Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2021-09-10
Due: 2021-09-17

ICS 2021 Problem Sheet #1

Problem 1.1: *minimum spanning trees* (3 points)

We have introduced Kruskal's algorithm for constructing random spanning trees (maze solutions). Edges are selected randomly and added to the spanning tree as long as the nodes connected by the edges belong to different equivalence classes. The original algorithm solves a slightly more difficult problem: Given a graph $G = (V, N)$ and a cost function c that indicates the cost of including the edge $e \in N$ in the spanning tree, calculate the spanning tree $S = (V, E)$ such that $C = \sum_{e \in E} c(e)$ is minimal (also called a minimum spanning tree). Kruskal's algorithm solves this problem by selecting in each step an edge that joins two equivalence classes and has the minimum cost of all edges still available.

You are given the graph $G = (V, N)$ with

$$V = \{a, b, c, d, e, f\}$$
$$N = \{(a, b), (a, e), (a, f), (b, c), (b, f), (c, d), (c, f), (d, e), (d, f), (e, f)\}$$

and the cost function c defined by the following table:

edge e:	(a,b)	(a,e)	(a,f)	(b,c)	(b,f)	(c,d)	(c,f)	(d,e)	(d,f)	(e,f)
cost $c(e)$:	10	9	8	7	6	5	4	3	2	1

Construct a minimal spanning tree $S(V, E)$ using Kruskal's algorithm. For each step, write down the set of equivalence classes A and the edges in E. What is the overall cost C of the resulting spanning tree? You start with:

$E = \{\}$ initialization, $C = 0$
$A = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}\}$

$E = \ldots$ step 1, $C = \ldots$
$A = \ldots$

$E = \ldots$ step 2, $C = \ldots$
$A = \ldots$

$\ldots$

Solution:

$E = \{\}$ $A = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}\}$	initialization, $C = 0$
$E = \{(e, f)\}$ $A = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e, f\}\}$	step 1, $C = 1$
$E = \{(e, f), (d, f)\}$ $A = \{\{a\}, \{b\}, \{c\}, \{d, e, f\}\}$	step 2, $C = 3$
$E = \{(e, f), (d, f), (c, f)\}$ $A = \{\{a\}, \{b\}, \{c, d, e, f\}\}$	step 3, $C = 7$
$E = \{(e, f), (d, f), (c, f), (b, f)\}$ $A = \{\{a\}, \{b, c, d, e, f\}\}$	step 4, $C = 13$
$E = \{(e, f), (d, f), (c, f), (b, f), (a, f)\}$ $A = \{\{a, b, c, d, e, f\}\}$	step 5, $C = 21$

Marking:

- *0.5pt for each correct step (steps 1-5)*
- *0.5pt for the correct overall cost C (last step)*

Problem 1.2: *boyer moore algorithm* (2+2+2 = 6 points)

You have designed a simple robot that can turn left (L), turn right (R), move one step forward (F), and pause (P) for short time. The robot is programmed by a sequence of robot instructions. For example, the sequence FFLFLFRFRFFLFRF will direct the robot through the maze shown on the slides discussing maze generation algorithms. Using the Boyer Moore algorithm, we can determine whether a robot program contains certain movement sequences.

Let $\Sigma = \{L, R, F, P\}$ be an alphabet and $t \in \Sigma^*$ be a text of length n describing a program for the robot. Let $p \in \Sigma^*$ be a pattern of length m. We are looking for the first occurrence of p in t.

Consider the text $t = FFLFLFRFRFFLFRF$ and the pattern $p = FFLFR$.

a) Execute the naive string search algorithm. Show all alignments and indicate comparisons performed by writing uppercase characters and comparisons skipped by writing lowercase characters. How many alignments are used? How many comparisons are done?

b) Execute the Boyer-Moore string search algorithm with the bad character rule only. How many alignments are used? How many comparisons are done?

c) Calculate the lookup table for the bad character rule that indicates the number of alignments that can be skipped if a comparison does not match.

Solution:

a) Naive string search (10 alignments, 22 comparisons)

```
F F L F L F R F R F F L F R F
F F L F R                             // default shift
  F F l f r                           // default shift
    F f l f r                         // default shift
```

```
      F F l f r                               // default shift
        F f l f r                             // default shift
          F F l f r                           // default shift
            F f l f r                         // default shift
              F F l f r                       // default shift
                F f l f r                     // default shift
                  F F L F R                   // match
```

b) Boyer-Moore with bad character rule only (6 alignments, 16 comparisons)

```
F F L F L F R F R F F L F R F
f f l f R                                     // skip 1
    F F L F R                                 // default shift
      f f l f R                               // default shift
        f f L F R                             // skip 2
              f f l f R                       // skip 1
                  F F L F R                   // match
```

c) The bad character rule lookup table indicating alignments that can be skipped looks as follows:

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline L & 0 & 1 & - & 0 & 1 \\ R & 0 & 1 & 2 & 3 & - \\ F & - & - & 0 & - & 0 \\ P & 0 & 1 & 2 & 3 & 4 \end{array}$$

Marking:

a) - *1pt for a correct execution of the naive search algorithm*
- *1pt for correctly counting the number of alignments and comparisons*

b) - *1pt for a correct execution of the Boyer-Moore algorithm*
- *1pt for correctly counting the number of alignments and comparisons*

c) - *0.5pt for every correct row in the lookup table*

Problem 1.3: *big O notation* (1+1 = 2 points)

a) Sort the functions

$$
\begin{aligned}
f_1(n) &= \frac{1}{2} n \log n \\
f_2(n) &= n^2 \\
f_3(n) &= \sqrt{n^3} \\
f_4(n) &= n^n \\
f_5(n) &= 100n^2 + 10n^3 \\
f_6(n) &= 2 \log n \\
f_7(n) &= (n^2)^2 \\
f_8(n) &= \log \log n
\end{aligned}
$$

in increasing order concerning their big O membership. (It is sufficient to provide the correct order.)

b) Given the functions $f, g, h : \mathbb{N} \to \mathbb{N}$, show that the following transitivity property holds: If $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.

Solution:

a) The order is: $f_8 < f_6 < f_1 < f_3 < f_2 < f_5 < f_7 < f_4$

b) Lets first write out what the two statements $f \in O(g)$ and $f \in O(h)$ mean according to the definitions we have:

- There exist a $k_f \in \mathbb{N}$ and an $n_f \in \mathbb{N}$ such that $f(n) \leq k_f \cdot g(n)$ for all $n > n_f$.

$$\exists k_f, n_f \in \mathbb{N}. \forall n > n_f : f(n) \leq k_f \cdot g(n)$$

- There exist a $k_g \in \mathbb{N}$ and an $n_g \in \mathbb{N}$ such that $g(n) \leq k_g \cdot h(n)$ for all $n > n_g$.

$$\exists k_g, n_g \in \mathbb{N}. \forall n > n_g : g(n) \leq k_g \cdot h(n)$$

We have show:

- There exist a $k_t \in \mathbb{N}$ and an $n_t \in \mathbb{N}$ such that $f(n) \leq k_t \cdot h(n)$ for all $n > n_t$.

$$\exists k_t, n_t \in \mathbb{N}. \forall n > n_t : f(n) \leq k_t \cdot h(n)$$

We choose $k_t = k_f \cdot k_g$ and $n_t = \max\{n_f, n_g\}$:

- For all $n > n_t$, the following holds:

$$\begin{aligned} f(n) &\leq k_f \cdot g(n) && \text{definition of } f \in O(g) \text{ with } n \geq n_f \\ &\leq k_f \cdot (k_g \cdot h(n)) && \text{definition of } g \in O(h) \text{ with } n \geq n_g \\ &\leq (k_f \cdot k_g) \cdot h(n) && \text{commutativity of multiplication} \\ &\leq k_t \cdot h(n) \end{aligned}$$

Marking:

a) - *-0.2pt for each incorrect order relationship*

b) - *1pt for a logically correct reasoning (does not have to be written in mathematical notation)*
- *0.5pt if the idea is correct but the writing is unclear or wrong*
- *0pt if only an example is given instead of a generic statement*