UNIVERSITY of OSLO

Faculty of Mathematics and Natural Sciences

Exam in:	**INF 4130/9135: *Algoritmer: Design og effektivitet***
Date of exam:	**16th December 2013**
Exam hours:	**09:00 – 13:00 (4 hours)**
Exam paper consists of:	**5 pages (plus appendices)**
Appendices:	**2 pages (two copies, use one as scratch paper)**
Permitted materials:	**All written and printed**

Make sure that your copy of this examination paper is complete before answering.

The first assignment below is to be answered using the attachment in the appendix. There are two copies of the attachment, one for INF 4130 and one for INF 9135. Tear out and deliver the copy with the appropriate course code together with the white copy of your answers (and you can use the other copy of the attachment to sketch solutions).

Read the text carefully, and good luck!

Assignment 1 *String Search with Boyer-Moore and Knuth-Morris-Pratt* (*11 %*)

Question 1.a (5 %)
We use the simplified Boyer-Moore algorithm (called Horspool on the slides) to search for the pattern P = `idefix` in the string T = `abcdefgfhifjk`. Obviously we will never get a full match, but your task is to show how the pattern is shifted when the algorithm is run. Use (or copy) the table in the appendix to give your answer as a series of shifts, like the first shift shown in the example below (note that the example uses a different pattern). In the Reason column just write the letter and shift value that *caused* the shift on that line.

String													Reason
a	**b**	**c**	**d**	**e**	**f**	**g**	**f**	**h**	**i**	**f**	**j**	**k**	
a	a	.	.	.	.								Initial
	a	a	.	.	.	.							Shift(?)=1 *

* You must of course indicate the correct shift value of an actual letter.

Question 1.b (6 %)
We now use the Knuth-Morris-Pratt algorithm to search for the pattern P = ababac in the string T = ababbabaabbac. (Harder to make funny words with this one…) It is still obvious that we never get a full match, and your task is again to show how the pattern is shifted when the algorithm is run. Use (or copy) the table in the appendix to give your answer as a series of shifts, like the first shift shown in the example below. In the Reason column indicate where the mismatch occurs, and what the overlapping prefixes and suffixes that *caused* the shift on that line are (if there is any overlap). Assume that $|T| = n$ and $|P| = m$, and that T is indexed $T[0], \ldots, T[n\text{-}1]$ and P is indexed $P[0], \ldots, P[m\text{-}1]$.

String													Reason
a	**b**	**a**	**b**	**b**	**a**	**b**	**a**	**a**	**b**	**b**	**a**	**c**	
a	a	a	.	.	.								Initial
	a	a	a	.	.	.							$T[i]\neq P[j], P[x..y]=P[z..w]$*

* Again, you must of course indicate the correct values of actual letters and prefix/suffixes.

Assignment 2 *The A*-algorithm used on a new form of edit distance* (*28 %*)

In class we used dynamic programming to find the edit distance between strings when we were allowed to insert, delete and substitute letters, one at the time. We are now going to look at a slightly different type of edit operations, and use the A*-algorithm to find a new type of edit distance called the "substi-distance" (see below).

Let our alphabet be the letters A, B, C and D, and let our operations be four substitutions, as defined by the following four rules, all with cost 1. Note that we always substitute two *successive* letters with one or two new letters:

1: AB ⇒ BA
2: AD ⇒ BC
3: BC ⇒ D
4: CD ⇒ DC

Example: Starting with the string ADB we can substitute AD with BC (rule 2) to get BCB, and then substitute BC with D (rule 3) to get DB, and now we can't substitute any more.

Definition: We define the *substi-distance* from a string S to another string T as the length of the shortest sequence of the above four operations that will transform S to T. If it is impossible to transform S to T with the above four operations, the *substi-distance* is defined as infinite. (If T is longer than S, the distance is obviously infinite.)

In the example above the substi-distance from ADB to BCB is 1, and from ADB to DB it is 2 (and zero from ADB to itself). Starting with ADB those are the only strings we can make, so the substi-distance to all other strings is infinite.

Question 2.a (5 %)
Draw the directed graph of all possible strings we can make with the above rules when starting from the string ABCD. There shall be one and only one node v_s for every distinct string s we can make, and there will be an edge from node v_s to node v_t if there is a rule we can apply once to string s to get string t. Label the edges with 1, 2, 3 or 4, to indicate which rule was used (see the table above).

Question 2.b (5 %)
Answer the following questions:

i) What is the substi-distance from ABCD to BBCC ?
ii) What is the substi-distance from ABCD to BDC ?
iii) What is the substi-distance from BACD to BDC ?
iv) What is the substi-distance from BACD to DD ?

Question 2.c (6 %)
We now want to find the substi-distance from a string S to a string T, by searching for T in a graph corresponding to the one in 2.a, with the A*-algorithm. We then need a heuristic function, and use the difference in length between the current string and the goal string T as our heuristic. For a node v_u with string u in the graph $h(v_u) = \text{Abs}(|u| - |T|)$ will be our heuristic value. (The difference in length between u and T; we take the absolute value since in a general setting u may be both shorter and longer than T.)

Decide whether this heuristic is monotone! You need not give a formal proof, only a brief explanation of why or why not.

Question 2.d (6 %)
List the order (or one of the orders) in which the nodes are removed from the priority queue when we run the A*-algorithm to find the substi-distance from ABCD to DD with the heuristic of 2.b. You shall simply list the strings, starting with ABCD.

Question 2.e (6 %)
Answer the following questions:

i) What will happen to the monotonicity of the above heuristic if we add substitution rules that transform a string into one which is longer?
ii) What will happen to the monotonicity of the above heuristic if we add substitution rules that transform a string into one which is at least two shorter?

Assignment 3 *Undecidability* (*12 %*)

Which of the following languages are undecidable? Sketch proofs for your answers.

Question 3.a (6 %)
$L_1 = \{M \mid$ Turing machine M does not halt for any input$\}$

Question 3.b (6 %)
$L_2 = \{M \mid$ Turing machine M decides $L_1\}$

Assignment 4 *Complexity* (*12 %*)

We have seen in class that Hamiltonicity – to determine whether the graph given as input has a simple cycle containing *all* vertices of the graph – is an NP-complete problem. (By a 'simple cycle' we mean a cycle where no repetition of vertices or edges is allowed, except for the starting and ending vertex.) What is the complexity of the following problems? Give complete proofs.

Question 4.a (6 %)
To determine whether the graph given as input has a simple cycle containing at least one half of its vertices.

Question 4.b (6 %) (***Not straight forward!*** *You might want to look at this as the last one.*)
To determine whether the graph given as input has a simple cycle containing at most one half of its vertices.

Assignment 5 *Understanding concepts and ideas* (*20 %*)

Give short (no longer than three sentences) answers to the following questions:

Question 5.a (5 %)
When discussing complexity of algorithms, we model problems by formal languages. In what way is an ordinary problem (consisting of input-output pairs) represented as a formal language?

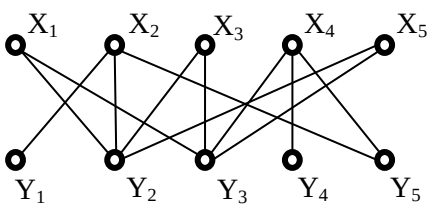
Question 5.b (5 %)
What is Church's, or the Church-Turing, thesis? Why is this a 'thesis' and not a theorem?

Question 5.c (5 %)
Name several strategies that are available for coping with intractability ('solving', in some specified, efficient way, NP-hard and harder problems).

Question 5.d (5 %)
Can we solve NP-complete problems efficiently by using parallel computers? Explain!

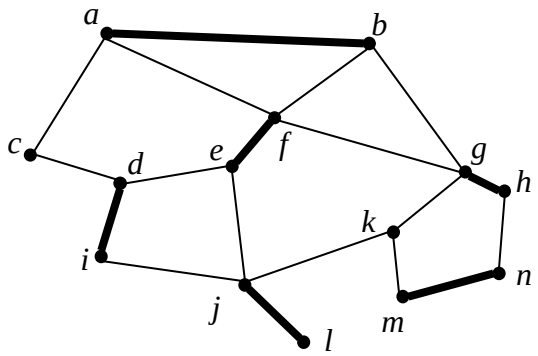
Assignment 6 *Matchings* (*17 %*)

Question 6.a (6 %)
We look at the following bipartite graph:



Find the largest number of edges a matching can have in this graph, and explain why your answer is correct. You don't have to show how you found the elements that go into your explanation.

Question 6.b (6 %)
We have the following graph, with a given matching *M* (marked with thick edges).



We will look for a larger matching, and use the Extended Hungarian Algorithm described on the slides. We start by making the unmatched node *k* the root, and start building an alternating tree by looking at the edges in the following order:

k-g, k-j, k-m, h-n, g-f, e-d, e-j

For each edge you look at you should complete the step according to the algorithm, and sometimes this will include that a set of nodes is merged (or collapsed) into one node. When a node occurs as an end node of an edge in the list above and this node is already merged with others, we shall mean the whole merged node it has become a part of.

The question is: How many mergings will occur during the steps given above, and what node-sets are merged?

Question 6.c (5 %)
Is there a larger matching than *M* in the graph? If so, draw the graph with the larger matching.

[END]

Attachment for answering Assignment 1

Side nr.:_____

Kandidatnr. : _____
Emne : **INF 4130**
Dato : **16.12.2013**

Answer 1.a

String													Reason
a	**b**	**c**	**d**	**e**	**f**	**g**	**f**	**h**	**i**	**f**	**j**	**k**	

Answer 1.b

String													Reason
a	**b**	**a**	**b**	**b**	**a**	**b**	**a**	**a**	**b**	**b**	**a**	**c**	

Attachment for answering Assignment 1

Side nr.:_____

Kandidatnr. : _____
Emne : **INF 9135**
Dato : **16.12.2013**

Answer 1.a

String													Reason
a	**b**	**c**	**d**	**e**	**f**	**g**	**f**	**h**	**i**	**f**	**j**	**k**	

Answer 1.b

String													Reason
a	**b**	**a**	**b**	**b**	**a**	**b**	**a**	**a**	**b**	**b**	**a**	**c**	