**Scope for Quiz 4**

- Sections 4.5 through 4.7
- Chapter 5 of DPV.

**Practice Problems**

**Problem 1.** PG Problem 542: Give a procedure that given:

1. \( n \geq 1 \),
2. \( i \in \{ 1, \ldots, n \} \), and
3. \( A[1..m] \) the array representation of an \( n \)-element max-heap (where \( m \geq n \)),
deletes the element \( A[i] \) from the heap in \( O(\log n) \) time and returns the updated array \( A \). Justify the \( O(\log n) \) run time.

**An answer.**

```markdown
swap A[i] and A[n]
// Now A[i] may be too small for the max-heap
// property to hold at i
sinkDown(i) // This restores the max-heap prop.
return A
```

Since sinkDown is \( O(\log n) \), so is the above.

**Problem 2.** PG, Supplemental problems, Greedy Algorithms, Problem 11 (on page 47): Prove the following lemma.

Suppose \( G \) is an undirected graph and \( e \) is the unique most expensive edge on a cycle in \( G \). Then \( e \) is not on any minimal-spanning tree for \( G \).

**An answer.** Suppose by way of contradiction that \( e \) is part of \( T \), a minimal spanning tree of \( G \). Let \( e' \notin T \) be an edge on a cycle with \( e \). Let \( T' = (T - \{ e \}) \cup \{ e' \} \). Since \( e \) and \( e' \) are both on the same cycle, it follows that \( T' \) is connected. Since \( T \) has \( |V| - 1 \) many edges, \( T' \) also has \( |V| - 1 \) many edges. Hence, since \( T' \) is connected, \( T' \) must be a tree. Also, \( \text{cost}(T') - \text{cost}(T) = \text{cost}(e') - \text{cost}(e) < 0 \). So, \( T' \) is a spanning tree with a lower cost than \( T \), a contradiction.

**Problem 3.** *This is a problem from CLRS.*

Professor Midas drives an automobile from Newark to Reno along Interstate 80. His car’s gas tank, when full, holds enough gas to travel \( n \) miles and his map gives the distances between gas stations along his route. The professor wishes to make as few gas stops as possible along the way. The professor wants an efficient method by which he can determine at which gas stations he should stop.

(a) Prove that there is an optimal solution to this problem which begins with stopping at the gas station on I80 which the last station you would pass before travelling more than \( n \) miles from Newark.

**An answer.** Suppose \( S \) is an optimal route which involves the sequence of stops \( s_0, s_1, \ldots, s_k \) where \( s_i = \) the miles from Newark of the \( i \)-th stop, \( s_0 = \) Newark, and \( s_k = \) Reno. Let \( s'_1 \) be the distance from Newark of the last station before going more than \( n \) miles. If \( s_1 = s'_1 \) we are done. So suppose \( s_1 \neq s'_1 \). Then because of the car’s \( n \)-mile limit we have: (i) \( s_1 < s'_1 \) and (ii) \( s_2 - s_1 \leq n \). Hence, \( s_2 - s'_1 \leq n \). So \( S' = s_0, s'_1, s_2, \ldots, s_k \) is feasible and has the same number of stops as \( S \). Hence, \( S' \) is optimal.

(b) Suppose \( S \) is an optimal route which involves the sequence of stops \( s_0, s_1, \ldots, s_k \) where \( s_i = \) the miles from Newark of the \( i \)-th stop, \( s_0 = \) Newark, and \( s_k = \) Reno. Show that for each \( i \) with \( 1 < i < k - 1 \), that \( s_0, s_1, \ldots, s_i \) is an optimal solution to the problem of travelling from \( s_0 \) to \( s_i \) making as few stops as possible, and that \( s_i, \ldots, s_k \) is an optimal solution to the problem of travelling from \( s_i \) to \( s_k \) making as few stops as possible.

**An answer.** Let \( S \) be any feasible solution to the \( s_0 \) to \( s_i \) problem and let \( S' \) be any feasible solution to the \( s_i \) to \( s_k \) problem. The only stop \( S \) and \( S' \) have in common is \( s_i \). Hence, whatever choices we make for \( S \), they have no effect on the choices we can make for \( S' \) and visa versa. Hence, an optimal solution that involves \( s_i \) must be made up of optimal solutions of the \( s_0 \) to \( s_i \) problem and the \( s_i \) to \( s_k \) problem.

(c) Suppose that there are \( m \) gas stations on I80 and \( g[1..m] \) is an array with \( g[i] = \) the distance from Newark to the \( i \)-th station and that

\[ g[0] < g[1] < g[2] < \ldots < g[m], \]
and $g[m]$ is a gas station in Reno. Give an $O(m)$ time greedy algorithm for determining Professor Midas’s stops.

**An answer.**

$S \leftarrow [0] / \text{The starting point}$
$t \leftarrow g[m] / \text{the total miles to Reno}$
$miles \leftarrow 0; \ i \leftarrow 1$

**while** miles $< t - n$ **do**

**while** $g[i] \leq miles + n$ **do** $i \leftarrow i + 1$ **endWhile**

$miles \leftarrow g[i - 1]$

**Add** $g[i - 1]$ to the end of the $S$ list

**endWhile**

return $S$

(d) Use parts (a) and (b) to prove that your algorithm of (c) produces an optimal solution.

**An answer.** Let $S = s_0 < \cdots < s_k$ be the final list of stops. **Claim:** For each $i = 1, \ldots, k$ $s_0 < \cdots < s_i$ is part of some optimal solution to the problem. **Proof by induction on $i$.** **Base case:** $i = 1$. This follows by part (a). **Induction step:** $i > 1$. **IH:** $s_0 < \cdots < s_{i-1}$ is part of an optimal solution to the problem. Then it follows from part (a) again that some optimal solution for the $s_{i-1}$ to Reno problem must start with $s_i$. Then by part (c) it follows that $s_0 < \cdots < s_i$ must be part of an optimal solution. Hence the claim follows. Finally, since $s_k + n \geq g[m]$, $S$ is a feasible solution to the problem. Hence, by the claim, $S$ is an optimal solution.

### Some Reference Math Facts

<table>
<thead>
<tr>
<th>a. $a^n \cdot a^m = a^{m+n}$</th>
<th>f. $c \cdot \log_b a = \log_b a^c$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b. $a^{mn} = (a^m)^n = (a^n)^m$.</td>
<td>g. $\log_2(a \cdot b) = (\log_2 a) + (\log_2 b)$.</td>
</tr>
<tr>
<td>c. $a^n \cdot b^n = (a \cdot b)^n$.</td>
<td>h. $(\log_b x)/\log_a b = \log_a x$.</td>
</tr>
<tr>
<td>d. $\log_2 a^n = n$.</td>
<td>i. $a^{\log_b c} = c^{\log_b a}$.</td>
</tr>
<tr>
<td>e. $a^{\log_b n} = n$.</td>
<td>j. $\sum_{k=1}^{n} (\frac{1}{k} \cdot n \cdot (n+1))$.</td>
</tr>
<tr>
<td></td>
<td>k. $\sum_{k=0}^{n} a^k = \frac{a^{n+1} - 1}{a-1}$.</td>
</tr>
<tr>
<td></td>
<td>l. $\sum_{k=1}^{n} 2^{-k} = 1$.</td>
</tr>
<tr>
<td></td>
<td>m. $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$.</td>
</tr>
<tr>
<td></td>
<td>n. $\sum_{k=1}^{n} k \cdot 2^{-k} = 2$.</td>
</tr>
<tr>
<td></td>
<td>o. For $a &gt; 1$ and $b, c &gt; 0$: $\lim_{n \to \infty} \frac{(\log_a n)^b}{n^c} = 0$.</td>
</tr>
<tr>
<td></td>
<td>p. For $a &gt; 1$ and $b, c &gt; 0$: $\lim_{n \to \infty} \frac{n^b}{a^c n} = 0$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q. If $\lim_{n \to \infty} f(n)/g(n) = c$, then:</th>
<th>r. The Master Recurrence Theorem: Suppose that for all $n &gt; n_0$, we have $T(n) = a \cdot T(n/b) + c \cdot n^k$ for $a, b \geq 1$ and $k \geq 0$, then:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = 0$</td>
<td><strong>If</strong> $a &lt; b^k$, then $T(n) \in \Theta(n^k)$.</td>
</tr>
<tr>
<td>$0 &lt; c &lt; \infty$</td>
<td><strong>If</strong> $a = b^k$, then $T(n) \in \Theta(n^k \log n)$.</td>
</tr>
<tr>
<td>$c = \infty$</td>
<td><strong>If</strong> $a &gt; b^k$, then $T(n) \in \Theta(n^{\log_b a})$.</td>
</tr>
</tbody>
</table>

Above, $\frac{n}{b}$ can be taken as $\lfloor \frac{n}{b} \rfloor + c$ or $\lceil \frac{n}{b} \rceil + c$ for constant $c$.

**Definition:** A tree is an acyclic, connected undirected graph.

**Tree properties.** Suppose $G$ is an undirected graph.

1. Deleting an edge from a cycle cannot disconnect a graph.
2. A tree on $n$ vertices has $n - 1$ edges.
3. If $G = (V, E)$ is connected and $|E| = |V| - 1$, then $G$ is a tree.
4. Suppose $G$ is a tree $\iff$ there is a unique path between any two vertices in $G$.

**Heap Utility Operations.**

**procedure** bubbleUp($ptr$)

**while** ($ptr \neq \text{root}$) $\&\&$ ($ptr$’s key $\geq$ parent($ptr$)’s key) **do**

swap $ptr$’s key with its parent’s key

$ptr \leftarrow \text{parent}(ptr)$

return $ptr$

**procedure** sinkDown($ptr$)

$q \leftarrow ptr; \ \ell \leftarrow \text{left}(ptr); \ r \leftarrow \text{right}(ptr)$

**while** ($q$ is not a leaf) **do**

if $\ell \neq \text{null}$ $\&\&$ $q$’s key $> \ell$’s key then $q \leftarrow \ell$

if $r \neq \text{null}$ $\&\&$ $q$’s key $> r$’s key then $q \leftarrow r$

if $q = \text{ptr}$ then return

swap $ptr$’s key and $q$’s key

$ptr \leftarrow q$

return $ptr$