These are some review questions to test your understanding of the material. Some of these questions may appear on an exam.

Graphs

0.1 Define *graph*, *undirected graph*, *directed graph*, and *weighted graph*, *sparse graph*.

0.2 Define *path* in a graph. Define *length of a path* in a graph.

0.3 Define the following:

1. Connected, undirected graph.
2. Strongly connected directed graph.
3. Weakly connected directed graph.

0.4 Let $G = (V, E)$ be an undirected graph with V the set of vertices and E the set of edges. Let $v_1, v_2, \ldots, v_p \in V$ be the members of V and let $q = |E|$ be the cardinality of E. Prove:

$$\sum_{i=1}^{p} \text{degree}(v_i) = 2q$$

0.5 Prove that in any undirected graph, the number of vertices of odd degree is even.

0.6 Write pseudo-code for breadth-first and depth-first traversals of undirected graphs. The code must be complete and must fully describe the operations.

0.7 Describe, in English, any *adjacency table* graph implementation. How does the implementation differ for directed and undirected graphs?

0.8 Describe, in English, any *adjacency list* graph implementation. How does the implementation differ for directed and undirected graphs?

0.9 Given a drawing of a directed or of an undirected graph, show its representation as an adjacency matrix.

0.10 Draw the directed graph represented by the adjacency matrix given below. The rows/columns in the matrix correspond to the vertices with labels A,B,C,D,E. A non-zero entry at [row,col] indicates that the vertex indicated by the row label is adjacent-to the vertex indicated by the col label.

```
  A B C D E
A 0 1 0 1 0
B 1 1 1 0 0
C 0 0 0 0 1
D 0 1 0 0 1
E 0 0 0 0 0
```

0.11 Given a drawing of a directed or of an undirected graph, show its representation as an adjacency list.

0.12 Draw the directed graph represented by the adjacency list given below. This is an "adjacent-to" representation.

```
v[1] (Label = A) --> 2 --> 5

v[2] (Label = B) --> 3 --> 5

v[3] (Label = C) --> 2 --> 4 --> 5

v[4] (Label = D) --> 5

v[5] (Label = E) --> empty
```

0.13 Discuss the characteristics of the *adjacency table* and *adjacency list* implementations of graphs. Include storage requirements and asymptotic worst-case performance of the operations:

```
Note:  u and v are vertices in the graph

  Degree(u)       returns the degree of vertex u  (undirected graphs)
  InDegree(u)     returns the indegree of vertex u (directed graphs)
  OutDegree(u)    returns the outdegree of vertex u (directed graphs)
  AdjacentTo(u)   returns a list of the vertices adjacent to u
  AdjacentFrom(u) returns a list of the vertices adjacent from u
  Connected(u,v)  returns true if there is an edge between
                      vertices u and v, returns false otherwise
```

0.14 Consider the directed graph represented by the following adjacency matrix (an adjacent-to representation):

```
  |  A B C D E
--+------------
A |  0 1 1 1 1
B |  0 1 1 0 0
C |  0 0 0 1 0
D |  0 0 0 1 1
E |  0 1 0 0 0
```

List the depth-first and breadth-first traversals of the graph beginning at vertex A. Repeat for vertex B. (Whenever a new vertex is to be visited and there is more than one possibility, use the vertex labelled with the letter than comes first in the alphabet.)

0.15 Define *directed acyclic graph*.

0.16 Define *topological ordering* of a directed acyclic graph.

0.17 Given a drawing of a graph, find all cycles.

0.18 Given a drawing of a directed acyclic graph, write the labels of its vertices in topological order. Explain how you obtained the ordering.

0.19 Given a drawing of a directed graph along with the "discovery" and "finish" times of its vertices after a depth-first search, write the labels of the graph in topological order. Explain how you obtained the ordering.

0.20 Given a drawing of a directed graph along with the "discovery" and "finish" times of its vertices after a depth-first search, identify the type of each edge (tree, back, forward, or cross). The following test is relevant, with `d[v]` the discovery time of vertex `v` and `f[v]` its finish time:

```
if ( (d[v1] < d[v2])  && (f[v1] > f[v2]) )
   (v1,v2) is a tree edge
else if (d[v1] > d[v2]  && f[v1] < f[v2])
   (v1,v2) is a back edge
else if (d[v1] > d[v2] && f[v1] > f[v2])
   (v1,v2) is a cross edge
else //  d[v1] < d[v2] - 1  && f[v1] > f[v2]
   (v1,v2) is a forward edge
```