Exam SAD1

23 Jan 2013

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.

You can answer the questions in Danish or English. Be neat and tidy. Most questions can be answered by a single word or sentence, some require a few sentences, and some require a well-chosen drawing or example.

CheapTravel

You are given a list of connections between airports together with the airfare[1]. You have to find the cheapest way of getting from ATL (Atlanta) to LAX (Los Angeles).

Input

The input contains n lines. Every line describes a connection from one airport to another, followed by the price (an integer, in Euro). The price is the same in each direction, so the line "`JFK MCO 354`" means that there is a direct flight from from JFK to MCO for 354 Euro, and also a direct flight from MCO to JFK for 354 Euro.

You can assume that ATL and LAX appear somewhere on the list of airports.

Output

The price of the cheapest way to get from ATL to LAX. You do not care about how often you need to change planes, or how long you have to wait to make a connection.

Input:

```
JFK MCO 354
ORD DEN 354
ORD HOU 653
DFW PHX 154
JFK ATL 634
ORD DFW 188
ORD PHX 201
ATL HOU 254
DEN PHX 91
PHX LAX 1235
JFK ORD 314
DEN LAS 89
DFW HOU 543
ORD ATL 150
LAS LAX 141
ATL MCO 231
HOU MCO 154
LAS PHX 824
```

Output:

672

[1] Billetpris

Pioneer

You are a pioneer on the planet of Maximegalon IV and want to buy as many pieces of land area as you can. You have 1000 universal credits.

Input

The input contains *n* lines, one for every piece of land that is available for sale. Each line contains, separated by comma, the name of the piece of land, its area (an integer, in Maximegalonian square kiloinch) and its price (an integer, in universal credits).

Output

A maximally long list of land names that can be bought for 1000 universal credits or less. (You don't care about the total area of your land. You don't care about the ordering of the list of names. If there is more than one solution, any of them will serve.)

Input:

```
Grumpfnoodel Valley, 201, 953
Pakka Vale, 13, 169
Greater Shroomphia, 54, 2916
North Koota, 32, 542
South Koota, 53, 143
Wallabingo, 32, 575
Eastern Plains of Moo, 143, 335
Northern Plains of Moo, 321, 6431
Hnkf2rrr, 123, 563
```

Output:

```
Pakka Vale
South Koota
Eastern Plains of Moo
```

Area

Just like in the Pioneer question, you want to buy land on Maximegalon IV. You have C universal credits at your disposal. This time, you're interested in maximising the total area of the land you buy.

Input

A first line containing C, followed by a list of land exactly like for Pioneer.

Output

A list of land names that can be bought for C universal credits or less, of maximal total area. (If there is more than one solution, any of them will serve.)

Input:
```
1000
Grumpfnoodel Valley, 201, 953
Pakka Vale, 13, 169
Greater Shroomphia, 54, 2916
North Koota, 32, 542
South Koota, 53, 143
Wallabingo, 32, 575
Eastern Plains of Moo, 143, 335
Northern Plains of Moo, 321, 6431
Hnkf2rrr, 123, 563
```
Output:
```
Eastern Plains of Moo
Hnkf2rrr
```

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 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. (That's what makes it difficult.)

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
```

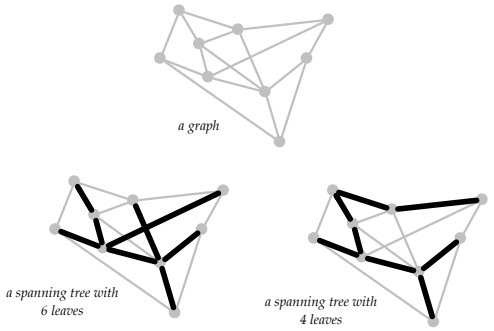
Output:
```
impossible
```

(*Please* make sure you understand these examples, in particular the third one.)

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. The point of this exercise is *not* to fool you into recognizing the MST problem and solve it by Kruskal's or Prim's algorithm. That won't work.)



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
```

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.

(a) (*1 pt.*) What is the running time of the following piece of code in terms of n and m?

```
for i = 1 to n:
    j = 1
    while j < m:
        print j
        j = j + 1
    endwhile
endfor
```

(b) (*1 pt.*) Give a recurrence relation for the running time of the following method as a function of n. (Don't solve the recurrence.)

```
function f(int n):
   if (n < 1) then return 5
   else return f(n-1) * f(n-2)
```

2. One of the problems in the set can be solved greedily.

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

(b) (*2 pt.*) Describe the algorithm, for example by writing it in pseudocode. (Ignore parsing the input.) 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, alphabetic, colour, age, size, x-coordinate, distance, number of neighbours, scariness, etc.). In other words, don't just write "sort the input."

(c) (*1 pt.*) 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 can be efficiently reduced to a shortest path or connectivity problem in graphs.

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

(b) (*2 pt.*) Explain how the graph looks: What are the vertices, and how many are there? What are the edges, and how many are there? Is the graph directed? Are there weigths on the edges? Draw a small example instance.

(c) (*1 pt.*) Briefly explain which algorithm you use (BFS? DFS? Dijkstra? Is is important?).

(d) (*1 pt.*) State the running time of your algorithm in terms of the original parameters. (It must be polynomial in the original size.)

4. One of the problems is solved by dynamic programming.

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

(b) (*3 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.

(c) (*1 pt.*) State the running time and space of the resulting algorithm.

5. 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]

6. We will show that Leaves belongs to NP by describing a certificate.

(a) (*1 pt.*) Is Leaves a decision problem? Answer "yes" or "no". If "no", describe the decision version of Leaves: what are the inputs, what are the outputs?

(b) (*1 pt.*) Describe a certificate for Leaves. In particular, give an example of such a certificate for a small instance. How long is this certificate in terms of the instance size?

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

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

(a) (*1 pt.*) Which problem is it? (Let's call it P_1.)

(b) (*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?

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

(d) (*3 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 parameters 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.

[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.

Exam SAD1

4 Apr 2013

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.

You can answer the questions in Danish or English. Be neat and tidy. Most questions can be answered by a single word or sentence, some require a few sentences, and some require a well-chosen drawing or example.

Zompf

The money changers on the planet Zompf are not very bright. A number of n coins are laid out in a line in front of you. The money changer lets you buy any coin for 1 Earth dollar each, no matter their value. You want to make as much money as you can. You have k Earth dollars.

The official currency rate is 87 Zompf dollars = 1 Earth dollar. Zompf coins come in all kinds of crazy values, but are always a positive integer of Zompf dollars.

Input

The first input line contains the value of k (an integer). On the next line are n integers $z_1, \ldots, z_n$, one for each Zompf coin.

Output

The indices of the coins you pick.

Input:
```
5
100 50 100 50 150 125 25 5 1 75
```

Output:
```
1 3 5 6
```

(Note that in this example I did not pick the last coin (75). This is because I would lose money by changing 1 Earth dollar for 75 Zompf dollars, so it's better to not change.)

Neighbours

A number of n coins are laid out in a line in front of you. You can take as many coins as you want, except that you may never take two neighbouring coins.

Coins come in all kinds of crazy values, but always positive integers.

Input

The input consists of one line of n integers $c_1, \ldots, c_n$, one for each coin.

Output

The indices of the coins you pick.

Input:
```
5
100 50 100 50 150 125 25 5 1 75
```

Output:
```
1 3 5 7 10
```

(In particular, note that in this example I didn't get to take the attractive 6th coin (with value 125).)

Stacks

A number of n coins are laid out in a line in front of you. You want to put the coins into two stacks, called left and right, of the same total value. Coins come in all kinds of crazy values, but always positive integers.

Input

The input consists of one line of n integers $c_1, \ldots, c_n$, one for each coin.

Output

If there is a solution, print the indices of the coins you pick for the left stack. Otherwise print `impossible`.

Input:
```
5
100 50 100 50 150 125 25 5 1 75
```

Output:
impossible

Input:
```
5
100 50 100 50 150 125 25 24 1 75
```

Output:
1 3 5

(In particular, note that in the second example, the left stack contains $100 + 100 + 150 = 350$ and the right stack contains $50 + 50 + 125 + 25 + 24 + 1 + 75 = 350$.)

Crisis

In the image below, you can see that if the Royal Bank of Scotland fails, then so does Sumitomo, a large Japanese *keiretsu* (group of businesses). Such financial failures propagate, so if (for example) Goldman Sachs fails then Bank of America fails as well, which implies that Mitsubishi UFJ and Mediabanca fail, too. And so on; potentially, a single failure can get the whole system to collapse.

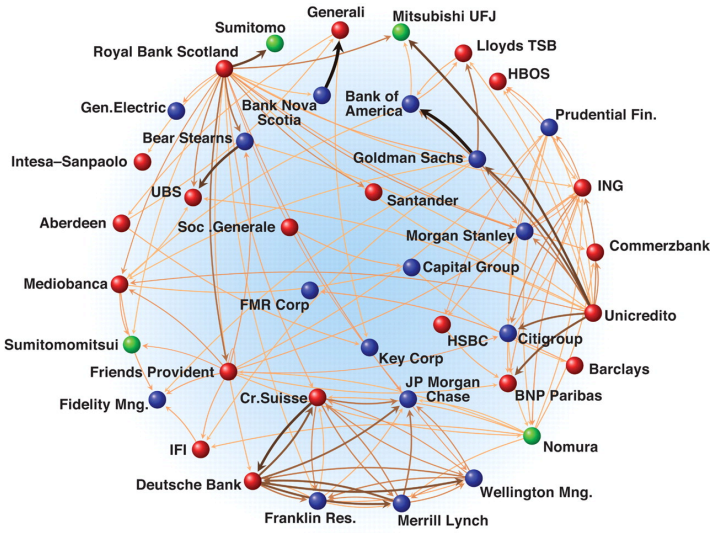


Figure 1: A network of financial institutions. Arrows show dependencies. (Ignore the colours on the arcs and nodes. I have no idea what they mean, I just grabbed this figure off the internet.)

You task is to find out if a given company can bring down the whole system.

Input

The first line of input contains the name X of a financial institution.

Then follow m lines. Every line contains two names, separated by `->`. If the left name fails, so does the right one.

Output

Print `yes` if the failure of X implies that *all* the other institutions in the input fail. Otherwise print `no`.

Input:

```
Bank Nova Scotia
Royal Bank Scotland -> Sumitomo
Royal Bank Scotland -> Gen.  Electric
Aberdeen -> Generali
(...)
Merill Lynch -> Deutsche Bank
```

Output:

```
no
```

Lockout

All schools are closed because *Kommunernes Landsforening* has lock-outed all teachers. You are the typical Scandinavian family. Both parents work full time, so children need to be taken care of by various aunts, uncles, grandparents, etc. Since your children come from various previous complicated relationships and marriages, it's a difficult puzzle.

Given a list of potential hosts, you need to get all your children cared for.

Input

The input starts with the number n, followed by n lines of children names. The next line contains the number m. The following m lines contain the name of a host, and (in parentheses) the number of children that the host has room for. The following lines contain the family relationships. They are of the form `Lisbeth -- Tante Clara`. This means that Lisbeth could potentially visit Tante Clara (given that there is room).

Output

A list of entries of the form `Lisbeth -- Tante Clara`, meaning that Tante Clara takes care of Lisbeth. Each child needs to be taken care of. Otherwise, print `Nintendo`.

Input:

```
3
Peter
Lisbeth
Kurt
2
Tante Clara (2)
Mormor og Morfar (2)
Peter -- Tante Clara
Lisbeth -- Tante Clara
Kurt -- Tante Clara
Lisbeth -- Mormor og Morfar
Kurt -- Mormor og Morfar
```

Output:

```
Peter -- Tante Clara
Lisbeth -- Tante Clara
Kurt -- Mormor og Morfar
```

In particular, note that Tante Clara couldn't take all three children (even though she likes all of them), because she has only room for 2.

Exam Questions

1. One of the problems in the set can be solved greedily.
(a) (*1 pt.*) Which one?
(b) (*2 pt.*) Describe the algorithm, for example by writing it in pseudocode. (Ignore parsing the input.) 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, alphabetic, colour, age, size, x-coordinate, distance, number of neighbours, scariness, etc.). In other words, don't just write "sort the input."
(c) (*1 pt.*) 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 graph connectivity problem.
(a) (*1 pt.*) Which one?
(b) (*2 pt.*) Explain how the graph looks: What are the vertices, and how many are there? What are the edges, and how many are there? Is the graph directed? Are there weigths on the edges? Draw a small example instance.
(c) (*1 pt.*) Briefly explain which algorithm you use (BFS? DFS? Dijkstra? Prim? Something else? Is is important?).
(d) (*1 pt.*) 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.
(a) (*1 pt.*) Which one?
(b) (*3 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.
(c) (*1 pt.*) State the running time and space 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 $O(n^{17} \log^{-3} \epsilon + \log^2 m)$.")[1]

5. We will show that Lockout belongs to NP by describing a certificate.
(a) (*1 pt.*) Is Lockout a decision problem? Answer "yes" or "no". If "no", describe the decision version of Lockout: what are the inputs, what are the outputs?
(b) (*1 pt.*) Describe a certificate for Lockout. In particular, give an example of such a certificate for a small instance. How long is this certificate in terms of the instance size?
(c) (*1 pt.*) Describe very briefly how your certificate can be checked. In particular, what is the running time of that procedure?

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

6. One of the problems in the set is NP-complete.[2]

(a) (*1 pt.*) Which problem is it? (Let's call it P_1.)

(b) (*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?

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

(d) (*3 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 parameters 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.

[2] 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.

Exam SALD1

17 Oct 2013

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.

You can answer the questions in Danish or English. Be neat and tidy. Most questions can be answered by a single word or sentence, some require a few sentences, and some require a well-chosen drawing or example.

Compression

The social network *FaceBook* is an undirected graph of n vertices $v_1, \ldots, v_n$ (called users), whose edges model a binary *friends* relation, so we say that v_i and v_j are friends if $\{v_i, v_j\}$ is an edge in F. Friendship is mutual.

The FaceBook Corporation is running out of server space. It needs to remove as many edges as possible. However, the connectivity of the FaceBook network should stay the same in the following sense: If there was a path from v_i to v_j originally, then there still should be a path from v_i to v_j in the compressed network (the new path may be different.)

Input

The input consists of n lines. Line i starts with the name of user v_i, followed by colon, followed by a comma-separated list of v_i's friends (possibly empty).

Output

A longest sequence of friendships that can be removed from the network such that connectivity is preserved.

Input:

```
Alice:  Bob, Charlie, Dory
Bob:  Alice, Charlie, Dory
Charlie:  Alice, Bob, Dory
Dory:  Alice, Bob, Charlie, Eric
Eric:  Dory
Fred:  Gina
Gina:  Fred
Hans:
```

Output:

Alice–Dory
Alice–Charlie
Bob–Charlie

Loveletter

(FaceBook is defined in the Compression problem.)

Romeo and Julia love each other, but due to pressure from their families they are not allowed to be FaceBook friends.

Romeo wants to send Julia a love letter. The messaging system of FaceBook only allows you to send messages to your friends, so Romeo can't send the letter directly. Instead, Romeo will split the letter into two encryped halves, and send each half on separate paths through the FaceBook network.

(Why exactly two letters are better than one is explained in figure 1 below. It's a simple (but cute) cryptographic idea that has nothing to do with the exercise, so you can safely ignore it.)

Input

Same as for the Compression problem. You can assume (in this exercise) that `Romeo` and `Juliet` both belong to the network, and aren't friends.

Output

Two sequences of names of FaceBook users, each starting in `Romeo` and ending in `Juliet`. No other person may appear on *both* sequences. If p is adjacent to q on a sequence then p and q must be friends.

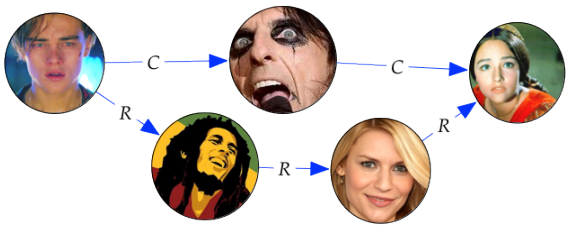
If this can't be done, write `impossible`.

Input:

```
Alice:  Bob, Claire, Eric, Juliet, Romeo
Bob:  Alice, Claire, Romeo
Claire:  Alice, Bob, Juliet
Eric:  Alice, Romeo
Juliet:  Alice, Claire
Romeo:  Alice, Bob, Eric
```

Output:

Romeo, Alice, Juliet
Romeo, Bob, Claire, Juliet



	I	L	O	V	E	Y	O	U
$M =$	8	11	14	21	4	24	14	20
$R =$	7	3	20	18	24	1	11	7
$C =$	1	23	14	12	18	21	5	1

Figure 1: How to transform a message into two messages that are indistinguishable from random noise. M is the message (in fact, the encoding of "ILOVEYOU"). R is a string of random numbers between 0 and 25. C is the ciphertext, $C[i] = M[i] + R[i] \mod 26$. In particular, both C and R *by themselves* are random noise and can be freely sent through the network, provided nobody unwanted receives both of them. Only Juliet, who receives both messages, can recover M (by computing $M[i] = C[i] - R[i] \mod 26$.)

Dinner

(See the Compression problem for a definition of *FaceBook*.)

The FaceBook Corporation arranges a dinner for all its users. There are k tables in the restaurant, of given sizes $r_1, r_2, \ldots, r_k$ with $r_1 + r_2 + \cdots + r_k = n$. Everybody must be seated at some table, and nobody should sit next to anybody they're not friends with.[1]

Input

The first line contains k. The second line contains the integers $r_1, r_2, \ldots, r_k$, separated by commas. The rest of the input is defined like as for the Compression problem.

Output

The output consists of k lines.

The ith line contains a list of r_i names $w_1, \ldots, w_{r_i}$ such that w_j is friends with w_{j+1} for each $1 \leq j < r_i$ and w_{r_i} is friends with w_1. Each FaceBook user must appear exactly once in the output.

If such an arrangement is impossible, output "impossible".

Input:

```
2
3, 3
Alice:  Bob, Claire, Eric, Juliet, Romeo
Bob:  Alice, Claire, Juliet, Romeo
Claire:  Alice, Bob, Juliet
Eric:  Alice, Romeo
Juliet:  Alice, Bob, Claire
Romeo:  Alice, Bob, Eric
```

Output:

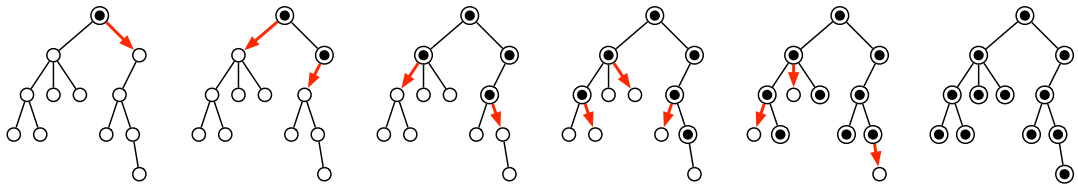
```
Romeo, Eric, Alice
Bob, Claire, Juliet
```

(example corrected 131017, 8:37): Bob–Juliet added

[1] If you want, you can assume that each r_i is always at least 2, so we can avoid arguing about whether a person is friends with themselves. It's not important.

Spread

Suppose we need to distribute a message to all the nodes in a rooted tree of n nodes. Initially, only the root node knows the message. In a single round, every node that knows the message can forward it to at most one of its children. Design an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes.



A message being distributed through a tree in five rounds.

Input

Each line contains two integers from $\{1, \ldots, n\}$ denoting an edge in the tree.

Output

The minimum number of rounds it takes to pass the message.

Input:

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

Output:

```
5
```

Fixpoint

An array $A = (A[0], \cdots, A[n-1])$ of nonnegative integers is a *weak staircase* if the entries are in nondecreasing order, and every entry is at most 1 larger than its predecessor. Formally, for every i with $1 \leq i < n$ we require $A[i] = A[i-1]$ or $A[i] = A[i-1] + 1$. For instance, the arrays $(1,1,1,2,3,3)$, $(1,2,3)$, $(1,1,1)$, and (1) are weak staircases. Let's also define the empty array to be a weak staircase, just to be complete. On the other hand, $(1,2,4)$ is not a weak staircase, and neither is $(4,3,2)$ nor $(-1,0,1)$.

The method

```
public static int fixpoint(int[] A)
```

expects a weak staircase A as input and shall return i such that $A[i] = i$. If no such i exists, return -1.

For instance, on input $A = (2,2,2,3,3)$, the method must return 2 or 3 (both answers are acceptable). On input $A = (2,2,2,2,3)$, the method must return 2. On input $A = (10,11,11,11,12)$, the method must return -1. On input $A = (4,3,2)$, the expected behaviour of the method is unspecified, so it can do what it wants. (It might even crash.)

Implement this method faster than linear time in n.

(example corrected 131017, 9:18): $A = (2,2,2,4,5)$ *changed to* $A = (2,2,2,2,3)$

(clarification 131017, 9:59): Changed the first example to avoid a possible misunderstanding.

Exam Questions

1. One of the problems in the set can be solved using divide-and-conquer.

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

(b) (*2 pt.*) Describe the algorithm, for example by writing it in pseudocode. Be short and precise.

(c) (*1 pt.*) Formulate a recurrence relation that describes the asymptotic running time of your algorithm. State the running time of your algorithm in terms of the original parameters.

2. One of the problems can be efficiently solved using standard graph traversal methods (such as the ones used for breadth-first search, depth-first search, shortest paths, connected components, MST, etc.), or a simple greedy algorithm, but without using more advanced design paradigms such as dynamic programming or network flows.

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

(b) (*2 pt.*) Describe your algorithm, for example in pseudocode, or by referring to existing algorithms from the course book.

(c) (*1 pt.*) State the running time of your algorithm.

3. One of the problems is solved by dynamic programming.

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

(b) (*3 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.

(c) (*1 pt.*) State the running time and space 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. We will show that the decision version of Spread belongs to NP.

(a) (*1 pt.*) Formulate Spread as a decision problem: what are the inputs, what is the output?

(b) (*1 pt.*) Describe a certificate for Spread. In particular, give an example of such a certificate for a small instance. How long is this certificate in terms of the instance size?

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

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

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

(a) (*1 pt.*) Which problem is it? (Let's call it P_1.)

(b) (*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?

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

(d) (*3 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 parameters 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.

[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.

Exam SALD1

27 Feb 2014

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.

You can answer the questions in Swedish or English. Be neat and tidy. Most questions can be answered by a single word or sentence, some require a few sentences, and some require a well-chosen drawing or example.

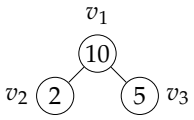
Zero

Recall that a *perfect binary* tree is a binary tree where all leaves are at the same depth and all internal nodes have exactly 2 children. Such a tree has $n = 2^k - 1$ nodes. To fix notation we assume that the nodes are numbered $v_1, \ldots, v_n$ from top to bottom and left to right. (In particular, the children of v_i are v_{2i} and v_{2i+1}.)

Given a perfect binary tree where each node v contains an integer $f(v)$, we want to find a leaf containing the value 0, if such a leaf exists. The tree is assumed to satisfy the following:

Product rule: *Let v_i be an internal node with children v_{2i} and v_{2i+1}. Then $f(v_i) = f(v_{2i}) \cdot f(v_{2i+1})$.*

Here is a small example:



We can represent such a tree as an array of $n + 1$ integers $A = [0, f(v_1), f(v_2), \ldots, f(v_n)]$. (The initial 0 is only there to avoid annoying index problems because arrays are traditionally numbered from 0. With this representation we have $A[i] = f(v_i)$ for each i with $1 \leq i \leq n$, which is nice.) For instance, the tree in the above example is represented by the array $[0, 10, 2, 5]$. We assume $n \geq 3$.

Write a method

```
int zeroleaf(int[] A)
```

that takes as input the representation of a perfect binary tree and returns a leaf with value 0, *i.e.*, an integer i with $n/2 < i \leq n$ such that $A[i] = 0$. If (and only if) no such i exists, then the method must return 0. For instance, `zeroleaf`$[0, 10, 2, 5]$ returns 0, `zeroleaf`$[0, 0, 0, 5]$ returns 2, and `zeroleaf`$[0, 0, 0, 0]$ could return either 2 or 3 (but not 0 or 1).

It is trivial to solve this problem in linear time. The task is to solve it *asympotically faster than linear time*.

Words

You illegally downloaded a book from a disreputable source. Unfortunately, all the punctuation has been removed, and now it looks like this:

```
thehobbitorthereandbackagaininaholeinthegroundtherelivedahobbit...
```

The text consists of n characters. You have at your disposal a constant-time method

```
boolean check(String w)
```

that checks if its argument string `w` is a valid word. For instance, `check("hobbit")` and `check("bit")` both return `true`, but `check("dahob")` returns `false`.

You want to reconstruct the original book.[1]

You can make no assumptions about valid words. (In particular, you cannot assume things like "No word is ever longer than 10 characters" or "No word includes the same four letters in a row" or "No word begins with an X.")

Input

A string S of n letters.

Output

A sequence of words $w_1, w_2, \ldots, w_k$, one on every line, with `check(`w_i`)=true` for each i with $1 \leq i \leq n$ and such that $S = w_1 w_2 \cdots w_k$.

Input:
```
thehobbitorthereandbackagaininaholeinthegroundtherelivedahobbit
```

Output:
```
the
hobbit
or
there
and
back
a
gain
in
a
hole
in
the
ground
the
relived
a
hobbit
```

[1] Strictly speaking, that's impossible, because you can never know if "`therebound`" came from "`the rebound`" or "`there bound`". So let's just say you want to construct a sequence of existing words that are consistent with the input.

Tables

The Superhappy Fun–Fun Corporation arranges a dinner for its n employees. There are 3 tables in the restaurant, of given sizes r_1, r_2, r_3 with $r_1 + r_2 + r_3 = n$. Everybody must be seated at some table, and nobody should sit at the same table as somebody they don't like.

Input

The first line contains the integers r_1, r_2, r_3, separated by commas. The rest of the input contains a line for each person. Each such line starts with the person's name, followed by a colon, followed by a comma-separted list of people he likes.

Output

The output consists of 3 lines.

The ith line contains a list of r_i names $w_1, \ldots, w_{r_i}$ such that w_i is friends with w_j for each i and j with $1 \leq i < j \leq r_i$. Each Employee must appear exactly once in the output.

If such an arrangement is impossible, output "impossible".

Input:

```
4, 1, 1
Alice:  Bob, Claire, Eric, Juliet, Romeo
Bob:  Alice, Claire, Juliet, Romeo
Claire:  Alice, Bob, Juliet
Eric:  Alice, Romeo
Juliet:  Alice, Bob, Claire
Romeo:  Alice, Bob, Eric
```

Output:

```
Alice, Bob, Claire, Juliet
Eric
Romeo
```

Drink

The members of the Edinburgh Single Malt Society have decided to empty their stores to make room for the next season. All bottles with only a few glasses left have to be consumed!

Not every Scotsman likes every whiskey. And even the most hardened members of the Society can drink only limited amounts. Etiquette demands that you cannot share a glass, so you must drink an integer amount of glasses.

Input

The first line contains n and m, the numbers of Society members and whiskeys, respectively.

Then follow n lines, one for each member. Each line contains the name, the number of glasses that person can drink (at most), and the whiskeys he or she likes. Then follow m lines, one for each whiskey. Each line contains the name and the number of glasses left.

Output

Yes or no: can the brave members finish off all the whiskey?

Input:
```
3 4
Connor, 4, Drumguish Laphroaig Teaninich
Sarah, 2, Bruichladdich Drumguish
Iain, 1, Bruichladdich Laphroaig
Bruichladdich, 2
Drumguish, 2
Laphroaig, 2
Teaninich, 1
```

Output:
yes

(For instance, Sarah can finish the 2 glasses of Bruichladdich, Iain helps himself to a glass of Laphroaig, and Connor takes care of the rest.)

Taxi

You are running the interplanetary shuttle service from the planet Maximegalon IV to the planet Huygen's Landing. Your spaceship has m seats and the trip takes t units of time one way. (The trip back takes t units of time as well, but you never bring passengers in that direction.) You know in advance when your passengers arrive at Maximegalon IV. Your task is to finish as fast as possible, *i.e.*, minimize the arrival time of the *last* passenger.

Input

The first line contains three integers, the number of passenger n, the capacity of your spaceship m, and the one-way trip time t (in Galactic hours). Then follow n lines, one for each passenger, giving their earliest departure time (in Galactic hours) from Maximegalon IV.

Output

The earliest time at which all passengers have arrived at Huygen's Landing.

Input:
```
3 2 10
10
30
40
```

Output:
```
50
```

(Do spend some time studying this example. In particular, make sure you understand why the answer is not `60`.)

Exam Questions

1. One of the problems in the set can be solved using divide-and-conquer.

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

(b) (*2 pt.*) Describe the algorithm, for example by writing it in pseudocode.

(c) (*1 pt.*) Formulate a recurrence relation that describes the asymptotic running time of your algorithm. State the running time of your algorithm in terms of the original parameters.

2. One of the problems can be efficiently solved using a greedy algorithm.

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

(b) (*3 pt.*) Describe your algorithm, for example in pseudocode. If you want to sort something, be very precise about what you sort and in which direction. (Use words like "non-increasing age". Better give an example.)

(c) (*1 pt.*) State the running time of your algorithm.

3. One of the problems is solved by dynamic programming.

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

(b) (*3 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.

(c) (*1 pt.*) State the running time and space 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. We will show that Spread belongs to NP.

(a) (*1 pt.*) Is Spread a decision problem? Answer "yes" or "no". If "no", describe the decision version of Spread: what are the inputs, what are the outputs?

(b) (*1 pt.*) Describe a certificate for Spread. In particular, give an example of such a certificate for a small instance. How long is this certificate in terms of the instance size?

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

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

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

(a) (*1 pt.*) Which problem is it? (Let's call it P_1.)

(b) (*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?

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

(d) (*3 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 parameters 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.

[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.