INOI Select Problems

Nebhrajani A.V.

February 15, 2021

Contents

1: INOI 2012

1.1 Triathlon

1.1.1 Question Statement

The Republic of Tutaria is celebrating its 37th year of independence. To mark the occasion, the nation is organising a contest where all its N citizens take part. The event has three tracks, a COBOL programming competition, pole vault, and a doughnut-eating competition. Each citizen takes part in these three tracks in the same order—a citizen starts with the programming competition, continues with the pole vault as soon as his or her COBOL masterpiece is ready, and then starts gorging on doughnuts as soon as the pole vault is done.

The Supreme Leader of Tutaria closely monitors all citizens and knows the exact amount of time each citizen will take in each of the three tracks. She wants to schedule the event so that it will finish as early as possible. However, the Republic of Tutaria has only one computer, and, as a result, only one person can participate in the COBOL programming event at a time. However, any number of people may simultaneously participate in the pole vault and doughnut-eating competitions.

The event works as follows. The Supreme Leader fixes the order in which contestants get access to the computer. At time 0, the first citizen in the list starts writing his or her COBOL program, while the remaining citizens wait for the computer to be free. As soon as the first citizen is done, he or she proceeds to the pole vault, and the second citizen gets the computer for the programming round. In general whenever the computer becomes free, the next citizen gets to use it. Whenever a citizen is done using the computer, he or she proceeds to the pole vault immediately, regardless of what the other citizens are doing. Similarly, whenever a citizen is done with the pole vault, he or she proceeds to the doughnut- eating track immediately, independently of the others. The event ends as soon as all the citizens have finished all the three tracks of the event.

For example, suppose $N = 3$, and the time they need for the three tracks are as follows:

Citizen ID	COBOL	Pole vault	Doughnut-eating
1	18	7	6
2	23	10	27
3	20	9	14

If the citizens start at time 0 and proceed in the order $1, 2, 3$, then citizen 1 will finish at time $18 + 7 + 6 = 31$, citizen 2 will finish at time $18 + 23 + 10 + 27 = 78$, and citizen 3 will finish at time $18 + 23 + 20 + 9 + 14 = 84$. The event ends at time $\max(31, 78, 84) = 84$.

On the other hand, if the citizens proceed in the order $2, 3, 1$, you can check that the event ends at $\max(60, 66, 74) = 74$. The Supreme Leader of Tutaria wants to fix the order in which the citizens proceed so that the event ends as early as possible. You can check that in this case 74 is the earliest time at which the event can end.

The first line of input has a single integer, N, the number of citizens of the Republic of Tutaria. The next N lines contain 3 space-separated integers each: line i gives the time taken by the citizen i for COBOL programming, pole vault, and doughnut-eating respectively.

The output should have a single line with a single integer, the earliest time at which the event can end.

Here is the sample input and output corresponding to the example above.

```
3
18 7 6
```

23 10 27
20 9 14

Sample output

74

1.1.2 Solution

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

typedef struct citizen
{
  int id;
  int sum;
} citizen;

bool comparator(citizen a, citizen b)
{
  return a.sum > b.sum;
}

int main(void)
{
  int n;
  std::cin >> n;

  std::vector<citizen> cvec(n);

  for (int i = 0; i < n; ++i) {
    int a, b, c;
    std::cin >> a >> b >> c;
    cvec[i] = {a, b+c};
  }

  std::sort(cvec.begin(), cvec.end(), comparator);

  int max = std::numeric_limits<int>::min();
  int total = 0;
  for (int i = 0; i < n; ++i) {
    total += cvec[i].id;
    if (max < total+cvec[i].sum) {
      max = total+cvec[i].sum;
    }
  }

  std::cout << max << "\n";

  return 0;
}
```

2: INOI 2014

2.1 Calvin's Game

2.1.1 Question Statement

Calvin wakes up early one morning and finds that all his friends in the hostel are asleep. To amuse himself, he decides to play the following game : he draws a sequence of N squares on the ground, numbered 1 to N, and writes an integer in each square. He starts at square k ($1 \leq k \leq N$). The game consists of one forward phase followed by one backward phase.

In the forward phase, Calvin makes zero or more moves of the following type : if his current position is p, he can jump to $p+1$ or $p+2$ as long as he stays within the N squares.

In the backward phase, Calvin makes zero or more moves of the following type : if his current position is p, he can jump to $p-1$ or $p-2$ as long as he stays within the N squares.

He plays such that he finally ends up at square 1, and then he stops. He starts with a score of 0, and each time he jumps from square i to square j, he adds the integer written in square j to his score. Find the maximum score Calvin can obtain by playing this game. Recall that Calvin must start at square k and end at square 1. The integer on the square where he starts is not included in his score.

For example, suppose $N = 5$ and the numbers in squares are $5, 3, 2, 1, 1$. If $k = 2$, Calvin starts on the second square. He can make a forward move to square 4, another to square 5, a backward move to square 4, another to square 2, and another to square 1. His total score is $1 + 1 + 1 + 3 + 5 = 11$. You can check that this is the maximum score possible.

Input format:

Line 1 : Two space-separated integers, N and k, with $1 \leq k \leq N$. Line 2 : A space-separated sequence of N integers, the numbers in squares $1, 2..., N$.

Output format:

A single line with a single integer, the maximum score Calvin can obtain by playing the game.

Sample input

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

Sample output

```
11
```

2.1.2 Solution

It is clear that this is a dynamic programming problem, and has optimal recursive substructure. Let us write a function, f which returns the *maximum* chain value ending at that index (inclusive).

$$f(i) = a[i] + f(i-1) + f(i-2)$$

Now, this function is applicable to both the forward and backward direction array sums (without defining a second function $b()$ for the backward direction), albeit with a different base case. The base cases for the forward direction are:

$$f_f(i) = \begin{cases} 0 & i <= k \\ a[i] + f_f(i-1) + f_f(i-2) & i > k \end{cases}$$

For the backward direction:

$$f_b(i) = \begin{cases} 0 & i = 0 \\ a[1] & i = 1 \\ a[i] + f_b(i-1) + f_b(i-2) & i > 1 \end{cases}$$

Let us reason this out. The forward direction is obvious: since we start at k, the value of $f(i)\ \forall\ i <= k = 0$. $\forall i > k$, we follow the recursive function.

The backward case is a bit more interesting. First, it is important to understand that we cannot simply define $b() = a[i] + b(i+1) + b(i+2)$ (the inverse of f_f). This is because, from the question statement, there is a forward stage and a backward stage, with a common element at the end of the forward stage and start of the backward stage. If we use function $b()$, we lose the information of where such a sequence began

Therefore, the function f_g actually needs to tell us what the maximum score can be if we actually *turn back* from that point. This makes this operation identical to f_f with the exception of base cases: $f_b(0) = 0$, $f_b(1) = a[1]$, $f_b(2) = a[1] + a[2]$. The base cases follow if you think about what's the highest score you can get from index 2.

We've therefore logically verified both functions f_f and f_b. Now, how do we combine them to get the required result?

The 'turn back point' can only be an index after k. For each index in $[k+1, N]$, we add $f_f(i) + f_b(i)$, then subtract the double counted $a[i]$. Of all these, we select the maximum demanded by the question.

This approach is shown below in C++.

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

int main(void)
{
  int n, k;
  std::cin >> n;
  std::cin >> k;

  std::vector<int> forw (n+1);
  std::vector<int> backw (n+1);
  std::vector<int> arr (n+1);

  for (int i = 1; i <= n; ++i) {
    std::cin >> arr[i];
  }

  for (int i = k+1; i <= n; ++i) {
    forw[i] = arr[i] + std::max(forw[i-1], forw[i-2]);
  }

  backw[1] = arr[1];
  backw[2] = arr[1] + arr[2];
```

```
23    for (int i = 3; i <= n; ++i) {
24      backw[i] = arr[i] + std::max(backw[i-1], backw[i-2]);
25    }
26
27    int ans = backw[k] - arr[k];
28
29    for (int i = k+1; i <= n; ++i) {
30      ans = std::max(ans, forw[i] + backw[i] - arr[i]);
31    }
32
33    std::cout << ans << "\n";
34
35    return 0;
36  }
```