EDAF05 Exam

28 May 2010, 8.00–13.00

Thore Husfeldt, Computer Science, Lund University

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 Most 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 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, you 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 you 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.

Algorithmic Problems

Square

Jan de Vries owns a huge rectangular flower planation $F[i, j]$, neatly arranged into n rows and m columns. At position (i, j) there is a tulip if $F[i, j] = 1$. Position (i, j) is empty if $F[i, j] = 0$.

It is Spring, and Dutch tulips are in high demand. Jan wants to quickly find the largest square entirely composed of tulips. Since n and m are huge, he cannot afford to run the obvious algorithm (test all possible squares), which would take time $O(nm \min\{n^2, m^2\})$.

Input

The input consists of a line containing n and m, followed by n lines of m bits describing F in the obvious manner.

Output

Integers i and j and r such $F[i', j'] = 1$ for all $i \leq i' < i + r$ and all $j \leq j' < j + r$, and r is maximal. (If there are several optimal answers, any of them will do.)

	Example 1	**Example 2**
Input		

	`2 3`
	`0 1 1`
	`1 1 1`
		`2 3`
		`1 0 1`
		`1 1 1`

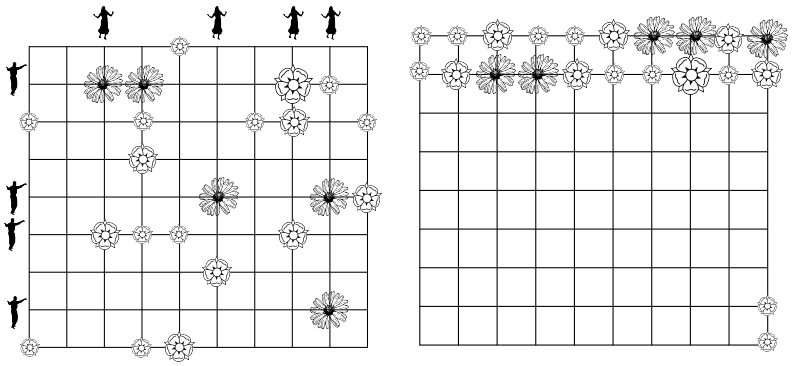
Output		
	`1 2 2`	`2 1 1`

Dance

The traditional Dutch wedding dance works like this. You need a huge, rectangular field of flowers $F[i, j]$, ($1 \leq i \leq n$, $1 \leq j \leq m$) and k engaged couples. At row i and column j there may be a flower ($F[i, j] = 1$) or not ($F[i, j] = 0$). The dance starts by each man picking a different row, and each woman picking a different column. When the music starts, the men dance along their row, and the women down their column, until each couple meets. The music stops, the man picks a flower (if there is one) and gives it to the woman.

For example, Hans has picked row 5 and Helga has picked column 6. They will meet in $(5, 6)$, where there is a flower. In fact, in the example to the left, all four couples will be able to get flowers, so the dance is a success.

On the other hand, in the example to the right, no matter how the 4 men pick their rows, there will only be 3 couples that are able to meet on a spot with a flower.



Input

The input consists of a line containing n and m and k, followed by n lines of m characters describing F in the obvious manner. The total number of flowers is r.

Output

If all couples can meet on a spot with a flower, output the row numbers for the men, followed by the column numbers for the women on a separate line.

Otherwise output "failure".

	Example 1	**Example 2**
Input		

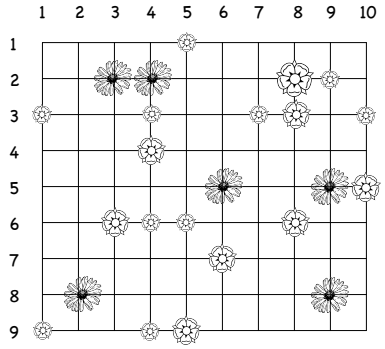
	`9 10 4`
	`0000100000`
	`0011000110`
	`1001001101`
	`0001000000`
	`0000010011`
	`0011100100`
	`0000010000`
	`0000000010`
	`1001100000`
		`9 10 4`
		`1111111111`
		`1111111111`
		`0000000000`
		`0000000000`
		`0000000000`
		`0000000000`
		`0000000000`
		`0000000001`
		`0000000001`

Output		
	`2 5 6 8` `3 6 8 9`	`failure`

Sell!

Pieter de Bert owns a huge rectangular flower planation $F[i, j]$, neatly arranged into n rows and m columns. At position (i, j) there is a flower if $F[i, j] = 1$. Position (i, j) is empty if $F[i, j] = 0$.

It is Spring, and Dutch flowers are in high demand. Pieter can sell off his flowers one entire column at a time, the worth of a column is the number of flowers in it. He wants to sell off exactly k columns, and he wants to make as much money as he can.



For example, if $k = 3$ in the above example, Pieter could sell the first three columns for a total profit of $2 + 1 + 2 = 5$. But it would be better to sell columns 4, 8, and 9 for a total profit of $5 + 3 + 3 = 11$.

Input

The input consists of a line containing n and m and k, followed by n lines of m bits describing F in the obvious manner. There are r flowers in total.

Output

The names of the columns Pieter should sell to maximise his profit.

Example 1

Input

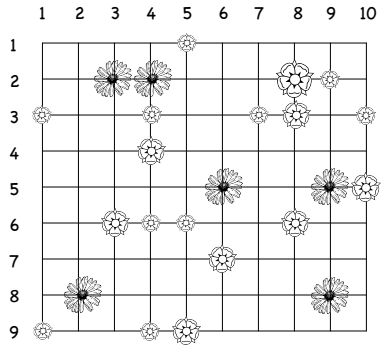
```
9 10 3
0000100000
0011000110
1001001101
0001000000
0000010011
0011100100
0000010000
0100000010
1001100000
```

Output

```
4 8 9
```

EU

Lazy Lars owns a huge flower plantation, neatly arranged into n rows and m columns. At position (i, j) there is a flower if $F[i, j] = 1$. Position (i, j) is empty if $F[i, j] = 0$.



Lars gets EU subventions for every row that is not empty. So Lars has made sure that every row contains at least one flower somewhere. To make even more money, he has decided to sell off most of his columns (Dutch law says that you may only sell off one entire column at a time). However, he wants to make sure that all his n rows still contain at least one flower each.

In the example above, Pieter couldn't sell off both columns 2 and 9, because then there would be no flower left in row 8. On the other hand, he could sell all columns 1, 3, 7, 8, 9, and 10 and still keep every row nonempty.

Input

The input consists of a line containing n and m, followed by n lines of m characters describing F in the obvious manner.

Output

Output a maximal list of column indices that Pieter can sell such that every row still contains at least one flower.

Example 1

Input

```
9 10
0000100000
0011000110
1001001101
0001000000
0000010011
0011100100
0000010000
0100000010
1001100000
```

Output

```
1 3 7 8 9 10
```

Exam Questions

Analysis of algorithms

1. Let $f(n) = ((n^2 + 2n)/n + \frac{1}{1000}n^{3/2}) \log n$. True of false?

(a) (*1 pt.*) $f(n) = O(n^2 \log n)$

A true B false

(b) (*1 pt.*) $f(n) = O(n^{3/2} \log n)$

A true B false

(c) (*1 pt.*) $f(n) = O(n^{3/2})$

A true B false

2. Consider the following piece of code:

```
1: for i = 1 to n
2:     j = 1
3:     while j ≤ n
4:         j = j * 2
```

(a) (*2 pt.*) What is the running time? (Choose the smallest correct estimate.)

A $O(n)$ B $O(\sqrt{n} \log n)$ C $O(n \log n)$ D $O(n^2)$

3. Assume you have a data structure that maintains a set S under insertion and deletion. The operation "**insert**(s, S)" makes sure that $s \in S$ holds, and the operation "**remove**(S)" chooses some $s \in S$ (assuming S is not empty) and removes it.

Consider the following piece of code, starting with an empty set S.

```
1: for i = 1 to n
2:     insert (i^2, S)
3: for i = 1 to n
4:     remove (S)
```

(a) (*1 pt.*) Assume both **insert** (i, S) and **remove**(S) take time $O(\log |S|)$. Then the total running time is: (Choose the smallest correct estimate.)

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

(b) (*1 pt.*) Assume I want the total running time to be $O(n)$. What are the requirements on the running time of **insert** (i, S) and **remove**(S)?

A Both must take constant time.

B **insert** (i, S) must take constant time, but **remove**(S) can take time $O(\log |S|)$.

C **remove**(S) must take constant time, but **insert** (i, S) can take time $O(\log |S|)$.

D This is impossible, no matter what you do.

4. Consider the following piece of code:

```
1: int f(int n) {
3:     if n > 3 then { return f(n − 1) + f(n − 3); }
4:     else return 3;
5: }
```

(a) (*2 pt.*) Which recurrence relation best characterises the running time of this method?

A $T(n) = T(n-1) + T(n-3) + O(1)$

B $T(n) = T(n-1) \cdot T(n-3) + O(1)$

C $T(n) = 2T(n-1) + O(n-3)$

D $T(n) = T(n-1) + O(n)$

(b) (*1 pt.*) For nonnegative integers, define $T(n) = T(n-2), \quad T(0) = 0, \quad T(1) = 5$. Give the smallest correct solution to this recurrence.

A $O(n)$

B $O(n \log n)$

C 5 if $n \geq 1$ is odd, 0 otherwise

D $((1+\sqrt{5})/2)^n$

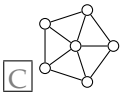
(c) (*1 pt.*) Find the solution to $T(n) = 2 \cdot T(n/2) + O(n \log^2 n), \quad T(1) = 2$. Give the smallest correct answer.

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

Graphs

5. For integer $r \geq 1$, the rth *wheel graph* $W_r = (V_r, E_r)$ consists of a center vertex v_1, connected to $(r-1)$ other vertices $v_2, \ldots, v_r$. Moreover, there is an edge between v_r and v_2; and for $2 \leq i < r$, there is an edge between v_i and v_{i+1}.

(a) (*1 pt.*) How does W_5 look?

A  B  C  D 

(b) (*1 pt.*) The number of edges in W_r is

A $2r-2$ B $\binom{r}{2}$ C r^2-1 D r

(c) (*1 pt.*) W_r is a tree

A True B False

(d) (*1 pt.*) W_r contains a Hamiltonian cycle

A True B False

(e) (*1 pt.*) W_r has a maximum independent set of size

A $O(1)$ B $O(\log r)$ C $\lfloor (r-1)/2 \rfloor$ D r

Greedy

6.

(a) (2 *pt.*) The following problem admits a greedy algorithm:

A Sell! B Square C EU D Dance

(b) (2 *pt.*) The solution involves processing the

A rows . . . B columns . . . C flowers . . . D squares . . .

(c) (2 *pt.*)

A . . . in ascending order of . . . B . . . in descending order of . . .

(d) (2 *pt.*)

A . . . the number of flowers in them.

B . . . their index.

C . . . the leftmost nonempty position.

D . . . the bottommost nonempty position.

(e) (2 *pt.*) The running time is (give the smallest possible)

A $O(nm + k \log m)$. B $O(n \log n)$. C $O(nm \log(nm))$. D $O(r \log r)$.

Dynamic programming

7.

(a) (2 *pt.*) One of the problems is solved by dynamic programming. (But not by a greedy approach.)

A Sell! B Square C EU D Dance

(b) (4 *pt.*) Following the book's notation, we let $\mathrm{OPT}(i,j)$ denote the value of a partial solution. Then OPT satisfies[1]

A

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

B

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

C

$$\mathrm{OPT}(i,j) = \min_{j \le k \le N} \{\, \mathrm{OPT}(i-1,k) + F(i,k) \,\}$$

D

$$\mathrm{OPT}(i,j) = \max\{\, \mathrm{OPT}(F(i,j),j), \mathrm{OPT}(i-1,j) \,\}$$

[1] There may be other cases, in particular, boundary conditions such as maybe for $\mathrm{OPT}(1,1)$ or $\mathrm{OPT}(N,0)$ 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.

E

$$\mathrm{OPT}(i,j) = \min_{j-1 \le k \le j+1} \{ \mathrm{OPT}(i-1,k) + F(i-1,k) \}$$

F

$$\mathrm{OPT}(i,j) = \begin{cases} \mathrm{OPT}(i-1,j) & \text{if } F(i,j) \\ \max\{\mathrm{OPT}(i-1,j), 1 + \mathrm{OPT}(i-1,j-1)\} & \text{otherwise} \end{cases}$$

(c) (*1 pt.*) The resulting running time is (give the smallest correct estimate)

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

(d) (*1 pt.*) The space usage is (give the smallest correct estimate)

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

Network flow

8.

(a) (*2 pt.*) One of the problems in the set is easily solved by a reduction to network flow. That problem is

A Sell! B Square C EU D Dance

(b) (*2 pt.*) The network consists of a node for[2]

A each row and each column B each man
C each woman D each man and each woman

(c) (*1 pt.*) The total number of nodes (including the soure and the sink), in terms of the parameters of the original problem, is

A $m + n + 2$ B mn C $O(r)$ D $O(k)$

(d) (*4 pt.*) Describe the reduction on the back of this page, or on a separate piece of paper. Be ridiculously precise about 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. I cannot stress how imporant such an example is: do it! What does a maximum flow mean in terms of the original problem, and what size does it have in terms of the original parameters?

Computational complexity

These questions are about *Square*, and include questions about NP. Don't take this as an indication that Square may or may not be NP-hard.

9. *(Decision version.)* The input to the decision version of Square includes

(a) (*1 pt.*) the values $F[i,j]$ for all i and j,

A true B false

(b) (*1 pt.*) the coordinates of the upper left corner of a maximal square or 1s

A true B false

[2] Apart from these, there may be a *constant* number of extra nodes, such as a source and a sink, or a single node representing Pieter, or Brussels, etc.

(c) (*1 pt.*) an integer k

A true B false

(d) (*1 pt.*) The output to the decision version of Square is

A "yes," if F contains a $k \times k$ square of 1s

B "yes," if F contains k flowers

C a single integer, which is the size of the largest square of 1s in F

D the coordinates of the upper left corner of a maximal square of 1s

10. *Membership in NP.* The decision version of Square is easily seen to be in NP, because it admits a certificate.

(a) (*2 pt.*) The certificate includes, (apart from the input already listed in answer above,) the following information

A "yes," if F contains a square of 1s of size $k \times k$

B "yes," if the solution can be found in polynomial time

C the coordinates of the upper left corner of a $k \times k$ square

D k^2 points (i, j) such that $F[i, j] = 1$.

(b) (*2 pt.*) The length of the certificate is (choose the smallest possible)

A $O(n + m)$ numbers B $O(k^2)$ numbers C $O(nm)$ numbers D $O(1)$ numbers

(c) (*2 pt.*) The certificate can be checked in time (choose the smallest possible)

A $O(n + m)$ B $O(k^2)$ C $O(nm)$ D $O(1)$

NP-hardness

One of the problems in the set is NP-hard.[3]

11.

(a) (*2 pt.*) The following problem (called P_1) is NP-hard:

A Sell! B Square C EU D Dance

(b) (*3 pt.*) The easiest way to see that is to take the following NP-hard problem, called P_2,

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

(c) (*2 pt.*) and prove

A $P_1 \leq_P P_2$ B $P_2 \leq_P P_1$

(d) (*1 pt.*) For this, an arbitrary instance of

A Graph colouring B 3-dim. matching C Independent set D Set Cover
E Vertex cover F 3-satisfiability G Travelling salesman H Hamiltonian path
I Sell! J Square K EU L Dance

(e) (*1 pt.*) is transformed into an instance of

A Graph colouring B 3-dim. matching C Independent set D Set Cover
E Vertex cover F 3-satisfiability G Travelling salesman H Hamiltonian path
I Sell! J Square K EU L Dance

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

(f) (*4 pt.*) Describe the reduction on the back of this page, or on a separate piece of paper. Do this both in general and for a small but complete example. In particular, be ridiculously precise about the paramters 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.