Exam EDAF05

25 August 2011

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.

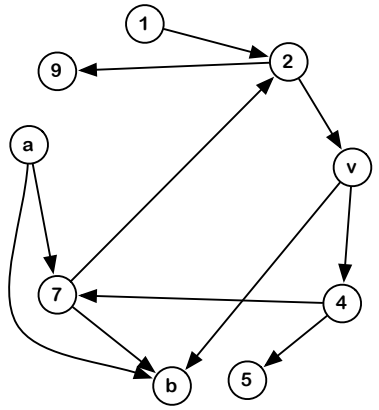
If you want to know the precise formula: if the question has k choices, and you checked a boxes, your score is $\log(k/a)$, provided you checked the correct answer, and $-a\log(k/a)/(k-a)$ if you only checked wrong answers. Moreover, I have weighted the questions by relevance (not necessarily difficulty),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

Manypath

Let $G = (V, E)$ be a directed graph on $n \geq 2$ vertices with m edges. The vertex set V includes two special vertices a and b. The task is to find as many paths from a to b as possible. The paths may not use any vertices twice, except for a and b. (Another way of saying that is that the paths are simple and internally vertex-disjoint.)



The above example has a solution of size 2. For example (a, b) and $(a, 7, 2, v, b)$.

Another solution of size two is (a, b) and $(a, 7, b)$.

Note that the graph conains three different paths from a to b, but they do not form a valid solution of size three since they use the vertex 7 twice.

Input

The input starts with a line containing n, the number of vertices. The next line contains the indices of the two special vertices a, b. (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.

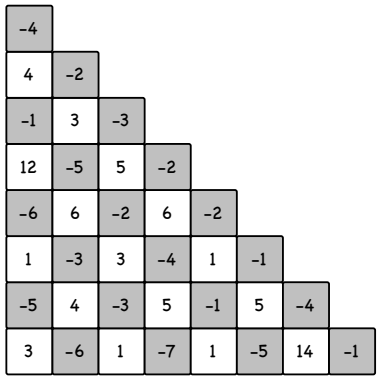
Output

A list of vertex indices describing a maximal number of paths from a to b.

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

Knightpath

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 move like a knight in chess, but only northeast. More formally, from (i, j) you can go to $(i + 1, j + 2)$ or to $(i + 2, j + 1)$. (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

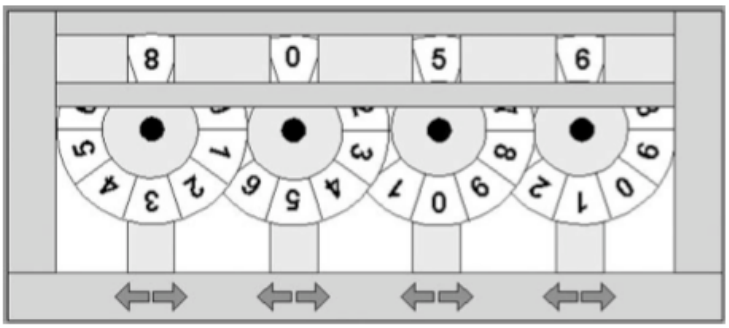
```
8
-4
4 -2
-1 3 -3
12 -5 5 -2
-6 6 -2 6 -2
1 -3 3 -4 1 -1
-5 4 -3 5 -1 5 -4
3 -6 1 -7 1 -5 14 -1
```

Output

```
(1,1),(2,3),(4,4),(5,6)
```

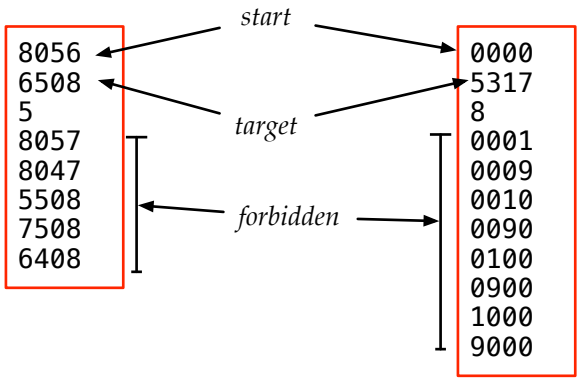
Superwheels

Consider a machine consisting of n wheels, like this (for $n = 4$):



Digits ranging from 0 to 9 are printed consecutively (clockwise) on the periphery of each wheel. The topmost digits of the wheels form a four-digit integer. For example, in the above figure the wheels form the integer 8056. Each wheel has two buttons associated with it. Pressing the button marked with a left arrow rotates the wheel one digit in the clockwise direction and pressing the one marked with the right arrow rotates it by one digit in the opposite direction.

The input consists of a start configuration, a target configuration, an integer k, and k forbidden configurations. Here are two small examples for $n = 4$.



Typically, k could be a few thousand for $n = 4$, but I can't be bothered to draw such an example. Your job is to write a program to calculate the minimum number of button presses required to transform the initial configuration to the target configuration without passing through a forbidden configuration, or report that no solution is possible. For example, in the left example, the output should be "14", and in the right example it should be "impossible".

ChessHate

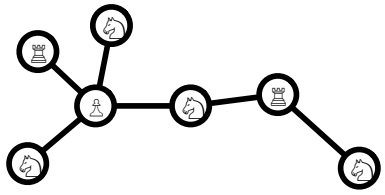
Let $G = (V, E)$ be an undirected graph.

You have to place chess pieces of three different types (Rook, Pawn, Knight) on G. The rules are as follows:

- Every vertex receives exactly one piece.
- Pieces of different types can never be placed on neighbouring vertices.

(You have as many pieces as you need of each type.)

Here's a valid placement on a small graph:



Input

The input consists of a line containing n and m, the number of vertices and edges. 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 there is a valid way to place pieces on G, and 0 if not.

Example 1

Input

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

Output

```
1
```

Exam Questions

Analysis of algorithms

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

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

f 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 + j;
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 $O(1)$. 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 $O(\sqrt{n})$. 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.

Greedy

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

4.

(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

5.

(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 Manypath B Knightpath C Superwheels D ChessHate

(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? (A small, complete example is normally the best way of explaining things. Please include one.)

State the running time of your algorithm in terms of the original parameters. Be careful about which letter you use for the number of vertices.

Dynamic programming

6.

(a) (*1 pt.*) One of the problems is solved by dynamic programming. (But not by a simple BFS or DFS search.)

A Manypath B Knightpath C Superwheels D ChessHate

(b) (*4 pt.*) Following the book's notation, we let $\mathrm{OPT}(i, j)$ denote the value of a partial solution. Define your dynamic programming solution by giving a recurrence relation for OPT.

(c) (*1 pt.*) State the running time and space of your solution.

Network flow

7.

(a) (*1 pt.*) One of the problems in the set is easily solved by a reduction to network flow. That problem is

A Manypath B Knightpath C Superwheels D ChessHate

(b) (*5 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 *Manypath*, and include questions about NP. Don't take this as an indication that Manypath may or may not be NP-hard.

8. *(Decision version.)* The input to the decision version of Manypath 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 Manypath 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

9. *Membership in NP.* The decision version of Manypath 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.[1]

10.

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

A Manypath B Knightpath C Superwheels D ChessHate

(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 Manypath J Knightpath K Superwheels L ChessHate

(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 Manypath J Knightpath K Superwheels L ChessHate

(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.

[1] 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.