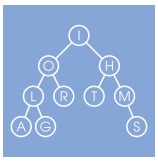


Module: Assembling Genomes Using de Bruijn Graphs (Week 2 out of 3)
Course: Genome Assembly Programming Challenge (Course 6 out of 6)
Specialization: Data Structures and Algorithms

Programming Assignment 3: Genome Assembly Faces Real Sequencing Data

Revision: June 24, 2018

Introduction

Welcome to the third programming assignment of the Genome Assembly Programming Challenge! In this assignment, you will face practical challenges introduced by quirks in modern sequencing technologies and practice using algorithmic techniques that have been devised to address these challenges.

Passing Criteria: 3 out of 6

Passing this programming assignment requires passing at least 3 out of 6 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: Finding a Circulation in a Network

Problem Introduction

In a circulation problem, one is given a directed graph $G(V, E)$ where each edge $e \in E$ is assigned a lower bound l_e and a capacity c_e such that $0 \le l_e \le c_e$. The goal is to check whether it is possible to assign a flow f_e to each edge so as to satisfy the following two conditions:

- capacity conditions: for each edge $e \in E$,

$$l_e \le f_e \le c_e\,;$$

- conservation of flow: for each vertex $v \in V$,

$$\sum_{(u,v)\in E} f_{(u,v)} = \sum_{(v,w)\in E} f_{(v,w)}\,.$$

The problem is similar to the maximum flow problem, but in the circulation problem we force each edge e to transfer at least l_e units of flow. Also, there is no source that produces a flow and there is no sink that absorbs a flow. For this reason, we do not maximize a flow in this problem, but simply check whether there is a flow satisfying the two types of constraints above.

Problem Description

Task. Given a network with lower bounds and capacities on edges, find a circulation if it exists.

Input Format. The first line contains integers n and m — the number of vertices and the number of edges, respectively. Each of the following m lines specifies an edge in the format "`u v l c`": the edge (u, v) has a lower bound l and a capacity c. (As usual, we assume that the vertices of the network are $\{1, 2, \ldots, n\}$.) The network does not contain self-loops, but may contain parallel edges.

Constraints. $2 \le n \le 40$; $1 \le m \le 1\,600$; $u \ne v$; $0 \le l \le c \le 50$.

Output Format. If there exists a circulation, output `YES` in the first line. In each of the next m lines output the value of the flow along an edge (assuming the same order of edges as in the input). If there is no circulation, output `NO`.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	3	3	4.5	15	4.5	6	15	15	9

Memory Limit. 512MB.

Sample 1.

Input:

```
3 2
1 2 0 3
2 3 0 3
```

Output:

```
YES
0
0
```

Sample 2.

Input:

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

Output:

```
YES
2
2
2
```

Sample 3.

Input:

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

Output:

```
NO
```

What To Do

Reduce this problem to the maximum flow problem.

Need Help?

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

2 Dataset Problem: Selecting the Optimal k-mer Size

In dataset problems, you solution is going to be tested against a *single* dataset as opposed to problems in the previous classes in this specialization. For this reason, there is no Constraints section in the problem description below. The sample section shows just a similar dataset. Your program is not going to be tested on this dataset.

Problem Introduction

As you may recall, because we are not guaranteed to be given every possible read (i.e., the substring of Genome beginning at every possible position), we cannot expect to use our reads directly and have our de Bruijn graph have an Eulerian Cycle. However, our way of combatting this is to choose some $k < ReadLength$ to break our reads down, and hopefully, for some value of k, the fragments will be small enough for there to exist an Eulerian Cycle in the de Bruijn graph created from the fragments, but not so small that the de Bruijn graph becomes too convoluted.

Problem Description

Task. Given a list of error-free reads, return an integer k such that, when a de Bruijn graph is created from the k-length fragments of the reads, the de Bruijn graph has a single possible Eulerian Cycle.

Dataset. The input consist of 400 reads of length 100, each on a separate line. The reads contain no sequencing errors. Note that you are **not** given the 100-mer composition of the genome (i.e., some 100-mers may be missing).

Output. A single integer k on one line.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	3	3	4.5	15	4.5	6	15	15	9

Memory Limit. 512MB.

Sample 1.

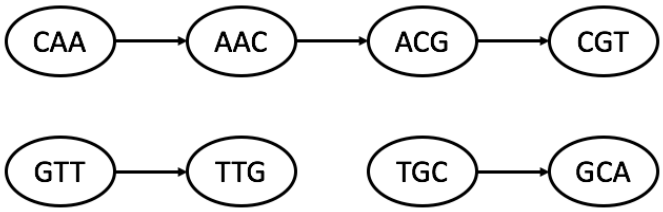
Input:

```
AACG
ACGT
CAAC
GTTG
TGCA
```

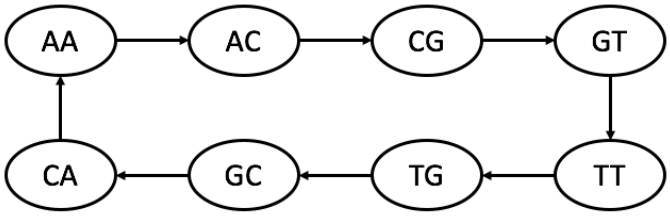
Output:

```
3
```

Below is the de Bruijn graph constructed from the input reads of length $k = 4$ (notice that there is no cycle in the de Bruijn graph because of imperfect coverage):



However, if we fragment the reads into k-mers of length $k = 3$, we'll see that the resulting de Bruijn graph does indeed have a cycle with which we would be able to reconstruct the genome:



Need Help?

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

3 Dataset Problem: Bubble Detection

In dataset problems, you solution is going to be tested against a *single* dataset as opposed to problems in the previous classes in this specialization. For this reason, there is no Constraints section in the problem description below. The sample section shows just a similar dataset. Your program is not going to be tested on this dataset.

Problem Introduction

We define a directed path in a directed graph as *short* if its length (in number of edges) does not exceed the bubble length threshold t. A path is *non-overlapping* if it traverses each of its nodes exactly once. Two paths between nodes v and w are called *disjoint* if they do not share any nodes (except for v and w). Given two distinct nodes v and w, a (v, w)-bubble is defined as a pair of short non-overlapping disjoint paths between v and w.

In this challenge, given a bubble length threshold of t, you will be given the task of detecting (v, w)-bubbles for all possible v and w in the de Bruijn graph generated from a simulated error-prone sequencing dataset.

Problem Description

Task. Given a list of error-prone reads and two integers, k and t, construct a de Bruijn graph from the k-mers created from the reads and perform the task of bubble detection on this de Bruijn graph with a path length threshold of t.

Dataset. The first line of the input contains two integers, k and t, separated by a single space. Each subsequent line of the input contains a single *read*. The reads are given to you in alphabetical order because their true order is hidden from you. Each read is 100 nucleotides long and contains a single sequencing error (i.e., one mismatch per read) in order to simulate the 1% error rate of Illumina sequencing machines. Note that you are not given the 100-mer composition of the genome (i.e., some 100-mers may be missing).

Output. A single integer (the number of (v, w)-bubbles) on one line.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	20	20	30	100	30	40	100	100	60

Memory Limit. 512MB.

Sample 1.

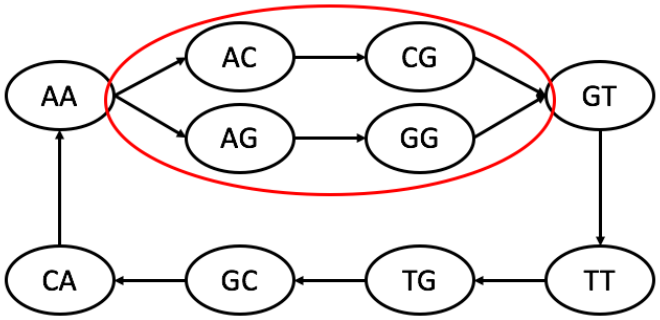
Input:

```
3 3
AACG
AAGG
ACGT
AGGT
CGTT
GCAA
GGTT
GTTG
TGCA
TTGC
```

Output:

```
1
```

Below is the de Bruijn graph constructed from the input reads after they have been broken down into 3-mers (the single (v, w)-bubble has been circled in red):



Need Help?

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

4 Dataset Problem: Tip Removal

In dataset problems, you solution is going to be tested against a *single* dataset as opposed to problems in the previous classes in this specialization. For this reason, there is no Constraints section in the problem description below. The sample section shows just a similar dataset. Your program is not going to be tested on this dataset.

Problem Introduction

Tips are error-prone ends of the reads that do not form a bubble but instead form a path starting in a vertex without incoming edges or ending in a vertex without outgoing edges in the de Bruijn graph. Tips should be removed iteratively because removing a tip can expose another tip. In this challenge, you will be given the task of removing tips from the de Bruijn graph generated from a simulated error-prone sequencing dataset.

Problem Description

Task. Given a list of error-prone reads, construct a de Bruijn graph from the 15-mers created from the reads and perform the task of tip removal on this de Bruijn graph.

Dataset. The input consist of 400 reads of length 100, each on a separate line. The reads are given to you in alphabetical order because their true order is hidden from you. Each read is 100 nucleotides long and contains a single sequencing error (i.e., one mismatch per read) in order to simulate the 1% error rate of Illumina sequencing machines. Note that you are not given the 100-mer composition of the genome (i.e., some 100-mers may be missing).

Output. A single integer (the number of edges removed during the Tip Removal process) on one line.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	20	20	30	100	30	40	100	100	60

Memory Limit. 512MB.

Sample 1.

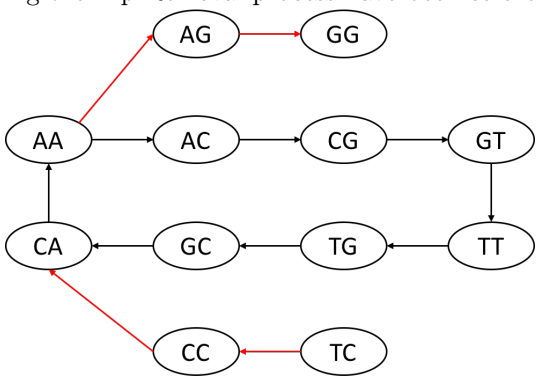
Input:

```
AACG
AAGG
ACGT
CAAC
CGTT
GCAA
GTTG
TCCA
TGCA
TTGC
```

Output:

```
4
```

Below is the de Bruijn graph constructed from the input reads after they have been broken down into 3-mers (the edges removed during the Tip Removal process have been colored red):



Need Help?

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

5 Dataset Problem: Assembling the phi X174 Genome from Error-Prone Reads using de Bruijn Graphs

In dataset problems, you solution is going to be tested against a *single* dataset as opposed to problems in the previous classes in this specialization. For this reason, there is no Constraints section in the problem description below. The sample section shows just a similar dataset. Your program is not going to be tested on this dataset.

Problem Introduction

In this challenge, you will be given the task of performing Genome Assembly on a simulated error-prone sequencing dataset using de Bruijn graphs.

Problem Description

Task. Given a list of error-prone reads, perform the task of Genome Assembly using de Bruijn graphs and return the circular genome from which they came. Break the reads into fragments of length $k = 15$ before constructing the de Bruijn graph, remove tips, and handle bubbles.

Dataset. Each line of the input contains a single *read*. The reads are given to you in alphabetical order because their true order is hidden from you. Each read is 100 nucleotides long and contains a single sequencing error (i.e., one mismatch per read) in order to simulate the 1% error rate of Illumina sequencing machines. Note that you are not given the 100-mer composition of the genome (i.e., some 100-mers may be missing).

Output. Output the assembled genome on a single line.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	10	10	15	50	15	20	50	50	30

Memory Limit. 512MB.

Sample 1.

Input:

```
AAC
ACG
GAA
GTT
TCG
```

Output:

```
ACGTTCGA
```

In this sample, the circular genome is ACGTTCGA (the sample output), and the reads were all generated from the genome: ACGTTCGA, ACGTTCGA, ACGTTCGA, ACGTTCGA, and ACGTTCGA. Note that we did not put mismatches in these reads because they are extremely short, so introducing the sequencing errors on such short reads would only make the solution difficult to see. On the real dataset, however, each read (of length 100) will have exactly one error.

Need Help?

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

6 Final Project: Implementing an Assembler

Problem Introduction

Now that you have solved the individual components of a genome assembler, it is time to put all of the pieces together. Using the de Bruijn graph approach to genome assembly as well as using all of the various error-correction techniques you have learned about, your task is to implement an assembler that takes as input reads or read-pairs and outputs the assembled contigs.

Note that, for read-pairs, there is typically a (roughly) known distance between the two reads: the distance is known by design of the sequencing experiment, but further explanation is out of the scope of this project. Also, note that we say "roughly" because, although biologists *attempt* to keep the distance of all read-pairs constant, there is slight deviation in distances. For example, if you were to design an experiment where you expect all read-pairs to have a distance of 400 bases between the two reads of the pair, in reality, you would observe some with distance 400, some with 401, some with 399, etc.

Also note that, in real sequencing experiments, each base of a read has a quality score associated with it, which effectively tells us how confident the sequencing machine is in calling the given base. Further explanation is out of the scope of this project, but feel free to read about the FASTQ file format and Phred-scaled quality scores. Basically, because of limitations in the sequencing technologies, reads tend to drop off in quality closer to their 3' ends. As a result, it is common practice for biologists to truncate reads once the quality drops off too far so we only attempt to assemble high quality segments of sequence data. As a result, when dealing with real datasets, the reads that are originally sequenced are all of equal length, but the truncated reads used in the assembly typically vary in length.

Of course, there are various metrics used to assess the quality of a genome assembly. One such metric is N50, the length for which the collection of all contigs of that length or longer covers at least 50% of assembly length. Note that the N50 metric does not depend on any extranneous information: it only depends on measurements of the alignment itself. A slightly better metric, which can only be used if the reference genome is known, is NG50, which is identical to N50, except instead of looking at coverage of at least 50% of the assembly length, we look at coverage of at least 50% of the reference genome. The last metric we will discuss is NGA50 (which is the metric we use to assess the quality of your assembler in this challenge), which is identical to NG50, except instead of simply looking at contigs with respect to the reference genome, we look at aligned blocks of the assembled genome against the reference genome.

In this challenge, we will be feeding your assembler multiple datasets: error-free as well as error-prone, single reads as well as read-pairs, and even a real WGS dataset. We have run each dataset through the SPAdes assembler and used QUAST to obtain NGA50 values for each assembly. We grade your assembler based on how well the NGA50 of the assembly generated by your assembler compares to that generated by SPAdes.

Problem Description

Task. Given a list of reads or read-pairs, generate all contigs in their assembly using the de Bruijn graph approach as well as all of the error-correction techniques you have learned.

Input Format. In this challenge, we generate t reads by randomly selecting k-mers from the genome. In the case of read-pairs, we randomly select the first read (just as with single reads), and then we randomly choose the desired distance from a uniform distribution centered around d, and using this distance, we choose the second read in the pair.

Some datasets may be error-free, and some may be error-prone. For the error-prone datasets, we randomly choose a single site to randomly mutate in each read.

The first line of the input is an integer t representing the number of reads or read-pairs that will follow (i.e., how many lines of input you will have to parse following the first line). Each subsequent line of the input contains a single read or a single read-pair. If the dataset constains single reads (i.e., not read-pairs), each line of sequence data will be a single k-mer. If the dataset contains read-pairs, each line of sequence data will have three parts: the first k-mer, the second k-mer, and d (the distance we *expect* to occur between the two reads, i.e., the d used as the center of the uniform distribution from which the true distance between the two reads was sampled), and these three components will be separated by the | symbol. In other words, each line will look like READ1|READ2|d (but as mentioned, note that the d listed might not be the actual distance between the two reads).

For the single-read datasets generated from N. deltocephalinicola, you will be given almost $t = 34\,000$ reads, each of length $k = 100$. For the single-read datasets generated from E. coli X, you will be given almost $t = 1\,400\,000$ reads, each of length $k = 100$. For the read-pair datasets generated from E. coli X, you will be given almost $t = 700\,000$ read-pairs, where each read is of length $k = 100$. For the real dataset N. deltocephalinicola, you will be given almost $t = 19\,679$ read-pairs, where the reads vary in length.

Output. Output all contigs in the assembly constructed from the reads (or read-pairs). Format your output in the FASTA format. The identifiers of your FASTA file do not matter.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	100	100	150	500	150	200	500	500	300

Memory Limit. 512MB.

Sample 1.

Input:

```
8
ATG
ATG
TGT
TGG
CAT
GGA
GAT
AGA
```

Output:

```
>CONTIG1
AGA
>CONTIG2
ATG
>CONTIG3
ATG
>CONTIG4
CAT
>CONTIG5
GAT
>CONTIG6
TGGA
>CONTIG7
TGT
```

7 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.

7.1 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.

7.2 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.

7.3 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.

7.4 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
```

7.5 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.

7.6 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 7.4) 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).

7.7 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 7.5. 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!

8 Frequently Asked Questions

8.1 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 7.3 and 7.4). 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.

8.2 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.

8.3 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.

8.4 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.

8.5 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.

8.6 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).

8.7 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.