CPSC 320 2017W1: Assignment 6

This assignment is to give you practice solving problems before the final exam. It will not be open for submission on GradeScope, and will not be graded.

Solutions will be released on the morning of **Thursday, December 7**. But, we recommend you try to solve every problem before you look at the solutions!

1 The Dividing Tree

Your friend proposes a divide-and-conquer approach to compute a minimum spanning tree of a weighted, undirected, connected graph $G = (V, E)$. The helper function

```
{G_1, G_2} = subgraphs(G)
```

takes a graph $G = (V, E)$ and partitions the vertices into two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that G_1 and G_2 are both connected and $|V_1|$ and $|V_2|$ differ by at most 1. If (and only if) such a partitioning is **not possible**, one or both of the graphs G_1 and G_2 will be disconnected.

```
DC_MST(G = (V, E)):

\\ Base cases:
if |E| = 0:                        \\ no edges
    return NONE
if |E| = 1:                        \\ 1 edge
    return E

{G_1, G_2} = subgraphs(G)
MST_1 = DC_MST(G_1)
MST_2 = DC_MST(G_2)
let e = the minimum-weight edge connecting G_1 and G_2
return [MST_1, e, MST_2]
```

We'll start by analyzing the runtime of this algorithm. Assume that the `subgraphs` function runs in $\Theta(n + m)$ time, where $n = |V|$ and $m = |E|$.

1. We'll find it helpful to examine the performance of this algorithm in the best and worst cases. For this part, we'll focus on connected graphs where `subgraphs` can indeed always generate connected G_1 and G_2 subgraphs.

   (a) The best case for this algorithm is a particular (simple and common) type of connected graph. What type of connected graph yields the best-case runtime? VERY briefly justify your answer.

   (b) What type of connected graph yields the worst-case runtime? VERY briefly justify your answer.

2. Again, assume here we have a connected graph where `subgraphs` can always generate connected G_1 and G_2 subgraphs.

(a) Give and briefly justify a good asymptotic **lower bound** on the **best-case** runtime of this algorithm in terms of n.

(b) Give and briefly justify a good asymptotic **upper bound** on the **worst-case** runtime of this algorithm in terms of n.

3. Does this algorithm always generate a minimum spanning tree? Justify your answer.

4. Now suppose that, when it isn't possible to partition G into two connected subgraphs of equal size, the `subgraphs` function instead returns two **connected** graphs G_1 and G_2, where the absolute difference between $|V_1|$ and $|V_2|$ is minimized.

   (a) Give and justify a good asymptotic **lower bound** on the **best-case** runtime of this algorithm in terms of n. Assume that, as before, the `subgraphs` function runs in $\Theta(n+m)$ time.

   (b) Give and justify a good asymptotic **upper bound** on the **worst-case** runtime of this algorithm in terms of n. In this case, your bound does not need to be tight; but please give something reasonably informative and well-justified. (For example, do not give a trivial upper bound like $O(n^n)$.)

   (c) Does this algorithm always generate a minimum spanning tree? Justify your answer.

2 The $3n$ Musketeers

To maximize their success on the latest assignment, the CPSC 320 class has decided to work in teams! They want to work on their assignment in groups of two. Additionally, they want to make sure that each pair has all the skills necessary to succeed on the assignment.

The class has $2n$ students, of whom n are good at understanding massive blocks of possibly ambiguous text (in the form of problem statements and proposed algorithms), and the other n are good at writing pseudocode. They want each of the n teams to have one Text Decipherer and one Pseudocode Writer.

The problem is that not everyone in the class likes each other, and some of them don't want to be partners. Each Decipherer is willing to work with some subset of the Writers, and each Writer is willing to work with some subset of the Decipherers. Your goal is to form a set of n teams so that everybody is paired with someone they're willing to work with. Assume that everyone is willing to work with at least one person from the opposite side.

We refer to this problem as the **TEAM2 Problem**. We frame this as a decision problem: we want to return YES if it's possible to generate a matching where all partners are willing to work with each other, and NO otherwise.

1. Give a small example of the TEAM2 problem in which it is not possible to generate a matching in which everyone is paired with someone they're willing to work with (i.e., provide a NO instance).

2. The **Maximum Bipartite Matching Problem** takes as input an undirected, unweighted bipartite graph $G = (V \cup W, E)$ and returns the *maximum matching* between V and E. Recall that a matching in a graph is a set of edges so that no two edges share an edgepoint, and the maximum matching is the largest such matching.

   Give a reduction from TEAM2 to Maximum Bipartite Matching, and briefly justify the correctness of your reduction.

Oh, no! Steve and Cinda, the evil CPSC 320 instructors, have demanded you do *formal proofs* as part of your assignment![1] Fortunately, the brave and valiant 320 students are fighting back by enlisting n Theorem Provers, who are (obviously) good at writing proofs.

[1] The preceding joke about Steve and Cinda being evil was brought to you by Susanne. The moral of the story is: never trust the TA's to draft your practice assignment...

Unsurprisingly, each Prover is only willing to work with some of the Decipherers and some of the Writers, and vice versa. We want to divide the class into n disjoint teams of three – each consisting of a Decipherer, Writer, and Prover – such that everyone is willing to work with both other people on their team. Specifically, we want the decision variant of this problem: so we return YES if such a matching is possible, and NO otherwise. We call this the **TEAM3 Problem**.

3. Here is a proposed reduction from TEAM3 to TEAM2 (where D, W, and P denote Decipherers, Writers, and Provers, respectively):

   ```
   return YES to TEAM3(D, W, P) iff:
   TEAM2(D, W) returns YES and
   TEAM2(W, P) returns YES and
   TEAM2(D, P) returns YES.
   ```

   Give a small counterexample (with n no greater than 2) to prove that this reduction is not correct.

Suppose now that, instead of having lists of people each person is willing to work with, you have a 3-dimensional array `happy`, which has value TRUE at `happy[i,j,k]` if and only if Decipherer i, Writer j, and Prover k are all willing to be on a team together. There are no logical constraints on the values in the `happy` array, other than that each value is TRUE or FALSE. For example: if `happy[1,1,k]`, `happy[1,j,1]`, and `happy[i,1,1]` are all TRUE for some values of i, j, and k, it could still be the case that `happy[1,1,1]` is FALSE. Now our goal is to determine whether there's a set of n disjoint teams (i, j, k) such that `happy[i,j,k]` is TRUE for every team.

4. The **3D Matching Problem** is defined as follows: you're given three disjoint sets X, Y, and Z, and $|X| = |Y| = |Z|$. Also given is a list T consisting of triples (x, y, z) for $x \in X, y \in Y$, and $z \in Z$. The 3D Matching Problem asks: given sets X, Y, Z and list of triples T, can we find a subset of T such that every element in X, Y and Z appears **exactly once** in one of the triples? The 3D Matching Problem is known to be NP-complete.

   Give a reduction to prove that TEAM3 is NP-hard. You do not need to prove the correctness of your reduction, but you should clearly explain the key elements of your reduction and why they are there.

5. (a) Assume that $P \neq NP$. Then, is TEAM2 also NP-hard? Prove your answer.

   (b) Your groupmate claims to have generated a polynomial-time reduction from TEAM3 to TEAM2, using a polynomial number of calls to TEAM2. Again assuming that $P \neq NP$, Explain why the new reduction cannot possibly be correct.

3 The Hungry Games

Imagine you're a poor, starving graduate student. You're going to a buffet with some money you earned by TA'ing CPSC 320. The buffet has different food items, and it charges you by the weight of each item you take. To minimize cost, all the foods are in the form of appetizing and nutritious slurries, from which you may dispense any fractional amount in grams you want into compostable paper cups. Oh, the joy.

Your goal is to select the amount (by weight) to take of each food item so that you maximize the amount of calories you consume, without spending more money than you brought with you to the restaurant. For each item, you've calculated the number of calories you get per dollar spent.

1. Design a greedy algorithm to optimally solve this problem. Note that the buffet has a limited quantity of food, and you can take no more than m_i grams of food item i.

2. Prove that your greedy algorithm is optimal.

Now, suppose the restaurant has switched from buffet style to à la carte: that is, instead of deciding how much weight of each item you can pick, you have to place an order for each dish off the menu. Assume that the amount of calories per dollar of each dish has remained the same. For example, if the restaurant used to have sweet-and-sour pork slurry for \$2 per 100 grams, and you came with \$11, you could spend \$11 on 550 grams of slurry. But now, they only sell a 400 gram dish of pork for \$8, which means you could spend \$8 on one order of pork, and have \$3 left for other purchases.

You know the cost of each dish on the menu and the number of calories the dish contains. Moreover, the restaurant has a limited quantity of each dish, and has said you can't place more than three orders for any particular dish.

3. Construct a small instance with **no more than two different dishes** proving that your greedy approach is no longer optimal.

4. Give a recursive function to compute the maximum number of calories you can consume, given an amount of money M you have to spend (assume this is an integer), the price p of each dish (also an integer), and the calories c in each dish.

5. Write pseudocode for a dynamic programming (iterative memoized) algorithm for computing the maximum number of consumable calories.

4 War and P

RECALL the **TUG-O-WAR problem**: given n people with weights $W = [w_1, \ldots, w_n]$ (which we assume to be integers), we want to generate two teams that are well-balanced.

Specifically, we want to partition all n people into two teams T_1 and T_2 such that the absolute difference in total weight between team 1 and team 2 is minimized (note: we do not require the same number of people on both teams). TUG-O-WAR is an optimization problem, but we can give a decision version by asking: given n people and their weights, *and a number* k, can we generate two teams whose weight difference is no more than k?

1. Give a good certificate for this problem and prove that TUG-O-WAR is in NP.

2. SUBSET-SUM takes a list of numbers A and a value w, and returns YES if and only if there exists a subset of A with sum w. SUBSET-SUM is NP-complete. Your friend gives the following reduction to show that TUG-O-WAR is NP-hard:

```
Given a TUG-O-WAR instance with list of weights W and value k,
we can reduce TUG-O-WAR to SUBSET-SUM as follows:

    Let m be the sum of all elements in W.

    // Run SUBSET-SUM multiple times in a loop. We define the loop
    // differently, depending on whether m is even or odd.
    If m is even:
        For i=0 to k:
            if SUBSET-SUM(W, m/2 + i) returns YES:
                // Found two teams with weight difference = i
                return YES to TUG-O-WAR
    Else: // m is odd
         For i=0 to k-1:
             if SUBSET-SUM(W, floor(m/2) + i) returns YES:
                 // Found two teams with weight difference = i+1
```

```
                return YES to TUG-O-WAR
    return NO to TUG-O-WAR
```

`SUBSET-SUM is NP-complete. Therefore, TUG-O-WAR is NP-hard.`

Explain why this does not prove that TUG-O-WAR is NP-hard.

3. Duly chastened, your friend tries again with the following proof. Note that the 320 instructor has told your friend he may assume that all elements in the SUBSET-SUM input list A are positive, and that the target value w is such that $\frac{m}{2} \leq w \leq m$, where m is the sum of all elements in A.

   `Given a SUBSET-SUM instance with list A and target w, we can reduce to TUG-O-WAR as follows:`

   > `Let m be the sum of all elements in A.`
   > `If a subset of A sums to w, the remaining elements in A sum to m-w; i.e.,`
   > `they form two ‘‘teams’’ with absolute weight difference w-(m-w)=2w-m.`
   > `Therefore, we call TUG-O-WAR(A, 2w-m) and return YES to SUBSET-SUM iff YES to TUG-O-WAR.`

   `SUBSET-SUM is NP-complete. Therefore, TUG-O-WAR is NP-hard.`

   Explain to your friend (by giving a counterexample) why his reduction is incorrect.

4. Prove that TUG-O-WAR is NP-hard with a reduction from SUBSET-SUM to TUG-O-WAR. You do not need to prove the correctness of your reduction, but you should clearly explain the key elements of your reduction and why they are there. As before, assume that all elements in the SUBSET-SUM input list A are positive and that $\frac{m}{2} \leq w \leq m$.