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
```

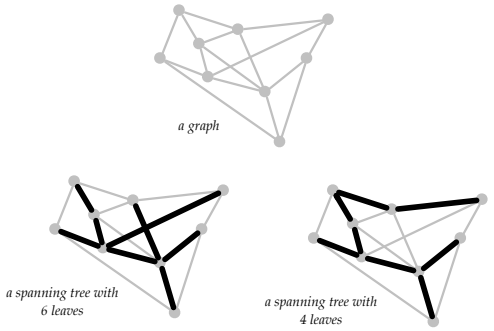
Output:
```
impossible
```

(*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.

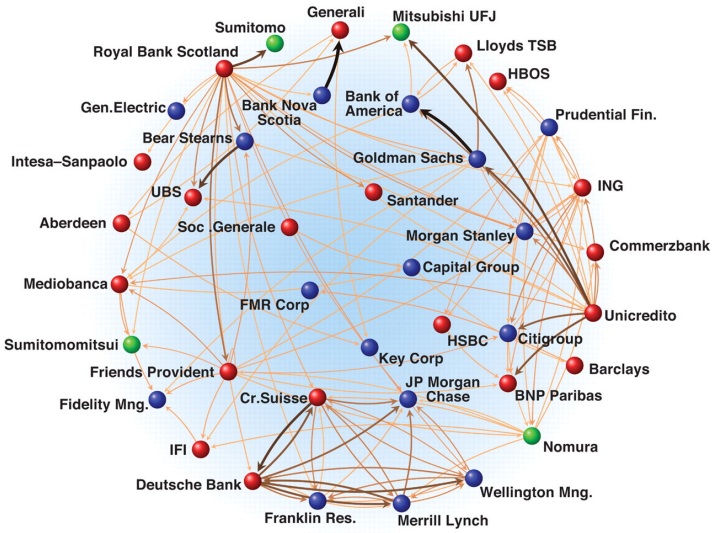


Figure 1: A network of financial institutions. Arrows show dependencies. (Ignore the colours on the arcs and nodes. I have no idea what they mean, I just grabbed this figure off the internet.)

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.

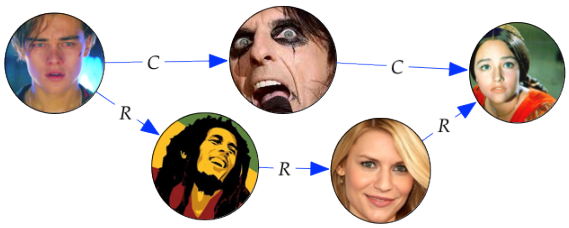
If this can't be done, write `impossible`.

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:

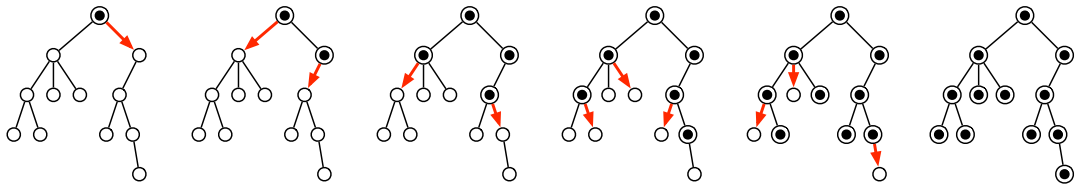
```
Romeo, Eric, Alice
Bob, Claire, Juliet
```

(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.

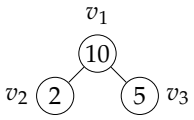
Zero

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.

Exam SAD1

23 October 2014

Thore Husfeldt

Instructions

The exam question are on pages 7 and 8 and relate to the algorithmic problems described on pages 2–6.

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.

Filling out the exam Try to answer with short, clear, and correct answers. Often, pseudocode and small (but complete) examples are a good way of communicating.

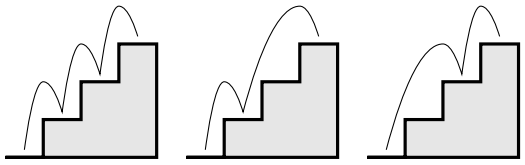
Use English or Danish.

I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. If there is a way to misunderstand what you mean, I will use it.

Count

I have pretty long legs, so when walking up a staircase, I can take 1 or 2 steps at a time. Thus, there are many different ways for me to climb n stairs. Let's call this number $S(n)$.

For instance, if $n = 3$ then I can climb the stairs in 3 different ways:



So $S(3) = 3$. Convince yourself that $S(1) = 1$, $S(2) = 2$, and $S(4) = 5$.

Your task is design an algorithm to implement the function

```
int S(int n)
```

that returns $S(n)$ given n. The algorithms must run in polynomial time in n.

Sweden

You are the head of state in an unnamed Scandinavian country. After the last election, n parties made it into parliament, and your job is to form a government. You country has a tradition for weak minority coalition governments consisting of many small parties, and you've decided to take this to the extreme. Your task is to form a coalition that

1. contains as many parties as possible
2. may not have a majority of seats in parliament (so the sum of the number of seats in the coalition parties has to be less than than half the total number of seats)

The new Prime Minister will be the head of the largest of these parties. (Ties are broken arbitrarily)

Let's agree to use m to denote the total number of seats in parliament. Let's also agree that n is at least 2.

Input

n lines, each containing a party name followed by comma, the number of seats of that party, and the leader of that party.

Output

The name of the new Prime Minister.

Input:
```
Party A, 10, Alice
Party B, 11, Bob
```

Output:
```
Alice
```

Input:
```
Party A, 7, Alice
Party B, 9, Bob
Party C, 14, Claire
Party D, 12, Dora
Party E, 9, Eric
```

Output:
```
Bob
```

Comments on the second example: "Eric" would also be a valid answer. The government consists of A, B, and E, forming a coalition with $7 + 9 + 9 = 25$ seats out of $m = 51$ total seats.

Detour

Given an undirected graph $G = (V, E)$ with n vertices and m edges, and three vertices s, t, w, find a path from s to t passing through w that uses no edge twice.

Comments. Please make sure you understand what this problem is. You are not asked to find a *simple* path. (So you *can* use a vertex twice. But you can't use an edge twice.) Also note that you're not asked to find a *shortest* path.

Input

The edges an undirected graph (endpoints separated by `--`); three of the vertices are named s, t, and w, respectively.

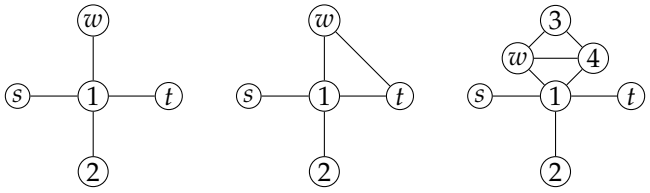
Output

A sequence of vertices describing the path, or the word impossible if no such path exists.

Input:
```
s--1
w--1
t--1
2--1
```
Output:
```
impossible
```

Input:
```
s--1
w--1
t--1
w--t
2--1
```
Output:
```
s 1 w t
```

Input:
```
s--1
w--1
t--1
2--1
w--3
3--4
w--4
1--4
```
Output:
```
s 1 w 3 4 1 t
```



Visitor

Given an undirected graph $G = (V, E)$ with N vertices and m edges, and subset $W \subseteq V$ of vertices, find a simple path in G passing through as many vertices from W as possible.

Input

The edges an undirected graph (endpoints separated by `--`). On the last line, the names of the vertices in W.

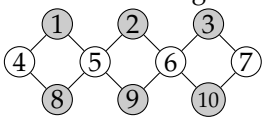
Output

A sequence of vertices describing a simple path in G visiting as many vertices from W as possible.

Input:
```
1--4
1--5
2--5
2--6
3--6
3--7
8--4
8--5
9--5
9--6
10--6
10--7
1 2 3 8 9 10
```

Output:
```
10 7 3 6 9 5 8 4 1
```

Here's a drawing of the above example; the 6 vertices belonging to W are shaded.



Courses

The University of Maximegalon IV has switched to online teaching. Each of its L courses is available online, and you can take it any time you want. Each course comes with with a list of *prerequisite* courses that you must complete before the course. (For instance, you cannot take "Algorithms and data structures" unless you completed "Introduction to programming" and "Discrete mathematics" first.) You want to learn Algorithmic Exobiology, and you want to do as little extra work possible.

Input

The problem input consists of L lines. Each line contains three tab-separated fields for the course code, a full course name, and a list of prerequisites (possibly empty). You can assume that Algorithmic Exobiology is on the list, and that the study administration has ensured that there are no cyclic dependencies.

Prog101	Introduction to programming	
AD	Algorithms and data structures	Prog101, DM
DM	Discrete mathematics	Calc, Alg
Calc	Calculus	
Alg	Algebra	
WA	Weird aliens: how to spot and avoid them	Run
Run	Running and personal fitness	
HR	Human resources and personell planning	
Econ101	Economics 101	Calc, Alg
AlgEx	Algorithmic Exobiology	AD, WA
MBA	Master of Business Administration	Econ101, HR
Lat	Latin	
Kl	Klingon	
Min1	Basic Minecraft	
Min2	Advanced Minecraft	Min1

Output

An ordered list of courses you need to take in that order. For the above example,

Calc, Alg, Prog101, DM, AD, Run, WA, AlgEx

would be a correct solution (there are many others). Your list should only include courses that you actually need, so

Min1, Calc, Alg, Prog101, DM, AD, Run, WA, AlgEx

would not be correct. Please note that the ordering needs to be valid, so

Calc, DM, Alg, Prog101, AD, Run, WA, AlgEx

would not be correct.

Exam Questions

1. Greedy. One of the problems in the set can be solved by a simple greedy algorithm.

(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. Graph traversal. 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, spanning trees, etc.), without using more advanced design paradigms such as dynamic programming or network flows.

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

(b) (*2 pt.*) Explain how. If you construct a graph, describe it (What are the vertices? How many are there? What are the edges? How many are there? Are the edges directed? Are there weigths on the edges?) Draw a small example graph. Describe the algorithm or algorithms that you use in your solution, using precise prose or pseudocode. You are very welcome to refer to existing algorithms from the course book. Don't forget to say expliclty what is output.

(c) (*1 pt.*) State the running time of your algorithm in terms of the paramaters of the original input.

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

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

(b) (*2 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. Flow. 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 $O(r^{17} \log^{-3} \epsilon + \log^2 k)$, where r is the number of froontzes and k denotes the maximal weight of a giraffe.")[1]

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

5. NP. We will show that the decision version of Detour belongs to NP.

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

(b) (*1 pt.*) Describe a certificate for Detour. 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?

6. NP-hard. One of the problems in the set is NP-hard.[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) (*0 pt.*) Convince yourself whether you want to establish $P_1 \leq_P P_2$ 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. For the love of all that is Good and Holy, please start your reduction with words like "Given an instance BLABLA to BLABLA, we will construct an instance of BLABLA as follows."

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

Exam SALD1

26 February 2015, 9–13, 5A14–16

Thore Husfeldt

Instructions

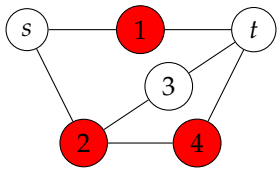
What to bring. You can bring any written aid you want. This includes the course book and a dictionary. In fact, these two things are the only aids that make sense, so I recommend you bring them and only them. But if you want to bring other books, notes, print-out of code, old exams, or today's newspaper 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.

Filling out the exam In those questions where you have to write or draw something, I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. Use English or Swedish. If there is a way to misunderstand what you mean, I will use it.

Avoid

Consider an undirected graph where some of the vertices are *red*. Two non-red vertices are marked s and t, respectively. Your task is to find a simple path from s to t that uses *no* red vertices. If no such path exists, output "impossible." To fix notation: The graph is $G = (V, E)$, we have $n = |V|$ and $m = |E|$ and let W denote the set of red vertices.



Input

First, the edges an undirected graph (endpoints separated by `--`), one edge per line; two of the vertices are named s, and t. Then, the red vertices, one vertex per line.

Output

A sequence of vertices describing the path, or the word "impossible" if no such path exists.

Example

Input:

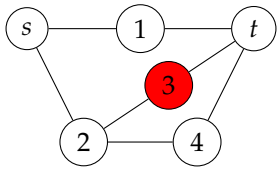
```
s--1
1--t
s--2
2--3
3--t
2--4
4--t
1
2
4
```

Output:

```
impossible
```

Forced

Consider an undirected graph where exactly one vertex is *red*. Two non-red vertices are marked s and t, respectively. You task is to find a simple path from s to t that goes *via* the red vertex. If no such path exists, output "impossible." To fix notation: The graph is $G = (V, E)$, we have $n = |V|$ and $m = |E|$ and let w denote the red vertex.



Input

As for *Avoid*.

Output

As for *Avoid*.

Example

Input:

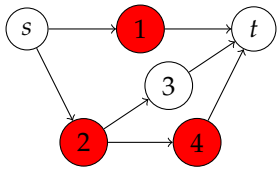
```
s--1
1--t
s--2
2--3
3--t
2--4
4--t
3
```

Output:

```
s 2 3 t
```

MoreDAG

Consider a directed acyclic graph where some of the vertices are *red*. Two non-red vertices are marked s and t, respectively. You task is to find a simple path from s to t that goes via as *many* red vertices as possible. If no such path exists, output "impossible." To fix notation: The graph is $G = (V, E)$, we have $n = |V|$ and $m = |E|$ and let W denote the set of red vertices.



Input

As for *Avoid*, except we use `->` for directed edges.

Output

As for *Avoid*.

Example

Input:

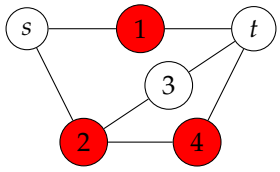
```
s->1
1->t
s->2
2->3
3->t
2->4
4->t
1
2
4
```

Output:

```
s 2 4 t
```

More

Consider an undirected graph where some of the vertices are *red*. Two non-red vertices are marked s and t, respectively. You task is to find a simple path from s to t that goes via as *many* red vertices as possible. If no such path exists, output "impossible." To fix notation: The graph is $G = (V, E)$, we have $n = |V|$ and $m = |E|$ and let W denote the set of red vertices.



Input

As for *Avoid*.

Output

As for *Avoid*.

Example

Input:

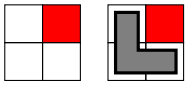
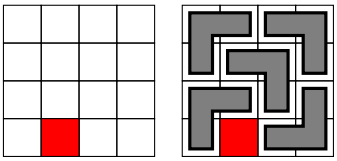
```
s--1
1--t
s--2
2--3
3--t
2--4
4--t
1
2
4
```

Output:

```
s 2 4 t
```

Hole

Consider a board of size $n \times n$ with a single red square at position (x, y). (Let's assume the board squares are $(0, 0)$ in the bottom left corner and $(n-1, n-1)$ at the top right.) You have to cover the non-red squares with $\frac{1}{3}(n^2 - 1)$ L-shaped tiles, without overlaps.

$n = 4, (x, y) = (1, 1)$:  $n = 4, (x, y) = (1, 0)$: 

Input

Three integers: n, x, y

Output

The positions and orientations of the correctly placed tiles, as a list of triples of the form (α, x, y) where $\alpha \in \{0, 90, 180, 270\}$ is a rotation of the basic L tile, and (x, y) determines the position of the 'center' (or 'corner') of the tile. For instance, in the $n = 2$ example above, the rotation is $\alpha = 0$ and the 'center' is at the bottom left corner $(0, 0)$, so the solution to that instance would be written as $(0, 0, 0)$. The $n = 4$ example is shown below. (The ordering of the sequence tuples is not important.) If no solution exists, output "impossible".

Example

Input:
```
4 1 0
```

Output:
```
(270,0,3), (180,3,3), (270,0,1), (90,3,0), (180,2,2)
```

Exam Questions

1. **Graph traversal.** 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, spanning trees, etc.), without using more advanced design paradigms such as dynamic programming or network flows.
   (a) (*1 pt.*) Which one?
   (b) (*2 pt.*) Explain how. If you construct a graph, describe it (What are the vertices? How many are there? What are the edges? How many are there? Are the edges directed? Are there weigths on the edges?) Draw a small example graph. Describe the algorithm you use, using precise prose or pseudocode. You are very welcome to refer to existing algorithms from the course book.
   (c) (*1 pt.*) State the running time of your algorithm in terms of the paramaters of the original input.

2. **Divide-and-conquer.** In this question (and this question *only*) we assume that n is a power of 2. One of the problems can be efficiently solved using a clean divide-and-conquer approach.
   (a) (*1 pt.*) Which one?
   (b) (*2 pt.*) Explain how, using precise prose or pseudocode.[1] Remember to consider the base case.
   (c) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the original input.

3. **Dynamic programming.** 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. **Flow.** 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 $O(r^{17} \log^{-3} \epsilon + \log^2 k)$, where r is the number of froontzes and k denotes the maximal weight of a giraffe.")[2]

5. **NP.** We will consider Avoid as an NP problem.
   (a) (*1 pt.*) Is Avoid a decision problem? Answer "yes" or "no". If "no", describe the decision version of Avoid: what are the inputs, what are the outputs?
   (b) (*1 pt.*) Describe a certificate for Avoid. 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] Because I was bored I actually implemented this. It can be done in 20 lines of python, including input–output.
[2] This part merely has to be correct. There is no requirement about choosing the cleverest flow algorithm.

6. NP-hard. One of the problems in the set is NP-hard.[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) (*0 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. For the love of all that is Good and Holy, please start your reduction with words like "Given an instance to BLABLA, we will construct an instance of BLABLA as follows."

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

Exam SAD1

5 January 2018, 9:00–13:00

Thore Husfeldt

Instructions

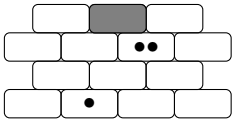
What to bring. You can bring any written aid you want. This includes the course book and a dictionary. In fact, these two things are the only aids that make sense, so I recommend you bring them and only them. But if you want to bring other books, notes, print-out of code, old exams, or today's newspaper 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.

Filling out the exam I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. Short, correct answers are preferred. Each question can be perfectly answered on half a page.

Clean the Wall

The Wall consists of N rows of overlapping bricks, each row alternatingly consisting of N or $N-1$ bricks. To keep it simple, we assume that N is even, so the bottom row has N bricks and the top row as $N-1$ bricks. Here's an example for $N = 4$:



Let's agree to index the positions as pairs of numbers, (i, j), where $i \in \{1, \ldots, N\}$ is the row numbers (starting from the bottom) and $j \in \{1, \ldots, N\}$ is the brick number in that row (starting from the left). A creature called Trump starts at the middle brick in the top row, $(N, N/2)$. He always climbs *down*, from a brick to one of the two connecting bricks below it. To be precise, if i is odd, Trump can go from (i, j) to $(i-1, j-1)$ or to $(i-1, j)$; if i is even, Trump can go from (i, j) to $(i-1, j)$ or to $(i-1, j+1)$. At each position, there is a number of gold coins for Trump to collect, call this number $c(i, j)$. In the picture above, $c(1, 2) = 1$ and $c(3, 3) = 2$. (All other $c(i, j)$ are 0.) Trump's goal is climb all the way down, maximising the amount of collected gold.

Input

First, one even integer N; the number of levels. Then, N lines of $N-1, N, \ldots, N-1, N$ integers, indicating the number of coins available at each position in the natural fashion, top to bottom and left to right.

Output

A sequence of characters `L` (for left) or `R` (for right) that gives a sequence of moves for collecting as many coins as possible.

Example

Input:
```
4
0 0 0
0 0 2 0
0 0 0
0 1 0 0
```

Output:
```
RLL
```

Input:
```
4
0 0 0
0 1 2 0
1 0 0
2 1 0 0
```

Output:
```
LLL
```

The left input file describes the wall shown at the top of the page. In this example, Trump can collect all the coins.

Russian Interference

You have an overview of the Internet, which we (in this exercise) view as a collection of routers connected by wires.[1] Here's a picture of the very early internet, when there were only four routers:

You want to help Russian intelligence to install as many *hacking devices* at the routers as possible. There are two rules:

1. Because of strange signal interference, you can never put two hacking devices at the end of the same wire, nor at the same router. The system would simply crash if you did that.
2. Hacking router i costs $c(i)$ dollars.

You have a total budget of B dollars.

Input

On the first line, the number R of routers, the number W of wires, and your budget B. Then, R lines; the ith line contains the name of router i and its hacking cost $c(i)$. Then, the endpoints of each wire (strings separated by `--`) on separate lines.

Output

An integer K, followed by colon, followed by a comma-separated list of K different router names that tell Russia where to install K hacking devices so that no wire has a hacking device at each end and the total cost is at most B. The value of K has to be as large as possible.

Examples

Input:
```
4 4 5
UTAH 3
SRI 1
UCLA 1
UCSB 2
UTAH -- SRI
UCSB -- SRI
UCSB -- UCLA
UCLA -- SRI
```

Output:
```
2:  UTAH, UCSB
```

Input:
```
4 4 1
UTAH 3
SRI 1
UCLA 3
UCSB 3
UTAH -- SRI
UCSB -- SRI
UCSB -- UCLA
UCLA -- SRI
```

Output:
```
1:  SRI
```

[1] The architecture of the real internet today is somewhat more complicated.

Ambassadors II

Your job is to select US Ambassadors to various countries from a list of candidates. (The candidates happen to be your own business associates, good friends, or family members.) Each candidate has *business ties* with some of the countries. Each country must receive exactly one candidate as ambassador. Each candidate i can be ambassador to at most $c(i)$ countries. You want to maximise the number of business ties.

Input

The number N of candidates, the number M of countries. Then follow the names of M countries, one per line. Then follow N lines in the following format: Each of these lines begins with the name of a candidate, followed by $c(i)$, followed by colon `:`, followed by a comma-separated list of countries they have business ties to.

Output

The maximum number of countries that can receive an ambassador with whom they have business ties, and such that candidate i is ambassador to at most $c(i)$ countries for $i \in \{1, \ldots, N\}$.

Example

Input:
```
3 3
Russia
Azerbajdzjan
Saudi Arabia
Rex 1:  Russia
Ivanka 1:  Azerbajdzjan, Russia
Jared 1:  Russia, Azerbajdzjan, Saudi Arabia
```

Output:
```
3
```

Input:
```
4 4
Russia
Azerbajdzjan
Saudi Arabia
North Korea
Rex 2:  Russia, Azerbajdzjan, Saudi Arabia
Ivanka 1:  North Korea
Jared 1:  North Korea
Bannon 1:  North Korea
```

Output:
```
2
```

East–West Relations

Consider a grid of numbers and letters N, S, E, and W. You can walk from a square to any of its 4 neighbours (up, down, left, right), provided it is not empty. You can not walk on N-squares or on S-squares. Walking on a square has a *cost*. To fix notation, we write $c(i, j)$ for the value in row i and column j, where $1 \leq i \leq r$ and $1 \leq j \leq l$. Otherwise, $c(i, j)$ is a letter (or empty). The cost of walking on square (i, j) is $c(i, j)$. Walking on E-squares or W-squares is free.

The task is to walk from exactly one of the `E` to any exactly one of the `W`.

Input

A space-separated sequence of squares, line by line, in the obvious fashion. A full stop `.` is used for the empty squares. Note that the examples here are small, in general you cannot assume that the cost of a square is a one-digit integer. You can assume (although it is not important for the solution) that the N, S, E, W squares form a boundary around the numbered squares, as shown in the two example inputs. You can assume that there at at least one E and W squares and that a solution exists.

Output

The cheapest cost of getting from a square labelled E to a square labelled W.

Example

Input:

```
. N N N N N
W 7 9 8 8 7 5 N . . . N N N
W 2 2 2 1 1 6 6 N N N 5 5 N E
W 1 2 3 2 2 2 2 4 5 5 4 2 5 E
. S S S 3 3 3 2 6 5 4 2 2 2 E
. . . . S S 2 2 2 2 3 7 2 2 E
. . . . . . S 2 3 2 7 7 7 7 E
. . . . . . . S 7 7 7 7 7 7 E
. . . . . . . S 7 7 7 S S 7 E
. . . . . . . . S 7 S . . S
. . . . . . . . . S
```

Output:

```
35
```

Input:

```
. N N N
W 1 1 3 E
W 3 1 1 E
. S S S
```

Output:

```
4
```

Exam Questions

1. Greed. Professor Gecko thinks that he can solve the *Clean the Wall* problem using the following simple greedy algorithm:

> Start with $i = N$ and $j = N/2$. If i is odd, consider $l = c(i-1, j-1)$ and $r = c(i-1, j)$. If i is even, consider $l = c(i-1, j)$ and $r = c(i-1, j+1)$. If $l > r$ then go left, i.e., print `L` and set the current position to $(i-1, j)$ (for i even) or $(i-1, j-1)$ (for i odd). Else go right, i.e., print `R` and set the current position to $(i-1, j+1)$ (for i even) or $(i-1, j)$ (for i odd). Repeat until $i = 1$.

(a) (*1 pt.*) Give an example input where Professor Gecko's algorithm is guaranteed to find optimal solution. (As a drawing. Smaller is better.)

(b) (*1 pt.*) Give an example input where Professor Gecko's algorithm is guaranteed to *not* find an optimal solution. (As a drawing. Smaller is better.)

(c) (*1 pt.*) What is the running time of Professor Gecko's algorithm as a function of N?

2. Graph traversal. 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, spanning trees, etc.), without using more advanced design paradigms such as dynamic programming or network flows.

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

(b) (*2 pt.*) Explain how you model the problem as a graph problem. Start by drawing the graph corresponding to the 2nd example input to the problem (the one to the right). Be sure to include all the vertices and edges, all the weights (if any), and all directions on the edges (if any).

(c) (*1 pt.*) Which algorithm do you use? (Your answer should at least include a clear statement like "I will find a X in this graph using algorithm Y where Z". Here, X is a combinatorial object (minimum spanning tree, connected component, shortest path, maximum matching, etc.), Y is the name of an algorithm (for instance, as a reference to the course book), and Z is something like "the starting node is the second antisnail in the input".) Faster running time is better.

(d) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the original input.

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

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

(b) (*3 pt.*) Following the book's notation, let $\mathrm{OPT}(\cdots)$ denote the value of a partial solution. (Maybe you need more than one parameter, like $\mathrm{OPT}(i, v)$. Who knows? Anyway, tell me what the parameters are—vertices, lengths, etc. And what their range is. Use words like "where $i \in \{1, \ldots, k^2\}$ denotes the length of BLABLA" or "where $v \in R$ is a red vertex".) 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. Flow. One of the problems in the set is easily solved by a reduction to network flow.

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

(b) (*3 pt.*) Explain the reduction. Start by drawing the graph corresponding to the 2nd example input to the problem (the one to the right). 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. Describe the reduction 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"). 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 $O(r^{17} \log^3 \epsilon + \log^2 k)$, where r is the number of froontzes and k denotes the maximal weight of a giraffe.")[2]

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

5. NP-hard. One of the problems in the set is NP-hard.[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) (*0 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. For the love of all that is Good and Holy, please start your reduction with words like "Given an instance to BLABLA, we will construct an instance of BLABLA as follows."

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

Exam Algorithm Design

7 January 2019, 15:00–19:00

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.

Filling out the exam I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. Short, correct answers are preferred. Each question can be perfectly answered on half a page.

Exam Questions

1. **Greedy.** One of the problems in the set can be solved by a simple greedy algorithm.
   (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 input parameters. (It must be polynomial in the input size.)

2. **Graph traversal.** 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, spanning trees, etc.), and without using more advanced design paradigms such as dynamic programming or network flows.
   (a) (*1 pt.*) Which one?
   (b) (*2 pt.*) Explain how you model the problem as a graph problem. Start by drawing the graph corresponding to Sample Input 1. Be sure to include all the vertices and edges, all the weights (if any), and all directions on the edges (if any).
   (c) (*2 pt.*) Describe your algorithm. As much as you can, make use of known algorithms. (For instance, don't re-invent a well-known algorithm. Instead, write something like "I will use Blabla's algorithm [KT, p. 342] to find a blabla in the blabla.")
   (d) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the input.

3. **Dynamic programming.** One of the problems is solved by dynamic programming.
   (a) (*1 pt.*) Which one?
   (b) (*3 pt.*) Following the book's notation, let $\mathrm{OPT}(\cdots)$ denote the value of a partial solution. (Maybe you need more than one parameter, like $\mathrm{OPT}(i, v)$. Who knows? Anyway, tell me what the parameters are—vertices, lengths, etc. And what their range is. Use words like "where $i \in \{1, \ldots, k^2\}$ denotes the length of BLABLA" or "where $v \in R$ is a red vertex".) 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 in terms of the input parameters.

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

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

(b) (*3 pt.*) Explain the reduction. Start by drawing the graph corresponding to Sample Input 1. 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. Describe the reduction 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"). 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 $O(r^{17} \log^3 \epsilon + \log^2 k)$, where r is the number of froontzes and k denotes the maximal weight of a giraffe.")[1]

5. NP-hard. One of the problems in the set is NP-hard.[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?

(c) (*0 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. For the love of all that is Good and Holy, please start your reduction with words like "Given an instance to BLABLA, we will construct an instance of BLABLA as follows."

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

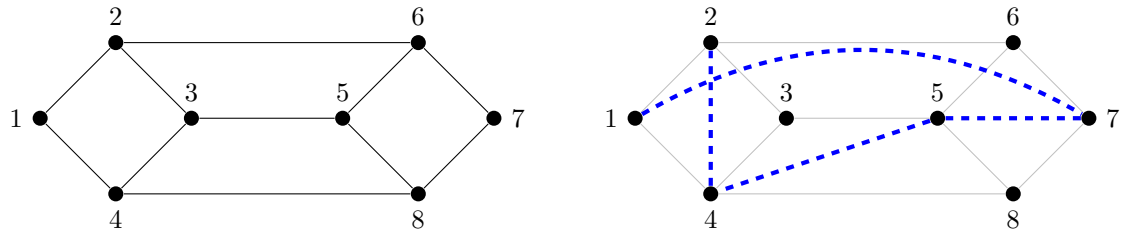
[2] If P = NP then *all* these problems are NP-hard. Thus, in order to avoid unproductive (but hypothetically correct) answers from smart alecks, this section assumes that P $\neq$ NP.

Antipathy

Problem ID: antipathy

Consider an undirected, unweighted graph. An *antipath* is a sequence of distinct vertices $v_1, \ldots, v_k$, such that no consecutive pair in the sequence is joined by an edge in the graph. (Formally, for each $i \in \{1, \ldots, k-1\}$, there can be be no edge between v_i and v_{i+1}.)

The graph below admits an antipath using 5 vertices; an example of such an antipath is shown to the right. This graph corresponds to Sample Input 1 below.



The *Antipathy* problem is given a graph G and integer k to find an antipath of length k, provided such a path exists.

Input

On the first line, three integers n (the number of verices), m (the number of edges), and k.

On the next m lines, the edges of G as pairs of vertex indices; we assume the vertex set is $\{1, \ldots, n\}$.

Output

A sequence of vertex indices of length k describing an antipath. If no such path exists, the word `impossible`.

Sample Input 1

```
8 11 5
1 2
2 3
3 4
4 1
5 6
6 7
7 8
8 5
3 5
4 8
2 6
```

Sample Output 1

```
1 7 5 4 2
```

Sample Input 2

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

Sample Output 2

```
impossible
```

Messy Arithmetic

Problem ID: messyarithmetic

You've finally found your old maths homework from school! Unfortunately, your handwriting when you were a kid was even worse than it is today, and you can't make out the difference between +, −, and $\star$. Also, the exercises and solutions are on separate sheets of paper, and you don't know which exercise corresponds to which solution.

You want to bring these important historical documents back in shape, in case your future biographer needs them when you're famous. Certainly there will be no time for this kind of busywork once you are a famous YouTuber of have won two Nobel prizes in a year.

Match each exercise, which is just a pair of numbers with an unreadable artihmetic operation between them, to a solution, which is also just a number. This requires you to determine the proper artithmetic operation between each pair of numbers.

To avoid having to think about rounding errors, let's assume all numbers are integers. (This is also why we exclude division from this exercise.)

Input

The first line of input consists of the integer n, the number of exercises. The next line contains the solutions $s_1, \ldots, s_n$ as n integers separated by space. Then follow n lines each containing a pair of integers a_i b_i for $i \in \{1, \ldots, n\}$, separated by space.

Output

The output consists of n lines of the form a_i op$_i$ $b_i = s'_i$ for $i \in \{1, \ldots, n\}$. The values a_i and b_i are as given in the input, and in the same order. The operator op$_i$ is one of +, −, $\star$. The set of values $\{s'_1, \ldots, s'_n\}$ is the same set as $\{s_1, \ldots, s_n\}$, but may be in a different order than given in the input.

You can assume that a solution exists. If there is more than one solution, any one of them will do.

Sample Input 1

```
5
0 1 2 3 9
1 2
1 -1
0 5
3 3
5 4
```

Sample Output 1

```
1 + 2 = 3
1 - -1 = 2
0 * 5 = 0
3 * 3 = 9
5 - 4 = 1
```

Sample Input 2

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

Sample Output 2

```
3 - 1 = 2
1 + 1 = 2
3 * 1 = 3
3 * 1 = 3
```

Faroe Islands
Problem ID: faroeislands

Aðalheiður is madly in love with Veturliði. They live in different villages on the Faroe Islands. (Aðalheiður lives in Skælingur, Veturliði lives in Trøllanes.) There are N villages in total. Some of the villages are on the same island and in the same valley—they can be easily reached from each other. (We say that these village form a *community* of villages.) There are C such communities. Other villages are impossible to reach—they are on a different island, or separated by impassable mountains.

Aðalheiður works for the Faroese local government and has a list of M planned infrastructure projects that connect individual villages to each other by tunnels or bridges. When will Aðalheiður finally be able to rach Veturliði?

Input

On the first line, the positive nonzero integers N, C, and M. Among the next C lines, the ith line (for $1 \leq i \leq C$) begins with c_i, the number of villages in the ith community, follwed by the names of the villages in that community, on the same line. Faroese village names contain funny letters but no spaces or hyphens.

Then follow M lines for the planned infrasctructure projects. The lines are sorted by date, and of the form `YYYY-MM-DD:` (for the date), followed by either `Tunnel between` or `Bridge between` followed by the names of two villages separated by `and`, followed by the expected cost in billions of DKK. The cost plays no role.

You can assume that both Skælingur and Trøllanes are part of the input, and that they belong to different communities initially. You can assume $C \leq N$ and $c_1 + \cdots + c_C = N$. You can make no assumption about the reasoning behind the projects. (As an extreme example, they might build a tunnel between a village and itself.)

Output

The earliest date at which Aðalheiður can reach Veturliði. Write `heartbreak` if this will never happen.

Sample Input 1

```
10 3 4
3 Skælingur Ánir Gásadalur
3 Hattarvík Norðradalur Mykines
4 Svínáir Trøllanes Hellur Elduvík
2020-04-12: Tunnel between Ánir and Mykines.         54 bDKK
2021-01-04: Bridge between Gásaldur and Mykines.    214 bDKK
2021-07-01: Tunnel between Hattarvik and Svínáir.     4 bDKK
2021-10-12: Tunnel between Skælingur and Trøllanes.  14 bDKK
```

Sample Output 1

```
2021-07-01
```

Sample Input 2

```
10 3 3
3 Skælingur Ánir Gásadalur
3 Hattarvík Norðradalur Mykines
4 Svínáir Trøllanes Hellur Elduvík
2020-04-12: Tunnel between Ánir and Mykines.         4 bDKK
2021-01-04: Bridge between Gásaldur and Mykines.   154 bDKK
2021-10-12: Tunnel between Skælingur and Ánir.      34 bDKK
```

Sample Output 2

```
heartbreak
```

Some Times
Problem ID: sometimes

Given an integer k and a sequence of n nonnegative numbers $a_1, \ldots, a_n$, where $1 \leq k \leq n$, select at least 1 and at most k numbers whose product is maximal.

For instance, if $k = 4$, $n = 10$, and the numbers are $10, 2, 3, 4, 9, 5, 6, 7, 9, 1$, then the largest product is $10 \cdot 9 \cdot 7 \cdot 9 = 5670$. Note that the numbers can be less than 1, so it doesn't always pay to use k of them. For instance, if $k = 2$ and $n = 2$ and the numbers are $\frac{1}{2}, \frac{1}{2}$ then the largest product is formed by taking just one of them: $\frac{1}{2}$. (If you used both numbers then their product would be $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$, which is smaller.)

These examples also make clear that you can use each of the a_i at most once, but that the a_i do not have to be distinct.

Input

The input consists of 2 lines. The first line contains the integers k and n, separated by space. The second line contains $a_1, \ldots, a_n$, as floating point numbers, separated by space.

Output

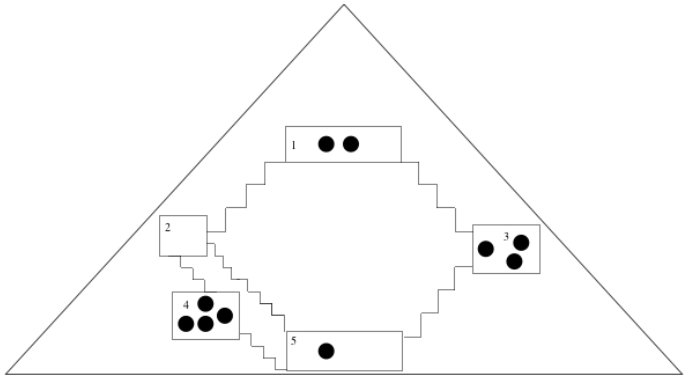
The largest value that can be formed multiplying at least 1 and at most k values from $a_1, \ldots, a_n$. To be precise, output the maximum product of the elements in S over all subsets $S \subseteq \{a_1, \ldots, a_n\}$ with $1 \leq |S| \leq k$.

Sample Input 1	**Sample Output 1**
`4 10` `10 2 3 4 9 5 6 7 9 1`	`5670`

Sample Input 2	**Sample Output 2**
`2 2` `0.5 0.5`	`0.5`

Grave Robber
Problem ID: graverobber

You are planning to rob the graves in the Great Pyramid of Ghis. You have all the virtues of a good criminal, being intelligent, well-prepared, and lazy. The pyramid conists of N rooms connected by staircases. Each room contains a quantity of gold. Each staircase either goes up or down. You have acquired a correct map of the pyramid from a seedy cartographer. You want to plan a path through the pyramid allowing you to collect as much gold as possible—but since you are lazy, you want to only ever go *down*. You start at the top of the pyramid, in room 1, from which all staircases go down.



The picture above corresponds to Sample Input 2.

To fix notation, let us say that $D(i)$ are the rooms adjacent to room i via a *downward* staircase. So if $j \in D(i)$ then there is a staircase from i that goes down to j. Similarly, $U(i)$ are the rooms adjacent to i via an *upward* staircase. You can assume that the pyramid 'makes sense' in that you will never encounter the same room twice if you always walk down. (No magical staircases!) We let g_i denote the amount of gold in room i, which is a nonnegative integer. The pyramid has a single exit, which is in room N. In the example above $D(2) = \{4, 5\}$, $U(3) = \{1\}$, and $g_3 = 3$.

Input

On the first line, the integer N, the number of rooms, and M the number of staircases. Then follows one line consisting of N integers, $g_1, \ldots, g_N$, the amount of gold in the rooms. The next M lines each consist of a pair of integers i and j, meaning that there is a downward staircase from room i to room j.

Output

The maximum amount of gold you can rob from the Great Pyramid of Ghis, starting in room 1, ending in room N, and only ever using downward staircases.

Sample Input 1

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

Sample Output 1

```
4
```

Sample Input 2

```
5 6
2 0 3 4 1
1 2
1 3
2 4
4 5
2 5
3 5
```

Sample Output 2

```
7
```

Exam Algorithm Design

11 March 2019, 9:00–13:00

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.

Filling out the exam I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. Short, correct answers are preferred. Each question can be perfectly answered on half a page.

Exam Questions

1. **Greedy.** One of the problems in the set can be solved by a simple greedy algorithm.
   (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 input parameters. (It must be polynomial in the input size.)

2. **Graph traversal.** 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, spanning trees, etc.), and without using more advanced design paradigms such as dynamic programming or network flows.
   (a) (*1 pt.*) Which one?
   (b) (*2 pt.*) Explain how you model the problem as a graph problem. Start by drawing the graph corresponding to Sample Input 1. Be sure to include all the vertices and edges, all the weights (if any), and all directions on the edges (if any).
   (c) (*3 pt.*) Describe your algorithm. As much as you can, make use of known algorithms. (For instance, don't re-invent a well-known algorithm. Instead, write something like "I will use Blabla's algorithm [KT, p. 342] to find a blabla in the blabla.") *Note*: Briefly motivate your choice: In no more than three sentences, explain if your choice of algorithm was crucial of if you could have used something else. (For instance, "I could have used DFS instead, because we don't care about the blabla." Or "I couldn't have used Prim's algorithm because of the blabla".)
   (d) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the input.

3. **Dynamic programming.** One of the problems is solved by dynamic programming.
   (a) (*1 pt.*) Which one?
   (b) (*3 pt.*) Following the book's notation, let $\mathrm{OPT}(\cdots)$ denote the value of a partial solution. (Maybe you need more than one parameter, like $\mathrm{OPT}(i, v)$. Who knows? Anyway, tell me what the parameters are—vertices, lengths, etc. And what their range is. Use words like "where $i \in \{1, \ldots, k^2\}$ denotes the length of BLABLA" or "where $v \in R$ is a red vertex".) 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 in terms of the input parameters.

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

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

(b) (*3 pt.*) Explain the reduction. Start by drawing the graph corresponding to Sample Input 1. 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. Describe the reduction 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"). 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 $O(r^{17} \log^3 \epsilon + \log^2 k)$, where r is the number of froontzes and k denotes the maximal weight of a giraffe.")[1]

5. NP-hard. One of the problems in the set is NP-hard.[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?

(c) (*0 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. For the love of all that is Good and Holy, please start your reduction with words like "Given an instance to BLABLA, we will construct an instance of BLABLA as follows."

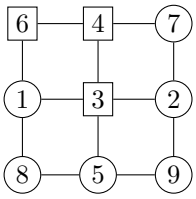
[1] This part merely has to be correct. There is no requirement about choosing the cleverest flow algorithm.

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

All squares

Problem ID: allsquares

The input is an undirected, unweighted graph G with vertex set V. Vertex 1 and 2 are called 'Alice' and 'Bob' and referred to as a and b. At most half of the remaining vertices of V are *special*, let's call them 'squares', and call their set S. We have $S \subseteq V \setminus \{a, b\}$ and $1 \leq |S| \leq \frac{1}{2}(|V| - 2)$.



Your task is to get from Alice to Bob visiting as many squares as possible without repeating vertices. In other words, find a simple path from a to b that includes as many vertices from S as it can.

Input

On the first line, n and m, the number of vertices and edges in the graph, respectively. On the second line, a space-separated list of the vertices in S, we assume $V = \{1, 2, \ldots, n\}$. On each of the remaining m lines, two space-separated integers u, v for the edge between u and v.

Output

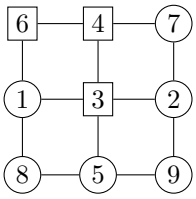
One line of space-separted vertices describing the desired path from a to b. If no path exists from a to b, print `impossible`.

Sample Input 1	Sample Output 1
9 12 3 4 6 1 3 2 3 3 4 3 5 1 6 1 8 5 8 5 9 2 7 4 7 4 6 2 9	1 6 4 3 2

Be a square

Problem ID: beasquare

The input is an undirected, unweighted graph G with vertex set V. Vertex 1 and 2 are called 'Alice' and 'Bob' and referred to as a and b. At most half of the remaining vertices of V are *special*, let's call them 'squares', and call their set S. We have $S \subseteq V \setminus \{a, b\}$ and $1 \leq |S| \leq \frac{1}{2}(|V| - 2)$.



Your task is to find a path from Alice to one of the squares, and a path for Bob to one of the squares. The paths have to be simple and not share any vertices (including their endpoints). In other words, find a pair of vertex-disjoint simple paths, one from a to some vertex in S (it doesn't matter which) and one from b to some vertex in S (it doesn't matter which – except it can't be the same as Alice's).

Input

On the first line, n and m, the number of vertices and edges in the graph, respectively. On the second line, a space-separated list of the vertices in S, we assume $V = \{1, 2, \ldots, n\}$. On each of the remaining m lines, two space-separated integers u, v for the edge between u and v.

Output

Two lines of space separated vertices. The first line describes the desired path from a. The second line describes the desired path from b. If no such pair of paths exists, print `impossible`.

Sample Input 1	**Sample Output 1**
9 12 3 4 6 1 3 2 3 3 4 3 5 1 6 1 8 5 8 5 9 2 7 4 7 4 6 2 9	1 8 5 3 2 7 4

Budget cake

Problem ID: budgetcake

In front of you are n pieces of cake, each with a number s_i of strawberries on it for $i \in \{1, \ldots, n\}$. Some pieces have no strawberries, so s_i could be 0. The price of the ith piece of cake is p_i for $i \in \{1, \ldots, n\}$, where p_i is a nonzero positive integer. To fix notation, set $S = s_1 + \cdots + s_n$ and $P = p_1 + \cdots + p_n$. (Somewhat unrealistically, we assume that cakes are unsplittable, you either eat the whole piece or none of it.)

So many cakes, so little money! You have b units of money. Get as many strawberries as you can, staying within your budget.

Input

On the first line, two integers n and b. On the following n lines, a pair of integers s_i and p_i.

Output

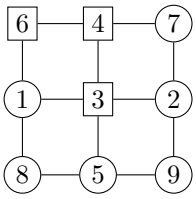
A single line containing the indices of the cakes that you eat. In the sample input, cakes number 2 and 5 give you 13 strawberries for 20 units of money.

Sample Input 1	**Sample Output 1**
5 20 13 17 8 10 1 4 0 1 5 10	2 5

Don't be a square

Problem ID: dontbeasquare

The input is an undirected, unweighted graph G with vertex set V. Vertex 1 and 2 are called 'Alice' and 'Bob' and referred to as a and b. At most half of the remaining vertices of V are *special*, let's call them 'squares', and call their set S. We have $S \subseteq V \setminus \{a, b\}$ and $1 \leq |S| \leq \frac{1}{2}(|V| - 2)$.



Your task is to get from Alice to Bob without visiting any squares. In other words, find a simple path from a to b that includes no vertices from S.

Input

On the first line, n and m, the number of vertices and edges in the graph, respectively. On the second line, a space-separated list of the vertices in S, we assume $V = \{1, 2, \ldots, n\}$. On each of the remaining m lines, two space-separated integers u, v for the edge between u and v.

Output

One line of space-separted vertices describing the desired path from a to b. If no path exists from a to b, print `impossible`.

Sample Input 1	Sample Output 1
9 12 3 4 6 1 3 2 3 3 4 3 5 1 6 1 8 5 8 5 9 2 7 4 7 4 6 2 9	1 8 5 9 2

Strawberries
Problem ID: strawberries

In front of you are n pieces of cake, each with a number s_i of strawberries on it for $i \in \{1, \ldots, n\}$. Some pieces have no strawberries, so s_i could be 0. The price of the ith piece of cake is p_i for $i \in \{1, \ldots, n\}$, where p_i is a nonzero positive integer. To fix notation, set $S = s_1 + \cdots + s_n$ and $P = p_1 + \cdots + p_n$. (Somewhat unrealistically, we assume that cakes are unsplittable, you either eat the whole piece or none of it.)

So many cakes, so little time! You want to eat k pieces of cake getting as many strawberries as you can. (Somewhat unrealistically, each piece of cake takes the same amount of time to eat. Even more unrealistically, you are infintely rich.)

Input

On the first line, two integers n and k. On the following n lines, a pair of integers s_i and p_i.

Output

A single line containing the indices of the cakes that you eat. In the sample input, cakes number 1, 2 and 5 get you $13 + 8 + 5 = 26$ strawberries.

Sample Input 1	Sample Output 1
5 3 13 17 8 10 1 4 0 1 5 10	2 1 5