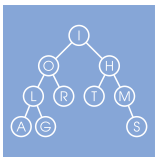


Module: Binary Search Trees (Week 4 out of 4)
Course: Data Structures (Course 2 out of 6)
Specialization: Data Structures and Algorithms

Programming Assignment 4: Binary Search Trees

Revision: December 8, 2016

Introduction

In this programming assignment, you will practice implementing binary search trees including balanced ones and using them to solve algorithmic problems. In some cases you will just implement an algorithm from the lectures, while in others you will need to invent an algorithm to solve the given problem using hashing.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply binary search trees to solve the given algorithmic problems.
2. Implement in-order, pre-order and post-order traversal of a binary tree.
3. Implement a data structure to compute range sums.
4. Implement a data structure that can store strings and quickly cut parts and patch them back.

Passing Criteria: 2 out of 3

Passing this programming assignment requires passing at least 2 out of 3 code problems from this assignment. In turn, passing a code problem 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 Problem: Binary tree traversals

Problem Introduction

In this problem you will implement in-order, pre-order and post-order traversals of a binary tree. These traversals were defined in the week 1 lecture on tree traversals, but it is very useful to practice implementing them to understand binary search trees better.

Problem Description

Task. You are given a rooted binary tree. Build and output its in-order, pre-order and post-order traversals.

Input Format. The first line contains the number of vertices n. The vertices of the tree are numbered from 0 to $n - 1$. Vertex 0 is the root.

The next n lines contain information about vertices 0, 1, ..., $n - 1$ in order. Each of these lines contains three integers key_i, $left_i$ and $right_i$ — key_i is the key of the i-th vertex, $left_i$ is the index of the left child of the i-th vertex, and $right_i$ is the index of the right child of the i-th vertex. If i doesn't have left or right child (or both), the corresponding $left_i$ or $right_i$ (or both) will be equal to -1.

Constraints. $1 \leq n \leq 10^5$; $0 \leq key_i \leq 10^9$; $-1 \leq left_i, right_i \leq n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \neq -1$ and $right_i \neq -1$, then $left_i \neq right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex.

Output Format. Print three lines. The first line should contain the keys of the vertices in the in-order traversal of the tree. The second line should contain the keys of the vertices in the pre-order traversal of the tree. The third line should contain the keys of the vertices in the post-order traversal of the tree.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 12 sec, `Python`: 6 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 6 sec, `Ruby`: 6 sec, `Scala`: 12 sec.

Memory Limit. 512MB.

Sample 1.

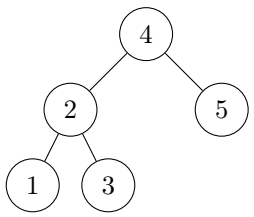
Input:

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

Output:

```
1 2 3 4 5
4 2 1 3 5
1 3 2 5 4
```

Explanation:



Sample 2.

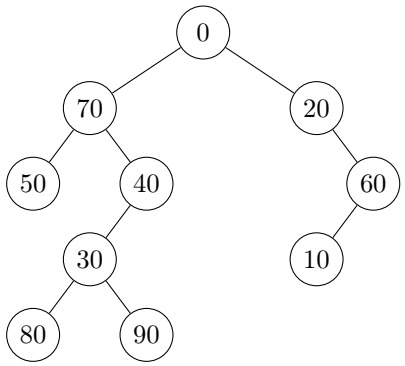
Input:

```
10
0 7 2
10 -1 -1
20 -1 6
30 8 9
40 3 -1
50 -1 -1
60 1 -1
70 5 4
80 -1 -1
90 -1 -1
```

Output:

```
50 70 80 30 90 40 0 20 10 60
0 70 50 40 30 80 90 20 60 10
50 80 90 30 40 70 10 60 20 0
```

Explanation:



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 input, define the methods to compute different traversals of the binary tree and write the output. You need to implement the traversal methods.

What to Do

Implement the traversal algorithms from the lectures. 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.

Need Help?

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

2 Problem: Set with range sums

Problem Introduction

In this problem, your goal is to implement a data structure to store a set of integers and quickly compute range sums.

Problem Description

Task. Implement a data structure that stores a set S of integers with the following allowed operations:

- **add**(i) — add integer i into the set S (if it was there already, the set doesn't change).
- **del**(i) — remove integer i from the set S (if there was no such element, nothing happens).
- **find**(i) — check whether i is in the set S or not.
- **sum**(l, r) — output the sum of all elements v in S such that $l \le v \le r$.

Input Format. Initially the set S is empty. The first line contains n — the number of operations. The next n lines contain operations. Each operation is one of the following:

- "+ i" — which means **add** some integer (not i, see below) to S,
- "- i" — which means **del** some integer (not i, see below)from S,
- "? i" — which means **find** some integer (not i, see below)in S,
- "s l r" — which means compute the **sum** of all elements of S within some range of values (not from l to r, see below).

However, to make sure that your solution can work in an online fashion, each request will actually depend on the result of the last **sum** request. Denote $M = 1\ 000\ 000\ 001$. At any moment, let x be the result of the last **sum** operation, or just 0 if there were no **sum** operations before. Then

- "+ i" means **add**$((i + x) \bmod M)$,
- "- i" means **del**$((i + x) \bmod M)$,
- "? i" means **find**$((i + x) \bmod M)$,
- "s l r" means **sum**$((l + x) \bmod M, (r + x) \bmod M)$.

Constraints. $1 \le n \le 100\ 000$; $0 \le i \le 10^9$.

Output Format. For each find request, just output "Found" or "Not found" (without quotes; note that the first letter is capital) depending on whether $(i + x) \bmod M$ is in S or not. For each **sum** query, output the sum of all the values v in S such that $((l+x) \bmod M) \le v \le ((r+x) \bmod M)$ (it is guaranteed that in all the tests $((l + x) \bmod M) \le ((r + x) \bmod M)$), where x is the result of the last **sum** operation or 0 if there was no previous **sum** operation.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 4 sec, `Python`: 120 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 120 sec, `Ruby`: 120 sec, `Scala`: 4 sec.

Memory Limit. 512MB.

Sample 1.

Input:

```
15
? 1
+ 1
? 1
+ 2
s 1 2
+ 1000000000
? 1000000000
- 1000000000
? 1000000000
s 999999999 1000000000
- 2
? 2
- 0
+ 9
s 0 9
```

Output:

```
Not found
Found
3
Found
Not found
1
Not found
10
```

Explanation:
For the first 5 queries, $x = 0$. For the next 5 queries, $x = 3$. For the next 5 queries, $x = 1$. The actual list of operations is:
find(1)
add(1)
find(1)
add(2)
sum$(1, 2) \to 3$
add(2)
find(2) $\to$ Found
del(2)
find(2) $\to$ Not found
sum$(1, 2) \to 1$
del(3)
find(3) $\to$ Not found
del(1)
add(10)
sum$(1, 10) \to 10$

Adding the same element twice doesn't change the set. Attempts to remove an element which is not in the set are ignored.

Sample 2.
Input:

```
5
? 0
+ 0
? 0
- 0
? 0
```

Output:

```
Not found
Found
Not found
```

Explanation:
First, 0 is not in the set. Then it is added to the set. Then it is removed from the set.

Sample 3.

Input:

```
5
+ 491572259
? 491572259
? 899375874
s 310971296 877523306
+ 352411209
```

Output:

```
Found
Not found
491572259
```

Explanation:
First, 491572259 is added to the set, then it is found there. Number 899375874 is not in the set. The only number in the set is now 491572259, and it is in the range between 310971296 and 877523306, so the sum of all numbers in this range is equal to 491572259.

Starter Files

The starter solutions in C++, Java and Python3 read the input, write the output, fully implement splay tree and show how to use its methods to solve this problem, but don't solve the whole problem. You need to finish the implementation. If you use other languages, you need to implement a solution from scratch.

Note that we strongly encourage you to use stress testing, max tests, testing for min and max values of each parameter according to the restrictions section and other testing techniques and advanced advice from this reading. If you're stuck for a long time, you can read ths forum thread to find out what other learners struggled with, how did they overcome their troubles and what tests did they come up with. If you're still stuck, you can read the hints in the next What to Do section mentioning some of the common problems and how to test for them, resolve some of them. Finally, if none of this worked, we included some of the trickier test cases in the starter files for this problem, so you can use them to debug your program if it fails on one of those tests in the grader. However, you will learn more if you pass this problem without looking at those test cases in the starter files.

What to Do

Use splay tree to efficiently store the set, add, delete and find elements. For each node in the tree, store additionally the sum of all the elements in the subtree of this node. Don't forget to update this sum each time the tree changes. Use split operation to cut ranges from the tree. To get the sum of all the needed

elements after split, just look at the sum stored in the root of the splitted tree. Don't forget to merge the trees back after the **sum** operation.

Some hints based on the problems some learners encountered with their solutions:

- Use the sum attribute to keep updated the sum in the subtree, don't compute the sum from scratch each time, otherwise it will work too slow.
- Don't forget to do splay after each operation with a splay tree.
- Don't forget to splay the node which was accessed last during the find operation.
- Don't forget to update the root variable after each operation with the tree, because splay operation changes root, but it doesn't change where your root variable is pointing in some of the starters.
- Don't forget to merge back after splitting the tree.
- When you detach a node from its parent, don't forget to detach pointers from both ends.
- Don't forget to update all the pointers correctly when merging the trees together.
- Test sum operation when there are no elements within the range.
- Test sum operation when all the elements are within the range.
- Beware of integer overflow.
- Don't forget to check for null when erasing.
- Test: Try adding nodes in the tree in such an order that the tree becomes very unbalanced. Play with this visualization to find out how to do it. Create a very big unbalanced tree. Then try searching for an element that is not in the tree many times.
- Test: add some elements and then remove all the elements from the tree.

Need Help?

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

3 Advanced Problem: Rope

We strongly recommend you start solving advanced problems only when you are done with the basic problems (for some advanced problems, algorithms are not covered in the video lectures and require additional ideas to be solved; for some other advanced problems, algorithms are covered in the lectures, but implementing them is a more challenging task than for other problems).

Problem Introduction

In this problem you will implement Rope — data structure that can store a string and efficiently cut a part (a substring) of this string and insert it in a different position. This data structure can be enhanced to become persistent — that is, to allow access to the previous versions of the string. These properties make it a suitable choice for storing the text in text editors.

This is a very advanced problem, harder than all the previous advanced problems in this course. Don't be upset if it doesn't crack. Congratulations to all the learners who are able to successfully pass this problem!

Problem Description

Task. You are given a string S and you have to process n queries. Each query is described by three integers i, j, k and means to cut substring $S[i..j]$ (i and j are 0-based) from the string and then insert it after the k-th symbol of the remaining string (if the symbols are numbered from 1). If $k = 0$, $S[i..j]$ is inserted in the beginning. See the examples for further clarification.

Input Format. The first line contains the initial string S.
The second line contains the number of queries q.
Next q lines contain triples of integers i, j, k.

Constraints. S contains only lowercase english letters. $1 \le |S| \le 300\,000$; $1 \le q \le 100\,000$; $0 \le i \le j \le n - 1$; $0 \le k \le n - (j - i + 1)$.

Output Format. Output the string after all q queries.

Time Limits. `C`: 3 sec, `C++`: 3 sec, `Java`: 6 sec, `Python`: 120 sec. `C#`: 4.5 sec, `Haskell`: 6 sec, `JavaScript`: 120 sec, `Ruby`: 120 sec, `Scala`: 12 sec.

Memory Limit. 512MB.

Sample 1.

Input:

```
hlelowrold
2
1 1 2
6 6 7
```

Output:

```
helloworld
```

Explanation:
hlelowrold → *hellowrold* → *helloworld*
When $i = j = 1$, $S[i..j] = l$, and it is inserted after the 2-nd symbol of the remaining string *helowrold*, which gives *hellowrold*. Then $i = j = 6$, so $S[i..j] = r$, and it is inserted after the 7-th symbol of the remaining string *hellowold*, which gives *helloworld*.

Sample 2.

Input:

```
abcdef
2
0 1 1
4 5 0
```

Output:

```
efcabd
```

Explanation:
$abcdef \to cabdef \to efcabd$

Starter Files

The starter solutions for C++ and Java in this problem read the input, implement a naive algorithm to cut and paste substrings and write the output. The starter solution for Python3 just reads the input and writes the output. You need to implement a data structure to make the operations with string very fast. If you use other languages, you need to implement the solution from scratch.

What to Do

Use splay tree to store the string. Use the split and merge methods of the splay tree to cut and paste substrings. Think what should be stored as the key in the splay tree. Try to find analogies with the ideas from this lecture.

Need Help?

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

4 General Instructions and Recommendations on Solving Algorithmic Problems

Your main goal in an algorithmic problem is to implement a program that solves a given computational problem in just few seconds even on massive datasets. Your program should read a dataset from the standard input and write an answer to the standard output.

Below we provide general instructions and recommendations on solving such problems. Before reading them, go through readings and screencasts in the first module that show a step by step process of solving two algorithmic problems: link.

4.1 Reading the Problem Statement

You start by reading the problem statement that contains the description of a particular computational task as well as time and memory limits your solution should fit in, and one or two sample tests. In some problems your goal is just to implement carefully an algorithm covered in the lectures, while in some other problems you first need to come up with an algorithm yourself.

4.2 Designing an Algorithm

If your goal is to design an algorithm yourself, one of the things it is important to realize is the expected running time of your algorithm. Usually, you can guess it from the problem statement (specifically, from the subsection called constraints) as follows. Modern computers perform roughly 10^8–10^9 operations per second. So, if the maximum size of a dataset in the problem description is $n = 10^5$, then most probably an algorithm with quadratic running time is not going to fit into time limit (since for $n = 10^5$, $n^2 = 10^{10}$) while a solution with running time $O(n \log n)$ will fit. However, an $O(n^2)$ solution will fit if n is up to $10^3 = 1000$, and if n is at most 100, even $O(n^3)$ solutions will fit. In some cases, the problem is so hard that we do not know a polynomial solution. But for n up to 18, a solution with $O(2^n n^2)$ running time will probably fit into the time limit.

To design an algorithm with the expected running time, you will of course need to use the ideas covered in the lectures. Also, make sure to carefully go through sample tests in the problem description.

4.3 Implementing Your Algorithm

When you have an algorithm in mind, you start implementing it. Currently, you can use the following programming languages to implement a solution to a problem: `C`, `C++`, `C#`, `Haskell`, `Java`, `JavaScript`, `Python2`, `Python3`, `Ruby`, `Scala`. For all problems, we will be providing starter solutions for `C++`, `Java`, and `Python3`. If you are going to use one of these programming languages, use these starter files. For other programming languages, you need to implement a solution from scratch.

4.4 Compiling Your Program

For solving programming assignments, you can use any of the following programming languages: `C`, `C++`, `C#`, `Haskell`, `Java`, `JavaScript`, `Python2`, `Python3`, `Ruby`, and `Scala`. However, we will only be providing starter solution files for `C++`, `Java`, and `Python3`. The programming language of your submission is detected automatically, based on the extension of your submission.

We have reference solutions in `C++`, `Java` and `Python3` which solve the problem correctly under the given restrictions, and in most cases spend at most 1/3 of the time limit and at most 1/2 of the memory limit. You can also use other languages, and we've estimated the time limit multipliers for them, however, we have no guarantee that a correct solution for a particular problem running under the given time and memory constraints exists in any of those other languages.

Your solution will be compiled as follows. We recommend that when testing your solution locally, you use the same compiler flags for compiling. This will increase the chances that your program behaves in the

same way on your machine and on the testing machine (note that a buggy program may behave differently when compiled by different compilers, or even by the same compiler with different 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 v6.3.0`). File extensions: `.js`. Flags:

```
nodejs
```

- `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
```

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

```
scalac
```

4.5 Testing Your Program

When your program is ready, you start testing it. It makes sense to start with small datasets (for example, sample tests provided in the problem description). Ensure that your program produces a correct result.

You then proceed to checking how long does it take your program to process a massive dataset. For this, it makes sense to implement your algorithm as a function like `solve(dataset)` and then implement an additional procedure `generate()` that produces a large dataset. For example, if an input to a problem is a sequence of integers of length $1 \le n \le 10^5$, then generate a sequence of length exactly 10^5, pass it to your `solve()` function, and ensure that the program outputs the result quickly.

Also, check the boundary values. Ensure that your program processes correctly sequences of size $n = 1, 2, 10^5$. If a sequence of integers from 0 to, say, 10^6 is given as an input, check how your program behaves when it is given a sequence $0, 0, \ldots, 0$ or a sequence $10^6, 10^6, \ldots, 10^6$. Check also on randomly generated data. For each such test check that you program produces a correct result (or at least a reasonably looking result).

In the end, we encourage you to stress test your program to make sure it passes in the system at the first attempt. See the readings and screencasts from the first week to learn about testing and stress testing: link.

4.6 Submitting Your Program to the Grading System

When you are done with testing, you submit your program to the grading system. For this, you go the submission page, create a new submission, and upload a file with your program. The grading system then compiles your program (detecting the programming language based on your file extension, see Subsection 4.4) and runs it on a set of carefully constructed tests to check that your program always outputs a correct result and that it always fits into the given time and memory limits. The grading usually takes no more than a minute, but in rare cases when the servers are overloaded it might take longer. Please be patient. You can safely leave the page when your solution is uploaded.

As a result, you get a feedback message from the grading system. The feedback message that you will love to see is: `Good job!` This means that your program has passed all the tests. On the other hand, the three messages `Wrong answer`, `Time limit exceeded`, `Memory limit exceeded` notify you that your program failed due to one these three reasons. Note that the grader will not show you the actual test you program have failed on (though it does show you the test if your program have failed on one of the first few tests; this is done to help you to get the input/output format right).

4.7 Debugging and Stress Testing Your Program

If your program failed, you will need to debug it. Most probably, you didn't follow some of our suggestions from the section 4.5. See the readings and screencasts from the first week to learn about debugging your program: link.

You are almost guaranteed to find a bug in your program using stress testing, because the way these programming assignments and tests for them are prepared follows the same process: small manual tests, tests for edge cases, tests for large numbers and integer overflow, big tests for time limit and memory limit checking, random test generation. Also, implementation of wrong solutions which we expect to see and stress testing against them to add tests specifically against those wrong solutions.

Go ahead, and we hope you pass the assignment soon!

5 Frequently Asked Questions

5.1 I submit the program, but nothing happens. Why?

You need to create submission and upload the file with your solution in one of the programming languages `C`, `C++`, `Java`, or `Python` (see Subsections 4.3 and 4.4). Make sure that after uploading the file with your solution you press on the blue "Submit" button in 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 is shown to you.

5.2 I submit the solution only for one problem, but all the problems in the assignment are graded. Why?

Each time you submit any solution, the last uploaded solution for each problem is tested. Don't worry: this doesn't affect your score even if the submissions for the other problems are wrong. As soon as you pass the sufficient number of problems in the assignment (see in the pdf with instructions), you pass the assignment. After that, you can improve your result if you successfully pass more problems from the assignment. We recommend working on one problem at a time, checking whether your solution for any given problem passes in the system as soon as you are confident in it. However, it is better to test it first, please refer to the reading about stress testing: link.

5.3 What are the possible grading outcomes, and how to read them?

Your solution may either pass or not. To pass, it must work without crashing and return the correct answers on all the test cases we prepared for you, and do so under the time limit and memory limit constraints specified in the problem statement. If your solution passes, you get the corresponding feedback "Good job!" and get a point for the problem. If your solution fails, it can be because it crashes, returns wrong answer, works for too long or uses too much memory for some test case. The feedback will contain the number of the test case on which your solution fails 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 test number 1 to the test with the biggest number.

Here are the possible outcomes:

`Good job!` Hurrah! Your solution passed, and you get a point!

`Wrong answer.` Your solution has output incorrect answer for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data, the output of your program and the correct answer. Otherwise, you won't know the input, the output, and the correct answer. Check that you consider all the cases correctly, avoid integer overflow, output the required white space, 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. See this reading on testing: link.

`Time limit exceeded.` Your solution worked longer than the allowed time limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data and the correct answer. Otherwise, you won't know the input and the correct answer. Check again that your algorithm has good enough running time estimate. Test your program locally on the test of maximum size allowed by the problem statement and see how long it works. Check that your program doesn't wait for some input from the user which makes it to wait forever. See this reading on testing: link.

`Memory limit exceeded.` Your solution used more than the allowed memory limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1,

you will also see the input data and the correct answer. Otherwise, you won't know the input and the correct answer. Estimate the amount of memory that your program is going to use in the worst case and check that it is less than the memory limit. Check that you don't create too large arrays or data structures. 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 test of maximum size allowed by the problem statement and look at its memory consumption in the system.

`Cannot check answer. Perhaps output format is wrong.` This happens when you output something completely different than expected. For example, you are required to output word "Yes" or "No", but you output number 1 or 0, or vice versa. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (this is not allowed, so please follow exactly the 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 division by zero, accessing memory outside of the array bounds, using uninitialized variables, too deep recursion that triggers stack overflow, sorting with 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 compilers and the same compiler options as we do. Try different testing techniques from this reading: link.

`Internal error: exception...` Most probably, you submitted a compiled program instead of a source code.

`Grading failed.` Something very wrong happened with the system. Contact Coursera for help or write in the forums to let us know.

5.4 How to understand why my program fails and to fix it?

If your program works incorrectly, it gets a feedback from the grader. For the Programming Assignment 1, when your solution fails, you will see the input data, the correct answer and the output of your program in case it didn't crash, finished under the time limit and memory limit constraints. If the program crashed, worked too long or used too much memory, the system stops it, so you won't see the output of your program or will see just part of the whole output. We show you all this information so that you get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail.

However, in the following Programming Assignments throughout the Specialization you will only get so much information for the test cases from the problem statement. For the next tests you will only get the result: passed, time limit exceeded, memory limit exceeded, wrong answer, wrong output format or some form of crash. We hide the test cases, because it is crucial for you to learn to test and fix your program even without knowing exactly the test on which it fails. In the real life, often there will be no or only partial information about the failure of your program or service. You will need to find the failing test case yourself. Stress testing is one powerful technique that allows you to do that. You should apply it after using the other testing techniques covered in this reading.

5.5 Why do you hide the test on which my program fails?

Often beginner programmers think by default that their programs work. Experienced programmers know, however, that their programs almost never work initially. Everyone who wants to become a better programmer needs to go through this realization.

When you are sure that your program works by default, you just throw a few random test cases against it, and if the answers look reasonable, you consider your work done. However, mostly this is not enough. To

make one's programs work, one must test them really well. Sometimes, the programs still don't work although you tried really hard to test them, and you need to be both skilled and creative to fix your bugs. Solutions to algorithmic problems are one of the hardest to implement correctly. That's why in this Specialization you will gain this important experience which will be invaluable in the future when you write programs which you really need to get right.

It is crucial for you to learn to test and fix your programs yourself. In the real life, often there will be no or only partial information about the failure of your program or service. Still, you will have to reproduce the failure to fix it (or just guess what it is, but that's rare, and you will still need to reproduce the failure to make sure you have really fixed it). When you solve algorithmic problems, it is very frequent to make subtle mistakes. That's why you should apply the testing techniques described in this reading to find the failing test case and fix your program.

5.6 My solution does not pass the tests? May I post it in the forum and ask for a help?

No, please do not post any solutions in 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). Recall the third item of the Coursera 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). This includes both solutions written by me, as well as any solutions provided by the course staff or others" (link).

5.7 My implementation always fails in the grader, though I already tested and stress tested it a lot. Would not it be better if you give 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, you always learn from your mistakes.

The process of trying to invent new test cases that might fail your program and proving them wrong is often enlightening. This thinking about the invariants which you expect your loops, ifs, etc. to keep and proving them wrong (or right) 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. 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, once again, it is important to be able to locate 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 a lot (considered all corner cases that you can imagine, constructed a set of manual test cases, applied stress testing), but your program still fails and you are stuck, 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 when writing such a post you will realize that you missed some corner cases!) and only then asking other learners to give you more ideas for tests cases.