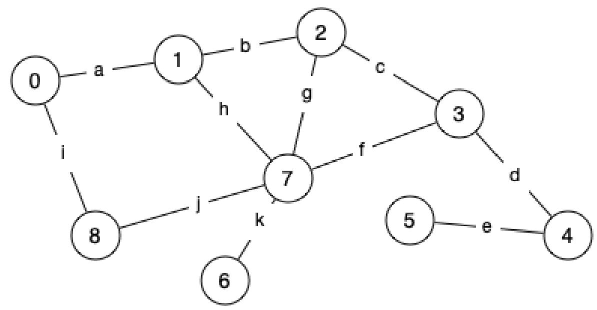
Week 05 Tutorial

Graphs (i)

[Show with no answers] [Show with all answers]

1. Consider the following graph



The edges are labelled simply for convenience in describing graph properties.

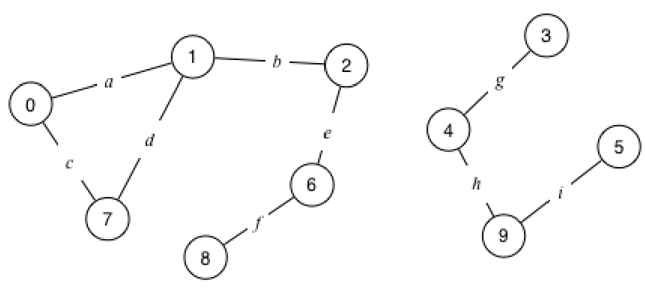
a. How many edges does it have?
b. How many *cycles* are there in the graph?
c. How many *cliques* are there in the graph?
d. What is the *degree* of each vertex?
e. How many edges in the *longest* path from 5 to 8?
(without traversing any edge more than once)

[show answer]

2. What is the difference between a *connected* graph and a *complete* graph? Give simple examples of each.

[show answer]

3. Consider the following graph with multiple components:



And a vertex-indexed connected components array that might form part of the `Graph` representation structure for this graph:

```
cc[] = {0, 0, 0, 1, 1, 1, 0, 0, 0, 1}
```

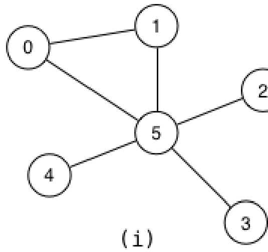
a. show how the `cc[]` array would change if edge d was removed

b. show how the `cc[]` array would change if edge b was removed

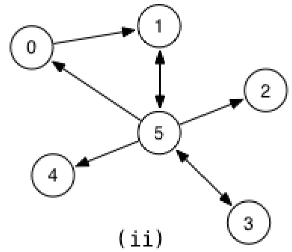
c. show how the `cc[]` array would change if an edge was added between 2 and 4

[show answer]

4. For each of the following graphs:



(i)



(ii)

Show the concrete data structures if the graph was implemented via:

a. adjacency matrix representation (assume full $V \times V$ matrix)
b. adjacency list representation (if non-directional, include both (v, w) and (w, v))

[show answer]

5. Consider the following map of streets in the Sydney CBD:

Represent this as a directed graph, where intersections are vertices and the connecting streets are edges. Ensure that the directions on the edges correctly reflect any one-way streets (this is a driving map, not a walking map). You only need to make a graph which includes the intersections marked with red letters Some things that don't show on the map: Castlereagh St is one-way heading south and Hunter St is one-way heading west.

For each of the following pairs of intersections, indicate whether there is a path from the first to the second. If there is a path, enumerate it as a set of vertices. If there is more than one path, show two different paths.

a. from intersection D on Margaret St to intersection L on Pitt St

b. from intersection J to the corner of Margaret St and York St (intersection A)

c. from the intersection of Margaret St and York St (A) to the intersection of Hunter St and Castlereagh St (M)

d. from intersection M on Castlereagh St to intersection H on York St

[show answer]

6. Consider the following simple algorithm for web crawling:

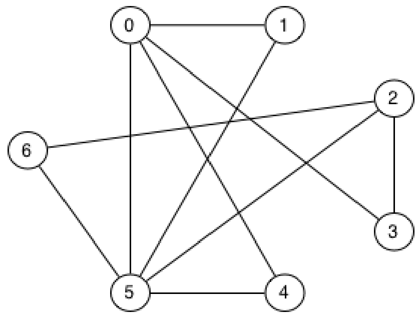
```
crawl(startURL)
{
    add startURL to Queue
    while (Queue is not empty) {
        nextURL = remove head of Queue
        page = open(nextURL)
        while ((url = scanForHyperlinks(page)) != NULL) {
            if (alreadySeen(url)) continue
            add url to Queue
        }
```

```
    }
}
```

Suggest how you might implement the `alreadySeen()` test.

[show answer]

7. Show what would be printed by the iterative depth-first and breadth-first traversals in the functions below on the following graph:



When choosing which neighbour to visit next, always choose the smallest unvisited neighbour. At each step, show the state of the `Stack` (DFS) or the `Queue` (BFS).

```
void breadthFirst (Graph g, Vertex v)
{
    int *visited = calloc (g->nV, sizeof (int));
    Queue q = newQueue ();
    QueueJoin (q, v);
    while (QueueLength (q) > 0) {
        Vertex x = QueueLeave (q);
        if (visited[x])
            continue;
        visited[x] = 1;
        printf ("%d\n", x);
        for (Vertex y = 0; y < g->nV; y++) {
            if (!g->edges[x][y])
                continue;
            if (!visited[y])
                QueueJoin (q, y);
        }
    }
}
```

```
void depthFirst (Graph g, Vertex v)
{
    int *visited = calloc (g->nV, sizeof (
    Stack s = newStack ();
    StackPush (s, v);
    while (!StackIsEmpty (s)) {
        Vertex x = StackPop (s);
        if (visited[x])
            continue;
        visited[x] = 1;
        printf ("%d\n", x);
        for (Vertex y = g->nV - 1; y >= 0;
            if (!g->edges[x][y])
                continue;
            if (!visited[y])
                StackPush (s, y);
        }
    }
}
```

Show the results for each of the following function calls:

```
depthFirst (g, 0);
depthFirst (g, 3);
breadthFirst (g, 0);
breadthFirst (g, 3);
```

[show answer]

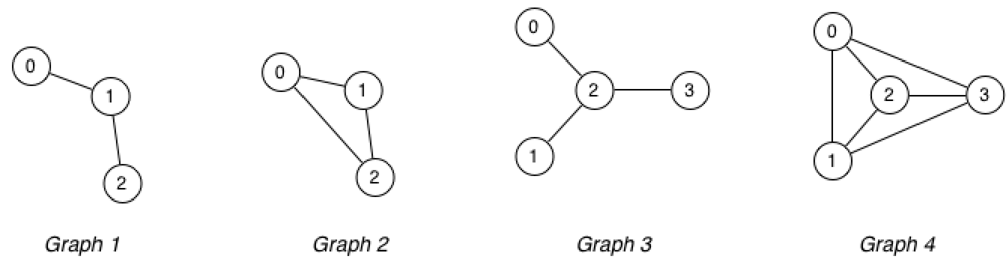
8. Write a C function that takes a `Graph` and a starting `Vertex` and returns a set containing all of the vertices that can be reached by following a path from the starting point. Use the function template:

```
Set reachable (Graph g, Vertex v) { ... }
```

You may use any of the ADTs mentioned at the start of the tute questions.

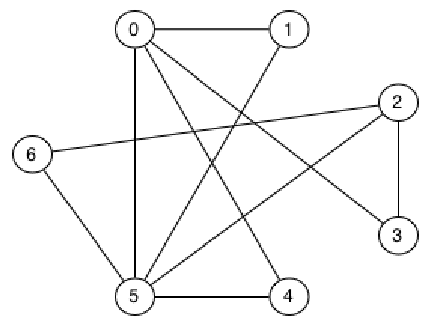
[show answer]

9. What is the difference between a Euler path/tour and a Hamilton path/tour? Identify any Euler/Hamilton paths/tours in the following graphs:



[show answer]

10. Find a Hamilton path and a Hamilton tour (if they exist) in the following graph:



If there is no Hamilton path or tour for the above graph, modify the edges so that such a path/tour exists.

[show answer]

11. Write a function to check whether a path, supplied as an array of `Edges` is an Euler path. Assume the function has interface:

```
bool isEulerPath (Graph g, Edge e[], int nE)
```

where `e[]` is an array of `nE` edges, in path order.

[show answer]

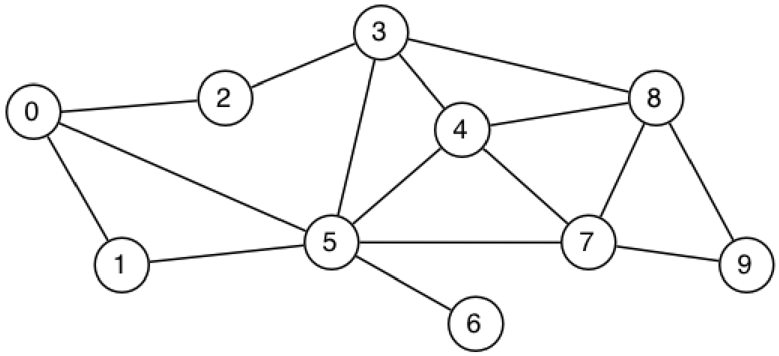
12. Using the following algorithm for finding a Hamiltonian path

```
hasHamiltonianPath(G,src,dest):
|  input:  graph G, plus src/dest vertices
|  output: true if Hamiltonian path src...dest, false otherwise
|
|  for all vertices v ∈ G do
|     visited[v]=false
|  end for
|  return hamiltonR(G,src,dest,#vertices(G)-1)

hamiltonR(G,v,dest,d)
|  input: G    graph
|         v    current vertex considered
|         dest destination vertex
|         d    distance "remaining" until path found
|
|  if v=dest then
|  |  if d=0 then return true else return false
|  else
|  |  visited[v]=true
|  |  for each (v,w) ∈ edges(G)  where not visited[w] do
|  |  |  if hamiltonR(G,w,dest,d-1) then
|  |  |  |  return true
|  |  |  end if
|  |  end for
|  end if
```

```
|  visited[v]=false
|  return false
```

trace the execution of the algorithm when searching for a Hamiltonian path from 1 to 6 on the following graph:



Consider neighbours in ascending order

[show answer]