Homework–1B

Fall 2024: CS 313: Computational Complexity Theory

Due: Sunday, September 29, 2024. Total Marks: 50

This homework can be discussed in groups of two, but must be **attempted individually**.

Question 1 [30 points]

Prove, for any **one** of the following languages, that it is **NP**-Complete, by giving a suitable mapping reduction. Consider only undirected graphs and improper subsets, and research the exact definitions of unknown terms.

- **Dominating Set** $= \{ (G, k) \mid$ Graph G contains a dominating set of size at most k $\}$.
- **Subset Sum** $= \{ (S, t) \mid S$ is a multiset of positive integers, and some subset of S sums to t $\}$.
- **Graph 3-Coloring** $= \{ G \mid$ Graph G is 3-colorable$\}$.
- **0/1 Integer Programming** $= \{ L \mid L$ is a list of inequalities with rational coefficients, satisfiable using an assignment of 0s and 1s to the variables only$\}$.
- **Comparative Divisibility** $= \{ (A, B) \mid A$ and B are strictly increasing sequences of positive integers, and some number c divides more elements of A than $B\}$.
- **Bipartite Subgraph** $= \{ (G, k) \mid$ There is a bipartite spanning subgraph of G with at least k edges$\}$.
- **Monochromatic Triangle** $= \{ G \mid$ The edges of graph G can be partitioned into two disjoint sets, such that neither of the spanning subgraphs formed using the sets contains a triangle $\}$.
- **Set Splitting** $= \{ (S, C) \mid C$ is a collection of subsets of S, such that for some disjoint partition-into-two of S, no element of C is a subset of either partition$\}$.

Solution: Claim 1: Graph 3-Coloring is **NP**-Complete.

To show that the problem is **NP**-Complete, we need to show that it is in **NP**, and that it is **NP**-Hard.

Proof:

To show that the problem is in **NP**, we can construct a verifier V such that V takes a graph $G(V, E)$, and a coloring c. Our verifier ensures that each edge $e \in E$ has differently colored endpoints, and that the graph uses 3 colors at most. Since the number of edges in any graph is bounded by $\mathcal{O}(n^2)$, where n is the number of vertices, the verifier runs in $\mathcal{O}(n^2)$ which is polynomial time.

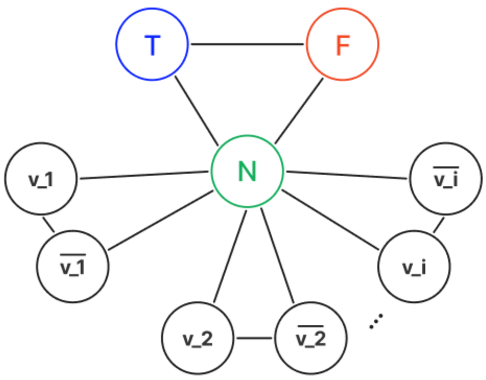
To show that the problem is **NP**-Hard, we can give a polynomial time reduction from **3-SAT** to **Graph 3-Coloring**:

3-SAT $\leq_p$ Graph 3-Coloring

Then we say, given a boolean formula ϕ of 3-SAT, we construct a graph $G = (V, E)$ where G is 3-colorable iff ϕ is satisfiable.

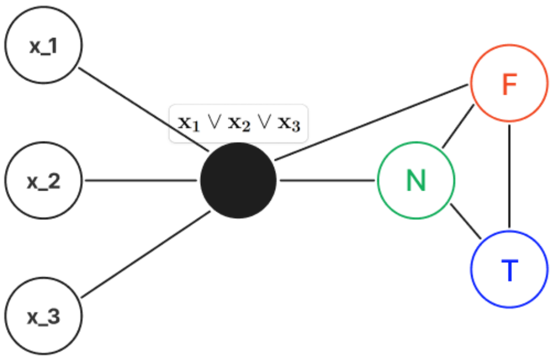
We know ϕ is a boolean formula in 3-CNF form, and is of form $C_1 \wedge C_2 \wedge ... \wedge C_n$ where each C_i is a clause of 3 literals. Then to construct the graph G from ϕ, we first add a triangle in G of three vertices that represent the colors, such as (T, F, N), where T is a true assignment, F is a false assignment, and N is a neutral assignment or base assignment, and each of these can also be thought of as colors. This forms a triangle, and we construct our graph in accordance to this triangle. It is trivial that a triangle would need 3 colors.

Then, for each variable $v_i \in \phi$, we add two vertices $v_i, \bar{v}_i$, add an edge between both of them and add it to the triangle such that they are connected at N. This construction is such since N is our neutral assignment, and obviously v_i and $\bar{v}_i$ cannot have the same color, as they are negations of each other. Hence, we get 3 color assignments. This creates a graph as shown in the figure below.



By this construction, we attach each literal and its negation together, and form a triangle with the neutral color node, thus, either v_i or $\bar{v}_i$ gets the assignment T or F, which we can interpret as a color or as a truth assignment.

In addition, for each clause $C_i = (x_1 \vee x_2 \vee x_3)$, we introduce a *clause gadget* in the form of an ORed output node connected to the three literals x_1, x_2, x_3, and the vertices representing the three colors T, F, N. We attach the output node to the neutral node N and the false node F, as shown in the figure above.



By the above construction, the gadget ensures that if any of the literals is true, the output node is colored true or T, since the OR of the literals will also be true, which ensures a a valid 3-coloring as it is connected to the neutral node N and the false node F. If none of the literals are true, then the output node is colored false or F, as the OR of the literals is false, which violates the 3-coloring constraint, and hence the clause is not satisfiable.

Next we prove that G is 3-colorable iff ϕ is satisfiable.

1. **If ϕ is satisfiable, then G is 3-colorable:**

   If ϕ is satisfiable, then there exists an assignment of truth values to the variables such that each clause is satisfied. Then for any literal x_i, if x_i is assigned true, then we assign the color T to x_i and F to $\bar{x}_i$, which in turn is connected to the neutral / base node, hence is a valid coloring. Further, since every clause C_i is satisfiable, the ORed output node is assigned the color T, and is connected to the neutral node N and the false node F, which is a valid 3-coloring.

2. **If G is 3-colorable, then ϕ is satisfiable:**

   Conversely, suppose G is 3-colorable, then we can construct a truth assignemnt to the literals of ϕ by setting x_i to T if v_i is colored T, and vice versa.

Now if this is not a satisfying assignment to ϕ, then there exists at least one clause $C_i = (x_1 \vee x_2 \vee x_3)$ that was not satisfiable, then all 3 literals were assigned the truth assignment of false, which would mean that the ORed output node was assigned the color F, which would contradict the 3-coloring constraint as per G, hence G is not 3-colorable.

Thus, we have demonstrated that any 3-SAT instance can be transformed into a Graph 3-Coloring instance in polynomial time, and that 3-Coloring is solvable iff the 3-SAT instance is satisfiable. Therefore, since 3-SAT is **NP**-Complete, and we have reduced it to Graph 3-Coloring, we can conclude that Graph 3-Coloring is **NP**-Hard. Coupled with the fact that the Graph 3-Coloring is in **NP**, we have shown that Graph 3-Coloring is **NP**-Complete.

■

Question 2 [20 points]

Define a *coding* κ to be a mapping, $\kappa : \Sigma^* \to \Sigma^*$ (not necessarily one-to-one).

For some string x, $x = \sigma_1 \cdots \sigma_n \in \Sigma^*$, we define $\kappa(x) = \kappa(\sigma_1) \cdots \kappa(\sigma_n)$ and for a language $L \subseteq \Sigma^*$, we define $\kappa(L) = \{\kappa(x) : x \in L\}$.

Show that the class **NP** is closed under *codings*.

Solution: We need to show that for any arbitrary language L, if $L \in$ **NP**, and if κ is a coding defined on the alphabet of L, then $\kappa(L) \in$ **NP**. Since $L \in$ **NP**, there exists a non-deterministic Turing Machine M that verifies L in polynomial time. Then we can construct a deterministic polynomial time verifier V for $\kappa(L)$ as follows:

$V =$ "On input $\langle w, \langle x, c \rangle \rangle$:

1. Compute $\kappa(x)$ from x. If $\kappa(x) \neq w$, reject. Else go to step 2.
2. Simualte M on x with certificate $c, \langle x, c \rangle$. If M accepts, accept. Else, reject."

The above shows that for any arbitrary string $w \in \kappa(L)$, we have $\langle x, c \rangle$ as a certificate of w, where c is the certificate for x in L. Thus, if $w = \kappa(x)$, then we make the string $\langle x, c \rangle$ as a certificate of w if c is the certificate for x. Further, the verifier V for $\kappa(L)$ can verify the string w in polynomial time by leveraging the verifier M for L. It uses the fact that if $w \in \kappa(L)$, then there must be some string $x \in L$ such that $\kappa(x) = w$, and x can be verified by M in polynomial time with the appropriate certificate c.

Hence, $\kappa(L) \in$ **NP**, and **NP**is closed under codings.

■