Assignment 3 solutions

1. The algorithm follows the proof of Dirac's theorem. However, we cannot start by finding the longest path (since this is NP-complete). But what we do notice in the proof Dirac's theorem is that if the supposedly longest path P that we start off with is not actually a longest path, we always arrive at a contradiction by finding a longer path.

   Thus, we can design a subroutine which takes the original input graph and a path in it as input and either outputs a longer path or a Hamiltonian cycle. We can then start with any path (e.g., a single vertex) and iteratively make it longer using our subroutine.

   We can simply this a bit since we expect the longest path to contain $|V(G)|$ vertices where G is the input graph. So we can just write a subroutine which gives us a longer path if the input path does not have length $|V(G)|$.

   We first describe the subroutine.

   Algorithm 1. An algorithm which takes an input graph G and path P and returns a longer path in G. Should only be called if $\deg(v) \geq |V(G)|/2$ for all $v \in V(G)$ and P does not contain all vertices of G.

   findlongerpath($G, P = \{p_1, \ldots, p_k\}$)
     //First, we check if any endpoints have neighbours not in the path.
     //If it does, extend the path by one of those neighbours.
     If $N(p_1) \setminus V(P)$ is non-empty then
       Add any vertex $v \in N(p_1) \setminus V(P)$ to the beginning of P and return this new path.
     If $N(p_k) \setminus V(P)$ is non-empty then
       Add any vertex $v \in N(p_k) \setminus V(P)$ to the end of P and return this new path.
     $S \leftarrow \{i|p_i \in N(p_1)\} \cap \{i+1|p_i \in N(p_k)\}$
     //We expect S to be non-empty for the same reason as in the proof of Dirac's theorem
     //(i.e., since the degree of every vertex is at least $|V(G)|/2$).
     $p_i \leftarrow$ any element of S (e.g., the first element).
     //We now expect $p_1, p_i, p_{i+1}, \ldots, p_{k-1}, p_k, p_{i-1}, \ldots, p_2$ to be a cycle.
     For j from 1 to k
       If $N(p_j) \setminus V(P)$ is non-empty then
         $v \leftarrow$ any element of $N(p_j) \setminus V(P)$
         If $j \leq i$ then
           Return the path $v, p_j, p_{j-1}, \ldots, p_1, p_i, p_{i+1}, \ldots, p_{k-1}, p_k, p_{i-1}, \ldots, p_{j+1}$
         Else //$j \geq i+1$
           Return the path $v, p_j, p_{j+1}, \ldots, p_k, p_{i-1}, p_{i-2}, \ldots, p_2, p_1, p_i, \ldots, p_{j-1}$
     //We expect one of the sets $N(p_j) \setminus V(P)$ to always be non-empty as in the proof of Dirac's theorem.

   We now describe the main algorithm.

   Algorithm 2. An algorithm which takes an input graph G and returns a Hamiltonian cycle in G. Should only be called if $\deg(v) \geq |V(G)|/2$ for all $v \in V(G)$.

   findhamcycle(G)
     If $V(G)$ is empty then return the empty path.
     $v \leftarrow$ any element of $V(G)$
     $P \leftarrow \{v\}$
     While $|P| < |V(G)|$
       $P \leftarrow$ findlongerpath(G, P)
     $S \leftarrow \{i|p_i \in N(p_1)\} \cap \{i+1|p_i \in N(p_k)\}$
     //We expect S to be non-empty for the same reason as in the proof of Dirac's theorem
     //(since the degree of every vertex is at least $|V(G)|/2$).
     Return $p_1, p_i, p_{i+1}, \ldots, p_{k-1}, p_k, p_{i-1}, \ldots, p_2$

Now the main while loop of algorithm runs at most n times since the length of P increases by 1 during every iteration. We can add an extra factor of n^2 for set operations (so that we do not need to analyze the running time of each specific set operation). The main loop of the subroutine runs in $k = |P| < n$ iterations. So the total running time of our algorithm is $O(n * n^2 * n) = O(n^4)$. The analysis and algorithm can both easily be improved.

2. (a) The construction we use to show that $G \times P_2$ contains a Hamiltonian cycle is the same as the construction for Gray codes.

   Suppose G has a Hamiltonian cycle $C = \{c_1, c_2, \ldots, c_n\}$. We claim that $\{(c_1, v_1), (c_2, v_1), \ldots, (c_n, v_1), (c_n, v_2), (c_{n-1}$ is a Hamiltonian cycle.

   By definition of the Cartesian product and the fact that C is a Hamiltonian cycle, there is an edge $((c_i, v_1), (c_{i+1}, v_1))$ for i from 1 to $n-1$ and there is an edge $((c_{i+1}, v_2), (c_i, v_2))$ for i from 1 to $n-1$. Thus it remains to check there exists edges $((c_n, v_1), (c_n, v_2))$ and $((c_1, v_2), (c_1, v_1))$ in $G \times P_2$. These edges exist by definition of the Cartesian product and the fact that (v_1, v_2) is an edge in P_2.

   This cycle visits all vertices of $G \times P_2$ since C visits all vertices of G.

   (b) Let $C = \{c_1, c_2, \ldots, c_n\}$ be a Hamiltonian cycle in G_1 and $C' = \{c'_1, c'_2, \ldots, c'_n\}$ be a Hamiltonian cycle in G_2.

   We claim that

$$\begin{array}{llll}
\{(c_1, c'_1), & (c_2, c'_1), & \ldots, & (c_n, c'_1), \\
(c_n, c'_2), & (c_1, c'_2), & \ldots, & (c_{n-1}, c'_2), \\
(c_{n-1}, c'_3), & (c_n, c'_3), & \ldots, & (c_{n-2}, c'_3), \\
(c_{n-2}, c'_4), & (c_{n-1}, c'_4), & \ldots, & (c_{n-3}, c'_4), \\
\vdots & \vdots & \ddots & \vdots \\
(c_2, c'_n), & (c_3, c'_n), & \ldots, & (c_1, c'_n)\}
\end{array}$$

   is a Hamiltonian cycle in $G_1 \times G_2$.

   Indeed, since C is a Hamiltonian cycle, by definition of the Cartesian product, for each i and j, $(c_i, c'_j), (c_{i+1 \bmod n}, c'_j)$ is an edge in $G_1 \times G_2$. Since C' is a Hamiltonian cycle, by definition of the Cartesian product, for each i and j, $(c_i, c'_j), (c_i, c'_{j+1 \bmod n})$ is an edge in $G_1 \times G_2$.

3. (a) This statement is false. For example, take $T_1 = P_1$ and $T_2 = P_2$. Then $|E(T_1)| = 1$ and $|E(T_2)| = 2$.

   (b) This statement is true.

   Let $e = (u, v)$ be an arbitrary edge in T. Removing e disconnects u from v since if there is a path from u to v, that path is a cycle in T when we add back e.

   (c) We prove this statement by induction on the number of vertices in T.

   If $|V(T)| = 2$, this is true since there is only one tree on 2 vertices, namely P_2. Both vertices of P_2 have degree 1.

   Suppose the statement is true when $|V(T)| = n - 1 \geq 2$. Let T be any tree on n vertices. By the lemma seen in class, T contains a vertex v of degree 1.

   We claim that $T - v$ is a tree (on $n - 1$ vertices). $T - v$ has no cycles as a cycle in $T - v$ is a cycle in T. $T - v$ is connected as a path in T between two vertices $u, w \in V(T - v)$ is still a path in $T - v$ since all internal vertices (non-endpoints) of a path have degree at least 2.

   By our inductive hypothesis, $T - v$ contains two vertices u and w, each of degree 1 in $T - v$. At least one of u or w is not adjacent to v in T. Thus, that vertex has the same degree in $T - v$ and T (namely, degree 1). Therefore, that vertex and v are two degree 1 vertices in T.

   Therefore, the original statement is true by induction.

4. (a) We sort the edges by weight and consider them in the following order $\{(C,E),(D,G),(E,G),(A,B),(D,E),(B,F),($

We do not add (D,E) since D,E,G would form a cycle.

We stop after adding (F,H) since we already have 7 edges and the graph contains 8 vertices (so any added edge would necessarily form a cycle).

(b) We only need to modify the algorithm slightly so that we only start with E_0 rather than the empty set of edges. The remainder of the algorithm remains the same.

Algorithm 3. (Modified Kruskal's algorithm)

Initialize F to E_0.
Sort the edges not in E_0 in ascending order of weights
For each edge e in this ordering.
  If $(V, F \cup \{e\})$ does not contain a cycle then add e to F
Return F

Theorem 1. *Modified Kruskal's algorithm gives the minimum weight output.*

The proof is essentially the same as the proof of Kruskal's algorithm. We make use of the following theorem we proved in class.

Theorem 2. *Let $T = (V, E)$ be a tree and $e \notin E(T)$. Then the graph $(V, E \cup \{e\})$ contains a unique cycle C (and C contains e).*

Proof. Let F^* an optimal solution which maximizes the number of edges of F it contains (that is, it maximizes $|F \cap F^*|$. If $F = F^*$ then we have proven the theorem (since F^* is a optimal).

Let F_i be the set of edges at the first iteration of the for-loop where we added an edge $e \notin F^*$ to F_i. So $F_{i+1} = F_i \cup \{e\}$. By Theorem 2, $(V, F^* \cup \{e\})$ contains a unique cycle C.

Since (V, F_{i+1}) contains no cycles, there is an edge f of C not in F_{i+1}. We claim that $w_f \geq w_e$.

Suppose not. If $w_f < w_e$ then we claim that our algorithm could have picked f instead of e (contradiction our choice of e). Indeed, if we cannot pick f, it is because f is contained in a cycle in $(V, F_i \cup \{f\})$ which is a path P between the endpoints of f in (V, F_i). Since F_i is contained in F^* (because e is the first edge not in F^*), P is a path in (V, F^*). But f is also in F^*. So P together with f is a cycle in F^*. Contradiction (to F^* being a tree).

Therefore, $w_f \geq w_e$. Since C is the unique cycle in $(V, F^* \cup \{e\})$ and f is in C, removing f from $(V, F^* \cup \{e\})$ gives a tree (which is still connected). But this new tree has lower or equal weight than (V, F^*) and contains more edges of F than F^* (namely, e). Contradiction to our choice of F^*. □

5. Suppose the statement is false. Let T be a counter-example and P_1, P_2 be two longest path in T which do not share a vertex.

Let P be a path from some vertex $u \in P_1$ to some vertex $v \in P_2$ with no internal vertices in P_1 or P_2 (such a path exists since T is connected and can be obtained by taking any path from any vertex in P_1 to any vertex in P_2 and shortening it).

We build four paths Q_1, Q_2, Q_3, Q_4, each obtain from P by adding a subpath of P_1 from u to an endpoint of P_1 and a subpath of P_2 from v to an endpoint of P_2 (we get different paths depending on which endpoint we chose).

We claim that one of these four path is longer than P_1 (or P_2). This would lead to a contradiction.

Taking the union of all vertices in Q_1, Q_2, Q_3, Q_4 (with repetition), we obtain a set of vertices which contains u and v four time, all other vertices of P_1 twice and all other vertices of P_2 twice.

Thus, the sum of the length of all four paths is at least to twice the length of P_1 plus twice the length of P_2 plus two. If k is the length of P_1 (and P_2), this sum is at least $4k + 2$. Thus, at least one of the four paths has length a quarter of this sum which $k + \frac{1}{2} > k$. Contradiction to k being the length of the longest path.

6. See code. Python's language specific code was avoided as much as possible (except for parsing the input) so that it would be easier to read for people not familiar with the language. The only thing to note is that square brackets denote lists. They can be concatenated using the "+" sign. A sublist can be obtained using ":" (e.g., `A[1:4]` is a list containing `A[1]`,`A[2]`,`A[3]`,`A[4]` and `A[4:1:-1]` cotains the elements `A[4]`,`A[3]`,`A[2]`).

   The code corresponds to the pseudo-code given in question 1 exactly.

   Also includes a function `checkhamcycle` to test your submissions.