CyberChallenge.IT 2020 - Programming
Commented solutions

Contents

1 Problem "Postfix"

1.1 Problem Statement

A postfix expression is an expression in which operators follow their operands, for example the following postfix expression *5 7 +* is equivalent to the standard expression *5 + 7*.
The main advantage of a postfix expression is that there is no need for brackets: *2 3 + 2 5 + ** is equivalent to *(2 + 3) * (2 + 5)*.
Another great advantage is that it is more easy to evaluate than infix expressions.
Using a stack we can evaluate the expression from left to right. Each time we scan a number we push it into the stack, each time we scan an operands we pop two elements from the top of the stack, evaluate the operation between them and push the result into the stack again. At the end of the scan, the result is the only element in the stack.
Write a program that evaluates a given postfix expression.

1.2 Problem Details

Input

Your program must read the input data from the standard input.
The first line of the input contains the integer **N**.
The second line contains **N** space-separated strings, which can be positive numbers or operands.

Output

Your program must write the output data into the standard output.
The output must contain only one integer, representing the integer result of the postfix expression evaluation.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- **Subtask 1** [**20 points**]: There are only + operands.
- **Subtask 2** [**20 points**]: All operands are after numbers.
- **Subtask 3** [**60 points**]: All possible operands are present.
- $3 \leq N \leq 99$ for all the test cases.
- All the expressions are well-formed for all the test cases.
- All the expressions contain only $+$, $-$ or $*$ operands for all the test cases.
- All the numbers, intermediate and final results fit in 31-bits integers for all the test cases.

1.3 Solution

This is an implementation task. It requires to implement the well-known **postfix notation**, also known as **Reverse Polish notation**. No clever idea is necessary to solve it, as the general solution is already described, but it requires a good familiarity with a programming language and at least a basic knowledge of the stack data structure. For all the three subcases, the time complexity is $O(N)$.

Subcase 1 (20 points)

Since only + operands are present, the simplest solution here is to add all the numbers provided, as the result would be the same as following the order prescribed by the notation, whatever it is.

Subcase 2 (20 points)

In this subcase, it is possible to store a partial result and traverse the operands in order, since all the operations to be performed are of the type $n \oplus p$, where n is one of the provided numbers, $\oplus$ is one of the provided operands and p is the partial result.

Subcase 4 (60 points)

The last subcase requires to implement the solution as described in the presentation of the problem: traverse the given elements in left to right order, whenever a number is found, it is pushed into the stack and whenever an operand is found, it is applied to the first two numbers of the stack starting from its top, which are replaced by the result of the operation.

1.4 Source Code

C++

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

int solve(int N, std::vector<std::string>& V) {
  std::stack<int> S;
  for(std::string& v : V) {
    if(v == "+" || v == "-" || v == "*") {
      int op1 = S.top();
      S.pop();
      int op2 = S.top();
      S.pop();
      int result = 0;
      if(v == "+") result = op2+op1;
      else if(v == "-") result = op2-op1;
      else if(v == "*") result = op2*op1;
      S.push(result);
    } else {
      S.push(std::stoi(v));
    }
  }
  return S.top();
}

int main() {
  int N;

  std::cin >> N;
  std::vector<std::string> V(N);

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

  std::cout << solve(N, V) << std::endl;
  return 0;
}
```

Python

```python
#!/usr/bin/python3

def solve(N, V):
  stack = []
  for v in V:
    if v in ['+', '-', '*']:
      op1 = stack.pop()
      op2 = stack.pop()
      res = eval(op2 + v + op1)
      assert((-(2**31)) <= res <= (2**31))
```

```
            stack.append(str(res))
        else:
            stack.append(v)
    return stack[0]

if __name__ == "__main__":
    N = int(input())
    V = input().strip().split(" ")
    assert(len(V) == N)
    print(solve(N, V))
```

2 Problem "Maxofmin"

2.1 Problem Statement

Given in input a vector V of N integers we want to find, for each size between 1 and N, the maximum of the minimum's of every contiguous subsequence in the vector.
Example
For $N = 6$ and $V[6] = [3, 1, 4, 6, 2, 9]$ we have this contiguous subsequences:

- Size 1: [**3**], [**1**], [**4**], [**6**], [**2**] and [**9**].
- Size 2: [3, **1**], [**1**, 4], [**4**, 6], [6, **2**] and [**2**, 9].
- Size 3: [3, **1**, 4], [**1**, 4, 6], [4, 6, **2**] and [6, **2**, 9].
- Size 4: [3, **1**, 4, 6], [**1**, 4, 6, 2] and [4, 6, **2**, 9].
- Size 5: [3, **1**, 4, 6, 2] and [**1**, 4, 6, 2, 9].
- Size 6: [3, **1**, 4, 6, 2, 9].

Where the minimum of each subsequence is bolded.
For each size then the maximum of the minimum's of every contiguous subsequence in the vector is:
9 (for size 1), **4** (for size 2), **2** (for size 3), **2** (for size 4), **1** (for size 5) and **1** (for size 6).

2.2 Problem Details

Input

Your program must read the input data from the standard input.
The first line of the input contains the integer **N** representing the size of the vector **V**.
The second line contains **N** space-separated integers representing the elements of **V**.

Output

Your program must write the outut data into the standard output.
The output must contain **N** space-separated integers: the maximum of the minimum's of every contiguous subsequence in the array, for each size between **1** and **N**.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- **Subtask 1** [**20 points**]: $N \leq 100$.
- **Subtask 2** [**20 points**]: $N \leq 1\,000$.
- **Subtask 3** [**20 points**]: $N \leq 10\,000$.
- **Subtask 4** [**40 points**]: $N \leq 100\,000$.
- $1 \leq V[i] \leq 1\,000\,000$ for each $0 \leq i < N$ for all the test cases.

2.3 Solution

The first subcases do not require particularly complex ideas, while the last one requires a more optimized solution to be solved. Data structures more complex than stacks should not be necessary.

Subcase 1 and 2 (20+20 points)

The simplest idea to solve this problem is to iterate over i, for i between 1 and N, to generate all the subsequences of length i and compute the minimum of each subsequence and the maximum of the minima. This solution has a time complexity of $O(N^3)$ and is enough to pass the first two subcases with a C++ implementation.

Subcase 3 (20 points)

The first improvement that can be done is to iterate over the starting element of the subsequence, and then, for each length from 1 to N, compute the minimum element of the subsequence and, at the same time, update the maximum of the mimima for that length. The time complexity thus decreases to $O(N^2)$.

Subcase 4 (40 points)

The general solution is more complex; the main steps are:

- for each element, compute the arrays of the indexes of the elements smaller than the current one on the left and on the right. If no smaller element is found, than the indexes are -1 for the left indexes and N (the size of the array) for the right indexes. This can be done in $O(N)$ using the algorithm for the Next Greater Element problem.
- for each index i, the element $V[i]$ is the minimum of the subsequence bounded by the minimum elements on the left and on the right, i.e. it is the minimum of a subsequence of length $R[i] - L[i] - 1$, if R and L are the arrays computed in the previous step. These values are obviously candidate values for the final answer. Also this step has time complexity of $O(N)$.
- Let us notice that the answer should be a non-increasing sequence; since the previous step can leave some lengths without a maximum (or it can leave some lengths with a non optimal value), the last step is to set the answer for each length as the maximum between the current, temporary, answer for that length and the answer for the next length (i.e. $\text{ans}[i] = \max(\text{ans}[i], \text{ans}[i+1])$, if ans is the array of the answers).

Each of the three steps has complexity $O(N)$, thus the global complexity is still $O(N)$.

2.4 Source Code

C++

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

void solve(int N, std::vector<int>& V){
  std::vector<int> s;
  std::vector<int> left(N+1, -1);
  std::vector<int> right(N+1, N);

  for(int i=0; i<N; i++) {
    while(s.size() > 0 && V[s.back()] > V[i])
      s.pop_back();

    if(s.size() != 0) left[i] = s.back();

    s.push_back(i);
  }

  s = std::vector<int>();

  for(int i=N-1; i>=0; i--) {
    while(s.size() > 0 && V[s.back()] >= V[i])
      s.pop_back();

    if(s.size() != 0) right[i] = s.back();

    s.push_back(i);
  }

  std::vector<int> ans(N+1, 0);

  for(int i=0; i<N; i++){
    int len = right[i] - left[i] - 1;
```

```cpp
      ans[len] = std::max(ans[len], V[i]);
    }

    for(int i=N-1; i>0; i--) {
      ans[i] = std::max(ans[i], ans[i+1]);
    }

    for(int i=1; i<=N; i++)
      std::cout << ans[i] << " ";
}

int main(){
    int N;
    std::cin >> N;
    std::vector<int> V(N);
    for(int i=0; i<N; i++) std::cin >> V[i];
    solve(N, V);
    return 0;
}
```

Python

```python
#!/usr/bin/python3

def solve(n, arr):
  s = []
  left = [-1]*(n+1)
  right = [n]*(n+1)

  for i in range(n):
    while (len(s) != 0 and arr[s[-1]] >= arr[i]):
      s.pop()

    if (len(s) != 0):
      left[i] = s[-1]

    s.append(i)

  s = []

  for i in range(n-1, -1, -1):
    while (len(s) != 0 and arr[s[-1]] >= arr[i]):
      s.pop()

    if(len(s) != 0):
      right[i] = s[-1]

    s.append(i)

  ans = [0]*(n+1)

  for i in range(n):
    Len = right[i] - left[i] - 1
    ans[Len] = max(ans[Len], arr[i])

  for i in range(n - 1, 0, -1):
    ans[i] = max(ans[i], ans[i + 1])

  return ans[1:]
```

```python

if __name__ == "__main__":
  N = int(input().strip())
  V = list(map(int, input().strip().split(" ")))
  print(" ".join(map(str, solve(N, V))))
```

3 Problem "Polynomials"

3.1 Problem Statement

Given two numbers n and k we consider a polynomial *valid* if its degree is n and its coefficients are all integers not exceeding k by the absolute values.
More formally, denote the coefficients with $a_0\ a_1\ \dots\ a_{n-1}\ a_n$.
Then the polynomial $P(x) = \sum_{i=0}^{n} a_i \cdot x^i = a_0 + a_1 \cdot x + \ldots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$ is valid if:

- a_i is integer for every i.
- $|a_i| \leq k$ for every i.
- $a_n \neq 0$.

Given a valid polynomial $P(X)$, such that $P(2) \neq 0$, we want to count in how many ways we can change only one coefficient to get a *valid* polynomial $Q(x)$ of degree n such that $Q(2) = 0$.
Example
Given $n = 3$ and $k = 12$ and the polynomial $P(x) = 10 - 9x - 3x^2 + 5x^3$.
Where $P(2) = 10 - 18 - 12 + 40 = 20 \neq 0$.
We can change one coefficient of $P(X)$ only in two different ways:

- $a_0 = -10$, then $Q(x) = -10 - 9x - 3x^2 + 5^3$ and $Q(2) = 0$
- $a_2 = -8$, then $Q(x) = 10 - 9x - 8x^2 + 5x^3$ and $Q(2) = 0$

Thus the solution is 2.

3.2 Problem Details

Input

Your program must read the input data from the standard input.
The first line of the input contains two space-separated integers **n** and **k** representing the degree of the polynomial and the limit for absolute value of coefficients, respectively.
The second line contains **n+1** space-separated integers, representing the coefficients $a_0\ a_1\ \dots\ a_{n-1}\ a_n$ of the polynomial.

Output

Your program must write the output data into the standard output.
The output should consist of only one line, containing the number of ways to change one coefficient to get a *valid* polynomial $Q(X)$ with $Q(2) = 0$.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- **Subtask 1** [**40 points**]: $n \leq 100$ and $k \leq 100$.
- **Subtask 2** [**60 points**]: $n \leq 100$ and $k \leq 10\,000$.
- $1 \leq n \leq 100$.
- $1 \leq k \leq 10\,000$.
- The given polynomial is always valid and $P(2) \neq 0$ for all the test cases.

3.3 Solution

This is the hardest problem in the problem set. It requires a mix of clever ideas and implementation skills to be fully solved.

Subcase 1 (40 points)

For this subcase a simple bruteforce is enough to try all possible values between $-k$ and k, giving a time complexity of $O(NK)$.

Subcase 2 (60 points)

This time a more clever approach is necessary. Let us start by writing

$$p(2) = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2 + a_0$$

our goal is to obtain a polinomial p' such that

$$p'(2) = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a'_i 2^i + \ldots + a_1 2 + a_0 = 0$$

This means that $p(2) - p'(2) = p(2)$, i.e. $a_i 2^i - a'_i 2^i = p(2)$, $a'_i = -\frac{p(2) - a_i 2^i}{2^i}$. It is sufficient to check if this value is in the allowed range, i.e. if $|a'_i| \leq k$.
However, if we implement the solution in a C-like language, there could be problems when representing integers. The adopted solution is to analyze the situation modulo some enough large primes; if with all of them the found solution is the same, then it is reasonable to assume that the solution is valid without the modulo operation.
The solution has to iterate over the coefficients, thus the complexity is $O(N)$.

3.4 Source Code

C++

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

using namespace std;

vector<long long int> primes = {1000000007, 1000000009, 1000000021, 1000000033, 1000000087};

long long int egcd(long long int a, long long int b, long long int *x, long long int *y){
    if (a == 0){
        *x = 0;
        *y = 1;
        return b;
    }

    long long int x1, y1;
    long long int g = egcd((b%a + a)%a, a, &x1, &y1);

    *x = y1 - (b/a) * x1;
    *y = x1;
    return g;
}

long long int inverse(long long int a, long long int m){
    long long int x, y;
    long long int g = egcd(a, m, &x, &y);
    if (g != 1)
        throw runtime_error("modular inverse does not exist");
    else {
        long long int res = (x % m + m) % m;
        return res;
    }
}

long long int power(long long x, unsigned long long y, long long  p){
    long long int res = 1;
    x = (x % p + p) % p;

    if (x == 0) return 0;

    while (y > 0){
        if (y & 1)
```

```cpp
            res = (res*x) % p;
        y = y>>1;
        x = (x*x) % p;
    }
    return (res % p + p) % p;
}

long long int poly_eval_mod(vector<int> coefs, long long int x, long long int p){
    long long int res = 0;
    for(int i=0; i<coefs.size(); i++){
        res = ((res + (coefs[i]*power(x, i, p)))%p + p)%p;
    }
    return res;
}

long long int solve(long long int N, long long int K, std::vector<int>& V) {
  int solution = 0;
  for(int i=0; i<N; i++){
    set<long long int> pos_sols, neg_sols;
    for(auto p: primes){
        long long int s = (((((poly_eval_mod(V, 2, p) - V[i]*power(2, i, p))%p + p) % p) *
                    inverse(power(2, i, p), p))%p + p) % p;
        pos_sols.insert(s);
    }
    if(pos_sols.size()==1 && *(pos_sols.begin())<=K && !(i==N && *(pos_sols.begin()) == 0)){
        solution++;
    }
    else if(neg_sols.size()==1 && *(neg_sols.begin())<=K && !(i==N && *(neg_sols.begin()) == 0)){
        solution++;
    }
  }
  return solution;
}

int main() {
  int N, K;
  std::cin >> N >> K;
  std::vector<int> V(N+1);
  for(int i=0; i<N+1; i++) std::cin >> V[i];
  std::cout << solve(N, K, V) << std::endl;
  return 0;
}
```

Python

```python
#!/usr/bin/python3

def solve(n, k, coefs):
  sol = 0
  n = len(coefs)-1
  p2 = sum(coefs[j]*2**j for j in range(len(coefs)))
  for i in range(len(coefs)):
    partial_results = []
    tmp_sol = (p2 - coefs[i]*2**i)
    if tmp_sol % (2**i) == 0:
      tmp_sol = tmp_sol // (2**i)
      if abs(tmp_sol) <= k and not(i == n and tmp_sol == 0):
        sol = sol + 1
  return sol
```

```python

if __name__ == "__main__":
  n, k = map(int, input().strip().split(" "))
  coefs = list(map(int, input().strip().split(" ")))
  print(solve(n, k, coefs))
```