MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Examinator: Lars Arvestad

Take-home exam in
DA3018 Computer Science for Mathematicians
7.5 hp
2020-05-25

- **Write clearly.** Hard-to-read answers risk zero points.
- Justify all your answers (unless otherwise stated).
- If you search for information in books or on the internet, clearly state your sources!
- **Grading thresholds:** E: 20, D: 24, C: 28, B: 32, A: 36

Mandatory preparatory assignment

Read, reflect, and adapt the following text (replace `<your_name>` with your name) to the top of your hand-in!

> *I, `<your_name>`, guarantee that I have solved the assignments on this take-home exam independently and without help from some else. In particular, I have not discussed the exam and my solutions with anyone.*

This is mandatory and required for a passing grade on the exam.

Assignments

1. Being an old C hacker, Kim is storing a set of m strings in an large character array A of size n and terminating each string with a zero element. It is a compact representation that Kim is very proud of. Now Kim needs help, because searching in A is slow.

(a) Write a helper function (in pseudocode) that, given A, returns an integer array S of length m such that $S[i]$ contains the index into A where word i is located. Note that $S[0] = 0$ and if the first word has length l, then $S[1] = l + 1$ is the index of the second word (the first l elements of A contains the first word, followed by a terminating zero).
Analyse the time complexity of the helper function. (3p)

(b) What would the time complexity be for sorting the strings, using S, with mergesort? We do not know the sizes of the strings and we are not changing the positions of the strings in A. Note that the strings are not moved around in A. (2p)

Index	0	1	2	3	4	5	6
Characters	`N`	`o`	`0`	`Y`	`e`	`s`	`0`

Figur 1: Example of Kim's string set datastructure, containing the words "No" and "Yes". Although this is a made-up example, a C programmer might very well use a similar datastructure. Combining characters and zeros like this is not really a disturbing type mismatch, because 0 (in positions 2 and 6) represents the null character.

2. Directed graphs can have source vertices, with only outgoing edges, and sink vertices, that only have incoming vertices.

(a) Suggest an algorithm to identify, given a directed graph G, the sources of G and, for each source vertex v, the set of sinks $\sigma(v) \subset G$ reachable from v. Your algorithm should be based on topics from this course, not advanced topics such as flow graphs, and be reasonably efficient (solutions requiring exponential time are not accepted). (5p)

(b) Analyse the time complexity of your algorithm. (2p)

3. Suppose you are trying to start a company running a phone-number lookup service and are considering setting up a server on a cheap computer you happen to have access to. The computer has 2 GB of RAM memory and four cores that run 64-bit instructions at 2 GHz, so you can estimate that it can execute in the order of 10^8 to 10^9 elementary CPU operations per second.

The primary service you want to start with is to be able to associate a phone number with a name. To make the lookups fast, you decide to use a hash-table. So how many phone records (number plus name) can your server store and how many lookups per second can it handle? Make some reasonable assumptions on your data, your computer's performance, and estimate the capacity of your server. (8p)

4. There are many possible variants of Quicksort. In this assignment you will decide on one variant and visualise how it works. Your Quicksort variant should work with any input types supporting a comparison operator $\prec$.

(a) Write pseudocode for a function `partition` that, given an array A of elements comparable using some order operator $\prec$ (for numbers, one would use $<$), returns an index k, and has changed the elements A such that $A[i] \preceq A[k]$ (i.e., $\prec$ or $=$) for $i < k$ and $A[i] \succ A[k]$ for $i > k$. The time complexity of `partition` must be $O(n)$. State clearly what principle you are using to choose k, what advantage it has, and what possible disadvantage it has. (5p)

(b) Write pseudocode for a recursive Quicksort using your `partition` function. (2p)

(c) Visualize how *your* Quicksort variant works on the following input, with $\prec$ being alphabetical order, by showing how A changes. Do not use more than one page for this visualisation. Show the workings for at least three levels of recursion. (3p)

HIF	IFKN	OFK	KFF	IKS	AIK	VBoIS	BKH	DIF	FFF	HIF	IFKG	MAIF	IFE	OSK	MFF

5. Propose a *computational problem* for the following loosely formulated need from an orienteering club. (See the Wikipedia article on the sport *orienteering* if you need background.)

> "We need help designing orienteering competitions for juniors. Kids get confused when controls are close to each other and lose interest if the run from one checkpoint to the next is more than 500 m. Checkpoints should also be close to marked points on our orienteering maps, like large rocks, path intersections, etc. How should we set this up? We have a good set of checkpoints set up now, but we don't want the same course for every practice and competition."

You do not need to solve the computational problem, but it should be adequately and suitably formulated. (5p)

6. The EBNF grammar below describes a part of the DOT format for describing graphs in text files (see Wikipedia for a simple description). Most terminals below are quoted strings (e.g., `"digraph"`), but an extra terminal for vertex identifiers is also used: `VERTEX`. Comments are written with as `some comment`.

The grammar describes for example the empty graph:

```
graph The_empty_graph {
}
```

A simple three vertex directed graph can be written as

```
digraph Triangle {
  a -> b;
  b -> c;
  c -> a;
}
```

or as

```
digraph Triangle {
  a -> b -> c -> a;
}
```

(a) Explain why we cannot design a recursive descent parser based on the following grammar. (2p)

(b) Fix the grammar such that it enables constructing a recursive descent parser from it. (3p)

```
graph = "digraph" graph_name "{" arc_list  "}"; (* directed graph *)
      | "graph"   graph_name "{" edge_list "}"; (* undirected graph *)
      ;

arc_list = (* the empty list *)
         | arc_list one_or_more_arcs
         ;

one_or_more_arcs = VERTEX "->" one_or_more_arcs
                 | VERTEX "->" VERTEX ";"

edge_list = (* the empty edge list *)
          | edge_list one_or_more_edges
          ;

one_or_more_edges = VERTEX "-" one_or_more_edges
                  | VERTEX "-" VERTEX ";"
```