Lab05-DynamicProgramming

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

$*$ If there is any problem, please contact TA Haolin Zhou.

$*$ Name: Wendi Chen   Student ID: 519021910071   Email: chenwendi-andy@sjtu.edu.cn

1. *Optimal Binary Search Tree.* Given a sorted sequence $K = \langle k_1, k_2, \dots, k_n \rangle$ 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* $d_0, d_1, d_2, \dots, d_n$ 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 $i = 1, 2, \dots, n-1$, 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 $k_i, \dots, k_j$, then this subtree T' must be optimal as well for the subproblem with keys $k_i, \dots, k_j$ and dummy keys $d_{i-1}, \dots, d_j$.

   (b) We define $e[i, j]$ as the expected cost of searching an optimal binary search tree containing the keys $k_i, \dots, k_j$. 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) Actually, because every k_i is an internal node and d_i is a leaf, when $k_i, \dots, k_j$ is determined, then $d_{i-1}, \dots, d_j$ is determined. Thus, we just have to consider $k_i, \dots, k_j$. Now we prove the optimal substructure property by contradiction. If there is a subtree T'' containing keys $k_i, \dots, k_j$ which has smaller expected search cost than T', we can just remove T' from T and replace it with T''. The new binary search tree will have smaller expected search cost then T, which contradicts. So this problem has the optimal substructure property.

(b) Now that we have proved the optimal substructure property, we can use it to construct the state transition equation. For an optimal binary search tree containing $k_i, \dots, k_j$, it must be composed of a root node $k_r (i \le r \le j)$ and an optimal subtree containing $k_i, \dots_{r-1}$ and another optimal subtree containing $k_{r+1}, \dots, k_j$. Then we just need to enumerate k_r to get the whole optimal tree. Note that if $r = i$ or $r = j$, then the left subtree or the right subtree will only contain d_{i-1} or d_j.
Then, we'll derive the state transition equation by cases.

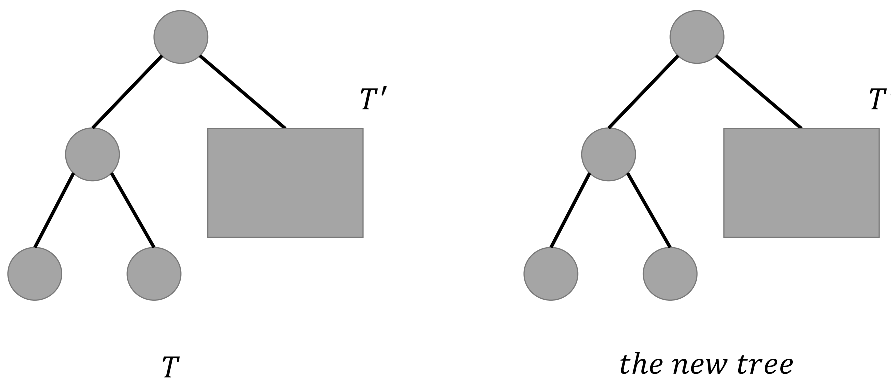


Figure 1: Proof for Optimal Substructure

i. If $j = r - 1$, then $e(i, j) = q_{i-1}$.

ii. If $j \geq i$, we can find that the expected search cost of the left subtree and right subtree will increase by $w(i, r-1)$ and $w(r+1, j)$ respectively. By enumeration, we get the state transition equation

$$e[i,j] = min_{i \leq r \leq j}\{p_r + e[i, r-1] + w(i, r-1) + e[r+1, j] + w(r+1, j)\}$$

In fact, we have $w(i,j) = p_r + w(i, r-1) + w(r+1, j)$, then we have

$$e[i,j] = min_{i \leq r \leq j}\{e[i, r-1] + e[r+1, j] + w(i,j)\}$$

Therefore, we have already derived the state transition equation

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

In order to optimise time efficiency, we definitely need to store sub-results. Then two 2-D arrays $e[1 \ldots n+1, 0 \ldots n]$ and $w[1 \ldots n+1, 0 \ldots n]$ are both needed. Also, we need $r[1 \ldots n, 1 \ldots n]$to store the root node of the current tree so that we can draw the tree after we find the final answer.
There still exists one problem——in which order should we compute $e[1 \ldots n+1, 0 \ldots n]$.
A natural idea is to compute intervals with shorter length first.
Thus, we can write the pseudocode (Alg.1).

(c) Please refer to *Code-OBST.cpp.*
Now let's analyze the time complexity. The first loop obviously has a time complexity of $\Theta(n)$. The inner part of the 3-layer loop cost $\Theta(1)$. Thus, the time complexity of the 3-layer loop is

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} \sum_{j=1}^{n-i+1} \sum_{k=j}^{j+i-1} \Theta(1) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n-i+1} \Theta(i) \\ &= \sum_{i=1}^{n} \Theta(i(n-i+1)) = \sum_{i=1}^{n} \Theta(in - i^2 + i) \\ &= \Theta(\frac{n^2(n+1)}{2} - \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}) = \Theta(n^3) \end{aligned}$$

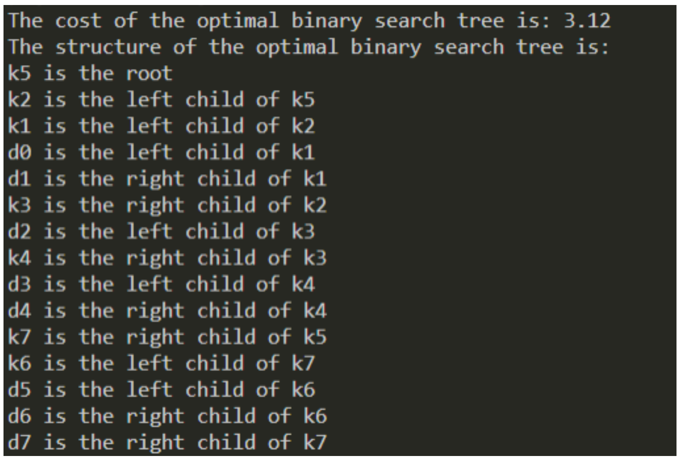
Thus, the time complexity is $\Theta(n^3)$.
The minimum search cost calculated by the algorithm is 3.12.

Algorithm 1: Find the Optimal Binary Search Tree through Dynamic Programming

```
Input: Two arrays of probability $p[1 \dots n], q[0 \dots n]$
Output: Two 2-D arrays $e[1 \dots n+1, 0 \dots n]$ and $r[1 \dots n, 1 \dots n]$,
        which imply the expected search cost and root of the subtree
        containing $k_i, \dots, k_j$

$e[1 \dots n+1, 0 \dots n] \leftarrow [0 \dots 0, 0 \dots 0]$;
$w[1 \dots n+1, 0 \dots n] \leftarrow [0 \dots 0, 0 \dots 0]$;
$r[1 \dots n, 1 \dots n] \leftarrow [0 \dots 0, 0 \dots 0]$;
for $i = 1$ to $n+1$ do
    $e[i, i-1] = q_{i-1}$;
    $w[i, i-1] = q_{i-1}$;
for $len = 1$ to $n$ do
    for $i = 1$ to $n - len + 1$ do
        $j = i + len - 1$;
        $w[i, j] = w[i, j-1] + p_j + q_j$;
        $e[i, j] = \infty$;
        for $root = i$ to $j$ do
            $temp = w[i, j] + e[i, root-1] + e[root+1, j]$;
            if $temp < e[i,j]$ then
                $e[i, j] = temp$;
                $r[i, j] = root$;
return $e$ and $r$;
```

(d) Because we have already recorded the root of every optimal subtree, we can just do a preorder traversal through the tree. We start with $r[1][n]$, and for each time, we can find a new root $r[i][j]$ and divide the original tree into two subtrees. We repeat these steps until $r[i][j] = i$ or $r[i][j] = j$, which mean we arrive the lowest internal nodes and we can just add "leaves"(d_{i-1} or d_j) to it. Then, we can draw the structure of the OBST.



```
The cost of the optimal binary search tree is: 3.12
The structure of the optimal binary search tree is:
k5 is the root
k2 is the left child of k5
k1 is the left child of k2
d0 is the left child of k1
d1 is the right child of k1
k3 is the right child of k2
d2 is the left child of k3
k4 is the right child of k3
d3 is the left child of k4
d4 is the right child of k4
k7 is the right child of k5
k6 is the left child of k7
d5 is the left child of k6
d6 is the right child of k6
d7 is the right child of k7
```

Figure 2: Structure of OBST Generated by the Program

□

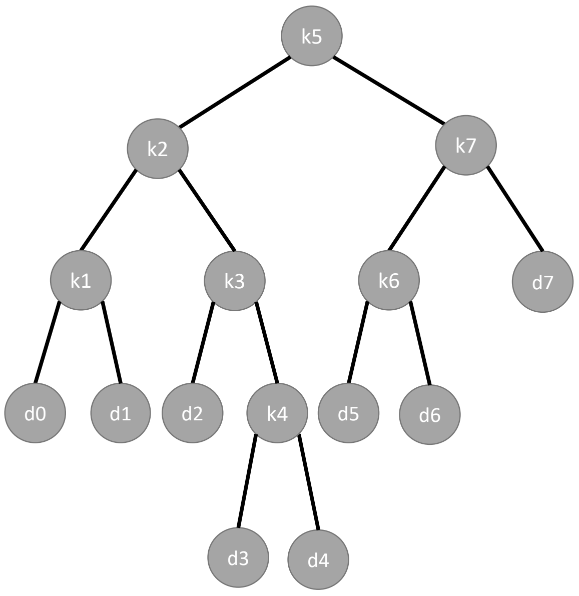


Figure 3: Structure of OBST

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:

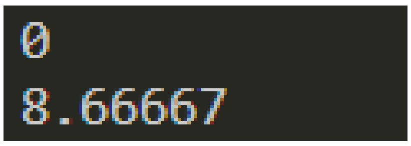
$$DTW(i, j) = |x_i - y_j| + \min(DTW(i, j - 1), DTW(i - 1, j), DTW(i - 1, j - 1))$$

   (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.

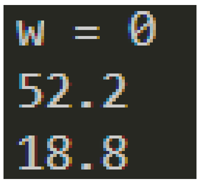
   (a) Please refer to *Code-DTW.cpp*.
   The result are shown as below.



```
0
8.66667
```

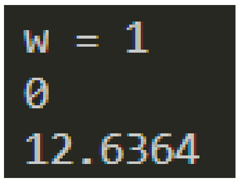
Figure 4: DTW Result Without Window Constraint

Let us analyze the time complexity. In fact, for the whole process of filling the cost matrix, we have a 2-layer loop, which has a time complexity of $\Theta(nm)$. For the process of generating the warping path, the time complexity is $O(n + m)$. Thus, the time complexity is $\Theta(nm)$.



```
w = 0
52.2
18.8
```

(a) w = 0



```
w = 1
0
12.6364
```

(b) w = 1

Figure 5: DTW Result With Window Constraint

(b) Please refer to *Code-DTW.cpp*.
To implement constraint window, we can check every time we access or revise $DTW[i][j]$.
The running result are shown in Figure 5.

□

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.