Homework 5 Solutions

1. 15-1 (p. 364)

Bitonic Euclidean Traveling Salesman Problem

1. Sort the cities using an O(nlogn) sort from the smallest to largest x-coordinate.
2. Suppose city v_n is adjacent to city v_i. Then we need the shortest path from v_i to v_1 and then to v_n, and then back to v_i that includes every other city as well.
3. We define t(i,j), i < j, to be the shortest path starting from v_i, reaching v_1, and going back to v_j. t(i,j) is computed as follows:

\[ t(i, j) = t(i, j-1) + d(v_{j-1}, v_j) \] if \( j \geq i + 2 \)

\[ t(i, j) = \min\{ t(k, i) + d(v_k, v_j) \mid \text{if } j = i+1 \text{ for } 1 \leq k \leq i - 1 \} \]

\( d(a, b) \) is the distance between cities a and b.

Note: Because the Traveling Salesman Problem is NP-complete, this yields only an approximate solution. This algorithm is not guaranteed to find the best solution.

2. 9-2 (p. 194)

Weighted Median

a.) Argue that the median of x_1, \ldots, x_n is the weighted median of the x_i with weights w_i = 1/n.

Since all x_i have the same weight, the median of x_1, \ldots, x_n, the median will be x_{ceiling(n/2)}.

If n is even, then n = 2k. Summing the first k x_i-values followed by the second k values yields the same weighted sum for each set:

\[ k/n = k/2k = 1/2 \]

The weighted lower median is thus x_k = x_{n/2}.

If n is odd, then n = 2k + 1. Summing the first k x_i-values followed by the second set of k + 1 x_i-values yields the following two equations:

First k x_i:\quad k/n = k/(2k+1) < 1/2
Second k+1 x_i:\quad (k+1)/n = (k+1)/(2k+1) > (k+1)/(2k+2) = 1/2
The weighted lower median is thus thus $x_k = x_{n/2} = x_{\text{ceiling}(n/2)}$ since $n$ is odd.

b.) Show how to compute the weighted median of $n$ elements in $O(n\log n)$ worst-case time using sorting.

1. Sort the $x_i$ such that $w_1 \leq \ldots \leq w_n$.
2. $\text{sum} \leftarrow 0$
3. for $j \leftarrow 1$ to $n$
   a. $\text{sum} \leftarrow \text{sum} + w_j$.
   i. if $\text{sum} \geq \frac{1}{2}$
      1. return $x_{j-1}$

Complexity: Sorting can be accomplish in $O(n\log n)$ time using your favorite $O(n\log n)$ sorting algorithm. Initializing the sum variable can be done in $O(1)$ time. In the worst case, the for loop iterates $n$ times. All of the operations inside of the for loop are $O(1)$. Therefore, the total time complexity is:

$$T(n) = O(n\log n) + O(1) + O(n)O(1) = O(n\log n)$$

d.) Argue that the weighted median is a best solution for the one-dimensional post office location problem, in which points are simply real numbers and the distance between points $a$ and $b$ is $d(a, b) = |a - b|$.

Note: In the original weighted mean problem, element $x_i$ had one attribute: weight $w_i$. In the one-dimensional weighted median problem, post office (element) $p_i$ has two attributes: weight $w_i$, and one-dimensional position coordinate $p_i$. Weighted median is a property that depends only on weights. Our goal is to find a value for $p$ that minimizes:

$$\sum_{i=1}^{n} w_i d(p, p_i) = \sum_{i=1}^{n} w_i | p_i - p |$$

This value indicates the optimal coordinate at which to place the new post office $p$.

We are asked to show that the $p$ that minimizes the objective function is the weighted median. The weighted median one of the $p_i$, which we can see from the definition of weighted median.

Suppose the weighted median of the post offices is $p_k$. Assume to the contrary that the value for $p$ that minimizes the weighted distance is not $p_k$.

We must consider two cases:
\[ p = p_k + \delta \]
\[ p = p_k - \delta \]

Let \( D \) be the weighted distance when \( p = p_k \), \( D_1 \) be the weighted distance when \( p = p_k + \delta \), and \( D_2 \) be the weighted distance when \( p = p_k - \delta \).

\[
D = \sum_{i=1}^{n} w_i |p_i - p_k| = w_1(p_1 - p_k) + w_2(p_2 - p_k) + \ldots + w_n(p_n - p_k)
\]

\[
D_1 = \sum_{i=1}^{n} w_i |p_i - (p_k + \delta)| = w_1(p_1 - p_k - \delta) + w_2(p_2 - p_k - \delta) + \ldots + w_n(p_n - p_k - \delta)
\]

\[
D_2 = \sum_{i=1}^{n} w_i |p_i - (p_k - \delta)| = w_1(p_1 - p_k + \delta) + w_2(p_2 - p_k + \delta) + \ldots + w_n(p_n - p_k + \delta)
\]

\[
D_1 - D = \delta \left( \sum_{i=1}^{k} w_i \right) - \delta \left( \sum_{j=k+1}^{n} w_i \right) = \delta(w_1 + \ldots + w_i) - (w_{i+1} + \ldots + w_n)
\]

\[
D_2 - D = \delta \left( \sum_{j=k+1}^{n} w_j \right) - \delta \left( \sum_{i=1}^{k} w_j \right) = \delta(w_{i+1} + \ldots + w_n) - \delta(w_1 + \ldots + w_j)
\]

Since \( p_k \) is weighted median,

\[
\left( \sum_{i=1}^{k} w_i \right) = (w_1 + \ldots + w_i) \geq \frac{1}{2}
\]

\[
\left( \sum_{j=k+1}^{n} w_j \right) = (w_{i+1} + \ldots + w_n) \leq \frac{1}{2}
\]

Therefore:

\[
D_1 - D \geq 0 \quad \text{and} \quad D_2 - D \leq 0
\]

\( D \) is a better solution than either \( D_1 \) or \( D_2 \). Therefore \( \delta = 0 \) is the best solution. In both cases, \( p = p_k \).

3. 16.1-3 (pg. 379)

Suppose that we have a set of activities to schedule among a large number of lecture halls. We wish to schedule all activities using a few lecture halls as possible. Give an efficient greedy algorithm to determine which activity should use which hall.
Note: Although the problem didn’t explain this, both the scheduling problem and the graph coloring problem are NP-Complete. Therefore, no perfect solution to either problem can be guaranteed in polynomial time.

Our algorithm works as follows:
1. Sort the activities in terms of length, such that the longest activity is at the front of the list.
2. Select the longest activity from the front of the list. Place it in a lecture hall.
3. Process each activity in the list, starting with the second-longest activity.
4. If it is possible to schedule this activity in one of the already-allocated lecture halls, schedule it there. Otherwise, open a new lecture hall and schedule the activity in the new hall.

In the worst case, each activity must be placed in its own hall. Therefore, the kth largest activity will have to determine that it cannot fit into any of the k-1 lecture halls that have already been appropriated. Therefore, the worst-case complexity is \(O(n^2)\) if a total of n activities must be scheduled.

16.2-5 (pg. 384)

Describe an efficient algorithm that, given a set \(x_1, \ldots, x_n\) of points on the real number line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

1. Presort the list of points such that \(x_1 \leq \ldots \leq x_n\).
2. While the list is not empty
   a. Select the leftmost point in the list, \(x_l\)
   b. Take the interval \([x_l, x_l + 1]\).
   c. Remove all points \(x_l, x_{l+1}, \ldots, x_{l+k}\) such that \(x_{l+k} - x_l \leq 1\) from the list.

Correctness proof: Assume to the contrary, that when considering the leftmost point \(x_l\), we selected an interval \([x_l - \alpha, x_l + \beta]\), such that \(\alpha \geq 0, \beta \geq 0, \text{ and } \beta - \alpha = 1\). Since \(x_l\) is the leftmost point in the current set of points, we know that we will not cover any points in the interval \([x_l - \alpha, x_l]\). Therefore, we will maximize the total number of possible points covered if and only if we select the interval \([x_l - \alpha, x_l + \beta]\), such that \(\alpha = 0\) and \(\beta = 1\).

16.3-2 (pg. 392)

What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

\[
a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21
\]
Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?
In general, for the nth Fibonacci number, we have:

<table>
<thead>
<tr>
<th>Fib(n)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib(n-1)</td>
<td>01</td>
</tr>
<tr>
<td>Fib(n-2)</td>
<td>001</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>Fib(2)</td>
<td>(0^{n-2}1)</td>
</tr>
<tr>
<td>Fib(1)</td>
<td>(0^{n-1})</td>
</tr>
</tbody>
</table>