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

Algorithms and Data Structures Exam

27. August 2020, 10:00 -13:00

Name: . . . .

Matriculation No.: . . . .

Signature: . . . .

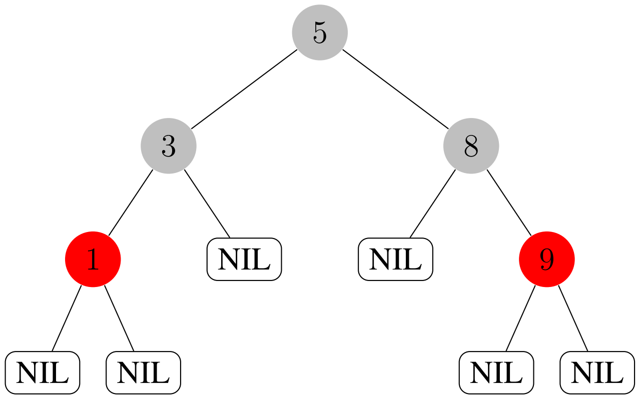
Do not open or turn until told so by the supervisor!

- 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.
- This is an **open book exam** therefore printed or hand-written material is allowed.
- **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.
- There is a **separate solution page** for each exercise.
- Write your name on **all sheets**!

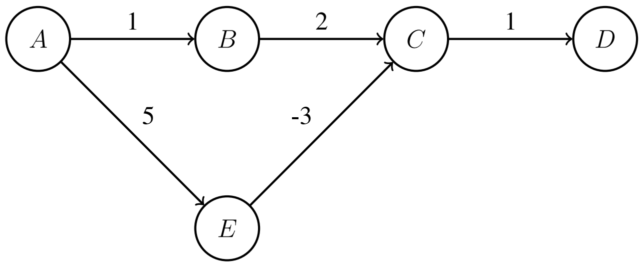
Task	1	2	3	4	5	6	7	8	**Total**
Maximum	18	19	17	10	17	12	15	12	120
Points									

Task 1: Short Questionns *(18 Points)*

(a) Perform the operation `delete(5)` on the following red-black tree. Draw the resulting tree. You may write the colors next to the nodes. (4 Points)



(b) Consider the following directed, weighted graph G. Run the Bellman-Ford algorithm on G starting from node A. Specify the computed distances of all nodes after each iteration of the outer loop. (4 Points)



(c) Draw an undirected, weighted graph $G = (V, E, w)$ and mark a starting node $s \in V$ such that every "Shortest Path Tree" rooted at s in G differs from any minimum spanning tree of G. (4 Points)

(d) Given an undirected, unweighted (not necessarily connected) graph $G = (V, E)$, we want to check if this graph is almost a spanning tree. By this, we mean that G consists of a spanning tree and at most c additional edges, where $c \in O(1)$ is a given constant. Describe an algorithm that performs this check in $O(|V|)$ time and justify the runtime. (6 Points)

Solution Task 1

Task 2: Sorting Algorithms *(19 Points)*

Given are k sorted arrays $A_1, \ldots, A_k$ with a total of n elements. We want to merge these arrays into a single sorted array A of length n.

(a) One possible solution is the following algorithm:

Algorithm 1 `sequential_merge`$(A_1, \ldots, A_k)$

```
A = A_1
for i = 2 to k do
    A = merge(A, A_i)
return A
```

where `merge()` is the merge operation as in the merge-sort algorithm.

Assume k is a divisor of n and all arrays have length $\frac{n}{k}$. State the runtime of `sequential_merge` as a function of n and k, and justify your answer. (7 Points)

(b) A student instead suggests writing all elements into an array of length n in any order, and then sorting this array using the merge-sort algorithm from the lecture. Is this approach faster or slower than `sequential_merge`? Justify your answer. (3 Points)

Hint: As in part (a), assume all arrays have length $\frac{n}{k}$.

(c) We now wish to solve the given problem in $O(n \log k)$ time for arbitrary values $k \leq n$, using binary heaps. Complete the following algorithm `heap_merge` (write the pseudocode that should replace the `???`). Justify the runtime.

Algorithm 2 `heap_merge`$(A_1, \ldots, A_k)$

```
H = create_binary_heap()                    ▷ creates an empty binary heap
for i = 1 to k do
    key = A_i[0]
    H.insert((i, 0), key)
A = Array of length n                       ▷ allocate an array of length n
for j = 0 to n − 1 do
    ???
return A
```

Hint: H manages data in the form (i, ℓ), where ℓ is a position in the array A_i. (9 Points)

Solution Task 2

Task 3: Big O-Notation *(17 Points)*

State whether the following statements are true or false. Prove or disprove each statement using the set definition of Landau notation (i.e., particularly without using limits).

(a) $n^2 - 3n \in \Omega(n^2)$ (4 Points)

(b) $(\log n)^2 \in O(\log(n^3))$ (4 Points)

(c) $n^2 \in O\left(\sum_{i=1}^{n} i\right)$ (4 Points)

(d) If $f(n) \in o(g(n))$ and $h(n)$ is monotonically increasing, then $h(f(n)) \in O(h(g(n)))$ (5 Points)

Solution Task 3

Task 4: Hashing with Open Addressing *(10 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)

Note: Write your solutions in the tables on the solution sheet provided for this question.

Solution Task 4

Task (a):

Hashtable after inserting all elements:

0	1	2	3	4	5	6	7	8	9	10

Task (b):

After inserting 3:

0	1	2	3	4	5	6

After innserting 10:

0	1	2	3	4	5	6

After inserting 7:

0	1	2	3	4	5	6

Task 5: Graphs *(17 Points)*

Given a directed, unweighted graph $G = (V, E)$, we define $G^2 = (V, E^2)$ such that $(u, v) \in E^2$ if and only if $u \neq v$ and there exists a directed path of length at most 2 from u to v in G.

(a) Describe an algorithm with runtime $O(|E| \cdot |V|)$ that computes the adjacency list representation of G^2 from the adjacency list representation of G. That is, v should be in the adjacency list of u if there is a path from u to v of length at most 2. Here, we allow v to appear multiple times in the list if there are multiple paths. Justify the runtime. (6 Points)

(b) Describe an algorithm with runtime $O(|E| \cdot |V|)$ that computes the adjacency list representation of G^2 without duplicate nodes in any adjacency list. Justify the runtime.

Hint: If you use a hash table, assume that inserting and checking values takes $O(1)$ time. (4 Points)

(c) Describe how to compute the adjacency matrix of G^2 from the adjacency matrix of G in $O(|V|^3)$ time. Explain the runtime. (7 Points)

Solution Task 5

Task 6: Mystery function *(12 Points)*

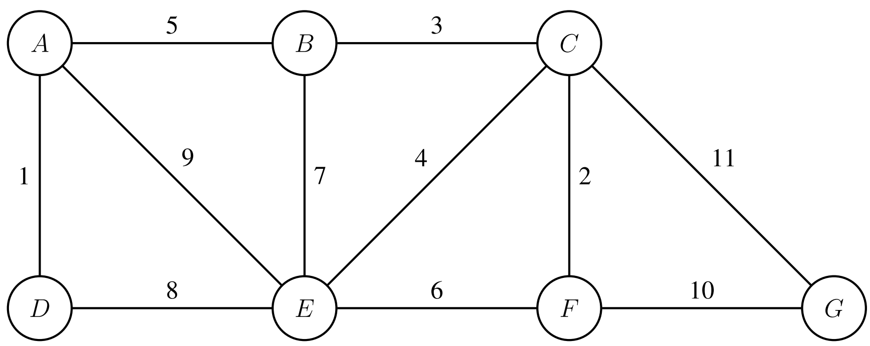
Consider the following algorithm in abstract pseudocode. It takes a weighted, undirected, connected graph $G = (V, E, w)$ as input.

Algorithm 3 `myst-edge-set`(V, E, w)

> **for** jedes $e \in E$ nach *absteigendem* Gewicht $w(e)$ **do** $\triangleright$ *beachte Iterationsreihenfolge!*
>   entferne e aus E
>   **if** (V, E) ist nicht zusammenhängend **then**
>   füge e zu E hinzu
> **return** E

(a) Run the algorithm `myst_edge_set(V, E, w)` on the graph below. Number each edge in the graph in the order that it is deleted by the algorithm (as a number next to the respective edge). Also mark all edges that are returned by the algorithm (by outlining or bolding them). (5 Points)

(b) What does `myst_edge_set(V, E, w)` return? Prove your answer. (7 Points)



Solution Task 6

Task 7: Separate Words *(15 Points)*

Given a dictionary D that contains words of maximum length $k \in O(1)$. Let T be a string of length n. We want to determine if T can be segmented into contiguous substrings, each of which is a word in D.

Example: Let $D = \{\text{'airplane'}, \text{'algorithms'}, \text{'train'}, \text{'awesome'}, \text{'are'}\}$. Then, for the inputs T_1 = 'algorithmsarestupid' or T_2 = 'airplanebus', the answer to the problem is False. For T_3 = 'algorithmsareawesome', the answer is True.

Hints:

- Assume k is provided as part of the input, and that you can check if a substring of length $\leq k$ is in D in $O(1)$ time (e.g., using hashing).
- Also assume that the characters of T are stored in an array $T[0..n-1]$.

(a) Let $t : \{0, \ldots, n-1\} \to \{\text{True}, \text{False}\}$ be a function such that $t(i) = \text{True}$ if and only if the substring $T[0..i]$ can be segmented into contiguous substrings, each of which is in D.

Derive a recursive relation for $t(i)$. That is, specify how $t(i)$ can be computed using $t(j)$ with $j < i$. Justify your answer. (8 Points)

(b) Provide an algorithm that solves the above problem in $O(n)$ time using dynamic programming. Justify the runtime. (7 Points)

Solution Task 7

Task 8: StringMatching *(12 Points)*

Given the pattern $P = \text{BACBAB}$ and the text $T = \text{CBACABBACBABACBABA}$.

(a) Provide the failure function (array S) of the Knuth-Morris-Pratt algorithm. (4 Points)

(b) Use the Knuth-Morris-Pratt algorithm to find all occurrences of P in T. Show the steps of the algorithm clearly. You may use the table provided on the solution sheet for this purpose. (8 Points)

Solution Task 8

C	B	A	C	A	B	B	A	C	B	A	B	A	C	B	A	B	A