1 Problem: Integrated Circuit Design

Problem Introduction

In this problem, you will determine how to connect the modules of an integrated circuit with wires so that all the wires can be routed on the same layer of the circuit.

Problem Description

Task. VLSI or Very Large-Scale Integration is a process of creating an integrated circuit by combining thousands of transistors on a single chip. You want to design a single layer of an integrated circuit. You know exactly what modules will be used in this layer, and which of them should be connected by wires. The wires will be all on the same layer, but they cannot intersect with each other. Also, each wire can only be bent once, in one of two directions — to the left or to the right. If you connect two modules with a wire, selecting the direction of bending uniquely defines the position of the wire. Of course, some positions of some pairs of wires lead to intersection of the wires, which is forbidden. You need to determine a position for each wire in such a way that no wires intersect.

This problem can be reduced to 2-SAT problem — a special case of the SAT problem in which each clause contains exactly 2 variables. For each wire i, denote by x_i a binary variable which takes value 1 if the wire is bent to the right and 0 if the wire is bent to the left. For each i, x_i must be either 0 or 1. Also, some pairs of wires intersect in some positions. For example, it could be so that if wire 1 is bent to the left and wire 2 is bent to the right, then they intersect. We want to write down a formula which is satisfied only if no wires intersect. In this case, we will add the clause $(x_1 \ OR \ \overline{x_2})$ to the formula which ensures that either x_1 (the first wire is bent to the right) is true or $\overline{x_2}$ (the second wire is bent to the left) is true, and so the particular crossing when wire 1 is bent to the left AND wire 2 is bent to the right doesn't happen whenever the formula is satisfied. We will add such a clause for each pair of wires and each pair of their positions if they intersect when put in those positions. Of course, if some pair of wires intersects in any pair of possible positions, we won't be able to design a circuit. Your task is to determine whether it is possible, and if yes, determine the direction of bending for each of the wires.

Input Format. The input represents a 2-CNF formula. The first line contains two integers V and C — the number of variables and the number of clauses respectively. Each of the next C lines contains two non-zero integers i and j representing a clause in the CNF form. If $i > 0$, it represents x_i, otherwise if $i < 0$, it represents $\overline{x_{-i}}$, and the same goes for j. For example, a line "`2 3`" represents a clause $(x_2 \ OR \ x_3)$, line "`1 -4`" represents $(x_1 \ OR \ \overline{x_4})$, line "`-1 -3`" represents $(\overline{x_1} \ OR \ \overline{x_3})$, and line "`0 2`" cannot happen, because i and j must be non-zero.

Constraints. $1 \leq V, C \leq 1\,000\,000$; $-V \leq i, j \leq V$; $i, j \neq 0$.

Output Format. If the 2-CNF formula in the input is unsatisfiable, output just the word "UNSATISFIABLE" (without quotes, using capital letters). If the 2-CNF formula in the input is satisfiable, output the word "SATISFIABLE" (without quotes, using capital letters) on the first line and the corresponding assignment of variables on the second line. For each x_i in order from x_1 to x_V, output i if $x_i = 1$ or $-i$ if $x_i = 0$. For example, if a formula is satisfied by assignment $x_1 = 0, x_2 = 1, x_3 = 0$, output "`-1 2 -3`" on the second line (without quotes). If there are several possible assignments satisfying the input formula, output any one of them.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	18	16	1.5	2	16	16	36

Memory Limit. 512MB.

Sample 1.

Input:

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

Output:

```
SATISFIABLE
1 2 -3
```

The input formula is $(x_1\ OR\ \overline{x_3})\ AND\ (\overline{x_1}\ OR\ x_2)\ AND\ (\overline{x_2}\ OR\ \overline{x_3})$, and the assignment $x_1 = 1, x_2 = 1, x_3 = 0$ satisfies it.

Sample 2.

Input:

```
1 2
1 1
-1 -1
```

Output:

```
UNSATISFIABLE
```

Explanation:
The input formula is $(x_1\ OR\ x_1)\ AND\ (\overline{x_1}\ OR\ \overline{x_1})$, and it is unsatisfiable.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a brute-force procedure that checks all possible variable assignments, and output the result. You need to implement a more efficient algorithm in this procedure if you're using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `circuit_design`

What To Do

You need to implement an algorithm described in the lectures.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

2 Problem: Plan a Fun Party

Problem Introduction

In this problem, you will design and implement an efficient algorithm to plan invite the coolest people from your company to a party in such a way that everybody is relaxed, because the direct boss of any invited person is not invited.

Problem Description

Task. You're planning a company party. You'd like to invite the coolest people, and you've assigned each one of them a fun factor — the more the fun factor, the cooler is the person. You want to maximize the total fun factor (sum of the fun factors of all the invited people). However, you can't invite everyone, because if the direct boss of some invited person is also invited, it will be awkward. Find out what is the maximum possible total fun factor.

Input Format. The first line contains an integer n — the number of people in the company. The next line contains n numbers f_i — the fun factors of each of the n people in the company. Each of the next $n-1$ lines describes the subordination structure. Everyone but for the CEO of the company has exactly one direct boss. There are no cycles: nobody can be a boss of a boss of a ... of a boss of himself. So, the subordination structure is a regular tree. Each of the $n-1$ lines contains two integers u and v, and you know that either u is the boss of v or vice versa (you don't really need to know which one is the boss, but you can invite only one of them or none of them).

Constraints. $1 \leq n \leq 100\,000$; $1 \leq f_i \leq 1\,000$; $1 \leq u, v \leq n$; $u \neq v$.

Output Format. Output the maximum possible total fun factor of the party (the sum of fun factors of all the invited people).

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	1.5	5	1.5	2	5	5	3

Memory Limit. 512MB.

Sample 1.

Input:

```
1
1000
```

Output:

```
1000
```

Explanation:
There is only one person in the company, the CEO, and the fun factor is 1000. We can just invite the CEO and get the total fun factor of 1000.

Sample 2.

Input:

```
2
1 2
1 2
```

Output:

```
2
```

Explanation:

There are two people, and one of them is the boss of another one. We can invite only one of them. If we invite the second one, the total fun factor is 2, and it is bigger than total fun factor of 1 that we get in case we invite the first one.

Sample 3.

Input:

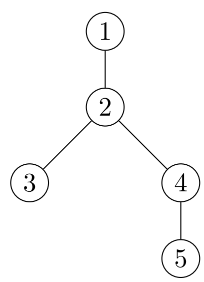
```
5
1 5 3 7 5
5 4
2 3
4 2
1 2
```

Output:

```
11
```

Explanation:

A possible subordination structure:



We can invite 1, 3 and 4 for a total fun factor of 11. If we invite 2, we cannot invite 1, 3 or 4, so the total fun factor will be at most 10, thus we don't invite 2 in the optimal solution. If we don't invite 4 also, we will get a fun factor of at most $1 + 3 + 5 = 9$, so we must invite 4. But then we can't invite 5, so we invite also 1 and 3 and get the total fun factor of 11.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a template procedure that implements depth-first search but doesn't compute anything, and output the result. You need to augment the template procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `plan_party`

What To Do

You need to implement an algorithm described in the lectures.