1 CS124 Section 7: Amortization and Hashing

Ted Pyne

Problem 1.1. Consider the sequence of operations:

\text{insert}(2), \text{insert}(3), \text{insert}(1), \text{deletemin}(), \text{insert}(5), \text{insert}(0), \text{deletemin}(), \text{insert}(6)

What does the data structure look like?

Proof. We track \( p, A \) and the HoH after each operation:

<table>
<thead>
<tr>
<th>t</th>
<th>p</th>
<th>A</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[2]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[2,3]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td>[2,3]</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>[5]</td>
<td>[2,3]</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>[5,0]</td>
<td>[2,3]</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td>[2,3],[5]</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>[6]</td>
<td>[2,3],[5]</td>
</tr>
</tbody>
</table>

Problem 1.2. Suppose we want to implement a dynamically sized array. Let the current number of elements be \( k \), and let \( m \) be the size of the current array. As additional appendLast() and removeLast() operations occur:

1. If \( k = m \), allocate a new array of size 2\( m \) and move all elements over.
2. If \( k = m/4 \), allocate a new array of size \( m/2 \) and move all elements over.

Assuming moving a single element takes \( O(1) \) time, show this data structure has \( O(1) \) amortized runtime for appendLast() and removeLast() operations.

Proof. We define a potential function there \( p \) is equal at all times to the number of appendLast or removeLast operations since the last resize. For each appendLast or removeLast operation that does not trigger a resize, implement the operation and increase \( p \) by 4. Both the real work and \( \Delta p \) are \( O(1) \).

For an operation that triggers a resize, we claim \( p \geq m/2 \). Let \( m \) be the current size of the array. When \( p \) was last set to zero, we had \( |k - m| \geq \min(m - m/2, m/2 - m/4) = m/4 \) by the description of the algorithm. Since moving the array takes \( m \) work, by setting \( p \) to zero we obtain an amortized resize runtime of

\[
m - \Delta p + O(1) = m - 4(m/4) + O(1) = O(1)
\]

As desired.

Problem 1.3. Assume \( u = \{0,1\}^s \) and \( m = \{0,1\}^k \) with + and * defined bitwise. Then let \( M \) be the set of all matrices with entries in \( \{0,1\} \) with \( k \) rows and \( s \) columns. For all \( A \in M \), define

\[
h_A(x) = Ax.
\]

Show \( \{h_A : A \in M\} \) is a hash family.
Proof. We wish to show that for all \(x, y \in \{0, 1\}^s\) where \(x \neq y\) that

\[
\Pr_h[h(x) = h(y)] = 1/m.
\]

By rearranging, this is equivalent to showing for \(x \neq y\):

\[
\Pr_{A \leftarrow M}[A(x - y) = 0] = 1/m.
\]

To show this, we remark that since \(x \neq y\) there is at least one bit, WLOG the first, where \(x - y\) is nonzero. Then for every \(A\) let \(A_i\) be the \(i\)th column of \(A\). We have:

\[
A(x - y) = A_1 + \sum_{i=2}^{s} A_i(x - y)_i
\]

Treating \(A_2, \ldots, A_s\) as fixed, we have that \(A(x - y) = A_1 + v\) where \(v\) is some fixed vector. But \(A_1\) is exactly uniform over \(\{0, 1\}^k\), and the distribution of a uniform vector over \(\mathbb{F}_2\) plus a constant vector is uniform, so \(A(x - y)\) is uniformly distributed over \(\{0, 1\}^k\) and we have the desired result. \(\square\)