Exam SALD – Algorithm Design

4 January 2011, 9.00–13.00, 2A12, 2A14

Thore Husfeldt, ITU

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.

Organisation Pages 2–6 describe a number of algorithmic problems. Questions 4–9 are about these problems. Questions 1–3 are not.

Filling out the exam Some questions (1–3) 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.

Questions 4–9 are free form, and you answer these on separate paper; marking the answers very clearly with the exercise numbers. I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. Use English or Danish. 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, 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.

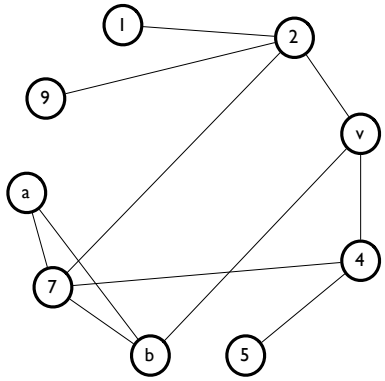
If you want to know the precise formula, if the question has k choices, and you checked a boxes, your score is $\log(k/a)$, provided you checked the correct answer, and $-a\log(k/a)/(k-a)$ if you only checked wrong answers. Moreover, I have weighted the questions by relevance (not necessarily difficulty),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

Viapath

Let $G = (V, E)$ be an undirected graph on $n \geq 3$ vertices with m edges. The vertex set V includes three special vertices a, v, and b.



Find a simple path from a to b via v if it exists. (A simple path is a path without repeated vertices.)

In the example above, a valid solution exists: $a, 7, 2, v, b$. Note that the path $a, 7, 2, v, 4, 7, b$ is not valid because it is not simple. Note that the paths a, b and $a, 7, b$ are not valid because they do not include v.

Input

The input starts with a line containing n, the number of vertices. The next line contains the indices of the three special vertices a, b, and v in that order. (Let us agree that the vertices are numbered $1, 2, \ldots, n$.) The n following lines describe the adjacency matrix of G, such that $c(i, j)$ is 1 if there is an edge from i to j, and 0 otherwise.

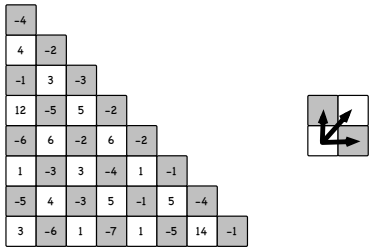
Output

A list of vertices forming a simple path from a to b via v, or the word "impossible" if no such path exists.

input	**output**
9 8 6 3 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0	8 7 2 3 6

Chesspath

Take an $n \times n$ chessboard and saw it off at the diagonal. So the legal positions are (i, j) for $i \geq 1, j \geq 1$ and $i + j \leq n + 1$. At each position (i, j) is a number of coins $c(i, j)$; the amount can be negative or positive, but never 0. On the black squares it is always negative, on the white squares it is always positive. The lower left corner $(1, 1)$ is white.



Your task is to move from $(1, 1)$ until you fall off the board, you can only move up and to the right, so from (i, j) you can only go to $(i + 1, j)$, $(i, j + 1)$, or $(i + 1, j + 1)$ (if the square does not exist, you fall off the board and are finished). You start with 0 gold pieces. At (i, j) you add $c(i, j)$ to your gold.

You have to get off the board with as much gold as possible.

Input

The input consists of a line containing n followed by n lines of integers describing $c(i, j)$ in the obvious manner.

Output

A list of positions (i, j) separated by commas, describing an optimal path. The first position is $(1, 1)$, and the last position is off the board.

Example 1

Input

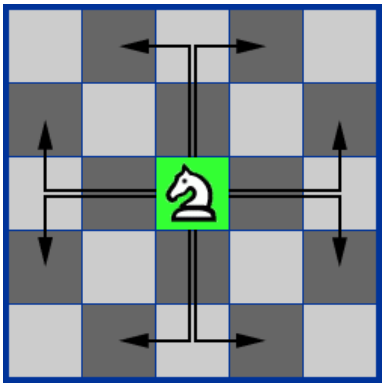
```
2
-4
5 -4
```

Output

```
(1,1), (2,2)
```

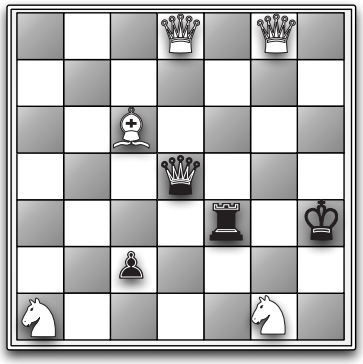
Superknight

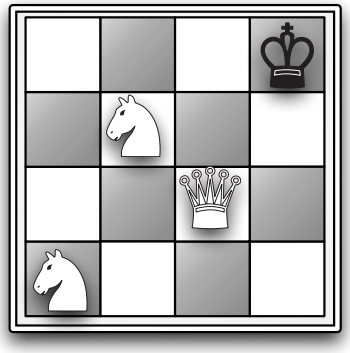
Imagine an $n \times n$ chess board. You have a white knight in the bottom left corner. Your knight follows the normal chess rules for movement, so it can go to 8 other positions (provided it doesn't fall off the board), as illustrated here:



To be precise, from (x, y) the knight can go to $(x+1, y-2)$, $(x+1, y+2)$, $(x-1, y-2)$, $(x-1, y+2)$, $(x+2, y-1)$, $(x+2, y+1)$, $(x-2, y-1)$, and $(x-2, y+1)$ in one step. (It jumps over other pieces.)

Your knight is *super* because it can move as often as you want. When it lands on an enemy (black) piece, the enemy piece is removed. You cannot land on friendly (white) pieces. The goal is to kill the enemy king with as few moves as possible using only your superknight.





In the example the left, you can kill the black king using 4 moves with your superknight. In the example to the right, you can't kill the black king using only your superknight.

Input

The input consists of a line containing n, the number of rows and columns, followed by n lines encoding each row, using letters K for king, Q for queen, R for rook, B for bishop, P for pawn, S for knight. Black pieces are given in upper case (KQRBSP), white pieces are in lower case (kqrbsp). Empty positions are given by a full stop. The white superknight is in the bottom left corner. There is only one black king. No other chess rules are used, so the board can be completely nonsensical from a chess player's point of view.

Output

An optimal sequence of moves from the white superknight that kills the black king, or the word "impossible" if no such sequence exists. The sequence is given as column–row co-ordinate pairs, numbering the rows and columns using $0, \ldots, n-1$ from the bottom left corner.

Example 1

Input

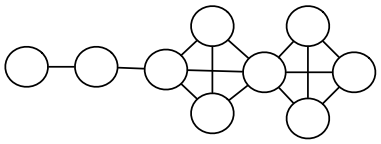
```
7
...q.q.
.......
..b....
...Q...
....R.K
..P....
s....s.
```

Output

```
(0,0), (2,1), (4,2), (5,4), (6,2)
```

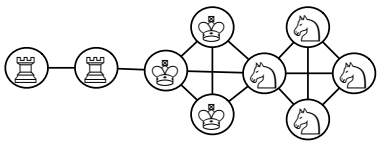
Graphchess

Let $G = (V, E)$ be an undirected graph, like this:



You have to place chess pieces of three different kinds (rook, king, knight)) on the vertices of the graph under the following rules: (1) Each vertex gets exactly one piece. (2) If two vertices receive the same kind of piece, then there must be an edge between them. (3) The number of rooks is at most the number of kings, which is at most the number of knights.

Here's a solution for the above graph:



Input

The input consists of a line containing n, the number of vertices. Then follow n lines containing the adjecency matrix of G, in the obvious manner, $c(i, j)$ is 1 if there is an edge between i and j, and 0 otherwise.

Output

Output 1 if the input has a solution, and 0 if not.

Example 1

Input

```
5
0 1 0 0 0
1 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Output

```
0
```

Exam Questions

Analysis of algorithms

1. Let $f(n) = (n^2 + 2n + \frac{1}{1000}n^{3/2}) \log n$. 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: int f(int n) {
2:     if n > 3: return f(n − 1) − f(n − 3);
3:     else: return 3;
4: }
```

(a) (*1 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 $T(n) = T(n-1) - T(n-3) + O(1)$

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

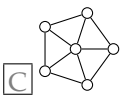
(b) (*1 pt.*) Find the solution to $T(n) = T(n/2) + T(n/2) + 27n \log^2 n$, $T(1) = 2$. Give the smallest correct answer.

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

3. For integer $r \geq 1$, the rth *star graph* $S_r = (V_r, E_r)$ consists of a center vertex v_1, connected to $(r-1)$ other vertices $v_2, \ldots, v_r$.

(a) (*1 pt.*) How does S_5 look?

A  B  C  D 

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

A $O(1)$ B $\binom{r}{2}$ C $r-1$ D r

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

A True B False

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

A True B False

(e) (*1 pt.*) S_r has a maximum independent set of size (choose the smallest correct estimate)

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

Graph connectivity

4. One of the problems can be efficiently reduced to undirected, unweighted graph connectivity (so it an be solved by something like DFS or BFS, for example):

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

(b) (*4 pt.*) Explain how the connectivity instance is constructed. What are the vertices, and how many are there? What are the edges, and how many are there? Draw an example of a vertex in the connectivity instance, including all vertices it is connected to (you don't have to draw the whole graph.)

Explain which connectivity algorithm you use. State the running time of your algorithm in terms of the original parameters. (It must be polynomial in the original size.)

Dynamic programming

5. One of the problems is solved by dynamic programming. (But not by a simple BFS or DFS search.)

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

(b) (*5 pt.*) Following the book's notation, we let OPT(i, j) denote the value of a partial solution. Give a recurrence relation for OPT, including relevant boundary conditions and base cases. State the running time of the resulting algorithm.

Network flow

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

(a) (*2 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 $O(n^{17} \log^{-3} \epsilon + log^2 m)$.")[1]

Computational complexity

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

7.

(a) (*1 pt.*) Formulate Viapath as a decision problem. What are the inputs? What is the output?

(b) (*2 pt.*) Show that the decision version of Viapath belongs to NP by describing a certificate. In particular, give an example of such a certificate for a small instance. How long is this certificate in terms of n and m?

(c) (*1 pt.*) Describe very briefly how your certificate can be checked. In particular, what is the running time of that procedure?

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

NP-hardness

One of the problems in the set is NP-hard.[2]

8.

(a) (*2 pt.*) Which one? (Let's call it P_1.)

(b) (*3 pt.*) The easiest way to see that is to consider another NP-hard problem (called P_2). Which one?

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

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

(d) (*1 pt.*) Do you now need to prove $P_1 \leq_P P_2$ or $P_2 \leq_P P_1$?

(e) (*4 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 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.

Greedy, approximation

I want to solve Chesspath greedily, like this: Always go to the position with the most coins, i.e., to the one that maximises $c(i+1,j)$, $c(i,j+1)$, $c(i+1,j+1)$.

9.

(a) (*2 pt.*) Show that this algorithm is not optimal by giving a concrete and complete example.

(b) (*2 pt.*) Is this algorithm a 2-approximation algorithm? (In other words, does it find a solution at least half as good as the optimum?) If yes, give a short proof. If no, give a small (but concrete and complete) counterexample.

(c) (*2 pt.*) Can you find another c for which this algorithm is a c-approximation algorithm? If yes, give a short proof. If no, give a small (but concrete and complete) counterexample.

[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.