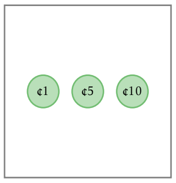
Chapter 3: Greedy Algorithms

In this chapter, you will learn about seemingly naive yet powerful greedy algorithms. After learning the key idea behind the greedy algorithms, some students feel that they represent the algorithmic Swiss army knife that can be applied to solve nearly all programming challenges in this book. Be warned: since this intuitive idea rarely works in practice, you have to prove that your greedy algorithm produces an optimal solution!



Money Change
Compute the minimum number of coins needed to change the given value into coins with denominations 1, 5, and 10.
Input. An integer *money*.
Output. The minimum number of coins with denominations 1, 5, and 10 that changes *money*.

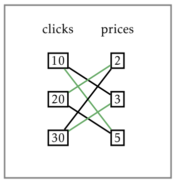
Maximizing the Value of the Loot
Find the maximal value of items that fit into the backpack.
Input. The capacity of a backpack W as well as the weights $(w_1,\ldots,w_n)$ and costs $(c_1,\ldots,c_n)$ of n different compounds.
Output. The maximum total value of fractions of items that fit into the backpack of the given capacity: i.e., the maximum value of $c_1 \cdot f_1 + \cdots + c_n \cdot f_n$ such that $w_1 \cdot f_1 + \cdots + w_n \cdot f_n \leq W$ and $0 \leq f_i \leq 1$ for all i (f_i is the fraction of the i-th item taken to the backpack).

Car Fueling

Compute the minimum number of gas tank refills to get from one city to another.

Input. Integers d and m, as well as a sequence of integers $stop_1 < stop_2 < \cdots < stop_n$.

Output. The minimum number of refills to get from one city to another if a car can travel at most m miles on a full tank. The distance between the cities is d miles and there are gas stations at distances $stop_1, stop_2, \ldots, stop_n$ along the way. We assume that a car starts with a full tank.

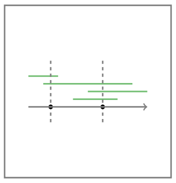


Maximum Product of Two Sequences

Find the maximum dot product of two sequences of numbers.

Input. Two sequences of n positive integers: $price_1, \ldots, price_n$ and $clicks_1, \ldots, clicks_n$.

Output. The maximum value of $price_1 \cdot c_1 + \cdots + price_n \cdot c_n$, where $c_1, \ldots, c_n$ is a permutation of $clicks_1, \ldots, clicks_n$.

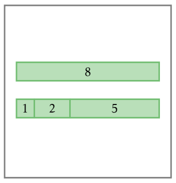


Covering Segments by Points

Find the minimum number of points needed to cover all given segments on a line.

Input. A sequence of n segments $[l_1, r_1], \ldots, [l_n, r_n]$ on a line.

Output. A set of points of minimum size such that each segment $[l_i, r_i]$ contains a point, i.e., there exists a point x from this set such that $l_i \leq x \leq r_i$.



Distinct Summands

Represent a positive integer as the sum of the maximum number of pairwise distinct positive integers.

Input. A positive integer n.

Output. The maximum k such that n can be represented as the sum $a_1 + \cdots + a_k$ of k distinct positive integers.

Largest Concatenate

Compile the largest number by concatenating the given numbers.

Input. A sequence of positive integers.

Output. The largest number that can be obtained by concatenating the given integers in some order.

3.1 The Main Idea

3.1.1 Examples

A greedy algorithm builds a solution piece by piece and at each step, chooses the most profitable piece. This is best illustrated with examples.

Our first example is the Largest Concatenate Problem: given a sequence of single-digit numbers, find the largest number that can be obtained by concatenating these numbers. For example, for the input sequence $(2,3,9,1,2)$, the output is the number 93221. It is easy to come up with an algorithm for this problem. Clearly, the largest single-digit number should be selected as the first digit of the concatenate. Afterward, we face essentially the same problem: concatenate the remaining numbers to get as large number as possible.

```
LargestConcatenate(Numbers):
result ← empty string
while Numbers is not empty:
   maxNumber ← largest among Numbers
   append maxNumber to result
   remove maxNumber from Numbers
return result
```

Our second example is the Money Change Problem: given a non-negative integer *money*, find the minimum number of coins with denominations 1, 5, and 10 that changes *money*. For example, the minimum number of coins needed to change $money = 28$ is 6: $28 = 10 + 10 + 5 + 1 + 1 + 1$. This representation of 28 already suggests an algorithm. We take a coin c with the largest denomination that does not exceed *money*. Afterward, we face essentially the same problem: change $(money - c)$ with the minimum number of coins.

```
Change(money, Denominations):
numCoins ← 0
while money > 0:
   maxCoin ← largest among Denominations that does not exceed money
   money ← money − maxCoin
   numCoins ← numCoins + 1
return numCoins
```

Exercise Break. What does LargestConcatenate([2, 21]) return?

Exercise Break. What does Change(8, [1, 4, 6]) return?

If you use the same greedy strategy, then LargestConcatenate([2, 21]) returns 212 and Change(8, [1, 4, 6]) returns 3 because $8 = 6 + 1 + 1$. But this strategy fails because the correct solutions are 221 (concatenating 2 with 21) and 2 because $8 = 4 + 4$!

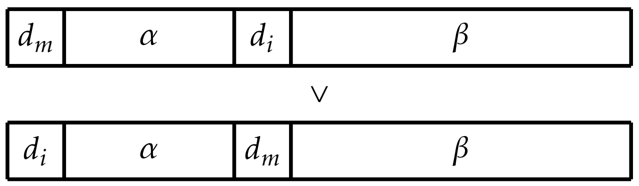
Thus, in *rare* cases when a greedy strategy works, one should be able to prove its correctness: A priori, there should be no reason why a sequence of *locally* optimal moves leads to a *global* optimum!

3.1.2 Proving Correctness of Greedy Algorithms

At each step, a greedy algorithm restricts the search space by selecting a most "profitable" piece of a solution. For example, instead of considering all possible concatenations, the LargestConcatenate algorithm only considers concatenations starting from the largest digit. Instead of all possible ways of changing money, the Change algorithm considers only the ones that include a coin with the largest denomination (that does not exceed *money*). What one needs to prove is that this restricted search space still contains at least one optimal solution. This is usually done as follows.

> Consider an arbitrary optimal solution. If it belongs to the restricted search space, then we are done. If it does not belong to the restricted search space, tweak it so that it is still optimum and belongs to the restricted search space.

Here, we will prove the correctness of LargestConcatenate for single digit numbers (the correctness of Change for denominations 1, 5, and 10 will be given in Section 3.2.1). Let N be the largest number that can be obtained by concatenating digits $d_1, \ldots, d_n$ in some order and let d_m be the largest digit. Then, N starts with d_m. Indeed, assume that N starts with some other digit $d_i < d_m$. Then N has the form $d_i \alpha d_m \beta$ where α, β are (possibly empty) sequences of digits. But if we swap d_i and d_m, we get a larger number!



Stop and Think. What part of this proof breaks for multi-digit numbers?

3.1.3 Implementation

A greedy solution chooses the most profitable move and then continues to solve the remaining problem that usually has the same type as the initial one. There are two natural implementations of this strategy: either iterative with a while loop or recursive. Iterative solutions for the Largest Concatenate and Money Change problems are given above. Below are their recursive variants.

```
LargestConcatenate(Numbers):
if Numbers is empty:
   return empty string
maxNumber ← largest among Numbers
remove maxNumber from Numbers
return concatenate of maxNumber and LargestConcatenate(Numbers)
```

```
Change(money, Denominations):
if $money = 0$:
   return 0
   maxCoin ← largest among Denominations that does not exceed money
return $1 +$ Change($money - maxCoin$, Denominations)
```

For all the following programming challenges, we provide both iterative and recursive Python solutions.

3.2 Programming Challenges

3.2.1 Money Change

Money Change Problem
Compute the minimum number of coins needed to change the given value into coins with denominations 1, 5, and 10.

Input: An integer *money*.
Output: The minimum number of coins with denominations 1, 5, and 10 that changes *money*.

In this problem, you will implement a simple greedy algorithm used by cashiers all over the world. We assume that a cashier has unlimited number of coins of each denomination.

Input format. Integer *money*.

Output format. The minimum number of coins with denominations 1, 5, 10 that changes *money*.

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

Sample 1.

Input:

```
2
```

Output:

```
2
```

$2 = 1 + 1$.

Sample 2.

Input:

```
28
```

Output:

```
6
```

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

3.2.2 Maximum Value of the Loot

Maximizing the Value of the Loot Problem
Find the maximal value of items that fit into the backpack.

Input: The capacity of a backpack W as well as the weights $(w_1,\ldots,w_n)$ and costs $(c_1,\ldots,c_n)$ of n different compounds.
Output: The maximum total value of fractions of items that fit into the backpack of the given capacity: i.e., the maximum value of $c_1 \cdot f_1 + \cdots + c_n \cdot f_n$ such that $w_1 \cdot f_1 + \cdots + w_n \cdot f_n \le W$ and $0 \le f_i \le 1$ for all i (f_i is the fraction of the i-th item taken to the backpack).

A thief breaks into a spice shop and finds four pounds of saffron, three pounds of vanilla, and five pounds of cinnamon. His backpack fits at most nine pounds, therefore he cannot take everything. Assuming that the prices of saffron, vanilla, and cinnamon are \$5 000, \$200, and \$10, respectively, what is the most valuable loot in this case? If the thief takes u_1 pounds of saffron, u_2 pounds of vanilla, and u_3 pounds of cinnamon, the total value of the loot is

$$5\,000 \cdot \frac{u_1}{4} + 200 \cdot \frac{u_2}{3} + 10 \cdot \frac{u_3}{5}.$$

The thief would like to maximize the value of this expression subject to the following constraints: $u_1 \le 4$, $u_2 \le 3$, $u_3 \le 5$, $u_1 + u_2 + u_3 \le 9$.

Input format. The first line of the input contains the number n of compounds and the capacity W of a backpack. The next n lines define the costs and weights of the compounds. The i-th line contains the cost c_i and the weight w_i of the i-th compound.

Output format. Output the maximum value of compounds that fit into the backpack.

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

Bells and whistles. Although the Input to this problem consists of integers, the Output may be non-integer. Therefore, 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 the rounding issues).

Sample 1.

Input:

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

Output:

```
180.0000
```

To achieve the value 180, the thief takes the entire first compound and the entire third compound.

Sample 2.

Input:

```
1 10
500 30
```

Output:

```
166.6667
```

The thief should take ten pounds of the only available compound.

3.2.3 Car Fueling

Car Fueling Problem
Compute the minimum number of gas tank refills to get from one city to another.

Input: Integers d and m, as well as a sequence of integers $stop_1 < stop_2 < \cdots < stop_n$.
Output: The minimum number of refills to get from one city to another if a car can travel at most m miles on a full tank. The distance between the cities is d miles and there are gas stations at distances $stop_1, stop_2, \ldots, stop_n$ along the way. We assume that a car starts with a full tank.

Try our Car Fueling interactive puzzle before solving this programming challenge!

Input format. The first line contains an integer d. The second line contains an integer m. The third line specifies an integer n. Finally, the last line contains integers $stop_1, stop_2, \ldots, stop_n$.

Output format. The minimum number of refills needed. If it is not possible to reach the destination, output -1.

Constraints. $1 \leq d \leq 10^5$. $1 \leq m \leq 400$. $1 \leq n \leq 300$. $0 < stop_1 < stop_2 < \cdots < stop_n < d$.

Sample 1.

Input:

```
950
400
4
200 375 550 750
```

Output:

```
2
```

The distance between the cities is 950, the car can travel at most 400 miles on a full tank. It suffices to make two refills: at distance 375 and 750. This is the minimum number of refills as with a single refill one would only be able to travel at most 800 miles.

Sample 2.

Input:

```
10
3
4
1 2 5 9
```

Output:

```
-1
```

One cannot reach the gas station at point 9 as the previous gas station is too far away.

Sample 3.

Input:

```
200
250
2
100 150
```

Output:

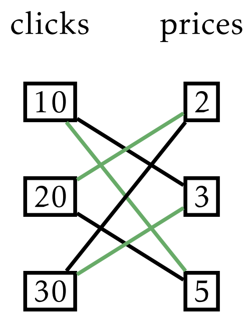
```
0
```

There is no need to refill the tank as the car starts with a full tank and can travel for 250 miles whereas the distance to the destination is 200 miles.

3.2.4 Maximum Advertisement Revenue

Maximum Product of Two Sequences Problem
Find the maximum dot product of two sequences of numbers.

> **Input:** Two sequences of n positive integers: $price_1, \ldots, price_n$ and $clicks_1, \ldots, clicks_n$.
> **Output:** The maximum value of $price_1 \cdot c_1 + \cdots + price_n \cdot c_n$, where $c_1, \ldots, c_n$ is a permutation of $clicks_1, \ldots, clicks_n$.



You have $n = 3$ advertisement slots on your popular Internet page and you want to sell them to advertisers. They expect, respectively, $clicks_1 = 10$, $clicks_2 = 20$, and $clicks_3 = 30$ clicks per day. You found three advertisers willing to pay $price_1 = \$2$, $price_2 = \$3$, and $price_3 = \$5$ per click. How would you pair the slots and advertisers to maximize the revenue? For example, the blue pairing shown above gives a revenue of $10 \cdot 5 + 20 \cdot 2 + 30 \cdot 3 = 180$ dollars, while the black one results in revenue of $10 \cdot 3 + 20 \cdot 5 + 30 \cdot 2 = 190$ dollars.

Input format. The first line contains an integer n, the second one contains a sequence of integers $price_1, \ldots, price_n$, the third one contains a sequence of integers $clicks_1, \ldots, clicks_n$.

Output format. Output the maximum value of $(price_1 \cdot c_1 + \cdots + price_n \cdot c_n)$, where $c_1, \ldots, c_n$ is a permutation of $clicks_1, \ldots, clicks_n$.

Constraints. $1 \le n \le 10^3$; $0 \le price_i, clicks_i \le 10^5$ for all $1 \le i \le n$.

Sample 1.

Input:

```
1
23
39
```

Output:

```
897
```

$897 = 23 \cdot 39$.

Sample 2.

Input:

```
3
2 3 9
7 4 2
```

Output:

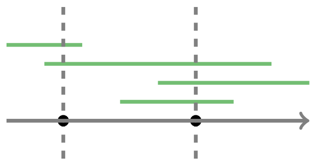
```
79
```

$79 = 7 \cdot 9 + 2 \cdot 2 + 3 \cdot 4$.

3.2.5 Collecting Signatures

Covering Segments by Points Problem
Find the minimum number of points needed to cover all given segments on a line.

> **Input:** A sequence of n segments $[l_1, r_1], \ldots, [l_n, r_n]$ on a line.
> **Output:** A set of points of minimum size such that each segment $[l_i, r_i]$ contains a point, i.e., there exists a point x from this set such that $l_i \leq x \leq r_i$.



You are responsible for collecting signatures from all tenants in a 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. For simplicity, we assume that when you enter the building, you instantly collect the signatures of all tenants that are in the building at that time.

Try our Touch All Segments interactive puzzle before solving this programming challenge!

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

Output format. The minimum number k of points on the first line and the integer coordinates of k points (separated by spaces) on the second line. You can output the points in any order. If there are multiple such sets of points, you can output any of them.

Constraints. $1 \leq n \leq 100$; $0 \leq l_i \leq r_i \leq 10^9$ for all i.

Sample 1.

Input:

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

Output:

```
1
3
```

All three segments $[1,3]$, $[2,5]$, $[3,6]$ contain the point with coordinate 3.

Sample 2.

Input:

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

Output:

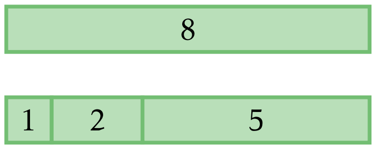
```
2
3 6
```

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 segments cannot be covered by a single point, since the segments $[1,3]$ and $[5,6]$ do not overlap. Another valid solution in this case is the set of points 2 and 5.

3.2.6 Maximum Number of Prizes

Distinct Summands Problem
Represent a positive integer as the sum of the maximum number of pairwise distinct positive integers.

> **Input:** A positive integer n.
> **Output:** The maximum k such that n can be represented as the sum $a_1 + \cdots + a_k$ of k distinct positive integers.



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

Try our Balls in Boxes interactive puzzle before solving this programming challenge!

Input format. An integer n.

Output format. In the first line, output the maximum number k such that n can be represented as the 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 multiple such representations, output any of them).

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

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
```

3.2.7 Maximum Salary

Largest Concatenate Problem
Compile the largest number by concatenating the given numbers.

> **Input:** A sequence of positive integers.
> **Output:** The largest number that can be obtained by concatenating the given integers in some order.

This is probably the most important problem in this book :). As the last question on an interview, your future boss gives you a few pieces of paper with a single number written on each of them and asks you to compose a largest number from these numbers. The resulting number is going to be your salary, so you are very motivated to solve this problem!

Try our Largest Concatenate interactive puzzle before solving this programming challenge!

Recall the algorithm for this problem that works for single digit numbers.

```
LargestConcatenate(Numbers):
yourSalary ← empty string
while Numbers is not empty:
    maxNumber ← $-\infty$
    for each number in Numbers:
        if number ≥ maxNumber:
            maxNumber ← number
    append maxNumber to yourSalary
    remove maxNumber from Numbers
return yourSalary
```

As we know already, this algorithm does not always maximize your salary: for example, for an input consisting of two integers 23 and 3 it returns 233, while the largest number is 323.

Not to worry, all you need to do to maximize your salary is to replace the line

```
    if $number \geq maxNumber$:
```

with the following line:

```
    if IsBetter($number, maxNumber$):
```

for an appropriately implemented function IsBetter. For example, IsBetter(3, 23) should return True.

Stop and Think. How would you implement IsBetter?

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

Output format. The largest number that can be composed out of $a_1, \ldots, a_n$.

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

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
```

The input consists of single-digit numbers only, so the algorithm above returns the correct answer.

Sample 3.

Input:

```
3
23 39 92
```

Output:

```
923923
```

The (incorrect) LargestNumber algorithm nevertheless produces the correct answer in this case, another reminder to always prove the correctness of your greedy algorithms!