CPSC 320 2018W2: Assignment 5 (Last assignment!)

Please follow the guidelines given in Assignment 1 for submission to Gradescope, and for group collaboration. Remember to provide short justifications for your answers. Submit by the deadline **Monday April 1, 2019 at 10PM**.

Use LaTeXto prepare your answers. Easiest will be to use the .tex file provided. Please enclose each paragraph of your solution in `\soln{Your solution here...}`.

Your solution will then appear in dark blue, making it a lot easier for TAs to find the parts that you wrote.

1 Subset sums mod n

You've now seen examples of dynamic programming problems where the recurrence describes a number (e.g., the length of a longest common subsequence), and one example where the recurrence value is Boolean (true or false, indicating whether or not a there is a subset of a set items whose sum is a given value). Here you'll work with a problem where the recurrence describes a set.

Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of nonnegative integers. We say that a value v, with $0 \leq v \leq n-1$, is *feasible* with respect to S if for some non-empty subset R of S,

$$\sum_{x \in R} x = v \pmod{n}.$$

1. For $1 \leq i \leq n$, let $V(i)$ be the set of feasible values that can be obtained from $\{x_1, x_2, \ldots, x_i\}$. Explain why the following recurrence holds for $i \geq 1$, if we define $V(0) = \emptyset$ (the empty set):

$$V(i) = V(i-1) + \cup_{v \in V(i-1)} \{v + x_i \pmod{n}\} \cup \{x_i \pmod{n}\}.$$

2. Design an algorithm that, given a set S of n nonnegative integers, determines whether 0 is feasible with respect to S. The algorithm outpus "Yes" or "No". Your algorithm should run in $O(n^2)$ time. Your pseudocode can use set options such as $\cup$ as in the recurrence above.

Algorithm Subset-Sum-mod-n($S = \{x_1, x_2, \ldots, x_n\}$)
// Returns "Yes" if for some subset R of S, $\sum_{x \in R} x = 0 \pmod{n}$
Create an array Soln[0..n]
// FILL IN THE REST OF THE CODE HERE

3. Suppose that indeed there is a subset R of S such that $\sum_{x \in R} x = 0 \pmod{n}$. Write an algorithm that finds such a subset R. You can assume that an array Soln[0..n] has already been pre-computed, where Soln[i] stores the set $V(i)$ defined in part 1, and your algorithm can use the array Soln. Your algorithm should run in time $O(n^2)$.

2 NP True or False

Let X and X' be decision problems, where both problems have "Yes" instances and "No" instances. State whether you think each of the following statements must be true, must be false, or is an open question. Justify your answer.

1. **Statement:** If $X \leq_p X'$ and X is *not* in NP, then X' is not in NP.

2. **Statement:** If $X \leq_p X'$, X is in P and X' is in NP, then X' must be in P.

3. Let $\overline{X}$ be the *complement* of problem X. That is, instance I of X is a No-instance of $\overline{X}$ if and only if I is a Yes-instance of X. For example, the problem $\overline{\text{SAT}}$ is the set of *unsatisfiable* 3-Sat Boolean formulas—formulas for which there is no satisfying truth assignment. Define $\overline{X'}$ similarly in terms of X'. The statement then is:

   Statement: If $X \leq_p X'$ then $\overline{X} \leq_p \overline{X}'$.

3 Common superstrings

A *common superstring* of the strings $s_1, s_2, \ldots, s_n$ is a string T such that each s_i is a substring of $T, 1 \leq i \leq n$. (Note that the letters of the substring must appear consecutively in T, unlike our definition of a subsequence.) For example, if the strings are $s_1 = 101, s_2 = 0111$, and $s_3 = 010$ then the strings 010111 and 0111010 are both common superstrings of s_1, s_2, and s_3.

An instance of the *Shortest Common Superstring* problem is a set of strings $\{s_1, s_2, \ldots, s_n\}$ and a number K. The problem is to determine whether there is a common superstring of $s_1, s_2, \ldots, s_n$ that has length at most k. Assume that none of the strings s_i is a substring of another string s_j (otherwise s_i can be removed from the list of strings without changing the length of the shortest common superstring). This problem has been important in the context of developing algorithms for genome sequence assembly.

An instance of the *Traveling Salesperson Problem* (TSP) is a set $\{C_1, C_2, \ldots, C_n\}$ of cities, nonnegative integer costs c_{ij} of traveling from city C_i to city C_j (or from C_j to C_i; the cost is the same either way), and a number K. The problem is to determine whether there is a tour of the cities of cost at most K. A *tour* is a path that starts at some city, say C_1, travels through every other city exactly once, and returns to the starting city C_1. The tour cannot go through a city more than once (except for C_1 which appears both at the start and at the end of the tour).

1. Describe a polynomial time reduction from the Shortest Common Superstring problem to the traveling salesperson problem (TSP).

2. Is the Shortest Common Superstring problem in NP? Justify your answer.

3. Does the reduction of part 1, plus your answer to part 2, imply that the Shortest Common Superstring problem is NP-complete? Why or why not?

4 Maximum Cuts (from Coulter Beeson)

In this problem you'll define and show that the MaxCut problem is NP-complete, using a reduction from a known NP-Complete problem, Partition. We'll first define Partition, which is already familiar to you, and then move on to MaxCut.

Partition is another name for the decision problem we saw in class, where thieves try to evenly share their loot. An instance of this problem is a set $I = \{v_1, v_2 ... v_n\}$ of values for n items. I is a Yes-instance if the items can be fairly split, that is, if we can partition the set $[1..n]$ into two nonempty subsets S and $\overline{S}$ such that $\sum_{i \in S} v_i = \sum_{j \in \overline{S}} v_j$.

A **cut** in an undirected graph $G = ([1..n], E)$ divides the set of nodes $[1..n]$ into two non-empty disjoint subsets. That is, a cut is a pair $(S, \overline{S})$ where $\overline{S} = V - S$ and $0 < |S| < n$. If weight $w(i, j)$ is associated with each edge (i, j) of the graph, then the weight of the cut $(S, \overline{S})$ is the sum of the weights of edges that cross the cut. That is,

$$w(S, \overline{S}) = \sum_{i \in S, j \in \overline{S}} w(i, j).$$

We would like to show that there is there is unlikely to be an efficient algorithm that takes as input a graph G and weight function w, and finds a cut of maximum weight. However NP-Completeness is only defined for decision problems.

1. Write down a decision version of the problem, which we'll call MaxCut. What is an instance I of MaxCut? Which are the Yes-instances of MaxCut?

2. Show that MaxCut $\in$ NP.

3. Here you'll reason about a useful building block for a reduction from Partition to MaxCut. Let $I = \{v_1, v_2, \ldots, v_n\}$ be an instance of Partition, with $\sum_{i=1}^{n} v_i = 2V$. From I we'll build a graph $G = ([1..n], E)$ where E has an undirected edge between every pair of nodes (i, j), with weight $w(i, j) = v_i v_j$. Show that if $(S, \overline{S})$ is a partition of $[1..n]$ such that $\sum_{i \in S} v_i = V$, then in G, the weight of the cut (S, S') is V^2.

4. Show that if $(S, \overline{S})$ is a partition of $[1..n]$ such that $\sum_{i \in S} v_i \neq V$, then in the graph G of part 3, the weight of the cut $(S, \overline{S})$ is less than V^2. *Hint:* use the fact that if $x + y = 2V$ then xy attains its unique maximum when $x = y = V$.

5. Describe an efficient reduction that maps instances I of Partition to instances I' of MaxCut. You can describe your reduction in words or implement it in pseudocode.

6. Explain why your reduction runs in polynomial time.

7. Show that if I is a Yes-instance of Partition then I' is a Yes-Instance of MaxCut. (Part 3 should be useful here, as well as your definition of part 1.)

8. Show that if I' is a Yes-instance of MaxCut then I is a Yes-Instance of Partition. (Part 4 should be useful here.)

9. In class we saw a dynamic programming algorithm for the Partition problem, with running time $\Theta(nV)$, where $2V = \sum_{i=1}^{n} v_i$. Why does this not imply that P = NP? (Think what is the size of a Partition instance; is the running time of the dynamic programming algorithm polynomial or exponential in the input size?)

10. **Bonus** Prove the hint given in part 4.