CPSC 320: Intermediate Algorithm Design and Analysis

Final Exam Review Worksheet Solution

You're a high-powered diplomat working for the Canadian Foreign Service in the wealthy European nation of Svenborgia. The holidays are coming up, and you want to throw a lavish party for the other foreign diplomats in the country – for hobnobbing, networking, and general holiday cheer.

Let the festivities begin!

1 A Certifiably Optimal Guest List

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.

1. Consider the two 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.

   Greedy Strategy 1: if no two diplomats speak the same foreign language, invite them all. Otherwise, remove the diplomat who speaks the most languages, and recurse.

   SOLUTION: not optimal. Suppose our instance is $\{\{1, 5, 6, 7\}, \{2, 3\}, \{1, 3, 4\}\}$. This is the same as the instance above, but with extra, irrelevant languages (irrelevant because they don't cause conflict with any other diplomats) added for diplomat 1. So the optimal solution is still to invite 1 and 2, but the greedy algorithm will discard 1, then discard 3, and leave you inviting only diplomat 2.

   Greedy Strategy 2: 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.

   SOLUTION: not optimal. Suppose our instance is $\{\{1, 5, 6, 7\}, \{2, 3, 8, 9\}, \{1, 3, 4\}\}$. Again, we've taken the instance above but added more irrelevant languages. The optimal solution is still to invite 1 and 2, but the greedy algorithm will choose 3, and then have to discard 1 and 2, leaving you with just diplomat 3 as the solution.

2. Consider now the *decision variant* of the IPG problem, namely: given a set of diplomats and list of foreign languages each one speaks, can I find at least k of them to invite to my party such that no two speak the same foreign language?

   Prove that IPG is in NP.

   SOLUTION: our certificate will be the set of diplomats we choose to invite. To verify it, we need to check that it has size at least k (takes constant or linear time), and we need to check that no two diplomats in the certificate speak any of the same languages (takes quadratic time, as we need to check every pair of diplomats). Therefore, the certificate is verifiable in polynomial time, and IPG is in NP.

3. RECALL the **Independent Set** (IS) problem: given a graph $G = (V, E)$, is there a set of at least k vertices in V such that no two are adjacent (i.e., no two share an edge)? Prove that IPG is NP-hard by completing the following reduction from IS to IPG:

```
Given: an instance {G=(V,E), k_IS} of Independent Set


Each v_i in V becomes a _________diplomat_____________________ in IPG


Each edge e_j in E becomes a _______language_____________________ in IPG


Let L be a list of the sets of languages spoken by the diplomats. L contains

___|V|____ entries, all initialized to {}.


For each edge e_k = (v_i, v_j) in E do:


                Append k to L[i] and L[j]









Define k_IPG = ___________k_IS_____________________________________


Solve IPG(L, k_IPG). Return YES to IS iff the answer to IPG is YES.
```

2 Sub-Par Catering

It's the night of your holiday party! Despite some NP-completeness-related difficulties in creating a guest list, you came up with a good one and are all set for a great party.

But there's a problem! Among other food items, you ordered a party platter of sub sandwiches. But the sub shop messed up your order, and is instead sending you a single k-foot long giant party sub. These foreign diplomats have high standards for their food, and won't want to eat a single huge sandwich that they have to share with other people. So, you're going to have to divide up the sandwich in some way.

Suppose that $n = 3k$ is the length, in reasonably sized 4-inch portions, of the party sub. You have a measure to quantify how happy someone would be to receive an i-portion length of party sub (defined for each $i \leq n$) all for themselves – you've defined this in terms of "happiness points," which you can give to your diplomat guests in exchange for goodwill and various political favours later on. Not every guest needs to receive a piece of the sandwich (you ordered other food), and a guest can be given more than one piece.

For example, if the sandwich is eight portions long, and the different lengths have the following happiness value, then the maximum obtainable happiness value is 19 (by cutting into two pieces of length 3 and 5):

length	1	2	3	4	5	6	7	8
happiness value	1	2	8	9	11	12	11	10

If the happiness values were listed as below, then the maximum obtainable happiness value would be 16 (by cutting into eight pieces of length 1):

length	1	2	3	4	5	6	7	8
happiness value	2	3	4	5	6	7	8	9

1. Complete the following recurrence to compute the maximum total happiness value in dividing a sandwich of length n, given the happiness values in an array H:

$$\text{MaxHappiness}(n) = \begin{cases} \boxed{0} & \text{when } n = 0 \\ \max\limits_{i=1,\ldots,n} \boxed{H[i] + \text{MaxHappiness}(n-i)} & \text{when } n \geq 1 \end{cases}$$

2. Give a dynamic programming (iterative memoized) algorithm to compute the maximum total happiness value in dividing a sandwich of length n, given the happiness values in an array H:

```
MaxHappiness(n, H):

     Define a length-(n+1) array MaxHappiness (assume 0-based indexing)

     MaxHappiness[0] = 0
     for i = 1 to n
          best_val = -infinity
          for j = 1 to i
               current_val = H[j] + MaxHappiness[i-j]
               if current_val > best_val
                   best_val = current_val
          MaxHappiness[i] = best_val

     return MaxHappiness[n]
```

3. Give a good asymptotic bound on the time and space complexity of your dynamic programming algorithm for question 2.

   SOLUTION: Space complexity: the table is size n, so memory use is $\Theta(n)$.

   Time complexity: there are n table entries, and filling in each requires looking at all table entries before it, which gives a time complexity of $\Theta(n^2)$.