Module:	Flows in Networks (Week 1 out of 4)
Course:	Advanced Algorithms and Complexity (Course 5 out of 6)
Specialization:	Data Structures and Algorithms

Programming Assignment 1: Flows in Networks

Revision: August 25, 2016

Introduction

Welcome to your first programming assignment of the Advanced Algorithms and Complexity class! In this programming assignment, you will be practicing implementing and applying algorithms for finding maximum flows in networks.

In this programming assignment, the grader will show you the input data if your solution fails on any of the tests. This is done to help you to get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail. However, for all the following programming assignments, the grader will show the input data only in case your solution fails on one of the first few tests (please review the questions 5.4 and 5.5 in the FAQ section for a more detailed explanation of this behavior of the grader).

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply Ford-Fulkerson and/or Edmonds-Karp algorithm to solve various computational problems efficiently. This will typically require you to invent a way to reduce the problem to a maximum flow or maximum matching problem and then use an efficient algorithm to solve it.
2. Design and implement efficient algorithms for the following computational problems:
   (a) evacuating people from the city as fast as possible;
   (b) assigning airline crews to the aircrafts efficiently;
   (c) visualizing stock price data compactly.

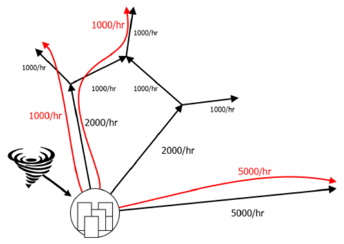
Passing Criteria: 2 out of 3

Passing this programming assignment requires passing at least 2 out of 3 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.

1 Problem: Evacuating People

Problem Introduction

In this problem, you will apply an algorithm for finding maximum flow in a network to determine how fast people can be evacuated from the given city.



Problem Description

Task. A tornado is approaching the city, and we need to evacuate the people quickly. There are several roads outgoing from the city to the nearest cities and other roads going further. The goal is to evacuate everybody from the city to the capital, as it is the only other city which is able to accomodate that many newcomers. We need to evacuate everybody as fast as possible, and your task is to find out what is the maximum number of people that can be evacuated each hour given the capacities of all the roads.

Input Format. The first line of the input contains integers n and m — the number of cities and the number of roads respectively. Each of the next m lines contains three integers u, v and c describing a particular road — start of the road, end of the road and the number of people that can be transported through this road in one hour. u and v are the 1-based indices of the corresponding cities.

The city from which people are evacuating is the city number 1, and the capital city is the city number n.

Note that all the roads are given as one-directional, that is, you cannot transport people from v to u using a road that connects u to v. Also note that there can be several roads connecting the same city u to the same city v, there can be both roads from u to v and from v to u, or there can be only roads in one direction, or there can be no roads between a pair of cities. Also note that there can be roads going from a city u to itself in the input.

When evacuating people, they cannot stop in the middle of the road or in any city other than the capital. The number of people per hour entering any city other than the evacuating city 1 and the capital city n must be equal to the number of people per hour exiting from this city. People who left a city u through some road (u, v, c) are assumed to come immediately after that to the city v. We are interested in the maximum possible number of people per hour leaving the city 1 under the above restrictions.

Constraints. $1 \leq n \leq 100$; $0 \leq m \leq 10\,000$; $1 \leq u, v \leq n$; $1 \leq c \leq 10\,000$. It is guaranteed that $m \cdot \mathit{EvacuatePerHour} \leq 2 \cdot 10^8$, where *EvacuatePerHour* is the maximum number of people that can be evacuated from the city each hour — the number which you need to output.

Output Format. Output a single integer — the maximum number of people that can be evacuated from the city number 1 each hour.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time in seconds	1	1	5	45	1.5	2	45	45	10

Memory Limit. 512Mb.

Sample 1.

Input:

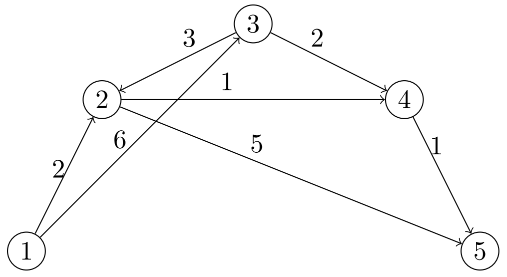
```
5 7
1 2 2
2 5 5
1 3 6
3 4 2
4 5 1
3 2 3
2 4 1
```

Output:

```
6
```

Explanation:

In this sample, the road graph with capacities looks like this:



We can evacuate 2 people through the route $1-2-5$, additional 3 people through the route $1-3-2-5$ and 1 more person through the route $1-3-4-5$ — for a total of 6 people. It is impossible to evacuate more people each hour, as the total capacity of all roads incoming to the capital city 5 is 6 people per hour.

Sample 2.

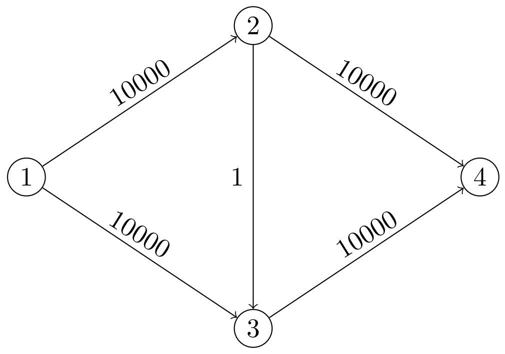
Input:

```
4 5
1 2 10000
1 3 10000
2 3 1
3 4 10000
2 4 10000
```

Output:

```
20000
```

Explanation:
In this sample, the road graph with capacities looks like this:



We can evacuate 10000 people through the route $1-2-4$ and additional 10000 people through the route $1-3-4$ totalling in 20000 people per hour. It is impossible to evacuate more people each hour, as the total capacity of the roads outgoing from the city number 1 is 20000 people per hour.

Pay attention to this example if you think of using a simple Ford–Fulkerson algorithm. Note how it works on such graph, and why it may be a bad idea to use this algorithm on big networks with large capacities.

Starter Files

The starter solutions for this problem read the data from the input, build a graph data structure optimized for finding maximum flow in the graph, pass it to a blank procedure for finding the maximum flow and output the result. You need to implement this procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `evacuation`

What To Do

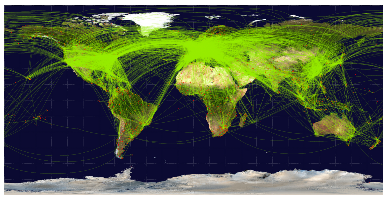
Implement an algorithm for finding maximum flow described in the lectures, but be careful with the choice of the algorithm, see the comments for the second example from the problem statement.

Need Help?

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

2 Problem: Assigning Airline Crews to Flights

Problem Introduction



In this problem, you will apply an algorithm for finding maximum matching in a bipartite graph to assign airline crews to flights in the most efficient way.

Problem Description

Task. The airline offers a bunch of flights and has a set of crews that can work on those flights. However, the flights are starting in different cities and at different times, so only some of the crews are able to work on a particular flight. You are given the pairs of flights and associated crews that can work on those flights. You need to assign crews to as many flights as possible and output all the assignments.

Input Format. The first line of the input contains integers n and m — the number of flights and the number of crews respectively. Each of the next n lines contains m binary integers (0 or 1). If the j-th integer in the i-th line is 1, then the crew number j can work on the flight number i, and if it is 0, then it cannot.

Constraints. $1 \leq n, m \leq 100$.

Output Format. Output n integers — for each flight, output the 1-based index of the crew assigned to this flight. If no crew is assigned to a flight, output -1 as the index of the crew corresponding to it. All the positive indices in the output must be between 1 and m, and they must be pairwise different, but you can output any number of -1's. If there are several assignments with the maximum possible number of flights having a crew assigned, output any of them.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time in seconds	1	1	1.5	5	1.5	2	5	5	3

Memory Limit. 512Mb.

Sample 1.

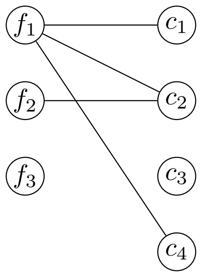
Input:

```
3 4
1 1 0 1
0 1 0 0
0 0 0 0
```

Output:

```
1 2 -1
```

Explanation:
In this sample, the bipartite graph of flights (on the left) and crews (on the right) looks like this:

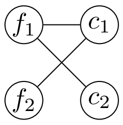


We can assign first crew to the first flight and second crew to the second flight, and no crews can work on the third flight, so this is an optimal assignment.

Sample 2.

Input:
```
2 2
1 1
1 0
```
Output:
```
2 1
```

Explanation:
In this sample, the bipartite graph of flights (on the left) and crews (on the right) looks like this:



If we assign the first crew to the first flight, we won't be able to assign any crew to the second flight. It is optimal to assign the second crew to the first flight and the first crew to the second flight, because this way we have a crew assigned to each flight.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a blank procedure that implements an incorrect greedy algorithm for finding the maximum matching, and output the result. You need to change this procedure to implement a correct algorithm for finding the maximum matching if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `airline_crews`

What To Do

Implement an algorithm for finding the maximum matching in the bipartite graph that was described in the lectures.

Need Help?

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

3 Advanced Problem: Stock Charts

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

We would like to thank the organizers of Google Code Jam — the annual worldwide programming competition held by Google — for allowing us to use this problem from one of the past rounds. Yes, that's right: you have a problem from a world championship in programming in your first homework, but hey, this is the fifth course in the Data Structures and Algorithms Specialization, and it has the words "Advanced Algorithms" in its name for a reason :) You've made it thus far — you can do this.

In this problem you will need to guess how to apply the algorithms from this module to find the most compact way of visualizing stock price data using charts.

Problem Description

Task. You're in the middle of writing your newspaper's end-of-year economics summary, and you've decided that you want to show a number of charts to demonstrate how different stocks have performed over the course of the last year. You've already decided that you want to show the price of n different stocks, all at the same k points of the year.

A simple chart of one stock's price would draw lines between the points $(0, price_0), (1, price_1), \ldots, (k-1, price_{k-1})$, where $price_i$ is the price of the stock at the i-th point in time.

In order to save space, you have invented the concept of an overlaid chart. An overlaid chart is the combination of one or more simple charts, and shows the prices of multiple stocks (simply drawing a line for each one). In order to avoid confusion between the stocks shown in a chart, **the lines in an overlaid chart may not cross or touch**.

Given a list of n stocks' prices at each of k time points, determine the minimum number of overlaid charts you need to show all of the stocks' prices.

Input Format. The first line of the input contains two integers n and k — the number of stocks and the number of points in the year which are common for all of them. Each of the next n lines contains k integers. The i-th of those n lines contains the prices of the i-th stock at the corresponding k points in the year.

Constraints. $1 \leq n \leq 100$; $1 \leq k \leq 25$. All the stock prices are between 0 and 1 000 000.

Output Format. Output a single integer — the minimum number of overlaid charts to visualize all the stock price data you have.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time in seconds	2	2	3	10	3	4	10	10	6

Memory Limit. 512Mb.

Sample 1.

Input:

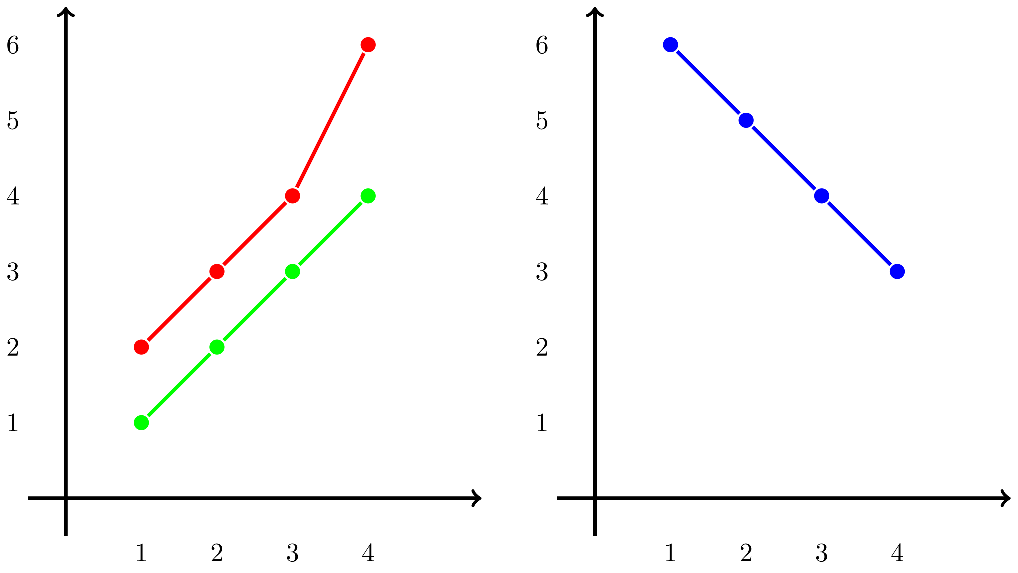
```
3 4
1 2 3 4
2 3 4 6
6 5 4 3
```

Output:

```
2
```

Explanation:

This data can be put into two following overlaid charts:



However, we cannot put all the data in one overlaid chart, as the lines corresponding to the third stock would touch the lines corresponding to the second stock, because they have the same price value at the third point.

Sample 2.

Input:

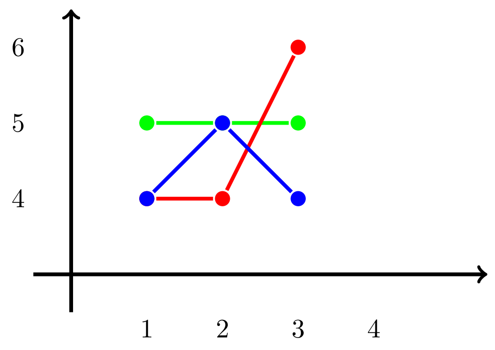
```
3 3
5 5 5
4 4 6
4 5 4
```

Output:

```
3
```

Explanation:

Each stock can be put on its own overlaid stock chart, of course. But no two stocks can be put on the same overlaid stock chart: first and second would intersect between points 2 and 3, first and third would touch in point 2, second and third would touch in point 1.



Starter Files

The starter solutions for this problem read the data from the input, pass it to a procedure and output the result. This procedure tries to pack the stock price data into the minimum possible number of overlaid charts using a greedy algorithm, but it sometimes fails. You need to implement another algorithm in this procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `stock_charts`

What To Do

Determine what are the conditions under which two stocks can be put on the same chart. Then think when more than 2 stocks can be put on the same chart.

Try to reduce the problem to the maximum matching in a bipartite graph problem, and to the first trick on the way is to create a bipartite graph of stocks with **two nodes for each stock**.

Need Help?

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