TCSS 343 - Week 6 - Monday

Jake McKenzie

February 27, 2019

Greedy Algorithsm and Dynamic Programming

"A distributed system is when some computer I didn't know existed fails and causes my computer to fail."
...
Leslie Lamport

"Greedy algorithms: do the thing that looks best right now, and repeat until nothing looks good anymore or you're forced to stop."
...
Jeremy Kun

"We don't much care if you don't approve of the software we write."
...
Eric Hughes

I was vacationing with my family in Europe,
in a country only rich people know about.
Svenborgia?
No, better.
But I can't tell you.

You want to invite as many people to your party as possible. But, there's a catch: you don't quite trust the other diplomats, all of whom speak multiple languages. So, you'd like to make sure that you can understand what everyone is saying at your party. As the Canadian ambassador to Svenborgia, you speak English, French, and Svenborgian. You want to make sure that no two of your party guests speak the same language, other than the three you speak. For each diplomat, you have a list of of every "foreign" language (i.e., other than English, French, or Svenborgian) that they speak. We refer to this as the International Party Guest, or IPG, problem.

For example: suppose there are three diplomats. If diplomat 1 speaks language 1, diplomat 2 speaks languages 2 and 3, and diplomat 3 speaks languages 1, 3, and 4, we would represent this instance as $\{\{1\}, \{2, 3\}, \{1, 3, 4\}\}$. The optimal solution to this instance is to invite diplomat 1 and diplomat 2.

Consider the four following greedy algorithms designed to maximize the number of guests you can invite. For each, determine whether the algorithm is optimal. If it is, briefly sketch a proof of optimality. If it's not, give a counterexample(next page).

0. **Greedy Strategy A:** if no two diplomats speak the same foreign language, invite them all. Otherwise, invite the diplomat who speaks the fewest languages only if they don't share a language with the next diplomat in the invite pool, and recurse.

1. **Greedy Strategy B:** if no two diplomats speak the same foreign language, invite them all. Otherwise, remove the diplomat who speaks more than half the possible languages, and recurse.

2. **Greedy Strategy C:** if no two diplomats speak the same foreign language, invite them all. Otherwise, remove the diplomat who speaks the most languages, and recurse.

3. **Greedy Strategy D:** if no two diplomats speak the same foreign language, invite them all. Otherwise, invite the diplomat who speaks the fewest languages, remove all other diplomats who share a language with the one you just invited, and recurse.

4. Prove that if $n \in \mathbb{Z}$ and $5n + 4$ is even, then n is even using proof by contradiction. (**Motivation:** Proof by contradiction is often used in proving greedy algorithms correct.)

5. In mathematics, a sequence of positive real numbers s_1, s_2,... is called *superincreasing* if each element in the sequence is greater than the sum of all previous elements in the sequence:

$$s_{n+1} > \sum_{i=1}^{n} s_i$$

For example: $\{2, 3, 7, 16, 65, 321, 4546\}$ is a superincreasing sequence, but $\{1, 1, 2, 5, 15, 52, 203, 877\}$ us not a superincreasing sequence.

Describe an algorithm that takes as input superincreasing sequence $s_1, \ldots, s_n$ and a positive integer k, please find a sequence of $s_1, \ldots, s_n$ with the sum equal to k. It is possible and desirable to find an algorithm that can accomplish this task in $O(n)$ time using dynamic programming. If you think you've come up with an algorithm that can accomplish this task attempt to prove that it is correct.

6. Look back at the cashier's algorithm you've seen in lecture, which I will denote as MC for "Make Change". Compactly we can write the algorithm as where:

$$MC(N) = \min_i \{MC(N - S_i) + 1\}$$

where $i \in \{1, 2, \ldots, m\}$ and $S_1 < S_2 < \cdots < S_m$
How many unique subproblems are there for this problem?

7. How much work do you have to do to go from your subproblems back to your original problem?

8. Using the prior two parts find the worst case runtime of $MC(N)$.

9. Challenge: Is this a polynomial time algorithm? (HINT: Look back at the subproblems)

10.