CodeU Coding Exercises - Session 4

Algorithmic Choices

There are a number of recurring themes in computer science algorithms. You have been working with many of them, but perhaps not thinking about them as variations on common themes. Some key ones that are useful to keep in mind are the following:

1. **Adding a level of indirection simplifies many tasks.** There is practically an entire level of indirection that is hidden by Java; if you look at C++, the difficulties of managing memory for int or char arrays goes away when you use std::vector or std::string objects. Iterators is another good example. In fact, many of the core “Design Patterns” achieve their power by adding a level of indirection.
2. **Trading off space for time.** Hash tables is the prime example of this. The table has to be larger than the set of keys it holds, but all the alternative key lookup schemes are much slower. There are many algorithms where creating and maintaining the right data structure (e.g., a stack, queue, or tree) will speed up your algorithm.
3. **Avoiding unnecessary computation.** This includes divide-and-conquer techniques and avoiding computing the same result multiple times. This is the basis for many dynamic programming algorithms, where intermediate results are kept in a table and reused instead of recomputing them.

There are other basic themes, of course; these three will take you a long way.

Please Help - Share your Feedback:

Session 4 Feedback Form

Lightning Round (DO AT LEAST THREE)

This week is similar to last week. The exercises are similar to coding questions you might get in an interview. You should work on each exercise in phases (**ALERT:** the phases are different from last week):

Phase 1 - see how far you can get in 20-25 minutes just thinking and writing - no compiling, no reference material lookups. If you complete the question, that's great - you can move on to the next. But try to at least come up with an outline (pseudocode, flowchart, etc.) of a solution.

Phase 2 - if you do not finish it in 25 minutes, note how far you are from completing it. If you can complete it with extra time, do it. If you need help, get help and then complete it.

Phase 3 (optional but highly recommended) - Take another 20 minutes and think about how you can improve your solution. Think about both time and space complexity. Can you save time by keeping extra state information about the algorithm? If you use recursion or a stack/queue, can you do it without using the extra data structure or stack space? Is there a common subproblem that you can do once and reuse the results? Sometimes looking at a problem bottom-up instead of top-down, or thinking about the problem "backwards" will give you ideas. Write down your ideas.

Phase 4 (optional) - after you have your completed solution that is ready to submit, then you may try to compile it and actually get it working.

In an interview, once the interviewer is satisfied that you understand the problem and can write a good solution, they will usually ask you about the performance of your solution. Then they may ask if you can do better. This should be the start of a conversation; you'll offer a tentative solution, and the interviewer will provide hints to steer you in the right direction (or away from the wrong direction). Or, the interviewer might supply some suggestions and then wait to see if you catch on and can start filling in the details. While working on these exercises you will have to fill both roles as you brainstorm about the problem. After you've spent time with the problem, go ahead and discuss the problem with fellow CodeU'ers, a professor, the web, or your CodeU mentor.

NOTE: most of these problems are widely-known, and you can easily find solutions and analysis (even some correct ones!) on the web. Don't! At least, not before trying it yourself. Also, you may have already done some of these. If you cannot find at least 3 of these that you have not done before, then contact your mentor. There are many more where these came from.

Exercise 1 - Palindrome #1 - bits

A palindrome is a sequence that is the same if it is reversed. The string "abba" is a palindrome. The string "aaba" is not. A sequence of digits, characters, numbers, or whatever can be palindromic. Natural language palindromes typically ignore spaces and punctuation (e.g., "Madam, I'm Adam") is considered a palindrome.

Write a method with integer argument K that finds and returns the K'th positive integer whose binary representation is a palindrome. The first such integers are 1 ('1'), 3 ('11'), 5 ('101'), 7 ('111'), etc.

There are actually two ways to do this problem; (1A) ignore leading zeros, or (1B) use as many leading zeros as you need. Using definition 1B, two additional integers, 2 ('010'), and 6 ('0110') are added to the list.

Exercise 2 - Palindrome #2 - string

Given a string, Find the longest palindromic subsequence of the string. Return the starting index and length of the first substring that qualifies.

For example, given the string "12111122221212121", the algorithm should return a starting index of 9 and a length of 7 (for "1211112222**1212121**").

Exercise 3 - Ugly numbers

Ugly numbers are defined as the positive integers whose prime factors are limited to 2, 3, or 5. By convention, 1 is also part of the sequence. Here are the first 10 Ugly numbers: 1, 2, 3, 4, 5, 6, 8, 9, 10, and 12. Write a method to print the K'th Ugly number.

Exercise 4 - Returning change

Given an integer N, and an unlimited supply of coins of M different denominations { D[0], D[1], .. D[M-1] }, compute the number of different ways you can make change for N "cents".

For example, if we name this method makeChange, makeChange(10, {2, 3, 5}) should return 4, since there are 4 ways that combinations of 2, 3, and 5 add up to 10 ({5,5}, {2,3,5}, {2,2,2,2,2}, and {2,2,3,3}). makeChange(1, {2, 3, 5}) should return 0.

Exercise 5 - Find dictionary words #1 - Split a string into words

Given an input string (with no spaces) and a dictionary of words, implement a method that breaks up the input string into a space-separated string of dictionary words that a search engine might use for "Did you mean?" For example, an input of "applepie" should yield an output of "apple pie", assuming that "apple" and "pie" are in the dictionary.

The most basic version of this problem assumes that:

1. We are only concerned with breaking up an input string into 2 dictionary words.
2. The dictionary fits in memory.
3. All feasible outputs are equally valid, e.g., for an input of "aman", we are indifferent between "a man" and "am an" as outputs.
4. We require that the entire input string be consumed, and do not allow any edits.

A useful (and harder) variation is to assume the input is an arbitrary number of words, and find all the candidate splits that produce 2 or more valid words. This is typically called segmentation.

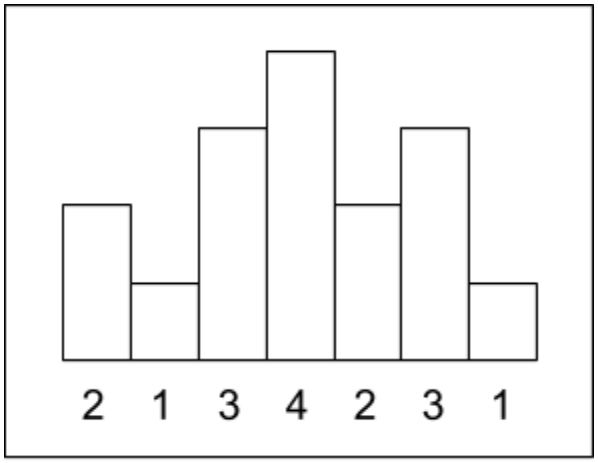
Exercise 6 - Find dictionary words #2 - Play Boggle™

Given a 4X4 grid of letters and a dictionary, find all the words from the dictionary that can be formed by the letters in the grid.

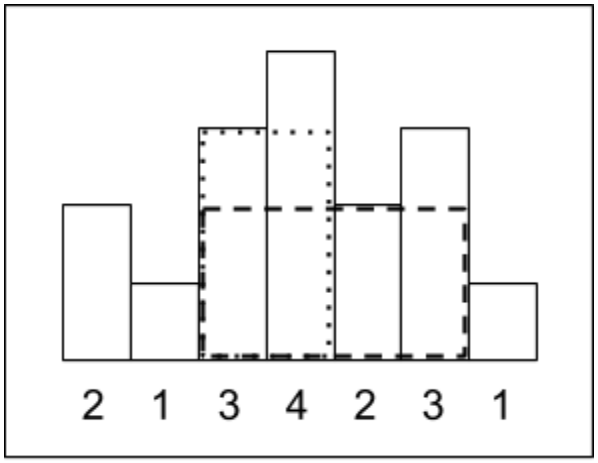
The rules for forming a word are: start at any position on the board, move in any of the up to 8 directions to choose another letter, repeat. You cannot re-use a letter in a given position in the same word.

Exercise 7 - Maximum rectangle under a histogram

Given an array of non-negative integers that represent the height of bars in a histogram, and treating each bar as having width 1, find the rectangle with the largest area that “fits” within the area covered by the histogram.
For example, the sequence {2, 1, 3, 4, 2, 3, 1} represents the histogram:



The diagram below shows the same histogram with two of the candidate rectangles shown as dotted and dashed lines. The 4x2 rectangle has the largest area:



Keep in mind that each histogram bar is also a rectangle.

Exercise 8 - Subdivide a sequence of numbers

People are asked to dress in blue and green for a party. People who come in blue colour are assigned “1” and people who come in green are assigned “2”. So the sequence might be something like 121111222212121212….

Divide this sequence into K groups; the size of each group must be 3 or larger. Each member of the original sequence must be in exactly one group. Also, the ordering of the sequence must be intact (each group must be constructed from a consecutive sequence of sequence members). Write a program that splits the sequence into the required K groups, so as to minimize the sum of the squares of the differences in the numbers of 1's and 2's in each group.

Huh? OK, here's an example: subdivide("1211112222121212", 4) might yield groups of "121", "111222", "2121", and "212", squares of differences = 1, 0, 0, 1, and a sum of 2.

Please Help - Share your Feedback:

Session 4 Feedback Form