Exam EDAF05

8 Jan 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.

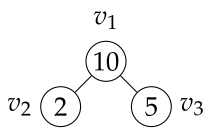
Zero

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:

```
thehobbitorthereandbackagainianaholeinthegroundtherelivedahobbit...
```

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]

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:

```
thehobbitorthereandbackagainianaholeinthegroundtherelivedahobbit
```

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 k tables in the restaurant, of given sizes $r_1, r_2, \dots, r_k$ with $r_1 + r_2 + \cdots + r_k = n$. Everybody must be seated at some table, and nobody should sit next to anybody they don't like.[2]

Input

The first line contains k. The second line contains the integers $r_1, r_2, \dots, r_k$, 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 k lines.

The ith line contains a list of r_i names $w_1, \dots, w_{r_i}$ 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 Employee 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
```

[2] If you want, you can assume $r_i \geq 2$ for each $1 \leq i \leq k$. Thus we avoid arguing about whether a person is friends with themselves. It's not important.

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 $O(n^{17} \log^{-3} \epsilon + \log^2 m)$.")[3]

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?

[3]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.[4]

(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.

[4]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.