In all discussions about graphs, assume \( n \) denote the number of vertices and \( m \) is the number of edges. In questions 1, 2 and 3 we refer to Rabin-Karp algorithm for string matching. The notation is as used in the slides.

1. This algorithm is used for finding all values \( s \) such that \( T[s+1 \ldots s+m] = P \). Assume that we actually only looking for a matchings where \( s = 16k \) where \( k \geq 0 \) is an integer. (that is, \( s = 0, 16, 32, 48, \ldots \)). Explain how to modify the algorithm so \( t'_s \) is computed only for these values of \( s \). For example, the case \( t'_3 \) need not be computed.

2. Assume the value of the prime \( q \) is known to you. You are also given integers \( m, n \). Explain how to generate a text \( T[1..n] \) and a pattern \( P[1..m] \) for which the number of false positive matchings (\( t' \neq p \) but \( t'_s = p'_s \)) is \( \Omega(n/m) \).

   **Answer:** Pick arbitrary an \( m \)-character string, and call it \( P \). Using the notation of the slides, let \( p \) be its binary value, and let \( p' = p \mod q \). Let \( \hat{p} = p + q \) and let \( \hat{P} \) be the \( m \) character string whose binary representation is \( \hat{p} \). Note that \( \hat{p} \mod q = t \mod q = t' \).

   Finally, given \( n \), we are concatenating \( n/m \) copies of \( \hat{P} \). This new string is \( T \).

3. Would you prefer \( q \) to be a smaller or a larger prime? Discuss for example the case \( q = 2 \) and \( q = 3 \) vs. the case \( q > 2^{20} \). How does this influence the way you construct \( \Pi \)?

   **Answer:** The smaller \( q \) is, the more likely it is to find a text and pattern that would be falsely align. For example, for \( q = 2 \), and fixed \( P = P[1..m] \), if we consider the set \( S \) of all possible \( m \)-characters strings \( T \), then half the strings of \( S \) causes a false positive.

   Hence we aim to find large primes \( q \). Note that we still want them to be smaller than \( 2^{64} \) so we could manipulate them in single computer access.

   One way to obtain it is delete from \( \Pi \) any small prime. Other is to avoid building \( \Pi \) explicitly, but only to look for large primes.

4. Let \( G(V, E) \) be a directed graph with positive weights on its edges, and let \( s \in V \) be a node. Suggest an \( O((m + n) \log n) \) time algorithm that finds, for every vertex \( v \in V \), the length of the shortest path from \( v \) to \( s \).

   **Answer:** Let \( G'(V, E') \) be the graph obtained from \( G \) by reversing the direction of each edge. That is

   \[
   (u, v) \in E \iff (v, u) \in E.
   \]

   Note that in \( G \) there is a path \( u \sim s \) of weight \( w \) iff in \( G' \) there exists a path \( \pi' \) \( s \sim u \) with weight \( W \). Hence we only need to run Dijkstra on \( G' \).

5. Assume you run Dijkstra algorithm on a directed graph that contain edges with arbitrary weights (some edges with positive weights, some with negative weights). However, the graph does not contain a negative cycle. Show an example in which the resulting output of the algorithm is incorrect.
6. Let \( G(V, E) \) be an undirected graph, with positive weights given for its edges, and assume \( S_0, S_1, \ldots, S_k \subseteq V \) be subsets of \( V \). (that is, each \( S_i \) might contain several vertices). Define
\[
\delta(S_i, S_j) = \min_{x \in S_i, y \in S_j} \delta(x, y),
\]
where \( \delta(x, y) \) is the weight of the edge \((x, y)\). Suggest an algorithm that computes \( \delta(S_0, S_i) \) for every \( i \), in time \( O((m + n) \log n) \). Assume that \( S_i \cap S_j = \emptyset \) for every \( 1 \leq i < j < k \).

**Answer:** We add new vertices \( s_0, s_1, \ldots, s_k \), and for each \( i \), we connect \( s_i \) to each vertex of \( S_i \) by edges of weight 0. It is easy to show that \( \delta(S_i, S_j) = \delta(s_i, s_j) \), and could be easily determined by running Dijkstra from \( s_0 \).

Other solutions prefers to modify Dijkstra algorithm: Initially \( d[u] = 0 \) each \( u \in S_i \). Once the algorithm terminates, we compute \( \min_{v \in S_j} d[v] \) for every \( j > 0 \).

7. Let \( G(V, E) \) be an undirected graph with positive and negative weights on its edges. Suggest an algorithm that in time \( O(m \log m) \) finds the smallest value \( d \) such that between every pair of vertices \( u, v \in V \) there is a path where the weights of each edge on this path is \( \leq d \).

Note that we are not interested in the sum of weights.

8. Let \( G(V, E) \) be a directed graph, and assume that each vertex \( v_i \in V \) is assigned with a positive weight \( w_i \). Edges do not have weights. We define the vertex-cost of a path in the graph to be the sum of weights of the vertices along this path. Given vertices \( s, t \in V \), suggest an algorithm with running time \( O((m + n) \log(m + n)) \), that computes a path from \( s \) to \( t \) with minimum vertex-cost.

**Hint:** Generate a new graph \( G'(V', E') \) where \( |V'| = 2n \) and \( |E'| \leq m + n \), and run Dijkstra on this graph.

**Answer:** We create a new graph \( G'(V', E') \), defined as follows: if for every \( v_i \in V \) we generate two vertices in \( V' \) namely \( v_i', v_i'' \). To create \( E' \), we use the following rule:
\[
(u, v) \in E \Leftrightarrow (u'', v') \in E',
\]
In addition we add an edge \((v'_i, v''_i)\) for every \( i \), whose weight is \( w_i \). The weights of any other edge is 0.

Next we see that for any path \( \pi \) in \( G \) leading from \( s \) to \( v \) and passing through vertices whose weight sums to \( W \), will be equivalent path \( \pi \) in \( G' \) leading from \( s \) to \( v \) and passing through edges whose weight sums to \( W \). Hence the shortest path in \( G \) is equivalent to the shortest path in \( G' \) and vice versa.

9. Given a graph \( G(V, E) \), where each edge \((u, v)\) is associated with a weight \( w(u, v) \), which is an integer between 1 to 17. Assume \( s \) and \( t \) are vertices in \( V \). Suggest an \( O(m + n) \)-time algorithm for computing the shortest path from \( s \) to \( t \).

10. Let \( G(V, E) \) be a directed graph, \( s \in V \) is a source. You are running Dijkstra's algorithm on this graph. Define \( U = \{v \in V \mid \text{there is no path from } s \text{ to } v \text{ in } G(V, E)\} \).
Explain how you could identify the vertices in $U$, by reading the array $\pi[1 \ldots n]$ given as output of Dijkstra algorithm.