

DD1344 Grundläggande datalogi

Grudat07 Take Home Omtentamen

2008-05-19

Take Home Exam (tentamen)

NOTES: (Please read these carefully!)

(i) Clearly mark at the top of each sheet you use: (a) your name, (b) the page number.

(ii) On the front page indicate (a) how many pages are contained in your work in total. Clearly mark: (b) your name (c) your personnummer, (d) your e-mail address (in case I need to contact you)

(iii) There are two ways to submit your work. (a) Your work may be handed in to NADA's studentexpedition no later than **Tuesday 20th May 2008 at 12.00 am.** After this time it will be marked as late, and marks will be subtracted pro-rata. (b) If for any reason you are unable to reach the studentexpedition yourself (for example if you are at work) then you may post your manuscript to:

Studentexpeditionen, CSC, KTH, 100 44 Stockholm.

The date on the postmark will be taken as the date and time of your submission. The same deadline applies to manuscripts submitted by post.

I do not accept electronic submissions under any circumstances.

(iv) If manuscripts are submitted in any other place or by any other means than those described in Part (iii) then the examiner and CSC cannot be held responsible in the case that manuscripts are lost.

(v) If you have any questions about the exam (for example, if you do not understand a question) for the fastest response you should call my mobile number 076 223 86 79. Please do not call before 09.00 or after 17.00!

(vi) You may use your kursbunt, and any course books, as well as a dictionary. You are allowed to use the internet including search engines. However, any work which you download and use must be credited to its source. You are not allowed to discuss or develop your answer with anyone else before all manuscripts have been handed in. You are not allowed to copy anyone else's work. By handing in your manuscript you are declaring that you have abided by these rules. In the case that cheating is suspected, actions will be taken against all students involved.

(vii) Write clearly. No marks will be awarded for work that I cannot read. You may write your answers in Swedish or in English.

(viii) Model answers will be placed on the course web page, but not before all students who are doing komplettering have finished and submitted their work satisfactorily.

Marking Scheme

This paper is divided into two parts I and part II.

The grades D and E, can be achieved by answering part I questions only to the level of 80% and 70% of the available marks respectively.

The grade Fx (which allows komplettering) requires 60% of the available marks on part I. The grades A, B and C are achieved by obtaining at least 80% on part I, and respectively 60%, 40% and 20% of the marks available on part II.

Part I questions cover basic material from the course, drawn from lectures 2 to 6 (as described on the course web page). Part II questions cover the more advanced material of the course drawn from lectures 7 to 11.

You should use the Python syntax for any algorithms which you are asked to design. Be aware that any syntax errors you make may result in your algorithm design being wrong in which case marks will be deducted accordingly. Minor and unimportant syntax errors may however be ignored by the examiner.

PART I

Question 1. (total 12 points)

According to the *Fundamental Theorem of Algebra*, every polynomial

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$$

can be represented in the form

$$P(x) = a_n (x - x_1) (x - x_2) \dots (x - x_n),$$

where x_1, x_2, \dots, x_n are the roots of the polynomial (generally, complex and not necessarily distinct). By considering the number of operations required, discuss which of these two representations is more convenient for each of the following operations on polynomials:

- (a) **polynomial evaluation** at a given point (**5 points**)
- (b) **addition** of two polynomials (**5 points**)
- (c) **multiplication** of two polynomials. (**2 points**)

Question 2 (total 20 points)

(1) (**6 points**) The following algorithm represents an attempt to compute the number of leaves in a binary tree:

```
def LeafCounter(T):  
    if isEmptyTree(T): return 0  
    else: return LeafCounter(T.left) + LeafCounter(T.right)
```

Is this algorithm correct or incorrect? Motivate your answer in as much detail as possible. If the algorithm is incorrect is it possible to correct it and how?

(2) (**3 points**) The *integer exponential function* $\text{exp}(x, y) = x^y$ where x and y are integers can be recursively defined using just addition and multiplication by means of the identity

$$x^y = x \cdot x \cdot \dots \cdot x \text{ "y-times"}$$

Write a recursive divide and conquer algorithm to implement this exponential function.

(3) (**10 points**) Write an *efficient* recursive divide and conquer algorithm to implement the exponential function that uses the *minimum number* of additions and multiplications to compute this function.

(4) (**1 point**) What can you conclude about writing divide and conquer methods.

Question 3 (total 18 points)

(1) (15 points) An array of strings contains just the strings “white”, “red” and “blue” mixed up. Write an efficient algorithm which rearranges these colours to produce an array formed like the French flag, i.e. all “blue” strings followed by all “white” strings followed by all “red” strings.

(2) (3 points) Discuss the worst case time complexity of your algorithm and characterise it by a simple mathematical function if possible.

PART II

Question 4 (total 20 points)

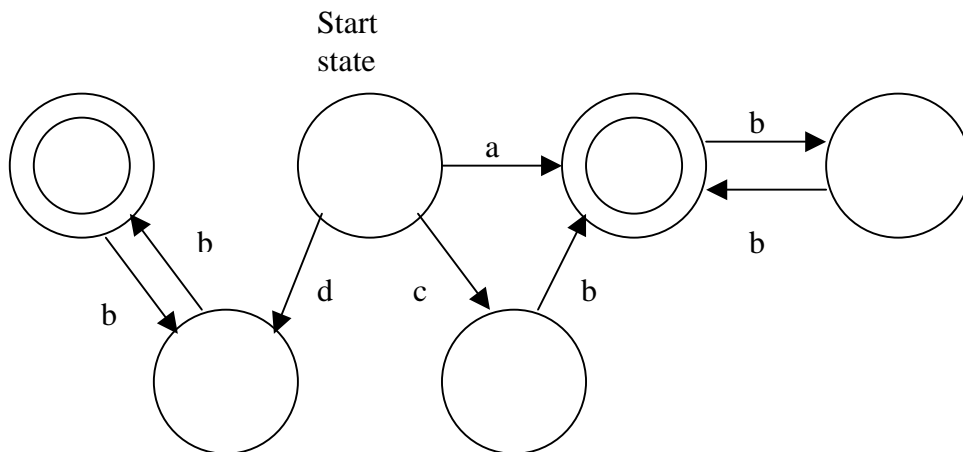
Given an n -by- m block of ice cream we wish to break it into a total of nm 1-by-1 square blocks by using the least number of cuts possible. We can cut the block only along a straight line, and we can only cut one block at a time.

(1) (15 points) Mathematically analyse what the minimum number of cuts must be. How many cuts are needed when $n = 5$ and $m = 5$?

(2) (5 points) Design an algorithm that breaks up the bar using this minimum number of cuts.

Question 5. (total 30 points)

Let A be the finite automaton defined in the diagram below



(a) (5 points) Give a recursive BNF definition of the language $L(A)$ accepted by A that does **not** make use of the Kleene star operator $(.)^*$.

- (b) **(6 points)** Give a regular BNF definition of the language $L(A)$ accepted by A that may use the Kleene star operator $(.)^*$, but does **not** use recursion. Make sure your regular BNF expression is as short and simple as possible by simplifying the expression where possible. Hence conclude that the language $L(A)$ is a regular language.
- (c) **(2 points)** With the help of your answer to part (a) or part (b) explain simply in words (Swedish or English) what the structure of words in the language $L(A)$ is like.
- (d) **(2 points)** Is the Kleene star operator as powerful as recursion when used to make BNF definitions of formal languages? Motivate your answer.
- (e) **(10 points)** Using your answer to part (a) write a recursive descent parser for the language $L(A)$ accepted by A using Python code to present your algorithm based on an appropriate set of parsing functions. You may assume that the input string to be parsed is stored in a queue.
- (f) **(5 points)** Using your answer to (b) draw a new finite automaton B that recognises the same language $L(A)$ which is recognised by A , but which simplifies the structure of A as much as possible and using the smallest number of states you can. Be sure to clearly label your state state and any accepting states.