**CSCI 3100 – Fall 2016**
**Algorithms**
**Michael H. Goldwasser**
Saint Louis University

Handout: Applications of Network Flow

Monday, 7 November 2016

# Applications of Network Flow

We will examine a range of interesting computational problems that can effectively be solved using a network flow model and the computation of a maximum flow. The interesting question for these is how to reduce the original problem to that of a standard maximum $s$-$t$ flow computation. Considerations for successfully defining such a strategy include:

- designating an appropriate flow network, if not already inherent in the problem, or modifying an existing graph structure to produce a meaningful flow network
- identifying appropriate capacity constraints for all edges in the flow network
- determining how a solution in the form of a maximum flow (or minimum cut) for the flow network can be used to produce the desired solution for the original problem

# Problems to Consider

1. **Multiple Sources and Sinks**
   Consider a situation in which there are an arbitrary set of vertices designated as sources that can produce flow, and there are an arbitrary set of other vertices designated as sinks, that can absorb flow. Now consider the maximum amount of flow that can be successfully delivered from the combination of sources to the combination of sinks. For motivation, this model might be used by Amazon in which they have multiple warehouses stocked with product, and they must deliver product to many locations (but they have flexibility in determining which warehouses supply which units of delivery). Show that this model can be reduced to a standard maximum-flow computation on a graph with a single source and sink.

2. **Vertex Capacities**
   In our standard model, each edge has a capacity that designates the maximum amount of flow that may pass through that edge. Consider a model in which we wish to enforce additional vertex constraints, namely that we have some capacity $c(v)$ for each vertex $v$, designating the maximum amount of flow that may pass through that vertex. Show that this can be reduced to a standard maximum-flow computation.

3. **Maximum Bipartite Matching**
   Consider an unweighted bipartite graph, that is, a graph in which there are two distinct sets of vertices and all edges go from a vertex in one set to a vertex in the other set. Such a graph may be used, for example, to model employees and jobs, with a vertex for each employee and a vertex for each job, and an edge between an employee and a job if that employee is qualified to perform that job.

   A matching is a subset of edges such that each vertex has at most one such incident edge. (In our example, it would be an assignment to which employee should perform which job, assuming that an employee may hold at most one job, and that a job is held by at most one employee).

   Show that the problem of computing the maximum matching of a bipartite graph can be reduced to a maximum-flow computation.

4. **Edge-Disjoint Paths**
   Consider an unweighted, undirected graph. We consider two paths to be *edge-disjoint* if they do not share any edges. Edge-disjoint paths provide some level of redundancy in a real-world model in which there may be edge failures (e.g., roads closed, bridges out, communication links downs).

   Our goal is to determine a maximal set of edge-disjoint paths that connect given vertices $s$ and $t$. Show how this can be determined using a maximum-flow computation.

   **Note well:** We consider two paths to be conflicting if they both use the same edge, even if they use that edge in opposite directions. Does your approach guarantee this property?

5. **Vertex-Disjoint Paths**
   We consider a different notion of paths that are *vertex-disjoint*, meaning that they do not contain any common vertices other than their endpoints. (Note that a set of vertex-disjoint paths are surely edge-disjoint, but a set of edge-disjoint paths are not necessarily vertex-disjoint).

   Show how to compute a maximal set of vertex disjoint paths between a given pair of vertices $s$ and $t$.

6. **Escape Problems**
   The problem of finding edge-disjoint or vertex-disjoint paths can be rephrased in the form of "escape" problems, as in the attached "Leaping Lizards" and "Down Went the Titatnic" programming contest problems.

7. **Minimum Cuts**
   Given the correspondence of the max-flow, min-cut theorem, we should keep in mind that we can use a maximum flow computation to compute the minimum cut, when that cut is desired. See, for example, the attached "Sabotage" contest problem.

8. **Other Optimization Problems with Constraints**
   See some other problem contest problems that can be solved as a flow network (once you realize how to model it):

   - My T-shirt Suits Me
   - Collectors Problem
   - Tile Cut

# Problem H: Leapin' Lizards

Source file: `lizards.{c,cpp,java}`
Input file: `lizards.in`

Your platoon of wandering lizards has entered a strange room in the labyrinth you are exploring. As you are looking around for hidden treasures, one of the rookies steps on an innocent-looking stone and the room's floor suddenly disappears! Each lizard in your platoon is left standing on a fragile-looking pillar, and a fire begins to rage below...

Leave no lizard behind! Get as many lizards as possible out of the room, and report the number of casualties.

The pillars in the room are aligned as a grid, with each pillar one unit away from the pillars to its east, west, north and south. Pillars at the edge of the grid are one unit away from the edge of the room (safety). Not all pillars necessarily have a lizard. A lizard is able to leap onto any unoccupied pillar that is within *d* units of his current one. A lizard standing on a pillar within leaping distance of the edge of the room may always leap to safety... but there's a catch: each pillar becomes weakened after each jump, and will soon collapse and no longer be usable by other lizards. Leaping onto a pillar does not cause it to weaken or collapse; only leaping off of it causes it to weaken and eventually collapse. Only one lizard may be on a pillar at any given time.

**Input:** The input file will begin with a line containing a single integer representing the number of test cases, which is at most 25. Each test case will begin with a line containing a single positive integer *n* representing the number of rows in the map, followed by a single non-negative integer *d* representing the maximum leaping distance for the lizards. Two maps will follow, each as a map of characters with one row per line. The first map will contain a digit (0-3) in each position representing the number of jumps the pillar in that position will sustain before collapsing (0 means there is no pillar there). The second map will follow, with an `'L'` for every position where a lizard is on the pillar and a `'.'` for every empty pillar. There will never be a lizard on a position where there is no pillar.

Each input map is guaranteed to be a rectangle of size $n$ x $m$, where $1 \le n \le 20$ and $1 \le m \le 20$. The leaping distance is always $1 \le d \le 3$.

**Output:** For each input case, print a single line containing the number of lizards that could not escape. The format should follow the samples provided below.

**Warning:** Brute force methods examining every path will likely exceed the allotted time limit.

| **Example input:** | **Example output:** |
|---|---|
| <pre>4<br>3 1<br>1111<br>1111<br>1111<br>LLLL<br>LLLL<br>LLLL<br>3 2<br>00000<br>01110<br>00000</pre> | <pre>Case #1: 2 lizards were left behind.<br>Case #2: no lizard was left behind.<br>Case #3: 3 lizards were left behind.<br>Case #4: 1 lizard was left behind.</pre> |

```
.....
.LLL.
.....
3 1
00000
01110
00000
.....
.LLL.
.....
5 2
00000000
02000000
00321100
02000000
00000000
........
........
..LLLL..
........
........
```

*Last modified on October 30, 2005 at 1:15 PM*.

# Problem D
# Down Went The Titanic

Time Limit: 8 Second

After the collision of the great Titanic with the iceberg, it went down. Now there are peoples floating in the cold water struggling with death. Some helping ship will arrive to save them. But they have to survive until the ships arrive. Now consider a water area with people, floating ices, large woods etc. Consider the following symbols:

| | |
|---|---|
| * | People staying on floating ice. People want to move from here as the floating ice cannot carry them for long time. Once a people move from here, the floating ice will get drowned. People can move to any of the four directions (north, east, west and south). |
| ~ | Water. People cannot go or move through them as water is extremely cold and not good enough for swimming. |
| . | Floating ice. People can move to a floating ice. But floating ices are so light that they cannot float for long time, so people should move from here as soon as possible and once a people move from here, the floating ice will get drowned. |
| @ | Large iceberg. People can move here but cannot stay here as they are extremely cold. These icebergs will remain floating all the time. Note that, no two people can stay on floating ice or large iceberg at the same time. |
| # | Large wood. This place is safe. People can move and stay here until the helping ships arrive. A large wood will get drowned if more than P people stay on it at the same time. |

Given the description of the area you have to find an optimal strategy that ensures the maximum number of living people.

## Input:

The input contains a number of test cases. Each test case starts with a line containing three integers X, Y and P, where X, Y is the dimensions of the area (0 < X, Y < 31) and P (P < 11) is the highest capacity of the large woods. Next X lines each contains Y characters. These lines contain no blank spaces or any characters other than asterisk (*), tilde (~), dot (.), at (@) and hash (#). Not more than 50% of the total area has a people. Input will terminate with end of file (EOF). There is a blank line between two consecutive test cases.

## Output:

For each test case print one line of output, an integer denoting the maximum number of survivors possible.

**SAMPLE INPUT**

```
3 4 2
*~~#
...@
.~.*

3 5 1
~~*~~
#.@.#
~~*~~

1 4 2
**#~
```

**OUTPUT FOR SAMPLE INPUT**

```
2
2
1
```

---

Problemsetter: Ishtiak Zaman
Special Thanks To: Md. Mahbubul Hasan

## Sabotage

The regime of a small but wealthy dictatorship has been abruptly overthrown by an unexpected rebellion. Because of the enormous disturbances this is causing in world economy, an imperialist military super power has decided to invade the country and reinstall the old regime.

For this operation to be successful, communication between the capital and the largest city must be completely cut. This is a difficult task, since all cities in the country are connected by a computer network using the Internet Protocol, which allows messages to take any path through the network. Because of this, the network must be completely split in two parts, with the capital in one part and the largest city in the other, and with no connections between the parts.

There are large differences in the costs of sabotaging different connections, since some are much more easy to get to than others.

Write a program that, given a network specification and the costs of sabotaging each connection, determines which connections to cut in order to separate the capital and the largest city to the lowest possible cost.

### Input

Input file contains several sets of input. The description of each set is given below.

The first line of each set has two integers, separated by a space: First one the number of cities, ***n*** in the network, which is at most 50. The second one is the total number of connections, ***m***, at most 500.

The following ***m*** lines specify the connections. Each line has three parts separated by spaces: The first two are the cities tied together by that connection (numbers in the range 1 - ***n***). Then follows the cost of cutting the connection (an integer in the range 1 to 40000000). Each pair of cites can appear at most once in this list.

Input is terminated by a case where values of ***n*** and ***m*** are zero. This case should not be processed. For every input set the capital is city number 1, and the largest city is number 2.

### Output

For each set of input you should produce several lines of output. The description of output for each set of input is given below:

The output for each set should be the pairs of cities (i.e. numbers) between which the connection should be cut (in any order), each pair on one line with the numbers separated by a space. If there is more than one solution, any one of them will do.

Print a blank line after the output for each set of input.

## Sample Input

```
5 8
1 4 30
1 3 70
5 3 20
4 3 5
4 5 15
5 2 10
3 2 25
2 4 50
5 8
1 4 30
1 3 70
5 3 20
4 3 5
4 5 15
5 2 10
3 2 25
2 4 50
0 0
```

## Sample Output

```
4 1
3 4
3 5
3 2

4 1
3 4
3 5
3 2
```

---

*Problem setter: Jesper Larsson, Lund University, Sweden*

# 4061 - My T-shirt suits me

Time limit: s
Memory limit: MB

## Problem Description

Our friend Victor participates as an instructor in an environmental volunteer program. His boss asked Victor to distribute N T-shirts to M volunteers, one T-shirt each volunteer, where N is multiple of six, and N>=M. There are the same number of T-shirts of each one of the six available sizes: XXL, XL, L, M , S, and XS. Victor has a little problem because only two sizes of the T-shirts suit each volunteer.

You must write a program to decide if Victor can distribute T-shirts in such a way that all volunteers get a T-shirt that suit them. If N != M, there can be some remaining T-shirts.

## Input

The first line of the input contains the number of test cases. For each test case, there is a line with two numbers N and M. N is multiple of 6, 1<=N<=36, and indicates the number of T-shirts. Number M, 1<=M<=30, indicates the number of volunteers, with N>=M. Subsequently, M lines are listed where each line contains, separated by one space, the two sizes that suit each volunteer (XXL, XL, L, M , S, or XS).

## Output

For each test case you are to print a line containing YES if there is, at least, one distribution where T-shirts suit all volunteers, or NO, in other case.

## Sample Input

```
3
18 6
L XL
XL L
XXL XL
S XS
M S
M L
6 4
S XL
L S
L XL
```

```
L XL
6 1
L M
```

## Sample Output

```
YES
NO
YES
```

---

**Problemsetter: boss**

# Collector's Problem

**Input:** standard input
**Output:** standard output
**Time Limit:** 5 seconds

Some candy manufacturers put stickers into candy bar packages. Bob and his friends are collecting these stickers. They all want as many different stickers as possible, but when they buy a candy bar, they don't know which sticker is inside.
It happens that one person has duplicates of a certain sticker. Everyone trades duplicates for stickers he doesn't possess. Since all stickers have the same value, the exchange ratio is always 1:1.
But Bob is clever: he has realized that in some cases it is good for him to trade one of his duplicate stickers for a sticker he already possesses.
Now assume, Bob's friends will only exchange stickers with Bob, and they will give away only duplicate stickers in exchange with different stickers they don't possess.
Can you help Bob and tell him the maximum number of different stickers he can get by trading stickers with his friends?

## Input

The first line of input contains the number of cases $T$ ($T<=20$).
The first line of each case contains two integers $n$ and $m$ ($2<=n<=10$, $5<=m<=25$). $n$ is the number of people involved (including Bob), and $m$ is the number of different stickers available.
The next $n$ lines describe each person's stickers; the first of these lines describes Bob's stickers.
The $i$-th of these lines starts with a number $k_i<=50$ indicating how many stickers person $i$ has.
Then follows $k_i$ numbers between $1$ and $m$ indicating which stickers person $i$ possesses.

## Output

For each case, print the test case number together with the maximum number of different stickers Bob can get.

## Sample Input

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

## Sample Output

```
Case #1: 1
Case #2: 3
```

## Explanation of the sample output:

In the first case, no exchange is possible, therefore Bob can have only the sticker with number 1.
In the second case, Bob can exchange a sticker with number 1 against a sticker with number 2 of the second person,
and then this sticker against a sticker with number 3 or 4 of the third person, and now he has stickers 1, 2 and 3 or 1, 2 and 4.

---

**Problem setter: Adrian Kuegel**

Problem K — limit 10 seconds

# Tile Cut

When Frodo, Sam, Merry, and Pippin are at the Green Dragon Inn drinking ale, they like to play a little game with parchment and pen to decide who buys the next round. The game works as follows:

Given an $m \times n$ rectangular tile with each square marked with one of the incantations `W`, `I`, and `N`, find the maximal number of triominoes that can be cut from this tile such that the triomino has `W` and `N` on the ends and `I` in the middle (that is, it spells `WIN` in some order). Of course the only possible triominoes are the one with three squares in a straight line and the two ell-shaped ones. The Hobbit that is able to find the maximum number wins and chooses who buys the next round. Your job is to find the maximal number.

Side note: Sam and Pippin tend to buy the most rounds of ale when they play this game, so they are lobbying to change the game to Rock, Parchment, Sword (RPS)!

## Input

Each input file will contain multiple test cases. Each test case consists of an $m \times n$ rectangular grid (where $1 \leq m, n \leq 30$) containing only the letters `W`, `I`, and `N`. Test cases will be separated by a blank line. Input will be terminated by end-of-file.

## Output

For each input test case, print a line containing a single integer indicating the maximum total number of tiles that can be formed.

**Sample Input**

```
WIIW
NNNN
IINN
WWWI

NINWN
INIWI
WWWIW
NNNNN
IWINN
```

**Sample Output**

```
5
5
```