Exam EDAF05

25 May 2011, 8.00–13.00, Vic1

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 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 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.

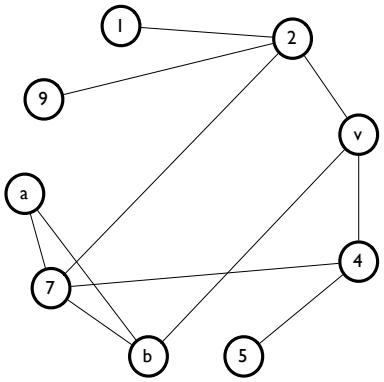
If you want to know the precise formula: if the question has k choices, and you checked a boxes, your score is $\log(k/a)$, provided you checked the correct answer, and $-a \log(k/a)/(k-a)$ if you only checked wrong answers. Moreover, I have weighted the questions by relevance (not necessarily difficulty), 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. The task is to find a simple path starting in vertex a, passing through vertex v, and ending in vertex b. (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 in V are numbered $1, 2, \ldots, n$.) The following n 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

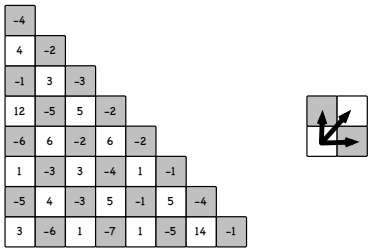
```
9
8 6 3
0 1 0 0 0 0 0 0 0
1 0 1 0 0 0 1 0 1
0 1 0 1 0 1 0 0 0
0 0 1 0 1 0 1 0 0
0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 1 1 0
0 1 0 1 0 1 0 1 0
0 0 0 0 0 1 1 0 0
0 1 0 0 0 0 0 0 0
```

output

```
8 7 2 3 6
```

Chesspath

Consider an $n \times n$ chessboard that has been cut off at the diagonal. In other words, the legal positions are (i, j) for $i \geq 1, j \geq 1$ and $i + j \leq n + 1$. At each position (i, j) I have placed a value $c(i, j)$; 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

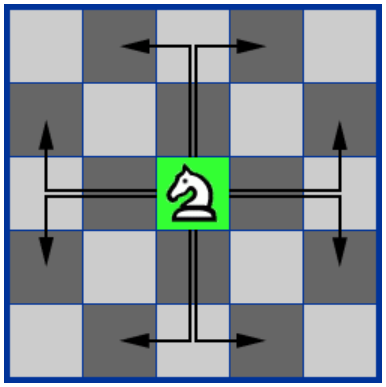
```
2
-4
5 -4
```

Output

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

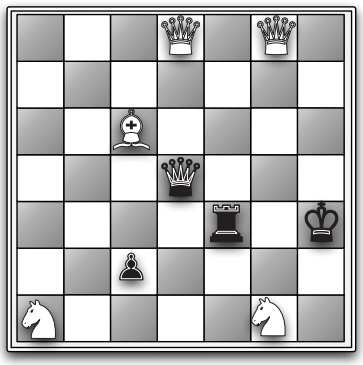
Superknight

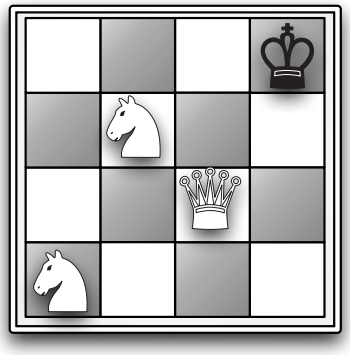
Imagine an $n \times n$ chess board. You have a white knight in the bottom left corner. Your knight follows the normal chess rules for movement, so it can go to 8 other positions (provided it doesn't fall off the board), as illustrated here:



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, the superknight can kill the black king using 4 moves. In the example to the right, the superknight can't kill the black king.

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)
```

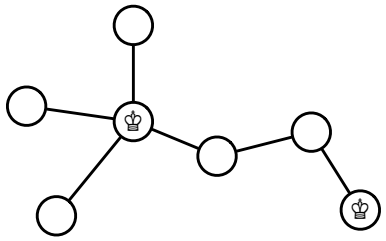
Graphkings

Let $G = (V, E)$ be an undirected graph.

You have to place as many Kings as possible on the graph. The rules for placement of the Kings are:

- There can be at most one King per vertex.
- No King may be placed "next to" another King. (This rule is inspired by the board game chess.) To be precise: If a King is placed on vertex u, then you cannot place another King on any vertex neighbour of u.

Here's a valid but not optimal placement of kings:



There is a way to place $k = 5$ Kings in this graph.

Input

The input consists of a line containing n and m, the number of vertices and edges. Then follows and integer k on a separate line. Then follow m lines containing the edges of G as pairs of integers, we use the convention that $V = \{1, 2, \ldots, n\}$.

Output

Output 1 if k Kings can be placed on G, and 0 if not.

Example 1

Input

```
7 6
5
1 3
2 3
3 4
3 5
5 6
6 7
```

Output

```
1
```

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: for i = 1 to n:
2:     {
3:     j = 1;
4:     while j ≤ n:
5:             j = j + 2;
6:     }
```

(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:     {
3:     insert (i, S);
4:     remove (S);
5:     }
```

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

(b) (*1 pt.*) Assume I want the total running time to be $O(n / \log n)$. 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) {
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 $T(n) = T(n-1) + O(n)$

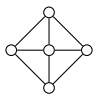
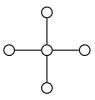
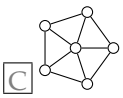
(b) (*1 pt.*) Find the solution to $T(n) = T(n/2) + T(n/2) + 27n\log^2 n, \quad T(1) = 2$. Give the smallest correct answer.

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 *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 $O(r)$

Greedy

I want to solve Chesspath greedily, like this: Always go to the position (out of three possible) with the largest value on it.

6.

(a) (*2 pt.*) Show that this algorithm is not optimal by drawing a concrete, small, and complete example instance here, on the back of the page, or on separate piece of paper:

Graph connectivity

7.

(a) (*2 pt.*) The following problem can be efficiently reduced to undirected, unweighted graph connectivity (so it an be solved by something like DFS or BFS, for example):

A Viapath B Chesspath C Superknight D Graphkings

(b) (*4 pt.*) On the back of this page, or on a separate piece of paper, 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 instance, but you can if you want.)
State the running time of your algorithm in terms of the original parameters. (It must be polynomial in the original size.)

Dynamic programming

8.

(a) (*2 pt.*) One of the problems is solved by dynamic programming. (But not by a simple BFS or DFS search.)

A Viapath B Chesspath C Superknight D Graphkings

(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

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

[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.

B

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

C

$$\mathrm{OPT}(i,j) = c(i,j) + \min\{\,\mathrm{OPT}(i-1,j), \mathrm{OPT}(i-1,j-1), \mathrm{OPT}(i,j-1)\,\}$$

D

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

E

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

F

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

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

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

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

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

Network flow

9.

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

A Viapath B Chesspath C Superknight D Graphkings

(b) (*1 pt.*) The network consists of a node for[2]

A every vertex in the original graph
B every board square
C every chess piece
D every edge in the original graph

(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 $O(n)$ D $O(n^2)$

(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?

[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 an extra node, or a chess piece, etc.

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.

10. *(Decision version.)* The input to the decision version of Viapath includes

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

A true B false

(b) (*1 pt.*) the indices of a, b, and v

A true B false

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

A true B false

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

A "yes," if G contains a simple path from a to b via v

B a list of vertices that form a simple path from a to b via v

C the length of the shortest simple path from a to b via v

D "yes" if G contains a simple path from a to b via v using at most k vertices

11. *Membership in NP.* The decision version of Viapath 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 G contains a simple path from a to b via v

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

C a sequence of vertex indices describing a simple path from a to b via v

D a vertex index w such that G contains a simple path from a to b via w

(b) (*1 pt.*) The length of the certificate is (choose the smallest possible)

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

(c) (*1 pt.*) The certificate can be checked in time (choose the smallest possible)

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

NP-hardness

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

12.

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

A Viapath B Chesspath C Superknight D Graphkings

(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

[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.

(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] Viapath [J] Chesspath [K] Super Vector Mario! [L] Graphchess

(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] Viapath [J] Chesspath [K] Super Vector Mario! [L] Graphchess

(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. I cannot stress how imporant such an example is: do it! 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.