Exam EDAF05

30 Aug 2012, 14.00–19.00

Thore Husfeldt

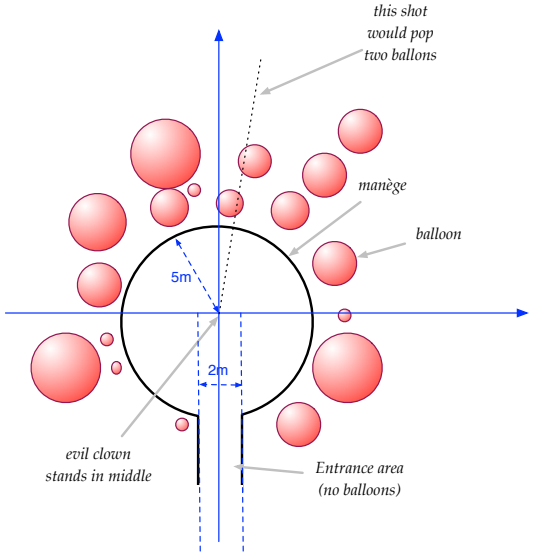
Instructions

What to bring. You can bring any written aid you want. This includes the course book and a dictionary. In fact, these two things are the only aids that make sense, so I recommend you bring them and only them. But if you want to bring other books, notes, print-out of code, old exams, or today's newspaper you can do so. (It won't help.)

You can't bring electronic aids (such as a laptop) or communication devices (such as a mobile phone). If you really want, you can bring an old-fashioned pocket calculator (not one that solves recurrence relations), but I can't see how that would be of any use to you.

Balloons

You are an evil and miserly[1] clown. Standing in the middle of your circus manège, you want to pop all the childrens's balloons, using as few shots of your flower-mounted balloon-popping dart-gun as possible.



Input

The input starts with a line containing n, the number of balloons. The next n lines each contain three integers, giving the position and radius of each balloon (x-coordinate, y-coordinate, radius). The coordinate system has the clown in the origin $(0,0)$, coordinates and balloon radii are in centimeters. No balloon is closer than 5 m to the clown. No balloons are allowed near the 2 m wide entrance (they'd interfere with the giraffes), so there are no balloons intersecting the area $\{\,(x,y) \mid -100 \leq x \leq 100, y < 0\,\}$. Balloons do not overlap (but they can touch).

Output

The number of shots needed to destroy all balloons. Shots travel in a direct line from the origin, popping all ballons they touch. We assume that the children are frozen with fear as you start shooting, so the balloons do not move.

(Don't worry about rounding errors. Evil clowns have no time for numerical analysis.)

Input:
```
3
0 600 10
0 700 5
550 0 50
```

Output:
```
2
```

[1] snål

Shuffle

The string *C* is a *shuffle* of the strings *A* and *B* if it is formed by merging the letters from *A* and *B* in some order. For example, both "evcloilwn" and "evilclown" are shuffles "evil" and "clown". The letters from each word most keep their internal ordering; "evclownli" is not a shuffle of "evil" and "clown".

Note also that there may be many ways of shuffling: "bananaaananas" is a shuffle of "banana" and "ananas" in (at least) three different ways:

```
banana            ban   ana        b  an a  na
      ananas         ana   nas      an  a na  s
```

Input

Three words *A*, *B*, *C*

Output

"yes" if *C* is a shuffle of *A* and *B*, otherwise "no".

Input:
```
banana
ananas
bananaaananas
```

Output:
```
yes
```

Rounding

You are given a list of floating point numbers that you must round to an nearest integer. Each number comes with a precision that tells you how far you are allowed to change the number. For example, if $x = 3.753$ with precision $p = 1.5$ then you are allowed to round x it to 3, 4, or 5. But no to 2, because the distance between x and 2 is larger than p.

Formally, x is to be rounded to an integer m such that $|x - m| \leq p$.

In the resulting list, no integer may appear more than once.

Input

The input starts with the number of numbers n. The following n lines each contain a number x_i and its precision p_i.

Output

A list of distinct integers $m_1, \ldots, m_n$ such that $|x_i - m_i| \leq p_i$ for all i.

If this cannot be done, print "`impossible`."

Input:
```
5
1.5 2
1.5 2
1.5 4.743
2.3 0.5
```

Output:
```
1 0 -1 2
```

Input:
```
1
1.5 0.1
```

Output:
```
impossible
```

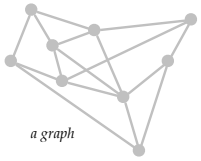
Input:
```
3
1.5 1
1.5 1
1.5 1
```

Output:
```
impossible
```

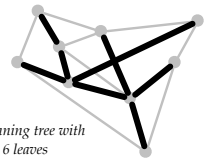
Leaves

Given a graph $G = (V, E)$ and integer k, does G has a spanning tree with exactly k leaves?

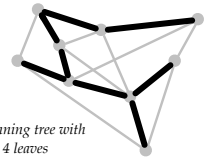
(Recall that a *spanning tree* of G is an acyclic, connected subgraph $T = (V, F)$ where $F \subseteq E$. Recall that a *leaf* is a vertex with only one neighbour in the tree. Note that this exercise does not ask about *minimum* spanning trees; there are no weighted edges or distances in this question.)



a graph



a spanning tree with 6 leaves



a spanning tree with 4 leaves

Input

The first line contains n and m, the number of vertices and edges, respectively. The next line contains k. The following m lines each contain 2 integers describing the endpoints of an edge.

Output

"yes" if G admits a spanning tree with k leaves. "no" otherwise.

Input:

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

Output:

```
no
```

Shrink!

You are given a set of n numbers with 10 digits, including 5,555,555,555. You can change one digit at a time, provided the resulting number is on the list. For instance, for this list ($n = 4$):

```
5755555595
5555555555
5755555555
5543555555
```

you can change your number from 5,555,555,555 to 5,755,555,555, and from there to 5,755,555,595. But you cannot get to 5,543,555,555. What is the numerically smallest number you can get to?

Input

The first line contains n, the number of words The following n lines contain a 10-digit integer each.

Output

The smallest number you can change to.

Input:
```
4
5755555595
5555555555
5755555555
5543555555
```

Output:
```
5555555555
```

Input:
```
5
5755555595
5555555555
5755555555
4755555595
1111111111
```

Output:
```
4755555595
```

Exam Questions

There are five exam questions in this set (on page 7 and 8), corresponding to the five algorithmic problems on pages 2–6. Answer them on a separate piece of paper.

1. One of the problems can be solved greedily.

(a) (*1 pt.*) Which one?

(b) (*4 pt.*) Explain the algorithm. (For example, in pseudocode.) You probably want to process the input in some order; be sure to make it clear *which* order this is (increasing or decreasing order of start time, colour, age, size, x-coordinate, distance, number of neighbours, scariness, etc.) State the running time of your algorithm in terms of the original parameters. (It must be polynomial in the original size.)

2. One of the problems can be efficiently reduced to undirected, unweighted graph connectivity (so it an be solved by something like DFS or BFS, for example):

(a) (*1 pt.*) Which one?

(b) (*4 pt.*) Explain how the connectivity instance is constructed. What are the vertices, and how many are there? What are the edges, and how many are there? Draw an example of a vertex in the connectivity instance, including all vertices it is connected to (you don't have to draw the whole graph.)
Explain which connectivity algorithm you use (BFS? DFS? Is is important?). State the running time of your algorithm in terms of the original parameters. (It must be polynomial in the original size.)

3. One of the problems is solved by dynamic programming. (But not by a simple BFS or DFS search.)

(a) (*1 pt.*) Which one?

(b) (*4 pt.*) Following the book's notation, we let $\mathrm{OPT}(i)$ denote the value of a partial solution. (Maybe you need more than one parameter, like $\mathrm{OPT}(i, j)$. Who knows?) Give a recurrence relation for OPT, including relevant boundary conditions and base cases. State the running time of the resulting algorithm.

4. One of the problems in the set is easily solved by a reduction to network flow.

(a) (*1 pt.*) Which one?

(b) (*3 pt.*) Describe the reduction. Be ridiculously precise about which nodes and arcs there are, how many there are (in terms of size measures of the original problem), how the nodes are connected and directed, and what the capacities are. Do this in general (use words like "every node corresponding to a giraffe is connected to every node corresponding to a letter by an undirected arc of capacity the length of the neck"), and also draw a small, but *complete* example for an example instance. What does a maximum flow mean in terms of the original problem, and what size does it have in terms of the original parameters?

(c) (*1 pt.*) State the running time of the resulting algorithm, be precise about which flow algorithm you use. (Use words like "Using Bellman–Ford (p. 5363 of the textbook), the total running time will be $O(n^{17} \log^{-3} \epsilon + \log^2 m)$.")[2]

5. One of the problems in the set is NP-complete.[3]

(a) (*2 pt.*) Which problem is it? (Let's call it P_1.)

(b) (*2 pt.*) Show that (the decision version) of the P_1 belongs to NP by describing a certificate. In particular, give an example of such a certificate for a small instance. How long is this certificate in terms of the instance size?

[2] This part merely has to be correct. There is no requirement about choosing the cleverest flow algorithm.

[3] If P = NP then *all* these problems are NP-complete. Thus, in order to avoid unproductive (but hypothetically correct) answers from smart alecks, this section assumes that P $\neq$ NP.

(c) (*1 pt.*) Describe very briefly how your certificate can be checked. In particular, what is the running time of that procedure?

(d) (*1 pt.*) The easiest way to show that P_1 is NP-hard is to consider another NP-hard problem (called P_2). Which one?

(e) (*1 pt.*) Do you now need to prove $P_1 \leq_P P_2$ or $P_2 \leq_P P_1$?

(f) (*4 pt.*) Describe the reduction. Do this both in general and for a small but complete example. In particular, be ridiculously precise about what instance is *given*, and what instance is *constructed* by the reduction, the paramters of the instance you produce (for example number of vertices, edges, sets, colors) in terms of the parameters of the original instance, what the solution of the transformed instance means in terms of the original instance, etc.