CS69001 Computing Laboratory – I

Practice Exercises: Set 2

---

1. Write a program to recognize balanced strings with the delimiters `(){}[]`. For example, the string `{(()[])}([][][]()){}` is balanced. Two strings that are not balanced are `{(()[])}([][][]()]{}` and `{(()[])}([][][](){}`. Use a stack.

2. You are given integers n, k with $0 \leqslant k \leqslant n$. Your need to print all the subsets of size k of the set $\{1, 2, 3, \ldots, n\}$. Use a FIFO queue in the following fashion. Insert the empty set $\emptyset$ to an initially empty queue Q. Repeat so long as Q is not empty: Let S be the element (a set) at the front of Q. Dequeue S from Q. Let the size of S be l, and let the maximum element of S be m (take $m = 0$ if $S = \emptyset$). If $l = k$, print S. Otherwise (that is, if $l < k$), enqueue the $(l+1)$-element sets $S \cup \{i\}$ in Q for $i = m+1, m+2, \ldots, n$.

3. You are given a min-heap H of integers in the contiguous representation. Write a function that takes H and an integer B as input, and prints all the elements of the heap that are $\leqslant B$. The function must not modify H. The running time of your function should be $O(k)$, where k is the number of integers printed.

4. You are given k sorted arrays $A_1, A_2, \ldots, A_k$ with a total of n elements. You are required to build an array B of size n by merging the k input arrays. You are allowed to use only $O(k)$ additional space (in addition to the arrays A_i and B). Your program should run in $O(n \log k)$ time. Use a min-priority queue to implement your algorithm. (**Hint:** At any point of time, the priority queue should store at most one element from each A_i.)

5. **(a)** A random binary tree T can be constructed as follows. Let n be the number of nodes in T. Randomly generate the numbers n_l and n_r of nodes in the left and right subtrees (so that $n = 1 + n_l + n_r$). Recursively build the left and the right subtrees, whichever is/are non-empty. Store random keys at the nodes.

(b) Let r be the root and v a leaf node in T. There is a unique r, v path $r = u_0, u_1, u_2, \ldots, u_l = v$ in T. Let the key stored at node u_i be k_i. The alternating sum of these key values is

$$altsum(v) = k_0 - k_1 + k_2 - + \cdots + (-1)^l k_l.$$

Write a function to print the alternating sums at all the leaf nodes in T.

6. Let T be a general rooted tree, in which each node can have any number of children.

(a) Let v be a node in T with c child nodes. In addition to a key, v stores an array of c child pointers (the count c should also be stored at v). The child pointers point to the c subtrees of v. Build a random general rooted tree using a constructor similar to that for binary trees in the last exercise.

(b) Write a function to compute the height of T.

(c) Write a function that, given the general tree T, prepares and returns a binary tree B which is the first-child-next-sibling representation of T.

(d) Write a function to compute the height of T if B is supplied as the only input.

7. Write an $O(h(T))$-time function to compute the immediate successor of a node in a BST T.

8. A three-way search tree (3ST) is a rooted tree with each node storing two keys and three child pointers L, M, R. Let v be a node in the 3ST storing the keys k_1, k_2. Let l (resp. m, r) be a key value stored in the left (resp. middle, right) subtree. We must have $l < k_1 < m < k_2 < r$.

(a) Write a function to search for a key in a 3ST.

(b) Write a function to insert a key in a 3ST.

9. **(a)** Write the insert function for binary search trees. Prepare a BST T by inserting random keys to an initially empty tree. Let n be the number of nodes in T after all these insertions.

(b) Convert T to a BST of height $n-1$ as follows. So long as the root has a left child, make a right rotation at the root. When the left subtree of the root becomes empty, move to the right child of the root, and repeat the process. (**Remark:** This is first stage of the Day–Stout–Warren (DSW) BST rebalancing algorithm.)

10. Write a function that, upon input h, returns an AVL tree of height h and with the minimum possible number of nodes. (These are called Fibonacci trees.)

Implement all the data structures yourself. Do not use STL data types and library calls.