CSCI 3104 Fall 2021
Instructors: Profs. Grochow and Waggoner

Problem Set 12

Due Date ..... TODO
Name ..... **john blackburn**
Student ID **jobl2177**
Collaborators ..... **List Your Collaborators Here**

Contents

1 Instructions

- The solutions **must be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only. Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.

- Posting to **any** service including, but not limited to Chegg, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.

- My submission is in my own words and reflects my understanding of the material.
- I have not collaborated with any other person.
- I have not posted to external services including, but not limited to Chegg, Discord, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

Agreed (john blackburn). □

3 Standard 26- Computational Complexity: Formulating Decision Problems

Problem 2. Recall the Maximum Bipartite Matching problem from class, where we took as input a bipartite graph $G(L \dot{\cup} R, E)$ and asked for a maximum cardinality matching. Formulate the decision variant of this problem using the Instance/Decision format from class. [**Note:** See Example 172 of M. Levet's Lecture Notes.]

Answer. Instance: Let $G(L \dot{\cup} R, E)$ be a bipartite graph, and let $x \in \mathbb{N}$
Decision: is the maximum cardinality matching of $G(L \dot{\cup} R, E)$ at most x?

□

4 Standard 27- Computational Complexity: Problems in P

Problem 3. Consider the decision variant of the Interval Scheduling problem.

- Instance: Let $\mathcal{I} = \{[s_1, f_1], \ldots, [s_n, f_n]\}$ be our set of intervals, and let $k \in \mathbb{N}$.
- Decision: Does there exist a set $S \subseteq \mathcal{I}$ of at least k pairwise-disjoint intervals?

Show that the decision variant of the Interval Scheduling problem belongs to P. You are welcome and encouraged to cite algorithms we have previously covered in class, including known facts about their runtime. [**Note:** To gauge the level of detail, we expect your solutions to this problem will be 2-4 sentences. We are not asking you to analyze an algorithm in great detail.]

Answer. To determine if the interval scheduling problem is in P we need to show that there is a polynomial time algorithm to solve it. We have solved this problem previously in class using a greedy algorithm that places all the intervals into a priority queue ordered with the earliest end time first and the latest end time last. The greedy algorithm then goes through each interval and builds the set from there. Once we have the set we check that it has atleast k pairwise-disjoint intervals. We determined that this algorithm run-time will be $O(n \log(n))$ with n representing the number of tasks done in the algorithm, therefore since the $n \log(n)$ is in polynomial time we can be sure that the interval scheduling problem is in P. □

5 Standard 28- Computational Complexity: Problems in NP

Problem 4. We consider an algebraic structure $Q = (S, \star)$, where S is our set of elements and $\star : S \times S \to S$ is our "multiplication" operation. That is, for any $i, j \in S$, the product $i \star j \in S$. When S is finite, we can write out its "multiplication table": the rows are indexed by the elements of S, the columns are indexed by the elements of S, and the entry in the (i, j) position in the table is $i \star j$. We say that $(S, \star)$ is a *quasigroup* if the multiplication table is a Latin square; that is, if each element of S appears exactly once in each row and exactly once in each column.

The Latin Square Isotopy problem is defined as follows.

- Decision: Let $Q_1 = (S_1, \star)$ and $Q_2 = (S_2, \diamond)$ be quasigroups, where S_1, S_2 are n-element sets. Suppose that $(S_1, \star)$ and $(S_2, \diamond)$ are given by their multiplication tables.
- Decision: Do there exist one-to-one functions $\alpha, \beta, \gamma : S_1 \to S_2$ such that for all $x, y \in S$:

$$\alpha(x) \diamond \beta(y) = \gamma(x \star y).$$

  Here, $\alpha(x) \diamond \beta(y)$ is a product being considered in Q_2, while $x \star y$ is a product being considered in Q_1.

Show that the Latin Square Isotopy problem belongs to NP. *

Answer. Suppose $Q_1 = (S_1, \star)$ and $Q_2 = (S_2, \diamond)$ are quasigroups, where S_1, S_2 are n-element sets. And Suppose that there are one-to-one functions $\alpha, \beta, \gamma : S_1 \to S_2$ such that for all $x, y \in S$:

$$\alpha(x) \diamond \beta(y) = \gamma(x \star y).$$

The functions α, β, γ serve as our certificate. We provide a polynomial time algorithm to verify that $\alpha, \beta, \gamma : S_1 \to S_2$ such that for all $x, y \in S$:

$$\alpha(x) \diamond \beta(y) = \gamma(x \star y).$$

To do so we check that for each pair of elements $x, y \in S$

$$\alpha(x) \diamond \beta(y) = \gamma(x \star y).$$

There are $n(n-1)/2$ such pairs to check. Thus, the algorithm takes $O(n^2)$ time to verify that $\alpha, \beta, \gamma : S_1 \to S_2$ such that for all $x, y \in S$:

$$\alpha(x) \diamond \beta(y) = \gamma(x \star y).$$

Overall, we can now see that the Latin Square Isotopy problem is in NP because I provided a viable certificate and a polynomial time verification algorithm. □

*It remains open whether the Latin Square Isotopy problem is in P. The Latin Square Isotopy problem, as well as closely related algebraic problems such as Group Isomorphism in the multiplication table model, serve as key barriers for placing Graph Isomorphism into P. The best known algorithm for Graph Isomorphism, due to Babai in 2016, runs in time $n^{\Theta(\log^2(n))}$. Babai's algorithm combines combinatorial techniques such as color-refinement (Weisfeiler–Leman) and algebraic techniques (e.g., permutation group algorithms, the Classification of Finite Simple Groups). This is a very significant result. There has been considerable work on Group Isomorphism in the last 10 years. Isomorphism testing is an active area of research in our CS Theory group!

6 Standard 30- Structure and Consequences of P vs. NP

6.1 Problem 5

Problem 5. A student has a decision problem L which they know is in the class NP. This student wishes to show that L is NP-complete. They attempt to do so by constructing a polynomial time reduction from L to SAT, a known NP-complete problem. That is, the student attempts to show that $L \leq_p \mathsf{SAT}$. Determine if this student's approach is correct and justify your answer.

Answer. No, this students approach is not correct. They have the right idea but not the full idea. In order to show that L is NP-complete they must reduce a known NP-complete problem such as the Hamiltonian Path problem to L successfully and then they must reduce L to a known NP-complete problem. These two reductions taken together prove that L is NP-complete. So, they were wrong but close to the right answer. □

6.2 Problem 6

Problem 6. The underlying model of computation used to define the complexity classes P and NP is the Turing Machine model. A Turing Machine can only read or write a single bit from memory at a given computation step. This will be the only fact about Turing Machines that we need.

Now define the complexity class PSPACE to be the set of languages L, where for a given string $x \in \Sigma^*$, there exists an algorithm (Turing Machine) A and polynomial $p(n)$ (depending on L) such that:

- $A(x)$ uses at most $p(|x|)$ bits,
- $A(x) = 1$ if and only if $x \in L$, and
- $A(x) = 0$ if and only if $x \notin L$.

We will show that NP $\subseteq$ PSPACE. The goal of this problem is to practice with the simulation technique.

(a) Suppose we have an algorithm (Turing Machine) that runs in time $T(n)$. Show that our algorithm uses at most $T(n)$ bits.

Answer. We know that at each step of computation, a Turing machine can at most read or write a single bit. Since each step uses a single bit we know the amount of steps the Turing Machine took to be the total bits used. Thus, if it took T(n) steps to compute a solution we know that our algorithm used at most T(n) bits in total. □

(b) In light of part (a), explain why P $\subseteq$ PSPACE. [**Note:** It is likely that your answer will only require a couple sentences.]

Answer. A problem is in P if there is a known algorithm to solve the problem in polynomial time. A problem is in PSPACE if there is an algorithm that uses at most a polynomial amount of bits relative to the input to solve the problem. We know that problems in each of these classes have known algorithms that use at most a polynomial amount of time to solve their problems. Thus, we can determine that P $\subseteq$ PSPACE. □

(c) The Hamiltonian Cycle problem takes as input a simple, undirected graph G. We ask whether G has a cycle that visits each vertex. It is well-known that the Hamiltonian Cycle problem is NP-complete. Show that the Hamiltonian Cycle problem belongs to PSPACE. [**Hint:** It suffices to show that a brute-force approach can be computed using a polynomial amount of space in the number of vertices. Note that if our graph has n vertices, then each vertex is represented using a binary string of length $\lceil \log_2(n) \rceil$.]

Answer. if we can use a brute force approach to find a solution in polynomial time then we know that the Hamiltonian Cycle problem is in PSPACE. To do that we can search all the vertices in the graph for a path, and assuming we have n vertices, each vertex is of length $\log_2(n)$. Thus to search the entire graph we would result in $O(n \log(n))$ time. Since I found a polynomial time solution we know that the Hamiltonian Cycle Problem is in PSPACE.

□

(d) Here, we show that NP $\subseteq$ PSPACE. Let $L \in$ NP, and let $\omega \in \Sigma^*$ be an input string. We want to decide whether $\omega \in L$. In light of parts (b) and (c), give a PSPACE algorithm to decide whether $\omega \in L$. This establishes that $L \in$ PSPACE, and so NP $\subseteq$ PSPACE. [**Hint:** Start by reducing L to Hamiltonian Cycle in polynomial-time. Is this reduction PSPACE-computable?]

Answer. I will construct a PSPACE algorithm that decides whether $\omega \in L$. First we can reduce L to the Hamiltonian Cycle problem which we already know is in PSPACE from part c. Then we can let ω be a graph. We can then check whether that graph has a Hamiltonian Cycle, if it does we send know that $\omega \in L$ and if it isn't we know that ω is not in L.

□