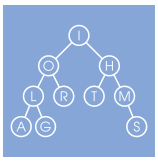


Module: Coping with NP-completeness (Week 4 out of 5)
Course: Advanced Algorithms and Complexity (Course 5 out of 6)
Specialization: Data Structures and Algorithms

# Programming Assignment 4: Coping with NP-completeness

Revision: July 8, 2019

## Introduction

Welcome to your last programming assignment of the Advanced Algorithms and Complexity class! In this programming assignment, you will be practicing solving NP-complete problems. Of course, currently we don't know efficient algorithms for solving these problems, but in some special cases there are efficient algorithms, and for some problems there are algorithms exponentially more efficient than the brute-force approach, although they still have exponential running time themselves.

## Learning Outcomes

Upon completing this programming assignment you will be able to:

1. design a part of integrated circuit;
2. plan a totally cool party;
3. find the optimal route for a school bus;
4. reschedule the exams.

## Passing Criteria: 2 out of 4

Passing this programming assignment requires passing at least 2 out of 4 code problems from this assignment. In turn, passing a code problem requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

# Contents

# 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.** $1 \le V, C \le 1\,000\,000$; $-V \le i, j \le V$; $i, j \ne 0$.

**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.** $1 \le n \le 100\,000$; $1 \le f_i \le 1\,000$; $1 \le u, v \le n$; $u \ne v$.

**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.**

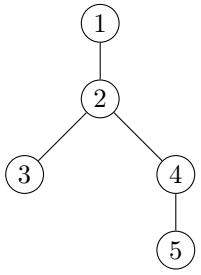
Input:

```
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.

## Need Help?

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

# 3 Problem: School Bus

## Problem Introduction

In this problem, you will determine the fastest route for a school bus to start from the depot, visit all the children's homes, get them to school and return back to depot.


## Problem Description

**Task.** A school bus needs to start from the depot early in the morning, pick up all the children from their homes in some order, get them all to school and return to the depot. You know the time it takes to get from depot to each home, from each home to each other home, from each home to the school and from the school to the depot. You want to define the order in which to visit children's homes so as to minimize the total time spent on the route.

This is an instance of a classical NP-complete problem called Traveling Salesman Problem. Given a graph with weighted edges, you need to find the shortest cycle visiting each vertex exactly once. Vertices correspond to homes, the school and the depot. Edges weights correspond to the time to get from one vertex to another one. Some vertices may not be connected by an edge in the general case.

**Input Format.** The first line contains two integers $n$ and $m$ — the number of vertices and the number of edges in the graph. The vertices are numbered from 1 through $n$. Each of the next $m$ lines contains three integers $u$, $v$ and $t$ representing an edge of the graph. This edge connects vertices $u$ and $v$, and it takes time $t$ to get from $u$ to $v$. The edges are bidirectional: you can go both from $u$ to $v$ and from $v$ to $u$ in time $t$ using this edge. No edge connects a vertex to itself. No two vertices are connected by more than one edge.

**Constraints.** $2 \le n \le 17$; $1 \le m \le \frac{n(n-1)}{2}$; $1 \le u, v \le n$; $u \ne v$; $1 \le t \le 10^9$.

**Output Format.** If it is possible to start in some vertex, visit each other vertex exactly once in some order going by edges of the graph and return to the starting vertex, output two lines. On the first line, output the minimum possible time to go through such circular route visiting all vertices exactly once (apart from the first vertex which is visited twice — in the beginning and in the end). On the second line, output the order in which you should visit the vertices to get the minimum possible time on the route. That is, output the numbers from 1 through $n$ in the order corresponding to visiting the vertices. Don't output the starting vertex second time. However, account for the time to get from the last vertex back to the starting vertex. If there are several solutions, output any one of them. If there is no such circular route, output just $-1$ on a single line. Note that for $n = 2$ it is considered a correct circular route to go from one vertex to another by an edge and then return back by the same edge.

**Time Limits.**

| language | `C` | `C++` | `Java` | `Python` | `C#` | `Haskell` | `JavaScript` | `Ruby` | `Scala` |
|---|---|---|---|---|---|---|---|---|---|
| time (sec) | 1 | 1 | 1.5 | 45 | 1.5 | 2 | 45 | 45 | 3 |

**Memory Limit.** 512MB.

**Sample 1.**

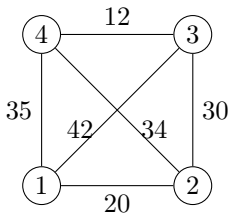
Input:

```
4 6
1 2 20
1 3 42
1 4 35
2 3 30
2 4 34
3 4 12
```

Output:

```
97
1 4 3 2
```

Explanation:




The suggested route starts in the vertex 1, goes to 4 in 35 minutes, from there to 3 in 12 minutes, from there to 2 in 30 minutes, from there back to 1 in 20 minutes, totalling in $35 + 12 + 30 + 20 = 97$ minutes. Check yourself that any other circular route visiting each vertex exactly once is longer.

**Sample 2.**

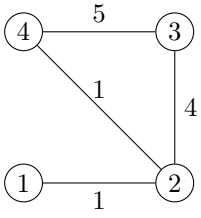
Input:

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

Output:

```
-1
```

Explanation:




There is no way to visit all the vertices exactly once, as there is only one edge from the vertex 1 (going to 2), so after leaving it you cannot return without visiting 2 twice.

## Starter Files

The starter solutions for this problem read the data from the input, pass it to a brute-force procedure that tries each possible order of visit, and output the result. You need to change the brute-force procedure to implement some more efficient algorithm if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `school_bus`

## 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.

# 4 Advanced Problem: Reschedule the Exams

We strongly recommend you start solving advanced problems only when you are done with the basic problems (for some advanced problems, algorithms are not covered in the video lectures and require additional ideas to be solved; for some other advanced problems, algorithms are covered in the lectures, but implementing them is a more challenging task than for other problems).

## Problem Introduction

In this problem, you will design and implement an efficient algorithm to reschedule the exams in such a way that every student can come to the exam she is assigned to, and no two friends will be passing the exam the same day.

## Problem Description

**Task.** The new secretary at your Computer Science Department has prepared a schedule of exams for CS-101: each student was assigned to his own exam date. However, it's a disaster: not only some pairs of students known to be close friends may have been assigned the same date, but also NONE of the students can actually come to the exam at the day they were assigned (there was a misunderstanding between the secretary who asked to specify available dates and the students who understood they needed to select the date at which they cannot come). There are three different dates the professors are available for these exams, and these dates cannot be changed. The only thing that can be changed is the assignment of students to the dates of exams. You know for sure that each student can't come at the currently scheduled date, but also each student definitely can come at any of the two other possible dates. Also, you must make sure that no two known close friends are assigned to the same exam date. You need to determine whether it is possible or not, and if yes, suggest a specific assignment of the students to the dates.

This problem can be reduced to a graph problem called 3-recoloring. You are given a graph, and each vertex is colored in one of the 3 possible colors. You need to assign another color to each vertex in such a way that no two vertices connected by and edge are assigned the same color. Here, possible colors correspond to the possible exam dates, vertices correspond to students, colors of the vertices correspond to the assignment of students to the exam dates, and edges correspond to the pairs of close friends.

**Input Format.** The first line contains two integers $n$ and $m$ — the number of vertices and the number of edges of the graph. The vertices are numbered from 1 through $n$. The next line contains a string of length $n$ consisting only of letters $R$, $G$ and $B$ representing the current color assignments. For each position $i$ (1-based) in the string, if it is $R$, then the vertex $i$ is colored red; if it's $G$, the vertex $i$ is colored green; if it's $B$, the vertex $i$ is colored blue. These are the current color assignments, and **each of them must be changed**. Each of the next $m$ lines contains two integers $u$ and $v$ — vertices $u$ and $v$ are connected by an edge (it is possible that $u = v$).

**Constraints.** $1 \le n \le 1\,000$; $0 \le m \le 20\,000$; $1 \le u, v \le n$.

**Output Format.** If it is impossible to reassign the students to the dates of exams in such a way that no two friends are going to pass the exam the same day, and each student's assigned date has changed, output just one word "Impossible" (without quotes). Otherwise, output one string consisting of $n$ characters $R$, $G$ and $B$ representing the new coloring of the vertices. Note that the color of each vertex must be

different from the initial color of this vertex. The vertices connected by an edge must have different colors.

**Time Limits.**

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

**Memory Limit.** 512MB.

**Sample 1.**

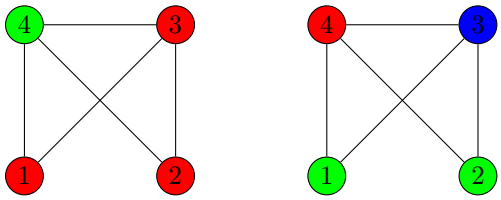
Input:

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

Output:

```
GGBR
```

Explanation:
The initial coloring and the new coloring:




Note that the vertices 1 and 2 are ok to be of the same color, as they are not connected by an edge.

**Sample 2.**

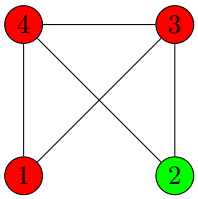
Input:

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

Output:

```
Impossible
```

Explanation:
The initial coloring:




It is impossible to recolor the vertices properly, because it means that none of the vertices 1, 3 and 4 can be red, but that leaves only two choices of colors to them — blue and green — but each pair of them is connected by an edge, so we need at least three different colors to color them properly.

## Starter Files

The starter solutions for this problem read the data from the input, pass it to a procedure that just tries to set the colors of the vertices in a fixed way without looking at the edges or the current colors. You need to change the main procedure to solve the problem correctly if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `reschedule_exams`

## What To Do

You need to reduce this problem to another problem you already know how to solve.

## Need Help?

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