Module: Greedy Algorithms (Week 3)
Course: Algorithmic Toolbox
Specialization: Data Structures and Algorithms

Programming Assignment 2: Greedy Algorithms

Revision: February 17, 2016

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 (see the questions 6.4 and 6.5 in the FAQ section).

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. Implement an efficient algorithm to optimally pack your bag before going to a hike.
3. Design and implement a greedy algorithm for a novel computational problem.
4. Design and implement a greedy algorithm for changing money optimally.

Passing Criteria: 2 out of 4

Passing this programming assignment requires passing at least 2 out of 4 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: Changing Money

Problem Introduction

In this problem, you will design an algorithm for changing money optimally.

Problem Description

Task. The goal in this problem is to find the minimum number of coins needed to change the given amount of money using 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.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
2
```

Output:

```
2
```

Explanation:
$2 = 1 + 1$.

Sample 2.

Input:

```
28
```

Output:

```
6
```

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

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

What To Do

To design a greedy algorithm for this problem, play with small values of m (consider, for example, sample tests).

2 Problem: Fractional Knapsack

Problem Introduction

Given a set of items and total capacity of a knapsack, find the maximal value of fractions of items that fit into the knapsack.

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 contain integers v_i and w_i — the weight and the value 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).

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

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

Output:

```
180.0000
```

Explanation:
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
```

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

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

3 Problem: Covering Segments by Points

Problem Introduction

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.)

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

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

Output:

```
1
3
```

Explanation:
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
```

Explanation:
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.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

What To Do

To design a greedy algorithm for this problem, consider a segment with the minimum right endpoint. What is a safe way to cover it by a point?

4 Problem: Pairwise Distinct Summands

Problem Introduction

This is an example of a problem where a subproblem of the corresponding greedy algorithm is slightly distinct from the initial problem.

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).

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
6
```

Output:

```
3
1 2 3
```

Sample 2.

Input:

```
8
```

Output:

```
5
1 2 5
```

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

What To Do

To find an algorithm for this problem, you may want to play a little bit with small numbers. Assume, for example, that we want to represent 8 as a sum of as many pairwise distinct summands as possible. Well, it is natural to try to use 1 as the first summand, right? Then, the remaining problem is to represent 7 as a sum of the maximum number of pairwise distinct positive integers none of which is equal to 1. We then take 2 and are left with the following problem: represent 5 as a sum of distinct positive integers each of which is

at least 3. Clearly, we cannot use two summands in this case (do you see why?). Overall, this gives us the following optimal representation: $8 = 1 + 2 + 5$.

In general, our subproblem is the following: represent an integer $k \geq 1$ as a sum of as many pairwise distinct integers each of which is at least $l \leq k$ as possible. If $k \leq 2l$, then the answer is clearly 1. Otherwise it is *safe* to use l as one of the summands. Let's state and prove this formally.

Lemma 4.1. *Let $k > 2l$ and let p be the maximum number such that k can be represented as a sum of p pairwise distinct summands each of which is at least l. Then there exists an optimal representation $k = a_1 + \cdots + a_p$ (that is, all a_i's are no smaller than l and are pairwise distinct) such that $a_1 = l$.*

Proof. Consider some optimal representation $k = b_1 + \cdots + b_p$. Without loss of generality assume that $b_1 < b_2 < \ldots < b_p$. We know that $p \geq 2$ (as $k > 2l$). If $b_1 = l$, then we are already done. Otherwise let $\Delta = b_1 - l \geq 1$. Consider the following representation of n:

$$n = (b_1 - \Delta) + b_2 + \cdots + (b_p + \Delta)\,.$$

It is not difficult to see that this is a valid and optimal representation (it contains p summands and they are all distinct since $(b_1 - \Delta) < b_2 < \ldots < (b_p + \Delta)$) □

It is not difficult to see that our initial problem has the same type: we need to represent an integer n as a sum of the maximum number of pairwise distinct integers each of which is at least 1 (hence $k = n$ and $l = 1$).

To summarize, initially we have $k = n$ and $l = 1$. To solve a (k, l)-subproblem, we do the following. If $k \leq 2l$, we just return 1. Otherwise we return 1 plus the answer for the subproblem $(k - l, l + 1)$.

5 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.

5.1 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.

5.2 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.

5.3 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++`, `Java`, `Python2`, `Python3`. 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. If you are going to use `C` or `Python2`, you need to implement a solution from scratch.

5.4 Compiling Your Program

For solving programming assignments, you can use any of the following programming languages: `C`, `C++`, `Java`, `Python2`, or `Python3`. 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.

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
```

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

```
g++ -pipe -O2 -std=c++11
```

  If your `C/C++` compiler does not recognize `-std=c++11` 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 (Open JDK 7)`. File extensions: `.java`. No flags:

```
javac
```

- `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: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python3"). No flags:

```
python3
```

5.5 Testing Your Program

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

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

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

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

5.6 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 5.4) 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).

5.7 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 5.5. 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!

6 Frequently Asked Questions

6.1 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 5.3 and 5.4). 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.

6.2 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.

6.3 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, the output of your program and the correct answer. Otherwise, you won't know the input, the output 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, the output of your program and the correct answer. Otherwise, you won't know the input, the output 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.

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

6.4 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.

6.5 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.

6.6 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).