15-750 Graduate Algorithms, Spring 2019
Homework 1 (100 pts) Due: February 13
Gary Miller TAs: David Wajc, Ruosong Wang

Instructions. Collaboration is permitted in groups of size at most three. You must write the names of your collaborators on your solutions and must write your solutions alone.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

(20) 1. k-Clique

In an undirected graph, a k-clique is a set of k distinct vertices such that there is an edge between every pair of these vertices.

a) Give an $O(n^k)$-time algorithm for deciding whether a graph contains any k-clique.
b) Let $A$ be the adjacency matrix of a graph $G = (V, E)$,

$$A_{i,j} = \begin{cases} 
1 & \text{if there is an edge between } i \text{ and } j \\
0 & \text{otherwise}
\end{cases}$$

What is $A^r_{i,j}$ for integer $r \geq 1$?
c) Give an $O(n^{2.99})$-time algorithm for deciding whether a graph contains any 3-clique.
d) Give an $O(n^{119.99})$-time algorithm for deciding whether a graph contains any 120-clique.

(20) 2. Melding Two Treaps

In this problem we will consider the problem of taking two treaps, $T_1$ and $T_2$, and forming a single treap, whose keys are precisely the union of the sets of keys of $T_1$ and $T_2$, with the same priority for each key in the new treap as in this element’s initial treap. We will
assume that all the keys in $T_1$ are strictly smaller than the keys in $T_2$, and that all the priorities of both $T_1$ and $T_2$ are independent. Let $n_1$ and $n_2$ be the number of keys in $T_1$ and $T_2$, respectively.

a) Write recursive pseudo code to meld $T_1$ and $T_2$, say, $Meld(T_1, T_2)$. Feel free to use pictures in your code and the following functions:

- $Root(T)$ = the root of $T$.
- $P(node)$ = the priority of node.
- $L(T)$ = the left subtree of $T$.
- $R(T)$ = the right subtree of $T$.

b) As a starting point to analyze your algorithm, give an asymptotic formula for the expected length of the right-most path in $T_1$ (i.e. the path down the tree starting at the root and following right children until reaching either a leaf or a node with only a left child). You should use the following random variable.

$$ V_i = \begin{cases} 
1 & \text{if } k_i \text{ is on the right-most path of } T_1. \\
0 & \text{otherwise}
\end{cases} $$

c) Use your answer to Part b to prove an upper bound of $O(\log n_1 + \log n_2)$ on the number of priority comparisons and overall running for your meld operation.

3. Finding a Rooted Vertex Set In a Tree

Given a tree $T = (V, E)$ and budget $B$. Each vertex $v \in V$ has weight $w_v > 0$ and cost $c_v > 0$. Find a set of vertices $S \subseteq V$ and an extra vertex $r \in V$ such that

a) All vertices in $S$ are descendants of $r$.

b) The total cost of vertices in $S$ is at most $B$, i.e.,

$$ \sum_{v \in S} c_v \leq B. $$

c) Maximize $w_r \cdot |S|$, where $|S|$ is the size of the set $S$ (the number of elements in it).

Your algorithm should run in $O(n \log n)$ time to receive full credit, where $n = |V|$ is the number of vertices.

**Hint:** What would your algorithm look like if $w_r = 1$ for all $r \in V$?

4. Predecessor/Successor in BST

Let $T$ be a BST. Show how to compute, given an element $x$, the smallest element in $T$ which is no smaller than $x$. Your algorithm should run in $O(h)$, where $h$ is the height of the tree (so, for example, $O(\log n)$ if the tree is a balanced BST).

5. Intervals

Design a data structure that maintains a set of closed intervals $S$ and supports the following three operations (in all operations $l \leq r$).
a) **INSERT([l, r]):** Insert an interval [l, r] into the set S.

b) **REMOVE([l, r]):** Remove an interval [l, r] from the set S.

c) **QUERY([l, r]):** For a given interval [l, r], decide whether there exists an interval in S that intersects with [l, r]. We say two intervals *intersect* with each other if and only if they share a common point.

Your data structure should support all the three operations in $O(\log n)$ time to receive full credit, where $n = |S|$ is the size of S.

**Hint:** There exists a *simple* solution. Full credit will not be given for interval trees.

(20) **6. Another Bound for Splays**

During the lecture we have seen the static optimality theorem of splays. At the core of the proof is the following version of access lemma. We restate it here for convenience.

**Lemma 1 (Strong Access Lemma)** For a splay T, suppose we assign a fixed weight $w(x)$ to all nodes $x$ in T. Let

$$s(x) = \sum_{y \in \text{subtree}(x)} w(y)$$

and

$$r(x) = \log s(x).$$

For the potential function

$$\Phi(T) = \sum_{x \in T} r(x),$$

the amortized cost of splay($x$) with root $t$ is upper bounded by

$$3(r(t) - r(x)) + 1.$$ 

In this problem we explore another running time bound of splays.

For simplicity let’s assume elements in the splay are $\{1, 2, \ldots, n\}$ and there is a special element $f \in \{1, 2, \ldots, n\}$. Suppose we sequentially search for $q_1, q_2, \ldots, q_m \in \{1, 2, \ldots, n\}$. Prove that the total running time is upper bounded by

$$O(m + n \log n + \sum_{i=1}^{m} \log(|q_i - f| + 1)).$$

(Intuitively, this bound states that if we mostly search for items very close in value to $f$, our amortized search time will be much better than $O(\log n)$.)

**Hint:** Set

$$w(x) = \frac{1}{(|f - x| + 1)^2}$$

and prove

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = O(1).$$