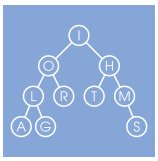


Module: NP-completeness (Week 3 out of 5)
Course: Advanced Algorithms and Complexity (Course 5 out of 6)
Specialization: Data Structures and Algorithms

Programming Assignment 3: NP-completeness

Revision: January 11, 2019

Introduction

Welcome to your third programming assignment of the Advanced Algorithms and Complexity class! In this programming assignment, you will be practicing reducing hard real-world problems to SAT problem, which can in turn often be solved efficiently in practice using specialized programs called SAT-solvers.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Reduce real-world problems to instances of classical NP-complete problems.
2. Design and implement efficient algorithms to reduce the following computational problems to SAT:
   (a) assigning frequencies to the cells of a GSM network;
   (b) determine whether it is possible to leave no signs of a party in the apartment;
   (c) determine whether there is a way to allocate advertising budget given a set of constraints.

Passing Criteria: 2 out of 3

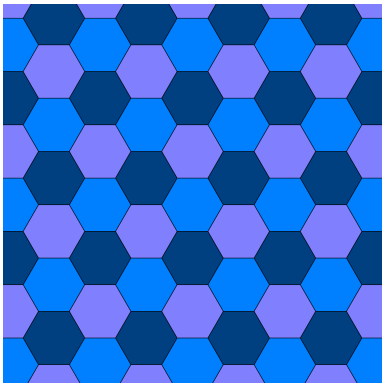
Passing this programming assignment requires passing at least 2 out of 3 code problems from this assignment. In turn, passing a code problem requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Problem: Assign Frequencies to the Cells of a GSM Network

Problem Introduction

In this problem, you will learn to reduce the real-world problem about assigning frequencies to the transmitting towers of the cells in a GSM network to a problem of proper coloring a graph into 3 colors. Then you will design and implement an algorithm to reduce this problem to an instance of SAT.



Problem Description

Task. GSM network is a type of infrastructure used for communication via mobile phones. It includes transmitting towers scattered around the area which operate in different frequencies. Typically there is one tower in the center of each hexagon called "cell" on the grid above — hence the name "cell phone". A cell phone looks for towers in the neighborhood and decides which one to use based on strength of signal and other properties. For a phone to distinguish among a few closest towers, the frequencies of the neighboring towers must be different. You are working on a plan of GSM network for mobile, and you have a restriction that you've only got 3 different frequencies from the government which you can use in your towers. You know which pairs of the towers are neighbors, and for all such pairs the towers in the pair must use different frequencies. You need to determine whether it is possible to assign frequencies to towers and satisfy these restrictions.

This is equivalent to a classical graph coloring problem: in other words, you are given a graph, and you need to color its vertices into 3 different colors, so that any two vertices connected by an edge need to be of different colors. Colors correspond to frequencies, vertices correspond to cells, and edges connect neighboring cells. Graph coloring is an NP-complete problem, so we don't currently know an efficient solution to it, and you need to reduce it to an instance of SAT problem which, although it is NP-complete, can often be solved efficiently in practice using special programs called SAT-solvers.

Input Format. The first line of the input contains integers n and m — the number of vertices and edges in the graph. The vertices are numbered from 1 through n. Each of the next m lines contains two integers u and v — the numbers of vertices connected by an edge. It is guaranteed that a vertex cannot be connected to itself by an edge.

Constraints. $2 \le n \le 500$; $1 \le m \le 1000$; $1 \le u, v \le n$; $u \ne v$.

Output Format. You need to output a boolean formula in the conjunctive normal form (CNF) in a specific format. If it is possible to color the vertices of the input graph in 3 colors such that any two vertices connected by an edge are of different colors, the formula must be satisfiable. Otherwise, the formula must be unsatisfiable. The number of variables in the formula must be at least 1 and at most 3000. The number of clauses must be at least 1 and at most 5000.

On the first line, output integers C and V — the number of clauses in the formula and the number of variables respectively. On each of the next C lines, output a description of a single clause. Each clause has a form $(x_4\ OR\ \overline{x_1}\ OR\ x_8)$. For a clause with k terms (in the example, $k = 3$ for x_4, x_1 and x_8), output first those k terms and then number 0 in the end (in the example, output "4 −1 8 0"). Output each term as integer number. Output variables $x_1, x_2, \ldots, x_V$ as numbers $1, 2, \ldots, V$ respectively. Output

negations of variables $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_V}$ as numbers $-1, -2, \ldots, -V$ respectively. Each number other than the last one in each line must be a non-zero integer between $-V$ and V where V is the total number of variables specified in the first line of the output. Ensure that $1 \le C \le 5000$ and $1 \le V \le 3000$.

See the examples below for further clarification of the output format.

If there are many different formulas that satisfy the requirements above, you can output any one of them.

Note that your formula will be checked internally by the grader using a SAT-solver. Although SAT-solvers often solve instances of SAT of the given size very fast, it cannot be guaranteed. If you submit a formula which cannot be resolved by the SAT-solver we use under a reasonable time limit, the grader will timeout, and the problem won't pass. We guarantee that there are solutions of this problem that output formulas which are resolved almost instantly by the SAT-solver used. However, don't try to intentionally break the system by submitting very complex SAT instances, because the problem still won't pass, and you will violate Coursera Honor Code by doing that.

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.

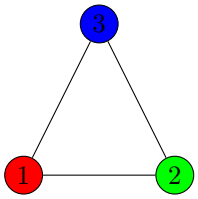
Sample 1.

Input:

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

Output:

```
1 1
1 -1 0
```



The input graph has just 3 vertices, so of course they all can be colored in different colors using only 3 colors. That's why we need to output a satisfiable formula. The formula in the output uses just 1 variable x_1 and 1 clause, and the only clause is $(x_1\ OR\ \overline{x_1})$ which is, of course, satisfiable: for any value of x_1, the boolean value of the formula is true. Note that you could output another satisfiable formula, like x_1 or $(x_1\ OR\ x_2\ OR\ x_3)\ AND\ (x_1\ OR\ x_2)$, or one of many others.

Sample 2.

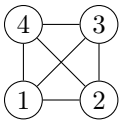
Input:

```
4 6
1 2
1 3
1 4
2 3
2 4
3 4
```

Output:

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

Explanation:



The input graph has 4 vertices, and each pair of them is connected by an edge. In a proper coloring, all these vertices must be of different colors, but we have only 3 different colors, so it is impossible. Thus, we need to output an unsatisfiable formula. The formula in the output has 2 clauses with one variable, it is $(x_1)AND(\overline{x_1})$. Note that you could output another formula, like $(x_1\ OR\ x_2)\ AND\ (\overline{x_1})\ AND\ (\overline{x_2})$, or one of many other unsatisfiable formulas.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a procedure that outputs a fixed satisfiable formula. You need to change the main procedure to implement some reduction of the graph coloring problem to SAT problem if you're using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `gsm_network`

What To Do

Create a separate variable x_{ij} for each vertex i of the initial graph and each possible color j, $1 \le j \le 3$ which means "vertex i has color j". Think how to write down conditions like "each vertex has to be colored by some color" and "vertices connected by an edge must have different colors" with clauses of CNF using these variables.

Need Help?

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

2 Problem: Cleaning the Apartment

Problem Introduction

In this problem, you will learn to determine whether it is possible to clean an apartment after a party without leaving any traces of the party. You will learn how to reduce it to the classic Hamiltonian Path problem, and then you will design and implement an efficient algorithm to reduce it to SAT.

Problem Description

Task. You've just had a huge party in your parents' house, and they are returning tomorrow. You need to not only clean the apartment, but leave no trace of the party. To do that, you need to clean all the rooms in some order. After finishing a thorough cleaning of some room, you cannot return to it anymore: you are afraid you'll ruin everything accidentally and will have to start over. So, you need to move from room to room, visit each room exactly once and clean it. You can only move from a room to the neighboring rooms. You want to determine whether this is possible at all.

This can be reduced to a classic Hamiltonian Path problem: given a graph, determine whether there is a route visiting each vertex exactly once. Rooms are vertices of the graph, and neighboring rooms are connected by edges. Hamiltonian Path problem is NP-complete, so we don't know an efficient algorithm to solve it. You need to reduce it to SAT, so that it can be solved efficiently by a SAT-solver.

Input Format. The first line contains two integers n and m — the number of rooms and the number of corridors connecting the rooms respectively. Each of the next m lines contains two integers u and v describing the corridor going from room u to room v. The corridors are two-way, that is, you can go both from u to v and from v to u. No two corridors have a common part, that is, every corridor only allows you to go from one room to one other room. Of course, no corridor connects a room to itself. Note that a corridor from u to v can be listed several times, and there can be listed both a corridor from u to v and a corridor from v to u.

Constraints. $1 \le n \le 30$; $0 \le m \le \frac{n(n-1)}{2}$; $1 \le u, v \le n$.

Output Format. You need to output a boolean formula in the CNF form in a specific format. If it is possible to go through all the rooms and visit each one exactly once to clean it, the formula must be satisfiable. Otherwise, the formula must be unsatisfiable. The sum of the numbers of variables used in each clause of the formula must not exceed 120 000.

On the first line, output integers C and V — the number of clauses in the formula and the number of variables respectively. On each of the next C lines, output a description of a single clause. Each clause has a form $(x_4\ OR\ \overline{x_1}\ OR\ x_8)$. For a clause with k terms (in the example, $k = 3$ for x_4, x_1 and x_8), output first those k terms and then number 0 in the end (in the example, output "4 -1 8 0"). Output each term as integer number. Output variables $x_1, x_2, \ldots, x_V$ as numbers $1, 2, \ldots, V$ respectively. Output negations of variables $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_V}$ as numbers $-1, -2, \ldots, -V$ respectively. Each number other than the last one in each line must be a non-zero integer between $-V$ and V where V is the total number of variables specified in the first line of the output. Ensure that the total number of non-zero integers in the C lines describing the clauses is at most 120 000.

See the examples below for further clarification of the output format.

If there are many different formulas that satisfy the requirements above, you can output any one of them.

Note that your formula will be checked internally by the grader using a SAT-solver. Although SAT-solvers often solve instances of SAT of the given size very fast, it cannot be guaranteed. If you submit a formula which cannot be resolved by the SAT-solver we use under a reasonable time limit, the grader will timeout, and the problem won't pass. We guarantee that there are solutions of this problem that output formulas which are resolved almost instantly by the SAT-solver used. However, don't try to intentionally break the system by submitting very complex SAT instances, because the problem still won't pass, and you will violate Coursera Honor Code by doing that.

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 2
2 3
3 5
4 5
```

Output:

```
1 1
1 -1 0
```

Explanation:

There is a Hamiltonian path $1-2-3-5-4$, so we need to output a satisfiable formula. The formula in the output uses just 1 variable x_1 and 1 clause, and the only clause is $(x_1\ OR\ \overline{x_1})$ which is, of course, satisfiable: for any value of x_1, the boolean value of the formula is true. Note that you could output another satisfiable formula, like x_1 or $(x_1\ OR\ x_2\ OR\ x_3)\ AND\ (x_1\ OR\ x_2)$, or one of many others.

Sample 2.

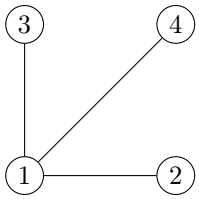
Input:

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

Output:

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

Explanation:



There is no way to visit each room exactly once: either we don't visit one of the rooms 2, 3 or 4, or we visit room 1 at least twice. Thus, we need to output an unsatisfiable formula. The formula in the output has 2 clauses with one variable, it is $(x_1)AND(\overline{x_1})$. Note that you could output another formula, like $(x_1\ OR\ x_2)\ AND\ (\overline{x_1})\ AND\ (\overline{x_2})$, or one of many other unsatisfiable formulas.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a procedure that outputs a fixed satisfiable formula. You need to change the main procedure to implement some reduction of the Hamiltonian path problem to SAT if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `cleaning_apartment`

What To Do

Create a separate variable x_{ij} for each vertex i and each position in the Hamiltonian path j. x_{ij} is true if vertex i is at the position j of Hamiltonian path.

Note that it is usually difficult to predict the running time of a SAT-solver on a CNF formula that you generate. In many cases, adding redundant clauses to a formula helps a SAT-solver to find a satisfying assignment (or to report that none exists) faster. For this reason, it is recommended that you include clauses describing all of the following constraints:

- Each vertex belongs to a path.
- Each vertex appears just once in a path. (Note that this restriction is already redundant: since we know that a path consists of n vertices and each vertex of a graph appears in it, it cannot be the case that some vertex appears more than once. Still, by adding such constraints you usually help a SAT-solver. Roughly, a solver does not need to spend time for figuring out that each vertex appears in a path just once; instead, it is given this information from the very beginning.)
- Each position in a path is occupied by some vertex.
- No two vertices occupy the same position of a path.
- Two successive vertices on a path must be connected by an edge. (In other words, if there is no edge $\{i, j\}$ in E, then for any k, it cannot be the case that both x_{ik} and $x_{j(k+1)}$ are True.)

Need Help?

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

3 Advanced Problem: Advertisement Budget Allocation

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 the previous programming assignment, you worked for an online advertisement system. In this programming assignment, you'll work for a big company that uses advertising to promote itself. You will need to determine whether it is possible to allocate advertising budget and satisfy all the constraints. You will learn how to reduce this problem to a particular type of Integer Linear Programming problem. Then you will design and implement an efficient algorithm to reduce this type of Integer Linear Programming to SAT.

Problem Description

Task. The marketing department of your big company has many subdepartments which control advertising on TV, radio, web search, contextual advertising, mobile advertising, etc. Each of them has prepared their advertising campaign plan, and of course you don't have enough budget to cover all of their proposals. You don't have enough time to go thoroughly through each subdepartment's proposals and cut them, because you need to set the budget for the next year tomorrow. You decide that you will either approve or decline each of the proposals as a whole.

There is a bunch of constraints you face. For example, your total advertising budget is limited. Also, you have some contracts with advertising agencies for some of the advertisement types that oblige you to spend at least some fixed budget on that kind of advertising, or you'll see huge penalties, so you'd better spend it. Also, there are different company policies that can be of the form that you spend at least 10% of your total advertising spend on mobile advertising to promote yourself in this new channel, or that you spend at least \$1M a month on TV advertisement, so that people always remember your brand. All of these constraints can be rewritten as an Integer Linear Programming: for each subdepartment i, denote by x_i boolean variable that corresponds to whether you will accept or decline the proposal of that subdepartment. Then each constraint can be written as a linear inequality. For example, $\sum_{i=1}^{n} spend_i \cdot x_i \leq TotalBudget$ is the inequality to ensure your total budget is enough to accept all the selected proposals. And $\sum_{i=1}^{n} spend_i \cdot x_i \leq 10 \cdot mobile$ corresponds to the fact that mobile advertisement budget is at least 10% of the total spending.

You will be given the final Integer Linear Programming problem in the input, and you will need to reduce it to SAT. **It is guaranteed that there will be at most 3 different variables with non-zero coefficients in each inequality of this Integer Linear Programming problem.**

Input Format. The first line contains two integers n and m — the number of inequalities and the number of variables. The next n lines contain the description of $n \times m$ matrix A with coefficients of inequalities (each of the n lines contains m integers, and at most 3 of them are non-zero), and the last line contains the description of the vector b (n integers) for the system of inequalities $Ax \leq b$. You need to determine whether there exists a binary vector x satisfying all those inequalities.

Constraints. $1 \le n, m \le 500$; $-100 \le A_{ij} \le 100$; $-1\,000\,000 \le b_i \le 1\,000\,000$.

Output Format. You need to output a boolean formula in the CNF form in a specific format. If it is possible to accept some of the proposals and decline all the others while satisfying all the constraints, the formula must be satisfiable. Otherwise, the formula must be unsatisfiable. The number of variables in the formula must not exceed 3000, and the number of clauses must not exceed 5000.

On the first line, output integers C and V — the number of clauses in the formula and the number of variables respectively. On each of the next C lines, output a description of a single clause. Each clause has a form $(x_4\ OR\ \overline{x_1}\ OR\ x_8)$. For a clause with k terms (in the example, $k = 3$ for x_4, x_1 and x_8), output first those k terms and then number 0 in the end (in the example, output "4 −1 8 0"). Output each term as integer number. Output variables $x_1, x_2, \ldots, x_V$ as numbers $1, 2, \ldots, V$ respectively. Output negations of variables $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_V}$ as numbers $-1, -2, \ldots, -V$ respectively. Each number other than the last one in each line must be a non-zero integer between $-V$ and V where V is the total number of variables specified in the first line of the output. Ensure that $1 \le C \le 5000$ and $1 \le V \le 3000$.

See the examples below for further clarification of the output format.

If there are many different formulas that satisfy the requirements above, you can output any one of them.

Note that your formula will be checked internally by the grader using a SAT-solver. Although SAT-solvers often solve instances of SAT of the given size very fast, it cannot be guaranteed. If you submit a formula which cannot be resolved by the SAT-solver we use under a reasonable time limit, the grader will timeout, and the problem won't pass. We guarantee that there are solutions of this problem that output formulas which are resolved almost instantly by the SAT-solver used. However, don't try to intentionally break the system by submitting very complex SAT instances, because the problem still won't pass, and you will violate Coursera Honor Code by doing that.

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:

```
2 3
5 2 3
-1 -1 -1
6 -2
```

Output:

```
1 1
1 -1 0
```

Explanation:
Here we have two inequalities: $5 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \le 6$ and $(-1) \cdot x_1 + (-1) \cdot x_2 + (-1) \cdot x_3 \le -2$. The binary vector $x_1 = 0, x_2 = 1, x_3 = 1$ satisfies all the inequalities, so we need to output a satisfiable formula. The formula in the output uses just one variable x_1 and one clause, and the only clause is $(x_1\ OR\ \overline{x_1})$ which is, of course, satisfiable: for any value of x_1, the boolean value of the formula is true. Note that you could output another satisfiable formula, like x_1 or $(x_1\ OR\ x_2\ OR\ x_3)\ AND\ (x_1\ OR\ x_2)$, or one of many others. You **do not** have to make sure that the formula you output is satisfied only by the binary vectors which satisfy the given system of inequalities, but the intended solution ensures that.

Sample 2.

Input:

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

Output:

```
1 1
1 -1 0
```

Explanation:
There are three inequalities of the form $x_i \leq 1$. Of course, any binary vector satisfies all those inequalities, so we need to output a satisfiable formula. The formula in the output is the same as in the previous example.

Sample 3.

Input:

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

Output:

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

Explanation:
There are two inequalities $x_1 \leq 0$ and $-x_1 \leq -1 \Rightarrow x_1 \geq 1$, but x_1 cannot be less than 0 and more than 1 simultaneously, so we need to output an unsatisfiable formula. The formula in the output has 2 clauses with one variable, it is $(x_1)AND(\overline{x_1})$. Note that you could output another formula, like $(x_1\ OR\ x_2)\ AND\ (\overline{x_1})\ AND\ (\overline{x_2})$, or one of many other unsatisfiable formulas.

Sample 4.

Input:

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

Output:

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

Explanation:
There are two inequalities: $x_1 + x_2 \leq 0$ and $(-1) \cdot x_2 + (-1) \cdot x_3 \leq -2 \Rightarrow x_2 + x_3 \geq 2$. From the first one, it follows for a binary vector x that $x_1 = x_2 = 0$. From the second one, it follows for a binary vector x that $x_2 = x_3 = 1$. But x_2 cannot be 0 and 1 simultaneously, so there is no binary vector x satisfying all the inequalities, and so we need to output an unsatisfiable formula. The formula in the output is the same as in the previous example.

Starter Files

The starter solutions for this problem read the data from the input, pass it to procedure that outputs a fixed satisfiable formula. You need to change the main procedure to implement some reduction of Integer Linear Programming problem to SAT if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `budget_allocation`

What To Do

In this case, you can use the binary vector x that you are looking for itself as the basis for the CNF formula. Think how to use the fact that there can be at most 3 non-zero coefficients in each inequality. Try thinking about assignments of all the corresponding variables that invalidate a particular inequality — how many different such assignments are there at most? You know that none of those assignments should hold if the binary vector x satisfies all the inequalities. Think how to write this condition compactly in the CNF form, then join those clauses for each inequality.

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.

References