CyberChallenge.IT 2017 - Programming
Commented solutions

Contents

1 Problem "Wimpy's Diet" 2
1.1 Problem Statement . . . 2
1.2 Problem Details . . . 2
1.3 Solution . . . 2
1.4 Source Code . . . 3

1 Problem "Wimpy's Diet"

1.1 Problem Statement

Wimpy is always hungry, but his doctor put him on a diet: he's only allowed to eat sandwiches in a strictly decreasing order of weight at any given meal.
His favorite restaurant, however, only serves sandwiches in a fixed order, so Wimpy has to decide which ones to pick.
You are to help him: write a program that removes sandwiches from a menu of N sandwiches so that the remaining sandwiches have the maximum possible total weight and are served in a strictly decreasing order of weight.
Example:
Given a menu of 8 sandwiches:
389 207 155 300 299 170 158 65

We can remove the sandwiches 207 and 155 to obtain a decreasing sequence:
389 300 299 170 158 65
Of maximum total weight 1381.

1.2 Problem Details

Input

The input consists of two lines:

- Line 1: The integer N, the number of sandwiches on a menu.
- Line 2: N integers space separated, the weight of the sandwiches.

Output

The output must contain two lines:

- Line 1: The number of sandwiches in the solution.
- Line 2: The weights of the remaining sandwiches of the menu.

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all testcases of which it is formed.

- **Subtask 1** [**40 points**]: $N \leq 1.000$.
- **Subtask 2** [**60 points**]: $N \leq 10.000$.

1.3 Solution

In order to make things easier, let's consider a small variation of this problem: the rules are the same, except that Wimpy is forced to always keep the first sandwich. Let's call V the vector containing all the weights of the sandwiches. The idea is to solve the problem for suffixes of this array (starting with the one with only 1 element), using the results on shorter suffixes to solve the problem for longer ones, in a sort of bottom-up dynamic programming approach. At the end we will have that one of the solutions for the suffixes will be optimal also for the original problem (since we must keep at least one sandwich).

Let W be an array of the same length of V, N, where in W_i we will store the best sum of weights of the sandwiches to keep, considering only the ones from index i to the end. Let's start with the problem using only one sandwich (the last): the best solution is clearly to keep it, so we will have $W_{N-1} = V_{N-1}$. Let's move to the second to last: if $V_{N-2} > V_{N-1}$, then we keep both of the sandwiches and $W_{N-2} = V_{N-2} + V_{N-1}$, otherwise we are forced to keep only V_{N-2} (because in our modified problem Wimpy must keep the first sandwich), discarding the other. Let's proceed with the problem on 3 elements: we keep V_{N-3} (so $W_{N-3} = V_{N-3}$), then:

- if $V_{N-2} < V_{N-3}$ we can add all the "queue" of sandwiches from the solution with 2 elements (that is, either V_{N-2} or V_{N-2} and V_{N-1}), setting $W_{N-3} = V_{N-3} + W_{N-2}$

- if $V_{N-1} < V_{N-3}$ and $V_{N-3} + W_{N-1} > W_{N-3}$, we have a better solution, so we set $W_{N-3} = V_{N-3} + W_{N-1}$.

The strategy is generalized pretty quickly: for the problem with the last k sandwiches, we initially set $W_{N-k} = V_{N-k}$, then for j from 1 to k we check if $V_{N-k} > V_{N-k+j}$ and, in case of positive response, we set W_{N-k} to the maximum between itself and $V_{N-k} + W_{N-k+j}$. Once we finish building the array W we look for its maximum, whose index m means that V_m will be the first sandwich to remain in the menu. In order to know the subsequent ones we can, for example, have a third array P of length N that stores, for each sandwich, the index of the next sandwich in the optimal solution that starts from it and -1 if it is the last sandwich in that optimal solution. All of this can be implemented in $O(N^2)$ and is enough to get the full score.

1.4 Source Code

C++

```cpp
#include <bits/stdc++.h>

int main()
{
  int N;
  std::cin >> N;
  std::vector<int> V(N);
  std::vector<int> W(N, 0);
  std::vector<int> P(N, -1);

  for(int i=0; i<N; i++) std::cin >> V[i];

  for(int i=N-1; i>=0; i--)
  {
    W[i] = V[i];
    for(int j=i+1; j<N; j++)
    {
      if(V[j] > V[i]) continue;
      if(W[j]+V[i] > W[i])
      {
          W[i] = W[j]+V[i];
          P[i] = j;
      }
    }
  }

  int idx = 0;
  for(int i=0; i<N; i++)
    if(W[i] > W[idx]) idx = i;

  std::vector<int> out;
  while(idx != -1)
  {
    out.push_back(V[idx]);
    idx = P[idx];
  }

  std::cout << out.size() << std::endl;
  for(int v : out) std::cout << v << " ";
  std::cout << std::endl;

  return 0;
}
```

Python

```python
#!/bin/env python3


N = int(input())
V = list(map(int, input().split(" ")))
W = [0 for _ in range(N)]
P = [-1 for _ in range(N)]

for i in range(N-1, -1, -1):
  W[i] = V[i]
  for j in range(i+1, N):
    if V[j] > V[i]:
      continue
    if W[j]+V[i] > W[i]:
       W[i] = W[j]+V[i]
       P[i] = j

idx = 0
for i in range(N):
  if W[i] > W[idx]:
    idx = i

out = []
while idx != -1:
  out.append(str(V[idx]))
  idx = P[idx]

print(len(out))
print(" ".join(out))
```