French-Australian Regional Informatics Olympiad

Friday 11th March, 2016

Duration: 4 hours

3 questions

All questions should be attempted

Problem 1
Hooliganism

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 1 second, 256 MB

The City of Sydney has been having problems with "hooliganism" as of late. Out of control youths are driving around non-stop all night, scaring hippos and driving over daisies. Now you must help implement a plan to lock out these hooligans once and for all.

The City of Sydney is made up of N intersections, numbered 1 to N, connected by R roads. Each night all roads in Sydney will become one-way. You decide which way each road goes. To prevent congestion, each intersection has a maximum number of roads that are allowed to leave from it.

You must find a way to make each of these roads one-way such that each intersection has no more than the specified number of roads leaving it, and that there is no way for hooligans to drive around all night. In particular, hooligans should not be able to drive through the same intersection more than once.

Input

- The first line of input will contain two space-separated integers N R, representing the number of intersections and roads in Sydney respectively.
- The following N lines of input will each contain an integer a_i, the i-th of which gives the maximum number of one-way roads allowed to leave intersection i.
- The following R lines of input will each contain two space-separated integers b c, indicating that intersections b and c have a road between them.

Output

If Sydney cannot prevent hooligans from driving around all night, then output `IMPOSSIBLE`.

If it is possible for Sydney to construct such a road network, then output R lines, each containing two integers b_i c_i, indicating a one-way road from intersection b_i to intersection c_i. These lines must describe every road in Sydney, and can be output in any order.

Sample Input 1

```
3 3
0
2
0
1 2
1 3
2 3
```

Sample Output 1

```
IMPOSSIBLE
```

Explanation 1

Intersections 1 and 3 are connected by a road, however neither of them are allowed to have any roads leaving them. Thus it is impossible to assign directions to the roads to satisfy the restrictions on the number of roads leaving each intersection.

Sample Input 2

```
3 3
1
1
1
1 2
1 3
2 3
```

Sample Output 2

```
IMPOSSIBLE
```

Explanation 2

Here it is possible to assign directions to the roads, however it requires you to make a ring going from intersection 1 to 2 to 3 and back to 1, or the opposite direction. This would allow hooligans to drive past the same location multiple times, so this case is also impossible.

Sample Input 3

```
3 3
1
2
0
1 2
1 3
2 3
```

Sample Output 3

```
1 3
2 1
2 3
```

Explanation 3

Once hooligans enter intersection 3, they cannot leave. The only place for hooligans to go from intersection 1 is intersection 3, thus once a hooligan arrives at intersection 1 they are no longer trouble. Similarly, a hooligan at intersection 2 can go to either intersection 1 or intersection 3, but either way they arrive at intersection 3 and cannot go any further. Thus no hooligan can drive past the same location more than once.

Subtasks & Constraints

For all subtasks, $1 \leq R \leq 100\,000$ and $2 \leq N \leq 100\,000$. Additionally, no two intersections will be directly connected by more than one road, and no intersection will be connected to itself. It is not necessarily possible to go from any intersection to any other.

- For Subtask 1 (20 points), $a_i = N$, that is, the maximum number of roads allowed to leave each intersection is the same as the number of intersections.
- For Subtask 2 (20 points), every intersection has exactly two other intersections connected to it.
- For Subtask 3 (30 points), $1 \leq R \leq 100\,000$ and $2 \leq N \leq 1\,000$.
- For Subtask 4 (30 points), no further constraints apply.

Scoring

The score for each input scenario will be 100% if a correct answer is written to the output file, and 0% otherwise.

Problem 2
No Ball

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 2 second, 512 MB

Today's the day you retire and you, like all retirees in *Graphland*, are shipped off to the beautiful country of *Flatland*. Suddenly flooded with nostalgia, you long for the good ol' days of playing cricket with your mates, whether it be in high school or at the far too rare reunions that you've had since. Of course you also remember the constant struggle to decide who would bring the kit and where you'd all play. You immediately decide to build your new house so that you and your mates will always have a great place to get together and play.

Flatland is a grid with the top row and left most column numbered 0, increasing from top to bottom and left to right. Each of your N friends lives in a cell of the grid. No two friends live in the same cell.

Unfortunately, Flatland is rather large so you will only be able to contact certain subsets of your friends. Specifically, you will be given a choice of Q *communication rectangles* and will be limited to only contacting the friends in the communication rectangle you choose. A cell (r, c) is inside the communication rectangle with upper-left and lower-right corners (R_1, C_1) and (R_2, C_2) respectively when $R_1 \leq r \leq R_2$ and $C_1 \leq c \leq C_2$.

You decide to find the best cell to build your house for each communication rectangle. This should be the cell that minimises the sum of the Manhattan distances to all of the friends in that communication rectangle. In case of a tie, pick the cell with the lowest row coordinate, then the lowest column coordinate.

Note that even though no two friends live on the same cell, you are allowed to build your house on top of an occupied cell[1].

Forever being "that friend who is obsessed with coding", you decide to write a program to find these locations for you.

Input

- The first line of input will contain two space-separated integers N Q, representing the number of friends and the number of communication rectangles respectively.
- The following N lines of input will each contain two space-separated integers r_i c_i, describing the row and column of your i-th friend's house.
- The following Q lines of input will each contain four space-separated integers R_1 C_1 R_2 C_2, describing the top-left and bottom-right corner of a communication rectangle.

Output

Your program must output Q lines each containing two space-seperated integers: the best row and column to build your house for the corresponding communication rectangle.

[1] Despite the name, Flatland has at least three dimensions

Sample Input

```
4 3
2 0
0 1
3 2
1 3
0 0 2 3
0 0 3 3
1 2 3 3
```

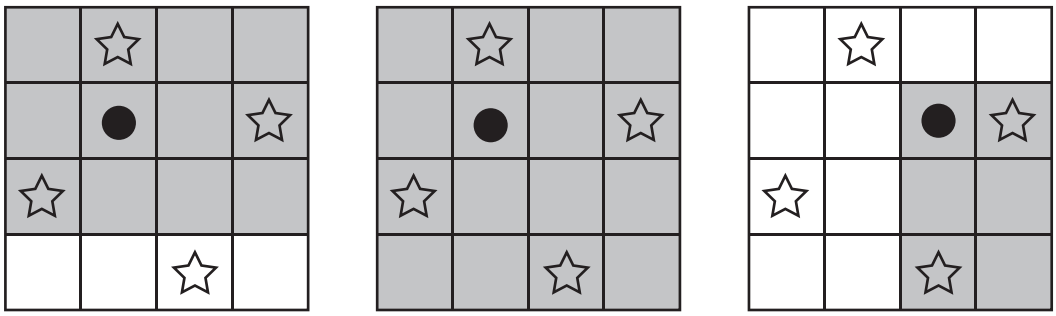
Sample Output

```
1 1
1 1
1 2
```

Explanation

In diagram below, we mark friends' locations with stars, communication rectangles with grey rectangles, and the ideal location for your house as a black circle.

In the first case, on the left, moving your house to any square results in a greater sum of Manhattan lengths to your friends. In the middle case, the interior 4 squares all have the same sum of Manhattan lengths, however you must output the top most, left most square. In the final case, all squares have the same summed Manhattan distance to your friends, but once again you must choose the top most, left most.



Subtasks & Constraints

For all subtasks, $1 \leq N \leq 100\,000$, $1 \leq Q \leq 10\,000$, $0 \leq r_i, c_i, R_1, C_1, R_2, C_2 \leq 1\,000\,000\,000$, $R_1 \leq R_2$ and $C_1 \leq C_2$. Additionally, you are guaranteed there is at least one friend per communication rectangle.

- For Subtask 1 (5 points), $1 \leq N, Q \leq 1\,000$, and $0 \leq r_i, c_i \leq 100\,000$.
- For Subtask 2 (10 points), $0 \leq c_i \leq 100\,000$, and $r_i = 0$ for all of your friends.
- For Subtask 3 (10 points), $0 \leq c_i \leq 1\,000\,000\,000$, and $r_i = 0$ for all of your friends.
- For Subtask 4 (35 points), $0 \leq r_i, c_i \leq 100\,000$ for all of your friends, and $R_1 = C_1 = 0$ for all communication rectangles.
- For Subtask 5 (30 points), $0 \leq r_i, c_i \leq 100\,000$.
- For Subtask 6 (10 points), no further constraints apply.

Scoring

The score for each input scenario will be 100% if a correct answer is written to the output file, and 0% otherwise.

Problem 3
Beacons

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 1 second, 256 MB

With the help of the eccentric Dr. Yes, you managed to save some of your country from the outbreak of *The Virus*[2]. Unfortunately, *The Virus* spread to other countries before you were able to deploy Dr. Yes's vaccine worldwide. Now Dr. Yes is convinced that the only hope for humanity is to attract the attention of extraterrestrial life.

To do so, Dr. Yes has built $2^N - 1$ special beacons in his (flat and very large) backyard. The beacons are indexed from 1 to $2^N - 1$, with the i-th beacon at coordinates (x_i, y_i). No two beacons lie at the same point, and no three beacons lie on a straight line. All that remains now is to activate them by connecting them.

To do this, Dr. Yes needs you to choose some pairs of beacons and join them with straight lengths of wire. To attract the attention of extraterrestials, you must also demonstrate that humans are intelligent, and Dr. Yes has concluded that you must connect the beacons so that a *complete binary tree* is formed with the beacons as nodes and the wires as edges.
Specifically, you must place $2^N - 2$ straight lengths of wire that join beacons such that:

- No two wires intersect at any point except beacons.
- Exactly one beacon has two other beacons connected to it by wire, this is the "root" beacon.
- Exactly 2^{N-1} beacons have only one other beacon connected to it, and their shortest path to the root beacon has precisely $N - 1$ pieces of wire in it. These beacons are the "leaf" beacons.
- All other $2^{N-1} - 2$ non-root, non-leaf beacons must be connected to precisely 3 other beacons.
- The total length of wire used is minimised.

The length of a wire between two beacons at points (x_i, y_i) and (x_j, y_j) is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ (i.e. the Euclidean distance between the two beacons). The total length of wire used is the sum of the lengths of all wires between the chosen pairs of beacons.

You are not required to find the best possible solution. Instead you must simply use the smallest total length of wire you can. Your solution will be compared against the judges' solutions, and better solutions will score more points. See the *Scoring* section for details.

Input

- The first line of input will contain a single integer N, with the number of beacons being $2^N - 1$.
- The following $2^N - 1$ lines of input will each contain two space-separated integers. The i-th of these lines will contain x_i y_i, the location of the beacon with index i.

[2]See FARIO issue #2011, 'The Virus'

Output

Your program must output $2^N - 2$ lines, each containing two integers a_i b_i, indicating that there is a straight length of wire between beacon a_i and beacon b_i. These lines may be in any order, and the ordering of the integers on each line does not matter.

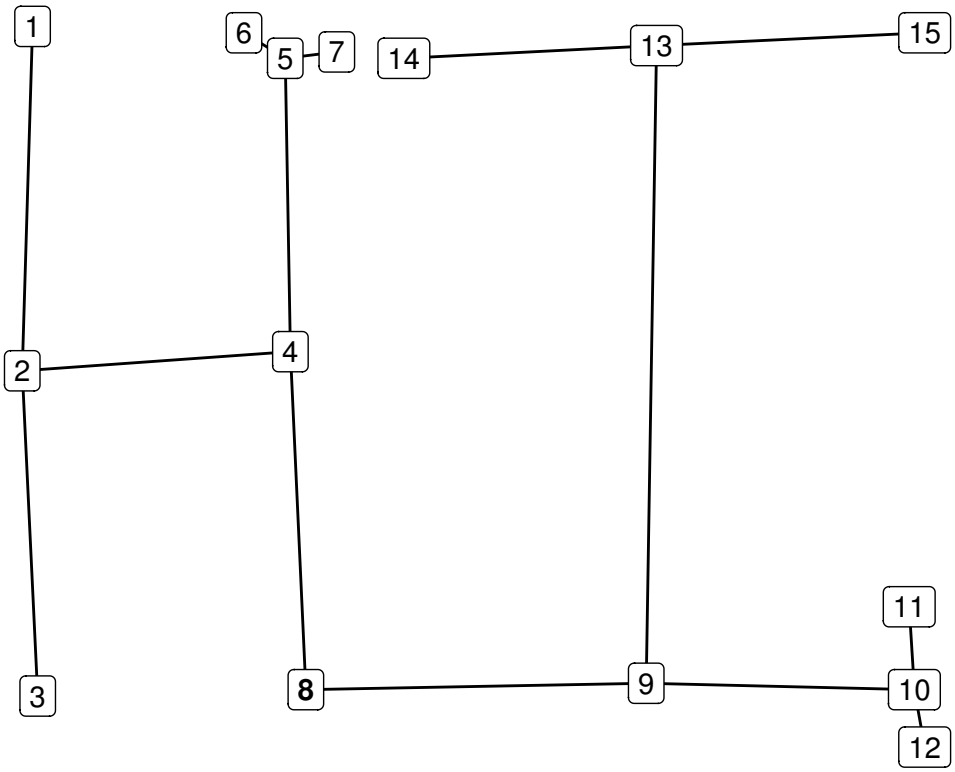
Sample Input

```
4
2 0
0 54
3 105
52 51
51 5
43 1
61 4
55 104
121 103
173 104
172 91
175 113
123 3
74 5
175 1
```

Sample Output

```
8 4
8 9
4 2
4 5
9 13
9 10
2 3
2 1
5 6
5 7
13 14
13 15
10 11
10 12
```

Explanation



We can see that beacon 8 is the root node of the tree, since it is the only node with two edges connected to it. From there, it has children 4 and 9 on its next level. 4 has children 2 and 5, then

2 has children 3 and 1, which have no children of their own. Thus 1 and 3 are 3 levels below the root node 8, and all non-leaf nodes have two children. Similarly all other leaf nodes (6, 7, 14, 15, 11, 12) are 3 levels below the root. This means that the tree we've described is a complete binary tree. Further, by observation it is a valid tree since none of the wires intersect.

Summing up the Euclidean lengths of the wires in the tree, we find that we have used 616.6759 metres of wire.

Scoring

For each test case, your solution will obtain a score determined as follows:

- If your solution does not describe a complete binary tree with no self-intersections, it will obtain a score of 0%.
- Otherwise, let P be the average distance between two beacons multiplied by $2^N - 2$, and let Q be the smallest total length of wire used by the judges' solutions. Your solution will be scored on a linear scale with P scoring 0% and Q scoring 90%, rounded up to at least 30%, and down to at most 100%. In other words your score will be $min(100, max(30, 90 \times \frac{P-X}{P-Q}))$, where X is the total length of wire used by your solution. You are guaranteed that $P > Q$.

For each subtask, your solution will receive the minimum score it obtained for any case in the subtask.

Subtasks & Constraints

For all subtasks, $0 \leq x_i, y_i \leq 1\,000\,000\,000$.

- For Subtask 1 (15 points), $N = 2$
- For Subtask 2 (15 points), $N = 3$
- For Subtask 3 (10 points), $N = 4$
- For Subtask 4 (10 points), $N = 5$
- For Subtask 5 (10 points), $N = 6$
- For Subtask 6 (10 points), $N = 7$
- For Subtask 7 (10 points), $N = 8$
- For Subtask 8 (10 points), $N = 9$
- For Subtask 9 (10 points), $N = 10$