1 Abstract Data Types & Data Structures

For each of the following problems, choose ONE ADT and ONE data structure to solve the problem and explain why your choice works better than the other options:

(a) Determine whether a string composed of '(', ')', '{', '}', '[', ']' are valid For example, "[()]" and "[{}]{()}" are valid but "[(])" and "{]" are not valid.

Abstract Data Type: ⃝ Stack  ○ Queue
Data Structure: ⃝ Array  ○ LinkedList with front  ○ LinkedList with front and back

One Sentence Explanation:

We’d use the Stack ADT because the most recent opening “thing” needs the matched closing first. Since we only push and pop on one end of the stack, it’s unnecessary to have back pointer. Asymptotically, Array and LinkedList with front are both good choices, but cache locality will likely be a problem with the LinkedList (arrays are contiguous in memory, but linked lists are stored using arbitrary pointers.)

We also accept LinkedList with front as an answer because the idea of cache locality is presented in CSE 351, but not this class. Note that, we would prefer LinkedList in the scenario where we don’t know the number of items, N.

(b) Cars drive onto a toll bridge from one end and exit from the other end of the bridge. The number of cars is unpredictable. Determine the next car to exit from the bridge anytime if necessary.

Abstract Data Type: ○ Stack  ⃝ Queue
Data Structure: ○ Array  ○ LinkedList with front  ⃝ LinkedList with front and back

One Sentence Explanation:

We’d use the Queue ADT because of the first-in-first-out behavior. Since it’s a queue, LinkedList with front could work but slower than other two (think about add and remove). Also, because we cannot predict the number of cars, Array might need to resize itself. So, LinkedList with front and back is the best option here.

(c) Given runners’ names in 1st, 2nd, ..., nth places, announce awards in the reversed rank (e.g., 1st place is announced the last). The announcer also needs to be able to learn the name of some specific runners beforehand.

Abstract Data Type: ⃝ Stack  ○ Queue
Data Structure: ⃝ Array  ○ LinkedList with front  ○ LinkedList with front and back

One Sentence Explanation:

We’d use the Stack ADT because it can help us reverse the input. If we use LinkedList with front, we might need to traverse almost the entire list to get the specific runner. Instead, the Array implementation gives us efficiency of finding the runner using array indexing, which is constant time.

Will you want to pick up your worksheet later? Circle one: Yes / No