South African Computer Olympiad Training DAY 2

Overview

Problem	Plant	Silly Game	Ships	Composite Numbers
Author	**Graham**	**Heinrich**	**Harry**	**Jacques**
Program name	`plant.exe`	`game.exe`	`ships.exe`	`comp.exe`

Source name	`plant.pas`
	`plant.jav`
	`plant.cpp`
		`game.pas`
		`game.jav`
		`game.cpp`
			`ships.pas`
			`ships.jav`
			`ships.cpp`
				`comp.pas`
				`comp.jav`
				`comp.cpp`

Input file	`plant.in`	`game.in`	`ships.in`	`comp.in`
Output files(10)	`Plant.out`	`game.out`	`ships.out`	`comp.out`
Time limit	2 seconds	2 seconds	1 second	2 seconds
Num. of tests	10	10	10	10
Points per test	10	10	10	10
Total points	**100**	**100**	**100**	**100**

The maximum total score for Day2 is 300 points.

Manufacturing Plant

Author: Graham Poulter

Introduction

An ailing heavy-vehicle manufacturing plant produces a number of types of heavy vehicles, each of which has a profit value per vehicle and takes a certain number of hours to produce. You have been hired to tell them how many of each type of vehicle to produce to maximise its profit within a given time period. Due to short supply of special-purpose parts, each vehicle type also has a maximum limit for the number that can be produced.

Unfortunately, the managers of the plant don't know exactly how many hours it takes to produce each kind of vehicle, although they do know the profit value of the finished product. Each vehicle type has a list of tasks, and each task takes a certain number of hours to complete. The tasks, which must be completed before a certain task can begin, are called the dependencies of that task. Some tasks have no dependencies and so can begin right away and in parallel. There are other tasks, which have dependencies but are not dependencies for any others, and so once all of those tasks are complete then there cannot be any more tasks outstanding and the vehicle is finished. Given the durations and dependencies of the tasks, you have to work out the Earliest Finish Time (EFT) for each type of vehicle, which is the shortest amount of time in which one of those vehicles can be produced.

To prove to the manager that you're not guessing the EFT's you also have to produce for each vehicle type an ordered list of tasks beginning with one which has no dependencies and ending with one which has no others depending on it, with each task in the list being a dependency of the task immediately to the right of it, such that the sum of the durations of the tasks in the list is equal to your answer for the EFT.

Constraints

- Due to lack of floor space, only one vehicle can be under production at any one time (i.e. you cannot build two vehicles in parallel), though work on the next vehicle can begin immediately after the current one is finished.
- There will never be a cycle of dependencies in the tasks, e.g. task C depends on B depends A depends on C.
- The total number of hours, *T*, available is between 50 and 10000.
- The number of vehicles types, *V*, is between 1 and 100
- The maximum quantity, *M*, of a certain vehicle type is between 1 and 50.
- The profit, *P*, from making one of a certain vehicle type is between 1 and 5000.
- The number of tasks, *Q*, in making a certain vehicle type is between 1 and 500
- The length in hours, *L*, for completing a certain task is between 0 and 10.
- The number of dependencies, *D*, for each task is between 0 and *Q*-1.
- Time limit is 2 seconds.

Input Format (plant.in)

Line 1: Two integers, *T* and *V*. *T* is the total amount of time available. *V* is the number of available vehicle types.

There are then *V* sections representing the vehicles 1.*V*. Each section has the following format:

Line 1: Three integers, *M*, *P* and *Q*. *M* is the maximum quantity of the vehicle, which can be produced. *P* is the profit, which can be made from selling one of the vehicles. *Q* is the number of tasks involved in making the vehicle.

Line 2…*Q*+1: Each line is one of the tasks 1…*Q* required to make the vehicle. First on the line are two integers, *L* and *D*. *L* is the number of hours required to do that particular task. *D* is the number dependencies for the task, which is zero for tasks that can begin right away. Then, on the same line, are *D* integers $P_1 \ldots P_D$ each in the range 1…*Q* representing those tasks on which the current task depends.

Output Format (plant.out)

Lines 1…2*V: Two lines for each type of vehicle. On the first line are three integers, *PROD*, *EFT*, and *LEN*. *PROD* is the number of that type of vehicle that must be produced in the time period *T* to maximise profit. *EFT* is the Earliest Finish Time for that type of vehicle. *LEN* is the number of tasks in a list of tasks whose sum is the EFT. On the second line are *LEN* integers each in the range 1…*Q* representing the ordered list of tasks, beginning with a task which has no dependencies and ending with a task on which no others depend, and each task depending on the task to the left of it, such that the sum of the times to complete the list of tasks equals the EFT.

Scoring

- There are 10 test cases, each worth 10 points.
- Any invalid output automatically scores 0 points for the case.
- If *PROD* is greater than *M* for any vehicle type, the output is invalid.
- If your list of tasks for an *EFT* is longer or shorter than *LEN*, does not obey dependencies or does not sum to exactly *EFT*, the output is invalid.
- If your total production time is greater than *T* using your *EFT*'s then the output is invalid (this is only once all your task-lists giving those *EFT*'s are found to be valid, even if non-optimal). Multiplying PROD and EFT for each vehicle type and summing find the total production time.

- A score out of 5 for your total profit (worked out by multiplying *PROD* and *P* for each vehicle type and summing) is calculated from the following ranges, where percentages are of the optimal profit and the score is in brackets. 100% of optimal scores 5 points.

  (0) 80% (1) 85% (2) 90% (3) 95% (4) 105% (3) 110% (2) 115% (1) 120% (0)

- For each *EFT*, exactly the same proportional scaling is done on the optimal EFT. The scores for the *EFTs* are then summed and divided by *V* for a score out of 5.

Sample Input

```
30 3     Time period 30 hours, there are 3 types of vehicles
7 4 3    Vehicle type 1 has a limit of 7, profit of 4 and 3 tasks
1 0      Task 1 has duration 1 and 0 dependencies
4 1 1    Task 2 has duration 4 and 1 dependency, task 1
3 1 2    Task 3 has duration 3 and 1 dependency, task 2
6 2 4    Vehicle type 2 has a limit of 6, profit of 2, & 4 tasks
3 0      Task 1 has duration 3 and 0 dependencies
1 1 1    Task 2 has duration 1 and 1 dependency, task 1
2 1 1    Task 3 has duration 2 and 1 dependency, task 1
2 2 2 3  Task 4 has duration 2, 2 dependencies, tasks 2 & 3
2 5 1    Vehicle type 3 has a limit of 2, profit of 5 &1 task
5 0      Task 1 has duration 5 and 0 dependencies
```

Sample Output

```
2 8 3    Vehicle type 1: produce 2, EFT is 8 hrs, 3 tasks in list
1 2 3    Tasks 1, 2 and 3 (completed in order take 8 hrs)
0 7 3    Vehicle type 2: produce 0, EFT is 7 hrs, 3 tasks in list
1 3 4    Tasks 1, 3 and 4 (completed in order take 7 hrs)
2 5 1    Vehicle type 3: produce 2, EFT is 5 hrs, 1 task in list
1        Task 1 (completed takes 5 hrs)
```

The total profit is 2*4 + 0*2 + 2*5 = 18 (the optimal profit)
The total production time is 2*8 + 0*7 + 2*5 = 26 hours (must be <= 30 for this case)

The Silly Game

AUTHOR: Heinrich du Toit

Description:

A 2-player game is played with a strange board.
The board contains N circles drawn on it.
There is one pawn that is placed in circle number 1.
Between the circles there are lines drawn on the board and on each line a number is written. The players take turns to move the pawn to another circle along the line. The player then collects the points on the line.
When the pawn reach circle N the game ends and the player with the most points win.

Task

You must write a program to play this game.
Read the board data from the file: game.in and play against the other player using stdin and stdout.

Input: game.in

line 1: one integer, N
line 2: one integer, R , the number of lines by which the pawn may move.
the next r lines each contain 3 integers.
Ai, Bi, Si
Means: The pawn may move from circle A to circle B (not the other way) and then the player collects S points.

Example

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

Constraints

Ai < Bi
2 <= N <= 1000
1 <= R <= 15000
0 <= Si <= 1000
There will be no more than 1 line between any 2 circles.
You must play against a computer opponent writing your moves to standard output and reading your opponent' s moves from standard input. You are player one and move first.
When it is your turn, write out a line containing a single integer that is the number of the circle to move to. When it is your opponent' s turn, read in a single line that will contain the circle to which your opponent moved. If you do something illegal (such as make an illegal move or put letters into your output) the evaluator will quit.
When the evaluator quits (because of something illegal or because the game is over) the standard input will close. See below for how to test for this in the languages available.
When you or the other player prints N, you should quit immediately. The other program will keep track of score and write output. Your program must not write an output file.
You will get more points if you win optimally. Winning optimally is defined as getting as many points ahead of your opponent as possible. It will always be possible for you to force a win.

PASCAL:

Use writeln(move) to make a move and readln(move) to receive a move from the opponent. Do not use the CRT unit as this may interfere with the redirection of input and output. You can test whether standard input has been closed (due to you doing something illegal) by checking eof(input). If you

don't do this check, you will get runtime error 106 the next time you try to readln.

C:

Use printf("%d\n", move) followed by fflush(stdout) to make a move, and use scanf("%d", &move) to receive a move. If standard input is closed then the call to scanf will return EOF.

C++:

Use cout << move << '\n' to make a move, and cin >> move to receive a move. If standard input has closed then after executing cin >> move, cin.eof() will be true (and move might be undefined).

Time

Time limit: 2 seconds.
Only your CPU time is counted; the time taken by the opponent and the time you spend waiting for the readln/scanf/cin call to return are not counted. There is also a 20 second real-time limit to catch programs that try to read a move when it is their turn to write a move.

Score:

If you win optimally: 100%
If you win but not optimally: 80%
If you draw: 50%
If you loose by less that 10 points: 10%
If you loose, time out or do something illegal: 0%

Ships

Author: Harry Wiggins

Description

There are N towns on both the north and south bank of the Mississippi river. Each town on the north bank has its unique friend town on the south bank. No two towns have the same friend.
Each pair of friend towns would like to have a ship line connecting them. They applied for permission to the government. Because it is often foggy on the river the government decided to prohibit intersection of ship lines (if two lines intersect there is a high probability of ship crash). Your task is to write a program to help government officials decide to which ship lines they should grant permission to get maximum number of non intersecting ship lines.

Input data

In the first line there are 2 integers M, N separated with exactly one space, M represents the length of the Mississippi river bank (10 <= M <= 6000), N represents the number of towns situated on both south and north riverbanks (1 <= N <= 5000). On each of the next N lines there are two positive integers A, B separated with space (A, B <= M), representing distances of the pair of friend towns from the western border of the Mississippi river measured along the riverbanks (A for the town on the north bank, B for the town on the south bank). There are no two towns on the same position on the same riverbank.

Output Data

The output file has to contain the maximum possible number of ship lines satisfying the conditions above.

SHIPS.IN

```
30 7
22 4
2 6
10 3
15 12
9 8
17 17
4 2
```

SHIPS.OUT

```
4
```

Scoring

You'll get partial marks if your answer is less than double the optimal answer. You have 1-second time limit.

Composite Numbers

Author: Jacques Conradie

Description

For a given set of K prime numbers S = {p_1, p_2, ..., p_K}, consider the set of all numbers whose prime factors are a subset of S. This set contains, for example, p_1, p_1p_2, p_1p_1, and $p_1p_2p_3$ (among others). This is the set of ' Composite numbers' for the input set S. Note: The number 1 is explicitly declared not to be a composite number.

Task

Your job is to find the Nth humble number for a given set S. Long integers (signed 32-bit) will be adequate for all solutions.

Input

Line 1. Two space separated integers: K and N.
Line 2. K space separated positive integers that comprise the set S.

Sample Input (file comp.in)

```
4 19
2 3 5 7
```

Output

The Nth composite number from set S printed alone on a line.

Sample Output (file comp.out)

```
27
```

Scoring

You will receive 100% for a correct solution and 0 otherwise.

Constraints

1 <= K <=100 and 1 <= N <= 100,000.