Exercise Two

Remarks: When you describe an NPC reduction remember to show it is polynomial. When you show a problem is NPC remember to show it belong to NP.

The problems which you may assume to be known to belong to NPC are:

1. SAT
2. 3-SAT
3. 3-SAT-3
4. Clique
5. Independent set
6. Vertex cover
7. Hamiltonian path
8. 3-coloring
9. Subset sum
10. TSP
11. Spanning tree with maximum degree at most d
12. Dense k-subgraph
13. Subgraph Isomorphism

1. Question 1:
   Show that the following problems belong to NPC:

   (a) The Min-leaf Spanning tree problem:

   **Input:** An undirected and connected graph $G$ and a number $d$.

   **Question:** Is there a spanning tree with at most $d$ leaves (a leaf is a vertex of degree 1)?

   **Answer:** A witness would be a subset $E'$ of the edges. Checking that $G(V, E')$ is a tree was described before (an $O(|E| + |V|)$ algorithm). Counting the degrees and checking how many vertices have degree 1 can also be done in $O(|E| + |V|)$. We now show that the problem is harder than Hamiltonian path. If we take $d = 2$ we require a spanning tree with at most 2 leaves. This must be an Hamiltonian path

   (b) The Knapsack problem:
**Input:** A vector $\mathbf{V} = (v_1, v_2, \ldots, v_n)$ of values and a vector $\mathbf{W} = (w_1, w_2, \ldots, w_n)$ of weights and an upper bound $\bar{W}$ and a lower bound $\underline{V}$

**Question:** Is there a subset $S$ of the indices so that:

$$\sum_{i \in S} w_i \leq W,$$

and

$$\sum_{i \in S} v_i \geq V?$$

**Answer:** The input size is at most $n \cdot \log M$ with $M$ the maximum value of a number in the input. The witness is the subset $S'$. Summing two numbers of at most $\log M$ bits can be done by $O(\log M)$ operations on bits (you add two digits and may have a carry, etc). Thus, summing the numbers of $S'$ requires no more than $O(\log M \cdot n)$, namely a polynomial time in the input size.

The reduction would be from the subset-sum problem. Let $A = \{a_1, a_2, \ldots, a_n\}$, $X$ be the input to the subset-sum question and the goal is to check if there is a subset $S$ of the elements whose sum is $X$. Define the following input for the Knapsack problem. Let $v_i = w_i = a_i$. Let $W = V = X$. Under this definition a feasible Knapsack solution will be one obeying:

$$\sum_{i \in S} a_i \leq X$$

and

$$\sum_{i \in S} a_i \geq X.$$  

In other words,

$$\sum_{i \in S} a_i = X.$$  

Hence the answer to the Knapsack instance is yes if and only if the answer to the subset-sum instance is yes.

(c) The almost Hamiltonian path question:

**Input:** An undirected graph $G$

**Question:** Does $G$ contain a simple path of length $n - 2$ or more?

**Answer:** The witness is a collection of $n - 1$ vertices $v_1, v_2, \ldots, v_{n-1}$. We need to check if $(v_1, v_2) \in E$, and $(v_2, v_3) \in E$ and so on. This requires $O(n)$ in total, if the graph is represented by a Matrix (or $O(|E|)$ otherwise).
The problem is harder than Hamiltonian path. Take \( G(V, E) \) a graph input to Hamiltonian path. Add a new vertex \( w \not\in V \) with no edges, to get a new graph \( G' \). If \( |V| = k \), then \( |V'| = |\{w\} \cup V| = k + 1 = n \).

Clearly, \( G' \) has a simple path of length at least \( n - 2 = k - 1 \) if and only if \( G \) has a path of length \( k - 1 \) (\( w \) who has no edges can not participate in a path like that). Thus, \( G \) must have an Hamiltonian path.

(d) The 7 coloring problem

**Input:** An undirected graph \( G \)

**Question:** Can \( G \) be colored by 7 colors?

**Answer:** The witness is an assignment of one of the numbers 1 to 7 to every \( v \in V \). Let \( c(v) \) be the number assigned to \( v \). We go over all edges \( e = (u, v) \) and only need to check that \( c(u) \neq c(v) \). This requires \( O(|E|) \) time.

The problem is harder than 3 coloring. Let \( G(V, E) \) be a graph instance for the 3—coloring problem. Add 4 new vertices \( v_1, v_2, v_3, v_4 \) (not in \( V \)) to \( G \). Connect each pair of \( v_i, v_j, i \neq j \) by an edge and connect every \( v_i, 1 \leq i \leq 4 \) to every \( v \in V \). Let \( G' \) be the new graph.

We will need to use 4 different colors for \( v_1, v_2, v_3, v_4 \). The colors must be different as the \( v_i, v_j \) pairs are neighbors. Now, only 3 colors are left to color the \( G \) vertices. These colors must be different then the \( v_i \) colors (as \( v_i, 1 \leq i \leq 4 \) are all connected to all of \( V \)). Thus, it is possible to 7—color \( G' \) if and only if \( G \) is 3—colorable.

(e) Maximum edges 3 colorable subgraph

**Input:** A graph \( G(V, E) \) and a number \( i \)

**Question:** Is there a subgraph \( G'(V', E') \) so that \( |E'| \geq i \) and \( G' \) is 3—colorable?

**Answer:** The witness is the subgraph \( G' \) and the coloring of the vertices. We need to check if the colors are legal by going over all edges in \( O(|E'|) = O(|E|) \) time.

The problem is harder than 3—coloring. Because, choose \( i = |E| \). So we need \( |E'| \geq |E| \). As \( E' \subseteq E \) this is only possible if \( E' = E \). This means that \( V' = V \) (unless we have vertices of degree 0 but these vertices can be ignored in the first place).

Thus the question becomes: is \( G \) 3—colorable?

2. **Question 2:** Show that the following problems are polynomially solvable. You may use the fact that distances between vertices in graphs can be found in polynomial time.

   (a) Find a spanning tree with maximum weight. (Recall the minimum spanning tree requires minimum weight)
Answer: Let $M$ be the maximum edge weight. Define new weights as follows. If $e$ has weight $w(e)$, its new weight is $M + 1 - w(e)$. Observe that $w'(e)$ are positive numbers. Now, run a minimum spanning tree algorithm. The algorithm has to choose a spanning tree $T(V, E')$ so as to minimize:

$$\sum_{e \in E'} w'(e) = \sum_{e \in E} (M + 1 - w(e)) = (n - 1) \cdot (M + 1) - \sum_e M_e.$$

The last inequality follows as every spanning tree has $n - 1$ edges. Since $(n - 1) \cdot (M + 1)$ is a constant (does not depend on $T$), the best solution finds a spanning tree with maximum value of $\sum_e M_e$. Namely, a maximum spanning tree.

(b) Find the connected components of a graph. (The connected components of an undirected graph are the connected parts).

Answer: Recall that the BFS algorithm accepts as input a graph $G$ (undirected in this case) and a vertex $s$ and finds $\text{dist}(s, u)$ (the unweighted distance from $s$ to $u$) for every $u \in V$.

The vertices of finite distance from $s$ are the vertices in the connected component of $s$. The vertices that are not reachable by a path from $s$ will have $\text{dist}(u, s) = \infty$.

Thus, run the BFS algorithm from $s$. Remove the resulting connected component from $G$. If $G$ is not empty, choose another vertex $s'$. The next connected component are the vertices of finite distance from $s'$. And so on.

This algorithm goes over every edge at most twice and on every vertex at most once hence runs in $O(|E| + |V|)$.

(c) Given a graph $G$ can $G$ be colored with 2 colors?

Answer: Let the two colors be 1 and 2. Without loss of generality, let $u$ be colored 1. We claim that if $G$ is 2 colorable then vertices of even distance from $v$ must be colored 1, and vertices of odd distance from $v$ must be colored 2. This is proved by induction. For distance $0$, $v$ is colored 1. For distance 1 (the neighbors of $v$) they must be colored 2 (because $v$ is colored 1).

Assume vertices at distance $2k$ are colored 1 and of $2k + 1$ are colored 2. Vertices $w$ of distance $2k + 2$ have a neighbor $u$ at distance $2k + 1$. By the induction hypothesis, $u$ is colored 2, hence $w$ must be colored 1. Similarly for $2k + 2$.

Hence, run BFS and compute the vertices $V_e$ and $V_o$ of even and odd distance from $v$. There should be no edges inside $V_e$ and $V_o$. If this is the case then the graph is 2-colorable. Otherwise, it is not. The algorithm runs in $O(|E| + |V|)$ time.

3. Question 3: In this question we show that 2-SAT is polynomial. Recall that 2-SAT is like the SAT problem except that every clause is an or of two literals (a literal is $x$ or $\bar{x}$). The answer appears in pages 184 of the book. But, try it alone first!
Let $F$ be the 2–SAT formula. Construct a graph $G$. For every boolean variable $x$, we have a vertex $v_x$ and $v_{\bar{x}}$.

Let $w, w'$ be some literals. Put a directed edge from $v_w$ to $v_{w'}$ if both $(w + \bar{w'})$ and $(w' + \bar{w})$ are clauses that belong to $F$.

**Remark:** If $w = \bar{x}$ then $\bar{w} = x$.

(a) Draw the graph for :

$$F = (\bar{x} + y) * (\bar{z} + x) * (\bar{y} + x) * (\bar{w} + z) * (w + z) * (\bar{x} + z).$$

(b) Show that if $x$ is assigned true, and there is an edge from $x$ to $y$, then $y$ must be assigned true as well (and otherwise, the assignment is not satisfying).

**Answer:** The clause $\bar{x} + y$ must exist. Thus, if $x$ is $T$, then in order to satisfy this clause you need $y = T$.

(c) Show that if there is in the graph a path from $v_x$ to $v_{\bar{x}}$ for some $x$ then the formula can not be satisfied.

**Answer:** The previous item implies that if the path is $v_x \rightarrow v_z \cdots \rightarrow v_{\bar{x}}$ then $z$ and all the other vertices in the path, including $\bar{x}$ must be assigned $T$ if $x = T$. But you cant have both $x = T$ and $\bar{x} = T$.

(d) Show that if the formula has a satisfying assignment, then for any $x$ there is no path from $v_x$ to $v_{\bar{x}}$.

**Answer:** Assume for the sake of contradiction that there is a path $v_x \rightarrow v_{y_1} \rightarrow v_{y_2} \cdots \rightarrow v_{y_k} \rightarrow v_{\bar{x}}$ in $G$. Thus, $x$ can not be $T$ (as explained above because this would imply that $\bar{x} = T$ as well). But the definition of $G$ is symmetric. Thus, $\bar{x}$ can not be $T$ as well. But one of $x$ or $\bar{x}$ must be true. A contradiction.

(e) Show that 2–SAT is polynomial time solvable.
**Answer:** Building $G$ is done in time linear in the input. For every one of the $n$ literals, you check in $O(n + m)$ time ($m$ is the number of clauses) if $x$ can reach $\bar{x}$ by a path, using $BFS$. Thus, in $O(n \cdot (n + m))$ we check if the formula can be satisfied.

**Question 4:** Show that the following problems are undecidable:

(a) **Input:** A program $P$ and two inputs $x$ and $y$

**Question:** Does $P$ stop at $x$ but does not stop at $y$?

**Answer:** Let $P', x$ be an instance of the Halting problem. We show that an oracle for the new problem can be used to solve the Halting problem. We can write a new program $P$ as follows:

(b) i. If the input $z \neq x$, $z \neq y$, then Return(true)

ii. Else, if $z = y$, then

A. $t \leftarrow 0$

B. While(True) do $t \leftarrow t + 1$

iii. Else, run $P$ on $x$

Clearly, $P$ does not stop at $y$. Thus, $P$ stops at $x$ but does not stop at $y$ if and only if $P'$ stops at $x$ and we got a reduction to the Halting problem.

**Input:** A program $P$ that accepts as input integers

**Question:** Does $P$ stops on input equal to 0?

**Answer:** Let $P', x$ be an instance of the Halting problem. Write $P$ with input $z$ as follows:

i. If $z \neq 0$ then exit

ii. Else, run $P'$ on $x$

Clearly $P$ stops at 0 if and only if $P'$ stops at $x$. Hence an oracle for the new problem implies an algorithm for the Halting problem.