Exam EDAF05

10 January 2012, 8.00–13.00, Sparta:A–D

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 Some 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

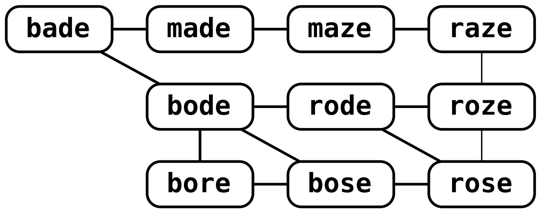
Introduction. There are four algorithmic problems in this set; all of them are variants on the same basic structure defined as follows.

Two words x and y are *neighbours* if

1. they have the same number of letters, i.e., $x = x_1, \ldots, x_k$ and $y = y_1, \ldots, y_k$,
2. the agree on exactly $k - 1$ positions, i.e., $x_i = y_i$ holds for $k - 1$ choices of i.

For example, "EXAM" and "ERAM" are neighbours, but "EXAM" and "EXOL" are not. Note that the letters must have the same positions, so "EXAM" and "AEMY" are not neighbours (even though they have 3 letters in common). Let's agree that a word is not its own neighbour.

To fix notation, we will maintain that the input consists of n such words. The words all have the same number of letters, k. The words define an undirected, simple, unweighted graph $G = (V, E)$ whose vertices are the words and whose edges are the neighbour relation. Here is the graph defined by the words *bade*, *bode*, *bore*, *bose*, *made*, *maze*, *raze*, *rode*, *rose*, and *roze*:



In your answer, I will assume that you use the terms *word*, *graph*, *vertex*, *edge*, *neighbour*, and the symbols G, V, E, k, and n in the way defined above, unless you explicitly define them to mean something else.

There is no requirement that the words make sense. They could be "xxxxyz" or some other nonsense. However, you can assume that the alphabet is finite, so let's just agree that the input words are always taken from the standard 24 letter English alphabet, so that we avoid silly ideas like encoding a 3-Sat instance in a single Chinese (or Klingon) symbol.

Ladder

Given two words s and t, find the shortest sequence of neighbouring words connecting s to t.

Input

The input starts with a line containing n, the number of words. The next n lines each contain a word, in alphabetical order. Exactly two of the words are marked with a star; these are s and t.

Output

Output the shortest sequence of neighbouring words from s to t, separated by "`-> `". If there is no such sequence, output the word "`impossible`".

Input:

```
10
bade
bode
bore
bose *
made *
maze
raze
rode
rose
roze
```

Output:

```
bose -> bode -> bade -> made
```

Input:

```
3
bade *
bode
xxya *
```

Output:

```
impossible
```

Cycle

Given two words s and t, find a *simple cycle* of neighbouring words passing through s and t. (A simple cycle is a sequence of words $w_1, w_2, \ldots, w_r$ such that $w_1 = w_r$ and no other words appear twice in the sequence. It does not have to be the shortest such cycle.)

Input

Exactly as for Ladder.

Output

If there is a simple cycle of neighbouring words containing both s and t, output the cycle as a sequence of words separated by "`-> `". The first and the last words in that sequence must be the same. No other words may appear more than once in the sequence. Both s and t must appear in the sequence.

If there is no such sequence, output the word "`impossible`".

Input:

```
10
bade
bode
bore
bose *
made *
maze
raze
rode
rose
roze
```

Output:

```
bade -> made -> maze -> raze -> roze
-> rode -> rose -> bose -> bore ->
bode -> bade
```

Input:

```
3
bade *
bode
bose
bore *
```

Output:

```
impossible
```

Manystars

Given *m* “starred” words, find a simple path connecting all of them. (The path does not have to be the shortest. The order in which the starred words are visited is not important.)

Input

The input starts with the number of words *n* in the dictionary, followed by the number *m* of starred words. Then follow the *n* words that make up the dictionary, in alphabetical order. Exactly *m* of these words are starred.

Output

If there is a simple path through the dictionary containing all the starred words, print such a path, separated by `->`. Otherwise print `impossible`.

Input:

```
10
3
bade *
bode *
bore
bose
made
maze
raze *
rode
rose
roze
```

Output:

```
bade -> made -> maze -> raze -> roze
-> rode -> bode
```

Input:

```
4
3
bode *
rode
rose *
roze *
```

Output:

```
impossible
```

Alphabetic

An *alphabetic ladder* is a sequence of neighbouring words that is alphabetically ordered. You have to find an alphabetic ladder that includes as many starred words as possible.

Input

Exactly as for Manystars.

Output

A sequence of neighbouring words in alphabetic order, separated by `->`, containing as many starred words as possible.

Input:

```
10
3
bade
bode *
bore
bose *
made *
maze *
raze *
rode *
rose
roze *
```

Output:

```
made -> maze -> raze -> roze
```

Exam Questions

Analysis of algorithms

1. Let $f(n) = (n^2 + n^3)\log n$. True of false?

(a) *(1 pt.)* $f(n) = O(n^3 \log n)$

$\boxed{\text{A}}$ true $\boxed{\text{B}}$ false

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

$\boxed{\text{A}}$ true $\boxed{\text{B}}$ false

(c) *(1 pt.)* $f(k) = O(k^3)$

$\boxed{\text{A}}$ true $\boxed{\text{B}}$ false

2. Consider the following piece of code:

```
1: for $i = 1$ to $n$:
2:     for $j = 1$ to $n$:
3:         for $k = 1$ to 10:
4:             print $k$;
```

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

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

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 **insert** (i, S) takes constant time and **remove**(S) takes time $O(\log |S|)$. Then the total running time is: (Choose the smallest correct estimate.)

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

(b) *(1 pt.)* Assume I change line 4 so that the remove operation is only executed if i is even. (So, every second time.) Like this:

```
4:     if ($i \mod 2 = 0$): remove ($S$);
```

What is the total running time?

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

4. Consider the following piece of code:

```
1: int f(int n) {
2:     if n > 1: return f(n − 1) + f(n − 2);
3:     else: return 1;
4: }
```

(a) (*1 pt.*) Which recurrence relation best characterises the running time of this method?

- [A] $T(n) = T(n-1) + T(n-2) + O(1)$
- [B] $T(n) = T(n-1) \cdot T(n-2) + O(1)$
- [C] $T(n) = T(n-1) - T(n-2) + O(1)$
- [D] $T(n) = T(n-1) + O(n)$

Greedy

I want to solve Alphabetic greedily, like this: Start at the alphabetically first word. From here, consider the set of neighbouring words N that are alphabetically after the current word. If any of the words in N has a star, go to that word. Otherwise go to an arbitrary word in N.

5.

(a) (*2 pt.*) Show that this algorithm is not optimal by drawing a concrete, small, and complete example instance.

Graph connectivity

One of the four problems in the set can be solved as a simple graph connectivity problem; so it can be solved by an algorithm like BFS or DFS.

6.

(a) (*6 pt.*) Which problem can be solved as a graph connectivity problem, and how? In you answer, address all the following questions. Your answer must be short and precise.

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.)

Which algorithm are you using? Why? (Could you use another algorithm?) State the running time of your algorithm in terms of the original parameters. (The running time must be polynomial in the original size.)

Dynamic programming

One of the four problems in the set has a straightforward dynamic programming solution. (But not a simple BFS/DFS or greedy solution.)

7.

(a) (*6 pt.*) Which problem can be solved using dynamic programming, and how? In you answer, address all the following questions. Your answer must be short and precise.

Following the book's notation, let $\mathrm{OPT}(\cdots)$ denote the value of a partial solution, and write an explicit recurrence relation for OPT. Make it clear what the arguments to OPT stand for. Do include the base and boundary cases.

What is the running time and the space usage of the resulting algorithm?

Network flow

One of the four problems in the set can be solved using a network flow algorithm.

8.

(a) (*6 pt.*) Which problem can be solved by a reduction to network flow, and how? In you answer, address all the following questions. Your answer must be short and precise.

Describe the reduction. 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?

Tell me which flow algorithm you are using (you don't have to motivate that choice) and what the resulting running time is.

Computational complexity

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

9.

(a) (*2 pt.*) Formulate a decision version of Alphabetic. What are the inputs, what is the output?

(b) (*3 pt.*) Show that the decision version is in NP by describing how a certificate for a yes-instance looks, and how the certificate can be checked. I recommend that you give *a complete example* of such a certificate. How long is the certificate, and how fast can it be checked?

NP-hardness

One of the problems in the set is NP-hard.[1]

10.

(a) (*6 pt.*) Which problem is NP-hard and why? In you answer, address all the following questions. Your answer must be short and precise.

Identify a known NP-hard problem and exhibit a reduction. Be ridiculously precise about that which problem is reduced to which, I *strongly recommend* to use the notation $P_1 \leq_P P_2$ and be very precise about which problem is P_1 and which is P_2.

Describe the reduction both in general and for a small but complete example. I cannot stress how imporant such an example is: do it! Be very precise about what is *given* and what is *constructed*. 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.