Exam Algorithm Design

21 December 2023, 9:00–13:00

Thore Husfeldt

Version 1.4

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 description of problem "Decrease", he is sure it can be solved by a greedy algorithm. His rough idea is the following: "First find the bitstring with the most 1s, say x, and add it to the solution. Then consider all bitstrings that x can be decreased into; among those pick the one with the most 1s. Make that the new x; and repeat until no bitstrings are 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 small but complete instance (as a concrete bitstring) on which Gordon's algorithm is guaranteed to fail 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.

Bit Shelter
Problem ID: bitshelter

Recall that "or" of two bits is defined as

$$a \vee b = \begin{cases} 1 & \text{if } a = 1 \text{ or } b = 1 \text{ (or both)} \\ 0 & \text{otherwise,} \end{cases}$$

and that this concept is extended to length-K bitstrings by defining $z = x \vee y$ as

$$z_i = x_i \vee y_i \qquad (1 \leq i \leq K)\,.$$

This is called the bitwise-or of x and y. For instance, $1001 \vee 0011 = 1011$. It extends naturally to more than two strings (the operation is associative), for instance, $10011 \vee 00111 \vee 11101 = 11111$.

Given N many bitstrings, each of length K, select as few as possible so that their bitwise-or is $1 \cdots 1$ (i.e., K many 1s).

Input

On the first line of input, the number N of bitstrings, and the length K of every bitstring. Each of the following N lines contains a bitstring.

It is guaranteed that the bitwise-or of *all* the input bitstrings is $1 \cdots 1$ (i.e., K many 1s.) You can assume that the bitstrings are all different.

Output

Output M lines, where $1 \leq M \leq N$. Each line must be a bitstring from the input. The bitwise-or of the M lines must be $1 \cdots 1$ (i.e., K many 1s), and M must be minimal.

Sample Input 1

```
5 14
00011110001111
11111110000000
00000001111111
01100000110000
10000001000000
```

Sample Output 1

```
00000001111111
11111110000000
```

Sample Input 2

```
4 2
11
01
10
00
```

Sample Output 2

```
11
```

Cluster

Problem ID: cluster

You are given a sequence of N bitstrings, each of length K. Your task is the find a largest *cluster of ones*. Such a cluster consists of 1s that are adjacent to other 1s, in either of the four directions left, right, up, and down.

For instance, in Sample Input 1 there are two such clusters; of size 8 and 7, respectively. Note that Sample Input 2 contains five clusters, the largest is the one in the top-right corner.

Input

On the first line, the number N of bitstrings and the length K of each bitstring.

Then follow N lines, each containing exactly one bitstring.

Output

A single integer: the number r of 1s in the largest cluster in the input.

Sample Input 1

```
5 5
11111
11000
10000
00111
01111
```

Sample Output 1

```
8
```

Sample Input 2

```
4 4
1001
0101
0010
0001
```

Sample Output 2

```
2
```

Sample Input 3

```
5 5
11111
10001
10101
10001
11111
```

Sample Output 3

```
16
```

Sample Input 4

```
1 10
0000000000
```

Sample Output 4

```
0
```

Decrease
Problem ID: decrease

A bit string can be *decreased* into another bitstring of the same length by turning some of its 1s into 0s. For instance, 0110 can be decreased into 0100 and into 0010 (and even into 0000). On the other hand, 0110 cannot be decreased into 1000.

Given some bitstrings of equal length, select a maximal sequence such that each bitstring can be decreased into its successor.

Input

On the first line of input, the number N of bitstrings, and the length K of every bitstring. Then follow N lines, each containing a bitstring.

Input

On the first line of input, the number N of bitstrings, and the lengh K of each bitstring. Then follow N lines, each containing a bitstring. You can assume that they are all different.

Output

Print M lines of bitstrings, where $1 \leq M \leq N$. Each bitstring must occur in the input, and they must all be different. Moreover, for $1 \leq i < M$, the ith bitstring in the output can be decreased into the $(i+1)$st. (In other words, if x is followed by y then $x_j \geq y_j$ for all $1 \leq j \leq K$.) The number of lines M must be maximal.

If more than one solution of maximal size M exists (such as in Sample Input 3), any will do.

Sample Input 1

```
5 5
00000
11000
00111
01000
11111
```

Sample Output 1

```
11111
11000
01000
00000
```

Sample Input 2

```
1 10
1101110111
```

Sample Output 2

```
1101110111
```

Sample Input 3

```
4 2
00
01
10
11
```

Sample Output 3

```
11
10
00
```

Expensive Ones

Problem ID: expensiveones

You are given N many bitstrings of length K and want to print out as many as you can. However, your printing shop is running out of 1s; there are only R many 1s left. (On the other hand, there are way more than $N \cdot K$ many 0s left, so you won't run out of those.)

Input

On the first line, the number N of bitstrings, the length K of every bitstring, and the number R of remaining 1s in the printing shop. You can assume $0 \leq R \leq N \cdot K$. You can assume that the bitstrings are different.

Output

Print M lines, with $0 \leq M \leq N$, each of which is a single bitstring from the input. They have to be all different, their order does not matter, and the total number of 1s in those M lines must be $\leq R$.

The number M of printed bistrings must be as large as possible. If there is more than one valid solution, any will do.

Sample Input 1

```
6 5 4
01110
01010
11110
10000
00000
01100
```

Sample Output 1

```
10000
00000
01110
```

Sample Input 2

```
2 3 6
111
110
```

Sample Output 2

```
110
111
```

Sample Input 3

```
4 10 0
0010000000
0000001000
0000000000
0000000001
```

Sample Output 3

```
0000000000
```

Sample Input 4

```
1 5 1
00011
```

Sample Output 4

To Xor Or Not To Xor?

Problem ID: toxorornottoxor

We recall the definition of bitwise boolean operations on bitstrings: First recall the bit operations $\wedge$ (also known as "and", conjunction), $\vee$ (also known as "or", disjunction), and $\oplus$ (also known as "xor", "exclusive or") defined on single bits $a, b \in \{0, 1\}$ as follows:

a	b	$a \vee b$	$a \wedge b$	$a \oplus b$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Then, for a pair of equal-length bitstrings $x_1 \cdots x_k, y_1 \cdots y_k \in \{0, 1\}^k$ we define the result of $\vee$, $\wedge$, and $\oplus$ element-wise:

- the ith elment of $x \vee y$ is $x_i \vee y_i$.
- the ith elment of $x \wedge y$ is $x_i \wedge y_i$.
- the ith elment of $x \oplus y$ is $x_i \oplus y_i$.

For instance, if $x = 011$ and $y = 110$ then $x \vee y = 111$, $x \wedge y = 010$, and $x \oplus y = 101$.

You are given N many pairs of bitstrings, each of length k. The task is to associate with each pair a single operation so as to produce the largest number d of *different* bitstrings.

Consider for instance Sample Input 1 and three possible sets of associations:

We would construct $d = 2$ different bitstrings like this:

$1110 \vee 1111 = 1111$
$1111 \vee 0001 = 1111$
$1010 \wedge 0101 = 0000$

We would construct $d = 3$ different bitstrings like this:

$1110 \vee 1111 = 1111$
$1111 \wedge 0001 = 0001$
$1010 \wedge 0101 = 0000$

A worst possible answer would be this, which creates the same bitstring for each pair, so $d = 1$.

$1110 \vee 1111 = 1111$
$1111 \vee 0001 = 1111$
$1010 \oplus 0101 = 1111$

Input

On the first line, the number n of pairs of bitstrings, and the length k of each bitstring. Then follow n lines, each with a pair of bitstrings separated by space.

Output

Output $n + 1$ lines. On the first line, the integer d of how many *different* bistrings you will write. Then follow n lines; the ith of these output lines contains either $\vee$, $\wedge$, or $\oplus$ of the ith input line. There must be exactly d different lines among these n lines, and d must be maximal. (If there are several valid outputs, any of them will do.)

Sample Input 1

```
3 4
1110 1111
1111 0001
1010 0101
```

Sample Output 1

```
3
1111
0001
0000
```

Sample Input 2

```
3 2
11 11
11 00
00 11
```

Sample Output 2

```
2
11
00
00
```