Programming Assignment 3: Greedy Algorithms

Revision: April 21, 2019

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
```

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

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.

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
```

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

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.

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
```

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`.