Programming Assignment 1: Basic Data Structures

Revision: July 8, 2019

Introduction

Welcome to your first Programming Assignment in the Data Structures course of the Data Structures and Algorithms Specialization!

In this programming assignment, you will be practicing implementing basic data structures and using them to solve algorithmic problems. In some of the problems, you just need to implement and use a data structure from the lectures, while in the others you will also need to invent an algorithm to solve the problem using some of the basic data structures.

In this programming assignment, the grader will show you the input and output data if your solution fails on any of the tests. This is done to help you to get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail. However, note that for very big inputs the grader cannot show them fully, so it will only show the beginning of the input for you to make sense of its size, and then it will be clipped. You will need to generate big inputs for yourself. For all the following programming assignments, the grader will show the input data only in case your solution fails on one of the first few tests (please review the questions **??** and **??** in the FAQ section for a more detailed explanation of this behavior of the grader).

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply the basic data structures you've just studied to solve the given algorithmic problems.
2. Given a piece of code in an unknown programming language, check whether the brackets are used correctly in the code or not.
3. Implement a tree, read it from the input and compute its height.
4. Simulate processing of computer network packets.
5. Extend the standard stack interface with a new method.
6. Slide a window through a sequence of integers and compute the maximum value efficiently in every window.

Passing Criteria: 2 out of 5

Passing this programming assignment requires passing at least 2 out of 5 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Check brackets in the code

Problem Introduction

In this problem you will implement a feature for a text editor to find errors in the usage of brackets in the code.

Problem Description

Task. Your friend is making a text editor for programmers. He is currently working on a feature that will find errors in the usage of different types of brackets. Code can contain any brackets from the set `[]{}()`, where the opening brackets are `[`,`{`, and `(` and the closing brackets corresponding to them are `]`,`}`, and `)`.

For convenience, the text editor should not only inform the user that there is an error in the usage of brackets, but also point to the exact place in the code with the problematic bracket. First priority is to find the first unmatched closing bracket which either doesn't have an opening bracket before it, like `]` in `]()`, or closes the wrong opening bracket, like `}` in `()[}`. If there are no such mistakes, then it should find the first unmatched opening bracket without the corresponding closing bracket after it, like `(` in `{}([]`. If there are no mistakes, text editor should inform the user that the usage of brackets is correct.

Apart from the brackets, code can contain big and small latin letters, digits and punctuation marks.

More formally, all brackets in the code should be divided into pairs of matching brackets, such that in each pair the opening bracket goes before the closing bracket, and for any two pairs of brackets either one of them is nested inside another one as in `(foo[bar])` or they are separate as in `f(a,b)-g[c]`. The bracket `[` corresponds to the bracket `]`, `{` corresponds to `}`, and `(` corresponds to `)`.

Input Format. Input contains one string S which consists of big and small latin letters, digits, punctuation marks and brackets from the set `[]{}()`.

Constraints. The length of S is at least 1 and at most 10^5.

Output Format. If the code in S uses brackets correctly, output "Success" (without the quotes). Otherwise, output the 1-based index of the first unmatched closing bracket, and if there are no unmatched closing brackets, output the 1-based index of the first unmatched opening bracket.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	1.5	5	1.5	2	5	5	3

Memory Limit. 512MB.

Sample 1.

Input:

```
[]
```

Output:

```
Success
```

Explanation:
The brackets are used correctly: there is just one pair of brackets `[` and `]`, they correspond to each other, the left bracket `[` goes before the right bracket `]`, and no two pairs of brackets intersect, because there is just one pair of brackets.

Sample 2.

Input:

```
{}[]
```

Output:

```
Success
```

Explanation:
The brackets are used correctly: there are two pairs of brackets — first pair of `{` and `}`, and second pair of `[` and `]` — and these pairs do not intersect.

Sample 3.

Input:

```
[()]
```

Output:

```
Success
```

Explanation:
The brackets are used correctly: there are two pairs of brackets — first pair of `[` and `]`, and second pair of `(` and `)` — and the second pair is nested inside the first pair.

Sample 4.

Input:

```
(())
```

Output:

```
Success
```

Explanation:
Pairs with the same types of brackets can also be nested.

Sample 5.

Input:

```
{[]}()
```

Output:

```
Success
```

Explanation:
Here there are 3 pairs of brackets, one of them is nested into another one, and the third one is separate from the first two.

Sample 6.

Input:

```
{
```

Output:

```
1
```

Explanation:
The code `{` doesn't use brackets correctly, because brackets cannot be divided into pairs (there is just one bracket). There are no closing brackets, and the first unmatched opening bracket is `{`, and its position is 1, so we output 1.

Sample 7.

Input:

```
{[}
```

Output:

```
3
```

Explanation:
The bracket `}` is unmatched, because the last unmatched opening bracket before it is `[` and not `{`. It is the first unmatched closing bracket, and our first priority is to output the first unmatched closing bracket, and its position is 3, so we output 3.

Sample 8.

Input:

```
foo(bar);
```

Output:

```
Success
```

Explanation:
All the brackets are matching, and all the other symbols can be ignored.

Sample 9.

Input:

```
foo(bar[i);
```

Output:

```
10
```

Explanation:
`)` doesn't match `[`, so `)` is the first unmatched closing bracket, so we output its position, which is 10.

Starter Files

There are starter solutions only for C++, Java and Python3, and if you use other languages, you need to implement solution from scratch. Starter solutions read the code from the input and go through the code character-by-character and provide convenience methods. You need to implement the processing of the brackets to find the answer to the problem and to output the answer.

What to Do

To solve this problem, you can slightly modify the `IsBalanced` algorithm from the lectures to account not only for the brackets, but also for other characters in the code, and return not just whether the code uses brackets correctly, but also what is the first position where the code becomes broken.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

2 Compute tree height

Problem Introduction

Trees are used to manipulate hierarchical data such as hierarchy of categories of a retailer or the directory structure on your computer. They are also used in data analysis and machine learning both for hierarchical clustering and building complex predictive models, including some of the best-performing in practice algorithms like Gradient Boosting over Decision Trees and Random Forests. In the later modules of this course, we will introduce balanced binary search trees (BST) — a special kind of trees that allows to very efficiently store, manipulate and retrieve data. Balanced BSTs are thus used in databases for efficient storage and actually in virtually any non-trivial programs, typically via built-in data structures of the programming language at hand.

In this problem, your goal is to get used to trees. You will need to read a description of a tree from the input, implement the tree data structure, store the tree and compute its height.

Problem Description

Task. You are given a description of a rooted tree. Your task is to compute and output its height. Recall that the height of a (rooted) tree is the maximum depth of a node, or the maximum distance from a leaf to the root. You are given an arbitrary tree, not necessarily a binary tree.

Input Format. The first line contains the number of nodes n. The second line contains n integer numbers from -1 to $n-1$ — parents of nodes. If the i-th one of them ($0 \le i \le n-1$) is -1, node i is the root, otherwise it's 0-based index of the parent of i-th node. It is guaranteed that there is exactly one root. It is guaranteed that the input represents a tree.

Constraints. $1 \le n \le 10^5$.

Output Format. Output the height of the tree.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

Sample 1.

Input:

```
5
4 -1 4 1 1
```

Output:

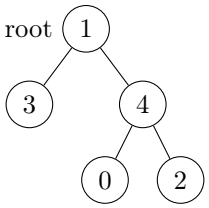
```
3
```

The input means that there are 5 nodes with numbers from 0 to 4, node 0 is a child of node 4, node 1 is the root, node 2 is a child of node 4, node 3 is a child of node 1 and node 4 is a child of node 1. To see this, let us write numbers of nodes from 0 to 4 in one line and the numbers given in the input in the second line underneath:

0 1 2 3 4

4 -1 4 1 1

Now we can see that the node number 1 is the root, because -1 corresponds to it in the second line. Also, we know that the nodes number 3 and number 4 are children of the root node 1. Also, we know that the nodes number 0 and number 2 are children of the node 4.



The height of this tree is 3, because the number of vertices on the path from root 1 to leaf 2 is 3.

Sample 2.

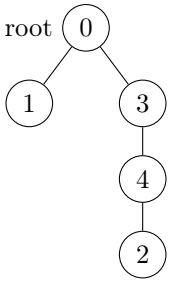
Input:

```
5
-1 0 4 0 3
```

Output:

```
4
```

Explanation:
The input means that there are 5 nodes with numbers from 0 to 4, node 0 is the root, node 1 is a child of node 0, node 2 is a child of node 4, node 3 is a child of node 0 and node 4 is a child of node 3. The height of this tree is 4, because the number of nodes on the path from root 0 to leaf 2 is 4.



Starter Files

The starter solutions in this problem read the description of a tree, store it in memory, compute the height in a naive way and write the output. You need to implement faster height computation. Starter solutions are available for C++, Java and Python3, and if you use other languages, you need to implement a solution from scratch.

What to Do

To solve this problem, change the height function described in the lectures with an implementation which will work for an arbitrary tree. Note that the tree can be very deep in this problem, so you should be careful to avoid stack overflow problems if you're using recursion, and definitely test your solution on a tree with the maximum possible height.

Suggestion: Take advantage of the fact that the labels for each tree node are integers in the range $0..n-1$: you can store each node in an array whose index is the label of the node. By storing the nodes in an array, you have $O(1)$ access to any node given its label.

Create an array of n nodes:

```
allocate nodes[n]
for i ← 0 to n − 1:
  nodes[i] =new Node
```

Then, read each parent index:

```
for child_index ← 0 to n − 1:
  read parent_index
  if parent_index == −1:
    root ← child_index
  else:
    nodes[parent_index].addChild(nodes[child_index])
```

Once you've built the tree, you'll then need to compute its height. If you don't use recursion, you needn't worry about stack overflow problems. Without recursion, you'll need some auxiliary data structure to keep track of the current state (in the breadth-first seach code in lecture, for example, we used a queue).

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

3 Network packet processing simulation

Problem Introduction

In this problem you will implement a program to simulate the processing of network packets.

Problem Description

Task. You are given a series of incoming network packets, and your task is to simulate their processing. Packets arrive in some order. For each packet number i, you know the time when it arrived A_i and the time it takes the processor to process it P_i (both in milliseconds). There is only one processor, and it processes the incoming packets in the order of their arrival. If the processor started to process some packet, it doesn't interrupt or stop until it finishes the processing of this packet, and the processing of packet i takes exactly P_i milliseconds.

The computer processing the packets has a network buffer of fixed size S. When packets arrive, they are stored in the buffer before being processed. However, if the buffer is full when a packet arrives (there are S packets which have arrived before this packet, and the computer hasn't finished processing any of them), it is dropped and won't be processed at all. If several packets arrive at the same time, they are first all stored in the buffer (some of them may be dropped because of that — those which are described later in the input). The computer processes the packets in the order of their arrival, and it starts processing the next available packet from the buffer as soon as it finishes processing the previous one. If at some point the computer is not busy, and there are no packets in the buffer, the computer just waits for the next packet to arrive. Note that a packet leaves the buffer and frees the space in the buffer as soon as the computer finishes processing it.

Input Format. The first line of the input contains the size S of the buffer and the number n of incoming network packets. Each of the next n lines contains two numbers. i-th line contains the time of arrival A_i and the processing time P_i (both in milliseconds) of the i-th packet. It is guaranteed that the sequence of arrival times is non-decreasing (however, it can contain the exact same times of arrival in milliseconds — in this case the packet which is earlier in the input is considered to have arrived earlier).

Constraints. All the numbers in the input are integers. $1 \le S \le 10^5$; $0 \le n \le 10^5$; $0 \le A_i \le 10^6$; $0 \le P_i \le 10^3$; $A_i \le A_{i+1}$ for $1 \le i \le n-1$.

Output Format. For each packet output either the moment of time (in milliseconds) when the processor began processing it or -1 if the packet was dropped (output the answers for the packets in the same order as the packets are given in the input).

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

Sample 1.

Input:

```
1 0
```

Output:

```

```

Explanation:
If there are no packets, you shouldn't output anything.

Sample 2.

Input:

```
1 1
0 0
```

Output:

```
0
```

Explanation:
The only packet arrived at time 0, and computer started processing it immediately.

Sample 3.

Input:

```
1 2
0 1
0 1
```

Output:

```
0
-1
```

Explanation:
The first packet arrived at time 0, the second packet also arrived at time 0, but was dropped, because the network buffer has size 1 and it was full with the first packet already. The first packet started processing at time 0, and the second packet was not processed at all.

Sample 4.

Input:

```
1 2
0 1
1 1
```

Output:

```
0
1
```

Explanation:
The first packet arrived at time 0, the computer started processing it immediately and finished at time 1. The second packet arrived at time 1, and the computer started processing it immediately.

Starter Files

The starter solutions for C++, Java and Python3 in this problem read the input, pass the requests for processing of packets one-by-one and output the results. They declare a class that implements network buffer simulator. The class is partially implemented, and your task is to implement the rest of it. If you use other languages, you need to implement the solution from scratch.

What to Do

To solve this problem, you can use a list or a queue (in this case the queue should allow accessing its last element, and such queue is usually called a deque). You can use the corresponding built-in data structure in your language of choice.

One possible solution is to store in the list or queue `finish_time` the times when the computer will finish processing the packets which are currently stored in the network buffer, in increasing order. When a new packet arrives, you will first need to pop from the front of `finish_time` all the packets which are already processed by the time new packet arrives. Then you try to add the finish time for the new packet in `finish_time`. If the buffer is full (there are already S finish times in `finish_time`), the packet is dropped. Otherwise, its processing finish time is added to `finish_time`.

If `finish_time` is empty when a new packet arrives, computer will start processing the new packet immediately as soon as it arrives. Otherwise, computer will start processing the new packet as soon as it finishes to process the last of the packets currently in `finish_time` (here is when you need to access the last element of `finish_time` to determine when the computer will start to process the new packet). You will also need to compute the processing finish time by adding P_i to the processing start time and push it to the back of `finish_time`.

You need to remember to output the processing start time for each packet instead of the processing finish time which you store in `finish_time`.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

4 Extending stack interface

Problem Introduction

Stack is an abstract data type supporting the operations `Push()` and `Pop()`. It is not difficult to implement it in a way that both these operations work in constant time. In this problem, you goal will be to implement a stack that also supports finding the maximum value and to ensure that all operations still work in constant time.

Problem Description

Task. Implement a stack supporting the operations `Push()`, `Pop()`, and `Max()`.

Input Format. The first line of the input contains the number q of queries. Each of the following q lines specifies a query of one of the following formats: `push v`, `pop`, or max.

Constraints. $1 \le q \le 400\,000$, $0 \le v \le 10^5$.

Output Format. For each `max` query, output (on a separate line) the maximum value of the stack.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	1.5	5	1.5	2	5	5	3

Memory Limit. 512MB.

Sample 1.

Input:

```
5
push 2
push 1
max
pop
max
```

Output:

```
2
2
```

Explanation:
After the first two `push` queries, the stack contains elements 1 and 2. After the `pop` query, the element 1 is removed.

Sample 2.

Input:

```
5
push 1
push 2
max
pop
max
```

Output:

```
2
1
```

Sample 3.

Input:

```
10
push 2
push 3
push 9
push 7
push 2
max
max
max
pop
max
```

Output:

```
9
9
9
9
```

Sample 4.

Input:

```
3
push 1
push 7
pop
```

Output:

```

```

Explanation:
The output is empty since there are no `max` queries.

Sample 5.

Input:

```
6
push 7
push 1
push 7
max
pop
max
```

Output:

```
7
7
```

Starter Files

The starter solutions in `C++`, `Java`, and `Python3` process the queries naively: for each `max` query they scan the current contents of the stack to find the maximum value. Hence the `max` query has running time proportional to the size of the stack. Your goal is to modify it so that its running time becomes constant. For other programming languages, you need to implement a solution from scratch.

What to Do

Think about using an auxiliary stack.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

5 Maximum in Sliding Window

Problem Introduction

Given a sequence $a_1, \ldots, a_n$ of integers and an integer $m \leq n$, find the maximum among $\{a_i, \ldots, a_{i+m-1}\}$ for every $1 \leq i \leq n - m + 1$. A naive $O(nm)$ algorithm for solving this problem scans each window separately. Your goal is to design an $O(n)$ algorithm.

Problem Description

Input Format. The first line contains an integer n, the second line contains n integers $a_1, \ldots, a_n$ separated by spaces, the third line contains an integer m.

Constraints. $1 \leq n \leq 10^5$, $1 \leq m \leq n$, $0 \leq a_i \leq 10^5$ for all $1 \leq i \leq n$.

Output Format. Output $\max\{a_i, \ldots, a_{i+m-1}\}$ for every $1 \leq i \leq n - m + 1$.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	1	1	1.5	5	1.5	2	5	5	3

Memory Limit. 512MB.

Sample 1.

Input:

```
8
2 7 3 1 5 2 6 2
4
```

Output:

```
7 7 5 6 6
```

What to Do

We give hints for three different solutions.

1. *Implement a queue using two stacks.* Use a queue data structure for sliding a window through a sequence: for shifting a window one position to the right, pop the leftmost element of the queue and push a new element from the new window. A queue can be implemented using two stacks such that each queue operation takes constant time *on average*. Then, use your implementation of the stack with maximum.

2. *Preprocess block suffixes and prefixes.* Partition the input sequence into blocks of length m and precompute the maximum for every suffix and every prefix of each block. Afterwards, the maximum in each sliding window can be found by considering a suffix and a prefix of two consecutive blocks.

3. *Store relevant items in a dequeue.* Use a double-ended queue (dequeue) to store elements of the current window. At the same time, store only relevant elements: before adding a new element drop all smaller elements.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

6 Appendix

6.1 Compiler Flags

C (`gcc 5.2.1`). File extensions: `.c`. Flags:

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (`g++ 5.2.1`). File extensions: `.cc`, `.cpp`. Flags:

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

If your `C/C++` compiler does not recognize `-std=c++14` flag, try replacing it with `-std=c++0x` flag or compiling without this flag at all (all starter solutions can be compiled without it). On Linux and MacOS, you most probably have the required compiler. On Windows, you may use your favorite compiler or install, e.g., `cygwin`.

C# (`mono 3.2.8`). File extensions: `.cs`. Flags:

```
mcs
```

Haskell (`ghc 7.8.4`). File extensions: `.hs`. Flags:

```
ghc -O2
```

Java (`Open JDK 8`). File extensions: `.java`. Flags:

```
javac -encoding UTF-8
java -Xmx1024m
```

JavaScript (`Node v10.15.3`). File extensions: `.js`. No flags:

```
nodejs
```

Kotlin (`Kotlin 1.2.21`). File extensions: `.kt`. Flags:

```
kotlinc
java -Xmx1024m
```

Python 2 (`CPython 2.7`). File extensions: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python2"). No flags:

```
python2
```

Python 3 (`CPython 3.4`). File extensions: `.py3` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python3"). No flags:

```
python3
```

Ruby (`Ruby 2.1.5`). File extensions: `.rb`.

```
ruby
```

Rust (`Rust 1.28.0`). File extensions: `.rs`.

```
rustc
```

Scala (`Scala 2.11.6`). File extensions: `.scala`.

```
scalac
```

6.2 Frequently Asked Questions

Why My Submission Is Not Graded?

You need to create a submission and upload the *source file* (rather than the executable file) of your solution. Make sure that after uploading the file with your solution you press the blue "Submit" button at the bottom. After that, the grading starts, and the submission being graded is enclosed in an orange rectangle. After the testing is finished, the rectangle disappears, and the results of the testing of all problems are shown.

What Are the Possible Grading Outcomes?

There are only two outcomes: "pass" or "no pass." To pass, your program must return a correct answer on all the test cases we prepared for you, and do so under the time and memory constraints specified in the problem statement. If your solution passes, you get the corresponding feedback "Good job!" and get a point for the problem. Your solution fails if it either crashes, returns an incorrect answer, works for too long, or uses too much memory for some test case. The feedback will contain the index of the first test case on which your solution failed and the total number of test cases in the system. The tests for the problem are numbered from 1 to the total number of test cases for the problem, and the program is always tested on all the tests in the order from the first test to the test with the largest number.

Here are the possible outcomes:

- `Good job!` Hurrah! Your solution passed, and you get a point!
- `Wrong answer.` Your solution outputs incorrect answer for some test case. Check that you consider all the cases correctly, avoid integer overflow, output the required white spaces, output the floating point numbers with the required precision, don't output anything in addition to what you are asked to output in the output specification of the problem statement.
- `Time limit exceeded.` Your solution worked longer than the allowed time limit for some test case. Check again the running time of your implementation. Test your program locally on the test of maximum size specified in the problem statement and check how long it works. Check that your program doesn't wait for some input from the user which makes it to wait forever.
- `Memory limit exceeded.` Your solution used more than the allowed memory limit for some test case. Estimate the amount of memory that your program is going to use in the worst case and check that it does not exceed the memory limit. Check that your data structures fit into the memory limit. Check that you don't create large arrays or lists or vectors consisting of empty arrays or empty strings, since those in some cases still eat up memory. Test your program locally on the tests of maximum size specified in the problem statement and look at its memory consumption in the system.
- `Cannot check answer. Perhaps the output format is wrong.` This happens when you output something different than expected. For example, when you are required to output either "Yes" or "No", but instead output 1 or 0. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (please follow the exact output format specified in the problem statement). Maybe your program doesn't output anything, because it crashes.
- `Unknown signal 6 (or 7, or 8, or 11, or some other).` This happens when your program crashes. It can be because of a division by zero, accessing memory outside of the array bounds, using uninitialized variables, overly deep recursion that triggers a stack overflow, sorting with a contradictory comparator, removing elements from an empty data structure, trying to allocate too much memory, and many other reasons. Look at your code and think about all those possibilities. Make sure that you use the same compiler and the same compiler flags as we do.
- `Internal error: exception...` Most probably, you submitted a compiled program instead of a source code.

- `Grading failed.` Something wrong happened with the system. Report this through Coursera or edX Help Center.

May I Post My Solution at the Forum?

Please do not post any solutions at the forum or anywhere on the web, even if a solution does not pass the tests (as in this case you are still revealing parts of a correct solution). Our students follow the Honor Code: "I will not make solutions to homework, quizzes, exams, projects, and other assignments available to anyone else (except to the extent an assignment explicitly permits sharing solutions)."

Do I Learn by Trying to Fix My Solution?

My implementation always fails in the grader, though I already tested and stress tested it a lot. Would not it be better if you gave me a solution to this problem or at least the test cases that you use? I will then be able to fix my code and will learn how to avoid making mistakes. Otherwise, I do not feel that I learn anything from solving this problem. I am just stuck.

First of all, learning from your mistakes is one of the best ways to learn.

The process of trying to invent new test cases that might fail your program is difficult but is often enlightening. Thinking about properties of your program makes you understand what happens inside your program and in the general algorithm you're studying much more.

Also, it is important to be able to find a bug in your implementation without knowing a test case and without having a reference solution, just like in real life. Assume that you designed an application and an annoyed user reports that it crashed. Most probably, the user will not tell you the exact sequence of operations that led to a crash. Moreover, there will be no reference application. Hence, it is important to learn how to find a bug in your implementation yourself, without a magic oracle giving you either a test case that your program fails or a reference solution. We encourage you to use programming assignments in this class as a way of practicing this important skill.

If you have already tested your program on all corner cases you can imagine, constructed a set of manual test cases, applied stress testing, etc, but your program still fails, try to ask for help on the forum. We encourage you to do this by first explaining what kind of corner cases you have already considered (it may happen that by writing such a post you will realize that you missed some corner cases!), and only afterwards asking other learners to give you more ideas for tests cases.