Exercise I-S

Remarks: All the graphs here are without self loops and parallel edges. We use the notation \( \delta(G) \) for the minimum degree in \( G \). In all the algorithms, always explain their correctness and analyze their complexity. The complexity should be as small as possible. A correct algorithm with large complexity, may not get full credit.

- **Question 1:** Let \( G \) be a graph and \( k \) a positive integer such that \( \delta(G) \geq k \). Prove that there is a simple path in \( G \) of length at least \( k \).

  **Solution:** Pick a vertex \( u \). Let this be a path of length 0. Each path has two end points. For each endpoint \( v_l \) and \( v_r \), see if it has a neighbor outside the path (to a vertex that is not right now in the path.) If there exist such a neighbor, extend the path by 1 as long as possible, in both directions.

  Let the number of vertices in the resulting path (after no more extensions are possible) be \( i \), let \( v_r \) be the current right end vertex of the path. Because \( \delta(G) \geq k \), \( \deg(v_r) \geq k \). Now, all the neighbors of \( v_r \) are within the path, otherwise we can extend the path. So, \( v_r \) has at least \( k \) neighbors in the path. In particular, beside \( v_r \), there are at least other \( k \) vertices in the path. Thus \( i \geq k + 1 \) and hence the length of the path (which is \( i - 1 \)) is at least \( k \).

- **Question 2:** Let \( G \) be a graph with \( n \) vertices and \( m \) edges. Show that there exist a subgraph \( G'(V', E') \) of \( G \) such that \( \delta(G') \geq m/n \). Deduce that every graph \( G \) has a simple path of length at least \( m/n \). Give an algorithm that finds such a path.

  **Solution:** If there is a vertex \( v \) with \( \deg(v) < m/n \), remove it and all its neighbors from the graph. Re-compute the degrees. Again, if there is a vertex whose (new) degree is less than \( m/n \), remove it. Note that the degrees change, but we use the original \( m \) and \( n \) all the time. Continue this way until there are no more vertices with degree smaller than \( m/n \).

  We must show only that the graph does not turn empty. This is easy: we removed no more than \( n \) vertices, and for each removed vertex, deleted strictly less than \( m/n \) edges. So we delete strictly less than \( m \) edges, so some edges remain, and the graph is thus not empty.

  Combining questions 1 and 2, we compute the subgraph with \( \delta \geq m/n \) and then find a path of length at least \( m/n \). The complexity is clearly bounded by \( O((E + V) \log V) \) (check it!).

- **Question 3:** Design an algorithm that gets \( G(V, E) \) and \( s, t \in V \), and computes the number of different shortest paths between \( s \) and \( t \).

  **Solution:** The solution uses dynamic programming. For simplicity, say that we compute the BFS distances from \( s \) to all other vertices first, and let \( D^s_i \) be the vertices of distance \( i \) from \( s \). We make a \((n + 1) \times n \) table \( M \), with the \( i \) row concerning the \( D^s_i \)
vertices. For \( j \in D_i^s \), the \( i, j \) row contains the number of shortest paths between \( s \) and \( j \). The first row is all 0 except for a 1 in the column of \( s \) (as there is a single path from \( s \) to \( s \) of zero distance, the empty path).

To fill the next row, consider a vertex \( q \in D_{i+1}^s \). Let \( q_1, q_2, \ldots \) be the vertices in \( D_i^s \) that have outgoing edges into \( q \) (\( q \) is their neighbor). Then you put in the \( M[i+1, q] \) entry of the matrix, the sum of \( M[i, q_1] + M[i, q_2] + \ldots \). This is because to get to \( q \) in a shortest path from \( s \), you must first reach \( q_1 \) or \( q_2 \), and so own.

The exact details of the algorithm are left for the reader. As described, the algorithm runs in \( O(n^2) \) steps, but can be made to run in \( O(E) \). The correctness is by a simple induction which is left for the reader.

- **Question 4:** Show an example of a graph with negative weights on which the algorithm of Dijkstra does not work.

  **Solution:** Consider the following example: \( s \) will label \( b \) by 2, and then \( b \) will label \( t \) by 1, both of these steps being a mistake.

- **Question 5:** A directed graph is called strongly connected if for every \( u, v \in V \) there exist a directed path from \( u \) to \( v \) and also a directed path from \( v \) to \( u \) (that is, its possible to get from any vertex to any other vertex). Design an algorithm that checks if a given graph is strongly connected.

  **Solution:** We first prove:

  **Lemma 0.1** A graph is strongly connected if and only if for every given vertex \( v \), there is a path for \( v \) to every other vertex \( u \), and vice-versa, a path from every other vertex \( u \) to \( v \).

  Note that in the above lemma \( v \) is fixed, namely, we do not require a path from every \( w \) to every \( u \), rather, only paths from \( v \) to the rest, and from the rest of the vertices to \( v \). In proof, if \( G \) is strongly connected, then certainly the claim holds. Now, in the other direction, suppose there is a path from \( v \) to any \( w \) and from any \( w \) to \( v \). Say that I want to reach \( z \) from \( t \). Now, go from \( t \) to \( v \), and then go from \( v \) to \( z \). Namely, go from \( t \) to \( z \) via \( v \). This completes the proof.
Now, choose an arbitrary $v$. Checking if it’s possible to reach all $V \setminus \{v\}$ from $v$ is done via a single BFS search, and then checking that $d[u] < \infty$ for all $u \in V$.

To check if it’s possible to get from all $V \setminus \{v\}$ to $v$ is done as follows: reverse the direction of the edges (i.e., compute the graph with each $u \leftarrow v$ turned into $u \rightarrow v$) and then do BFS from $v$. Clearly, this gives what’s needed. The time is twice $O(E+V)$ which is still $O(E+V)$.

- **Question 6:** Let $G$ be an undirected graph. A pair of vertices $u, v \in V$ is called a connected pair, if there is a path between $v$ and $u$ in the graph. Design an algorithm that finds the number of different connected pairs in $G$.

  **Solution:** Compute the connected components (as explained in class). In a connected component with $j$ vertices, there are $j \cdot (j - 1)/2$ connected pairs (this is $j$ choose 2).

  So, if $j_1, \ldots, j_t$ is the number of vertices in each of the $t$ connected components, the answer is $\sum_{i=1}^{t} j_i \cdot (j_i - 1)/2$. The running time is $O(E + V)$.  
