COMPSCI 530: Design and Analysis of Algorithms

Qualifier Examination

Fall 2013

This booklet has 12 pages (including the cover page).

INSTRUCTIONS

1. No external help (books, notes, laptops, tablets, phones, etc.) or collaboration is allowed.
2. Answers should be brief and to the point.
3. When asked for an algorithm, give a short description in English. Unless asked for, do not give a pseudocode, a proof of correctness, or running time analysis.

DO NOT WRITE BELOW THIS LINE

<table>
<thead>
<tr>
<th>Problem</th>
<th>Recurrence</th>
<th>/ 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>Recurrence</td>
<td>/ 10</td>
</tr>
<tr>
<td>Problem 2</td>
<td>Sorting via Suffix Flips</td>
<td>/ 10</td>
</tr>
<tr>
<td>Problem 3</td>
<td>Range Queries</td>
<td>(a) / 5, (b) / 15</td>
</tr>
<tr>
<td>Problem 4</td>
<td>Max coloring on a path</td>
<td>(a) / 10, (b) / 15, (c) / 5</td>
</tr>
<tr>
<td>Problem 5</td>
<td>Modified Bellman-Ford</td>
<td>(a) / 5, (b) / 5, (c) / 15, (d) / 5</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>/ 100</td>
</tr>
</tbody>
</table>
Problem 1 (Recurrence). [10 points] Solve the following recurrence:

\[ T(n) = \sqrt{n} \cdot T(\sqrt{n}) + O(n). \]
Problem 2 (Sorting via Suffix Flips). [10 points] Suppose we need to sort an array $A$ of $n$ distinct integers in increasing order. The only two operations we are allowed to use are:

- $\text{FINDMIN}(j)$: This returns the index of the smallest element among $A[j], A[j+1], \ldots, A[n]$.
- $\text{FLIP}(j)$: This flips the elements in the suffix $A[j], A[j+1], \ldots, A[n]$, i.e. the new $A[j]$ is the previous $A[n]$, the new $A[j+1]$ is the previous $A[n-1]$, etc.

Sort the elements of $A$ using $O(n)$ $\text{FINDMIN}$ and $\text{FLIP}$ operations.
Problem 3 (Range Queries). Suppose that you have an array $A$ of $n$ numbers. In this question, we will design data structures for range queries. The data structures are constructed upfront (before any query arrives) and the construction can take arbitrary time but must meet given space restrictions. Once the data structures have been constructed, queries arrive one at a time and have to be answered using these data structures within specified time limits.

(a) [5 points] First, suppose that the queries are of the form $SUM[i, j]$, where $1 \leq i \leq j \leq n$. The algorithm must return the value of $\sum_{i \leq k \leq j} A[k]$. Describe a data structure using $O(n)$ space that can answer $SUM[i, j]$ queries in $O(1)$ time. (Hint: Prefix/suffix computation)
(b) [15 points] Now, let us consider queries of the form \(\text{MAX}[i,j]\), where \(1 \leq i \leq j \leq n\). The algorithm must now return the value of \(\max_{i \leq k \leq j} A[k]\). Describe a data structure using \(O(n \log n)\) space that can answer \(\text{MAX}[i,j]\) queries in \(O(1)\) time. (Hint: Using \(O(n)\) space, divide the problem into two subproblems of equal size.)
Problem 4 (Max-coloring on a Path). Suppose we have a path of $n$ vertices $1, 2, \ldots, n$, where vertex $i$ has a positive weight $w_i$. We are given an unlimited number of colors and have to color the vertices such that no two adjacent vertices get the same color. The weight of a color $c$, denoted $w[c]$, is the maximum weight among the vertices in that color class (i.e. vertices that are colored using $c$). The goal is to minimize the overall weight of the coloring, i.e. the sum of weights of the colors used.

(a) [10 points] Prove that any optimal coloring uses at most 3 colors. (Hint: Proof by contradiction. Given any solution using more than 3 colors, show that you can recolor the vertices in the minimum weight color class so that the overall weight of the coloring decreases.)
(b) [15 points] We will use colors red, green, and blue where \( w[\text{red}] \geq w[\text{green}] \geq w[\text{blue}] \). Clearly, \( w[\text{red}] \) must be the maximum weight among all vertices. So, our goal is to minimize \( w[\text{green}] + w[\text{blue}] \). Given fixed values \( x \) and \( y \), give an \( O(n) \)-time algorithm that either produces a coloring where \( w[\text{green}] \leq x \) and \( w[\text{blue}] \leq y \) or declares that no such coloring exists.
(c) [5 points] Use the algorithm in part (b) to obtain a minimum weight coloring in $O(n^3)$ time.
Problem 5 (Modified Bellman-Ford). Given an undirected graph $G = (V, E)$ where edge $(u, v) \in E$ has a non-negative length $\ell[u, v]$, the Single Source Shortest Path (SSSP) problem asks for the length of the shortest path from a given source vertex $s$ to all other vertices in $V$. A classical algorithm for this problem is Bellman-Ford’s algorithm:

for all $v \in V$ do
dist[v] $\leftarrow \infty$
end for

dist[s] $\leftarrow 0$

for all $i \leftarrow 1$ to $|V| - 1$ do
for all $(u, v) \in E$ do

dist[v] $\leftarrow \min(dist[u] + \ell[u, v], dist[v])$
end for
end for

return dist

(a) [5 points] Suppose that, in addition to the distances, we want to find actual shortest paths. To succinctly represent such shortest paths, let us introduce a variable $prev[v]$ for each vertex $v \in V$ such that the following inductive procedure gives us a shortest path from $s$ to $v$: find a shortest path from $s$ to $prev[v]$ inductively, and attach the edge $(prev[v], v)$ to it. How would you update $prev[v]$ inside the loop?
(b) **5 points** Use part (b) to show that there exist a set of shortest paths from $s$ to all vertices in $V$ whose edges form a tree. (Hint: Use the $(prev[v], v)$ edges and order vertices in terms of $dist[v]$ to show that these edges cannot form a cycle.)
Suppose that for each vertex $v \in V$, instead of finding the length of the shortest path from $s$, we now want to find the length of the shortest path that is not a multiple of a given integer $k$. For example, if $k = 2$, for a vertex that has paths of length 4, 7, and 8, we have to output 7, whereas for a vertex that has paths of length 5 and 8, we have to output 5. Modify Bellman-Ford’s algorithm to obtain an algorithm for this problem. (Hint: Define $dist_k[v]$ for vertex $v$ as the length of the shortest path from $s$ to $v$ such that $(dist_k[v] − dist[v]) \mod k ≠ 0$.)
(d) [5 points] The correctness of the original Bellman-Ford algorithm is based on the following property: At the end of iteration $i$, $dist[v]$ stores the length of the shortest path from $s$ to $v$ that uses at most $i$ edges (and $\infty$ if no such path exists). State the corresponding property for the algorithm you designed in part (c).