University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn

Algorithms and Data Structures Exam

12th march 2025, 14:00 -17:00

Name: .....

Matriculation No.: .....

Signature: .....

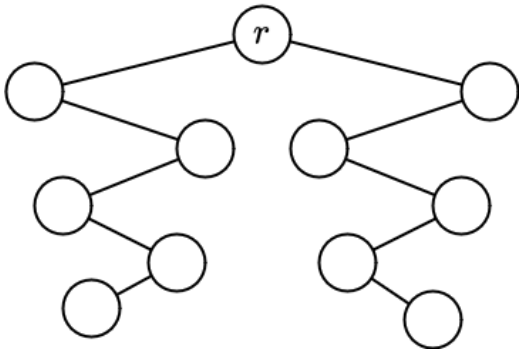
Do not open or turn until told so by the supervisor!

- Put your **student ID** in front of you or on the table next to you.
- Write your **name** and **matriculation number** on this page and **sign** the document.
- Your **signature** confirms that you have answered all exam questions yourself without any help, and that you have notified exam supervision of any interference.
- You are allowed to use a summary of **six handwritten, single-sided A4 pages**.
- **No electronic devices** are allowed.
- Write legibly and only use a pen (ink or ball point). Do **not use red**! Do **not use a pencil**!
- You may write your answers in **English or German** language.
- Only **one solution per task** is considered! Make sure to strike out alternative solutions, otherwise the one yielding the minimal number of points is considered.
- **Detailed steps** might help you to get more points in case your final result is incorrect.
- The keywords **Show...**, **Prove...**, **Explain...** or **Argue...** indicate that you need to prove or explain your answer carefully and in sufficient detail.
- The keywords **Give...**, **State...** or **Describe...** indicate that you need to provide an answer solving the task at hand but without proof or deep explanation (except when stated otherwise).
- You may use information given in a **Hint** without further explanation.
- **Read each task thoroughly** and make sure you understand what is expected from you.
- **Raise your hand** if you have a question regarding the formulation of a task or if you need additional sheets of paper.
- A total of **45 points** is sufficient to pass and a total of **90 points** is sufficient for the best grade.
- Write your name on **all sheets**!

Task	1	2	3	4	5	6	7	**Total**
Maximum	25	15	10	20	15	20	15	120
Points								

Task 1: Short Questions *(25 Points)*

(a) Give a sequence of insert operations for an (unbalanced) binary tree datastructure, that results in the tree structure depicted below. (So it should be exactly 11 insert operations.) *(4 Points)*



(b) Prove or disprove: A binary search tree can have depth $o(\log n)$, where n is the number of elements in the binary search tree. *(5 Points)*

(c) Give an algorithm that decides for a given connected, directed, weighted graph $G = (V, E, w)$ whether G contains a positive cycle. All edge weights are real numbers ($w : E \to \mathbb{R}$). We call a cycle positive, if the sum of the weights of its edges is strictly positive. Argue why your algorithm is correct. What is the asymptotic complexity of your algorithm? *(4 Points)*

(d) What is the asymptotic complexity of the following pseudocode? Give the complexity in terms of $\Theta(\cdot)$ notation and give an explanation.

Algorithm 1 `Mystery`(n)

```
for i = n/10 to n do
    y ← i
    while y < n do
        y ← y · 2 + 1
```

(5 Points)

(e) Given a weighted graph $G = (V, E, w)$, with $w : E \to \mathbb{R}$, and $T \subset E$, a Minimum Spanning Tree of G. Prove that T is also an MST of $G' = (V, E, w')$, where $w'(e) = (w(e))^3$. *(7 Points)*

Sample Solution

(a) insert(6), insert(1), insert(5), insert(2), insert(4), insert(3), insert(11), insert(7), insert(10), insert(8), insert(9)

(b) The statement is false.
It is known from the lecture, that a binary tree has at most 2^t nodes at depth t (the root has depth 0). Let n be large enough, such that the depth $t \leq \frac{1}{2}\log n - 1$, then the maximum number of nodes in a tree is

$$\sum_{i=0}^{t} 2^i = 2^{t+1} - 1 = 2^{\frac{1}{2}\log n} - 1 = \sqrt{n} - 1 < n$$

So a tree of depth $\frac{1}{2}\log n - 1$ can't even fit $\sqrt{n}$ nodes.

(c) We multiply all edgeweights with -1, as a result any positive cycle is now a negative cycle. Now we run Bellman Ford to decide whether there exists a negative cycle in the reweighted graph. Changing the edgeweights takes at most $O(|E|)$ time and running Bellman Ford takes $O(|V| \cdot |E|)$ time.

(d) The outer for loop runs for $\frac{9}{10}n \in O(n)$ iterations. The inner while loop runs for at most 4 iterations. This is because the initial value is at least $\frac{n}{10}$. As a result after doubling this value at most 4 times, it will always be larger than n. So the while loop only ever runs for $O(1)$ iterations. As a result the overall running time is $O(n)$.

(e) The change in edgeweights $w'(e) = (w(e))^3$ does not change the relative order of the edges. So for any two edges $e, e' \in E$ it holds that:

$$w(e) < w(e') \Rightarrow w'(e) < w'(e')$$

So if an edge e was a safe edge in G, it will remain a safe edge in G'. Since all edges in T are safe edges in G, all of them are safe edges in G'. So T is an MST in G'.

Task 2: Landau-Notation *(15 Points)*

(a) Below are 5 functions in $n \in \mathbb{N}$. Give a sorted order of these functions in terms of asymptotic growth. That is if f appears before g in the sorted order, then $f(n) \in \mathcal{O}(g(n))$. You are only required to give the correct order, proofs are not necessary. *(5 Points)*

- $a(n) = 20^{20^{20^{20^{234567898765434567890987654345678909876543567890987654345678909876543456789098765434567890987654560}}}}$
- $b(n) = 2^n \cdot \log n$
- $c(n) = 16^{\log_2(n)} + 3n^2$
- $d(n) = 78n^3$
- $e(n) = 4^{n/2}$

(b) Prove or disprove based on the definition of the Landau-Notation:

$$n^{1/4} \cdot n^{1/3} \cdot n^{1/2} \in \Theta(n)$$

(4 Points)

(c) Prove using the definition of the Landau-Notation:

$$\frac{64}{6} \cdot \left\lceil \frac{n}{2} \right\rceil ! \in \Omega(2^n)$$

You may assume that n is always even. *(6 Points)*

Sample Solution

(a) $a(n) <_O d(n) <_O c(n) <_O e(n) <_O b(n)$

(b)

$$n^{1/4} \cdot n^{1/3} \cdot n^{1/2} = n^{13/12} = n^{1/12} \cdot n$$

The statement is false. we will show $n^{1/12} \cdot n \notin O(n)$. To that end, let $n_0 > 0$ and $c > 0$.

$$\begin{aligned} n^{1/12} \cdot n &\leq c \cdot n \\ n^{1/12} &\leq c \\ n &\leq c^{12} \end{aligned}$$

But since c is a constant, this cannot hold for all $n > n_0$.

(c) The statement is true, set $n_0 = 4, c = 1$, we first replace all terms larger than 4 with 4 and then replace all 4's by $2 \cdot 2$ doubling the number of terms.

$$\begin{aligned} \frac{64}{6} \cdot \left(\frac{n}{2}\right)! &= \frac{64}{6} \cdot \frac{n}{2} \cdot \ldots \cdot 4 \cdot 3 \cdot 2 \cdot 1 \\ &\geq \frac{2^6}{6} \cdot 4 \cdot \ldots \cdot 4 \cdot 3 \cdot 2 \\ &= 2^6 \cdot (2 \cdot 2) \cdot \ldots \cdot (2 \cdot 2) \cdot \frac{3 \cdot 2}{6} \\ &= 2^n = 1 \cdot 2^n = 1 \end{aligned}$$

Task 3: String Isomorphisms *(10 Points)*

Let $S_1 = c_1c_2c_3c_4 \dots c_n$ and $S_2 = d_1d_2 \dots d_n$ be two strings of the same length. C is the set of all characters that appear in S_1, and D is the set of all characters that appear in S_2. We say that S_1 and S_2 are **isomorphic** if there exists a bijective function $f : C \to D$ such that for all $1 \leq i \leq n$, we have $f(c_i) = d_i$.
Example: Let $S_1 =$ "paper" and $S_2 =$ "title". Then the sets of unique characters are: $C = \{a, e, p, r\}$ and $D = \{e, i, l, t\}$.
A valid function f can be defined as: $f(\text{p}) = \text{t}, f(\text{a}) = \text{i}, f(\text{e}) = \text{l}, f(\text{r}) = \text{e}$, which proves that S_1 and S_2 are isomorphic.
Design an efficient algorithm that, given two strings S_1 and S_2, determines whether they are isomorphic. If they are, your algorithm must output a valid isomorphism.
Argue why your algorithm is correct and analyze its runtime in terms of the length n of the two strings.

Sample Solution

We can solve this problem efficiently using a hash map (dictionary) to track character mappings:

1. If S_1 and S_2 have different lengths, return false.
2. Create two dictionaries:
   - $map1$: Maps characters from S_1 to S_2.
   - $map2$: Maps characters from S_2 to S_1.
3. Iterate through both strings simultaneously:
   - If c_i in S_1 has been mapped before, check if $map1[c_i] = d_i$. If not, return false.
   - If d_i in S_2 has been mapped before, check if $map2[d_i] = c_i$. If not, return false.
   - Otherwise, store the mappings: $map1[c_i] = d_i$ and $map2[d_i] = c_i$.
4. If we successfully traverse both strings, return true.

Complexity Analysis

- The algorithm processes each character of the strings once, leading to a time complexity of $O(n)$.
- Here we use the fact that operations on Hashtables have a timecomplexity of $O(1)$.

Proof of Correctness

1. If an isomorphic mapping exists, the algorithm ensures that each character in S_1 maps uniquely to a character in S_2 and vice versa.
2. If a character violates the bijection property (i.e., maps to different characters at different positions), the algorithm returns false.
3. By iterating through the entire string and enforcing consistent mappings, we guarantee correctness.

Task 4: A Thiefs Plan *(20 Points)*

There are n houses built in a line, each of which contains some money $m_1, m_2, \ldots, m_n$ in it. A robber wants to steal money from these houses, but they can't steal from two adjacent houses (e.g. they could steal from 2 and 4, but not from 3 and 4).
Give an efficient algorithm that takes as input the values $m_1, m_2 \ldots, m_n$ and outputs a set of house-numbers $i_1, i_2, \ldots, i_k$, such that the collected money

$$\sum_{1 \leq j \leq k} m_{i_j}$$

is maximised. Furthermore, no two consecutive numbers are contained among $i_1, i_2, \ldots, i_k$.
What is the asymptotic complexity of your algorithm? Give an explanation.

Sample Solution

This problem can be solved using dynamic programming:

- Define $dp[i]$ as the maximum amount of money that can be stolen when considering only the first i houses.
- The recurrence relation is:

$$dp[i] = \max(dp[i-1], dp[i-2] + m_i) \tag{1}$$

  where $dp[i-1]$ represents skipping house i, while $dp[i-2] + m_i$ means robbing house i but not house $i-1$.

- Base cases:

$$dp[1] = m_1$$
$$dp[2] = \max(m_1, m_2)$$

- Iterate from $i = 3$ to n to compute $dp[i]$.
- The final solution is $dp[n]$.

Complexity Analysis

Since the procedure iterates once over the n houses, and only spends $O(1)$ time per i, the time complexity is $O(n)$.

Task 5: Hashing with Open Addressing *(15 Points)*

We consider hash tables with open addressing and two methods for resolving collisions: double hashing and cuckoo hashing. Let m be the size of the hash table. We define

$$h_1(x) := (5 \cdot x) \mod m,$$
$$h_2(x) := 1 + (2x \mod (m-1)),$$
$$h_3(x) := (3 \cdot x - 2) \mod m.$$

(a) Let $h_d(x, i) := (h_1(x) + i \cdot h_2(x)) \mod m$. Insert the keys 13, 14, 2, 3, 11 sequentially into a hash table of size $m := 11$. Use h_d and double hashing for collision resolution. *(5 Points)*

(b) Insert the values 3, 10, 7 sequentially into a hash table of size $m := 7$. Use cuckoo hashing with the functions h_1 and h_3 for collision resolution. Provide the intermediate state of the table after each insertion (i.e., three tables in total). *(5 Points)*

(c) Prove or disprove: For any $m \in \mathbb{N}$ and universe $\mathbb{N}$, it holds that the set of hashfunctions $H = \{h_i(x) = x \cdot i \mod m \mid 0 \leq i \leq m-1\}$ is 2-universal. *(5 Points)*

Note: Write your solutions for (a) and (b) in the tables on the solution sheet provided for this question.

Sample Solution

Task (a):

Table after inserting all values:

3			11	14					2	13
0	1	2	3	4	5	6	7	8	9	10

Task (b):

After inserting 3:

	3					
0	1	2	3	4	5	6

After inserting 10:

3	10					
0	1	2	3	4	5	6

After inserting 7:

10	3				7	
0	1	2	3	4	5	6

- (c) The statement is false. Choose $x = 0, y = m$ then for all $h \in H$ it holds that $h(x) = 0 = h(y)$ and so H is not c-universal for any c.

Task 6: k-near *(20 Points)*

Given a (undirected) graph $G = (V, E)$ and two nodes $u, v \in V$, we are interested to determine wether or not u and v are at most $k \in \mathbb{N}$ hops apart.

(a) Give an algorithm to determine whether two given nodes u, v are at most k hops apart.

The input to your algorithm consists of the graph $G = (V, E)$, a parameter $k \in \mathbb{N}$ and two nodes $u, v \in V$.

The algorithm must run in linear time (i.e. the algorithm must terminate after at most $O(n+m)$ operations).

Argue why your algorithm is correct and analyze the running time. *(5 Points)*

We now change the graph a bit, we now give weights to the edges. Each edge either has weight 1, or weight 0. So we are now given a weighted graph $G = (V, E, w)$ with $w : E \to \{0, 1\}$. We are now interested in determining whether, or not two nodes u, v are at distance at most k.

(b) Give an algorithm to determine whether two given nodes u, v have distance at most k.

The input to your algorithm consists of the graph $G = (V, E, w)$, a parameter $k \in \mathbb{N}$ and two nodes $u, v \in V$.

The algorithm must run in linear time (i.e. the algorithm must terminate after at most $O(n+m)$ operations).

Argue why your algorithm is correct and analyze the running time. *(15 Points)*

Sample Solution

a) This problem can be solved using a breadth-first search (BFS), as BFS finds the shortest path in an unweighted graph efficiently:

(a) Initialize a queue with node u and set its distance to 0.
(b) Perform BFS, exploring neighbors level by level.
(c) If we reach node v within k steps, return true.
(d) If BFS completes without reaching v within k steps, return false.

Complexity Analysis: Since BFS explores each node and edge at most once, the time complexity is $O(|V| + |E|)$. The space complexity is also $O(|V|)$ due to the queue and visited set.

b) If the graph is weighted with edge weights $w : E \to \{0, 1\}$, we use a modified BFS where:

- Instead of a standard queue, use two priority queues, one for edges with weight 1 and one for edges with weight 0.
- Always work through the priority queue for the 0 edges first.
- Stop if v is reached within weight sum $\leq k$.

Correctness: The algorithm behaves like Dijkstra's, as it processes one of the nodes with the shortest subpath at each deque operation. The distances only increase when a node is removed from the "1" queue. All nodes that are processed in the "0" queue in the meantime have exactly the same distance. Therefore, the order of processing in the "0" queue does not matter. When new nodes are inserted into the "1" queue, they have a distance that is at least as large as all other nodes in the "1" queue.

Hence the correctness follows in the same way as it did for Dijkstra's algorithm.

Complexity Analysis: The time complexity is still $O(|V| + |E|)$, since every edge is processed in only one of the queues.

Task 7: Rotated Text *(15 Points)*

Given two string $C = c_0c_1 \dots c_{n-1}$ and $D = d_0d_1 \dots d_{n-1}$ both of length n, we say that D is a rotation of C if there exists a value j, such that for all $0 \leq i < n$ it holds that $c_i = d_{i+j \mod n}$. In other words D is C, but circularly shifted by j.
Example: The string "thmAlgori" is a rotation of "Algorithm", and in this case $j = 3$. On the other hand "lgoAritmh" is not a rotation of "Algorithm" (it is a permutation, but it is not a rotation).
Give an algorithm that takes as input two strings of length n and determines in $O(n)$ time, whether one is a rotation of the other. Prove that your algorithm is correct and that it satisfies the asymptotic running time constraints.

Sample Solution

This problem can be solved efficiently using string concatenation:

1. If C and D have different lengths, return false.
2. Construct a new string $C' = C + C$ (concatenation of C with itself).
3. Check if D is a substring of C'.
4. If D is found within C', return true; otherwise, return false.

Complexity Analysis

- Constructing C' takes $O(n)$.
- Checking if D is a substring of C' using the Knuth-Morris-Pratt (KMP) algorithm takes $O(n)$.
- The total time complexity is $O(n)$.
- Space complexity is $O(n)$ due to the concatenated string C'.

Proof of Correctness

1. If D is a rotation of C, then D must appear as a contiguous substring within $C + C$.
2. If D appears in $C + C$, then a valid rotation index j exists such that the rotation condition holds.
3. This guarantees that our method correctly identifies rotations in $O(n)$ time.