Problem 8: Optimal Search Trees [HackerRank]
By Ruijie Fang

1 Background: Searching in Databases

You are working for a very large database company and they have a database index consisting of \( n \) sorted integer keys \( S := s_1, s_2, ..., s_n \) with \( s_i \leq s_{i+1} \) for \( 1 \leq i < n \), each mapping to some value. The \( n \) keys are some predetermined sparse, big integers, which you don’t know, meaning they are non-trivial to hash (if you don’t know what this means, don’t worry about it). Furthermore, you know that the company workload every day generates a fixed access pattern to the database, resulting in access frequencies \( F_1, F_2, ..., F_n \in \mathbb{Z} \).

You can of course use a dynamic search tree, like a splay tree, to answer this problem. However your boss wants you to be optimal: Let \( d(i) \) denote the depth of the \( i \)-th node in the array \( (S) \) in the search tree. Let the root node in the search tree have depth 0 (i.e. it can be accessed at no cost). An optimal search tree is a search tree that minimizes the cost function \( \sum_{i=1}^{n} F_i \cdot d(i) \). Intuitively, this means you want the most frequently accessed values to be toward the top, and least frequently accessed toward the bottom.

2 Definitions

**Binar Search Tree**: Given a set \( S := \{k_1, k_2, ..., k_n\} \) of \( n \) integer keys with \( k_1 \leq k_2 \leq ... \leq k_n \), a binary search tree \( T_2 \) is a tree of \( n \) binary search tree nodes. A binary search tree node can be represented as \((x, c_1, c_2)\) where \( x \in S \) is a key in \( S \) uniquely represented in \( T_2 \), \( c_1 \) is the leftmost child node of \( x \) and \( c_2 \) is the rightmost child node of \( x \). Let \( S(1, x) \) denote the set of keys in the subtree rooted by \( c_1 \), and let \( S(2, x) \) and \( S(3, x) \) be defined analogously. We impose the constraint that \( x \) has to be either greater than or equal to all keys in \( S(1, x) \) and \( S(2, x) \), or \( x \) is greater than or equal to all keys in \( S(2, x) \) and \( S(3, x) \).

**Ternary Search Tree**: Defined analogously. Given a set \( S := \{k_1, k_2, ..., k_n\} \) of \( n \) integer keys with \( k_1 \leq k_2 \leq ... \leq k_n \): A ternary search tree \( T_3 \) is a tree of \( n \) ternary search tree nodes. A ternary search tree node can be represented as \((x, c_1, c_2, c_3)\) where \( x \in S \) is a key in \( S \) uniquely represented in \( T_3 \), \( c_1 \) is the leftmost child node of \( x \), \( c_2 \) is the middle child node of \( x \), and \( c_3 \) is the rightmost node of \( x \). Let \( S(1, x) \) denote the set of keys in the subtree rooted by \( c_1 \), and let \( S(2, x) \) and \( S(3, x) \) be defined analogously. We impose the constraint that \( x \) has to be either greater than or equal to all keys in \( S(1, x) \) and \( S(2, x) \) and less than or equal to all keys in \( S(3, x) \), or \( x \) is greater than or equal to keys in \( S(1, x) \) and \( S(3, x) \) and less than or equal to all keys in \( S(2, x) \) and \( S(3, x) \).
**Check.** Why does accessing a key in a well-balanced ternary search tree take $O(\log_3 n)$ comparisons?

## 3 Problem Statements

Given $n$ keys and associated access frequencies $F_1, F_2, \ldots, F_n$, write a program to find out either an optimal binary search tree in $O(n^2)$-time, or an optimal ternary search tree in $O(n^3)$-time. There are two subproblems; solving any part gets you the 15 points of credit.

1. Given $\overrightarrow{F}$ and $n$ keys $k_1, k_2, \ldots, k_n \in \mathbb{Z}$ sorted in increasing order, write a program to optimal cost of an optimal binary search tree, in $O(n^2)$-time.
   - Input constraint: $1 \leq n \leq 2000$. Use 64-bit signed integer to for your solution so that in case of overflow the behavior will be the same.
   - Running time limit: 1 second.
   - Hint: An $O(n^3)$ solution will not work.

2. Given $\overrightarrow{F}$ and $n$ keys $k_1, k_2, \ldots, k_n \in \mathbb{Z}$ sorted in increasing order, write a program that outputs the optimal cost of an optimal ternary search tree, in $O(n^3)$-time.
   - Input constraint: $1 \leq n \leq 300$. Use 64-bit signed integer to for your solution so that in case of overflow the behavior will be the same.
   - Running time limit: 1 second.
   - Hint: An $O(n^4)$ solution will not work.

3. Extra credit opportunity: Email coscon.written.submission@gmail.com and attach a proof of the following math problem. This can potentially get you an additional 3 points max.\footnote{If you have no idea what the inequality about $g$ is, maybe solving (2) will be a little more accessible to you. This is strictly not necessarily for solving any part of this problem.}

A cute math problem. We say a function $f : \mathbb{R} \rightarrow \mathbb{R}$ is convex if $\frac{1}{2}(f(x) + f(y)) \leq f(\frac{x+y}{2})$. Let $g : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ be defined as $g(a,b) = f(b - a)$. Show that $g(a,c) + g(b,d) \geq g(b,c) + g(a,d) \iff f$ is convex, for $a \leq b \leq c \leq d$.

Concrete input format descriptions start on the next page.
Part 1 Input/Output Format: Everything is on a single line. The first number read in will be $n$, the number of nodes. The next $n$ space-separated integers represent $F_1, \ldots, F_n$, respectively. Your program should output a single number represent the optimal cost of a BST on these nodes and frequencies.

Sample Input for Part 1:
6 1 4 8 3 6 5

Sample Output for Part 1:
28

Part 2 Input/Output Format: Everything is on a single line. The first number read in will be $n$, the number of nodes. The next $n$ space-separated integers represent $F_1, \ldots, F_n$, respectively. Your program should output a single number represent the optimal cost of a TST on these nodes and frequencies.

First Sample Input for Part 2:
6 1 4 8 3 6 5

First Sample Output for Part 2:
23

Second Sample Input for Part 2:
4 3 8 4 5

Second Sample Output for Part 2:
12

Third Sample Input for Part 2:
4 1 3 4 8

Third Sample Output for Part 2:
9