Errata to
Introduction to Algorithms and Data Structures
Markus Bläser February 2, 2012

Section 1.3, Definition 1.2 (p. 5) 3. $\Theta(f) = O(f) \cap \Omega(f) \Omega(g)$.

Section 1.4.2, Lemma 1.10 (p. 8) Let $g_1, \ldots, g_\ell : \mathbb{N} \rightarrow \mathbb{N}$ be functions such that ... reported on 2012-01-11

Section 3.1.1 (p. 17) The last layer might be shorter and is stored in $A[2^{h-1}..\text{heap-size}]$. Here $h = \log(\text{heap-size}) - 1 = \lfloor \log_2(\text{heap-size}) \rfloor$ is the height of the tree, [...]. reported on 2012-01-25

Section 3.1.3 (p. 19) Now assume we have an array $A[1..n]$ and we want to convert it into a heap. We can use the procedure Heapify in a bottom-up manner. Because the indices $[\lfloor n/2 \rfloor, \ldots, n]$ are all leaves, the $[\lfloor n/2 + 1 \rfloor, \ldots, n]$ all represent leaves, each subtree with a root $j > \lfloor n/2 \rfloor$ at $j > \lfloor n/2 \rfloor$ is a heap. Then we apply Heapify and ensure the heap property in a layer by layer fashion. The correctness of the approach can be easily proven by reverse induction on $i$. reported on 2011-11-18

Section 4.2, Proof of Lemma 4.4 (p. 28) Since $m$ is the median of the medians, $\lceil \frac{1}{2}(r-1) \rceil$ medians are larger and $\lceil \frac{1}{2}(r-1) \rceil$ medians are smaller than $m$. reported on 2011-11-23

Section 4.2, before Remark 4.6 (p. 29) We can use Lemma 4.5 to solve (4.1). We can bound $\frac{1}{15}n + 2$ from above by $\frac{11}{15}n$ for $n > 60$. Since $\frac{1}{5} + \frac{11}{15} + \frac{2}{60} = \frac{29}{30} < 1$, we get that $t(n) \leq c \cdot n$ with $c = \max\{d - (\epsilon_1 + \epsilon_2 + + \frac{\ell}{n}), e\} = \max\{\frac{17}{15}, 8\} = 102$.

The parameter choices corresponding to equation (4.1) are

- $\ell = 2$, $\epsilon_1 = \frac{1}{5}$, $\epsilon_2 = \frac{11}{15}$, $d = \frac{17}{5}$, $N = 60$, $e = 8$.

Thus, $c = \max\{\frac{d}{1 - (\epsilon_1 + \epsilon_2 + + \frac{\ell}{n})}, e\} = \max\{\frac{17/5}{1-29/30}, 8\} = 102$.

Section 6.1 (p. 38) If $\text{Key}(r) = k^+$, then we are done. reported on 2011-12-12
Algorithm 26 BST-search

**Input:** node \( x \), key \( k \)

**Output:** a node \( y \in T(x) \) with \( \text{Key}(y) = k \) if such a \( y \) exists, NULL otherwise

1: if \( x = \text{NULL} \) or \( k = \text{Key}(x) \) then
2: \( k = \text{Key}(x) \)
3: if \( k < \text{Key}(x) \) then
4: return BST-search(Left(\( x \)), \( k \))
5: else
6: return BST-search(Right(\( x \)), \( k \))

Section 7.1, Proof of Lemma 7.2 (p. 45)  We show by induction on \( h \) that...

Section 7.2.2, before Observation 7.4 (p. 46–47) ...a virtual leaf is replaced by an internal node...

Section 7.2.2, first table (p. 48)

<table>
<thead>
<tr>
<th></th>
<th>before insertion</th>
<th>after insertion</th>
<th>after rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Bal}(x) )</td>
<td>(-1)</td>
<td>(-2)</td>
<td>(-0)</td>
</tr>
<tr>
<td>( \text{Bal}(y) )</td>
<td>(0)</td>
<td>(-1)</td>
<td>(0)</td>
</tr>
<tr>
<td>( \text{Height}(T_1) )</td>
<td>(h)</td>
<td>(h)</td>
<td>(h)</td>
</tr>
<tr>
<td>( \text{Height}(T_2) )</td>
<td>(h+1)</td>
<td>(h)</td>
<td>(h)</td>
</tr>
<tr>
<td>( \text{Height}(T_3) )</td>
<td>(h+2)</td>
<td>(h+1)</td>
<td>(h+1)</td>
</tr>
<tr>
<td>( \text{Height}(T(x)) )</td>
<td>(h+2)</td>
<td>(h+1)</td>
<td>(h+1)</td>
</tr>
<tr>
<td>( \text{Height}(T(y)) )</td>
<td>(h+3)</td>
<td>(h+2)</td>
<td>(h+2)</td>
</tr>
</tbody>
</table>

All numbers in rows 4–7 were decreased by exactly one.

Section 9.1 (p. 61)  Of course, in the worst case, every bit has to be changed to \( 0 \) and we have to flip all \( \ell \) bits (and get an overflow error).

Section 9.1.1 (p. 62)  Therefore, the total time is

\[
t(n) = \sum_{i=0}^{\ell-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq n \cdot \sum_{i=0}^{\ell-1} \frac{1}{2^i} \leq n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n
\]
Chapter 10, Theorem 10.1 (p. 71)

3. If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some \( \epsilon > 0 \) and \( a f([n/b]) \leq df(n) \) \( a f([n/b]) \leq df(n) \) for some constant \( d < 1 \) and all sufficiently large \( n \), then \( t(n) = O(f(n)) \).

Chapter 10, Exercise 10.1 (p. 71) Let \( f : \mathbb{N} \rightarrow \mathbb{N}, f \neq 0 \). Show that if \( f \) fulfills \( f([n/b]) \leq df(n) \) \( a f([n/b]) \leq df(n) \) for some constant \( d < 1 \) and all sufficiently large \( n \), then \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some \( \epsilon > 0 \).

Chapter 10, Proof of Theorem 10.1 (p. 72) We start with the first two cases. Let \( e := \log_b a \) and \( \gamma := a/b^e \), respectively.

Section 11.1, before Exercise 11.1 (p. 74) The chromatic number \( \chi(G) \) of a graph \( G \) is the smallest number \( k \) such that there is a proper \( k \)-coloring of \( G \).

Section 11.1, after Exercise 11.1 (p. 74) \( \ldots \) how can we decide whether \( G \) has a proper \( k \)-coloring? First, we can try all proper \( k \)-colorings.

Chapter 12, after Exercise 12.1 (p. 80) A cycle is a walk such that \( v_0 = v_k \), \( k > 0 \) (if \( G \) is directed) or \( \rightarrow_k \), \( k > 2 \) (if \( G \) is undirected), \( \ldots \)

Section 12.1 (p. 81 bottom) With an adjacency-list matrix-representation, however, \( \ldots \)

Section 12.2.2 (p. 85) If we have an adjacency-matrix representation, then the running time is \( O(|V|^2) \).

Section 13.2, Proof of Theorem 13.2 (p. 90) \( \ldots \) It remains to prove why this spanning tree is in fact minimal. Assume that \( e \) is not of minimal weight, i.e. there exists an edge \( f \) with lower weight. Thus, \( f \) would have been handled by the algorithm before \( e \) (line 5). Since \( S \) is a connected component of \( E_T \) it holds that \( E_T \cup \{e\} \) used to be acyclic for all previous iterations of the algorithm. But then, \( f \) would have already been added to \( E_T \), contradicting the fact that \( f \) is an edge of the cut of \( S \).

Hence, no \( f \) with lower weight exists, so \( e \) is an edge of minimal weight in the cut \((S,V \setminus S)\), and by Theorem 13.1, the spanning tree augmented by \( e \) is minimal.
Section 14.1, Algorithm 52 (p. 94)

Algorithm 52 Relax

Input: nodes u and v with \((u, v) \in E\)

Output: \(d[v]\) and \(p[v]\) are updated

\[
\text{if } d[v] > d[u] + w((u, v)) \text{ then} \\
\quad d[v] := d[u] + w((u, v)) \\
\quad p[v] := u
\]

---

Section 14.2, after Algorithm 53 (p. 95)  
If we implement \(Q\) by an ordinary array, then the Insert and Decrease-min Decrease-key operations that take time \(O(1)\) while Extract-min takes \(O(|V|)\). [...] If we implement \(Q\) with binary binomial or binary heaps, then ...