Practice Problems

Henry Z. Lo

June 19, 2014

These practice problems should get you ready for the exam. You will need to know:

- Dynamic programming, both bottom-up and top-down.
- Greedy algorithms.
- Divide and conquer.
- Hashing and sets.

Solutions for the problems are discussed, and also given in code (ProblemSolutions.java). Also note F.java, which has some convenience functions, and ProblemSolutionsTest.java, which contains test cases for the solution functions. The test will be similar to this format.

Problem 1. *Count the number of inversions (out of order pairs) in an array. Let the elements be $x_1, \ldots, x_n$. There is an inversion if $x_i > x_j$ and $i < j$.*

We can divide and conquer like in merge sort. The idea:

1. Divide array into `left` and `right`.
2. Continue step 1 until we get to one element subarrays. Return 0 inversions when we do.
3. Calculate inversions between the two by combining `left` and `right`.
   (a) Let the current index in `left` be `j` and the index in `right` be `k`.
   (b) If `right[k] < left[j]`, everything between the two is out of order.
4. Inversions = invs in `left` + invs in `right` + invs between `left` and `right`.
5. Return inversions.

This procedure runs in $O(n \log n)$ time. There are $\log n$ split / merge steps, processing n elements each.

Problem 2. *Suppose you are investing. You want to buy high, then sell low. You have an array x of integers representing future prices, and can make one buy and one sell. What is the most you can make?*

This is another divide and conquer problem.

1. Divide array into `left` and `right`.
2. Return 0 when we get to arrays of size 1.
3. When we merge, the maximum profit is the maximum of the profit in `left`, the profit in `right`, and the maximum profit between the two.
4. The max profit between is the difference of the maximum in `right` and the minimum in `left`.
5. Return maximum profit.

Runtime is the same as the previous problem, $O(n \log n)$.

Problem 3. *It has been hypothesized that couples which are closer in social desirability are more likely to have a stable relationship. Suppose we have n heterosexual men, and n heterosexual females. Each male has a social desirability score m_i, and each female has a score f_i. Pair up males and females so that the total difference in scores is minimized, and return this minimal score.*

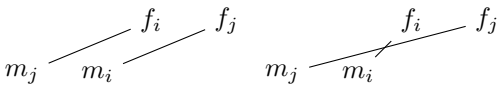
One greedy approach we can use is to pair the closest male and female until done. However, this does not always work. Consider this group:

females	6	9	13
males	3	7	8

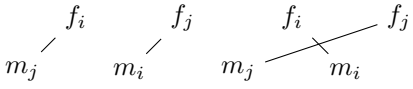
This greedy strategy would pair f_1 and m_2, then f_2 and m_3, then f_3 and m_1. This yields a total difference of 12, when the optimal is 10.

Another method is to sort all males and females by social desirability. Then just assign the same ranked individuals with each other.

We have to justify this approach by showing that having equally ranked pairs decreases the total difference. Consider the following situation for two males and two females. The elements to the right have higher scores than elements to the left:



Regardless of how m_i, m_j, f_i, f_j are paired up, the total distance is the same. However, in the other configuration, only pairing the sorted elements reduces the total distance, *even if* f_i and m_i are closer, because it avoids "crossing over".



The argument is made much more convincing geometrically. Once we are convinced of this result, we can write the greedy algorithm very simply. This algorithm is $O(n \log n + n) = O(n \log n)$.

Problem 4. *Suppose you have a rod of length n, which you can cut into sections. You can cut up sections of length $i = 1, \ldots, n$ inches. Each different length fetches a different price. For example:*

length i	*1*	*2*	*3*	*4*	*5*	*6*
price p_i	*2*	*3*	*5*	*6*	*7*	*10*

How can we cut up a rod to maximize profit?

There are two ways to do this problem. One is to work from n down to 0, and one is to work from 0 up to n. Both involve trying many different slice combinations to maximize profit.

Using a hash map m, the top-down (memoization) solution is given here:

1. Start at length n.
2. We can slice it to lengths $0, 1, \ldots, n-1$, each yielding a different amount of profit.
3. Recursively slice until we slice down to length 0. At this point, return 0.
4. For each rod length, calculate the best price you can get for it by either:
   - Calculating it recursively, considering each slice you can make.
   - Considering where each slice will take you; if the slice takes you to a length already calculated, just look it up in m, and add the value of the slice.
5. Store the best price for this rod length in m.
6. Return the best price for n.

As with any memoization solution, this is simply trying every possibility while memorizing subcomputations. Thus, the code is not far from a brute force solution, but performance is much better.

Using a hash map m, the top-down (memoization) solution is given here:

1. Start at length n.
2. We can slice it to lengths $0, 1, \ldots, n-1$, each yielding a different amount of profit.
3. Recursively slice until we slice down to length 0. At this point, return 0.
4. For each rod length, calculate the best price you can get for it by either:
   - Calculating it recursively, considering each slice you can make.

- Considering where each slice will take you; if the slice takes you to a length already calculated, just look it up in m, and add the value of the slice.

5. Store the best price for this rod length in m.

6. Return the best price for n.

As with any memoization solution, this is simply trying every possibility while memorizing subcomputations. Thus, the code is not far from a brute force solution, but performance is much better.

We can also use dynamic programming:

1. Start at length 0.

2. We can arrive here from some higher length `newLength`.

3. Add that profit to the profit of where you are now. Call this `value`.

   - If `newLength` does not exist in m, place it in m with the value `value`.
   - Otherwise, if `value` is higher than the current value for `newLength`, replace it in m.

4. Return the value for n in m.

Problem 5. *You have three sets of elements, s_1, s_2, s_3. Find all elements in exactly one set.*

We can do this using hash tables.

1. Construct hash sets for s_1, s_2 and s_3.

2. Iterate over s_1. Remove every element in s_1 from s_2 and s_3.

3. Repeat step 2 for s_2 and s_3.

4. Return the union of s_1, s_2, s_3.