Name: Andrii Bezrukov **UČO:** 570943

0007 *list* 1 *učo* 570943 *body*

0 1 2 3 4 5 6 7 8 9

Suppose you have n keys $a_1 < a_2 < \ldots < a_n$ and you want to build a binary search tree that allows efficient search for these keys. We assume that the search time for a key is proportional to its depth (distance from root). If the search queries are distributed uniformly across the keys than the optimal solution is to build a balanced binary tree. However, if some of the search queries are more frequent then others then a balanced binary tree might not be the most efficient structure.
Given search frequencies of n keys, say $f_1, f_2, \ldots, f_n$, describe an efficient *dynamic programming* algorithm to compute the binary tree that minimizes the expected search time defined as $\sum_i f_i d_i$, where d_i is the depth of the node containing a_i. Provide the running time of your algorithm and justify the correctness of your solution.

---

1. Notation and Setup

- Let $OPT[i, j]$ be the minimum expected search cost for a BST containing keys $a_i, a_{i+1}, \ldots, a_j$ - Let $w[i, j] = \sum_{k=i}^{j} f_k$ be the sum of frequencies from a_i to a_j

2. Recurrence Relation

For a given subproblem $OPT[i, j]$: - We try each key a_r $(i \leq r \leq j)$ as the root - The left subtree contains keys $a_i, \ldots, a_{r-1}$ with cost $OPT[i, r-1]$ - The right subtree contains keys $a_{r+1}, \ldots, a_j$ with cost $OPT[r+1, j]$ - All nodes in this subtree have an additional cost of $w[i, j]$ because each node's depth increases by 1 when placed in a subtree
Thus, our recurrence relation is:

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

3. Base Cases

- $OPT[i, i-1] = 0$ (empty tree) - $OPT[i, i] = f_i$ (single node tree)

Name: Andrii Bezrukov **UČO:** 570943

0007 *list* 2 *učo* 570943 *body*

0123456789

4. Pseudocode

```
1  OptimalBSTa[1...n], f[1...n] Initialize OPT[1...n+1, 0...n] and w[1...n+1, 0...n] with zeros
   Initialize root[1...n, 1...n]
2  for i = 1 to n do
3      OPT[i, i] = f[i];
4      w[i, i] = f[i];
5  for i = 1 to n + 1 do
6      OPT[i, i − 1] = 0;
7      w[i, i − 1] = 0;
8  for len = 2 to n do
9      for i = 1 to n − len + 1 do
10         j = i + len − 1 OPT[i, j] = ∞ w[i, j] = w[i, j − 1] + f[j]
11         for r = i to j do
12             cost = OPT[i, r − 1] + OPT[r + 1, j] + w[i, j] if cost < OPT[i, j] then
13                 OPT[i, j] = cost;
14                 root[i, j] = r;
15 return OPT[1, n] and root table
```

5. Running Time Analysis

- Computing all $w[i, j]$ values: $O(n^2)$ time - We have $O(n^2)$ subproblems (all possible $[i, j]$ ranges) - Each subproblem requires trying $O(n)$ possible roots - Total time complexity: $O(n^3)$ - Space complexity: $O(n^2)$ for the tables

6. Matrix Traversal Strategy

The DP matrix (with indices i and j) is filled in **diagonal order** by increasing the length $l = j-i+1$ of the interval:

1. **Base Case (Length 1):** For all $i = 1, \ldots, n$, fill [e[i,i] = f_i and root[i,i] = i.] These correspond to the diagonal elements of the DP matrix.

2. **Intervals of Increasing Length:** For $l = 2$ to n, compute $e[i, j]$ for all subintervals $[i, j]$ satisfying $j = i + l - 1$. That is, [for i = 1, 2, …, n-l+1 set j = i+l-1.] Within this loop, we consider all possible roots r with $i \leq r \leq j$ and compute [cost = $\big(e[i, r-1]$ if $r > i\big) + \big(e[r+1, j]$ if $r < j\big) + w[i, j]$.] We select the r that minimizes the cost.

Filling the table diagonally guarantees that when we compute $e[i, j]$, the subproblems $e[i, r-1]$ and $e[r+1, j]$ (which represent shorter intervals) have already been computed. The dynamic programming matrix is traversed in order of increasing subinterval length. Diagonal entries (intervals of length 1) are computed first, and for each subsequent length, the algorithm ensures that the dependent subproblems have already been solved. This guarantees that when computing $e[i, j]$, the optimal values for $e[i, r-1]$ and $e[r+1, j]$ are available, allowing the correct computation of the cost using the recurrence.

Name: Andrii Bezrukov **UČO:** 570943

0007 *list* 3 *učo* 570943 *body*

0 1 2 3 4 5 6 7 8 9

7. Correctness Justification

The algorithm is correct because:
1. **Optimal Substructure Property**: If the optimal BST for keys a_i through a_j has root a_r, then the left subtree must be optimal for keys a_i through a_{r-1} and the right subtree must be optimal for keys a_{r+1} through a_j.
2. **Consideration of All Possibilities**: We try all possible keys as the root of each subtree and select the one that minimizes the total expected search time.
3. **Correct Accounting of Costs**: The cost formula correctly accounts for the depth of each node by adding $w[i, j]$ to the sum of costs of the two subtrees.
Therefore, our algorithm guarantees finding the binary search tree with the minimum expected search time, given the search frequencies of the keys.