**1 (a)**

**Edges in a tension tree**

Assume that G is a connected graph with 12 nodes and 20 edges and that S is a spanning tree of G.

How many edges S have?

Answer here:

**1 (b)**

**Edges in a span forest**

The graph G has 6 nodes and 6 edges.

We use Kruskal's algorithm to find F (the spanning forest of G).

Depending on how G looks like, what is the minimum number of edges F can contain?

**1 (c)**

**Minimal spanning trees**

Check the one or more algorithms, which given a connected weighted graph, finds a **minimal spanning tree**for the graph.

**Select one or more options:**

* Width-first-search (BFS)
* Kruskal
* Depth-first-search (DFS)
* Prim
* Borůvka

**1 (d)**

**FEM**

Given the following problem, called FEM:

INSTANCE: an integer **n**

QUESTION: is n = 5?

Is FEM in NP?

**Choose one option:**

* Yes, because FEM is in P
* No, because FEM is in P
* Yes, because FEM is not in P
* No, because FEM is not in P

Max points: 1

**Note:** You get 1 point for the correct answer, 0 points for the unanswered answer and -1 point for the wrong answer.

This means that you are expected to get fewer points for pure betting than by not answering.

Only one option gives 1 point.

**1 (e) SUDOKU CHECK**



We say that a sudoku board is in order n, if the boxes, columns and rows are of size n2. That is, you must fill in the numbers from 1 to n2. For example, a regular sudoku board (1-9) will have order 3.

In an unfinished sudocubrate of order n, each box, column and row contains the numbers from 1 to n2 exactly one time. Some unfinished sudoku boards may have more than one unique solution (for example, an empty board).

Alice has created a program called Aloku.

Aloku is pretty good at solving sudoku boards.

Given an unresolved sudocuboard of order n, the program does the following:

* If n > 100, the board is too large and the program terminates without trying to resolve it
* If n ≤ 100 and the board has exactly one unique solution, the program will return this solution in final time
* If n ≤ 100 and the board has more than one unique solution, the program will never terminate (run forever)

SUDOKU CHECK:

INSTANCE: An unresolved sudoku board of order n

QUESTION: Will Aloku solve the board in final time?

In this thesis we assume that N ≠ NP.

Select one option regarding the complexity of the SUDOKU CHECK, which is given an unresolved sudoku board will decide whether Aloku finds a solution or not.

**Choose one option:**

* SUDOKU CHECK is indeterminate, as it is possible that Aloku will run forever. We must therefore solve the stopping problem that is indefinable.
* SUDOKU CHECK is NP-complete, because it is known that SUDOKU is NP-complete. We do not have to solve the stopping problem.
* SUDOKU CHECK can be solved in constant time, because Aloku can only solve finally many boards.

**Note:** You get 5 points for the correct answer, 0 points for the unanswered and -3 points for the wrong answer. It means that you are expected to get fewer points for pure betting than by not responding. Only one option gives 5 points.

**1 (f)**

**Code analysis**

Give the runtime of the code snippet in O-notation:



**5-CLIQUE**

One k-click, or a click of size k, in a graph G = <V, E> is a subset C ⊆V of K nodes which forms a complete graph. That is, if u,v ∈ C are two different nodes in the click, so must {u, v} ∈ E.

In the 5-CLIQUE issue, determine if a graph contains a click of a size 5.

5-CLIQUE

INSTANCE: A graph G

QUESTION: Contains G a 5-click?

**Tips & Warnings** See previous exercise.

**Choose one option:**

* 5-CLIQUE is NP-complete
* 5-CLIQUE is in P

**Shortest distances in graphs**

For each graph type, select the **fastest** algorithm that finds the shortest distances in the graph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Breadth-First Search** | Dijkstra  | Topological Sorting | Bellman-Ford |
| No negative cycles |  |  |  |  |
| No negative edges |  |  |  |  |
| WEIGHTED DAG |  |  |  |  |
| Unweighted |  |  |  |  |

Explanation:

* Unweighted - an unweighted graph. The graph is either corrected or incorrect.
* Weighted DAG - a weighted, directed acyclic graph. The edges may have a negative weight.
* No negative edges - the graph is weighted, but no edges have negative weight. The graph is either corrected or wrong.
* No negative cycles - the graph is weighted, but contains no cycles with negative weight. The graph is either directed or undirected.

**1 (i)**

**Huffman code**

Given this huffman code table:

|  |  |
| --- | --- |
| A | 0 |
| C | 100 |
| G | 101 |
| T | 11 |

Which text string does the code 1010111101000 stand for?

**1 (j)**

**Closed hashing**

Given a hash table of length 5 that can store integers.

We use closed hashing with linear testing and the hash function h(k) = k mod 5.

We add the following elements to the table in this order:

17, 98, 59, 32, 40.

On which index does the last element (40) end up?

**Introduction**

This section is divided into nine tasks (ai) that are partly based on each other. If you are stuck on one task, it is possible to move on to the next, but it is highly recommended to do them in it

the order they are given. Below is important information about the task that applies throughout the section. In this exercise you will help to design worlds in a computer game.

A (game) world consists of different rooms. In each room there is a panel that shows the player which other rooms he can walk to from that room.

**The example world**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Room | A | B | C | D | E | F | G | H |
| Panel | B,C,F | A,H | A,D,E | G | F,G,H | E,H | H |  |

From a room, the player selects a new room from the panel. When a room is selected, the player is given a task that must solved to get to the room he chose. After the task is solved, the player is transported to the selected room.

The goal in every world is to reach a **treasury** in the shortest possible time.

A room K is a **treasury** if and only if it meets the following requirements:

* It is not possible to reach any other rooms from K.
* It is possible to get to K from any other room in the world.

In the example world above, H is a treasure trove.

Each world begins with the player being placed in an arbitrary **starting room** .

A room S is a **starting room**if and only if it meets the following requirement:

* It is possible to get from S to any other room in the world.

In the example world above, there are three starting rooms, namely A, B and C

**2 (a)**

**Representation**

We will model a world as a directed graph where the nodes represent the rooms in the world.

1. What are the edges of the graph?

2. What determines the degree of a node?

3. What is the degree of a treasurer?

4. In the example world, what is the degree to room D?

**Note: Your**answers should apply to a general world, not just the example world. This does not apply to

point (4).

**2 (b)**

**Component graph**

You are given a world and represent it as a directed graph G = <V,E> as in task a. We can calculate the strongly connected components of G and create the **component**graph C. The component graph C is a graph in which each strongly connected component in G constitutes the nodes in C. There is

a directed edge in C from one component to another, if there is an edge in G, from a node in the first component to a node in the other component.

Here you get a started drawing of the component graph of the example world.

Note that {A, B, C} is a node in C, because {A, B, C} is a strongly connected component in G, and that that node has out grade 3.

Complete the drawing of the component graph of the sample world.

**2 (c)**

**Treasury I**

You are given a world and represent it as a directed graph G = <V,E> as in task a. Let node k ∈ V be a treasury.

Let Ck  be the strongly connected component of k.

For each of the questions below, give a short justification (1-3 sentences).

1. How many nodes contain Ck  ?

2. What is the out degree of Ck

?

**Note: Your**answers should apply to a general world, not just the example world.

**2 (d)**

**Treasury II**

Briefly explain why a world cannot contain more than one treasury.

**Note: Your**answer should apply to a general world, not just the example world.

**Write your answer here:**

**2 (e) Startroom**

You are given a world and represent it as a directed graph G = <V,E> as in task a. Let the node s ∈ V be a starting room.

Let Cs be the strongly connected component of s. We will now show that the starting rooms in the world areexactly the nodes in Cs .

* Explain why it is so that if t ∈ V also is in Cs, so must t be a starting room.
* Show that all starting rooms in the world are in Cs.

.

**Note: Your**answers should apply to a general world, not just the example world.

**2 (f)**

**Algorithm design**

A game designer has made a proposal for a world and represented it as a directed graph G=<V,E>. You are asked to check if the world represented by G has the following characteristics:

* the world has exactly one treasury
* the world has exactly three starting rooms.

Write an algorithm that checks if a proposed world has the two properties above. For simplicity, you can assume that you get the graph as input. You can also assume that for each node you get a list of outgoing and incoming edges for the node.

Here you can, if desired, reuse results from the previous sub-tasks (even those you have not answered). You can use natural language and / or pseudocode to describe your algorithm. In addition, it is possible to reuse algorithms we have looked at in book. For example, if you want to traverse graph with depth-first search, you do not need to explain how depth-first search works. If you wish to

use a modified version of an algorithm from the course, the modifications must be clearly stated.

In this task a fast algorithm will pay off more

than a less fast algorithm. In the next task, you will be asked to analyze the runtime of your algorithm.

**Note: Your**answer should apply to a general world, not just the example world.

**2 (g)**

**Analysis**

Give an analysis of the runtime of your algorithm from subtask f using O-notation. If you used algorithms known from the book, you do not need to explain why they have the running time they have.

**Note: Your**answer should apply to a general world, not just the example world.

**2 (h)**

**Minimum execution time**

In this sub-task you will also be given the minimum time needed to solve the various tasks. The panels now consist of pairs of rooms and positive integers that give the least time it is possible to solve the current task.

We now want to find the minimum execution time for a world.

You will use Dijkstra's algorithm to find the fastest possible way to get to Treasury H, in the example world above. The player begins in room A.

List all the estimates of the treasury during the calculation.

Here you should give a list where the first element is ∞ and the last element is the correct distance to the treasury.

 **2 (i)**

**Uncompressible worlds**

Dijkstra's algorithm uses, as is known from the book O(|E| log |V|), (possibly O(|V|2) ) time. In the last sub-task we will look at a special type of world where we can calculate the minimum

execution time in linear time.

An **uncompressible**world is a world such that if we represent it as a graph G = <V,E> , and then calculating the component graph C, we get that the number of nodes in G will be the same as the number of nodes in C. This will be the case if all nodes are "alone" in their strongly connected component.

Suppose we are given an uncompressible world with exactly one treasure trove and exactly one starting room and with minimum times for all tasks.

Explain briefly why it is possible to find the minimum execution time from the starting room to the treasury in

linear (O(|V| + |E|)) time.

Here it is not the intention that you should implement an algorithm, we are only looking for a brief justification for why it is possible.

**Introduction**

An easy way to sort is to put the items to be sorted in a priority queue (PQ) with the method **add** . Only when all the elements have been entered, we take out one element at a time using

PQ’s **removeMin** method. They will then come in ascending order. In this section we will look at different sorting strategies for PQ.

In all the tasks we add the same 6 elements inn PQ using **add** , as follows:

PQ.add (17);

PQ.add (43);

PQ.add (98);

PQ.add (11);

PQ.add (43);

PQ.add (56);

PQ uses an array as an internal structure. What you will find out in the tasks is what this array looks like after 56 is inserted depending on the sorting method the task specifies that PQ uses.

**3 (a)**

**PQ uses selection sorting**

How will the inner array look after the calls on **add** if PQ uses selection sort to find the item with the highest priority?

**3 (b)**

**PQ uses Insertion sorting**

How will the inner array look after the calls on **add** if PQ uses insertion sorting to find the item with the highest priority?

**3 (c)**

**PQ uses BST**

PQ now uses, instead of an array, a binary search tree with smaller values ​​to the left, and larger or equal values ​​to the right. What will be the order of the numbers if we do a BFS traversal from the root in this tree and print the numbers? We traverse left children before right children.

Write the answer as a comma-separated list .

 **3 (d)**

**PQ uses heap sorting**

How will the inner array look after the calls on **add** if PQ uses heap sorting (with a *minimumsheap* where **add**puts new item at the back and lets this bubble upwards to the right place) to find the item with the highest priority?

Write the answer as a comma-separated list (as in the PDF) here:

**3 (e)**

**Requirements for a heap**

Explain the two requirements that must be met for a binary tree to be a minimum heap.

**3 (f)**

**heapSort-Explain**

The start of the heap sorting algorithm turns the integer **array** A into a **max heap** :



The expression n/2 is n integer-divided with 2.

Explain briefly why it is unnecessary to start the pre-loop on n - 1, but that it keeps on starting n/2 as the code does.

**3 (g)**

**heapSort Code**

The start of the heap sorting algorithm turns the integer **array** A into a **max heap** :



The expression n/2 is n integer-divided with 2.

Complete the heapSort code so that A is sorted in ascending order when A is returned.

Here you can call downHeap, without implementing the method. In the next task you will be asked to implement downHeap so that heapSort works properly

**3 (h) downHeap code**

The start of the heap sorting algorithm turns the integer **array A** into a **max heap** :



The expression n/2 is n integer-divided with 2.

Write java code for downHeap so that heapSort works properly.