Exam Algorithm Design

16 December 2020, 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 on pages 3–7 can be efficiently solved using (possibly several applications of) standard graph traversal methods (such as 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.*) Describe your algorithm. As much as you can, make use of known algorithms. (For instance, don't reinvent a well-known algorithm. Instead, write something like "I will use Blabla's algorithm [KT, p. 342] to find a blabla in the blabla.")
   (c) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the input.

3. **Dynamic programming.** One of the problems on pages 3–7 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. Which values of OPT are used to answer the problem?
   (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 on pages 3–7 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 on pages 3–7 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?

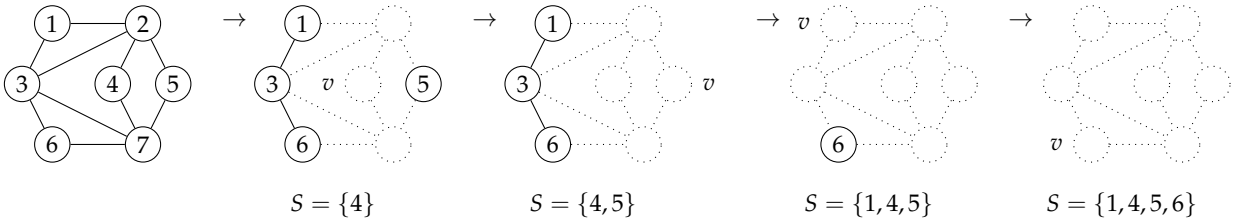
(c) (*0 pt.*) Do you now need to prove $P_1 \leq_P P_2$ or $P_2 \leq_P P_1$?

(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."

6. Professor Gecko. Professor Gecko has come up with a greedy algorithm for the Independent Set problem.[3] His idea is to start with an empty S and repeatedly pick a vertex $v \in V$ of minumum degree (= a vertex with fewest neighbours, ties are broken arbitrarily), add v to S, and remove v and its neighbours (including the incident edges). Repeat until $|S| = k$. To be precise:

$S = \emptyset$
while $|S| < k$ and $V \neq \emptyset$:
    v = a vertex of G with minimum degree
    $S := S \cup \{v\}$
    U = neighbours of v
    remove edges with at least one endpoint in U from E
    remove v and every $u \in U$ from V
if $|S| = k$ **return** "yes" **else return** "no"

He proudly shows you that this works on the graph he found in a textbook, where $k = 4$.



You want to convince him that his algorithm is not correct. To make sure he realises his folly, you should give two different types of arguments.

(a) (*2 pt.*) Argue by counterexample: Draw a concrete instance on which Professor Gecko's algorithm gives the wrong answer. *Hint*: This is not entirely easy. The smallest instance on which Gecko's algorithm *can* fail (because of suboptimal tie breaking) has size 6; that instance is a perfectly valid answer. The smallest instance where it must fail (no matter how ties are broken) has 7 vertices.

(b) (*2 pt.*) Give a complexity-theoretic argument: Explain that *if* Professor Gecko's algorithm were correct then P = NP. Be brief (three sentences is fine), but do include a relevant analysis of the running time of the good professor's algorithm.

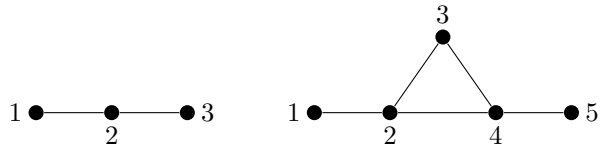
[1] This part merely has to be correct. There is no requirement about choosing the cleverest flow algorithm.

[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.

[3] To fix notation, the Independent Set problem is given an undirected graph $G = (V, E)$ and a number k, decide if G contains an independent set of size k.

Diameter
Problem ID: diameter

Let G be a connected, undirected and unweighted graph. The *diameter* of G is the longest distance between two vertices. Recall that the distance between two vertices is the length of a shortest path between them. Let us agree that the *length* of a path is its number of edges.



In the examples above, the graph on the left has diameter 2 (because vertices 1 and 3 are at distance 2, and no pair of vertices are at distance 3.) The graph on the right has diameter 3. Observe that there is a simple path of length 4 from vertex 1 to vertex 5, but this is not a shortest path and therefore has no bearing on the diameter.

Input

On the first line, the number n of vertices and the number m of edges. On the following m lines, the endpoints u and v of each edge, with $u \neq v$ and $u, v \in \{1, \ldots, n\}$.

The vertices are enumerated as $1, \ldots, n$. You can assume that the input graph is connected.

Output

The diameter of G.

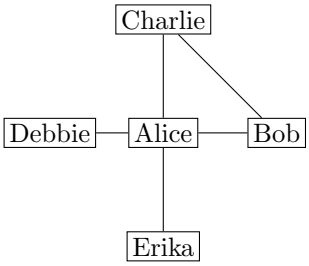
Sample Input 1	Sample Output 1
3 2 1 2 2 3	2

Sample Input 2	Sample Output 2
5 5 1 2 2 3 2 4 3 4 4 5	3

Anti-social Network
Problem ID: antisocialnetwork

The pandemic response of the new government will be much more efficient than the previous. Social networks are the first target. To prevent any kind of contagion, all social networks must be purged (*udrenset*) so that the remaining people are not friends, except if one (or both) of them has a mask. You have t many masks and want to retain (*beholde*) as many people as possible.

The social network uses a symmetric friendship relationship (if u is friends with v then v is friends with u), so we will model it as an undirected, unweighted graph.



In the example, if we have $t = 1$ then we can make Alice wear a mask and retain 4 people, for instance, Alice, Charlie, Erika, and Debbie. An alternative is to give the mask to Charlie and retain Charlie, Bob, Debbie, and Erika. If instead you gave the mask to Debbie then you could only retain 3 people.

Input

On the first line, the number $n > 0$ of people, the number $m > 0$ of friendships, and the number $t \geq 0$ of available masks. Then follow n lines containing the names of the people. Then follow m lines containing a pair u and v of people for each friendship.

You can assume that all people are different, and that nobody is friends with themselves.

Output

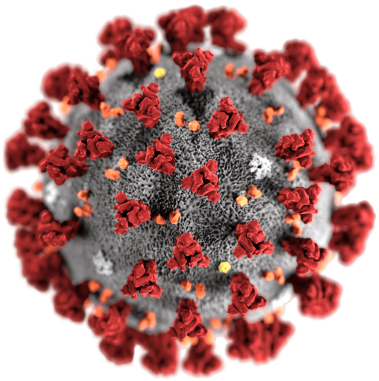
The size of the largest set of people you can retain, followed by the list of at most t users that must wear a mask in your solution.

Sample Input 1	Sample Output 1
5 5 1 Alice Bob Charlie Debbie Erika Alice Bob Alice Charlie Alice Debbie Alice Erika Bob Charlie	4 Alice

Isolation
Problem ID: isolation

You are the supreme leader of a planned economy largely based on exporting animal products. Your comrade the State Epidemiologist has produced a list of which species can directly infect others. You want to protect your loyal subjects from the virus. What is the smallest number of species you need to exterminate?

You cannot destroy the virus nor your loyal subjects.



A coronavirus.

Input

On the first line, T, the number of direct infection possibilities. Each of the following T lines has the form "f `->` t" and means that infection can transmit from species f to species t.

To fix notation, let us agree that the number of different species in the input is S.

Output

A list of species of minimal lengh. (In Sample 3, "`gnus, zebras`" would not have been a correct answer, because it has length 2, whereas "`mink`" has length 1.)

If there is more than one correct answer, any will do. (In Sample 1; both seagulls and pigs would have been valid answers as well.) If there is no solution, write `impossible`

Sample Input 1

```
4
virus -> mink
mink -> seagulls
seagulls -> pigs
pigs -> loyal subjects
```

Sample Output 1

```
mink
```

Sample Input 2

```
6
virus -> dolphin
virus -> mink
dolphin -> shark
shark -> whale
whale -> dolphin
mink -> loyal subjects
```

Sample Output 2

```
mink
```

Sample Input 3

```
5
virus -> mink
mink -> gnus
mink -> zebras
zebras -> loyal subjects
gnus -> loyal subjects
```

Sample Output 3

```
mink
```

Mink Farms
Problem ID: minkfarms


A mink farm. Source: Wikimedia Commons.

You are the Supreme Leader of a planned economy who has decided to get rid of all mink farms. You can issue executive orders that force single farms to close the farm and slaughter all their mink. But be careful! If the total number of resulting dead mink surpasses a certain number k, the country's otherwise docile press and cowed opposition will activate and stop the order before it is executed.

Your job is to close as many farms as possible.

For a small example, assume there are four farms and $k = 1000$. The number of mink on the four farms are 700, 450, 600, and 450 mink, respectively. Then it makes sense to order the two smallest farms to slaughter their livestock (*besætning*), for a total of $450 + 450 = 900$ mink.

Note that you cannot order a farm to slaughter only a fraction of their mink.

Input

On the first line, two integers: the number n of mink farms and the number k of mink that can be slaughtered without activating the docile press and cowed opposition. Then follow n lines, the ith of which contains a single integer m_i: the number of mink in the ith farm for $i \in \{1, \ldots, n\}$.

Output

A single number: the maximum number of mink farms you can close.

Sample Input 1

```
4 1000
700
450
600
450
```

Sample Output 1

```
2
```

Sample Input 2

```
3 1000
1001
1000
999
```

Sample Output 2

```
1
```

Sample Input 3

```
1 15000000
16000000
```

Sample Output 3

```
0
```

Mink Massacre

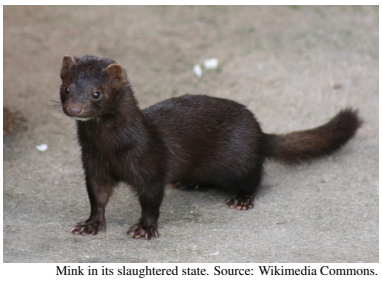
Problem ID: minkmassacre

You are the Supreme Leader of a planned economy who has decided to get rid of mink. You can issue executive orders that force single farms to slaughter all their mink. But be careful! If the total number of resulting dead mink surpasses a certain number k, the country's otherwise docile press and cowed opposition will activate and stop the order before it is executed.

Your job is to slaughter as many mink as possible.

For a small example, assume there are four farms and $k = 1000$. The number of mink on the four farms are 700, 450, 600, and 450 mink, respectively. Then it makes sense to order the two smallest farms to slaughter their livestock (*besætning*), for a total of $450 + 450 = 900$ mink.

Note that you cannot order a farm to slaughter only a fraction of their mink.



Mink in its slaughtered state. Source: Wikimedia Commons.

Input

On the first line, two integers: the number n of mink farms and the number k of mink that can be slaughtered without activating the docile press and cowed opposition. Then follow n lines, the ith of which contains a single integer m_i: the number of mink in the ith farm for $i \in \{1, \ldots, n\}$.

Output

A single number: the maximum number of mink you can slaughter.

Sample Input 1	Sample Output 1
4 1000 700 450 600 450	900

Sample Input 2	Sample Output 2
3 1000 1001 1000 999	1000

Sample Input 3	Sample Output 3
1 15000000 16000000	0

Exam Algorithm Design

21 December 2021, 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).

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 on pages 3–7 can be efficiently solved using (possibly several applications of) standard graph traversal methods (such as 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.*) Describe your algorithm. As much as you can, make use of known algorithms. (For instance, don't reinvent a well-known algorithm. Instead, write something like "I will use Blabla's algorithm [KT, p. 342] to find a blabla in the blabla.")
   (c) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the input.

3. **Dynamic programming.** One of the problems on pages 3–7 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. Which values of OPT are used to answer the problem?
   (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 on pages 3–7 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 on pages 3–7 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."

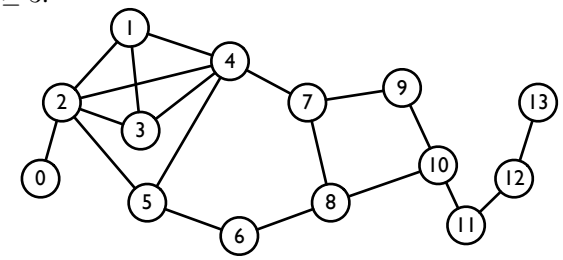
[1] This part merely has to be correct. There is no requirement about choosing the cleverest flow algorithm.

[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.

Big cycle

Problem ID: bigcycle

The input is a connected, undirected, unweighted graph G without self-loops or parallel edges, and an integer $r \geq 3$.



Recall that a *cycle*[1] is a sequence $v_1, \ldots, v_k$ of vertices such that $k \geq 4$, neighbouring vertices v_i and v_{i+1} share an edge for $i \in \{1, \ldots, k-1\}$, the first $k-1$ vertices are distinct, and $v_1 = v_k$. In the example above, the sequence $1, 2, 3, 1$ is a cycle, and so is $2, 4, 7, 8, 6, 5, 2$. The *length* of the cycle is $k-1$ (the number of distinct vertices).

Your task is to find a cycle in the graph of length at least r if it exists.

Input

On the first line, the integers n, m, and r, which are the number of vertices $\{0, \ldots, n-1\}$, the number m of edges in the graph, and the parameter r as introduced above. On the following m lines, two integers u and v with $u, v \in \{0, \ldots, n-1\}$ indicate that there is an edge between vertex u and v.

Output

A sequence of at least $r+1$ vertex names describing a cycle in the graph. (If there are more than one cycle in the graph of sufficient length, any of them is fine.) If no such cycle exists, output *impossible*.

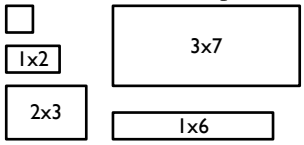
Sample Input 1	Sample Output 1
14 19 5 0 2 1 2 1 3 1 4 2 3 2 4 2 5 3 4 4 5 4 7 5 6 6 8 7 8 7 9 8 10 9 10 10 11 11 12 12 13	6 5 4 7 9 10 8 6

[1] Such a cycle is sometimes called a *simple* cycle

Nesting rectangles

Problem ID: nestingrectangles

Consider some axis-parallel rectangles with integer side lengths.



I want to put rectangles inside of other rectangles. The rule is that I can put an $a \times b$ rectangle inside a $c \times d$ rectangle if either ($a < c$ and $b < d$) or ($a < d$ and $b < c$). In particular, one of the rectangles can be rotated by ninety degrees.

The more rectangles I can nest, the better!

In the above example, I can put 1×6 inside of 3×7, for a nesting depth of 2 rectangles. But I can do better: put the tiny 1×1 square inside of 2×3 and then put those inside 3×7 for a nesting depth of 3. In the example, that's the best I can do. (Note that there is also a different way of achieving nesting depth 3, namely using 1×2, 2×3, and 3×7.)

Some clarifications: Note that I can't put 1×1 inside 1×2; the side lengths have to be *strictly* smaller. Also note that I'm after nesting *depth*, not number of nested rectangles. So even though both 1×1 and 1×2 could be put next to each other inside 3×7 with room to spare, the resulting depth is still only 2.

Input

On the first line, the number n of rectangles. Then follow n lines of the form $a \times b$, where a and b are integers. You can assume $a \leq b$ and that no pair appears twice. (This doesn't make a difference; we can't nest $a \times b$ inside $a \times b$ anyway, so we may as well ignore duplicates.)

Output

The maximum nesting depth.

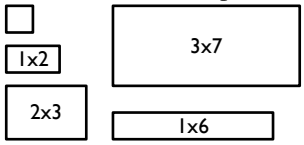
Sample Input 1	Sample Output 1
5 1x1 1x2 1x6 2x3 3x7	3

Sample Input 2	Sample Output 2
3 10x10 2x11 1x10	2

Painting rectangles

Problem ID: paintingrectangles

Consider some axis-parallel rectangles with integer side lengths.



I want to paint them pink, but I only have enough paint for area A.

A rectangle of dimension $a \times b$ requires $a \cdot b$ much paint. I want to have as many pink rectangles as possible; partially painted rectangles don't count.

In the above example, with $A = 3$, I could paint the 1×1 and the 1×2 rectangles. In the above example, with $A = 11$, I could additionally paint the 1×6 rectangle, for a total of 3 many rectangles. (I'd have some spare paint, but that doesn't matter.)

Input

On the first line, the number n of rectangles, and the amount A of paint. Then follow n lines of the form $a \times b$, where a and b are integers. You can assume $a \leq b$.

Output

The maximum number of rectangles I can fully paint with A much paint.

Sample Input 1

```
5 3
1x1
1x2
1x6
2x3
3x7
```

Sample Output 1

```
2
```

Sample Input 2

```
5 11
1x1
1x2
1x6
2x3
3x7
```

Sample Output 2

```
3
```

Sample Input 3

```
3 100
1x1
1x1
1x1
```

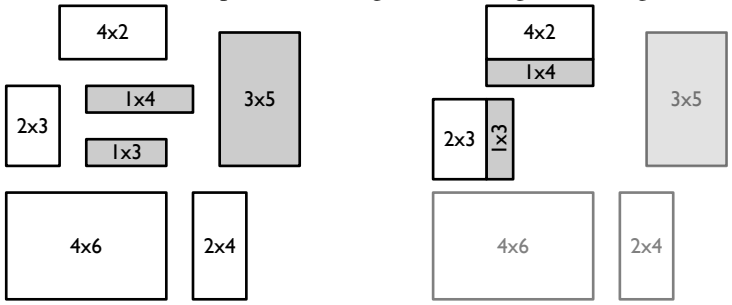
Sample Output 3

```
3
```

Pairing rectangles

Problem ID: pairingrectangles

Consider some axis-parallel rectangles with integer side lengths. Some of them are gray, others white.



I want to pair them up, one from each colour, so that the sides align nicely.

To be precise, a white rectangle of dimension $a \times b$ can be paired up with gray rectangle of dimension $c \times d$ exactly if $a = c$ or $a = d$ or $b = c$ or $b = d$. I want to have as many rectangles paired up as possible.

In the above example, I can pair up 2×3 with 1×3 (the latter is rotated to make it look nice), and 2×4 with 1×4. That gives 2 pairs, which is the best I can do. In particular, I can't pair the two remaining white rectangles (even though they share the side length 4), because I want the pairs to be gray–white. Neither can I use the remaining gray 3×5 rectangle, because no white rectangle has side length 3 or 5.

No rectangle can appear in more than one pair.

Input

On the first line, the numbers g and w of gray and white rectangles, respectively. Then follow $g + w$ lines of the form $a \times b$, where a and b are integers. The first g lines are the gray rectangles, the remaining w the white.

Output

The maximum number of pairs of rectangles that can be formed.

Sample Input 1

```
3 4
1x4
1x3
3x5
4x2
2x3
4x6
2x6
```

Sample Output 1

```
2
```

Sample Input 2

```
1 1
1x1
2x2
```

Sample Output 2

```
0
```

Sample Input 3

```
2 2
1x1
1x1
1x1
1x1
```

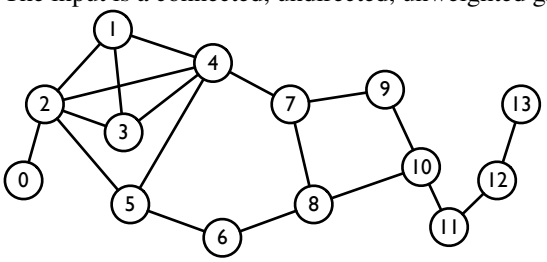
Sample Output 3

```
2
```

Some cycle

Problem ID: somecycle

The input is a connected, undirected, unweighted graph G without self-loops or parallel edges.



Recall that a *cycle*[1] is a sequence $v_1, \ldots, v_k$ of vertices such that $k \geq 4$, neighbouring vertices v_i and v_{i+1} share an edge for $i \in \{1, \ldots, k-1\}$, the first $k-1$ vertices are distinct, and $v_1 = v_k$. The *length* of the cycle is $k-1$ (the number of distinct vertices). In the example above, the sequence $1, 2, 3, 1$ is a cycle of length 3, and $2, 4, 7, 8, 6, 5, 2$ is a cycle of length 6.

The task is to find a cycle in the graph.

Input

On the first line, the integers n and m, which are the number of vertices $\{0, \ldots, n-1\}$ and the number of edges in the graph. On the following m lines, two integers u and v with $u, v \in \{0, \ldots, n-1\}$, such that there is an edge between vertex u and v.

Output

A sequence of at least four vertex names describing a cycle in the graph. (If there are more than one cycle in the graph, any of them is fine.) If the graph contains no cycles at all, output *impossible*.

Sample Input 1	Sample Output 1
14 19 0 2 1 2 1 3 1 4 2 3 2 4 2 5 3 4 4 5 4 7 5 6 6 8 7 8 7 9 8 10 9 10 10 11 11 12 12 13	1 2 3 1

[1] Such a cycle is sometimes called a *simple* cycle

Vegan

Problem ID: vegan

You love you friends dearly, but their dietary restrictions make hosting a party quite the challenge.

Each ingredient comes in a vegan and non-vegan form, and you need to decide which to order. Each of your friends is flexible enough that they accept any one of three different dishes. For instance, Claire would be perfectly happy if you served vegan cake (eggs don't agree with her) or vegan hamburgers (she avoids meat for ethical reasons) or non-vegan cafe lattes (she finds the taste of soy or oat milk disgusting).

You want to accomodate all your friends, so everyone needs to have at least one of their wishes satisfied.

Input

One the first line, the number $f \geq 1$ of friends.

Then follow $4f$ lines, four lines for every friend. The first of these lines is the (unique) name of your friend, and then 3 bulleted lines of what they like to eat. Each of these starts with the word 'vegan' or 'non-vegan', followed by more symbols. (None of these lines need to make culinary sense.)

To fix notation, let's say there are m many different types of food (such as cake), each which can come in two different forms (vegan cake and non-vegan cake).

Output

A shopping list such that all your friends get something they like. Each type of food can appear at most once (either in vegan or non-vegan form, but not both.) If more than one valid solution exists, any of them will do. (It's OK to buy redundant food; in the sample input, you could have bought some vegan hamburger as well to give CLaire and Dennis more of a choice. But it's not important. They like you anyway.)

If no valid solution exists, write "impossible".

Sample Input 1	Sample Output 1
4 Alice * vegan cafe latte * non-vegan hamburger * vegan cake Bob * vegan duck a l'orange * non-vegan hamburger * vegan cake Claire * non-vegan cafe latte * vegan hamburger * vegan cake Dennis * non-vegan duck a l'orange * vegan hamburger * non-vegan cake	vegan cake non-vegan duck a'lorange

Exam Algorithm Design

21 December 2021, 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).

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 on pages 3–7 can be efficiently solved using (possibly several applications of) standard graph traversal methods (such as 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.*) Describe the graph. It is often useful to use a concrete instance (such as one of the sample inputs) and *draw* the graph. In general, what are the vertices? What are the edges? (Are they directed? Do they have weights?) How many vertices and edges are there in terms of the input parameters?
   (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.") Remember to include phrases like "from starting node BLA" or "beginning with the edge BLA" if that makes sense.
   (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 on pages 3–7 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".) Then give a recurrence relation for OPT, including relevant boundary conditions and base cases. Which values of OPT are used to answer the problem?
   (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 on pages 3–7 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 on pages 3–7 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.

Books
Problem ID: books

So many famous authors, so little time. The pile of books you want to read is getting larger and larger, but for this summer holiday you finally decided to do something about this. You want to read as many books as you can *from different authors*.

Reading a book means reading *all* its pages. Reading a page takes a minute. You never read the same book twice.

Input

On the first line, the number m of minutes you've set aside for reading in your summer holidays, and the number n of unread books in your pile. Then follow n lines, one for each book, containing title, author name, and number of pages, separated by comma. To fix notation, the ith book has title t_i, author a_i and consists of p_i many pages, for $1 \leq i \leq n$. The number of pages p_i is a nonzero positive integer.

You can asume that book titles are all different (but author names and page numbers need not be.) You can also assume that there is time to read at least one book.

Output

Write the names of the books you plan to read (and finish!) so as to maximise the *number of books* read. The order is not important. All books must be written by different authors. If there are more than one valid solution (such as in sample 1), any of them is acceptable.

In sample 1, note that there is no valid solution of size four, although the four books *Don Quixote*, *One Hundred Years of Solitude*, *Romeo and Juliet*, and *Hamlet* together take only $320 + 315 + 96 + 256 = 987 <= 1000$ minutes to read. (You don't want to read two Shakespeares.)

Sample Input 1

```
1000 8
Don Quixote,  Miguel de Cervantes, 320
Finnegans Wake,  James Joyce, 813
Hamlet, William Shakespeare, 315
In Search of Lost Time, Marcel Proust, 1235
One Hundred Years of Solitude, Gabriel Garcia Marquez, 96
Romeo and Juliet, William Shakespeare, 256
The Great Gatsby, F. Scott Fitzgerald, 514
Ulysses,  James Joyce, 862
```

Sample Output 1

```
Don Quixote
One Hundred Years of Solitude
Romeo and Juliet
```

Sample Input 2

```
239 1
Kongens Fald, Johannes V. Jensen, 239
```

Sample Output 2

```
Kongens Fald
```

Sample Input 3

```
2000 2
Buddenbrooks, Thomas Mann, 768
Der Zauberberg, Thomas Mann, 1008
```

Sample Output 3

```
Buddenbrooks
```

Completionist

Problem ID: completionist

So much to read, so little time. The pile of books you want to read is getting larger and larger, but for this summer holiday you finally decided to do something about this. You want to read books so as to maximise the number of *pages read*. One of your psychological weaknesses is that you are unable to abandon books mid-way—once you start a book, you must read it to the end.

Reading a book means reading *all* its pages. Reading a page takes a minute. You never read the same book twice.

Input

On the first line, the number m of minutes you've set aside for reading in your summer holidays, and the number n of unread books in your pile. Then follow n lines, one for each book, containing title, author name, and number of pages, separated by comma. To fix notation, the ith book has title t_i, author a_i and consists of p_i many pages, for $1 \leq i \leq n$. The number of pages p_i is a nonzero positive integer.

You can asume that book titles are all different (but author names and page numbers need not be.) You can also assume that there is time to read at least one book.

Output

Write the maximum number of pages you can read.

In sample 1, an optimal solution is to read *Don Quixote*, *Hamlet*, *One Hundred Years of Solitude*, and *Romeo and Juliet*, for a total of $320 + 315 + 96 + 256 = 987$ pages.

Sample Input 1

```
1000 8
Don Quixote,  Miguel de Cervantes, 320
Finnegans Wake,  James Joyce, 813
Hamlet, William Shakespeare, 315
In Search of Lost Time, Marcel Proust, 1235
One Hundred Years of Solitude, Gabriel Garcia Marquez, 96
Romeo and Juliet, William Shakespeare, 256
The Great Gatsby, F. Scott Fitzgerald, 514
Ulysses,  James Joyce, 862
```

Sample Output 1

```
987
```

Sample Input 2

```
239 1
Kongens Fald, Johannes V. Jensen, 239
```

Sample Output 2

```
239
```

Sample Input 3

```
2000 2
Buddenbrooks, Thomas Mann, 768
Der Zauberberg, Thomas Mann, 1008
```

Sample Output 3

```
1776
```

Inequality

Problem ID: inequality

You just finished a maths exam and want to remember all the questions.

You are pretty sure about the numbers and equations, but not about the arithmetic operations involved. Your task is to reassemble a consistent set of equalities and inequalities from what you remember.

Input

On the first four lines, the four arithmetic operations addition, subtraction, multiplication and (integer) divisions.[1] Each is followed by a nonnegative integer: the number of times the operation occurred. Let's call these numbers a, s, m, and d to fix notation. Then follow n lines, each describing an equality or inequality between 3 integers, with the operation missing. The expression is either =, <, or >.

You can assume $n = a + s + m + d$.

Output

The same expressions, correct, with "?" replaced by either +, −, $\star$, or /, such that there are a many +s, s many −s, m many $\star$s, and d many /s.

If there is more than one valid solution, any of them will do. If no solution exists, write "impossible".

Sample Input 1

```
+ 2
- 0
* 2
/ 1
1 ? 1 > 1
5 ? 0 < 1
2 ? 2 > 1
1 ? 1 = 1
10 ? 4 < 7
```

Sample Output 1

```
1 + 1 > 1
5 * 0 < 1
2 + 2 > 1
1 * 1 = 1
10 / 4 < 7
```

Sample Input 2

```
+ 1
- 0
* 1
/ 0
100 ? 200 > 300
99 ? 201 > 300
```

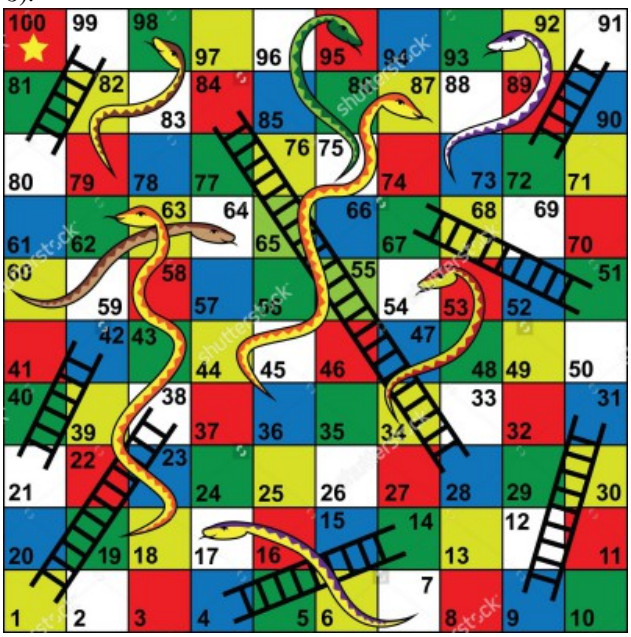
Sample Output 2

```
impossible
```

[1] It makes no difference for the exercise, but let's agree that / is integer division, so $5/3 = 1$, just so we don't have to worry about floating point types. It's not important.

Snakes and ladders
Problem ID: snakesandladders

You can move on the board of the popular children's board game *snakes and ladders* like this:[1] You start at position 1 and want to reach the highest-numbered position. (In the example image, that is position 100.) In one step, you can advance from position i to position $i + 1$. Some positions are the endpoints of either a snake or a ladder. You can move your token from the lower-numbered end of a "ladder" (such as 9) to the ladder's higher-numbered square in one step. (Thus, from 9 you have two choices. You can take a single step to 10 or a single step to 31.) Similarly, from the higher-numbered square of a "snake" (such as 93), you can move down to the snake's lower-numbered square (such as 73).



What is the minimum number of steps you need?

Input

The first line countains d, the dimension of the $d \times d$ board, and the number l of ladders and s of snakes. You can assume $d > 1$. (In other words, there are d^2 squares, numbered $1, 2, \ldots, d^2$.) Then follows the word "ladders", followed by l lines each of which gives the endpoints of a ladder. Then follows the word "snakes", followed by s lines each of which gives the endpoints of a snake.

Output

The smallest number of steps from 1 to d^2.

[1] Forget what you might know about the actual game, which is played with dice and has different rules.

Vegan

Problem ID: vegan

You love you friends dearly, but their dietary restrictions make hosting a party quite the challenge.

Each ingredient comes in a vegan and non-vegan form, and you need to decide which to order. Each of your friends is flexible enough that they accept any one of three different dishes. For instance, Claire would be perfectly happy if you served vegan cake (eggs don't agree with her) or vegan hamburgers (she avoids meat for ethical reasons) or non-vegan cafe lattes (she finds the taste of soy or oat milk disgusting).

You want to accomodate all your friends, so everyone needs to have at least one of their wishes satisfied.

Input

One the first line, the number $f \geq 1$ of friends.

Then follow $4f$ lines, four lines for every friend. The first of these lines is the (unique) name of your friend, and then 3 bulleted lines of what they like to eat. Each of these starts with the word 'vegan' or 'non-vegan', followed by more symbols. (None of these lines need to make culinary sense.)

To fix notation, let's say there are m many different types of food (such as cake), each which can come in two different forms (vegan cake and non-vegan cake).

Output

A shopping list such that all your friends get something they like. Each type of food can appear at most once (either in vegan or non-vegan form, but not both.) If more than one valid solution exists, any of them will do. (It's OK to buy redundant food; in the sample input, you could have bought some vegan hamburger as well to give CLaire and Dennis more of a choice. But it's not important. They like you anyway.)

If no valid solution exists, write "impossible".

Sample Input 1	Sample Output 1
4 Alice * vegan cafe latte * non-vegan hamburger * vegan cake Bob * vegan duck a l'orange * non-vegan hamburger * vegan cake Claire * non-vegan cafe latte * vegan hamburger * vegan cake Dennis * non-vegan duck a l'orange * vegan hamburger * non-vegan cake	vegan cake non-vegan duck a'lorange

Exam Algorithm Design

9 January 2023, 9:00–13:00

Thore Husfeldt and Nutan Limaye

Version 1.3 (post-exam)

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).

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. **Counterexample.** Looking through the problems on pages 3–7, as soon as Gordon Gecko reads the desription of problem "Sale", he is sure it can be solved by a greedy algorithm. His rough idea is the following: "First determine which value x appears most often on the tiles. (Careful with tiles of the form (x, x); you want to count them only once.) Now buy all tiles containing this maximally frequent value x, which costs 1 kr. in total. Continue doing this until there are no more tiles left."
   It's clear that Gordon hasn't really thought a lot about data structures or running times, but you could probably fix that for him. However, his idea is wrong on a more fundamental level.
   (a) (*3 pt.*) Give a concrete, complete instance (either as a cute drawing of tiles or in the input format specified in Sale) on which Gordon's algorithm fails to find an optimal solution. Specify a nonoptimal solution that would be found by Gordon's algorithm, and what an optimal solution would be instead.

2. **Greedy.** One of the problems on pages 3–7 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.)

3. **Graph traversal.** One of the problems on pages 3–7 can be efficiently solved using (possibly several applications of) standard graph traversal methods (such as 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.*) Describe your algorithm. As much as you can, make use of known algorithms. (For instance, don't reinvent a well-known algorithm. Instead, write something like "I will use Blabla's algorithm [KT, p. 342] to find a blabla in the blabla.")
   (c) (*1 pt.*) State the running time of your algorithm in terms of the parameters of the input.

4. Dynamic programming. One of the problems on pages 3–7 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. Which values of OPT are used to answer the problem?

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

5. Flow. One of the problems on pages 3–7 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 Krampfmeier–Strumpfnudel's algorithm ((5.47) in 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]

6. NP-hard. One of the problems on pages 3–7 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 well-known 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.

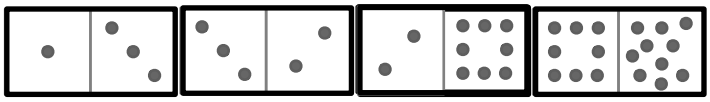
[2] If P = NP then *all* these problems are NP-hard. Thus, in order to avoid unproductive (but hypothetically correct) answers from smart alecks, this section assumes that P $\neq$ NP.

Line

Problem ID: line

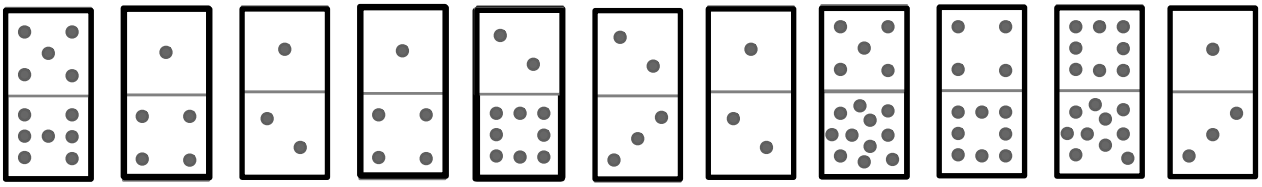
A generalised domino tile (henceforth, *tile*) is a rectangular tile displaying two nonzero positive numbers (a, b), not necessarily different. Tiles display these numbers as small dots called *pips*.

Horizontally oriented tiles can be arranged into a *line* if the number of pips on their adjacent sides match. For instance, here is a line of four tiles:

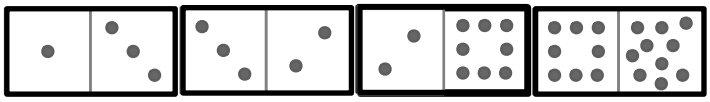


Given T tiles, we want to form a line that begins with a tile whose left side shows 1 pip and ends with a tile whose right side shows the largest possible number of pips.

For instance, if we are given these tiles:



then we can reach a tile containing a side with 10 pips like this:



(Note that we don't care about the number of tiles used – in fact, there is another valid solution reaching 10 that uses only three tiles instead of four.)

Tiles cannot be reused (but there can be duplicate tiles in the input).

Input

On the first line, the number T of tiles.

Then follow T lines, one for every tile. Each of these lines contains two integers a b, which describes a tile containing the numbers a and b.

There is at least one tile with $a = 1$ or $b = 1$. There can be duplicate tiles, and a can be equal to b.

Output

Print a single integer r: the largest number of pips that can appear in a line starting from a 1-pip tile.

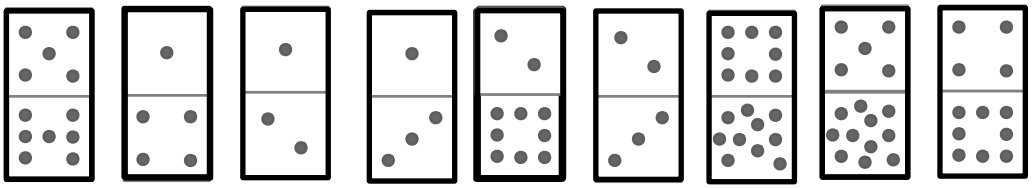
Sample Input 1	Sample Output 1
11 5 7 1 4 1 2 1 4 2 8 2 3 1 2 5 11 4 8 8 10 1 3	10

Pip price

Problem ID: pipprice

A generalised domino tile (henceforth, *tile*) is a rectangular tile displaying two nonzero positive numbers (a, b), not necessarily different. Tiles display these numbers as small dots called *pips*.

The price of a tile is the total number $a + b$ of pips on it. For instance, the tile $(3, 7)$ costs 10 kr. You have k kr. and want to buy as many tiles as you can from T given tiles. For example, if you have 16 kr. then you can buy 3 of the following tiles:



namely the tiles $(1, 4)$, $(2, 3)$ and $(1, 2)$ (which cost $1 + 4 + 2 + 3 + 1 + 2 = 13 \leq 16$ kr.).

Input

On the first line, the number T of tiles and your money k.

Then follow T lines, one for every tile. Each of these lines contains two integers a b, which describes a tile containing the numbers a and b.

There can be duplicate tiles, and a can be equal to b.

Output

Print a single integer r: the number of tiles you can buy.

Sample Input 1

```
9 16
5 7
1 4
1 2
1 3
2 8
2 3
8 10
5 11
4 8
```

Sample Output 1

```
3
```

Sample Input 2

```
2 10
1 3
2 2
```

Sample Output 2

```
2
```

Sample Input 3

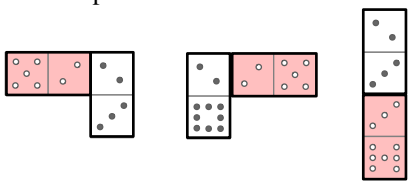
```
1 10
6 7
```

Sample Output 3

```
0
```

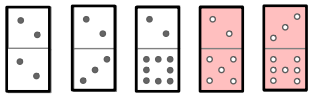
Red and white
Problem ID: redandwhite

A generalised domino tile (henceforth, *tile*) is a rectangular tile displaying two nonzero positive numbers (a, b), not necessarily different. Tiles display these numbers as small dots called *pips* and come in two colours, red and white. Two tiles of different colours form a *couplet* if they can be placed next to each other such that the numbers agree along the shared edge. Here are three examples of couplets:

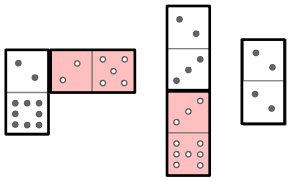


It does not matter if the two tiles in a couplet are arranged horizontally, vertically, form an L-shape, etc. No couplet can consist of more than two tiles.

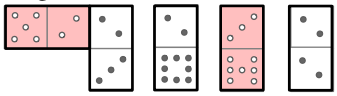
Given some tiles, how many couplets can be created by an optimal arrangement?
For instance, the five tiles



can be arranged as



forming two couplets.
Note the following less-than-optimal arrangement of the same tiles that achieves only one couplet:



Input

On the first line, the number T of tiles. Then follow T lines, one for every tile. Each of these lines contains three values a b c, which describes a tile containing the numbers a and b of colour c, where $c \in \{R, W\}$ (for 'red' and 'white', respectively). There can be duplicate tiles, and a can be equal to b.

Output

Print a single integer r: the maximum number of couplets that can be achieved.

Sample Input 1	Sample Output 1
5 2 2 W 2 3 W 2 8 W 2 5 R 3 7 R	2

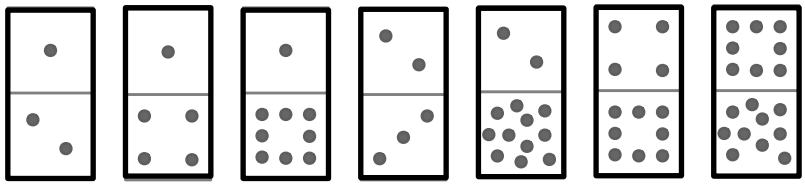
Sample Input 2	Sample Output 2
4 2 3 W 3 8 W 2 7 R 2 9 R	1

Sale
Problem ID: sale

A generalised domino tile (henceforth, *tile*) is a rectangular tile displaying two nonzero positive numbers (a, b), not necessarily different.

The Domino Tile Shop is having a sale! For as little as 1 kr. you can buy *all* the tiles containing a specific value, for instance, *all* tiles containing a 4.

For example, you can buy all these tiles:



for only 3 kr. by bying all tiles containing a 2, all tiles containing a 4, and all tiles containing an 8.

Given T tiles, how many kroner do you need to buy all of them?

Input

On the first line, the number T of tiles.

Then follow T lines, one for every tile. Each of these lines contains two integers a b with $1 \leq a \leq b$, which describes a tile containing the numbers a and b.

There can be duplicate tiles, and a can be equal to b.

Output

Print a single integer k: the smallest cost of buying all tiles. Then print k values $p_1, \ldots, p_k$: the number on the tiles that you buy. To be precise, if you buy all tiles containing p_1, then all tiles containing p_2 among the remaining tiles, and so on up to p_k, then you will have bought all T tiles.

Sample Input 1

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

Sample Output 1

```
3
1 2 8
```

Sample Input 2

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

Sample Output 2

```
3
2 3 4
```

Sample Input 3

```
3
5 5
5 5
5 5
```

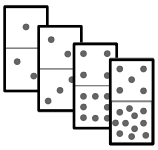
Sample Output 3

```
1
5
```

Stack
Problem ID: stack

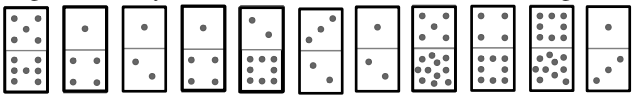
A generalised domino tile (henceforth, *tile*) is a rectangular tile displaying two nonzero positive numbers (a, b), not necessarily different. Tiles display these numbers as small dots called *pips*.

Tiles can be *stacked* if the are "strictly increasing on both sides" in the following sense: If tile (a, b) is placed below tile (a', b') then we require both $a < a'$ and $b < b'$ or, by turning the topmost tile, $a < b'$ and $b < a'$. For legibility, I've drawn a stack below by making the rectangles slightly overlap, but you should think of the four tiles as being neatly stacked exactly on top of each other. For instance, the 4-pip part is exactly below the 5-pip part of the topmost tile:



maybe better drawn as

Given T tiles, what is the highest stack you can build? For instance, if we are given these tiles:



then we can build the stack consisting of $(1, 2)$, $(2, 3)$, $(4, 8)$ and $(5, 11)$ shown in the topmost illustration. (Note that we've turned the $(3, 2)$-tile.) Another way of building a height-4 stack is $(1, 2)$, $(2, 3)$, $(4, 8)$, $(8, 10)$.

Input

On the first line, the number T of tiles. Then follow T lines, one for every tile. Each of these lines contains two integers a b, which describes a tile containing numbers a and b. There can be duplicate tiles, and a can be equal to b.

Output

Print a single integer r: the highest stack (measured in number of tiles) that can be built.

Sample Input 1

```
11
5 7
1 4
1 2
1 4
2 8
3 2
1 2
5 11
4 8
8 10
1 3
```

Sample Output 1

```
4
```

Sample Input 2

```
5
1 1
5 5
10 10
207 6
208 7
```

Sample Output 2

```
4
```

Exam Algorithm Design

15 March 2023

Thore Husfeldt and Nutan Limaye

Version 1.1

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).

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 on pages 3–7 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 on pages 3–7 can be efficiently solved using (possibly several applications of) standard graph traversal methods (such as 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) (*1 pt.*) Explain how you model the problem as a graph problem – what are the vertices, what are the edges, how many are there in terms of the parameters of the problem statement, are they directed, weighted, etc. Draw the graph(s) corresponding to the sample input(s).
   (c) (*2 pt.*) Describe your algorithm. Be explicit about arguments such as start vertices, stopping conditions, etc. 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 on pages 3–7 is solved by dynamic programming.
   (a) (*1 pt.*) Which one?
   (b) (*4 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. Which values of OPT are used to answer the problem?
   (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 on pages 3–7 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 Krampfmeier–Strumpfnudel's algorithm ((5.47) in 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 on pages 3–7 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 well-known 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.

Lab Leak
Problem ID: lableak

Oh no! A pathogen has escaped from your lab by infecting one of the scientists. Let's just hope it doesn't reach you.

Thanks to excellent surveillance technology, you have information of all human contacts in the city. In particular, you know for each pair of individuals u and v if a pathogen could be transmitted from u to v (and vice versa) due to human contact. Let's agree that individual 0 is the infected scientist, and $n-1$ represents you.

Input

On the first line of input, the number n of individuals and the number m of connections. Both numbers are integers with $n \geq 2$ and $m \geq 0$; we assume the individuals are numbered $0, \ldots, n-1$. Then follow m lines each containing two integers u and v with $0 \leq u < v < n$, meaning that the pathogen would be transmitted from u to v and from v to u.

Output

Print 'oh, no!' if the pathogen can be transmitted from 0 to $n-1$. Otherwise print 'phew!'

Sample Input 1	Sample Output 1
5 4 0 1 0 2 1 2 3 4	phew!

Sample Input 2	Sample Output 2
5 4 0 1 1 2 2 3 3 4	oh, no!

Mette
Problem ID: mette

Your are the leader of the largest political party of a small nation and are tasked with forming a stable government. A government is stable if it has a majority of seats in parliament. (To be precise, a government consists of a subset of parties, and its number of seats is the sum of the seats of the individual parties.)

To make your life as a government leader easy, you want to include as few parties as possible.

For example, in sample input 1, there are 175 seats in total. The 3 parties 'Moderaterne', 'Socialdemokratiet', and 'Venstre' together have $16 + 50 + 23 + = 89$ seats, which is strictly more than $\frac{175}{2} = 87.5$. In this election, there is no way to form a government with only 2 parties that have at least 88 seats in total, so '3' is the minimum number of parties needed.

For simplicity, you can assume that the total number of seats is odd, and that there is exactly one largest party (namely, yours).

Input

On the first line, the number p of parties, a nonzero positive integer. For $1 \leq i \leq p$, the ith of the of the following p lines contains the name of the ith party and its number s_i of seats. You can assume that s_i is a nonnegative integer.

Output

The smallest number of parties needed to form a government.

Sample Input 1

```
14
Alternativet 6
Danmarksdemokraterne 14
Dansk Folkeparti 5
Enhedslisten 9
Frie Grønne 0
Konservative 10
Kristendemokraterne 0
Liberal Alliance 14
Moderaterne 16
Nye Borgerlige 6
Radikale Venstre 7
Socialdemokratiet 50
Socialistisk Folkeparti 15
Venstre 23
```

Sample Output 1

```
3
```

Sample Input 2

```
1
The Only Party 101
```

Sample Output 2

```
1
```

Mirror
Problem ID: mirror

A string of uppercase letters is a *mirror* if it is the same backwards and forwards. For instance `OTTO` is a mirror, and so are `LEVEL` and `RACECAR`. By definition, one-letter strings such `A` or `L` are mirrors. On the other hand `MIRROR` is not a mirror, nor is `BANANA`.

What is the minimum number of characters that must be inserted into a given string to make it a mirror? For instance, `BANANA` is turned into a mirror by inserting a single `B` (at the end), and `LOVELY` can be turned into `YLOEVEOLY` by inserting 3 letters.

Input

A string S of n uppercase letters from the English alphabet, `A` to `Z`.

Output

The minimum number of letters that must be inserted into S so that it becomes a mirror.

Sample Input 1	Sample Output 1
LOVELY	3

Sample Input 2	Sample Output 2
LEVEL	0

Sample Input 3	Sample Output 3
BANANA	1

Pipeline

Problem ID: pipeline

To your annoyance, the enemy is sending fossil fuel through a vast system of pipes to one of your vassal states. You have decided to put an end to this by destroying some of the pipes. Each such operation takes resources and jeopardises your international standing, so you want to destroy as few pipes as possible.

Input

On the first line, the integers n and m. We have $1 < n \leq m$. There are n states; the enemy is state 0 and the vassal state is $n - 1$. Then follow m lines each containing two integer u and v with $0 \leq u < v < n$, meaning that there is a pipe between state u and state v that allows fossil fuel to flow in either direction.

You can assume that the entire system of pipes is connected; in particular there is at least one sequence of pipes connecting state 0 to state $n - 1$.

Output

A single integer: the number of pipes you must destroy so that the enemy can not send any fossil fuel to the vassal state.

Sample Input 1

```
6 8
0 1
0 2
1 2
3 5
4 5
3 4
1 3
2 4
```

Sample Output 1

```
2
```

Sample Input 2

```
6 7
0 1
0 2
1 2
3 5
4 5
3 4
1 3
```

Sample Output 2

```
1
```

Polynomials

Problem ID: polynomials

You are given m many polynomials in n variables $x_1, \ldots, x_n$, such as

$$(x_1)^2 - 1 \quad \text{and} \quad x_1 + x_2 + x_3 + 3\,.$$

(Recall that a polynomial is a sum of products of variables and constants.) You want to determine if you can make all polynomials equal to 0 by setting the variables $x_1, \ldots, x_n$ to integer values.

In the above example, this is possible by setting $x_1 = -1$, $x_2 = +5$ and $x_3 = -2$, because

$$(-1)^2 - 1 = 0 \text{ and } (-1) + (+5) + (-2) + 3 = 0\,.$$

There are many other solutions.

In general, a solution may not possible; here's a simple example of $m = 2$ polynomials in just a single variable:

$$x_1 - 1 \quad \text{and} \quad x_1 + 1\,.$$

No matter which value we choose for x_1, at least one of the resulting expressions will be nonzero.

Input

One the first line, the number n of variables and m of polynomials.

We assume that the variables are called $x_1, \ldots, x_n$ and that $1 \leq n$ and $0 \leq m$. Then follow m lines, each containing a polynomial. Each polynomial is given as an expression in a modern programming language, where the variables are written using brackets, like `x[35]`, multiplication is written using the asterisk `*`.

Output

Output a sequence of n integers, the values of $x_1, \ldots, x_n$ so that all polynomials evaluate to 0.

If no valid solution exists, write "impossible".

Sample Input 1

```
3 2
x[1] * x[1] - 1
x[1] + x[2] + x[3] + 3
```

Sample Output 1

```
-1 5 -2
```

Sample Input 2

```
1 2
x[1] - 1
x[1] + 1
```

Sample Output 2

```
impossible
```