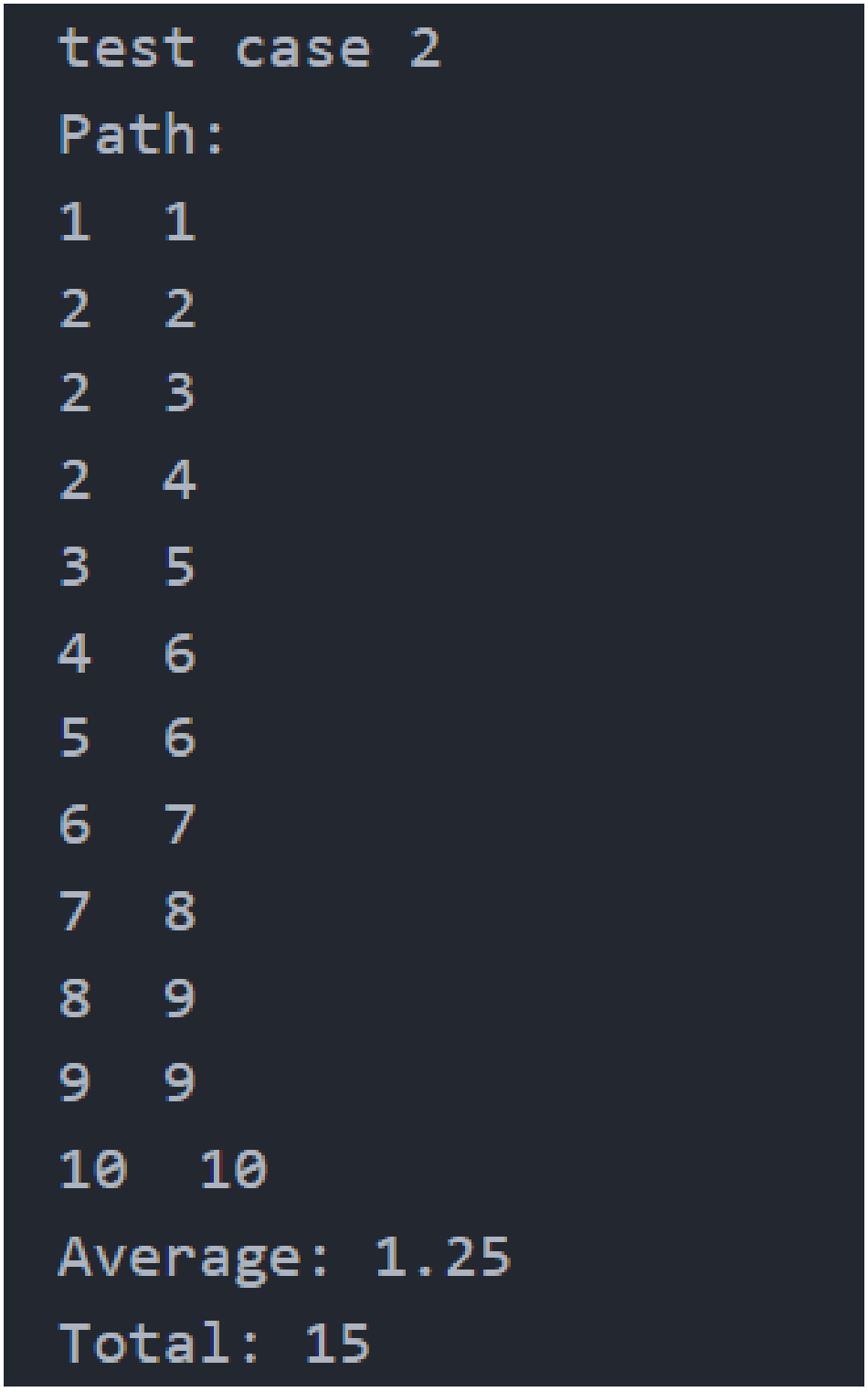

Figure 4: DTW2

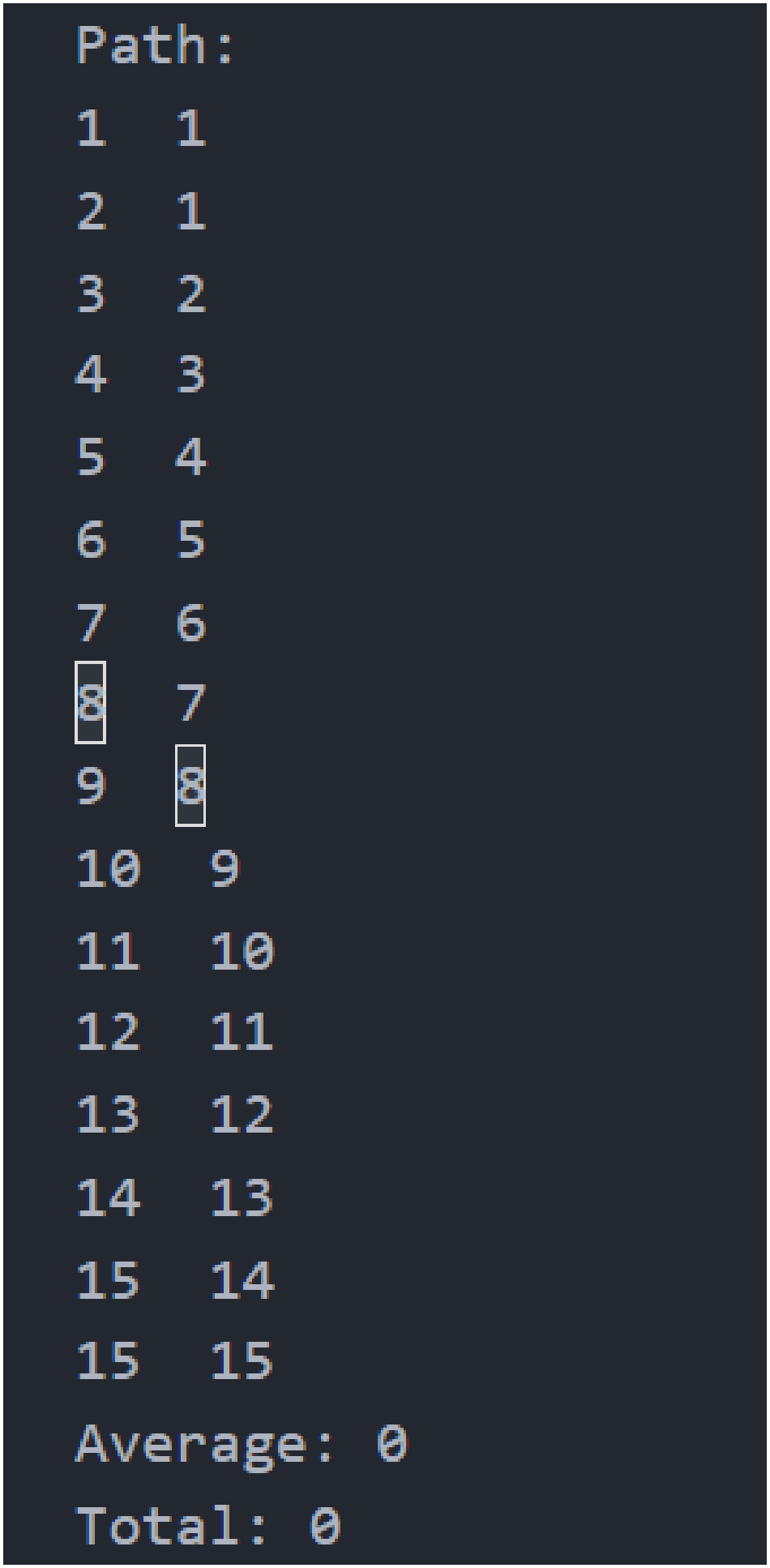

Figure 5: DTW3

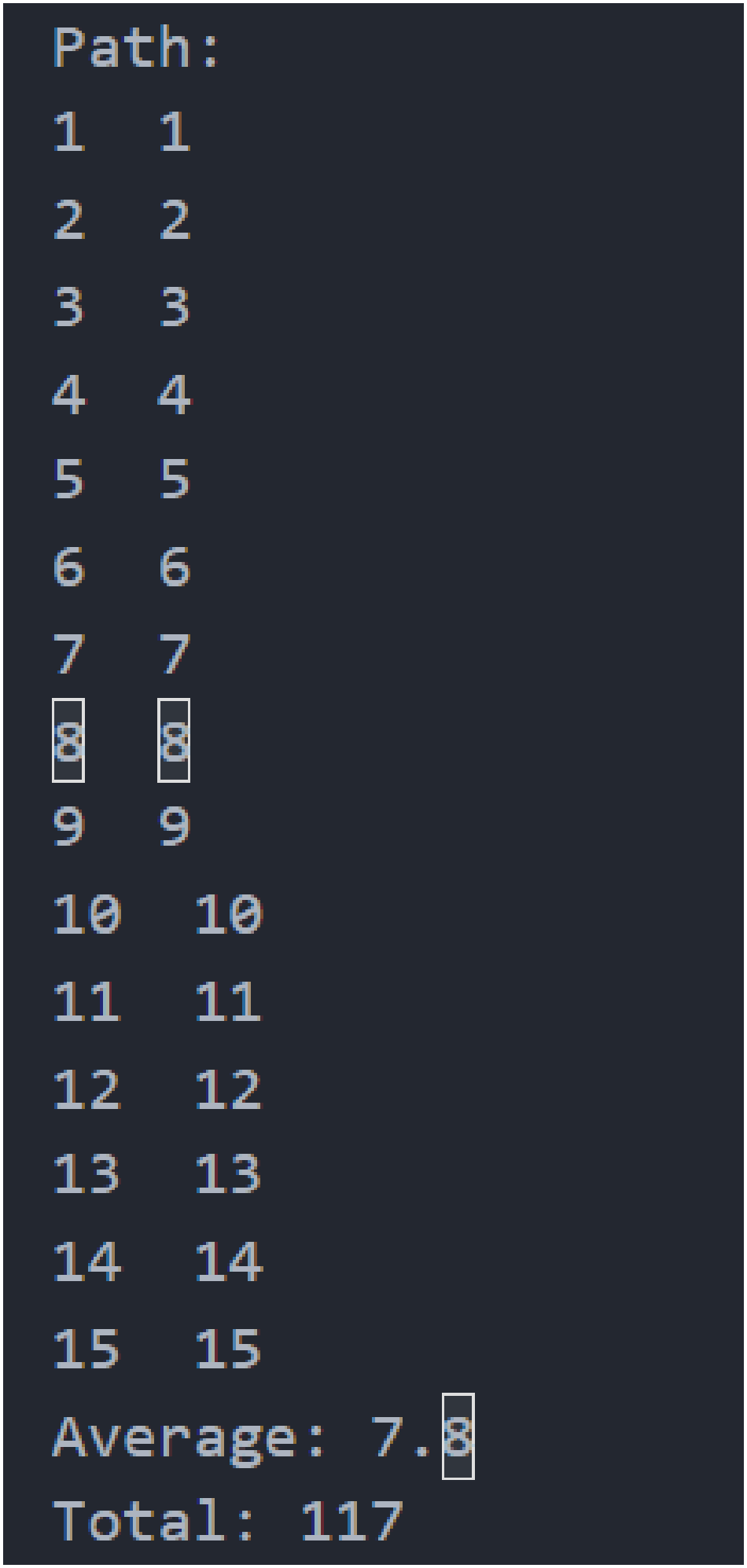

Figure 6: DTW4

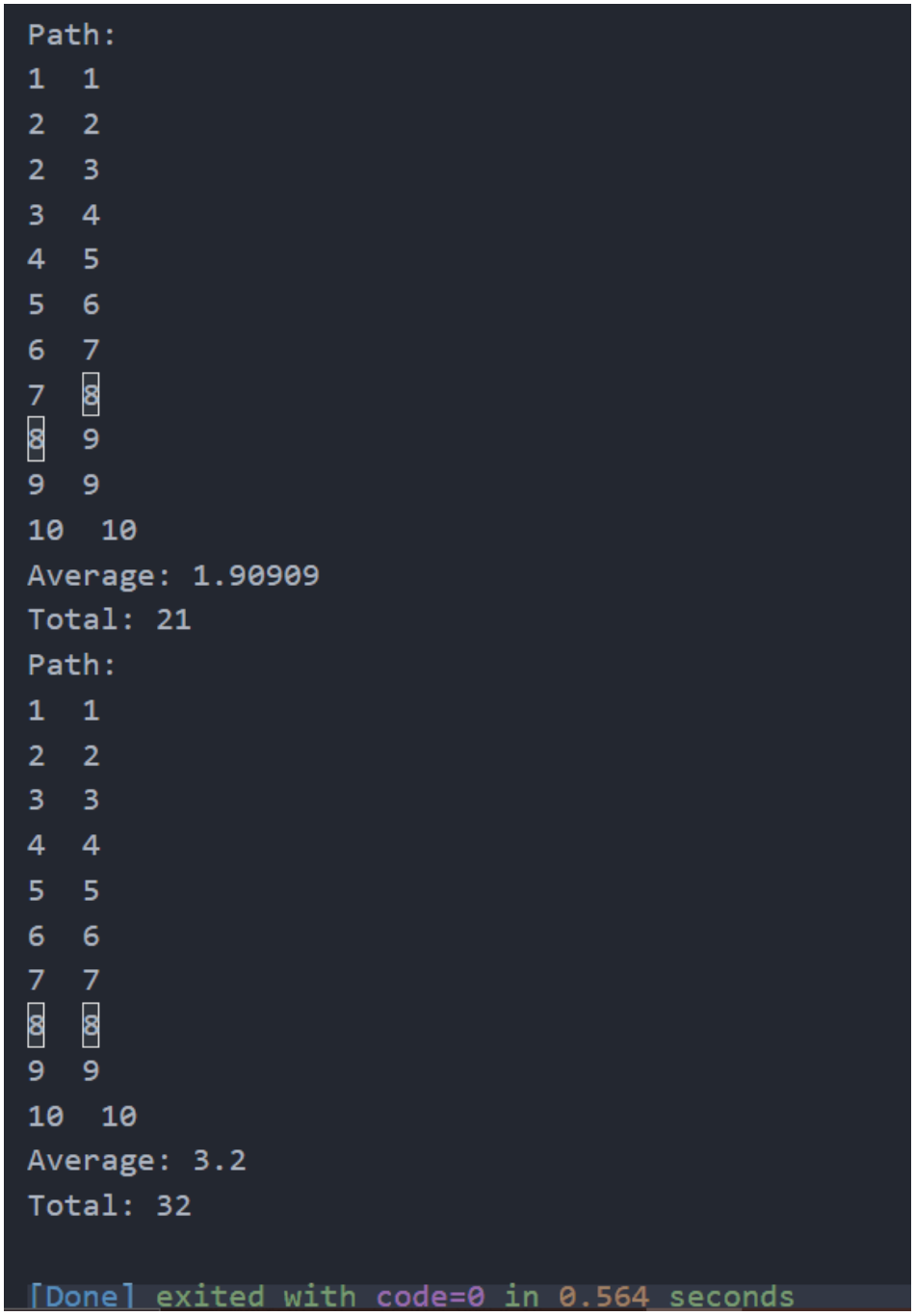

Figure 7: DTW5