Algorithmic Toolbox at Coursera: Programming Challenges (Week 1)

January 11, 2018

Contents

To introduce you to our automated grading system, we will discuss two simple programming challenges and walk you through a step-by-step process of solving them. We will encounter several common pitfalls and will show you how to fix them.

Below is a brief overview of what it takes to solve a programming challenge in five steps:

Reading problem statement. The problem statement specifies the input-output format, the constraints for the input data as well as time and memory limits. Your goal is to implement a fast program that solves the problem and works within the time and memory limits.

Designing an algorithm. When the problem statement is clear, start designing an algorithm and don't forget to prove that it works correctly.

Implementing an algorithm. After you developed an algorithm, start implementing it in a programming language of your choice.

Testing and debugging your program. Testing is the art of revealing bugs. Debugging is the art of exterminating the bugs. When your program is ready, start testing it! If a bug is found, fix it and test again.

Submitting your program to the grading system. After testing and debugging your program, submit it to the grading system and wait for the message "Good job!". In the case you see a different message, return back to the previous stage.

1 Sum of Two Digits

Sum of Two Digits Problem
Compute the sum of two single digit numbers.

Input: Two single digit numbers.
Output: The sum of these numbers.

$$2 + 3 = 5$$

We start from this ridiculously simple problem to show you the pipeline of reading the problem statement, designing an algorithm, implementing it, testing and debugging your program, and submitting it to the grading system.

Input format. Integers a and b on the same line (separated by a space).

Output format. The sum of a and b.

Constraints. $0 \le a, b \le 9$.

Sample.

Input:

```
9 7
```

Output:

```
16
```

Time limits (sec.):

C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Rust	Scala
1	1	1.5	5	1.5	2	5	5	1	3

Memory limit. 512 Mb.

1.1 Implementing an Algorithm

For this trivial problem, we will skip "Designing an algorithm" step and will move right to the pseudocode.

```
SumOfTwoDigits(a, b):
return a + b
```

Since the pseudocode does not specify how we input a and b, below we provide solutions in `C++`, `Java`, and `Python3` programming languages as well as recommendations on compiling and running them. You can copy-and-paste the code to a file, compile/run it, test it on a few datasets, and then submit (the source file, not the compiled executable) to the grading system. Needless to say, we assume that you know the basics of one of programming languages that we use in our grading system.

C++

```
#include <iostream>

int main() {
  int a = 0;
  int b = 0;
  std::cin >> a;
  std::cin >> b;
  std::cout << a + b;
  return 0;
}
```

Save this to a file (say, `aplusb.cpp`), compile it, run the resulting executable, and enter two numbers (on the same line).

Java

```
import java.util.Scanner;

class APlusB {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int a = s.nextInt();
    int b = s.nextInt();
```

```
    System.out.println(a + b);
  }
}
```

Save this to a file `APlusB.java`, compile it, run the resulting executable, and enter two numbers (on the same line).

Python3

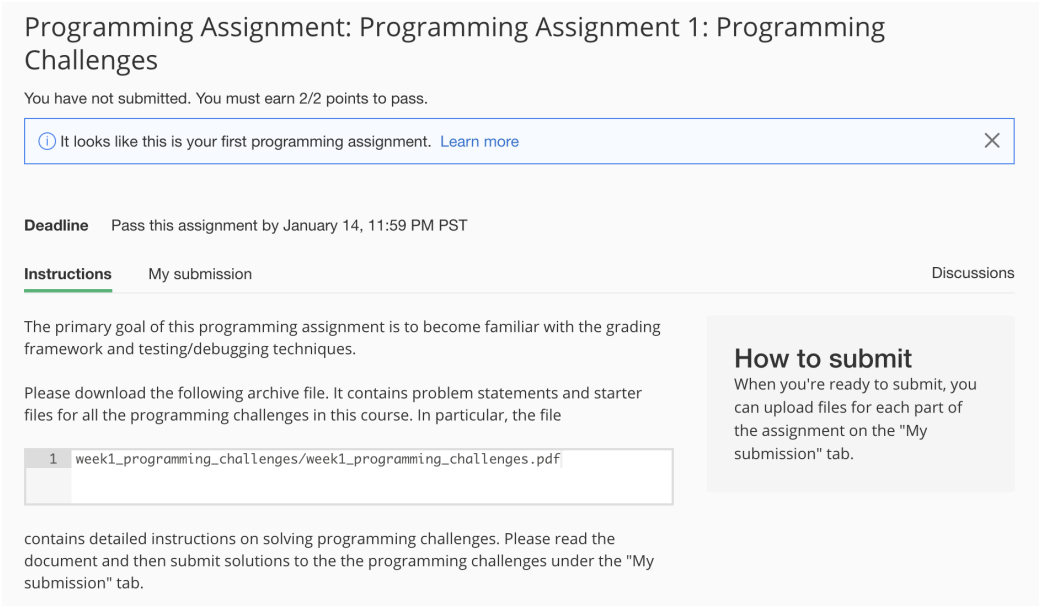
```
# Uses python3
import sys

input = sys.stdin.read()
tokens = input.split()
a = int(tokens[0])
b = int(tokens[1])
print(a + b)
```

Save this to a file (say, `aplusb.py`), run it, and enter two numbers on the same line. To indicate the end of input, press `ctrl-d`/`ctrl-z`. (The first line in the code above tells the grading system to use `Python3` rather `Python2`.)

Your goal is to implement an algorithm that produces a correct result under the given time and memory limits for any input satisfying the given constraints. You do not need to check that the input data satisfies the constraints, e.g., for the Sum of Two Digits Problem you do not need to check that the given integers a and b are indeed single digit integers (this is guaranteed).

1.2 Submitting to the Grading System at Coursera

This is what a fresh submission page looks like:



The blue bubble suggests you to read a general information about programming assignments. We encourage you to read it if you haven't solved programming assignments before.

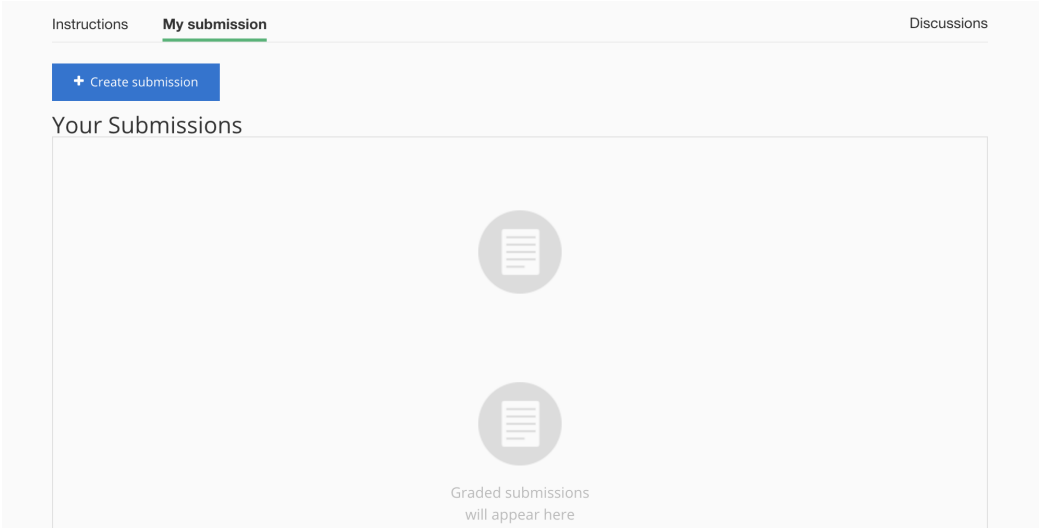
The line above the bubble indicates that there are two programming challenges in this assignment. To pass this assignment, you need to solve both programming challenges. This should not be difficult as in this document we give detailed instructions for solving them. For all the remaining programming assignments, the passing threshold is usually around 50%. To pass the course, you need to pass all the programming assignments.

The line below the blue bubble indicates the deadline for this assignment. Please not that this is a suggestion rather than a hard deadline. If you miss a deadline, you may safely switch to the next session of the course (a new session starts automatically every two weeks). All your progress will be transferred to the new session in this case. See Coursera help article on switching sessions.

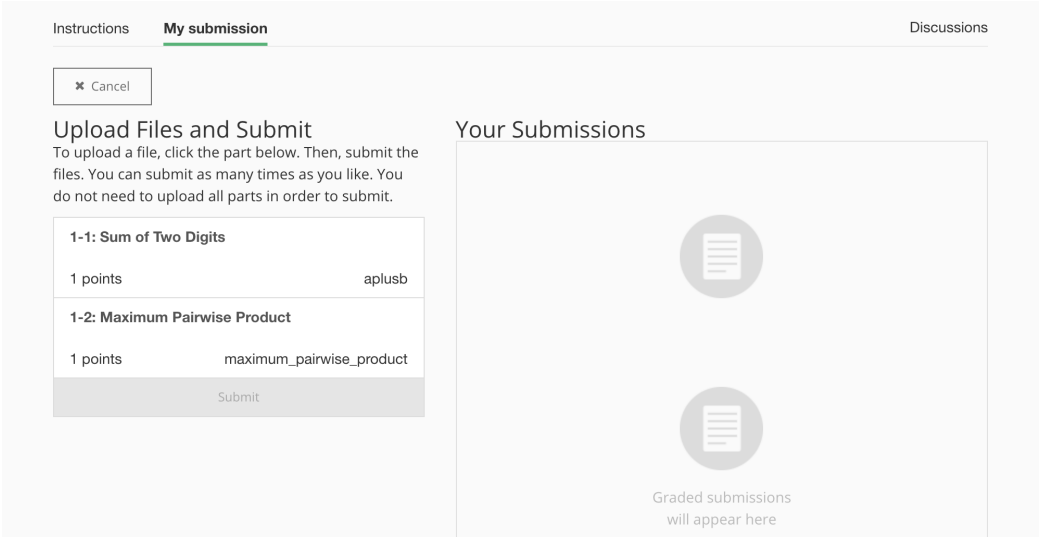
At the right, there is a link to the discussion forum attached to this assignment. We encourage you to visit the forum on a regular basis to ask

questions and to help other learners. Note also that for each programming challenge, there is a separate thread at the forum. The link to this particular thread can be found at the end of the programming challenge statement.

When you are ready to submit a solution to a programming challenge, go the the "My submission" tab that looks as follows:

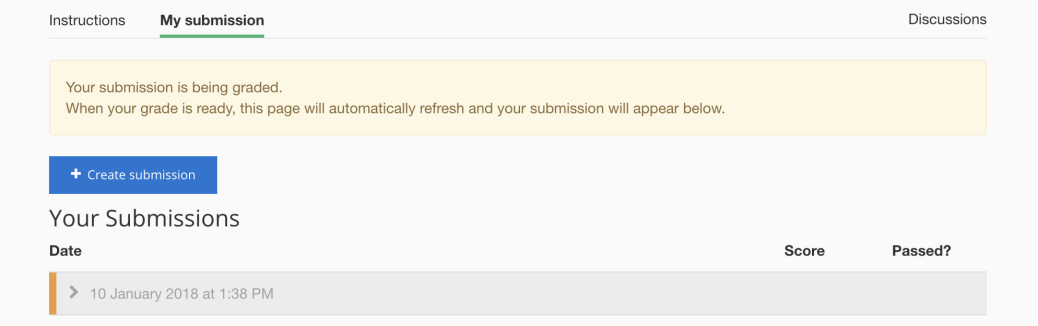


Click on the "Create submission" button. The page now looks as follows.

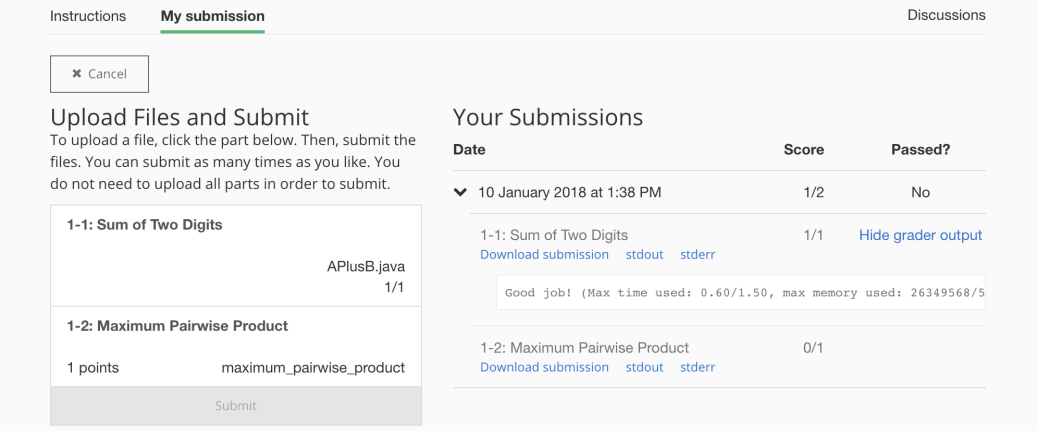


In this particular assignment, there are two programming challenges (Sum

of Two Digits and Maximum Pairwise Product). Click on a challenge you want to submit, upload a source file (rather than executable) of your solution, and press the “Submit” button. The page now looks as follows:



You may safely leave the page while your solution is being graded (in most cases, it is done in less than a minute, but when the servers are overloaded it may take several minutes; please be patient). When the grading is done, the submission page will look like this:



In this particular case, it says that the first programming challenge is passed (score is 1/1) but the whole programming assignment is not passed yet (since one needs to pass both programming challenges for this).

2 Maximum Pairwise Product

Maximum Pairwise Product Problem
Find the maximum product of two distinct numbers in a sequence of non-negative integers.

Input: A sequence of non-negative integers.
Output: The maximum value that can be obtained by multiplying two different elements from the sequence.

	5	6	2	7	4
5		30	10	35	20
6	30		12	42	24
2	10	12		7	4
7	35	42	14		28
4	20	24	8	28	

Given a sequence of non-negative integers $a_1, \ldots, a_n$, compute

$$\max_{1 \le i \ne j \le n} a_i \cdot a_j .$$

Note that i and j should be different, though it may be the case that $a_i = a_j$.

Input format. The first line contains an integer n. The next line contains n non-negative integers $a_1, \ldots, a_n$ (separated by spaces).

Output format. The maximum pairwise product.

Constraints. $2 \le n \le 2 \cdot 10^5$; $0 \le a_1, \ldots, a_n \le 2 \cdot 10^5$.

Sample 1.

Input:

```
3
1 2 3
```

Output:

```
6
```

Sample 2.

Input:

```
10
7 5 14 2 8 8 10 1 2 3
```

Output:

```
140
```

Time and memory limits. The same as for the previous problem.

2.1 Naive Algorithm

A naive way to solve the Maximum Pairwise Product Problem is to go through all possible pairs of the input elements $A[1 \dots n] = [a_1, \dots, a_n]$ and to find a pair of distinct elements with the largest product:

```
MaxPairwiseProductNaive(A[1...n]):
product ← 0
for i from 1 to n:
  for j from 1 to n:
    if i ≠ j:
      if product < A[i] · A[j]:
        product ← A[i] · A[j]
return product
```

This code can be optimized and made more compact as follows.

```
MaxPairwiseProductNaive(A[1...n]):
product ← 0
for i from 1 to n:
  for j from i + 1 to n:
    product ← max(product, A[i] · A[j])
return product
```

Implement this algorithm in your favorite programming language. If you are using C++, Java, or Python3, you may want to download the starter files (we provide starter solutions in these three languages for all the problems in the book). For other languages, you need to implement your solution from scratch.

Stater solutions for C++, Java, and Python3 are shown below.

C++

```
#include <iostream>
#include <vector>

using std::vector;
using std::cin;
using std::cout;
using std::max;

int MaxPairwiseProduct(const vector<int>& numbers) {
  int product = 0;
  int n = numbers.size();
  for (int i = 0; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
      product = max(product, numbers[i] * numbers[j]);
    }
  }
  return product;
}

int main() {
    int n;
    cin >> n;
    vector<int> numbers(n);
    for (int i = 0; i < n; ++i) {
        cin >> numbers[i];
    }

    int product = MaxPairwiseProduct(numbers);
    cout << product << "\n";
    return 0;
}
```

Java

```
import java.util.*;
import java.io.*;
```

```
public class MaxPairwiseProduct {
    static int getMaxPairwiseProduct(int[] numbers) {
        int product = 0;
        int n = numbers.length;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                product = Math.max(product,
                                   numbers[i] * numbers[j]);
            }
        }
        return product;
    }

    public static void main(String[] args) {
        FastScanner scanner = new FastScanner(System.in);
        int n = scanner.nextInt();
        int[] numbers = new int[n];
        for (int i = 0; i < n; i++) {
            numbers[i] = scanner.nextInt();
        }
        System.out.println(getMaxPairwiseProduct(numbers));
    }

    static class FastScanner {
        BufferedReader br;
        StringTokenizer st;

        FastScanner(InputStream stream) {
            try {
                br = new BufferedReader(new
                    InputStreamReader(stream));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        String next() {
            while (st == null || !st.hasMoreTokens()) {
                try {
```

```
                st = new StringTokenizer(br.readLine());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return st.nextToken();
    }

    int nextInt() {
        return Integer.parseInt(next());
    }
  }

}
```

Python

```
# Uses python3
n = int(input())
a = [int(x) for x in input().split()]

product = 0

for i in range(n):
    for j in range(i + 1, n):
        product = max(product, a[i] * a[j])

print(product)
```

After submitting this solution to the grading system, many students are surprised when they see the following message:

```
Failed case #4/17: time limit exceeded
```

After you submit your program, we test it on dozens of carefully designed test cases to make sure the program is fast and error proof. As the result, we usually know what kind of errors you made. The message above tells that the submitted program exceeds the time limit on the 4th out of 17 test cases.

Stop and Think. Why does the solution not fit into the time limit?

MaxPairwiseProductNaive performs of the order of n^2 steps on a sequence of length n. For the maximal possible value $n = 2 \cdot 10^5$, the number of steps is of the order $4 \cdot 10^{10}$. Since many modern computers perform roughly 10^8–10^9 basic operations per second (this depends on a machine, of course), it may take tens of seconds to execute MaxPairwiseProductNaive, exceeding the time limit for the Maximum Pairwise Product Problem.

We need a faster algorithm!

2.2 Fast Algorithm

In search of a faster algorithm, you play with small examples like $[5, 6, 2, 7, 4]$. Eureka—it suffices to multiply the two largest elements of the array—7 and 6!

Since we need to find the largest and the second largest elements, we need only two scans of the sequence. During the first scan, we find the largest element. During the second scan, we find the largest element among the remaining ones by skipping the element found at the previous scan.

```
MaxPairwiseProductFast(A[1...n]):
index_1 ← 1
for i from 2 to n:
  if A[i] > A[index_1]:
    index_1 ← i
index_2 ← 1
for i from 2 to n:
  if A[i] ≠ A[index_1] and A[i] > A[index_2]:
    index_2 ← i
return A[index_1] · A[index_2]
```

2.3 Testing and Debugging

Implement this algorithm and test it using an input $A = [1, 2]$. It will output 2, as expected. Then, check the input $A = [2, 1]$. Surprisingly, it outputs 4. By inspecting the code, you find out that after the first loop, $index_1 = 1$. The algorithm then initializes $index_2$ to 1 and $index_2$ is never

updated by the second for loop. As a result, $index_1 = index_2$ before the return statement. To ensure that this does not happen, you modify the pseudocode as follows:

```
MaxPairwiseProductFast(A[1...n]):
index_1 ← 1
for i from 2 to n:
  if A[i] > A[index_1]:
    index_1 ← i
if index_1 = 1:
  index_2 ← 2
else:
  index_2 ← 1
for i from 1 to n:
  if A[i] ≠ A[index_1] and A[i] > A[index_2]:
    index_2 ← i
return A[index_1] · A[index_2]
```

Check this code on a small datasets $[7, 4, 5, 6]$ to ensure that it produces correct results. Then try an input

```
2
100000  90000
```

You may find out that the program outputs something like 410 065 408 or even a negative number instead of the correct result 9 000 000 000. If it does, this is most probably caused by an *integer overflow*. For example, in C++ programming language a large number like 9 000 000 000 does not fit into the standard `int` type that on most modern machines occupies 4 bytes and ranges from -2^{31} to $2^{31} - 1$, where

$$2^{31} = 2\,147\,483\,648.$$

Hence, instead of using the C++ `int` type you need to use the `int64_t` type when computing the product and storing the result. This will prevent integer overflow as the `int64_t` type occupies 8 bytes and ranges from -2^{63} to $2^{63} - 1$, where

$$2^{63} = 9\,223\,372\,036\,854\,775\,808.$$

You then proceed to testing your program on large data sets, e.g., an array $A[1 \dots 2 \cdot 10^5]$, where $A[i] = i$ for all $1 \le i \le 2 \cdot 10^5$. There are two ways of doing this.

1. Create this array in your program and pass it to MaxPairwiseProductFast (instead of reading it from the standard input).

2. Create a separate program, that writes such an array to a file `dataset.txt`. Then pass this dataset to your program from console as follows:

```
yourprogram < dataset.txt
```

Check that your program processes this dataset within time limit and returns the correct result: 39 999 800 000. You are now confident that the program finally works!

However, after submitting it to the testing system, it fails again...

```
Failed case #5/17: wrong answer
```

But how would you generate a test case that make your program fail and help you to figure out what went wrong?

2.4 Can You Tell Me What Error Have I Made?

You are probably wondering why we did not provide you with the 5th out of 17 test datasets that brought down your program. The reason is that nobody will provide you with the test cases in real life!

Since even experienced programmers often make subtle mistakes solving algorithmic problems, it is important to learn how to catch bugs as early as possible. When the authors of this book started to program, they naively thought that nearly all their programs are correct. By now, we know that our programs are *almost never* correct when we first run them.

When you are confident that your program works, you often test it on just a few test cases, and if the answers look reasonable, you consider your work done. However, this is a recipe for a disaster. To make your program *always* work, you should test it on a set of carefully designed test cases. Learning how to implement algorithms as well as test and debug your programs will be invaluable in your future work as a programmer.

2.5 Stress Testing

We will now introduce *stress testing*—a technique for generating thousands of tests with the goal of finding a test case for which your solution fails.

A stress test consists of four parts:

1. Your implementation of an algorithm.
2. An alternative, trivial and slow, but correct implementation of an algorithm for the same problem.
3. A random test generator.
4. An infinite loop in which a new test is generated and fed into both implementations to compare the results. If their results differ, the test and both answers are output, and the program stops, otherwise the loop repeats.

The idea behind stress testing is that two correct implementations should give the same answer for each test (provided the answer to the problem is unique). If, however, one of the implementations is incorrect, then there exists a test on which their answers differ. The only case when it is not so is when there is the same mistake in both implementations, but that is unlikely (unless the mistake is somewhere in the input/output routines which are common to both solutions). Indeed, if one solution is correct and the other is wrong, then there exists a test case on which they differ. If both are wrong, but the bugs are different, then most likely there exists a test on which two solutions give different results.

Here is the the stress test for MaxPairwiseProductFast using MaxPairwiseProductNaive as a trivial implementation:

```
StressTest(N, M):
while true:
  n ← random integer between 2 and N
  allocate array A[1...n]
  for i from 1 to n:
    A[i] ← random integer between 0 and M
  print(A[1...n])
  result_1 ← MaxPairwiseProductNaive(A)
  result_2 ← MaxPairwiseProductFast(A)
  if result_1 = result_2:
    print("OK")
  else:
    print("Wrong answer: ", result_1, result_2)
    return
```

The `while` loop above starts with generating the length of the input sequence n, a random number between 2 and N. It is at least 2, because the problem statement specifies that $n \geq 2$. The parameter N should be small enough to allow us to explore many tests despite the fact that one of our solutions is slow.

After generating n, we generate an array A with n random numbers from 0 to M and output it so that in the process of the infinite loop we always know what is the current test; this will make it easier to catch an error in the test generation code. We then call two algorithms on A and compare the results. If the results are different, we print them and halt. Otherwise, we continue the while loop.

Let's run StressTest(10, 100 000) and keep our fingers crossed in a hope that it outputs "Wrong answer." We see something like this (the result can be different on your computer because of a different random number generator).

```
...
OK
67232 68874 69499
OK
6132 56210 45236 95361 68380 16906 80495 95298
OK
62180 1856 89047 14251 8362 34171 93584 87362 83341 8784
OK
```

```
21468 16859 82178 70496 82939 44491
OK
68165 87637 74297 2904 32873 86010 87637 66131 82858 82935
Wrong answer: 7680243769 7537658370
```

Hurrah! We've found a test case where MaxPairwiseProductNaive and MaxPairwiseProductFast produce different results, so now we can check what went wrong. Then we can debug this solution on this test case, find a bug, fix it, and repeat the stress test again.

Stop and Think. Do you see anything suspicious in the found dataset?

Note that generating tests automatically and running stress test is easy, but debugging is hard. Before diving into debugging, let's try to generate a smaller test case to simplify it. To do that, we change N from 10 to 5 and M from 100 000 to 9.

Stop and Think. Why did we first run StressTest with large parameters N and M and now intend to run it with small N and M?

We then run the stress test again and it produces the following.

```
...
7 3 6
OK
2 9 3 1 9
Wrong answer: 81 27
```

The slow MaxPairwiseProductNaive gives the correct answer 81 ($9 \cdot 9 = 81$), but the fast MaxPairwiseProductFast gives an incorrect answer 27.

Stop and Think. How MaxPairwiseProductFast can possibly return 27?

To debug our fast solution, let's check which two numbers it identifies as two largest ones. For this, we add the following line before the `return` statement of the MaxPairwiseProductFast function:

```
print(index1, index2)
```

After running the stress test again, we see the following.

```
...
7 3 6
1 3
```

```
OK
5
2 9 3 1 9
2 3
Wrong answer: 81 27
```

Note that our solutions worked and then failed on exactly the same test cases as on the previous run of the stress test, because we didn't change anything in the test generator. The numbers it uses to generate tests are pseudorandom rather than random—it means that the sequence looks random, but it is the same each time we run this program. It is a convenient and important property, and you should try to have your programs exhibit such behavior, because deterministic programs (that always give the same result for the same input) are easier to debug than non-deterministic ones.

Now let's examine $index_1 = 2$ and $index_2 = 3$. If we look at the code for determining the second maximum, we will notice a subtle bug. When we implemented a condition on i (such that it is not the same as the previous maximum) instead of comparing i and $index_1$, we compared $A[i]$ with $A[index_1]$. This ensures that the second maximum differs from the first maximum by the value rather than by the index of the element that we select for solving the Maximum Pairwise Product Problem. So, our solution fails on any test case where the largest number is equal to the second largest number. We now change the condition from

$$A[i] \neq A[index_1]$$

to

$$i \neq index_1$$

After running the stress test again, we see a barrage of "OK" messages on the screen. We wait for a minute until we get bored and then decide that MaxPairwiseProductFast is finally correct!

However, you shouldn't stop here, since you have only generated very small tests with $N = 5$ and $M = 10$. We should check whether our program works for larger n and larger elements of the array. So, we change N to 1 000 (for larger N, the naive solution will be pretty slow, because its running time is quadratic). We also change M to 200 000 and run. We again see the screen filling with words "OK", wait for a minute, and then

decide that (finally!) MaxPairwiseProductFast is correct. Afterwards, we submit the resulting solution to the grading system and pass the Maximum Pairwise Product Problem test!

As you see, even for such a simple problems like Maximum Pairwise Product, it is easy to make subtle mistakes when designing and implementing an algorithm. The pseudocode below presents a more "reliable" way of implementing the algorithm.

```
MaxPairwiseProductFast(A[1...n]):
index ← 1
for i from 2 to n:
  if A[i] > A[index]:
    index ← i
swap A[index] and A[n]
index ← 1
for i from 2 to n − 1:
  if A[i] > A[index]:
    index ← i
swap A[index] and A[n − 1]
return A[n − 1] · A[n]
```

In this book, besides learning how to design and analyze algorithms, you will learn how to implement algorithms in a way that minimizes the chances of making a mistake, and how to test your implementations.

2.6 Even Faster Algorithm

The MaxPairwiseProductFast algorithm finds the largest and the second largest elements in about $2n$ comparisons.

Exercise Break. Find two largest elements in an array in $1.5n$ comparisons.

After solving this problem, try the next, even more challenging Exercise Break.

Exercise Break. Find two largest elements in an array in $n + \log_2 n$ comparisons.

And if you feel that the previous Exercise Break was easy, here are the next two challenges that you may face at your next interview!

Exercise Break. Prove that no algorithm for finding two largest elements in an array can do this in less than $n + \log_2 n$ comparisons.

Exercise Break. What is the fastest algorithm for finding three largest elements?

2.7 A More Compact Algorithm

The Maximum Pairwise Product Problem can be solved by the following compact algorithm that uses sorting (in non-decreasing order).

```
MaxPairwiseProductBySorting(A[1..n]):
Sort(A)
return A[n-1]·A[n]
```

This algorithm does more than we actually need: instead of finding two largest elements, it sorts the entire array. For this reason, its running time is $O(n \log n)$, but not $O(n)$. Still, for the given constraints ($2 \le n \le 2 \cdot 10^5$) this is usually sufficiently fast to fit into a second and pass our grader.

3 Solving a Programming Challenge in Five Easy Steps

Below we summarize what we've learned in this chapter.

3.1 Reading Problem Statement

Start by reading the problem statement that contains the description of a computational task, time and memory limits, and a few sample tests. Make sure you understand how an output matches an input in each sample case.

If time and memory limits are not specified explicitly in the problem statement, the following default values are used.

Time limits (sec.):

C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Rust	Scala
1	1	1.5	5	1.5	2	5	5	1	3

Memory limit: 512 Mb.

3.2 Designing an Algorithm

After designing an algorithm, prove that it is correct and try to estimate its expected running time on the most complex inputs specified in the constraints section. If you laptop performs roughly 10^8–10^9 operations per second, and the maximum size of a dataset in the problem description is $n = 10^5$, then an algorithm with quadratic running time is unlikely to fit into the time limit (since $n^2 = 10^{10}$), while a solution with running time $O(n \log n)$ will. However, an $O(n^2)$ solution will fit if $n = 1\,000$, and if $n = 100$, even an $O(n^3)$ solutions will fit. Although polynomial algorithms remain unknown for some hard problems in this book, a solution with $O(2^n n^2)$ running time will probably fit into the time limit as long as n is smaller than 20.

3.3 Implementing an Algorithm

Start implementing your algorithm in one of the following programming languages supported by our automated grading system: `C`, `C++`, `C#`, `Haskell`, `Java`, `JavaScript`, `Python2`, `Python3`, `Ruby`, or `Scala`. For all problems, we provide starter solutions for `C++`, `Java`, and `Python3`. For other programming languages, you need to implement a solution from scratch. The grading system detects the programming language of your submission automatically, based on the extension of the submission file.

We have reference solutions in `C++`, `Java`, and `Python3` (that we don't share with you) which solve the problem correctly under the given constraints, and 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.

In the Appendix, we list compiler versions and flags used by the grading system. We recommend using the same compiler flags when you test your solution locally. 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).

3.4 Testing and Debugging

Submitting your implementation to the grading system without testing it first is a bad idea! Start with small datasets and make sure that your program produces correct results on all sample datasets. Then proceed to checking how long it takes to process a large dataset. To estimate the running time, 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 10^5, pass it to your `solve()` function, and ensure that the program outputs the result quickly.

Check the boundary values to ensure that your program processes correctly both short sequences (e.g., with 2 elements) and long sequences (e.g., with 10^5 elements). If a sequence of integers from 0 to, let's 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 $10^6, 10^6, \ldots, 10^6$. Afterwards, check it also on randomly generated data. Check degenerate cases like an empty set, three points on a single line, a tree which consists of a single path of nodes, etc.

After it appears that your program works on all these tests, proceed to stress testing. Implement a slow, but simple and correct algorithm and check that two programs produce the same result (note however that this is not applicable to problems where the output is not unique). Generate random test cases as well as biased tests cases such as those with only small numbers or a small range of large numbers, strings containing a single letter "a" or only two different letters (as opposed to strings composed of all possible Latin letters), and so on. Think about other possible tests which could be peculiar in some sense. For example, if you are generating graphs, try generating trees, disconnected graphs, complete graphs, bipartite graphs, etc. If you generate trees, try generating paths, binary trees,

stars, etc. If you are generating integers, try generating both prime and composite numbers.

3.5 Submitting to the Grading System

When you are done with testing, submit your program to the grading system! Go to the submission page, create a new submission, and upload a file with your program (make sure to upload a source file rather than an executable). The grading system then compiles your program and runs it on a set of carefully constructed tests to check that it outputs a correct result for all tests and that it fits into the time and memory limits. The grading usually takes less 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. You want to see the "`Good job!`" message indicating that your program passed all the tests. The messages "`Wrong answer`", "`Time limit exceeded`", "`Memory limit exceeded`" notify you that your program failed due to one of these reasons. If you program fails on one of the first two test cases, the grader will report this to you and will show you the test case and the output of your program. This is done to help you to get the input/output format right. In all other cases, the grader will *not* show you the test case where your program fails.

4 Appendix: Compiler Flags

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

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

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

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

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

Java (`OpenJDK 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
```

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

```
scalac
```

Programming Assignment 2: Algorithmic Warm-up

Revision: January 11, 2018

Introduction

Welcome to your second programming assignment of the Algorithmic Toolbox at Coursera! It consists of seven programming challenges. The first three challenges require you just to implement carefully the algorithms covered in the lectures. The remaining four challenges will require you to first design an algorithm and then to implement it. For all the challenges, we provide starter solutions in `C++`, `Java`, and `Python3`. These solutions implement straightforward algorithms that usually work only for small inputs. To verify this, you may want to submit these solutions to the grader. This will usually give you a "`time limit exceeded`" message for `Python` starter files and either "`time limit exceeded`" or "`wrong answer`" message for `C++` and `Java` solutions (the reason for wrong answer being an integer overflow issue). Your goal is to replace a naive algorithm with an efficient one. In particular, you may want to use the naive implementation for stress testing your efficient implementation.

In this programming assignment, the grader will show you the input data if your solution fails on any of the tests. This is done to help you to get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail. However, for all the following programming assignments, the grader will show the input data only in case your solution fails on one of the first few tests.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. See the huge difference between a slow algorithm and a fast one.
2. Play with examples where knowing something interesting about a problem helps to design an algorithm that is much faster than a naive one.
3. Implement solutions that work much more faster than straightforward solutions for the following programming challenges:
   (a) compute a small Fibonacci number;
   (b) compute the last digit of a large Fibonacci number;
   (c) compute a huge Fibonacci number modulo m;
   (d) compute the last digit of a sum of Fibonacci numbers;
   (e) compute the last digit of a partial sum of Fibonacci numbers;
   (f) compute the greatest common divisor of two integers;
   (g) compute the least common multiple of two integers.
4. Implement the algorithms covered in the lectures, design new algorithms.
5. Practice implementing, testing, and debugging your solution. In particular, you will find out how in practice, when you implement an algorithm, you bump into unexpected questions and problems not covered by the general description of the algorithm. You will also check your understanding of the algorithm itself and most probably see that there are some aspects you did not think of before you had to actually implement it. You will overcome all those complexities, implement the algorithms, test them, debug, and submit to the system.

Passing Criteria: 4 out of 7

Passing this programming assignment requires passing at least 4 out of 7 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Fibonacci Number

Problem Introduction

Recall the definition of Fibonacci sequence: $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$. Your goal in this problem is to implement an efficient algorithm for computing Fibonacci numbers. The starter files for this problem contain an implementation of the following naive recursive algorithm for computing Fibonacci numbers in `C++`, `Java`, and `Python3`:

```
FIBONACCI(n):
if n ≤ 1:
  return n
return FIBONACCI(n − 1) + FIBONACCI(n − 2)
```

Try compiling and running a starter solution on your machine. You will see that computing, say, F_{40} already takes noticeable time.

Another way to appreciate the dramatic difference between an exponential time algorithm and a polynomial time algorithm is to use the following visualization by David Galles: http://www.cs.usfca.edu/~galles/visualization/DPFib.html. Try computing F_{20} by a recursive algorithm by entering "20" and pressing the "Fibonacci Recursive" button. You will see an endless number of recursive calls. Now, press "Skip Forward" to stop the current algorithm and call the iterative algorithm by pressing "Fibonacci Table". This will compute F_{20} very quickly. (Note that the visualization uses a slightly different definition of Fibonacci numbers: $F_0 = 1$ instead of $F_0 = 0$. This of course has almost no influence on the running time.)

Problem Description

Task. Given an integer n, find the nth Fibonacci number F_n.

Input Format. The input consists of a single integer n.

Constraints. $0 \leq n \leq 45$.

Output Format. Output F_n.

Sample 1.

Input:

```
10
```

Output:

```
55
```

$F_{10} = 55$.

Need Help?

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

2 Last Digit of a Large Fibonacci Number

Problem Introduction

Your goal in this problem is to find the last digit of n-th Fibonacci number. Recall that Fibonacci numbers grow exponentially fast. For example,

$$F_{200} = 280\,571\,172\,992\,510\,140\,037\,611\,932\,413\,038\,677\,189\,525\,.$$

Therefore, a solution like

```
F[0] ← 0
F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
print(F[n] mod 10)
```

will turn out to be too slow, because as i grows the ith iteration of the loop computes the sum of longer and longer numbers. Also, for example, F_{1000} does not fit into the standard C++ `int` type. To overcome this difficulty, you may want to store in $F[i]$ not the ith Fibonacci number itself, but just its last digit (that is, F_i mod 10). Computing the last digit of F_i is easy: it is just the last digit of the sum of the last digits of F_{i-1} and F_{i-2}:

```
F[i] ← (F[i − 1] + F[i − 2]) mod 10
```

This way, all $F[i]$'s are just digits, so they fit perfectly into any standard integer type, and computing a sum of $F[i-1]$ and $F[i-2]$ is performed very quickly.

Problem Description

Task. Given an integer n, find the last digit of the nth Fibonacci number F_n (that is, F_n mod 10).

Input Format. The input consists of a single integer n.

Constraints. $0 \le n \le 10^7$.

Output Format. Output the last digit of F_n.

Sample 1.

Input:

```
3
```

Output:

```
2
```

$F_3 = 2$.

Sample 2.

Input:

```
331
```

Output:

```
9
```

$F_{331} = 668\,996\,615\,388\,005\,031\,531\,000\,081\,241\,745\,415\,306\,766\,517\,246\,774\,551\,964\,595\,292\,186\,469$.

Sample 3.

Input:

```
327305
```

Output:

```
5
```

F_{327305} does not fit into one line of this pdf, but its last digit is equal to 5.

Need Help?

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

3 Greatest Common Divisor

Problem Introduction

The greatest common divisor $\text{GCD}(a, b)$ of two non-negative integers a and b (which are not both equal to 0) is the greatest integer d that divides both a and b. Your goal in this problem is to implement the Euclidean algorithm for computing the greatest common divisor.
Efficient algorithm for computing the greatest common divisor is an important basic primitive of commonly used cryptographic algorithms like RSA.

$$\begin{aligned} & \text{GCD}(1344, 217) \\ = & \text{GCD}(217, 42) \\ = & \text{GCD}(42, 7) \\ = & \text{GCD}(7, 0) \\ = & 7 \end{aligned}$$

Problem Description

Task. Given two integers a and b, find their greatest common divisor.

Input Format. The two integers a, b are given in the same line separated by space.

Constraints. $1 \leq a, b \leq 2 \cdot 10^9$.

Output Format. Output $\text{GCD}(a, b)$.

Sample 1.

Input:

```
18 35
```

Output:

```
1
```

18 and 35 do not have common non-trivial divisors.

Sample 2.

Input:

```
28851538 1183019
```

Output:

```
17657
```

$28851538 = 17657 \cdot 1634$, $1183019 = 17657 \cdot 67$.

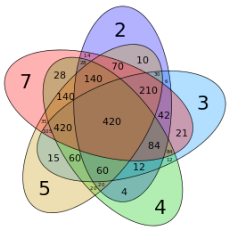
Need Help?

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

4 Least Common Multiple

Problem Introduction

The least common multiple of two positive integers a and b is the least positive integer m that is divisible by both a and b.



Problem Description

Task. Given two integers a and b, find their least common multiple.

Input Format. The two integers a and b are given in the same line separated by space.

Constraints. $1 \leq a, b \leq 2 \cdot 10^9$.

Output Format. Output the least common multiple of a and b.

Sample 1.

Input:

```
6 8
```

Output:

```
24
```

Among all the positive integers that are divisible by both 6 and 8 (e.g., 48, 480, 24), 24 is the smallest one.

Sample 2.

Input:

```
28851538 1183019
```

Output:

```
1933053046
```

1933053046 is the smallest positive integer divisible by both 28851538 and 1183019.

Need Help?

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

5 Fibonacci Number Again

Problem Introduction

In this problem, your goal is to compute F_n modulo m, where n may be really huge: up to 10^{18}. For such values of n, an algorithm looping for n iterations will not fit into one second for sure. Therefore we need to avoid such a loop.

To get an idea how to solve this problem without going through all F_i for i from 0 to n, let's see what happens when m is small — say, $m = 2$ or $m = 3$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_i	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610
$F_i \bmod 2$	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
$F_i \bmod 3$	0	1	1	2	0	2	2	1	0	1	1	2	0	2	2	1

Take a detailed look at this table. Do you see? Both these sequences are periodic! For $m = 2$, the period is 011 and has length 3, while for $m = 3$ the period is 01120221 and has length 8. Therefore, to compute, say, $F_{2015} \bmod 3$ we just need to find the remainder of 2015 when divided by 8. Since $2015 = 251 \cdot 8 + 7$, we conclude that $F_{2015} \bmod 3 = F_7 \bmod 3 = 1$.

This is true in general: for any integer $m \geq 2$, the sequence $F_n \bmod m$ is periodic. The period always starts with 01 and is known as Pisano period.

Problem Description

Task. Given two integers n and m, output $F_n \bmod m$ (that is, the remainder of F_n when divided by m).

Input Format. The input consists of two integers n and m given on the same line (separated by a space).

Constraints. $1 \leq n \leq 10^{18}$, $2 \leq m \leq 10^5$.

Output Format. Output $F_n \bmod m$.

Sample 1.

Input:

```
239 1000
```

Output:

```
161
```

$F_{239} \bmod 1\,000 = 39\,679\,027\,332\,006\,820\,581\,608\,740\,953\,902\,289\,877\,834\,488\,152\,161 \pmod{1\,000} = 161$.

Sample 2.

Input:

```
2816213588 30524
```

Output:

```
10249
```

$F_{2\,816\,213\,588}$ does not fit into one page of this file, but $F_{2\,816\,213\,588} \bmod 30\,524 = 10\,249$.

Need Help?

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

6 Last Digit of the Sum of Fibonacci Numbers

Problem Introduction

The goal in this problem is to find the last digit of a sum of the first n Fibonacci numbers.

Problem Description

Task. Given an integer n, find the last digit of the sum $F_0 + F_1 + \cdots + F_n$.

Input Format. The input consists of a single integer n.

Constraints. $0 \le n \le 10^{14}$.

Output Format. Output the last digit of $F_0 + F_1 + \cdots + F_n$.

Sample 1.

Input:

```
3
```

Output:

```
4
```

$F_0 + F_1 + F_2 + F_3 = 0 + 1 + 1 + 2 = 4$.

Sample 2.

Input:

```
100
```

Output:

```
5
```

The sum is equal to 927 372 692 193 078 999 175, the last digit is 5.

What To Do

Instead of computing this sum in a loop, try to come up with a formula for $F_0 + F_1 + F_2 + \cdots + F_n$. For this, play with small values of n. Then, use a solution for the previous problem.

Need Help?

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

7 Last Digit of the Sum of Fibonacci Numbers Again

Problem Introduction

Now, we would like to find the last digit of a *partial* sum of Fibonacci numbers: $F_m + F_{m+1} + \cdots + F_n$.

Problem Description

Task. Given two non-negative integers m and n, where $m \leq n$, find the last digit of the sum $F_m + F_{m+1} + \cdots + F_n$.

Input Format. The input consists of two non-negative integers m and n separated by a space.

Constraints. $0 \leq m \leq n \leq 10^{18}$.

Output Format. Output the last digit of $F_m + F_{m+1} + \cdots + F_n$.

Sample 1.

Input:

```
3 7
```

Output:

```
1
```

$F_3 + F_4 + F_5 + F_6 + F_7 = 2 + 3 + 5 + 8 + 13 = 31$.

Sample 2.

Input:

```
10 10
```

Output:

```
5
```

$F_{10} = 55$.

Sample 3.

Input:

```
10 200
```

Output:

```
2
```

$F_{10} + F_{11} + \cdots + F_{200} = 734\,544\,867\,157\,818\,093\,234\,908\,902\,110\,449\,296\,423\,262$

Need Help?

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

Programming Assignment 3: Greedy Algorithms

Revision: January 11, 2018

Introduction

In this programming assignment, you will be practicing implementing greedy solutions. As usual, in some problems you just need to implement an algorithm covered in the lectures, while for some others your goal will be to first design an algorithm and then to implement it. Thus, you will be practicing designing an algorithm, proving that it is correct, and implementing it.

Recall that starting from this programming assignment, the grader will show you only the first few tests.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply greedy strategy to solve various computational problems. This will usually require you to design an algorithm that repeatedly makes the most profitable move to construct a solution. You will then need to show that the moves of your algorithm are safe, meaning that they are consistent with at least one optimal solution.
2. Design and implement an efficient greedy algorithm for the following problems:
   (a) changing money with a minimum number of coins;
   (b) maximizing the total value of a loot;
   (c) maximizing revenue in online ad placement;
   (d) minimizing work while collecting signatures;
   (e) maximizing the number of prize places in a competition;
   (f) finally, maximizing your salary!

Passing Criteria: 3 out of 6

Passing this programming assignment requires passing at least 3 out of 6 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Money Change

Problem Introduction

In this problem, you will design and implement an elementary greedy algorithm used by cashiers all over the world millions of times per day.

Problem Description

Task. The goal in this problem is to find the minimum number of coins needed to change the input value (an integer) into coins with denominations 1, 5, and 10.

Input Format. The input consists of a single integer m.

Constraints. $1 \leq m \leq 10^3$.

Output Format. Output the minimum number of coins with denominations 1, 5, 10 that changes m.

Sample 1.

Input:

```
2
```

Output:

```
2
```

$2 = 1 + 1$.

Sample 2.

Input:

```
28
```

Output:

```
6
```

$28 = 10 + 10 + 5 + 1 + 1 + 1$.

Need Help?

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

2 Maximum Value of the Loot

Problem Introduction

A thief finds much more loot than his bag can fit. Help him to find the most valuable combination of items assuming that any fraction of a loot item can be put into his bag.

Problem Description

Task. The goal of this code problem is to implement an algorithm for the fractional knapsack problem.

Input Format. The first line of the input contains the number n of items and the capacity W of a knapsack. The next n lines define the values and weights of the items. The i-th line contains integers v_i and w_i—the value and the weight of i-th item, respectively.

Constraints. $1 \le n \le 10^3$, $0 \le W \le 2 \cdot 10^6$; $0 \le v_i \le 2 \cdot 10^6$, $0 < w_i \le 2 \cdot 10^6$ for all $1 \le i \le n$. All the numbers are integers.

Output Format. Output the maximal value of fractions of items that fit into the knapsack. The absolute value of the difference between the answer of your program and the optimal value should be at most 10^{-3}. To ensure this, output your answer with at least four digits after the decimal point (otherwise your answer, while being computed correctly, can turn out to be wrong because of rounding issues).

Sample 1.

Input:

```
3 50
60 20
100 50
120 30
```

Output:

```
180.0000
```

To achieve the value 180, we take the first item and the third item into the bag.

Sample 2.

Input:

```
1 10
500 30
```

Output:

```
166.6667
```

Here, we just take one third of the only available item.

Need Help?

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

3 Maximum Advertisement Revenue

Problem Introduction

You have n ads to place on a popular Internet page. For each ad, you know how much is the advertiser willing to pay for one click on this ad. You have set up n slots on your page and estimated the expected number of clicks per day for each slot. Now, your goal is to distribute the ads among the slots to maximize the total revenue.

Problem Description

Task. Given two sequences $a_1, a_2, \ldots, a_n$ (a_i is the profit per click of the i-th ad) and $b_1, b_2, \ldots, b_n$ (b_i is the average number of clicks per day of the i-th slot), we need to partition them into n pairs (a_i, b_j) such that the sum of their products is maximized.

Input Format. The first line contains an integer n, the second one contains a sequence of integers $a_1, a_2, \ldots, a_n$, the third one contains a sequence of integers $b_1, b_2, \ldots, b_n$.

Constraints. $1 \le n \le 10^3$; $-10^5 \le a_i, b_i \le 10^5$ for all $1 \le i \le n$.

Output Format. Output the maximum value of $\sum_{i=1}^{n} a_i c_i$, where $c_1, c_2, \ldots, c_n$ is a permutation of $b_1, b_2, \ldots, b_n$.

Sample 1.

Input:

```
1
23
39
```

Output:

```
897
```

$897 = 23 \cdot 39$.

Sample 2.

Input:

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

Output:

```
23
```

$23 = 3 \cdot 4 + 1 \cdot 1 + (-5) \cdot (-2)$.

Need Help?

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

4 Collecting Signatures

Problem Introduction

You are responsible for collecting signatures from all tenants of a certain building. For each tenant, you know a period of time when he or she is at home. You would like to collect all signatures by visiting the building as few times as possible.
The mathematical model for this problem is the following. You are given a set of segments on a line and your goal is to mark as few points on a line as possible so that each segment contains at least one marked point.

Problem Description

Task. Given a set of n segments $\{[a_0, b_0], [a_1, b_1], \ldots, [a_{n-1}, b_{n-1}]\}$ with integer coordinates on a line, find the minimum number m of points such that each segment contains at least one point. That is, find a set of integers X of the minimum size such that for any segment $[a_i, b_i]$ there is a point $x \in X$ such that $a_i \leq x \leq b_i$.

Input Format. The first line of the input contains the number n of segments. Each of the following n lines contains two integers a_i and b_i (separated by a space) defining the coordinates of endpoints of the i-th segment.

Constraints. $1 \leq n \leq 100$; $0 \leq a_i \leq b_i \leq 10^9$ for all $0 \leq i < n$.

Output Format. Output the minimum number m of points on the first line and the integer coordinates of m points (separated by spaces) on the second line. You can output the points in any order. If there are many such sets of points, you can output any set. (It is not difficult to see that there always exist a set of points of the minimum size such that all the coordinates of the points are integers.)

Sample 1.

Input:

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

Output:

```
1
3
```

In this sample, we have three segments: $[1, 3], [2, 5], [3, 6]$ (of length $2, 3, 3$ respectively). All of them contain the point with coordinate 3: $1 \leq 3 \leq 3$, $2 \leq 3 \leq 5$, $3 \leq 3 \leq 6$.

Sample 2.

Input:

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

Output:

```
2
3 6
```

The second and the third segments contain the point with coordinate 3 while the first and the fourth segments contain the point with coordinate 6. All the four segments cannot be covered by a single point, since the segments $[1, 3]$ and $[5, 6]$ are disjoint.

Need Help?

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

5 Maximum Number of Prizes

Problem Introduction

You are organizing a funny competition for children. As a prize fund you have n candies. You would like to use these candies for top k places in a competition with a natural restriction that a higher place gets a larger number of candies. To make as many children happy as possible, you are going to find the largest value of k for which it is possible.

Problem Description

Task. The goal of this problem is to represent a given positive integer n as a sum of as many pairwise distinct positive integers as possible. That is, to find the maximum k such that n can be written as $a_1 + a_2 + \cdots + a_k$ where $a_1, \ldots, a_k$ are positive integers and $a_i \neq a_j$ for all $1 \leq i < j \leq k$.

Input Format. The input consists of a single integer n.

Constraints. $1 \leq n \leq 10^9$.

Output Format. In the first line, output the maximum number k such that n can be represented as a sum of k pairwise distinct positive integers. In the second line, output k pairwise distinct positive integers that sum up to n (if there are many such representations, output any of them).

Sample 1.

Input:

```
6
```

Output:

```
3
1 2 3
```

Sample 2.

Input:

```
8
```

Output:

```
3
1 2 5
```

Sample 3.

Input:

```
2
```

Output:

```
1
2
```

Need Help?

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

6 Maximum Salary

Problem Introduction

As the last question of a successful interview, your boss gives you a few pieces of paper with numbers on it and asks you to compose a largest number from these numbers. The resulting number is going to be your salary, so you are very much interested in maximizing this number. How can you do this?

In the lectures, we considered the following algorithm for composing the largest number out of the given *single-digit numbers*.

```
LargestNumber(Digits):
answer ← empty string
while Digits is not empty:
  maxDigit ← −∞
  for digit in Digits:
    if digit ≥ maxDigit:
      maxDigit ← digit
  append maxDigit to answer
  remove maxDigit from Digits
return answer
```

Unfortunately, this algorithm works only in case the input consists of single-digit numbers. For example, for an input consisting of two integers 23 and 3 (23 is not a single-digit number!) it returns 233, while the largest number is in fact 323. In other words, using the largest number from the input as the first number *is not a safe move*.

Your goal in this problem is to tweak the above algorithm so that it works not only for single-digit numbers, but for arbitrary positive integers.

Problem Description

Task. Compose the largest number out of a set of integers.

Input Format. The first line of the input contains an integer n. The second line contains integers $a_1, a_2, \ldots, a_n$.

Constraints. $1 \leq n \leq 100$; $1 \leq a_i \leq 10^3$ for all $1 \leq i \leq n$.

Output Format. Output the largest number that can be composed out of $a_1, a_2, \ldots, a_n$.

Sample 1.

Input:

```
2
21 2
```

Output:

```
221
```

Note that in this case the above algorithm also returns an incorrect answer 212.

Sample 2.

Input:

```
5
9 4 6 1 9
```

Output:

```
99641
```

In this case, the input consists of single-digit numbers only, so the algorithm above computes the right answer.

Sample 3.

Input:

```
3
23 39 92
```

Output:

```
923923
```

As a coincidence, for this input the above algorithm produces the right result, though the input does not have any single-digit numbers.

What To Do

Interestingly, for solving this problem, all you need to do is to replace the check $digit \geq maxDigit$ with a call IsGreaterOrEqual($digit, maxDigit$) for an appropriately implemented function IsGreaterOrEqual. For example, IsGreaterOrEqual(2, 21) should return `True`.

Need Help?

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

Programming Assignment 4: Divide-and-Conquer

Revision: January 11, 2018

Introduction

In this programming assignment, you will be practicing implementing divide-and-conquer solutions.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply the divide-and-conquer technique to solve various computational problems efficiently. This will usually require you to design an algorithm that solves a problem by splitting it into several disjoint subproblems, solving them recursively, and then combining their results to get an answer for the initial problem.
2. Design and implement efficient algorithms for the following computational problems:
   (a) searching a sorted data for a key;
   (b) finding a majority element in a data;
   (c) improving the quick sort algorithm;
   (d) checking how close a data is to being sorted;
   (e) organizing a lottery;
   (f) finding the closest pair of points.

Passing Criteria: 3 out of 6

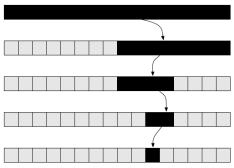
Passing this programming assignment requires passing at least 3 out of 6 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Binary Search

Problem Introduction

In this problem, you will implement the binary search algorithm that allows searching very efficiently (even huge) lists, provided that the list is sorted.



Problem Description

Task. The goal in this code problem is to implement the binary search algorithm.

Input Format. The first line of the input contains an integer n and a sequence $a_0 < a_1 < \ldots < a_{n-1}$ of n pairwise distinct positive integers in increasing order. The next line contains an integer k and k positive integers $b_0, b_1, \ldots, b_{k-1}$.

Constraints. $1 \le n, k \le 10^4$; $1 \le a_i \le 10^9$ for all $0 \le i < n$; $1 \le b_j \le 10^9$ for all $0 \le j < k$;

Output Format. For all i from 0 to $k-1$, output an index $0 \le j \le n-1$ such that $a_j = b_i$ or -1 if there is no such index.

Sample 1.

Input:

```
5 1 5 8 12 13
5 8 1 23 1 11
```

Output:

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

In this sample, we are given an increasing sequence $a_0 = 1$, $a_1 = 5$, $a_2 = 8$, $a_3 = 12$, $a_4 = 13$ of length five and five keys to search: $8, 1, 23, 1, 11$. We see that $a_2 = 8$ and $a_0 = 1$, but the keys 23 and 11 do not appear in the sequence a. For this reason, we output a sequence $2, 0, -1, 0, -1$.

Need Help?

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

2 Majority Element

Problem Introduction

Majority rule is a decision rule that selects the alternative which has a majority, that is, more than half the votes.
Given a sequence of elements $a_1, a_2, \ldots, a_n$, you would like to check whether it contains an element that appears more than $n/2$ times. A naive way to do this is the following.

```
MajorityElement(a_1, a_2, ..., a_n):
for i from 1 to n:
  currentElement ← a_i
  count ← 0
  for j from 1 to n:
    if a_j = currentElement:
      count ← count + 1
  if count > n/2:
    return a_i
return "no majority element"
```

The running time of this algorithm is quadratic. Your goal is to use the divide-and-conquer technique to design an $O(n \log n)$ algorithm.

Problem Description

Task. The goal in this code problem is to check whether an input sequence contains a majority element.

Input Format. The first line contains an integer n, the next one contains a sequence of n non-negative integers $a_0, a_1, \ldots, a_{n-1}$.

Constraints. $1 \le n \le 10^5$; $0 \le a_i \le 10^9$ for all $0 \le i < n$.

Output Format. Output 1 if the sequence contains an element that appears strictly more than $n/2$ times, and 0 otherwise.

Sample 1.

Input:

```
5
2 3 9 2 2
```

Output:

```
1
```

2 is the majority element.

Sample 2.

Input:

```
4
1 2 3 4
```

Output:

```
0
```

There is no majority element in this sequence.

Sample 3.

Input:

```
4
1 2 3 1
```

Output:

```
0
```

This sequence also does not have a majority element (note that the element 1 appears twice and hence is not a majority element).

What To Do

As you might have already guessed, this problem can be solved by the divide-and-conquer algorithm in time $O(n \log n)$. Indeed, if a sequence of length n contains a majority element, then the same element is also a majority element for one of its halves. Thus, to solve this problem you first split a given sequence into halves and make two recursive calls. Do you see how to combine the results of two recursive calls?

It is interesting to note that this problem can also be solved in $O(n)$ time by a more advanced (non-divide and conquer) algorithm that just scans the given sequence twice.

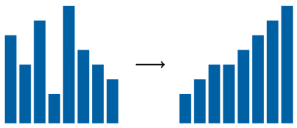
Need Help?

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

3 Improving Quick Sort

Problem Introduction

The goal in this problem is to redesign a given implementation of the randomized quick sort algorithm so that it works fast even on sequences containing many equal elements.



Problem Description

Task. To force the given implementation of the quick sort algorithm to efficiently process sequences with few unique elements, your goal is replace a 2-way partition with a 3-way partition. That is, your new partition procedure should partition the array into three parts: $< x$ part, $= x$ part, and $> x$ part.

Input Format. The first line of the input contains an integer n. The next line contains a sequence of n integers $a_0, a_1, \ldots, a_{n-1}$.

Constraints. $1 \le n \le 10^5$; $1 \le a_i \le 10^9$ for all $0 \le i < n$.

Output Format. Output this sequence sorted in non-decreasing order.

Sample 1.

Input:

```
5
2 3 9 2 2
```

Output:

```
2 2 2 3 9
```

Starter Files

In the starter files, you are given an implementation of the randomized quick sort algorithm using a 2-way partition procedure. This procedure partitions the given array into two parts with respect to a pivot x: $\le x$ part and $> x$ part. As discussed in the video lectures, such an implementation has $\Theta(n^2)$ running time on sequences containing a single unique element. Indeed, the partition procedure in this case splits the array into two parts, one of which is empty and the other one contains $n - 1$ elements. It spends cn time on this. The overall running time is then

$$cn + c(n-1) + c(n-2) + \ldots = \Theta(n^2)\,.$$

What To Do

Implement a 3-way partition procedure and then replace a call to the 2-way partition procedure by a call to the 3-way partition procedure.

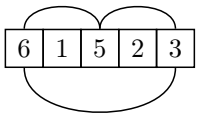
Need Help?

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

4 Number of Inversions

Problem Introduction

An inversion of a sequence $a_0, a_1, \ldots, a_{n-1}$ is a pair of indices $0 \le i < j < n$ such that $a_i > a_j$. The number of inversions of a sequence in some sense measures how close the sequence is to being sorted. For example, a sorted (in non-descending order) sequence contains no inversions at all, while in a sequence sorted in descending order any two elements constitute an inversion (for a total of $n(n-1)/2$ inversions).



Problem Description

Task. The goal in this problem is to count the number of inversions of a given sequence.

Input Format. The first line contains an integer n, the next one contains a sequence of integers $a_0, a_1, \ldots, a_{n-1}$.

Constraints. $1 \le n \le 10^5$, $1 \le a_i \le 10^9$ for all $0 \le i < n$.

Output Format. Output the number of inversions in the sequence.

Sample 1.

Input:

```
5
2 3 9 2 9
```

Output:

```
2
```

The two inversions here are $(1, 3)$ $(a_1 = 3 > 2 = a_3)$ and $(2, 3)$ $(a_2 = 9 > 2 = a_3)$.

What To Do

This problem can be solved by modifying the merge sort algorithm. For this, we change both the `Merge` and `MergeSort` procedures as follows:

- `Merge`(B, C) returns the resulting sorted array and the number of pairs (b, c) such that $b \in B$, $c \in C$, and $b > c$;
- `MergeSort`(A) returns a sorted array A and the number of inversions in A.

Need Help?

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

5 Organizing a Lottery

Problem Introduction

You are organizing an online lottery. To participate, a person bets on a single integer. You then draw several ranges of consecutive integers at random. A participant's payoff then is proportional to the number of ranges that contain the participant's number minus the number of ranges that does not contain it. You need an efficient algorithm for computing the payoffs for all participants. A naive way to do this is to simply scan, for all participants, the list of all ranges. However, you lottery is very popular: you have thousands of participants and thousands of ranges. For this reason, you cannot afford a slow naive algorithm.

Problem Description

Task. You are given a set of points on a line and a set of segments on a line. The goal is to compute, for each point, the number of segments that contain this point.

Input Format. The first line contains two non-negative integers s and p defining the number of segments and the number of points on a line, respectively. The next s lines contain two integers a_i, b_i defining the i-th segment $[a_i, b_i]$. The next line contains p integers defining points $x_1, x_2, \ldots, x_p$.

Constraints. $1 \le s, p \le 50000$; $-10^8 \le a_i \le b_i \le 10^8$ for all $0 \le i < s$; $-10^8 \le x_j \le 10^8$ for all $0 \le j < p$.

Output Format. Output p non-negative integers $k_0, k_1, \ldots, k_{p-1}$ where k_i is the number of segments which contain x_i. More formally,

$$k_i = |\{j : a_j \le x_i \le b_j\}| .$$

Sample 1.

Input:

```
2 3
0 5
7 10
1 6 11
```

Output:

```
1 0 0
```

Here, we have two segments and three points. The first point lies only in the first segment while the remaining two points are outside of all the given segments.

Sample 2.

Input:

```
1 3
-10 10
-100 100 0
```

Output:

```
0 0 1
```

Sample 3.

Input:

```
3 2
0 5
-3 2
7 10
1 6
```

Output:

```
2 0
```

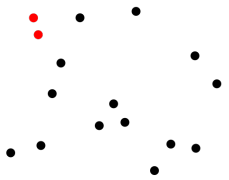
Need Help?

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

6 Closest Points

Problem Introduction

In this problem, your goal is to find the closest pair of points among the given n points. This is a basic primitive in computational geometry having applications in, for example, graphics, computer vision, traffic-control systems.



Problem Description

Task. Given n points on a plane, find the smallest distance between a pair of two (different) points. Recall that the distance between points (x_1, y_1) and (x_2, y_2) is equal to $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Input Format. The first line contains the number n of points. Each of the following n lines defines a point (x_i, y_i).

Constraints. $2 \le n \le 10^5$; $-10^9 \le x_i, y_i \le 10^9$ are integers.

Output Format. Output the minimum distance. The absolute value of the difference between the answer of your program and the optimal value should be at most 10^{-3}. To ensure this, output your answer with at least four digits after the decimal point (otherwise your answer, while being computed correctly, can turn out to be wrong because of rounding issues).

Sample 1.

Input:

```
2
0 0
3 4
```

Output:

```
5.0
```

There are only two points here. The distance between them is 5.

Sample 2.

Input:

```
4
7 7
1 100
4 8
7 7
```

Output:

```
0.0
```

There are two coinciding points among the four given points. Thus, the minimum distance is zero.

Sample 3.

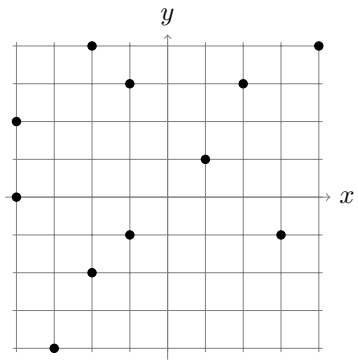
Input:

```
11
4 4
-2 -2
-3 -4
-1 3
2 3
-4 0
1 1
-1 -1
3 -1
-4 2
-2 4
```

Output:

```
1.414213
```

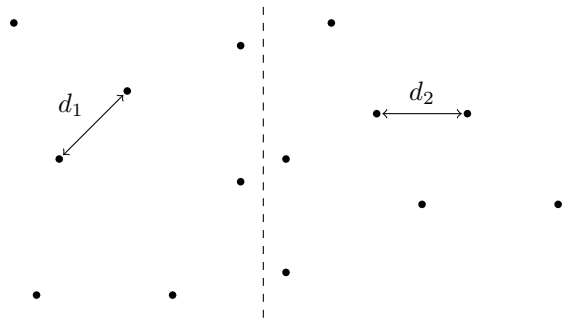
The smallest distance is $\sqrt{2}$. There are two pairs of points at this distance: $(-1,-1)$ and $(-2,-2)$; $(-2,4)$ and $(-1,3)$.



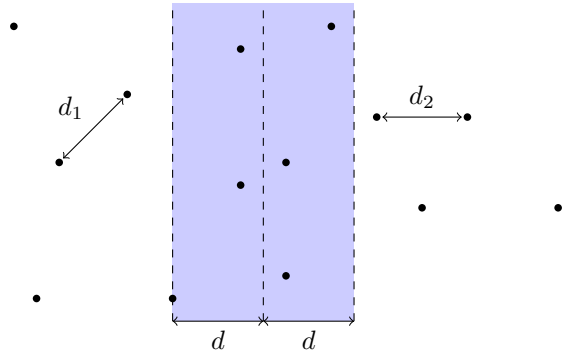
What To Do

This computational geometry problem has many applications in computer graphics and vision. A naive algorithm with quadratic running time iterates through all pairs of points to find the closest pair. Your goal is to design an $O(n \log n)$ time divide and conquer algorithm.

To solve this problem in time $O(n \log n)$, let's first split the given n points by an appropriately chosen vertical line into two halves S_1 and S_2 of size $\frac{n}{2}$ (assume for simplicity that all x-coordinates of the input points are different). By making two recursive calls for the sets S_1 and S_2, we find the minimum distances d_1 and d_2 in these subsets. Let $d = \min\{d_1, d_2\}$.

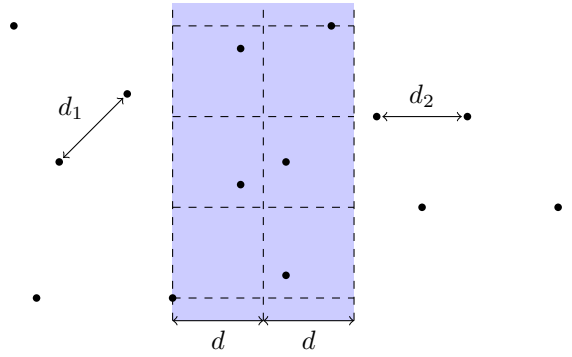


It remains to check whether there exist points $p_1 \in S_1$ and $p_2 \in S_2$ such that the distance between them is smaller than d. We cannot afford to check all possible such pairs since there are $\frac{n}{2} \cdot \frac{n}{2} = \Theta(n^2)$ of them. To check this faster, we first discard all points from S_1 and S_2 whose x-distance to the middle line is greater than d. That is, we focus on the following strip:



Stop and think: Why can we narrow the search to this strip? Now, let's sort the points of the strip by their y-coordinates and denote the resulting sorted list by $P = [p_1, \ldots, p_k]$. It turns out that if $|i - j| > 7$, then the distance between points p_i and p_j is greater than d for sure. This follows from the Exercise Break below.

Exercise break: Partition the strip into $d \times d$ squares as shown below and show that each such square contains at most four input points.



This results in the following algorithm. We first sort the given n points by their x-coordinates and then split the resulting sorted list into two halves S_1 and S_2 of size $\frac{n}{2}$. By making a recursive call for each of the sets S_1 and S_2, we find the minimum distances d_1 and d_2 in them. Let $d = \min\{d_1, d_2\}$. However, we are not done yet as we also need to find the minimum distance between points from different sets (i.e, a point from S_1 and a point from S_2) and check whether it is smaller than d. To perform such a check, we filter the initial point set and keep only those points whose x-distance to the middle line does not exceed d. Afterwards, we sort the set of points in the resulting strip by their y-coordinates and scan the resulting list of points. For each point, we compute its distance to the seven subsequent points in this list and compute d', the minimum distance that we encountered during this scan. Afterwards, we return $\min\{d, d'\}$.

The running time of the algorithm satisfies the recurrence relation

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n \log n)\,.$$

The $O(n \log n)$ term comes from sorting the points in the strip by their y-coordinates at every iteration.

Exercise break: Prove that $T(n) = O(n \log^2 n)$ by analyzing the recursion tree of the algorithm.

Exercise break: Show how to bring the running time down to $O(n \log n)$ by avoiding sorting at each recursive call.

Need Help?

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

Programming Assignment 5: Dynamic Programming 1

Revision: January 11, 2018

Introduction

In this programming assignment, you will be practicing implementing dynamic programming solutions. As usual, in some code problems you just need to implement an algorithm covered in the lectures, while for some others your goal will be to first design an algorithm and then implement it.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply the dynamic programming technique to solve various computational problems. This will usually require you to design an algorithm that solves a problem by solving a collection of overlapping subproblems (as opposed to the divide-and-conquer technique where subproblems are usually disjoint) and combining the results.
2. See examples of optimization problems where a natural greedy strategy produces a non-optimal result. You will see that a natural greedy move for these problems is not safe.
3. Design and implement an efficient algorithm for the following computational problems:
   (a) Implement an efficient algorithm to compute the difference between two files or strings. Such algorithms are widely used in spell checking programs and version control systems.
   (b) Design and implement a dynamic programming algorithm for a novel computational problem.

Passing Criteria: 3 out of 5

Passing this programming assignment requires passing at least 3 out of 5 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Money Change Again

As we already know, a natural greedy strategy for the change problem does not work correctly for any set of denominations. For example, if the available denominations are 1, 3, and 4, the greedy algorithm will change 6 cents using three coins $(4 + 1 + 1)$ while it can be changed using just two coins $(3 + 3)$. Your goal now is to apply dynamic programming for solving the Money Change Problem for denominations 1, 3, and 4.

Problem Description

Input Format. Integer *money*.

Output Format. The minimum number of coins with denominations 1, 3, 4 that changes *money*.

Constraints. $1 \leq money \leq 10^3$.

Sample 1.

Input:

```
2
```

Output:

```
2
```

$2 = 1 + 1$.

Sample 2.

Input:

```
34
```

Output:

```
9
```

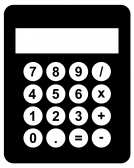
$34 = 3 + 3 + 4 + 4 + 4 + 4 + 4 + 4 + 4$.

Need Help?

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

2 Primitive Calculator

Problem Introduction



You are given a primitive calculator that can perform the following three operations with the current number x: multiply x by 2, multiply x by 3, or add 1 to x. Your goal is given a positive integer n, find the minimum number of operations needed to obtain the number n starting from the number 1.

Problem Description

Task. Given an integer n, compute the minimum number of operations needed to obtain the number n starting from the number 1.

Input Format. The input consists of a single integer $1 \leq n \leq 10^6$.

Output Format. In the first line, output the minimum number k of operations needed to get n from 1. In the second line output a sequence of intermediate numbers. That is, the second line should contain positive integers $a_0, a_2, \ldots, a_{k-1}$ such that $a_0 = 1$, $a_{k-1} = n$ and for all $0 \leq i < k - 1$, a_{i+1} is equal to either $a_i + 1$, $2a_i$, or $3a_i$. If there are many such sequences, output any one of them.

Sample 1.

Input:

```
1
```

Output:

```
0
1
```

Sample 2.

Input:

```
5
```

Output:

```
3
1 2 4 5
```

Here, we first multiply 1 by 2 two times and then add 1. Another possibility is to first multiply by 3 and then add 1 two times. Hence "1 3 4 5" is also a valid output in this case.

Sample 3.

Input:

```
96234
```

Output:

```
14
1 3 9 10 11 22 66 198 594 1782 5346 16038 16039 32078 96234
```

Again, another valid output in this case is "1 3 9 10 11 33 99 297 891 2673 8019 16038 16039 48117 96234".

Starter Files

Going from 1 to n is the same as going from n to 1, each time either dividing the current number by 2 or 3 or subtracting 1 from it. Since we would like to go from n to 1 as fast as possible it is natural to repeatedly reduce n as much as possible. That is, at each step we replace n by $\min\{n/3, n/2, n-1\}$ (the terms $n/3$ and $n/2$ are used only when n is divisible by 3 and 2, respectively). We do this until we reach 1. This gives rise to the following algorithm and it is implemented in the starter files:

```
GreedyCalculator(n):
numOperations ← 0
while n > 1:
  numOperations ← numOperations + 1
  if n mod 3 = 0:
    n ← n/3
  else if n mod 2 = 0:
    n ← n/2
  else:
    n ← n − 1
return numOperations
```

This seemingly correct algorithm is in fact incorrect. You may want to submit one of the starter files to ensure this. Hence in this case moving from n to $\min\{n/3, n/2, n-1\}$ is not *safe*.

Need Help?

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

3 Edit Distance

Problem Introduction

The edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions of symbols) to transform one string into another. It is a measure of similarity of two strings. Edit distance has applications, for example, in computational biology, natural language processing, and spell checking. Your goal in this problem is to compute the edit distance between two strings.

Problem Description

Task. The goal of this problem is to implement the algorithm for computing the edit distance between two strings.

Input Format. Each of the two lines of the input contains a string consisting of lower case latin letters.

Constraints. The length of both strings is at least 1 and at most 100.

Output Format. Output the edit distance between the given two strings.

Sample 1.

Input:

```
ab
ab
```

Output:

```
0
```

Sample 2.

Input:

```
short
ports
```

Output:

```
3
```

An alignment of total cost 3:

s	h	o	r	t	–
–	p	o	r	t	s

Sample 3.

Input:

```
editing
distance
```

Output:

```
5
```

An alignment of total cost 5:

e	d	i	–	t	i	n	g	–
–	d	i	s	t	a	n	c	e

Need Help?

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

4 Longest Common Subsequence of Two Sequences

Problem Introduction

Compute the length of a longest common subsequence of three sequences.

Problem Description

Task. Given two sequences $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$, find the length of their longest common subsequence, i.e., the largest non-negative integer p such that there exist indices $1 \le i_1 < i_2 < \cdots < i_p \le n$ and $1 \le j_1 < j_2 < \cdots < j_p \le m$, such that $a_{i_1} = b_{j_1}, \ldots, a_{i_p} = b_{j_p}$.

Input Format. First line: n. Second line: $a_1, a_2, \ldots, a_n$. Third line: m. Fourth line: $b_1, b_2, \ldots, b_m$.

Constraints. $1 \le n, m \le 100$; $-10^9 < a_i, b_i < 10^9$.

Output Format. Output p.

Sample 1.

Input:

```
3
2 7 5
2
2 5
```

Output:

```
2
```

A common subsequence of length 2 is $(2, 5)$.

Sample 2.

Input:

```
1
7
4
1 2 3 4
```

Output:

```
0
```

The two sequences do not share elements.

Sample 3.

Input:

```
4
2 7 8 3
4
5 2 8 7
```

Output:

```
2
```

One common subsequence is $(2, 7)$. Another one is $(2, 8)$.

Need Help?

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

5 Longest Common Subsequence of Three Sequences

Problem Introduction

Compute the length of a longest common subsequence of three sequences.

Problem Description

Task. Given three sequences $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_m)$, and $C = (c_1, c_2, \ldots, c_l)$, find the length of their longest common subsequence, i.e., the largest non-negative integer p such that there exist indices $1 \le i_1 < i_2 < \cdots < i_p \le n$, $1 \le j_1 < j_2 < \cdots < j_p \le m$, $1 \le k_1 < k_2 < \cdots < k_p \le l$ such that $a_{i_1} = b_{j_1} = c_{k_1}, \ldots, a_{i_p} = b_{j_p} = c_{k_p}$

Input Format. First line: n. Second line: $a_1, a_2, \ldots, a_n$. Third line: m. Fourth line: $b_1, b_2, \ldots, b_m$. Fifth line: l. Sixth line: $c_1, c_2, \ldots, c_l$.

Constraints. $1 \le n, m, l \le 100$; $-10^9 < a_i, b_i, c_i < 10^9$.

Output Format. Output p.

Sample 1.

Input:

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

Output:

```
2
```

A common subsequence of length 2 is $(1, 3)$.

Sample 2.

Input:

```
5
8 3 2 1 7
7
8 2 1 3 8 10 7
6
6 8 3 1 4 7
```

Output:

```
3
```

One common subsequence of length 3 in this case is $(8, 3, 7)$. Another one is $(8, 1, 7)$.

Need Help?

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

Programming Assignment 6: Dynamic Programming 2

Revision: January 11, 2018

Introduction

In this programming assignment, you will continue practicing implementing dynamic programming solutions.

Passing Criteria: 2 out of 3

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

Contents

1 Maximum Amount of Gold

Problem Introduction

You are given a set of bars of gold and your goal is to take as much gold as possible into your bag. There is just one copy of each bar and for each bar you can either take it or not (hence you cannot take a fraction of a bar).

Problem Description

Task. Given n gold bars, find the maximum weight of gold that fits into a bag of capacity W.

Input Format. The first line of the input contains the capacity W of a knapsack and the number n of bars of gold. The next line contains n integers $w_0, w_1, \ldots, w_{n-1}$ defining the weights of the bars of gold.

Constraints. $1 \leq W \leq 10^4$; $1 \leq n \leq 300$; $0 \leq w_0, \ldots, w_{n-1} \leq 10^5$.

Output Format. Output the maximum weight of gold that fits into a knapsack of capacity W.

Sample 1.

Input:

```
10 3
1 4 8
```

Output:

```
9
```

Here, the sum of the weights of the first and the last bar is equal to 9.

Starter Files

Starter files contain an implementation of the following greedy strategy: scan the list of given bars of gold and add the current bar if it fits into the current capacity (note that, in this problem, all the items have the same value per unit of weight, for a simple reasons: they are all made of gold). As you already know from the lectures, such a greedy move is not safe. You may want to additionally submit a starter file as a solution to the grading system to ensure that this greedy algorithm indeed might produce a non-optimal result.

Need Help?

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

2 Partitioning Souvenirs

You and two of your friends have just returned back home after visiting various countries. Now you would like to evenly split all the souvenirs that all three of you bought.

Problem Description

Input Format. The first line contains an integer n. The second line contains integers $v_1, v_2, \ldots, v_n$ separated by spaces.

Constraints. $1 \leq n \leq 20$, $1 \leq v_i \leq 30$ for all i.

Output Format. Output 1, if it possible to partition $v_1, v_2, \ldots, v_n$ into three subsets with equal sums, and 0 otherwise.

Sample 1.

Input:

```
4
3 3 3 3
```

Output:

```
0
```

Sample 2.

Input:

```
1
40
```

Output:

```
0
```

Sample 3.

Input:

```
11
17 59 34 57 17 23 67 1 18 2 59
```

Output:

```
1
```

$34 + 67 + 17 = 23 + 59 + 1 + 17 + 18 = 59 + 2 + 57$.

Sample 4.

Input:

```
13
1 2 3 4 5 5 7 7 8 10 12 19 25
```

Output:

```
1
```

$1 + 3 + 7 + 25 = 2 + 4 + 5 + 7 + 8 + 10 = 5 + 12 + 19$.

3 Maximum Value of an Arithmetic Expression

Problem Introduction

In this problem, your goal is to add parentheses to a given arithmetic expression to maximize its value.

$$\max(5-8+7\times4-8+9) = ?$$

Problem Description

Task. Find the maximum value of an arithmetic expression by specifying the order of applying its arithmetic operations using additional parentheses.

Input Format. The only line of the input contains a string s of length $2n+1$ for some n, with symbols $s_0, s_1, \ldots, s_{2n}$. Each symbol at an even position of s is a digit (that is, an integer from 0 to 9) while each symbol at an odd position is one of three operations from `{+,-,*}`.

Constraints. $1 \leq n \leq 14$ (hence the string contains at most 29 symbols).

Output Format. Output the maximum possible value of the given arithmetic expression among different orders of applying arithmetic operations.

Sample 1.

Input:

```
1+5
```

Output:

```
6
```

Sample 2.

Input:

```
5-8+7*4-8+9
```

Output:

```
200
```

$200 = (5 - ((8 + 7) \times (4 - (8 + 9))))$

Need Help?

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