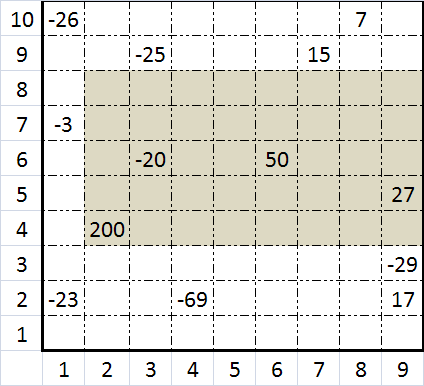
**Java program:** Prob11.java

**Input File:** Prob11.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

A farmer wants to buy a rectangular plot of land, but some parts of the land are worth more than others. The land has been partitioned into a grid, with the value of each grid section given in thousands of dollars (see figure). Grid locations with no value have a value of $0.

Your task is to write a program that will find the most valuable rectangular plot of land that is the largest in size (represented by the shaded area in the figure).

**Program Input**

The file Prob11.in.txt will contain data about the land. Numbers in the file will be separated by commas. The file will contain two sections:

1. The first line will give the total size of the available land for purchase. The number of columns will be listed first, and the number of rows will be listed second.
2. The rest of the file will contain land value data in the following format:

column,row,value

The value will be an integer, representing the value of the land in thousands of dollars.

The "origin" will be at the bottom left of the land, so for a plot of land with C columns and R rows:

* Point 1,1 will be the grid section at the bottom left of the land
* Point C,R will be the grid section at the top right of the land

**Example Input:**

9,10

1,2,-23

4,2,-69

9,2,17

9,3,-29

2,4,200

9,5,27

3,6,-20

6,6,50

1,7,-3

3,9,-25

7,9,15

1,10,-26

8,10,7

**Program Output**

Your output should contain the following three lines:

1. The location of the bottom left corner of the land the farmer should purchase.
2. The location of the top right corner of the land the farmer should purchase.
3. The total value of the plot of land that the farmer should purchase, with a dollar sign before the value and a lower case k after value.

**Example Output:**

2,4

9,8

$257k

**Java program:** Prob12.java

**Input File:** Prob12.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

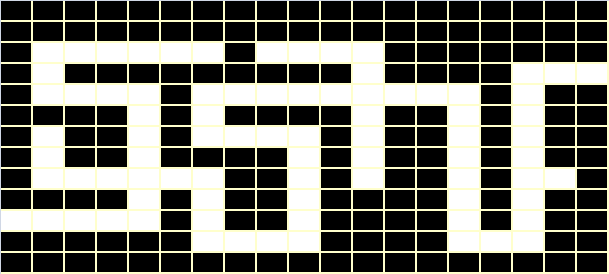
To help your team navigate through the dense jungle, you will be presented with the layout of the terrain as captured by an aerial photo. Your job is to write a program to map out the one and only route to pass completely through the jungle.

**Program Input**

The file Prob12.in.txt will contain a representation of the aerial photo of the jungle. Each line represents a new row of data. An X represents impassible jungle, and an O represents a navigable path.

**Example Input:**

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXX

XOOOOOOXOOOOXXXXXXX

XOXXXXXXXXXOXXXXOOO

XOOOOXOOOOOOOOOXOXX

XXXXOXOXXXXOXXOXOXX

XOXXOXOOOOXOXXOXOXX

XOXXOXXXXOXOXXOXOXX

XOOOOOOXXOXOXXOXOOX

XXXXOXOXXOXXXXOXOXX

OOOOOXOXXOXXXXOXOXX

XXXXXXOOOOXXXXOOOXX

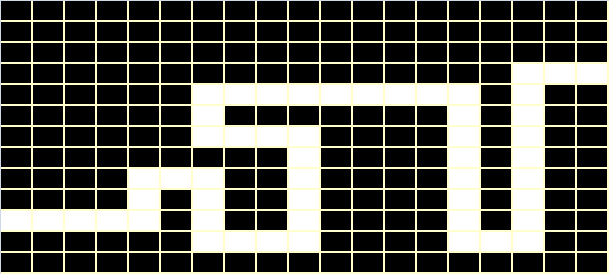
XXXXXXXXXXXXXXXXXXX

**Program Output**

Your program should output a representation of the jungle with all paths that do not belong to the maze solution removed and replaced with an X. That way, if your team were to navigate through the jungle you would know which way to go at each intersection. There will be no loops on the correct path, and any orphaned paths (i.e., paths that do not connect to the maze solution) should also be removed. Only horizontal and vertical movement is permitted.

**Example Output:**

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXOOO

XXXXXXOOOOOOOOOXOXX

XXXXXXOXXXXXXXOXOXX

XXXXXXOOOOXXXXOXOXX

XXXXXXXXXOXXXXOXOXX

XXXXOOOXXOXXXXOXOXX

XXXXOXOXXOXXXXOXOXX

OOOOOXOXXOXXXXOXOXX

XXXXXXOOOOXXXXOOOXX

XXXXXXXXXXXXXXXXXXX

**Java program:** Prob13.java

**Input File:** Prob13.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Grading tests is hard – just ask your teachers! Your task is to write a program that will read in an answer key and student responses, and grade the students’ responses based on the key.

**Program Input**

The file Prob13.in.txt will contain two sections:

1. **The answer key**

Each line of the answer key section will be in the following format:

ProblemNumber Points Operand AnswerList

Here are some notes on the answer key:

* ProblemNumber will be an integer
* Points will be an integer representing how many points to add or subtract from a student if their answer matches the criteria given by the operand and the answer list
* Operand will be one of the following three strings and shall invoke the following scoring rules:
  + EQ: This stands for equals. The student’s answer must exactly match one of the answers in the answer list to get credit for this answer. For example, if the key said EQ "250.0" and the student’s answer was "250.00", the student would not receive points.
  + IN: This stands for includes. At least one of the answers in the answer list must be contained in the student’s answer for the student to get credit. For example, if the key said IN "250.0" and the student’s answer was "250.00", the student would receive points.
  + EX: This stands for excludes. None of the answers in the answer list can be contained in the student’s answer for the student to get credit. For example, if the key said EX "250.0" and the student’s answer was "250.00", the student would not receive points.
* AnswerList will be any number of quoted text strings separated by commas. Text strings will not contain quotation marks.
* A line containing problem number 999 will mark the end of the key. Problem 999 may or may not have an answer.

1. **The student responses**

Each line of the student answer section will be in the following format:

StudentId ProblemNumber Answer

Here are some notes on the student answers:

* StudentId will be a three digit number, and may contain leading zeroes. Each student has a unique Id number.
* ProblemNumber will be an integer
* Answer will be a quoted text string which is the student's answer to that question. Only one text string will occur for each answer. Text strings will not contain quotation marks.

**Example Input:**

1 7 IN "36 IN","3 F"

2 5 EQ "250","250.0"

2 2 EX "N/A"

3 7 IN "ADAMS"

3 -7 IN "Q"

999

001 1 "36 INCHES"

001 2 "N/A"

001 3 "QUEEN ELIZABETH"

002 1 "33 FURLONGS"

002 2 "250 or 395"

002 3 "JOHN QUINCY ADAMS"

003 1 "3 FEET or 36 INCH"

003 2 "250.0"

003 3 "EDIE ADAMS OR JAMES MADISON"

**Program Output**

Your program should display the points earned by each student for each question, as well as displaying each student’s total points earned.

Grading rules:

* Questions may have more than one line in the key, and a student’s answer may match multiple key lines.
* Some key lines will be worth negative points. In the event that a student has negative points for a particular question, they should receive a score of 0 for that question.
* Each key line should only be counted once per student when awarding points.

For individual question output, use the following format:

STUDENT StudentId ProblemNumber PointsAwarded

For student total output, use the following format:

STUDENT StudentId TOTAL TotalPoints

**Example Output:**

STUDENT 001 1 7

STUDENT 001 2 0

STUDENT 001 3 0

STUDENT 001 TOTAL 7

STUDENT 002 1 7

STUDENT 002 2 2

STUDENT 002 3 0

STUDENT 002 TOTAL 9

STUDENT 003 1 7

STUDENT 003 2 7

STUDENT 003 3 7

STUDENT 003 TOTAL 21

**Java program:** Prob14.java

**Input File:** Prob14.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

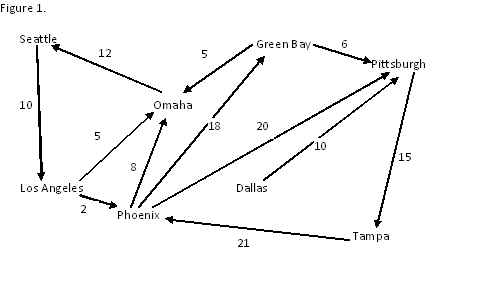
With the cost of gas at an all time high, it is useful to know how to get places quickly. A certain airline uses a series of one-way flights to get its customers from place to place. Your task is to write a program that will take route information and figure out the cheapest way to get a passenger from point A to point B.

**Program Input**

The file Prob14.in.txt will contain two sections:

1. The first line will contain the departure and arrival cities separated by a semicolon.
2. The rest of the lines will contain route data. Each route will have a departure city, an arrival city, and the cost of traveling that route all separated by semicolons. All routes are one way, but it is possible to have two routes between two cities by having two different route information lines. All costs will be integers.

**Example Input:**



Phoenix;Tampa

Seattle;Los Angeles;10

Los Angeles;Omaha;5

Los Angeles;Phoenix;2

Omaha;Seattle;12

Phoenix;Omaha;8

Phoenix;Green Bay;18

Phoenix;Pittsburgh;20

Green Bay;Omaha;5

Green Bay;Pittsburgh;6

Dallas;Pittsburgh;10

Pittsburgh;Tampa;15

Tampa;Phoenix;21

**Program Output**

Your program should output the flight path and also give the total cost in the following format:

DepartureCity->Intermediate->Cities->Destination=TotalCost

**Example Output:**

Phoenix->Pittsburgh->Tampa=35