CS 170 HW 12

Due **2020-11-23, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer “Yes”, “Yes but anonymously”, or “No”

2 Survivable Network Design

Survivable Network Design is the following problem: We are given two $n \times n$ matrices: a cost matrix d_{ij} and a (symmetric) connectivity requirement matrix r_{ij}. We are also given a budget b. We want to find a undirected graph $G = (\{1, ..., n\}, E)$ such that the total cost of all edges (i.e. $\sum_{(i,j)\in E} d_{ij}$) is at most b and there are exactly r_{ij} edge-disjoint paths between any two distinct vertices i and j, or if no such G exists, output “None”. (A set of paths is edge-disjoint if no edge appears in more than one of them)

Show that Survivable Network Design is NP-Complete. (Hint: Reduce from a NP-Hard problem in Section 8 of the textbook.)

3 (3,3)-SAT

Consider the (3,3)-SAT problem, which is the same as 3-SAT except each literal or its negation appears *at most* 3 times across the entire formula. Notice that (3,3)-SAT is reducible to 3-SAT because every formula that satisfies the (3,3)-SAT constraints satisfies those for 3-SAT. We are interested in the other direction.

Show that (3,3)-SAT is NP-Complete via reduction from 3-SAT. By doing so, we will have eliminated the notion that the ”hardness” of 3-SAT was in the repetition of variables across the formula. (Hint: If variable x_i appears in k clauses, your reduction should replace x_i with k variables $x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(k)}$. How can we enforce that the copies of x_i all have the same value?).

Give a precise description of the reduction and prove its correctness.

4 Randomization for Approximation

This is a solo question.

Oftentimes, extremely simple randomized algorithms can achieve reasonably good approximation factors.

(a) Consider Max 3-SAT (given a set of 3-clauses, find the assignment that satisfies as many of them as possible). Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies k clauses, your algorithm should produce an assignment that satisfies at least $\frac{7}{8} * k$ clauses in expectation. You may assume that every clause contains exactly 3 distinct variables.

(b) Given a Max 3-SAT instance I, let OPT_I denote the maximum fraction of clauses in I satisfied by any assignment. What is the smallest value of OPT_I over all instances I? In other words, what is $\min_I OPT_I$? (Again, you may assume that every clause contains exactly 3 distinct variables)

5 Approximately Coloring a 3-Colorable Graph

Recall that the 3-coloring problem is NP-hard. In this problem, we will devise a polynomial-time algorithm for $O(\sqrt{n})$-coloring a 3-colorable graph.

(a) Let G be a graph of maximum degree Δ. Show that G is $(\Delta + 1)$-colorable.

(b) Suppose G is a 3-colorable graph. Let v be any vertex in G. Show that the graph induced on the neighborhood of v is 2-colorable (the graph induced on the neighborhood of v refers to the subgraph of G obtained from the set V' of vertices adjacent to v and all edges of G with both endpoints in V').

(c) Give a polynomial time algorithm that takes in a 3-colorable n-vertex graph G as input and outputs a valid coloring of its vertices using $O(\sqrt{n})$ colors. Just an algorithm and proof of correctness are needed. You may use the polynomial-time algorithm for 2-coloring as a black box.
Hint: Use the previous two parts. Your algorithm should first color "high-degree" vertices and their neighborhoods using part b, and then color the rest of the graph using part a.

6 Fast Modular Exponentiation

This is a solo question.

Give a polynomial time algorithm for computing $a^{b^c} \bmod p$ given prime p and integers a, b, and c. Just the algorithm description is needed.

(Remember that by "polynomial time", we mean that if a, b, c, p are all at most n bits long, your algorithm should take time polynomial in n.)

7 Wilson's Theorem

Wilson's theorem says that a number N is prime if and only if

$$(N-1)! \equiv -1 \pmod{N}.$$

(a) If p is prime, then we know every number $1 \leq x < p$ is invertible modulo p. Which of these numbers are their own inverse?

(b) By pairing up multiplicative inverses, show that if p is prime then $(p-1)! \equiv -1 \pmod{p}$.

(c) Complete the proof of Wilson's Theorem by showing that if $(N-1)! \equiv -1 \pmod{N}$ then N is prime. (Hint: If $p > 1$ divides x, it doesn't divide $x+1$)

(d) Unlike Fermat's Little theorem, Wilson's theorem is an if-and-only-if condition for primality. Why can't we immediately base an efficient primality test on Wilson's theorem?

8 (Extra Credit) Approximation Hardness of Independent Set

Recall the maximum independent set problem: Given an undirected graph G, we want to find the largest independent set, i.e. largest set of vertices S such that no two vertices in S are adjacent. An algorithm is a α-approximation algorithm for maximum independent set if given a graph G, it outputs an independent set of size at least OPT/α, where OPT is the size of the largest independent set.

Show that if there is a polynomial-time $2^{2^{100}}$-approximation algorithm for maximum independent set, there is a polynomial-time 2-approximation algorithm for maximum independent set.