Chapter 6

1. This problem is in P.

   Here is a polynomial-time algorithm to solve it: scan the array once.

   This takes $O(n^1)$ time.

3. This problem is in P.

   Here is a polynomial-time algorithm to solve it: run Floyd-Warshall algorithm. Than scan the final table. If all numbers are smaller than or equal to k, return YES, otherwise return NO.

   This takes $O(n^3)$ time.

5. This problem is in NP.

   We first argue that it is in P, which implies that it is in NP since $P \subseteq NP$ (as seen in class).

   Here is a polynomial-time algorithm to solve it: If $A[j] \leq A[i]$, return false. If $A[j] > A[i]$, run the following algorithm.

   First, scan A and replace each number $x \in A$ by a pair (x, k), where k is the index of x in A. This takes $O(n)$ time. Second, sort A (with respect to the first number in each pair) using Merge sort. This takes $O(n \log(n))$ time. Third, scan A to find the smallest difference between two consecutive numbers (only consider the first number in each pair) and keep track of these two consecutive numbers. This takes $O(n)$ time. Suppose that the two consecutive numbers that were found are (x, i') and (y, j'), then return i' and j'. This takes $O(1)$ time. In total, this algorithm takes $O(n) + O(n \log(n)) + O(n) + O(1) = O(n \log(n))$ time.

   Let i' and j' be the indices returned by this algorithm. If $A[j] - A[i] = A[j'] - A[i']$, return true, otherwise, return false.

   This takes $O(n \log(n)) = O(n^2)$ time.

7. This is true. We proved in class that $P \subseteq NP$. So if $L \in P$, then $L \in NP$.

9. Recall that

   $VERTEX\text{−}COVER = \{(G, k) \mid G \text{ is a graph which contains a vertex cover with } k \text{ vertices.}\}$

   The verification algorithm V takes as input:

   - a graph $G = (V, E)$ together with an integer k
   - and a certificate $v_1, v_2, ..., v_k$.

Step 1: Check if $\{v_1, v_2, ..., v_k\} \subseteq V$.

Step 2: Check if $|\{v_1, v_2, ..., v_k\}| = k$.

Step 3: For all edges $e \in E$, check if at least one endpoint of e is in $\{v_1, v_2, ..., v_k\}$.

Step 4: If Steps 1, 2 and 3 were successful, return YES, otherwise return NO.

We have

$$
\begin{aligned}
& (G, k) \in VERTEX - COVER \\
\iff \; & G \text{ contains a vertex cover with } k \text{ vertices.} \\
\iff \; & \text{there exists a set } \{v_1, v_2, ..., v_k\} \subseteq V \text{ with } k \text{ vertices such that} \\
& \text{each edge in } E \text{ has at least one enpoint in } \{v_1, v_2, ..., v_k\}. \\
\iff \; & \text{there exists a certificate } \{v_1, v_2, ..., v_k\} \text{ such that} \\
& \text{such}
\end{aligned}
$$

* the length of the certificate $= k = O(|V|) = O(\text{size of } G)$,
* $V(G, \{v_1, v_2, ..., v_k\})$ returns YES
* and the running time of $V = O(k|E|) = O(|V|\,|E|) = O\left((\text{size of } G)^2\right)$.

11. The definition of 2SAT is similar to the one for 3SAT. We just change 3 for 2!

Consider a Boolean formula ϕ with variables $x_1, x_2, ..., x_n$ of the form

$$\phi = C_1 \wedge C_2 \wedge ... \wedge C_m,$$

where each C_i is of the form

$$C_i = \ell_1^i \vee \ell_2^i.$$

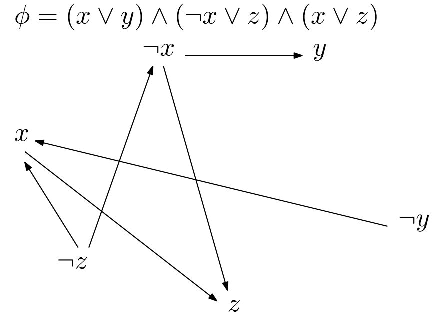
Each ℓ_j^i is a variable or the negation of a variable. Then

$$2SAT = \{\phi \mid \phi \text{ is of the form above and } \phi \text{ is satisfiable}\}$$

How do we show that it is in P?

Let ϕ be a 2SAT Boolean formula with m clauses and n variables. Here is the algorithm. We build a *directed* graph $G = (V, E)$ "corresponding" to ϕ in the following way. The set V is made of $2n$ vertices, one for each variable and one for each negated variable. For each clause $(\ell_1 \vee \ell_2)$, we have two edges in E, namely $(\neg\ell_1, \ell_2)$ and $(\neg\ell_2, \ell_1)$.

$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (x \vee z)$



Here is an example:

An edge $(\neg \ell_1, \ell_2)$ means "if ℓ_1 is false, then ℓ_2 must be true". Therefore, ϕ is unsatisfiable if and only if there is a variable α such that in G, there is a path from α to $\neg\alpha$, and, there is a path from $\neg\alpha$ to α.

13. Recall that if $G = (V, E)$ is a graph and k is an integer, then $f(G, k) = \left(\overline{G}, n - k\right)$, where $\overline{G}$ is the complement of G and $n = |V|$.

    For the first condition to be satisfied, we need to show that any input (G, k) to $CLIQUE$ is transformed into an input to $VERTEX-COVER$. Since $f(G, k)$ returns a graph and an integer, then $f(G, k)$ is an input to $VERTEX-COVER$.

    For the third condition to be true, we need to show that $f(G, k)$ can be computed in polynomial time. Computing the complement of a graph can be computed in $O(|V| + |E|)$ time. Refer to the exercises of Chapter 3. The number $n - k$ can be computed in $O(1)$ time.

15.

    (a) We need a function f such that
        (1) $f : (A[1..n], x) \longrightarrow (A'[1..n'], x')$
        (2) x is the minimum element in $A[1..n]$ if and only if x' is the maximum element in $A'[1..n']$
        (3) f can be computed in polynomial time

        The function f will produce an array A' where A' and A have the same size. Let $A'[1..n]$ be such that for all $1 \leq i \leq n$, $A'[i] = -A[i]$. The function f is $f(A[1..n], x) = (A'[1..n], -x)$.
        Since f produces an array and a number, the first condition is satisfied.
        For the second condition,

$$\begin{array}{rl} & x \text{ is minimum in } A \\ \Longleftrightarrow & \text{for all } 1 \leq i \leq n,\ x \leq A[i] \\ \Longleftrightarrow & \text{for all } 1 \leq i \leq n,\ -x \geq -A[i] \\ \Longleftrightarrow & \text{for all } 1 \leq i \leq n,\ -x \geq A'[i] \\ \Longleftrightarrow & -x \text{ is maximum in } A' \end{array}$$

        Computing A' takes $O(n)$ time and computing $-x$ takes $O(1)$ so the third condition is satisfied.

    (b) The proof is symmetric.

17. Let $G = (V, E)$ be an undirected graph, where $n = |V|$.

    Notice the following property: let $V' \subset V$ be a vertex cover of G of size $k < n$. Let $v \in V \setminus V'$ be a vertex that is not in the vertex cover. Then $V' \cup \{v\}$ is also a vertex cover of G.

We can prove that little property in the following way: let $e \in E$ be an arbitrary edge. Since V' is a vertex cover, e has at least one endpoint in V'. Therefore, e has at least one endpoint in $V' \cup \{v\}$. Therefore, $V' \cup \{v\}$ is a vertex cover.

From that property, we can say that if there is no vertex cover of size k, then there is no vertex cover of size $k' < k$.

Suppose that the algorithm $\mathcal{A}$ to solve VERTEX-COVER takes $O(n^c)$ time, where $c > 0$ is a constant. Then for all $1 \leq i \leq n$, we call $\mathcal{A}(G, i)$. As soon as we find a value i for which there exists a vertex cover of size i, we stop. This is the size of a smallest vertex cover. In the worst case, the running time is $O(n^{c+1})$.

We can also use a binary search strategy and we get a running time of $O(n^c \log(n))$.

19. Since $L \in P$ and $P \subseteq NP$ (theorem presented in class), then $L \in NP$. Then, since L' is NP-Complete, $L \leq_P L'$ (from the definition of NP-Complete).

21. We have to find a function f such that

    (1) f takes any input G to $HAMCYCLE$ and produce an input (G', K) for TSP.

    (2) $G \in HAMCYCLE$ if and only if $f(G) \in TSP$.

    (3) $f(G)$ can be computed in polynomial time.

    We first describe f, then we show that f satisfies the three "famous" properties. Let $G = (V, E)$ and let $(G', K) = f(G)$, where $G' = (V', E')$. Recall that G' has to be a complete directed weighted graph.

    Take $K = n$, where $n = |V|$, and $V' = V$ (so the set of vertices is the same). For the set E' of edges, proceed in the following way. For each edge $\{u, v\} \in E$, add an edge (u, v) in E' with weight 1 and add an edge (v, u) in E' with weight 1. For each pair of vertices $w, z \in V$ not connected by an edge in G, add an edge (z, w) in G' with weight ∞ and add an edge (w, z) in G' with weight ∞. Refer to Figure 1.

    Property (1) is easily verified. Indeed, $f(G)$ does produce a complete directed weighted graph together with an integer. So it does produces an input for TSP.

    Property (3) is also easy to verify. In a complete directed graph, there are $n(n-1)$ edges, so G' can be computed in $O(n^2)$ time.

    Let us now see why Property (2) is verified.
    [$\Longrightarrow$] Assume that G has a Hamiltonian cycle. If you follow the same sequence of vertices in G', you get a path with weight n. So TSP solves to true.

    [$\Longleftarrow$] Assume that there is a Hamiltonian cycle in G' with total weight at most n. Then, this cycle cannot involve edges with weight ∞. Therefore, it only involves edges that exist in G. Therefore, this Hamiltonian cycle correspond to a Hamiltonian cycle in G.

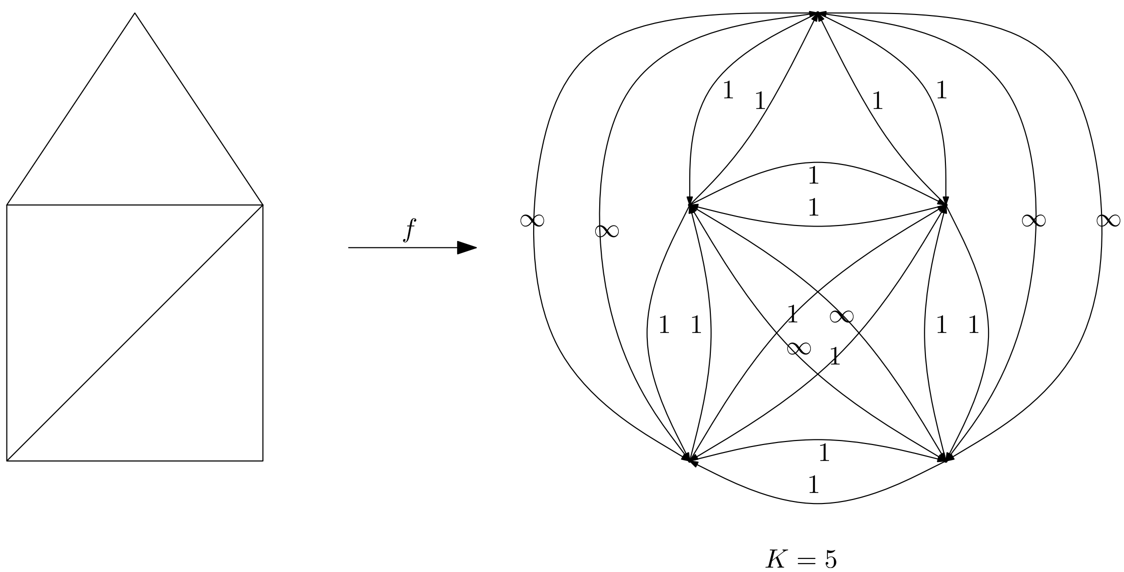


Figure 1: Illustration of Question 21.