CSCI 1010 Theory of Computation

HW11

Due: December 7, 2017

Attach a fully filled-in cover sheet to the front of your printed homework. Your name should not appear anywhere; the cover sheet and each individual page of the homework should include your Banner ID only.

While collaboration is encouraged in this class, please remember not to take away notes from any labs or collaboration sessions.

Please monitor Piazza, as we will post clarifications of questions there. Hand in your solutions by 12:55 to the CSCI 1010 bin on the second floor of the CIT. Late homeworks are not accepted.

Problem 1

Jeepers! The newest craze in Yew Nork is logarithmic-space computation. U. K. Liptus’s laurel-shea trees were the standout competitors in the Game of Cones, and they’ve improved logarithmic-space computation by leaves and bounds. In a desperate attempt to compete, Douglas and Connie Ferr have set Will Owens up with some monotone circuits, but he needs some help with the logging.

Recall from class that the language MonotoneCircuitValue is P-complete. In an instance of MonotoneCircuitValue that corresponds to an instance of CircuitValue in the reduction from class, show that numbers can be assigned to gates in the instance of MonotoneCircuitValue so that the reduction can be done in logarithmic space.
Problem 2

Gee whiz! Give a strategy to pebble the following graph using at most three pebbles. The start vertices are $S_1$, $S_2$, and $S_3$. The goal vertices are $S_7$ and $S_8$.

Express your strategy as a list of rules in the following forms in the order in which they should be taken:

- Place $S_n$
- Move $S_n$ to $S_m$
- Delete $S_n$

Keep in mind that moving and placing are different. You can only move pebbles up the graph; you cannot move a pebble to a start vertex. You can, however, delete a pebble and then place it at a start vertex.
Lab Problem 1

Gee willikers! In this problem, we will look at a language that is $\text{NP}$-hard not only to decide, but even to approximate. The SetCover problem is a canonical $\text{NP}$-hard problem in which an instance is a base set $\mathcal{U}$ and a set of sets $S = \{S_1, S_2, \ldots, S_m\}$ containing subsets of $\mathcal{U}$ (the “universe”). A solution is a covering of $\mathcal{U}$ using some of the sets in $S$. The cost of a solution is the number of sets in the covering; this cost function is known as the $\text{MinSetCover}$ function. It turns out that this problem is also hard to approximate! In fact, we have a useful theorem about the hardness of approximating SetCover:

**Theorem:** Let $n$ be the total number of elements in the base set of a SetCover problem. There exists some constant $c > 0$ such that if there exists a polynomial-time $(c \ln n)$-approximation algorithm for SetCover, then $\mathcal{P} = \mathcal{NP}$.

Though the proof of this theorem is beyond the scope of the class, you will see that it has some useful applications. In particular, consider the problem of landscaping arboreta with intermingled paths of grasslike plants, SedgeCover. Given as input a directed graph $G = (V, E)$, nonnegative edge weights $w_{i,j}$ for edges $(i,j) \in E$, a root vertex $r$, and a set of terminals $T \subset V$, the goal is to find a minimum-cost tree such that for each $i \in T$ there exists a directed path from the root $r$ to terminal $i$.

The SedgeCover problem is $\text{NP}$-hard. Using the theorem above, prove that for some constant $c$ there can be no polynomial-time $(c \ln |T|)$-approximation algorithm for the SedgeCover problem, unless $\mathcal{P} = \mathcal{NP}$. 
Lab Problem 2

Zounds! A shocking development has just taken place at the Game of Cones awards ceremony. U. K. Liptus took the Iron Cone for her work with the laurel-shea trees, but her acceptance speech was hijacked by the Illuminatree. They declare their intent to restore the Pinary Tree Company to the Pine family, under the guidance of the newly returned Holly Pine. Out comes Holly Pine, who is... U. K. Liptus?

Has she been working her way up Yew Nork society only to take back her family’s company from the Pines? Was the Discrete Logging Company only a front for the Illuminatree? What does U. K. even stand for?¹ There are still so many questions to resolve, and anyone who wants to make sense of it all will need to be a superb concentrator.

An \textit{n-superconcentrator} is a directed acyclic graph \( G = (V, E) \) with \( n \) input vertices and \( n \) output vertices such that for any \( r \) inputs and \( r \) outputs, with \( 1 \leq r \leq n \), there are \( r \) vertex-disjoint paths in \( G \) connecting these inputs and outputs. Paths are \textit{vertex-disjoint} if they have no vertices in common.

a. For \( S + 1 \leq n \), prove that to pebble any \( S + 1 \) outputs on an \( n \)-superconcentrator from an initial placement of \( S \) pebbles requires that at least \( n - S \) different inputs be pebbled.

b. Use the result of part (a) to show that to pebble an \( n \)-superconcentrator with \( S \) pebbles in time \( T \) requires \( S \) and \( T \) to satisfy the following inequality:

\[
(S + 1)T \geq \frac{n^2}{2}
\]

\textbf{Hint}: Divide time up into consecutive intervals, chosen so that each interval has the same number of outputs pebbled during it. Apply the results of part (a) to obtain a lower bound on the sum of the number of input and output vertices that are pebbled during the interval.

¹“If it stands for nothing, what’ll it fall for?” —Spoken in conversation with the little-known other brother, Aaron Ferr.
Lab Problem 3

Gadzooks! A **probabilistic Turing machine** is a Turing machine that, in addition to the usual tape, has a second tape filled with random bits. This machine may sometimes accept and sometimes reject the same input $x$, depending on the content of the random tape.

The concept of a probabilistic Turing machine gives rise to a whole new set of complexity classes, namely probabilistic polynomial time complexity classes. Some of the main ones are $\text{BPP}$, $\text{RP}$, $\text{coRP}$, and $\text{ZPP}^2$, and they are defined as follows:

- A language $L$ is in $\text{BPP}$ if there exists a probabilistic Turing machine $M$ that runs in polynomial time such that the following two conditions hold:
  
  **Completeness**: For every $x \in L$, $\Pr[M \text{ accepts } x] \geq 2/3$.

  **Soundness**: For every $x \notin L$, $\Pr[M \text{ accepts } x] \leq 1/3$.

  This Turing machine $M$ is called a **Monte Carlo** algorithm for $L$.

- A language $L$ is in $\text{RP}$ if there exists a probabilistic Turing machine $M$ that runs in polynomial time such that the following two conditions hold:

  **Completeness**: For every $x \in L$, $\Pr[M \text{ accepts } x] \geq 2/3$.

  **Perfect soundness**: For every $x \notin L$, $\Pr[M \text{ accepts } x] = 0$.

- A language $L$ is in $\text{coRP}$ if its complement is in $\text{RP}$. Equivalently, a language $L$ is in $\text{coRP}$ if there exists a Turing machine $M$ that runs in polynomial time such that the following two conditions hold:

  **Perfect completeness**: For every $x \in L$, $\Pr[M \text{ accepts } x] = 1$.

  **Soundness**: For every $x \notin L$, $\Pr[M \text{ accepts } x] \leq 1/3$.

\[^2\text{For the curious: BPP stands for bounded-error probabilistic polynomial time, RP stands for randomized polynomial time, and ZPP stands for zero-error probabilistic polynomial time.}\]
A language $L$ is in $\mathbf{ZPP}$ if there exists a probabilistic Turing machine $M$ that, in addition to the usual accepting and rejecting states, has an “unsure” state in which the machine can halt. $M$ will run in polynomial time and at termination time will be in the accept, reject, or unsure state, such that the following two conditions hold:

**Completeness:** For every $x \in L$, $\Pr[M \text{ accepts } x] \geq 2/3$, $\Pr[M \text{ rejects } x] = 0$, and $\Pr[M \text{ is unsure on } x] \leq 1/3$.

**Soundness:** For every $x \notin L$, $\Pr[M \text{ accepts } x] = 0$, $\Pr[M \text{ rejects } x] \geq 2/3$, and $\Pr[M \text{ is unsure on } x] \leq 1/3$.

a. Show that $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{BPP}$ and $\mathbf{P} \subseteq \mathbf{coRP} \subseteq \mathbf{BPP}$.

b. Show that $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$.

c. Using either the given definition for $\mathbf{ZPP}$ or the one you just derived in part (b), demonstrate that the following definition is also equivalent: A language $L$ is in $\mathbf{ZPP}$ if there exists a probabilistic Turing machine $M$ (which does not have “unsure” states) and some polynomial $p$ such that:

**Runtime:** For all inputs $x$ of length $n$, $E[\text{runtime of } M \text{ on } x] \in O(p(n))$.

**Completeness:** For every $x \in L$, $\Pr[M \text{ accepts } x] = 1$.

**Soundness:** For every $x \notin L$, $\Pr[M \text{ accepts } x] = 0$.

**Hint:** You may find Markov’s inequality to be helpful. If $X$ is a nonnegative random variable and $a > 0$, then:

$$\Pr[X \geq a] \leq \frac{E[X]}{a}$$