Analysis of Algorithms: Solutions 5

<table>
<thead>
<tr>
<th>number of homeworks</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>grades</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Problem 1
Give a recursive version of the TREE-INSERT procedure.

The following procedure inserts a new node $z$ into a subtree rooted at a node $x$. Initially, $x$ must be the root of a tree; that is, in order to insert $z$ into a tree $T$, we make the top-level call_recursive-insert(root[T], z). For simplicity, we assume that the tree is not empty.

**Recursive-Insert**($x, z$)

if $key[z] < key[x]$ and left-child[$x]$ ≠ NIL
  then **Recursive-Insert**(left-child[$x$], z)

if $key[z] ≥ key[x]$ and right-child[$x$] ≠ NIL
  then **Recursive-Insert**(right-child[$x$], z)

$parent[z] ← x$

if $key[z] < key[x]$
  then left-child[$x$] ← $z$
else right-child[$x$] ← $z$

Problem 2
Suppose we apply the CONNECTED-COMPONENTS algorithm to an undirected graph $G$, with vertices $G[V] = \{a, b, c, d, e, f, g, h, i, j, k\}$, and its edges $E[G]$ are processed in the following order: $(d, i), (f, k), (g, i), (b, g), (c, e), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e)$. Using Figure 22.1 in the textbook as a model, illustrate the steps of CONNECTED-COMPONENTS on this graph.

The final result of applying CONNECTED-COMPONENTS is the following three sets:

$$\{a, c, e\} \quad \{b, d, f, g, i, j, k\} \quad \{h\}$$
Problem 3
Write pseudocode for MAKE-SET, FIND-SET, and UNION, using the linked-list representation of disjoint sets. UNION must always append the shorter list to the longer one.

We use four fields for each element $x$ of a linked list:

- $next[x]$: pointer to the next element of the list; NIL if $x$ is the last element
- $rep[x]$: pointer to the set representative, that is, to the first element of the list
- $last[x]$: if $x$ is the first element of a list, then this field points to the last element
- $size[x]$: if $x$ is the first element, then this field contains the size of the list

If $x$ is not the first element of a list, then the algorithms do not use its $last$ and $size$ fields, and the information in these fields may be incorrect.

**MAKE-SET($x$)**

```plaintext
next[x] ← NIL
rep[x] ← x
last[x] ← x
size[x] ← 1
```

**FIND-SET($x$)**

```plaintext
return rep[x]
```

**UNION($x$)**

```plaintext
if size[rep[x]] > size[rep[y]]
    then APPEND(rep[x], rep[y])
    else APPEND(rep[y], rep[x])
```

**APPEND($x$, $y$)**

```plaintext
next[last[x]] ← y
size[x] ← size[x] + size[y]
z ← y
while $z ≠ NIL$  ▷ change the rep pointers in the second list
    do rep[z] ← x
        z ← next[z]
```

Problem 4
Suppose that you are using a programming language that allows only integer numbers and supports three operations on them: addition, subtraction, and multiplication. Write an efficient algorithm DIVIDE($n$, $m$) that computes $\lfloor n/m \rfloor$, where $n$ and $m$ are positive integers.

**Simple algorithm**
The following brute-force computation takes $\Theta(\lfloor n/m \rfloor + 1)$ time and $\Theta(1)$ space.

**SIMPLE-DIVIDE($n$, $m$)**

```plaintext
ratio ← 0
while ratio $\cdot m ≤ n$
    do ratio ← ratio + 1
return ratio – 1
```
Fast algorithm
The following recursive algorithm runs is $\Theta(\lg[n/m] + 1)$ time. The space complexity is not constant: the algorithm requires $\Theta(\lg[n/m] + 1)$ memory for the stack of recursive calls.

Fast-Divide$(n, m)$
if $n < m$
    then return 0
ratio ← 2 · Fast-Divide$(n, 2 \cdot m)$
if $n - ratio < m$
    then return ratio
else return ratio + 1