CPSC 320 2018W1: Assignment 4

November 7, 2018

1 Nuts and bolts

1. In the quiz, you investigated two different algorithms for the Nuts and Bolts problem: algorithm `Nuts-and-Bolts` is simple but inefficient, running in $O(n^2)$ time in both the average case and the worst case. Algorithm `NB-Improved` is much better, but assumes irrealistically that the nuts are given to you sorted by size.

   After thinking about the version of the problem where nuts come in a bag for a while, you realize that you might be able to accomplish the task more efficiently by using the nut and bolt you matched as a way to filter the rest.

```
Algorithm NB-Quick(Nut-Set, Bolt-Set)
  If Nut-Set is empty, then
    Return the empty set
  Else If Nut-Set contains exactly one nut, say N, then
    Let B be the single bolt in Bolt-Set
    Return {(N, B)}
  Else
    Remove a nut, say N, from Nut-Set
    Partner-found = False
    Tried-Bolts = ∅
    While not Partner-found
      Remove any bolt, say B, from Bolt-Set
      If bolt B threads into nut N then
        Partner-found = True
      Else
        Add B to Tried-Bolts
    For each nut in Nut-Set
      If the nut is too lose for B
        Add it to the set Loose-Nuts
      Else add it to the set Tight-Nuts
    For each bolt in Bolt-Set ∪ Tried-Bolts
      If the bolt is too large for N
        Add it to the set Large-Bolts
      Else add it to the set Small-Bolts
    Return {(N, B)}∪ NB-Quick(Loose-Nuts, Large-Bolts)
                   ∪ NB-Quick(Tight-Nuts, Small-Bolts)
```

   Consider the case where, at every recursive call, both of the sets Tight-Nuts and Loose-Nuts have size in the range $[n/k, (k-1)n/k]$, for some integer $k > 2$. Write a recurrence relation that gives a good asymptotic (big-O) *upper-bound* on the running time of this algorithm, as a function of both n and k.

$$T'(n) \leq \begin{cases} c, & \text{when } n = 0 \text{ or } n = 1 \quad // \text{ base cases} \\ \boxed{} & //\text{recursive case} \end{cases}$$

2. Which of the following relationships holds between the function T from question 1, and the function T' from question 3 of the nuts and bolts question from quiz 4?

   ◯ $T' \in o(T)$ ◯ $T' \in \Theta(T)$ ◯ $T' \in \omega(T)$ ◯ None of these hold

2 Array-chopping

An arithmetic array is one whose elements form an arithmetic sequence, in order—i.e., they're arrays of the form $A = [a_1, a_1+c, a_1+2c, \ldots, a_1+(n-1)c]$, where A has length n (for $n \geq 2$). You're given an arithmetic array with k elements missing from somewhere in the middle (i.e., it's not the first or last element that's been removed). For example, the missing numbers in $[3, 6, 12, 18, 24, 27]$ are 9, 15 and 21. The missing numbers in $[1, 22, 29, 36]$ are 8 and 15.

1. Give an example that shows that if you are not told what k is, then you can not determine c, no matter how many elements the array you are given contains.

2. Suppose now that you are not told k, but you are given an upper bound K on its value. You will be able to determine c and k as long as the array you receive contains at least $f(K)$ elements. What is $f(K)$?

   $f(K) =$ [ ]

3. Describe an algorithm to compute the values of c and k, assuming you know K and are given an array with at least $f(K)$ elements.

4. Finally suppose that you are told what k is. Describe an efficient algorithm that takes as input the array and the value of k, and returns an array containing all missing elements.

5. What is the worst-case running time of your algorithm, as a function of k and the length n of the array?

3 More longest common subsequences

An instance of the 3-sequence longest common subsequence problem (3LLCS) consists of three sequences $x = x_1x2 \ldots x_n$, $y = y_1y_2 \ldots y_n$ and $z = z_1z_2 \ldots z_n$; for simplicity we assume that all are of the same length. The problem is to find the length of the longest sequence that is a subsequence of all three sequences. We denote this length by 3LLCS(x, y, z).

1. What is 3LLCS(brute, force, searches)?

2. Give a recurrence that expresses 3LLCS(x, y, z) in terms of 3LLCS of smaller strings.

3. Design a polynomial-time, iterative, dynamic programming algorithm for solving this problem. (You may well want to first design a memoized recursive algorithm for the problem, but you do not need to submit a description of this algorithm.)

4. State what is the running time of your algorithm of part 3 (you do not need to provide justification).

5. Describe how to adapt your algorithm so that it returns the the actual longest common subsequence of the three input sequences (not just the length).

6. [Bonus] Give an example that proves that the LCS of three sequences x, y and z can be **neither** of LCS(x, LCS(y, z)), LCS(y, LCS(x, z)), LCS(z, LCS(x, y)).

4 DNA free energies

Free energy is a fundamental property of DNA duplexes (double-stranded DNA), and identifying collections of DNA strands with a particular free energy is useful in many biotechnologies. You will tackle a simplified version of this problem here.

- Let $s = s_1 s_2 \ldots s_n$ be a string over the alphabet $\{A, C, G, T\}$. (The string represents a DNA sequence, which can form a duplex with its Watson-Crick complement, but these details need not concern you.) Throughout assume that $n \geq 2$.
- Let F be a table $F[x : y]$ of 16 nonnegative parameters, where $x, y \in \{A, C, G, T\}$.
- Let $\Delta G(s)$ be the sum
$$F[s_1 : s_2] + F[s_2 : s_3] + \ldots + F[s_{n-1} : s_n].$$
For example, if some of the table entries are as follows;
$$F[A : A] = 2, \quad F[A : C] = 3, \quad F[C : T] = 7, \quad F[T : A] = 4, \quad F[C : G] = 12,$$
then $\Delta G(AACTACG) = 31$, $\Delta G(CT) = 7$, and $\Delta G(AAA) = 4$.
- For any integer g, let $\#\Delta(n, g)$ be the total number of length-n strings over alphabet $\{A, C, G, T\}$ that have $\Delta G = g$.
- Let $\#\Delta(n, g, X)$ be the total number of length-n strings s that start with the letter X and have $\Delta G(s) = g$.

1. Which of the following recurrences is correct for $\#\Delta(n, g, X)$? You do not need to explain your answer here, just circle which option you believe to be correct. Assume that for all of the cases, the base conditions are

$$\#\Delta(n, g, X) = \begin{cases} 0, & \text{if } n \geq 2 \text{ and } g < 0, \\ \text{the number of table entries } F[X : Y] \text{ that have value } = g, & \text{if } n = 2 \text{ and } g \geq 0. \end{cases}$$

   ○ $\#\Delta(n, g, X) = \sum_{Y \in \{A,C,G,T\}} \#\Delta(n-1, g - F[X : Y], Y), \;\; n > 2, g \geq 0$

   ○ $\#\Delta(n, g, X) = \sum_{Y \in \{A,C,G,T\}} (F[X : Y] + \#\Delta(n-1, g - F[X : Y], Y)), \;\; n > 2, g \geq 0$

   ○ $\#\Delta(n, g, X) = \sum_{Y \in \{A,C,G,T\}} \#\Delta(n-2, g - F[X : Y], Y), \;\; n > 2, g \geq 0$

   ○ $\#\Delta(n, g, X) = \sum_{Y \in \{A,C,G,T\}} (F[X : Y] + \#\Delta(n-2, g - F[X : Y], Y)), \;\; n > 2, g \geq 0$

2. Give an expression for $\#\Delta(n, g)$ in terms of $\#\Delta(n, g, X), X \in \{A, C, G, T\}$.
3. Design a memoized recursive algorithm that computes $\Delta(n, g, X)$ when $n \geq 2$ and $X \in \{A, C, G, T\}$. Your algorithm should have running time $O(ng)$.
4. Explain why the running time of your algorithm is $O(ng)$.