MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Examinator: Lars Arvestad

Tentamensskrivning i
DA3018 Datalogi för matematiker
7.5 hp
2024-02-07

- **Write clearly.** Hard-to-read answers risk zero points.
- **Use one side of the paper**.
- Justify your answers (unless otherwise stated).
- **Grading thresholds:** E: 20, D: 26, C: 32, B: 38, A: 44

---

1. (a) Why do we study *asymptotic* time complexity? (2p)

(b) Explain *unit cost* is in relation to time complexity. (2p)

(c) Explain what a *single-linked list* is, with illustration. (2p)

(d) Explain what a *stack* is in Computer Science, with illustration. (2p)

2. (a) Give a formal description of the computational problem of sorting of integers. (3p)

(b) What time complexity does a computer scientist expect from a standard sorting algorithm? (2p)

(c) Describe *merge sort* with pseudo code. (3p)

3. Suppose we want to use a hash table for storing strings with maximal length 100. We will use chaining for handling collisions.

(a) Explain what *chaining* means in relation to hash tables. (2p)

(b) Explain what *load* means in relation to hash tables. (2p)

(c) Draw an illustration of a small hash table that stores the strings "DA3018", "DA2004", and "MM2001". The illustration should visualize the main features of a hash table for strings. (3p)

(d) Suggest, with pseudocode, a simple hash function for this problem and justify why the hash function is suitable. (3p)

4. Describe an algorithm for multiplying two polynomials of degree k and analyze its time complexity.

The polynomials are given as arrays of size k containing coefficients. The element on position i is the coefficient for the term with degree i. For example, $x^2 + 2x + 4$ is represented by $[4, 2, 1]$. (5p)

5. (a) What is a *binary search tree*? (2p)

(b) Suppose we implement a binary search tree for storing n data points containing geographical coordinates (latitude and longitude) stored as floats and a string containing up to 100 characters. How much memory is needed? Use and explain reasonable assumptions. (4p)

(c) Give a recursive algorithm for deciding whether x is found in the binary search tree or not. (3p)

6. **The center of a tree.** The distance between two vertices a and b in a tree T is the number of edges on a path from a to b and is written $d(a, b)$. The *eccentricity* of u in T is $\mathrm{ecc}(u) = \max_{v \in V(T)} d(u, v)$, i.e., the largest distance from u to any other vertex. The *center* of T are the vertices minimizing eccentricity: $\{u : \mathrm{ecc}(u) = \min_{v \in V(T)} \mathrm{ecc}(v)\}$. One can prove that the center of a tree is always one vertex or two vertices on an edge.

(a) The pseudocode in Figure 1 implements an algorithm for returning a center vertex (regardless if there are one or two centers) for a binary tree T. Analyze its time complexity. (3p)

(b) What is the name of the algorithm technique used in the helper function `get_eccentricity` (see Figure 1)? (2p)

(c) Suggest changes to the algorithm such that the time complexity improves to linear time. (5p)

```
def get_center_vertex(T):
   candidate = NULL
   min_eccentricity = |V(T)| # Initialize
   for v in V(T):
      v_ecc = get_eccentricity(v, T)
      if v_ecc < min_eccentricity:
          min_eccentricity = v_ecc
          candidate = v
   return candidate


def get_eccentricity(v, T):
   for u in V(T):
      visited[u] = False

   Q = new Queue()
   Q.enqueue(v)
   visited[v] = True
   d(v) = 0
   while not Q.empty():
      w = Q.dequeue()
         for u in neighbors(w):
            if not u.visited:
               Q.enqueue(u)
               u.visited = True
               d(u) = d(w) + 1

   eccentricity = 0
   for u in V(T):
      if d(u) > eccentricity:
         eccentricity = d(u)
   return eccentricity
```

Figure 1: An algorithm for computing the center of a tree. The helper function `get_eccentricity(v,T)` computes the longest distance from v to a leaf in T.