Exam EDAF05

29 May 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.

Filling out the exam Some questions are multiple choice. Mark the box or boxes with a cross or a check-mark. If you change your mind, completely black out the box and write your answer's letter(s) in the left margin. In case it's unclear what you mean, I will choose the least favourable interpretation.

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.

Scoring For the free form choice questions, you get between 0 and the maximum number of points for that question. Short, correct answers are preferred.

Each multiple choice question has exactly *one* correct answer. To get the maximum score for that question, you must check that answer, and only that. However, to reflect partial knowledge, you *may* check several boxes, in case you're not quite sure (this lowers your score, of course – the more boxes you check, the fewer points you score). If you check *no* boxes or *all* boxes, your score for that question is 0. If the correct answer is not among the boxes you checked, your score is negative, so it's better to *not* answer a question where you're on very thin ice. The worst thing you can do is to check all boxes except the correct one, which gives you a large negative score.

Want an example? Assume a question worth maximum 2 points has $k = 4$ possible answers (one of them correct).

- If you select only the correct answer, you receive 2 points.
- If you select 2 answers, one of which is correct, you receive 1 point.
- If you select 3 answers, one of which is correct, you receive 0.41 points.
- if you select no answer or all answers, you receive 0 point.
- If you select only one answer, and it is wrong, you receive -0.67 points.
- If you select 2 answers that are both wrong, you receive -1 point.
- If you select 3 answers that are all wrong, you receive -1.25 points.

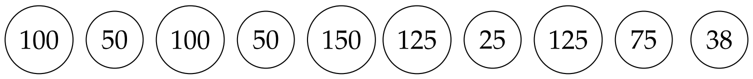
As a special case, for a yes/no question, you receive 1, 0, or -1 points, depending on whether your answer is correct, empty, or wrong.

If you want to know the precise formula: if the question has k choices, and you checked a boxes, your score is $\log(k/a)$, provided you checked the correct answer, and $-a\log(k/a)/(k-a)$ if you only checked wrong answers. Moreover, I have weighted the questions by relevance (not necessarily difficulty), and indicated the maximum points with each question.

You really care why this scoring system makes sense? Then read [Gudmund Skovbjerg Frandsen, Michael I. Schwartzbach: A singular choice for multiple choice. SIGCSE Bulletin 38(4): 34–38 (2006)]. For example, random guessing will give you exactly 0 points, at least in expectation.

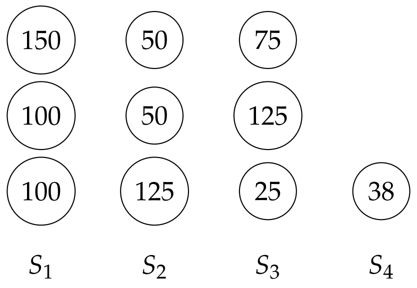
Buildstacks

A number of n coins are laid out in a line in front of you, like this:



For some given $m < n$, you want to make $k = \lceil n/m \rceil$ stacks $S_1, \ldots, S_k$ of coins. Each stack contains m coins, except maybe the last, which contains between 1 and m coins. The rule is that that total sum of values in the resulting stacks has to be nonincreasing.

For instance, if $m = 3$ you could arrange the above input like this:



The total stack values are $350 \geq 225 \geq 225 \geq 38$. There are many other valid solutions.

Input

The first input line contains the values of n and m. On the next line are n integers $c_1, \ldots, c_n$, one for each coin.

Output

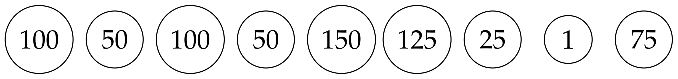
$\lceil m/n \rceil$ lines; the ith line contains the values of the coins in stack i in some order, separated by space.

Input:
```
10 3
100 50 100 50 150 125 25 125 75 38
```

Output:
```
100 100 150
125 50 50
75 125 25
38
```

Neighbours

A number of n coins are laid out in a line in front of you, like this:



You have to take as much money as you can, 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 containing n, followed on the next line by integers $c_1, \dots, c_n$, separated by space, one for each coin from left to right.

Output

The indices of the coins you pick.

Input:

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

Output:

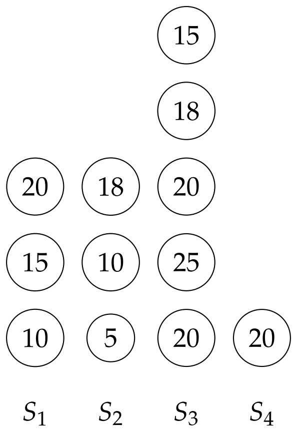
```
1 3 5 7 10
```

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

Takestacks

In front of you are m stacks of coins. You want to take as many stacks of coins as you can, but you are not allowed to ever take two coins of the same value.

For example, here are $m = 4$ stacks.



The best you can do is to pick 2 stacks, S_4 and S_2. You cannot pick S_1 and S_2 because that would make you take coin 10 twice.[1] Coins can come in all kinds of crazy values. You can assume that no stack contains the same coin value twice.

Input

The first line contains n, the number of different values of coin. On the next line follow n different integers $c_1, \ldots, c_n$ separated by space, the different values of coins. The third line contains m, the number of stacks. Then follow m lines $l_1, \ldots, l_m$. Line l_j contains the values of the coins in the jth stack, listed from bottom to top as integers separated by space.

Output

Print the maximal number of stacks you can take.

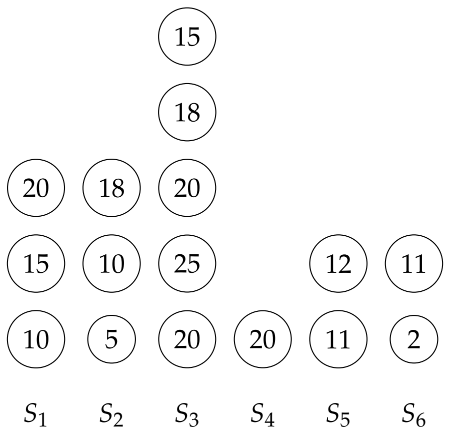
Input:
```
7
10 15 20 5 18 25 15
4
10 15 20
5 10 18
20 25 20 18 15
20
```

Output:
```
2
```

[1]To avoid a possible misunderstanding: Note that just picking stack S_3 earns you 5 coins, but that's not what we're trying to optimize: you want to pick as many *stacks* as possible, not *coins*.

Rotten

In front of you are m stacks of coins like this (for $m = 4$):



You can spread a deadly coin disease by infecting a single type of coin (say, "10"). This will immediatly make all coins of that value very sick, and the sickness will spread through the entire stack of a sick coin, possibly infecting other coins. And so on.

For instance, if I infect "25" in the above setting, this infects all of S_3. In particular, all coins of value "18" become sick, infecting S_2, which infects S_1 (via "10"), and so on. Note that when the process is finished, S_5 and S_6 do not get infected.

Your task is to infect all the stacks, if possible.

Input

The first line contains n, the number of different values of coin. On the next line follow n different integers $c_1, \ldots, c_n$ separated by space, the different values of coins. The third line contains m, the number of stacks. Then follow m lines $l_1, \ldots, l_m$. Line l_j contains the values of the coins in the jth stack, listed from bottom to top as integers separated by space.

Output

Print the value of a coin that would infect all stacks (if such a coin exists.) Otherwise print `no`.

Input:
```
8
2 5 11 12 15 18 20 25
6
10 15 20
5 10 18
20 25 20 18 15
20
11 12
2 11
```
Output:
```
no
```

Strike

All schools are closed because of a strike. 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 $c_1, \ldots, c_n$. 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 `Sara -- Moster Clara`. This means that Sara could potentially visit Moster Clara (given that there is room).

Output

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

Input:
```
3
Peter
Sara
Kurt
2
Moster Clara (2)
Mormor och morfar (2)
Peter -- Moster Clara
Sara -- Moster Clara
Kurt -- Moster Clara
Sara -- Mormor och morfar
Kurt -- Mormor och morfar
```

Output:
```
Peter -- Moster Clara
Sara -- Moster Clara
Kurt -- Mormor och morfar
```

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

Exam Questions

Analysis of algorithms

1. Consider the following piece of code:

```
1: if $n > 10$:
2:     for $i = 1$ to $n$:
3:         for $j = 1$ to $n$:
4:             print $j$;
5: else
6:     for $i = 1$ to $n$:
7:         print $i$;
```

(a) (*1 pt.*) What is the running time? (Choose the smallest correct estimate.)

A $O(n)$ B $O(n \log n)$ C $O(n^2)$ D $O(n^3)$

2. Consider the following piece of code. (The only difference to the above question is the inequality in the first line.)

```
1: if $n < 10$:
2:     for $i = 1$ to $n$:
3:         for $j = 1$ to $n$:
4:             print $j$;
5: else
6:     for $i = 1$ to $n$:
7:         print $i$;
```

(a) (*1 pt.*) What is the running time? (Choose the smallest correct estimate.)

A $O(n)$ B $O(n \log n)$ C $O(n^2)$ D $O(n^3)$

Greedy

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

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

A Buildstacks B Neighbours C Takestacks D Rotten E Strike

(b) (*1 pt.*) The algorithm begins by sorting the input by

A coin value... B stack size... C stack sum... D neighbour gap...
E capacity... F age... G something else...

(c) (*1 pt.*) ... in

A ascending order ("smallest first"). B descending order ("largest first").

(d) (*3 pt.*) Write the output when your run your greedy algorithm on the example instance.

(e) (*1 pt.*) The total running time is (give the smallest correct estimate)

A $O(n \log n)$. B $O(m \log m)$.

Graph connectivity

4. One of the problems can be efficiently reduced to graph connectivity problem.

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

A Buildstacks B Neighbours C Takestacks D Rotten E Strike

(b) (*2 pt.*) Draw the graph corresponding to the example instance.

(c) (*1 pt.*) How many vertices does the graph have in terms of the parameters of the original problem?

A n B m

(d) (*2 pt.*) Which algorithm will you run on the graph? Choose the correct claim. (Only one is correct.)

A I have to use BFS. DFS will not necessarily give the right answer because it does not find the shortest paths.

B Because the edges are weighted, I need to use Dijkstra's algorithm.

C Prim's algorithm would work, but it's not the fastest way to solve the problem. (BFS would be faster.)

D I cannot use Dijkstra because the graph has negative cycles.

(e) (*1 pt.*) State the running time of your algorithm in terms of the original parameters. (Choose the smallest correct estimate.)

A $O(n+m)$ B $O(n \log n)$ C $O(m \log n)$ D $O((n+m) \log n)$

Dynamic programming

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

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

A Buildstacks B Neighbours C Takestacks D Rotten E Strike

(b) (*4 pt.*) Following the book's notation, we let OPT(i, j) denote the value of a partial solution. Then OPT satisfies[2]

A

$$\text{OPT}(i,j) = \begin{cases} 1 + \min\{\text{OPT}(i-1,j), \text{OPT}(i-1,j-1), \text{OPT}(i,j-1)\}, & \text{if } c_i \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

B

$$\text{OPT}(i,j) = \begin{cases} \text{OPT}(i+1,j-1), & \text{if } c_i = 1, \\ 1 + \min\big(\text{OPT}(i+1,j), \text{OPT}(i,j-1), \text{OPT}(i+1,j-1)\big), & \text{otherwise.} \end{cases}$$

C

$$\text{OPT}(i,j) = c_i + \min\{\, \text{OPT}(i-1,j), \text{OPT}(i-1,j-1), \text{OPT}(i,j-1) \,\}$$

D

$$\text{OPT}(i,j) = \max\{\, \text{OPT}(c_i,j), \text{OPT}(i-1,j) \,\}$$

E

$$\text{OPT}(i,j) = \min_{j-1 \leq k \leq j+1} \{\, \text{OPT}(i-1,k) + c_k \,\}$$

F

$$\text{OPT}(i,j) = \begin{cases} \text{OPT}(i-1,j) & \text{if } c_i + c_j \neq 0 \\ \max\{\text{OPT}(i-1,j), 1 + \text{OPT}(i-1,j-1)\} & \text{otherwise} \end{cases}$$

G

$$\text{OPT}(i) = \max\{\text{OPT}(i-1), c_i + \text{OPT}(i-2)\}$$

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

A $O(n)$ B $O(nm)$ C $O(n \log n)$ D $O(n^2)$

Network flow

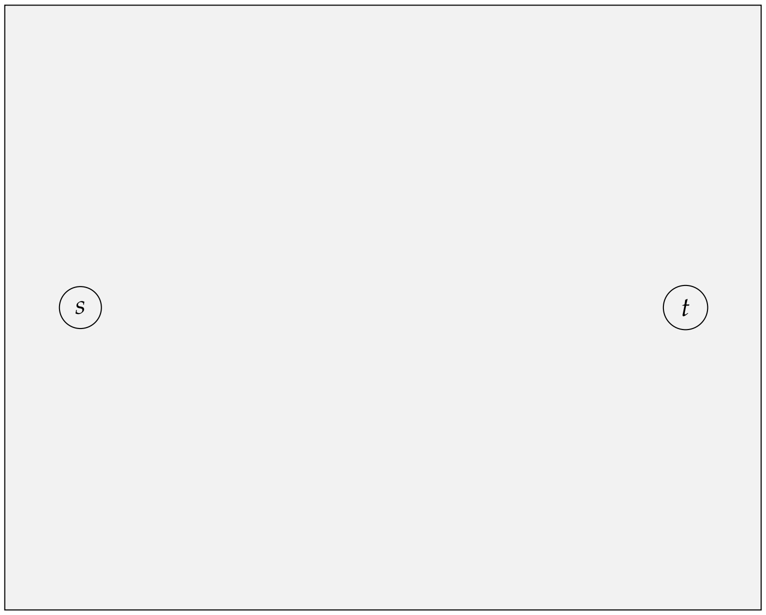
6. One of the problems in the set is easily solved by a reduction to network flow.

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

A Buildstacks B Neighbours C Takestacks D Rotten E Strike

[2] There may be other cases, in particular, boundary conditions such as maybe for OPT$(1, 1)$ or OPT$(n, 0)$ or $i = j$, etc. Don't worry about them. This question is just about the most central part of the recurrence relation, otherwise this exercise becomes too large.

(b) (*3 pt.*) Draw the resulting network for the example instance. Make sure your draw all nodes, all edges, and all capacities. (I've started by already drawing the source and the sink for you.)



(c) In general, the number of nodes in the resulting graph in terms of the original problem's parameters is: (Choose the smallest correct estimate). (*1 pt.*)

$\boxed{\text{A}}$ $O(n)$ $\boxed{\text{B}}$ $O(m)$ $\boxed{\text{C}}$ $O(nm)$ $\boxed{\text{D}}$ $O(n+m)$

Computational complexity

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.)

$\boxed{\text{A}}$ Buildstacks $\boxed{\text{B}}$ Neighbours $\boxed{\text{C}}$ Takestacks $\boxed{\text{D}}$ Rotten $\boxed{\text{E}}$ Strike

(b) (*1 pt.*) The easiest way to see that is to take the following NP-hard problem, called P_2,

$\boxed{\text{A}}$ Graph colouring $\boxed{\text{B}}$ 3-dim. matching $\boxed{\text{C}}$ Independent set $\boxed{\text{D}}$ Set Cover
$\boxed{\text{E}}$ Vertex cover $\boxed{\text{F}}$ 3-satisfiability $\boxed{\text{G}}$ Travelling salesman $\boxed{\text{H}}$ Hamiltonian path

(c) (*1 pt.*) and prove

$\boxed{\text{A}}$ $P_1 \leq_P P_2$ $\boxed{\text{B}}$ $P_2 \leq_P P_1$

(d) (*5 pt.*) Describe the reduction on a separate piece of paper. 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. Please, please, please start your answer with words like this "Given an instance to problem *blabla*, . . . ,".

[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.