CS Problem Statement 2 - Trading Model - Submit by 23:59 8OCt17

locked

by torquecode

Problem Submissions Leaderboard Discussions

Submissions will no longer be placed on the leaderboard. You may still attempt this problem for practice.

For use by intended recipient only and not for further distribution

A trader wants to design a test to test the optimality of his/her algorithmic trading model. The trading model functions as follows.

There are total N number of stocks in a stock exchange up for trading, numbered from 1 to N. Each stock i, based on its previous performance has a strength score $S[i]$.

Moreover, there is a **similarity relation** defined between stocks. Total of $N-1$ unique unordered pairs of stocks are in the similarity relation. The stock exchange has put up a restriction that if a trader buys a stock i, he/she can only buy stocks related to i after that. Formally, we have graph G, where stocks are vertices, similarity relation defines edges in G, and G is connected.

In the test, to emulate the real stock exchange where multiple traders compete against each other to buy stocks, there are two opponent trading models A and B. The aim of the trading model is to buy stocks such that in the end, the sum of strength of the stocks bought by it are the highest. The model with highest strength is the winner. **In a case where both models end up buying the stocks summing up to same strength, model A is assumed to be the winner.**

The process of buying stocks is the following:

1. A has stock u_i initially, while B has stock v_i.
2. A and B buy stocks in turns. In one turn, first, A can buy any stock which has not yet been bought and has a similarity relation with at least one stock which he/she has already bought. Next, B can buy any stock who has not yet been bought and has a similarity relation with at least one stock he/she has already bought. If a trading model cannot buy any stock, then the next model proceeds. The model has to buy a stock if it can buy one.
3. The trading ends when in a turn, none of the models can buy any stock.

You have to design a program which will give the trader expected values for each test case in the test i.e. the optimal result. The program will process Q queries. Each query contains two numbers u and v denoting that A has stock u initially, while B has stock v initially.

After processing all queries, your task is to report 4 integers: A, B, C and D, where:

- $A =$ Number of queries in which A won the game
- $B =$ Cumulative strength of A's stocks in all queries
- $C =$ Number of queries in which B won the game
- $D =$ Cumulative strength of B's stocks in all queries

Evaluation Criteria

- Correctness and efficiency. A good solution should be optimized w.r.t both time and memory.
- Code Quality. Good code design and readability is expected.

Input Format

- First line of input contains a single integer N.

- Second line contains N space separated integers $S[1], S[2], \ldots, S[N]$ denoting strenghts of all the stocks
- The i-th of subsequent $N-1$ lines contains two space-separated integers, a_i and b_i, describing a similarity relation between stock a_i and b_i.
- Next line contains Q, the number of queries.
- The i-th of subsequent Q lines contains two space-separated integers, u_i and v_i denoting the initial stock respectively with A and B.

Constraints

- $2 \leq N \leq 10^5$
- $1 \leq Q \leq 2 \cdot 10^5$
- $1 \leq a_i, b_i \leq N$
- $1 \leq u_i, v_i \leq N, u \neq v$
- $1 \leq S[i] \leq 10^6$

Output Format

In a single line, print exactly 4 space separated integers: A, B, C and D.

Sample Input 0

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

Sample Output 0

```
0 7 2 33
```

Sample Input 1

```
5
1 2 3 4 5
1 2
1 3
1 4
4 5
2
2 5
4 2
```

Sample Output 1

```
1 19 1 11
```

Submissions: 194
Max Score: 500
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++

```
#include <cmath>
#include <cstdio>
#include <vector>
```

```cpp
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code