UNIVERSITY of OSLO

Faculty of Mathematics and Natural Sciences

Exam in:	**INF 4130/9135:** ***Algorithms: Design and efficiency***
Date of exam:	**14th December 2015**
Exam hours:	**09:00 – 13:00 (4 hours)**
Exam paper consists of:	**6 pages**
Appendices:	**None**
Permitted materials:	**All written and printed**

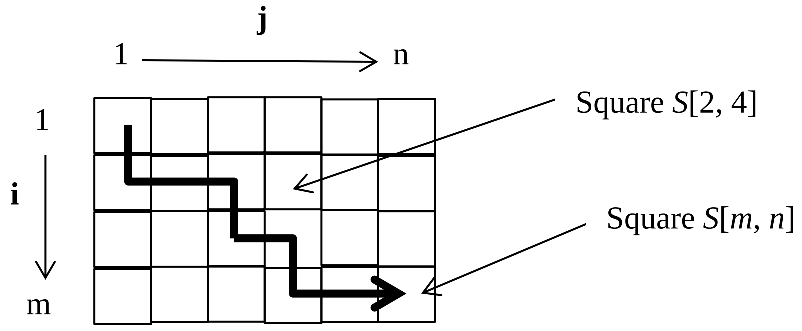
Make sure that your copy of this examination paper is complete before answering.
You can give your answers in English or in Norwegian, as you like.

Read the text carefully, and good luck!

Assignment 1 *Dynamic programming* (20 %)

We have a rectangular grid (or a matrix) S with m x n squares (where $m \geq 1$ and $n \geq 1$). The individual squares are referred to as $S[i, j]$, where $S[1, 1]$ is the upper left square, $S[m, 1]$ is the lower left one, $S[1, n]$ is the upper right one, and $S[m, n]$ is the lower right square. We shall look at paths from the upper left square to the lower right one, where each step along the path are either a single move downwards or to the right. Such paths are called *complete paths*. See figure below.

Each square $S[i, j]$ also has a positive (non-zero) integer "cost" $C[i, j]$. We can thus also talk about the *cost of a complete path,* defined as the sum of the costs in all the squares it passes through (including $S[1, 1]$ and $S[m, n]$). The cost values are given in an integer array "$C[1..m, 1..n]$" (and when programming you can generate arrays e.g. as follows: "**new** integer array $A[d..e, f..g]$"). The task here is to find the minimal cost that a complete path can have over all complete paths. You shall use dynamic programming to find this cost.



A rectangular grid with a *complete path*. An integer (the cost $C[i, j]$) is given for each square $S[i, j]$, but these are not shown in this figure.

1.a

As we shall use dynamic programming, we need some type of table to store the intermediate results. Describe the table *T* you want to use for the above problem, and what the entries of *T* should contain, in relation to the original problem.

1.b

Here you should first write down the recurrence relation you want to use to fill table *T*. Then, you should specify exactly the initializations you want to do, so that you afterwards can fill in the table *T* in a bottom up fashion.

1.c

Write a program that fills in the table in a bottom up fashion. Make sure to also include the initializations you want. The program should be written in a suitable (and understandable) language and you shall write the program as a "method" (or "procedure" or "function") that receives the intergers *m* and *n,* and the array *C* as parameters. The method should be named "minCost", and it should only compute the cost of a minimal path (not the path itself), and it should return this cost as its result. The method itself is responsible for generating the table *T*.

1.d

We now introduce a new element into the problem, which is that, in addition to *m*, *n* and *C*, an integer *F* ("forbidden") is given. Also, a new rule is introduced:

> **Rule:** If the cost *C*[*i*, *j*] is equal to *F*, then no complete path is allowed to pass through square *S*[*i*, *j*]. That is: Any such path will simply not qualify as a complete path.

Thus, there may in some instances be no complete paths at all through the grid (and this is true e.g. if *C*[1,1] = *F*). In such a case the value for the "minimum cost of a complete path" is defined to be zero.

Your task here is to answer the questions **1.a** and **1.b** again, but for this new situation. It is enough to specify what you will now do differently than in **1.a** and **1.b.**

1.e

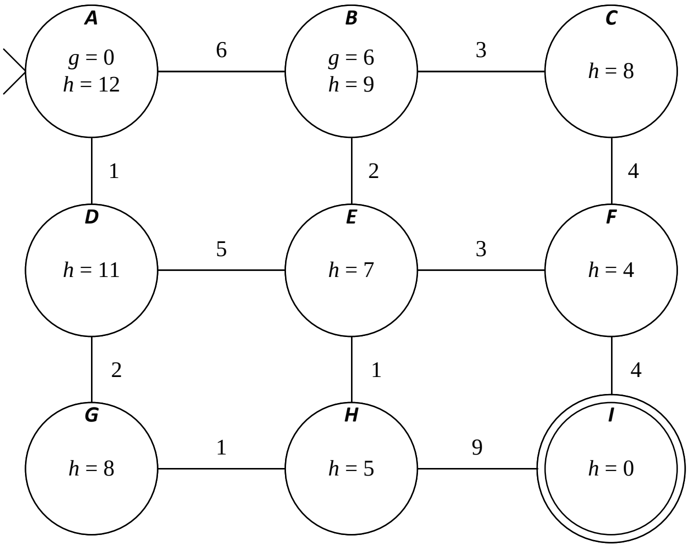
We consider using memoization to solve the above problems. We wonder whether there are instances where we, by using memoization, will end up looking at fewer entries in the table *T* than we will when filling the table bottom up. You should answer this question both for the situation in **1.a** and for the one in **1.d**. Explain.

Assignment 2 *Shortest Path* (15%)

We shall use the A*-algorithm to find the shortest path in a grid (Remember that the A*-algorithm requires the heuristic to meet certain constraints, and that Dijkstra´s algorithm is a special case without any heuristic.)

2.a

Given the following (undirected) grid graph, we want to find the shortest path from the start node ***A*** to the goal node ***I*** with the A*-algorithm. Edge lengths and heuristic values for the nodes are indicated in the grid. Your task is to show what happens in each step of the algorithm.



As an example showing how you shall describe what happens during the A*-algorithm, we give the table below. However, note that this table describes what happens in the first few steps of *Dijkstra's algorithm* when it searches for the shortest distance from ***A*** to ***I***.

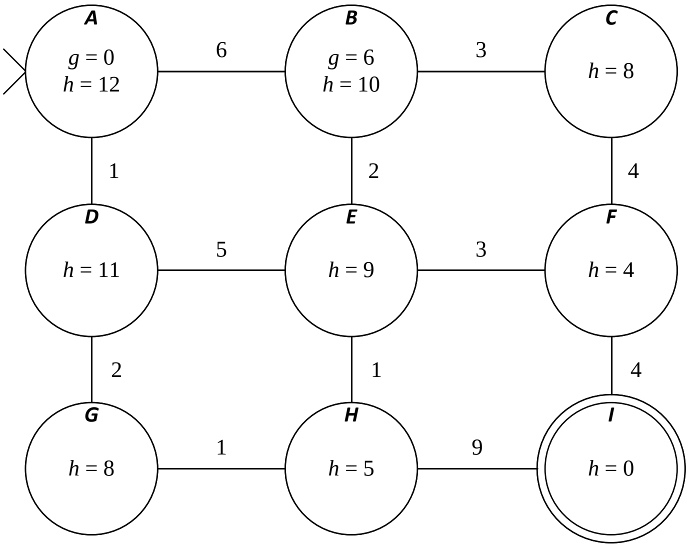
The important elements to describe after each step is what nodes reside in the priority queue, and the priorities of these nodes. In the Action part of each line we indicate what node will be taken out of the priority queue and what nodes will change priority, in the following step. To get a unique order we decide that neighbours of a node are treated alphabetically, and that the FIFO-principle is used if nodes in the priority queue have equal priorities.

Step	Priority queue		Action
	Node	Priority	
1	***A***	0	***A*** is dequeued (gets inserted in the tree)
2	***B***	6	
	D	1	***D*** is dequeued
3	***B***	6	
	E	6	
	G	3	***G*** is dequeued
4	***B***	6	
	E	6	***E***'s priority is changed from 6 to 5
	H	4	***H*** is dequeued
5	***B***	6	
	E	5	***E*** is dequeued
…	**…**	…	…

Your task is to fill in a similar table for all the steps of the A*-algorithm using the heuristic indicated in the grid graph above. In the column "Priority" you shall show the priority relevant to the A*-algorithm, but also how this priority is the result of adding two values, by giving these values. The first few *g*-values (the actual distance from node ***A***) are indicated in the nodes.

2.b

We still want to find the shortest path from the start node ***A*** to the goal node ***I*** in the same grid, but now with a slightly adjusted heuristic. The heuristic values in the figure below are ever so slightly adjusted compared the ones used in question **2.a**.



Again, your task is to fill in a table similar to the one in question 2.a with the new heuristic indicated in the figure above.

2.c

Comment on the usefulness of the heuristics used above. Your comments should discuss the special heuristics used in **2.a** and **2.b**, but might also include more general considerations.

Assignment 3 *Find algorithms* (15 %)

We look at the following situation: Two schools A and B agree that some of their pupils should have a "net-friend" at the other school. Therefore n pupils are chosen at each school, and a party is held for all the 2∗*n* pupils so that they should get to know each other. Afterwards, each pupil *p* makes a list with *k*(*p*) pupils picked from the *n* pupils at the other school that he/she would like to have as net-friends. The number *k*(*p*) of pupils on this list is chosen individually, in the rage from 1 to *n*. Each pupil *p* also gives a number *m*(*p*) (in the range from 1 to *k*(*p*)), and this is the maximum number of net-friends that he/she would like to have. These lists and all the numbers are given to the school administrations.

The administrations of the two schools decide that they should only make a net-friend pair of two pupils if both of them want the other as a net-friend.

3.a

We first assume that the number *m*(*p*) is 1 for all pupils p (that is: Each pupil should have at most one net-friend). Specify an algorithm that the administrations can use to decide whether it is possible to find a net-friend for all the 2∗*n* pupils under these conditions, and if so, finds a net-friend to each of the 2∗*n* pupils. You can refer to algorithms in the curriculum, and you should explain why your method will work for this case.

3.b

We now assume that the values of *m*(*p*) can be higher than 1. Your task now is to describe how you can then solve the problem by transforming it to a problem discussed in the curriculum. Describe the necessary transformation, and explain why this solves the problem. You may indicate the idea of the transformation by sketching a simple example of a transformed instance.

Assignment 4 *Undecidability* (10 %)

Which of the following languages are undecidable? Prove your answer.

4.a

L = {M_1, M_2 | Turing machines M_1 and M_2 will, when started with blank tapes, write the same symbol in step number *n*, for all *n*}

4.b

L = {M_1, M_2 | Turing machines M_1 and M_2 will, for any (may be different) inputs, write different symbols in step number *n*, for all *n*}

Assignment 5 *NP-completeness* (10 %)

The input in each of the following languages is a set *S*, positive integer weights *w*(*s*) for each element *s* of *S*, and a positive integer *N*. Determine the complexity of the following problems and prove your answers.

5.a

Decide if there is a way to choose 100 elements of *S* so that their values add up to *N*.

5.b

Decide if there is a way to choose any number of elements of S so that their values add up to *N*.

Assignment 6 *General knowledge* (15 %)

Provide a short answer or explanation to each of the following.

6.a

What is achieved by representing problems as formal languages?

6.b

What is achieved by proving that a certain problem belongs to a certain class?

6.c

What is achieved by defining the Turing machine?

6.d

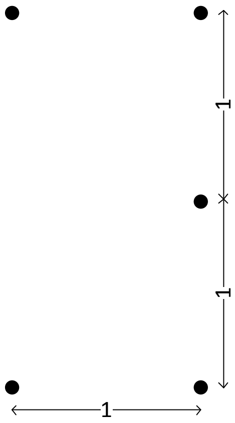
What difficulties are involved in using average-case complexity?

6.e

What limitations exist in speeding up solutions to problems by using parallel machine?

Assignment 7 *Triangulation* (15 %)

We are given the following five points in the plane. Distances are indicated by the arrows.



7.a

Draw all triangulations of the five points. Omit symmetries and rotations.

7.b

Which triangulation from 7.a is the Delaunay triangulation? Give a short justification/proof! Hint: Possible angles are 18,4°, 26,6°, 45°, 63,4°, 90°, and 135°. (This is after all not an exercise in trigonometry.)

7.c

Draw the Voronoi diagram for the five points.

[END]