Lab05-DynamicProgramming

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

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name: Zilong Li Student ID: 518070910095 Email: logcreative-lzl@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.

   (a) **Proof.** □

   (b) **Solution.** □

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 $\bar{X} = (x_1 \ldots x_n)$ and $\bar{Y} = (y_1 \ldots y_m)$, 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. (a) The implementation code.

Listing 1: Code-DTW.cpp

```cpp
#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 two series to be compared.
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 traversing to bottom left. The traversal path is identified based on the neighbor with minimum value.
i.e., When we reach the point (i, j) in the matrix, the next position is to choose the 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 the selected points.
*/
using namespace std;
double distance(vector<int> x, vector<int> y) {
    int n = x.size();
    int m = y.size();
    vector<vector<int>> DTW(n, vector<int>(m, -1));
    // Use the given state transition function to fill in the cost matrix. -- diagonal calculation
    // Clear the left and bottom border first.
    DTW[0][0] = abs(x[0] - y[0]);
    for (int i = 1; i < n; ++i)
        DTW[i][0] = abs(x[i] - y[0]) + DTW[i - 1][0];
    for (int j = 1; j < m; ++j)
        DTW[0][j] = abs(x[0] - y[j]) + DTW[0][j - 1];
    // traverse diagonally as a snake
    int i = 1, j = 1;
    int dir = 1;
    bool flag = false;
    while (true) {
        DTW[i][j] = abs(x[i] - y[j]) + min(min(DTW[i][j - 1], DTW[i - 1][j]), DTW[i - 1][j - 1]);
        if (!flag && (j == 1 || j == m - 1)) {
            if (i + 1 == n) ++j;
            else ++i;
            dir *= -1; flag = true;
        }
        else if (!flag && (i == 1 || i == n - 1)) {
            if (j + 1 == m) ++i;
            else ++j;
            dir *= -1; flag = true;
        }
        else {
            if (i == n - 1 && j == m - 1) break;
            i = i + dir; j = j - dir; flag = false;
        }
    }

    vector<int> d;
    //Identify the warping path. (n - 1,m - 1) -> (0,0)
    i = n - 1; j = m - 1;
    while (i >= 0 && j >= 0) {
        d.push_back(DTW[i][j]);

        if (i == 0) { j = j - 1; continue; }
        else if (j == 0) { i = i - 1; continue; }

        if (DTW[i - 1][j - 1] <= DTW[i][j - 1]) {
            if (DTW[i - 1][j - 1] <= DTW[i - 1][j]) { i = i - 1; j = j - 1; }
            else i = i - 1;
        }
        else if (DTW[i][j - 1] <= DTW[i - 1][j]) j = j - 1;
        else i = i - 1;
```

```
    }

    double ans = 0;
    //Calculate th time normalized distance
    for (auto di : d) ans += di;
    return ans / d.size();
}

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 distance function has thus three
        inputs: X, Y, and the size of window w.
    return 0;
}
```

The result is:

```
0
8.66667
```

The time complexity of every step is:

i. Initalization is regraded as $O(1)$.

ii. Clearing the border costs $O(m+n)$, and traversing in a diagonal way costs $O(mn)$.

iii. Finding the path costs $O(m+n)$.

iv. Calculating the average costs $O(m+n)$.

So, it is of $O(mn)$ time complexity.

□

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.