Lab05-DynamicProgramming

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

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name:Beichen Yu   Student ID:519030910245   Email: polarisybc@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) If there is a subtree T'' has a smaller expected search cost than T'. Then we can delete T' from T, and using T'' to substitute. Then we get a new binary search tree, which has a smaller expected search cost than T. So we got the contradiction, which means T' is the optimum solution.

   (b) If $j < i - 1$, there is nothing the subtree.

   If $j = i - 1$, there is only a dummy key d_{i-1} inside the subtree.

   If $j > i - 1$, we can find the root node k_r of the subtree with nodes $k_i, k_{i+1}, \cdots, k_{j-1}, k_j$. The cost of the left subtree of k_r is $e[i, r-1]$, and the cost of its right subtree is $e[r+1, j]$. If we link a subtree contains $k_i, k_{i+1}, \cdots, k_j$ to a root node, the depth of each node in the subtree adds 1, which means the total cost adds $w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$. So we have $e[i, j] = p_r + (e[i, r-1] + w(i, r-1)) + (e[r+1, j] + w(r+1, j))$. Because $w(i, r-1) + p_r + w(r+1, j) = w(i, j)$, we can conclude that $e[i, j] = e[i, r-1] + e[r+1, j] + w(i, j)$.

   So the state transition equation is:

$$e[i,j] = \begin{cases} q_{i-1}, & j = i - 1; \\ \min_{i \leqslant r \leqslant j}\{e[i, r - 1] + e[r + 1, j] + w(i, j)\}, & i \leqslant j. \end{cases}$$

To reduce complexity, we can use $w[i, j]$ to store the value of $w(i, j)$. When $j \geqslant i$ we have $w[i, j] = w[i, j - 1] + p_j + q_j$.

Next we propose our algorithm for optimal binary search tree:
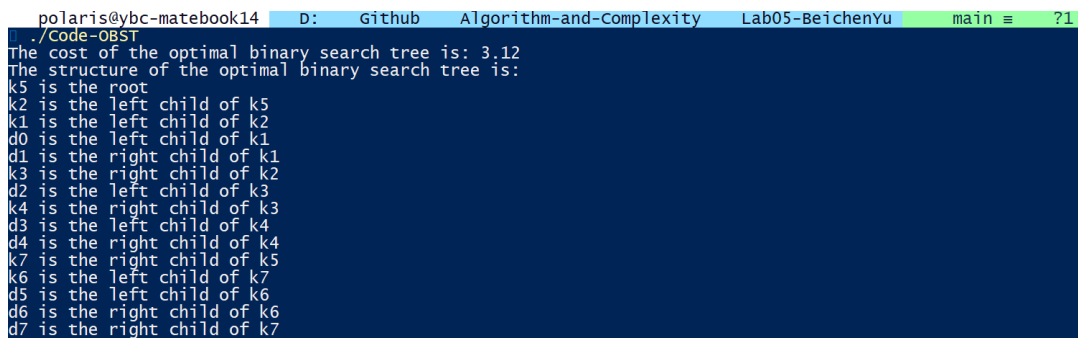

Algorithm 1: Optimal-BST

---
1 Initialize two-dimensional arrays e, w and $root$;
2 **for** $i = 1$ **to** $n + 1$ **do**
3 $\quad$ $e[i][i - 1] \leftarrow q_{i-1}$;
4 $\quad$ $w[i][i - 1] \leftarrow q_{i-1}$;
5 **for** $l = 1$ **to** n **do**
6 $\quad$ **for** $i = 1$ **to** $n - l + 1$ **do**
7 $\quad\quad$ $j \leftarrow i + l - 1$;
8 $\quad\quad$ $e[i][j] \leftarrow \infty$;
9 $\quad\quad$ $w[i][j] \leftarrow w[i][j - 1] + p_j + q_j$;
10 $\quad\quad$ **for** $r = 1$ **to** j **do**
11 $\quad\quad\quad$ $t \leftarrow e[i, r - 1] + e[r + 1, j] + w[i, j]$;
12 $\quad\quad\quad$ **if** $t < e[i][j]$ **then**
13 $\quad\quad\quad\quad$ $e[i][j] \leftarrow t$;
14 $\quad\quad\quad\quad$ $root[i][j] \leftarrow r$;

15 **return** $e, root$;

(c) The code is included in the .zip file.

In the first for-loop, the time complexity is $O(n)$. In the second one, there are there for-loops inside. The time complexity is $O(1^2 + 2^2 + \cdots + n^2) = O(n^3)$. So the total time complexity is $O(n^3)$.

This is the result:

Figure 1: The result of the OBST

(d) I draw this optimal binary search tree with powerpoint. The procress of drawing it is easy, just according to the output of the program and draw from the root to leaves.
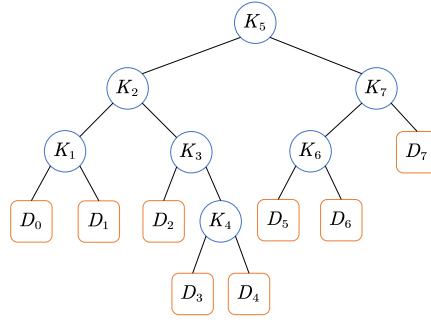
Figure 2: The optimal binary search tree

☐

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 $\bar{X} = (x_1 \ldots x_n)$ 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 $|i - j| \le w$. 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 code is included in the .zip file. In the first several loops, there are only one for-loop, so the time complexity of these parts is $O(max\{m, n\})$. And in the biggest while-loop, there are $i = n - 1$ and $j = m - 1$. Each time i or j or both of them minus one, so in this part the time complexity is $O(mn)$. So the finial time complexity is $O(mn)$.

This is the result:

Figure 3: The result of the DTW

(b) The code is included in the .zip file.

This is the result:

Figure 4: The result of the DTW with constrain

□

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.