| | |
|---|---|
| Module: | Linear Programming (Week 2 out of 5) |
| Course: | Advanced Algorithms and Complexity (Course 5 out of 6) |
| Specialization: | Data Structures and Algorithms |

# Programming Assignment 2: Linear Programming

Revision: April 6, 2018

## Introduction

Welcome to your second programming assignment of the Advanced Algorithms and Complexity class! In this programming assignment, you will be practicing reducing real-world problems to linear programming and implementing algorithms to solve them.

Recall that starting from this programming assignment, the grader will show you only the first few tests (see the questions 5.4 and 5.5 in the FAQ section).

## Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Implement Gaussian Elimination, brute-force algorithm for Linear Programming and Simplex Method.
2. Design and implement efficient algorithms for the following computational problems:
   (a) inferring energy values of ingredients from the menu with calorie counts;
   (b) optimal diet problem;
   (c) online advertisement allocation problem.

## 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: Infer Energy Values of Ingredients

### Problem Introduction

In this problem, you will apply Gaussian Elimination to infer the energy values of ingredients given a restaurant menu with calorie counts and ingredient lists provided for each item.

### Problem Description

**Task.** You're looking into a restaurant menu which shows for each dish the list of ingredients with amounts and the estimated total energy value in calories. You would like to find out the energy values of individual ingredients (then you will be able to estimate the total energy values of your favorite dishes).

**Input Format.** The first line of the input contains an integer $n$ — the number of dishes in the menu, and it happens so that the number of different ingredients is the same. Each of the next $n$ lines contains description $a_1, a_2, \ldots, a_n, E$ of a single menu item. $a_i$ is the amount of $i$-th ingredient in the dish, and $E$ is the estimated total energy value of the dish. If the ingredient is not used in the dish, the amount will be specified as $a_i = 0$; **beware that although the amount of any ingredient in any real menu would be positive, we will test that your algorithm works even for negative amounts** $a_i < 0$.

**Constraints.** $0 \leq n \leq 20$; $-1000 \leq a_i \leq 1000$.

**Output Format.** Output $n$ real numbers — for each ingredient, what is its energy value. These numbers can be non-integer, so output them with at least 3 digits after the decimal point.

Your output for a particular test input will be accepted if all the numbers in the output are considered correct. The amounts and energy values are of course approximate, and the computations in real numbers on a computer are not always precise, so each of the numbers in your output will be considered correct if either absolute or relative error is less than $10^{-2}$. That is, if the correct number is 5.245000, and you output 5.235001, your number will be considered correct, but 5.225500 will not be accepted. Also, if the correct number is 1001, and you output 1000, your answer will be considered correct, because the relative error will be less than $10^{-2}$, but if the correct answer is 0.1, and you output 0.05, your answer will not be accepted, because in this case both the absolute error (0.05) and the relative error (0.5) are more than $10^{-2}$. **Note that we ask you to output at least 3 digits after the decimal point, although we only require precision of $10^{-2}$, intentionally: if you output only 2 digits after the decimal point, your answer can be rejected while being correct because of the rounding issues. The easiest way to avoid this mistake is to output at least 3 digits after the decimal point.**

**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:

```
0
```

Output:

```

```

There are no dishes in the menu — you don't need to output anything in this case.

**Sample 2.**

Input:

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

Output:

```
1.000000 5.000000 4.000000 3.000000
```

Explanation:

This is an easy test. Each dish contains just one component, and the amount used is exactly 1, so the energy value of each ingredient is just equal to the energy value of the whole dish in which it is used.

**Sample 3.**

Input:

```
2
1 1 3
2 3 7
```

Output:

```
2.000000 1.000000
```

Explanation:

You can see that the numbers match: $1 \cdot 2.0 + 1 \cdot 1.0 = 3$ and $2 \cdot 2.0 + 3 \cdot 1.0 = 7$. **If you output** 1.994000 **and** 1.009000 **instead of** 2.000000 **and** 1.000000 **respectively, your answer will still be accepted, but don't forget to output at least 3 digits after the decimal point!**

**Sample 4.**

Input:

```
2
5 -5 -1
-1 -2 -1
```

Output:

```
0.200000 0.400000
```

Explanation:

**Beware that there will be tests with negative amounts and negative total energy values, although this is impossible in reality! Also note that the answers can be non-integer!** You can check that the numbers match: $5 \cdot 0.2 + (-5) \cdot 0.4 = -1$ and $(-1) \cdot 0.2 + (-2) \cdot 0.4 = -1$.

## Starter Files

The starter solutions for this problem read the data from the input, pass it to a blank procedure and output the result. They also contain some convenience functions and data structures. You need to change the main procedure to implement Gaussian Elimination if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `energy_values`

## What To Do

Implement the Gaussian Elimination algorithm from the lectures.

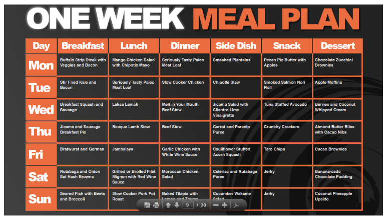
## Need Help?

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

## 2 Problem: Optimal Diet Problem

### Problem Introduction

In this problem, you will implement an algorithm for solving linear programming with only a few inequalities and apply it to determine the optimal diet.



ONE WEEK MEAL PLAN

| Day | Breakfast | Lunch | Dinner | Side Dish | Snack | Dessert |
|---|---|---|---|---|---|---|
| Mon | Buffalo Strip Steak with Veggies and Bacon | Mango Chicken Salad with Chipotle Mayo | Seriously Tasty Paleo Meat Loaf | Smashed Plantains | Pecan Pie Butter with Apples | Chocolate Zucchini Brownies |
| Tue | Stir Fried Kale and Bacon | Seriously Tasty Paleo Meat Loaf | Slow Cooker Chicken | Chipotle Slaw | Smoked Salmon Nori Roll | Apple Muffins |
| Wed | Breakfast Squash and Sausage | Laksa Lemak | Melt in Your Mouth Beef Stew | Jicama Salad with Cilantro Lime Vinaigrette | Tuna Stuffed Avocado | Berries and Coconut Whipped Cream |
| Thu | Jicama and Sausage Breakfast Pie | Basque Lamb Stew | Beef Stew | Carrot and Parsnip Puree | Crunchy Crackers | Almond Butter Bliss with Cacao Nibs |
| Fri | Bratwurst and German | Jambalaya | Garlic Chicken with White Wine Sauce | Cauliflower Stuffed Acorn Squash | Taro Chips | Cacao Brownies |
| Sat | Rutabaga and Onion Sat Hash Browns | Grilled or Broiled Filet Mignon with Red Wine Sauce | Moroccan Chicken Salad | Celeriac and Rutabaga Puree | Jerky | Banana-cado Chocolate Pudding |
| Sun | Seared Fish with Beets and Broccoli | Slow Cooker Pork Pot Roast | Baked Tilapia with Lemon and Thyme | Cucumber Wakame Salad | Jerky | Coconut Pineapple Upside |

### Problem Description

**Task.** You want to optimize your diet: that is, make sure that your diet satisfies all the recommendations of nutrition experts, but you also get maximum pleasure from your food and drinks. For each dish and drink you know all the nutrition facts, cost of one item, and an estimation of how much you like it. Your budget is limited, of course. The recommendations are of the form "total amount of calories consumed each day should be at least 1000" or "the amount of water you drink in liters should be at least twice the amount of food you eat in kilograms", and so on. You optimize the total pleasure which is the sum of pleasure you get from consuming each particular dish or drink, and that is proportional to the amount $amount_i$ of that dish or drink consumed.

The budget restriction and the nutrition recommendations can be converted into a system of linear inequalities like $\sum_{i=1}^{m} cost_i \cdot amount_i \leq Budget$, $amount_i \geq 1000$ and $amount_i - 2 \cdot amount_j \geq 0$, where $amount_i$ is the amount of $i$-th dish or drink consumed, $cost_i$ is the cost of one item of $i$-th dish or drink, and $Budget$ is your total budget for the diet. Of course, you can only eat a non-negative amount $amount_i$ of $i$-th item, so $amount_i \geq 0$. The goal to maximize total pleasure is reduced to the linear objective $\sum_{i=1}^{m} amount_i \cdot pleasure_i \to \max$ where $pleasure_i$ is the pleasure you get after consuming one unit of $i$-th dish or drink (some dishes like fish oil you don't like at all, so $pleasure_i$ can be negative). Combined, all this is a linear programming problem which you need to solve now.

**Input Format.** The first line of the input contains integers $n$ and $m$ — the number of restrictions on your diet and the number of all available dishes and drinks respectively. The next $n + 1$ lines contain the coefficients of the linear inequalities in the standard form $Ax \leq b$, where $x = amount$ is the vector of length $m$ with amounts of each ingredient, $A$ is the $n \times m$ matrix with coefficients of inequalities and $b$ is the vector with the right-hand side of each inequality. Specifically, $i$-th of the next $n$ lines contains $m$ integers $A_{i1}, A_{i2}, \ldots, A_{im}$, and the next line after those $n$ contains $n$ integers $b_1, b_2, \ldots, b_n$. These lines describe $n$ inequalities of the form $A_{i1} \cdot amount_1 + A_{i2} \cdot amount_2 + \cdots + A_{im} \cdot amount_m \leq b_i$. The last line of the input contains $m$ integers — the pleasure for consuming one item of each dish and drink $pleasure_1, pleasure_2, \ldots, pleasure_m$.

**Constraints.** $1 \leq n, m \leq 8$; $-100 \leq A_{ij} \leq 100$; $-1\,000\,000 \leq b_i \leq 1\,000\,000$; $-100 \leq cost_i \leq 100$.

**Output Format.** If there is no diet that satisfies all the restrictions, output "No solution" (without quotes). If you can get as much pleasure as you want despite all the restrictions, output "Infinity" (without quotes). If the maximum possible total pleasure is bounded, output two lines. On the first line, output "Bounded solution" (without quotes). On the second line, output $m$ real numbers — the optimal $amounts$ for each dish and drink. Output all the numbers with at least 15 digits after the decimal point.

The amounts you output will be inserted into the inequalities, and all the inequalities will be checked. An inequality $L \leq R$ will be considered satisfied if actually $L \leq R + 10^{-3}$. The total pleasure of your solution will be calculated and compared with the optimal value. Your output will be accepted if all the inequalities are satisfied and the total pleasure of your solution differs from the optimal value by at most $10^{-3}$. **We ask you to output at least 15 digits after the decimal point, although we will check the answer with precision of only $10^{-3}$. This is because in the process of checking the inequalities we will multiply your answers with coefficients from the matrix $A$ and with the coefficients of the vector *pleasure*, and those coefficients can be pretty large, and computations with real numbers on a computer are not always precise. This way, the more digits after the decimal point you output for each amount — the less likely it is that your answer will be rejected because of precision issues.**

**Time Limits.**

| language | C | C++ | Java | Python | C# | Haskell | JavaScript | Ruby | Scala |
|---|---|---|---|---|---|---|---|---|---|
| time (sec) | 1 | 1 | 2 | 30 | 1.5 | 2 | 30 | 30 | 4 |

**Memory Limit.** 512MB.

**Sample 1.**

Input:

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

Output:

```
Bounded solution
0.000000000000000 2.000000000000000
```

Explanation:
Here we have only two items, and we know that $(-1) \cdot amount_1 + (-1) \cdot amount_2 \le -1 \Rightarrow amount_1 + amount_2 \ge 1$ from the first inequality, and also that $amount_1 \le 2$ and $amount_2 \le 2$ from the second and the third inequalities. We also know that all amounts are non-negative. We want to maximize $(-1) \cdot amount_1 + 2 \cdot amount_2$ under those restrictions — that is, we don't like dish or drink number 1, and we twice as much like dish or drink number 2. It is optimal then to consume as few as possible of the first item and as much as possible of the second item. It turns out that we can avoid consuming the first item at all and take the maximum possible amount 2 of the second item, and all the restrictions will be satisfied! Clearly, this is a diet with the maximum possible total pleasure! **Note that integers 0 and 2 in the output are printed with 15 digits after the decimal point. Don't forget to print at least 15 digits after the decimal point, as the answers to some tests will be non-integer, and you don't want to get your answer rejected only because of some rounding problems.**

**Sample 2.**

Input:

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

Output:

```
No solution
```

Explanation:
The first inequality gives $amount_1 + amount_2 \leq 1$ and the second inequality gives $(-1) \cdot amount_1 + (-1) \cdot amount_2 \leq -2 \Rightarrow amount_1 + amount_2 \geq 2$. But $amount_1 + amount_2$ cannot be less than 1 and more than 2 simultaneously, so there is no solution in this case.

**Sample 3.**

Input:

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

Output:

```
Infinity
```

Explanation:
The restrictions in this case are only that all amounts are non-negative (these restrictions are always there, because you cannot consume negative amount of a dish or a drink) and that $amount_3 \leq 3$. There is no restriction on how much to consume of items 1 and 2, and each of them has positive *pleasure* value, so you can take as much of items 1 and 2 as you want and receive as much total pleasure as you want. In this case, you should output "Infinite" (without quotes).

## Starter Files

The starter solutions for this problem read the data from the input, pass it to a blank procedure and output the result. You need to implement this procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `diet`

## What To Do

There are at most 8 inequalities (16 if you count in the inequalities $amount_i \geq 0$) with at most 8 variables. You can use this fact and the fact that the optimal solution is always in a vertex of the polyhedron corresponding to the linear programming problem. At least $m$ of the inequalities become equalities in each vertex of the polyhedron. If there are $n$ regular inequalities, $m$ variables and $m$ inequalities of the form $amount_i \geq 0$, you need to take each possible subset of size $m$ out of all the $n + m$ inequalities, solve the system of linear equations where each equation is one of the selected inequalities changed to equality, check whether this solution satisfies all the other inequalities, and in the end select the solution with the largest value of the total pleasure out of those which satisfy all inequalities. The running time of this algorithm is $O(2^{n+m}(m^3 + mn))$, which is good enough to pass. $2^{n+m}$ is to go through all the subsets of the inequalities (although you will need only subsets of size $m$), $m^3$ is for Gaussian Elimination and $mn$ is to check a solution of a system of linear equations against all the inequalities. Various ways to traverse all the subsets of some set are described here and here.

The only case that you would miss this way is the case when the correct answer is “Infinity”. It is guaranteed that in all test cases in this problem, if a solution is bounded, then

$$amount_1 + amount_2 + \cdots + amount_m \le 10^9 .$$

Thus, to distinguish between bounded and unbounded cases, just add this inequality to the initial problem. If the resulting program has the last inequality among those $m$ that define the vertex, output “Infinity”, otherwise output “Bounded solution” and the solution of the augmented problem on the second line.

## Need Help?

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

## 3 Advanced Problem: Online Advertisement 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

Online and mobile advertising is one of the most profitable businesses in the world. Google and Facebook are generating many billions of dollars of revenue each year, and around 90% of their revenues come from advertisement. In this problem you will help an online advertising system like Google AdSense or Yandex Direct to allocate the ad impressions in its Advertising Network so as to maximize revenue while satisfying all the advertisers' requirements.

### Problem Description

**Task.** You have $n$ clients, they are advertisers, and each of them wants to show their ads to some number of internet users specified in the contract (or more) next month. Your online advertising network has $m$ placements overall on all the sites connected to the network. You know how many users each advertiser wants to reach, how many users will see each of the $m$ ad placements next month, and how much each advertiser is willing to pay for one user who sees their ad through each particular ad placement (different placements can be on different sites attracting different types of users, and each advertiser is more interested in the visitors of some sites than the others). You can show different ads of different advertisers in the same ad placement throughout the next month or show always the same ad of the same advertiser, but the total number of users that will see some ad in that placement is estimated and fixed. You want to maximize your total revenue which is the sum of amounts each advertiser will pay you for all the users who have seen their ads.

If we denote by $x_{ij}$ the number of users who have seen an ad of advertiser $i$ in the ad placement $j$, then all the restrictions can be written as linear equalities and inequalities in $x_{ij}$. For example, if the total number of users that will see ad placement $j$ is $S_j$, then we add an equality $\sum\limits_{i=1}^{m} x_{ij} = S_j$. If the $i$-th advertiser wants to show the ad to at least $U_i$ users, we add an inequality $\sum\limits_{j=1}^{n} x_{ij} \geq U_i \Leftrightarrow \sum\limits_{j=1}^{n} (-1) \cdot x_{ij} \leq -U_i$. Of course, each $x_{ij}$ is non-negative: $x_{ij} \geq 0$. If advertiser $i$ wishes to pay $c_{ij}$ cents for each user who sees her advertisement through ad placement $j$, then the goal to maximize the total revenue is given by linear objective $\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{m} c_{ij} x_{ij} \to \max$. This leads to a linear programming problem which you need to solve. This time it will contain more variables and inequalities, because the number of advertisers and the number of different ad placements can be large.

**Input Format.** You are given the ad allocation problem reduced to a linear programming problem of the form $Ax \leq b, x \geq 0, \sum\limits_{i=1}^{q} c_i x_i \to \max$, where $A$ is a matrix $p \times q$, $b$ is a vector of length $p$, $c$ is a vector of length $q$ and $x$ is the unknown vector of length $q$.

The first line of the input contains integers $p$ and $q$ — the number of inequalities in the system and the number of variables respectively. The next $p + 1$ lines contain the coefficients of the linear inequalities in the standard form $Ax \leq b$. Specifically, $i$-th of the next $p$ lines contains $q$ integers $A_{i1}, A_{i2}, \ldots, A_{iq}$, and the next line after those $p$ contains $p$ integers $b_1, b_2, \dots, b_p$. These lines describe $p$ inequalities of the form $A_{i1} \cdot x_1 + A_{i2} \cdot x_2 + \cdots + A_{iq} \cdot x_q \le b_i$. The last line of the input contains $q$ integers — the coefficients $c_i$ of the objective $\sum_{i=1}^{q} c_i x_i \to \max$.

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

**Output Format.** If there is no allocation that satisfies all the requirements, output "No solution" (without quotes). If you can get as much revenue as you want despite all the requirements, output "Infinity" (without quotes). If the maximum possible revenue is bounded, output two lines. On the first line, output "Bounded solution" (without quotes). On the second line, output $q$ real numbers — the optimal values of the vector $x$ (recall that $x = x_{ij}$ is how many users will see the ad of advertiser $i$ through the placement $j$, but we changed the numbering of variables to $x_1, x_2, \dots, x_q$). Output all the numbers with at least 15 digits after the decimal point. Your solution will be accepted if all the inequalities are satisfied and the answer has absolute error of at most $10^{-3}$. **See the previous problem output format description for the explanation of what this means and why do we ask to output at least 15 digits after the decimal point.**

**Time Limits.**

| language | C | C++ | Java | Python | C# | Haskell | JavaScript | Ruby | Scala |
|---|---|---|---|---|---|---|---|---|---|
| time (sec) | 1 | 1 | 1.5 | 6 | 1.5 | 2 | 6 | 6 | 3 |

**Memory Limit.** 512MB.

**Sample 1.**

Input:

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

Output:

```
Bounded solution
0.000000000000000 2.000000000000000
```

Explanation:
Here we have only two variables, and we know that $(-1) \cdot x_1 + (-1) \cdot x_2 \le -1 \Rightarrow x_1 + x_2 \ge 1$ from the first inequality, and also that $x_1 \le 2$ and $x_2 \le 2$ from the second and the third inequalities. We also know that all amounts are non-negative. We want to maximize $(-1) \cdot x_1 + 2 \cdot x_2$ under those restrictions. It is optimal to minimize $x_1$ and maximize $x_2$. It turns out that we can set $x_1 = 0$ (the minimum possible) and $x_2 = 2$ (the maximum possible), and all the requirements will be satisfied. **Note that integers $0$ and $2$ in the output are printed with $15$ digits after the decimal point. Don't forget to print at least $15$ digits after the decimal point, as the answers to some tests will be non-integer, and you don't want to get your answer rejected only because of some rounding problems.**

**Sample 2.**

Input:

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

Output:

```
No solution
```

Explanation:
The first inequality gives $x_1 + x_2 \leq 1$ and the second inequality gives $(-1) \cdot x_1 + (-1) \cdot x_2 \leq -2 \Rightarrow x_1 + x_2 \geq 2$. But $x_1 + x_2$ cannot be less than 1 and more than 2 simultaneously, so there is no solution in this case.

**Sample 3.**

Input:

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

Output:

```
Infinity
```

Explanation:
The restrictions in this case are only that all amounts are non-negative (these restrictions are always there, because you cannot show an ad to negative number of users) and that $x_3 \leq 3$. There is no upper bound on $x_1$ and $x_2$, and both $c_1$ and $c_2$ are positive, so you can set $x_1$ and $x_2$ big enough and generate as much revenue as you want. In this case, you should output "Infinite" (without quotes).

## Starter Files

The starter solutions for this problem read the data from the input, pass it to a blank procedure and output the result. You need to implement this procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `ad_allocation`

## What To Do

You will need to implement the Simplex Method from the lectures to solve this problem.

## 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 $0, 0, \dots, 0$ or a sequence $10^6, 10^6, \dots, 10^6$. 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