Coding Assignment 3
ESO207 2024-25-I

September 2, 2024

1 Introduction

In this homework, you will write a C program to implement the three array-based data structures: stack, queue, and binary min heap. Each item of these data structures will be a positive integer.

The full marks will be 200. The deadline of this homework is Sunday, September 15, 11 PM IST. Your submissions will not be accepted after this time. Submit a single C file titled 'hw.c' to the Gradescope active assignment titled 'Structures'.

2 Problem Statement

The input to the program will consist of several lines. First line will have two integers o MAX separated by a space. o denotes the option number: 0 for stack, 1 for queue, 2 for binary min heap. MAX denotes the MAX value for the maximum number of items the data structure can hold.

2.1 Stack

For stack, each of the next few lines will consist of either one number a or two numbers a b separated by a space. a, b can take the following options:

- $a = 0$ means print the stack in which case there is no b. The stack must be printed bottom to top i.e. the top of stack is the last item to be printed. Each number will be followed by a single space. After printing the last number followed by a space, print a newline '\n'. If the stack is empty, only print '\n'.
- $a = 1$ means push in which case b is the integer to be pushed. If the stack is full and we are trying to push, simply print '-1\n' and terminate
- $a = 2$ means pop in which case there is no b. Print the integer that was popped followed by a '\n'. If the stack is empty simply print '-1\n' and terminate
- $a = 3$ means end of input in which case there is no b. The program must be terminated.

Visible Input 1

```
0 3
1 10
1 20
0
2
1 12
1 7
2
0
3
```

Visible Output 1

```
10 20
20
7
10 12
```

Explanation The first line says we need to create a stack of maximum 3 items. Then we push 10 and 20, and print bottom to top. Then we pop which will print 20 followed by a newline, push 12, and 7. The queue will be 10, 12, 7 at this point from bottom to top. Then we pop 7 which will print 7 followed by a newline and print the queue. Note that there is a space and a newline after 20 and 12 in output lines 1 and 4. But there is no space after the 20 and 7 at output lines 2 and 3. Then terminate.

Visible Input 2

```
0 4
1 10
1 20
1 5
0
2
0
1 9
1 4
1 7
```

Visible Output 2

```
10 20 5
5
10 20
−1
```

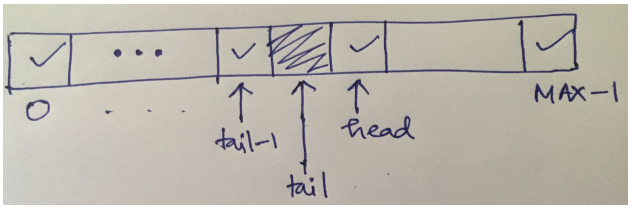

Figure 1: Only Q[tail] is empty so the queue is full

2.2 Queue

Implementing a queue is little bit trickier than stack. Make a note of the following points where we are using 0-indexed arrays as it happens in C.

- The maximum number of items the queue can accommodate is MAX-1. Notice it is 1 less than the stack. Q's indices would be 0, 1, ..., MAX-1. But 1 of the indices will never be used which will be used to keep track of whether the queue is full or not.
- The head is the first item's index in the queue and tail is the index of the upcoming item. Q wraps around. Therefore, the queue will have the following two cases:
  - If head<tail, queue consists of items head, head+1, ..., tail-1
  - If head>tail, queue consists of items head, head+1, ..., MAX-1, 0, 1, tail-1
  - If head=tail, the queue is empty
  - If (head=tail+1) or (head=0 and tail=MAX-1), the queue is full
  - Enqueue will check if the queue is full or not. If not full, it will put a new item at Q[tail] and increment the tail
  - Dequeue will check if the queue is empty or not. If not empty, it will simply return the previous head and decrement the head

  See Figure 1 for an illustration that the queue is full.

For queue, each of the next few input lines will consist of either one number a or two numbers a b separated by a space. a, b can take the following options:

- $a = 0$ means print the queue in which case there is no b. The head of queue is the first item to be printed. Each number will be followed by a single space. After printing the last number followed by a space, print a newline '\n'. If the queue is empty, only print '\n'.
- $a = 1$ means enqueue in which case b is the integer to be added to the end. If queue is full, print '-1\n' and terminate
- $a = 2$ means dequeue in which case there is no b. In this case, remove the front item. Print the removed item followed by '\n'. If queue is empty, print '-1\n' and terminate
- $a = 3$ means end of input in which case there is no b. The program must be terminated.

Visible Input 3

```
1 6
1 1
1 2
1 3
2
1 4
1 1
2
1 5
1 7
0
1 10
```

Visible Output 3

```
1
2
3 4 1 5 7
-1
```

Explanation The first line says we need to create a queue of maximum 4 items. Then we enqueue 1, 2, 3, and dequeue. Dequeue will print '1\n'. This makes the queue 2, 3. Then we enqueue 4, 1. The queue becomes 2, 3, 4, 1. This time dequeue will print '2\n'. Then we enqueue 5, 7. Then the queue will be printed as 3, 4, 1, 5, 7. When we see the 3, we terminate the execution. Note that the array itself will look like: 7, blank, 3, 4, 1, 5. However, the head will be at index 3 and tail at index 2 which is why the queue must be printed as above. At this point queue has MAX-1=5 items. Any attempt of further enqueue will result in termnination by printing a '-1\n'.

Note that there will be space and a newline after 7 at the third output line. There will be new lines after the 1, 2, and -1 at the other three output lines respectively.

2.3 Heap

For heap, each of the next few lines will consist of several numbers a_1 a_2 ... each separated by a space. Each such input line can take the following options.

- $a_1 = 0$ means print the heap in which case you must print the heap in the usual order of the array i.e. from lower to higher index. Each number will be followed by a single space. After printing the last number followed by a space, print a newline '\n'. If the heap is empty, only print '\n'. If a_1 is 0, there will not be any other integers in this input line.
- $a_1 = 1$ means build-heap, in which case a_2 will be the number of items, and a_3 a_4 ... a_{a_2+2} will be the a_2 many items to be put inside the heap. If $a_2 > MAX$ print '-1\n' and terminate
- $a_1 = 2$ means decrease-key. a_2-th item will be decreased to the new value a_3. If a_2-th item's key is already less than a_3 or a_2 is an invalid index, nothing will happen. In this option, a_4 onwards will be empty. Be careful that C is 0-indexed, so you need to actually decrease the item at index $(a_2 - 1)$.
- $a_1 = 3$ means extract the minimum in which case a_2 onwards will be empty. If the Heap is empty and you are trying to extract, print '-1\n' and terminate
- $a_1 = 4$ means end of input in which case a_2 onwards will be empty. Terminate the program.

We'll make sure the inputs have a unique answer. In particular, no two children will have the same key so that during heapify the chosen smallest-key child will be unique.

If the heap is empty and we call extract-min terminate the program after printing a "-1". Similarly, if we are building a heap whose size is larger than the MAX value, terminate the program after printing a "-1".

Visible Input 4

```
2 10
1 7 17 16 15 14 13 12 11
0
3
3
0
2 4 8
0
4
```

Visible Output 4

```
11 13 12 14 16 17 15
11
12
13 14 15 17 16
8 13 15 14 16
```

Example Output The first line says to implement a heap of maximum 10 items. The next line asks us to build a heap of 7 items starting from the array: 17, 16, 15, 14, 13, 12, 11. After arranging these numbers we scan from end to beginning. The heap property is first violated at key 15. So, keys 15 and its minimum child 11 will be exchanged. Then the heap property will violate at 16. Similarly, 16 and 13 will be exchanged. Finally, the heap property will violate at the root. So, 17 will be first exchanged with 11. Then 17 will be exchanged with 12. The resulting heap will be printed as: 11, 13, 12, 14, 16, 17, 15, which is a valid heap. See the Figure 2 for an illustration.

Next we call extract min two times. Therefore, 11 and 12 will be gone. The resulting heap is illustrated in Figure 3. At this point the heap will be printed as 13, 14, 15, 17, 16.

Then we decrease the key of item 4 to 8. So, the 17 will become 8. This 8 will be floated up. The resulting heap is shown in Figure 4. Now if we print the heap we will get: 8, 13, 15, 14, 16. Then we terminate the program.

Note that there will be a space and a newline after 15, 16, and 16 in the three output lines.

For any input scenario not covered in the above, your answer can be arbitrary. It is promised that the hidden and visible testcases will only cover the scenarios mentioned in the above.

3 Test Cases

There will be 8 visible test cases provided to you in a file called 'testcases.zip' attached to the helloIITK assignment. They will include the 4 test cases mentioned above. Each of them carries 10 marks. Additionally, there will be 6 hidden test cases, each carrying 20 marks. Therefore, the total marks will be 200.

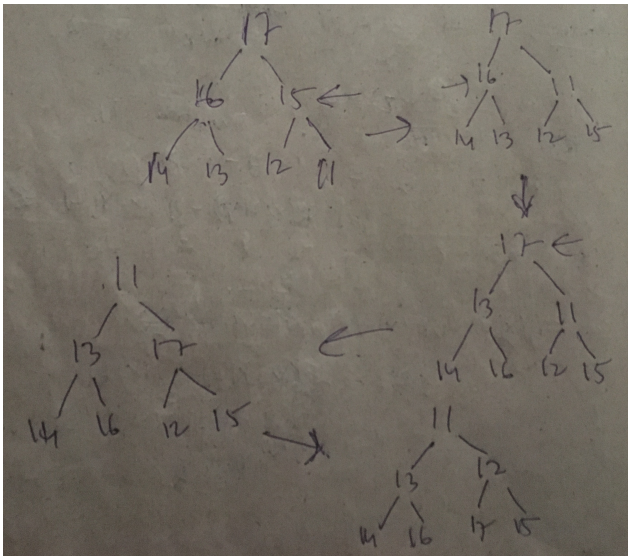

Figure 2: Min-Heapify calls during Build-Heap

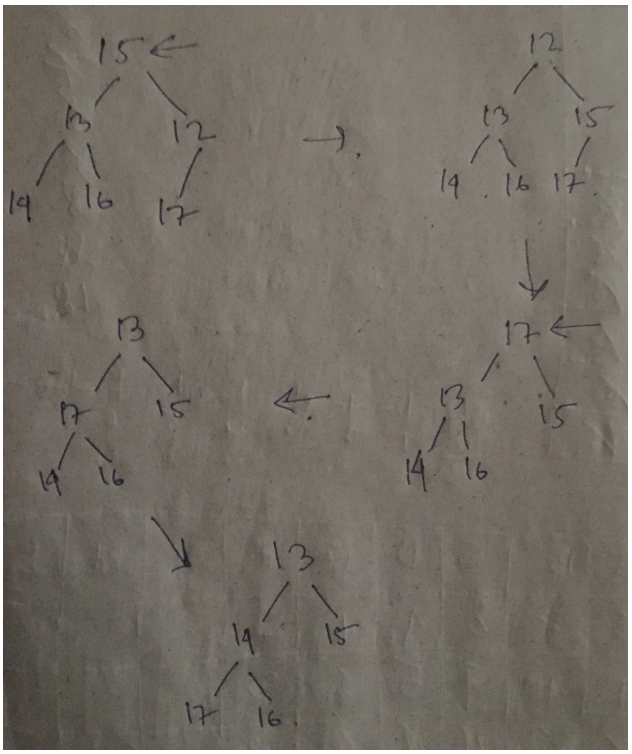

Figure 3: Heap after extracting 11 and 12

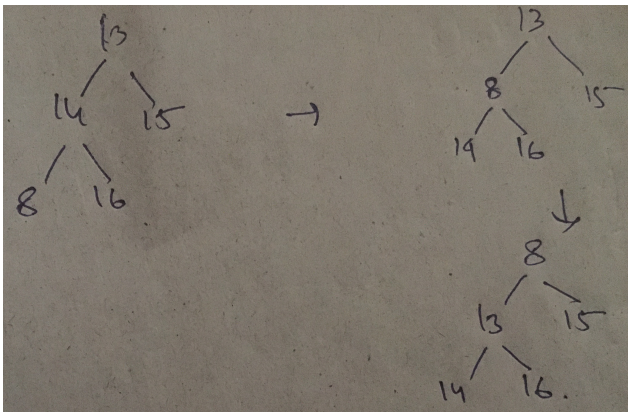

Figure 4: Heap after decreasing 17 to 8

4 Submission Instructions

- You must submit a C program titled 'hw.c'. Other programming languages such as C++, Python, etc. are not allowed. Your code must take the input from "stdin" and write the output to "stdout".
- Your code will be automatically graded in Gradescope on some test cases as above which will be hidden from you. Therefore, you must make sure that you understand and precisely follow the expected input-output behavior.
- Please write a single C code and name it as 'hw.c'. This is extremely important. If you violate this, your code will not pass the automatic test cases even if your code runs correctly on your local machine. Common examples of failures include:
  - if you write a C++ program that has the correct input-output behavior
  - if you write two or more different C codes or .h header codes and link them
  - or write a single correct code but name it as 'test.c'

  In any of the above cases, your code will fail. Thus, while you are perfectly allowed to develop your code in your local machine and it works correctly, your code may run into problems in Gradescope until and unless you follow the above instructions.
- Submit your code on *Gradescope* active assignment titled 'Structures'. Otherwise your code will not be graded. In particular, do NOT submit on hello IITK or over email. Email to us (instructor or the TAs), meet us, or start a discussion in helloIITK if you run into any issues.