Answer the questions below.

1. The Hamiltonian path problem asks if there is a path in a graph that visits every vertex exactly once. Give a polynomial time reduction of the Hamiltonian path problem (in an undirected graph) to the CNF SAT problem.

https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-2463548775

Solution:

Any Hamiltonian path in a graph with n vertices can be represented as a one-to-one function $\sigma : \{1, \ldots n\} \to V$, where $\sigma(1)$ is to be interpreted as the first vertex in the path, $\sigma(2)$ the second, and so forth. Letting $V = \{1, \ldots, n\}$, σ becomes a permutation over $\{1, \ldots n\}$.

To produce a CNF formula that is satisfiable if and only if the input graph has Hamiltonian path, we define the boolean variables $X[i][j]$ for $i \in \{1, \ldots n\}, j \in V$.

Intuitively, the idea is that $X[i][j]$ is true exactly when $\sigma(i) = j$. For example, if $n = 3$ and $\sigma(1) = 2,\ \sigma(2) = 3, \sigma(3) = 1$, then the values for $X[i][j]$ should be

$$\begin{bmatrix} F & T & F \\ F & F & T \\ T & F & F \end{bmatrix}$$

Note how there is exactly one T in every row and column. This represents a Hamiltonian path that starts at vertex 2 then goes to vertex 3 then to vertex 1.

More precisely, the first set of clauses makes it possible to define a mapping between satisfying assignments and permutations.

- For all $i \in \{1, \ldots, n\}$, include the clause $(X[i][1] \vee \ldots \vee X[i][n])$. This ensures that for any i there is a j such that $X[i][j]$ is true.
- For all $i \neq i' \in \{1, \ldots n\}$ and $j \in V$, include the clauses $(\neg X[i][j] \vee \neg X[i'][j])$. This ensures that for any $j \in V$, no two variables $X[i][j]$ and $X[i'][j]$ both are true.

Together these imply that for any satisfying assignment there is a unique permutation σ such that $\sigma(i) = j$ if and only if $X[i][j]$ is true under the assignment.

It remains to ensure that a satisfying assignment maps (in the way described above) to a permutation σ that represents a path in the graph. Therefore, for every $(j, j') \notin E$ and $i \in \{1, \ldots n-1\}$, we include the clause $(\neg X[i][j] \vee \neg X[i+1][j'])$. In other, word the path may not go from j to j' if (j, j') is not in the graph.

To prove that this reduction is correct, first suppose that the graph G has a Hamiltonian path σ. Then setting $X[i][j]$ to be true if $\sigma(i) = j$ and false otherwise yields a satisfying assignment. On the other hand, any satisfying assignment produces a well-defined σ such that for all $(\sigma(i), \sigma(i+1)) \in E$ for all $i \in \{1, \dots n-1\}$. Thus, it corresponds to a Hamiltonian path in a graph.

In python, the reduction can be expressed as

```
def reduce_it(G):
    """ Input: n adjacency matrix G represented as
                a list of lists.
        Output: the clauses of the cnf formula output
                in pycosat format.  Each clause is
                be represented as a list of nonzero integers.
                Positive numbers indicate positive literals,
                negatives negative literal.  Thus, the clause
                (x_1 \vee \not x_5 \vee x_4) is represented
                as [1,-5,4].  A list of such lists is returned."""
    n = len(G)

    def X(i, j):
        """
        Return the number of the variable corresponding to
        the ith vertex in the path being j.
        """
        return n * (i-1) + j

    V = range(1,n+1)

    clauses = []

    #Every place in the cycle gets a vertex
    for i in range(1,n+1):
        clauses.append([ X(i,j) for j in V])

    #No vertex can be visited twice
    for i in range(1,n+1):
        for k in range(i+1,n+1):
            for j in V:
                clauses.append([-X(i,j), -X(k,j)])

    #Adjacency Requirements
    for j in V:
        for k in V:
            if j != k and G[j-1][k-1] == 0:
```

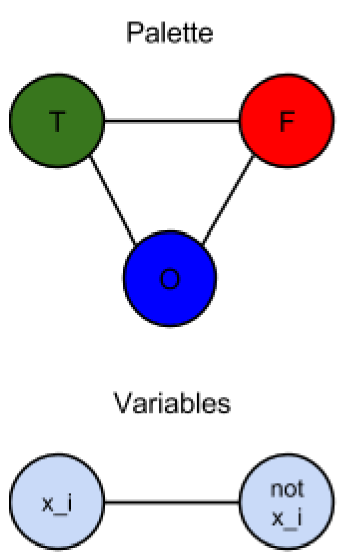
```
        for i in range(1,n):
            clauses.append([-X(i,j), -X(i+1,k)])

    return clauses
```

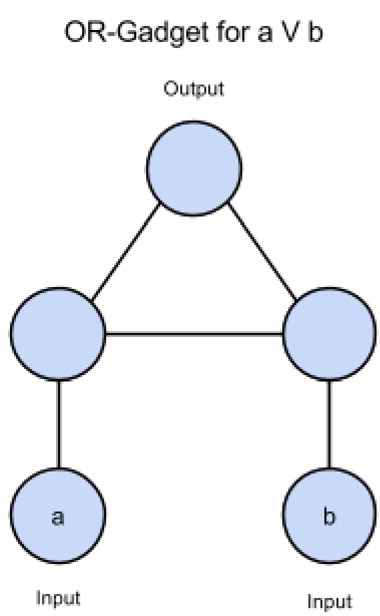
2. Show that if P = NP then there is a polynomial-time computable function f such that if ϕ is satisfiable then $f(\phi)$ is a satisfying assignment. Hint: Use a polynomial-time algorithm for satisfiability repeatedly to find an assignment variable by variable.

Solution:

Let $\phi_{x_1 \to T}$ be the formula ϕ with the variable x_1 replace with the constant true. Use the P=NP algorithm to check whether $\phi_{x_1 \to T}$ is satisfiable. If no then $\phi_{x_1 \to F}$ must be satisfiable. In the first case consider $\phi_{x_1 \to T, x_2 \to T}$ to see if we can set the second variable to true. (If we're in the second case we consider $\phi_{x_1 \to F, x_2 \to T}$ instead). We repeat for all the variables until what remains is a satisfying assignment.

3. A k-coloring of a graph is an assignment of one of k colors to each vertex of the graph such that no edge is incident on vertices of the same color. Show that 3-coloring a graph is NP-complete. (Hint: Reduce 3-CNFSAT. You may find the subgraphs below useful).

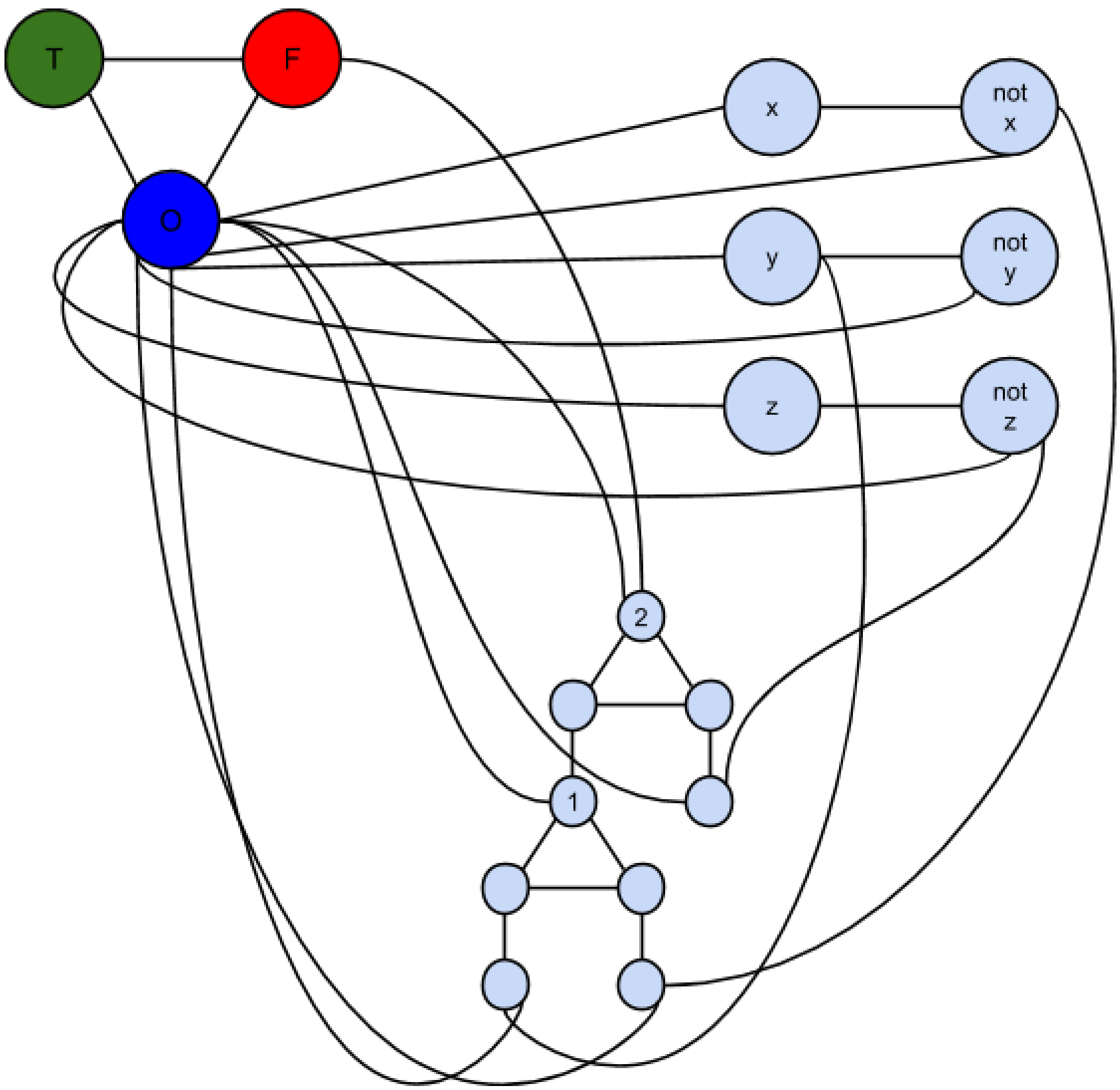




Solution:

First, observe that 3-coloring is in NP. Given a coloring as a certificate, we need only examine each edge to check that the vertices have different colors. This should take only $O(|E|)$ time on a RAM.

Now for the reduction. Suppose we have variables x, y and z and clause $(x \vee \overline{y} \vee z)$



One can check that node 1 can only be colored T if either x is colored T or y is colored F. Node 2 can be colored T only if node 1 is colored T or z is colored T. So node 2 can be colored T only if node x is colored T or node y is colored F or node z is colored T, matching the clause $(x \vee \overline{y} \vee z)$. Node 2 must be colored T for this to be a valid coloring.

Additional clauses and variables are added the same way.

4. Show that whether a graph has a 2-coloring can be determined in polynomial time.

Solution:

Suppose our colors are Red and Green. If we have an edge between u and v and u is colored Red then v must be colored Green.

Here is an algorithmic sketch.

1. Pick any uncolored node u.
2. Color node u red.
3. Color neighbors of u green.
4. Color the neighbors of those nodes red, etc.
5. If ever an edge connects two red nodes or two green nodes then the graph cannot be two colored.
6. If done and there are uncolored nodes (because graph wasn't connected) go to 1.
7. If all nodes are colored we have our two coloring.

5. Given a complete graph G = (V, E) with non-negative integer costs on the edges $c : E \to Z^*$ and an integer bound k, the Traveling Salesperson problem (TSP) is to decide if there is a Hamiltonian cycle in G whose total cost is at most k. Recall that a Hamiltonian cycle visits every vertex once.

   Metric TSP is the constrained case where the edge cost function c satisfies the triangle inequality. That is, for all vertices $u, v, w \in V$, we have $c(u,w) \leq c(u,v) + c(v,w)$.

   Give a polynomial time reduction from TSP problem to Metric TSP. Hint: You only need to change the capacity function *c.*

Solution:

Consider the following transformation.

1. Find the largest cost in the input graph, say $C_{max} = \max\limits_{u,v} c(u,v)$.
2. Output the same graph with the new cost function $c' = c + C_{max}$ (add C_{max} to every edge cost) and with a new bound of $k' = nC_{max} + k$.

Both steps are linear in the size of the input, so this reduction is polynomial time.

This new cost function c' satisfies the triangle inequality, because
$c'(u,v) = c(u,v) + C_{max} \leq 2C_{max} \leq c(u,w) + C_{max} + c(w,v) + C_{max} = c'(u,w) + c'(w,v)$.

To show that the above transformation is a reduction, we argue as follows. Any cycle can be represented as a collection of edges $(e_1, \ldots e_n)$. From the definition of c', we have that

$$\sum_{i=1}^{n} c'(e_i) = \sum_{i=1}^{n}(c(e_i) + C_{max}) = nC_{max} + \sum_{i=1}^{n} c(e_i).$$

Therefore, there is a cycle of cost at most k, under the cost function c if and only if there is a cycle of cost $k + nC_{max}$ under the cost function c'.

6. The Knapsack problem may be stated as follows. Given n items with weights $w_1 \ldots w_n$ and corresponding values $v_1 \ldots v_n$, is there a subset of items $S \subseteq \{1 \ldots n\}$ such that the sum of the weights is at most some capacity W (i.e. $\sum_{i \in S} w_i \leq W$) and the sum of the values is at least V (i.e. $\sum_{i \in S} v_i \geq V$)? The traditional interpretation is that a robber wants to rob an art gallery and make off with most valuable collection of pieces that he can carry in his knapsack. Show that Knapsack is NP-complete (Hint: reduce from Subset-Sum).

Solution:

Knapsack is in NP. Given a subset S as a certificate, we can confirm that the two inequality constraints hold in $O(|S|)$ time on a RAM.

We reduce Subset-Sum to Knapsack. An instance of the Subset-Sum problem is a list of numbers $a_{1,} \ldots a_n$ and a number k. The decision problem is whether there is a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = k.$

We transform this into a Knapsack problem through the following redefinition. Given list $a_{1,} \ldots a_n$ and number k, the transformation returns

1. a list of weights $w_1 \ldots w_n = a_1 \ldots a_n$
2. a maximum capacity of $W = k$
3. a list of values $v_1, \ldots v_n = a_1 \ldots a_n$
4. a minimum value of $V = k$.

Since the transformation is mostly a matter of copying, we claim that it is polynomial.

To see that the transformation is a reduction, suppose that there is a subset $S \subseteq \{1, \ldots n\}$ such that $\sum_{i \in S} a_i = k.$ Then this same subset satisfies the requirements for a positive instance of the knapsack problem: $\sum_{i \in S} w_i = \sum_{i \in S} a_i = k = W$ and $\sum_{i \in S} v_i = \sum_{i \in S} a_i = k = V$.

On the other hand, suppose that $S \subseteq \{1, \ldots, n\}$ satisfies the requirements for the knapsack problem. Then

$k = V \leq \sum_{i \in S} v_i = \sum_{i \in S} a_i = \sum_{i \in S} w_i \leq W = k$, so $\sum_{i \in S} a_i = k$. Therefore, S satisfies the Subset-Sum requirement as well. Hence the above transformation is a polynomial reduction.