Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC
  - Predecessor and Successor
  - Insertion Sort
  - Link a List
  - The United Sums of Arrays
  - Bubble Sort
  - Pretty Queues Revisited
  - Just About Sorted
  - Brick Sort
  - All My Descendants
  - Mr C likes a Majority
  - Cocktail Sort
  - All My Descendants - Part II

Pretty Queues Revisited

LAB-PRAC-14_SORT-MISC

Pretty Queues Revisited [20 marks]

Problem Statement
We have implemented queues in the previous weeks and used them to schedule malloc and free requests. In this problem, we will re-implement queues, but using linked lists. A queue is, as you may recall, a data structure which allows entry at the end and exit at the beginning (you may see Week 10 Tuesday question 1 problem statement from the practice problems in the folder LAB-PRAC-10_MAT-FUN if you have forgotten or have not written down details of queues in your notebook).

The queue is a FIFO data structure since the first element to enter the queue is also the first one to exit the queue. We will implement a queue using a linked lists where elements will be inserted (i.e. enqueued) at the tail of the linked list and removed (i.e dequeued) from the head of the linked list.

The first line of the input will give you n, a strictly positive integer. The next line will give you n integers, separated by a space. These are the occupants of a linked list we are going to construct. However, the linked list does not contain these elements in this order. Store these n integers in an array arr.

After this, in the next line, we will give you the index of the head of the linked list in the array arr. After this, there will be n-1 more lines which will contain pairs of numbers of the form a b (i.e. the two numbers will be separated by a space). This will indicate that the element at index a in the array arr points to the element at index b in the array arr. Create a linked list with n nodes with the elements in the given order. Please refer to the hints to see an easy way to do so. After this, on the next line, we will give you a strictly positive number t, indicating the number of operations we wish to perform on this queue. In the next t lines we will give you pairs of integers which are to be interpreted as follows

1. If the line contains 1 x where x is an integer (negative, positive or zero), insert the element at the end of the queue, i.e. insert x at the tail of the linked list and then print the updated queue (i.e. the updated linked list) on a new line
2. If the line contains 2 0, then dequeue the element at the front of the queue i.e. delete the head of the linked list. Print the value of the deleted element on a new line and then print the update queue (i.e. the updated linked list) on a new line.

In your output, in the first line, print the linked list by printing elements from the head to tail, two elements separated by a single space. There should be no spaces at the end of the line. The, for every operation we ask, perform that operation and print the required outputs. For insertion operations, simply print the updated linked list on a new line. For deletion operations, first print the deleted element on a new line and then print the updated linked list on a new line.

Caution

1. We assure you that n + t will be at most 100.
2. We assure you that we will never as you to dequeue from an empty queue, i.e. delete an element from an empty linked list. We additionally promise you that the queue will never become empty so that you do not have to worry about printing empty lists.
3. n may be any strictly positive integer, even 1 in which the linked list will have a single element.

4. Elements may repeat in the initial list we give you, as well as the same integer may get enqueued again in the operations. An integer that has been dequeued (from the head) may be asked to be enqueued again (at the tail) in a subsequent operation.
5. The integers in the list, as well as those being inserted (enqueued) may be positive, negative or zero.
6. There should be no extra spaces at the end of any of the lines.

HINTS: Following these steps might make your life easier:

1. Create a structure node to store an integer and a pointer.
2. Make an array of 100 nodes to store the n integers as given as well as the pointers to the next element. This array will also store any new elements that are enqueued into the queue. As we have promised, n + t will always be less than or equal to 100.
3. Store the location of the head node as well as the tail node. It will help you maintain the queue easily. You may maintain location of the head and the tail nodes as global variables to help make your code easier. However, it is not very good programming practice.
4. For each pair (a b) given as input, make the node at index a in the array point to the node at index b in the array
5. Write a function to print a linked list.
6. If asked to insert an element into the list, the tail will get updated since elements get enqueued at the tail of a queue. You may store the newly inserted elements in the array itself. Since n + t is never more than 100, you will never run out of space.
7. If asked to delete an element , the head will get updated since elements get dequeued from the head of a queue.

Note that we are asking you to use a static data structure like an array, to store a linked list, only to make this problem less complicated. What you have created is not a very efficient dynamic data structure.

EXAMPLE:
INPUT
5
1 2 3 4 5
2
1 0
3 4
0 3
2 1
3
1 6
1 7
2 0

OUTPUT:
3 2 1 4 5
3 2 1 4 5 6
3 2 1 4 5 6 7
3
2 1 4 5 6 7

Explanation: The head is index 2 in the array i.e. the element 3. The links in the linked list are shown below
2 => 1
4 => 5
1 => 4
3 => 2

Thus, the linked list is 3 => 2 => 1 => 4 => 5 with 3 as the head and 5 as the tail. Three operations are to be performed on the linked list.

1. First we enqueue 6 at the tail of the linked list. The list looks like 3 => 2 => 1 => 4 => 5 => 6 after this step.
2. We then enqueue 7 at the tail of the linked list. The list looks like 3 => 2 => 1 => 4 => 5 => 6 => 7 after this step.
3. We then dequeue an element from the head of the linked list. That element is 3 and the new list is 2 => 1 => 4 => 5 => 6 => 7 after this step.

Note that in case of deletion, we first print the deleted (dequeued) element on a new line and then on a new line, print the new state of the linked list.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

Start Solving! (/editor/practice/6290)