1 Analysis

Optimal binary tree is a binary tree with the minimun search cost, where as the parent cost is the sum of itself and the children node. For stability, after obtaining the keys and probabilities, the data is sorted relatively to the keys from smallest to largest using quicksort. Then three main array is create, *root*, e, and w. The array *root* is a 2D table with the size of N number of elements, storing the optimal root relative to the elements in the subtree. The array e is also a 2D table with the size of N+1 number of elements, storing the optimal cost for building the subtree. The array w is similar to array e, but it is storing the acumulative sum of the probabilities or the weights instead. For indexing, i and j are the row and column respectively.
The algorithm begin with diagonal line of the w and e initialized to 0 since the cost of binary tree with length $l = 0$ is 0 ($l = j - i$). The first for loop iterates through tree with $l = 1$ to N. The second for loop (first inner loop) iterates through all the possible optimal solutions relative to l. It also initialize the $e_{i,j}$ to very large number and the $w_{i,j}$ to $w_{i,j-1} + prob_j$. The third for loop (inner most loop) calculate the optimal root cost using prior optimal solution. To find the minimun cost for $e_{i,j}$, we take the minimun of the sum of a possible $l - 1$ cost and the cost of the new root. So, $e_{i,j} = min\{e_{i,r} + e_{r+1,j} + w_{i,j} : r = i \text{ to } j\}$. Basically, the sum of the optimal cost of the left subtree $e_{i,r}$ and right subtree $e_{r+1,j}$ and the total cost to add new root. For example, let node $N2$, $N1$, and $N3$ be the root, left child, and right child respectively, then the total cost is $cost = N2 + 2 * N1 + 2 * N3 = N1_{(e_{i,r})} + N3_{(e_{r+1,j})} + (N1 + N2 + N3)_{(w_{i,j})}$. After updating the optimal cost, the optimal root for tree size of l is stored into the root array. This algorithm of the finding the optimal cost is $O(n^3)$.

The root array can be used to build inorder or postorder traversal tree. We can do this recursively. Let l and u be the range for all element of the subtree, so $root_{l,u}$ is the optimal root for a subtree consist of all elements between l and u. For convience, the optimal root is a index m. So, the optimal root of left subtree is $root_{l,m-1}$ and right subtree is $root_{m+1,u}$. To build the final tree using inorder traversal, the optimal root inserts into array only after the recursive call to left subtree is done. Similarly, postorder traversal inserts optimal root after both left and right subtree is done. This take $O(n)$.

2 Results

The program results are store in the results folder. All of the result are correct and what I have expected. The only thing should be noted is that the space character is converted to 0.

3 Source Code

I use a standard practice to organize my work environment. All of the header files are stored include folder and source files are stored in src folder, which include the optimal binary search tree and quick sort classes. The example input files are stored in the examples folder. The root folder includes the main.cpp, Makefile, and the executable. The program takes in the a file. It reads from the file and insert new keys and probabilities into vectors while also keep track of the number of element. The program outputs the result in a file in the results folder. The result file includes the original data sorted, the cost matrix (E matrix), root matrix, and inorder and postorder traversal of the final tree. To compile the program, simply run "make" and the executable "optimal_bst" is created.