

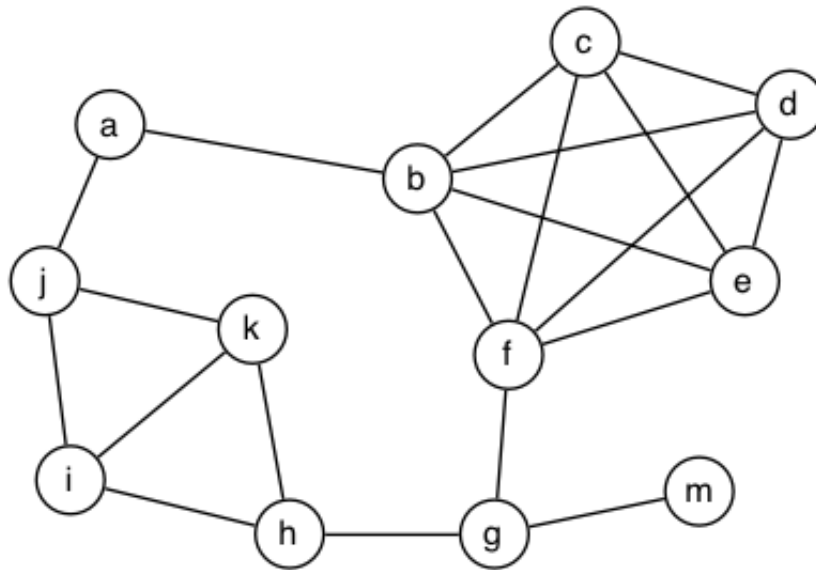
Week 06 Problem Set

Graph Data Structures and Search

[\[Show with no answers\]](#) [\[Show with all answers\]](#)

1. (Graph fundamentals)

For the graph



give examples of the smallest (but not of size/length 0) and largest of each of the following:

- path
- cycle
- spanning tree
- vertex degree
- clique

[\[show answer\]](#)

2. (Graph properties)

- Write pseudocode for computing
 - the minimum and maximum vertex degree
 - all 3-cliques ("triangles")

of a graph g with n vertices.

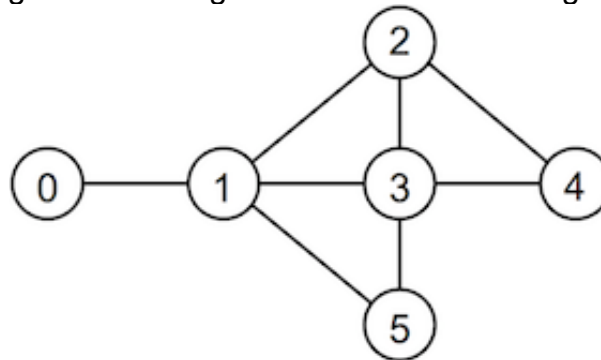
Your methods should be representation-independent; the only function you should use is to check if two vertices $v, w \in \{0, \dots, n-1\}$ are adjacent in g .

- b. Determine the asymptotic complexity of your two algorithms. Assume that the adjacency check is performed in constant time, $O(1)$.
- c. Implement your algorithms in a program `graphAnalyser.c` that
 1. builds a graph from user input:
 - first, the user is prompted for the number of vertices
 - then, the user is repeatedly asked to input an edge by entering a "from" vertex followed by a "to" vertex
 - until any non-numeric character(s) are entered
 2. computes and outputs the minimum and maximum degree of vertices in the graph
 3. prints all vertices of minimum degree in ascending order, followed by all vertices of maximum degree in ascending order
 4. displays all 3-cliques of the graph in ascending order.

Your program should use the Graph ADT from the lecture ([Graph.h](#) and [Graph.c](#)). These files should not be changed.

Hint: You may assume that the graph has a maximum of 1000 nodes.

An example of the program executing is shown below for the graph



```

prompt$ ./graphAnalyser
Enter the number of vertices: 6
Enter an edge (from): 0
Enter an edge (to): 1
Enter an edge (from): 1
Enter an edge (to): 2
Enter an edge (from): 4
Enter an edge (to): 2
Enter an edge (from): 1
Enter an edge (to): 3
Enter an edge (from): 3
Enter an edge (to): 4
Enter an edge (from): 1
Enter an edge (to): 5
Enter an edge (from): 5
Enter an edge (to): 3
Enter an edge (from): 2
Enter an edge (to): 3
Enter an edge (from): done
Finished.
  
```

```

Minimum degree: 1
Maximum degree: 4
Nodes of minimum degree:
0
Nodes of maximum degree:
1
3
Triangles:
1-2-3
1-3-5
2-3-4

```

Note that any non-numeric data can be used to 'finish' the interaction.

We have created a script that can automatically test your program. To run this test you can execute the dryrun program that corresponds to the problem set and week. It expects to find a program named `graphAnalyser.c` in the current directory. You can use dryrun as follows:

```
prompt$ -cs9024/bin/dryrun prob06
```

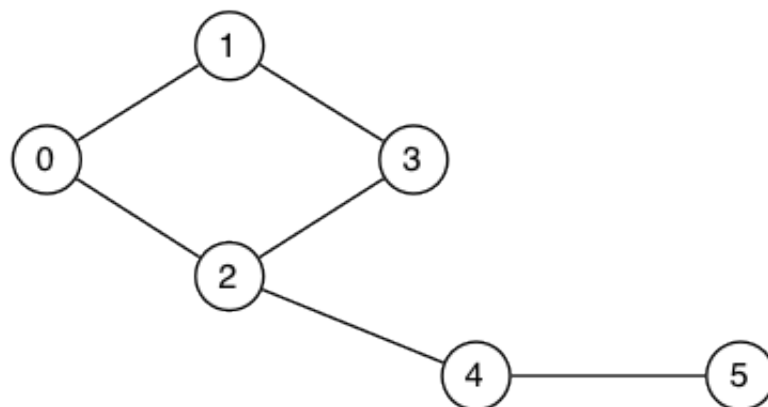
Please ensure that your program output follows exactly the format shown in the sample interaction above. In particular, the vertices of minimum and maximum degree and the 3-cliques should be printed in ascending order.

[\[show answer\]](#)

3. (Graph representations)

Show how the following graph would be represented by

- an adjacency matrix representation ($V \times V$ matrix with each edge represented twice)
- an adjacency list representation (where each edge appears in two lists, one for v and one for w)



[\[show answer\]](#)

4. (Storage costs)

- Consider the adjacency matrix and adjacency list representations for graphs. Analyse

the storage costs for the two representations in more detail in terms of the number of vertices V and the number of edges E . Determine roughly the $V:E$ ratio at which it is more storage efficient to use an adjacency matrix representation vs the adjacency list representation.

For the purposes of the analysis, ignore the cost of storing the GraphRep structure. Assume that: each pointer is 4 bytes long, a Vertex value is 4 bytes, a linked-list node is 8 bytes long and that the adjacency matrix is a complete $V \times V$ matrix. Assume also that each adjacency matrix element is **1 byte** long. (*Hint: Defining the matrix elements as 1-byte boolean values rather than 4-byte integers is a simple way to improve the space usage for the adjacency matrix representation.*)

- b. The standard adjacency matrix representation for a graph uses a full $V \times V$ matrix and stores each edge twice (at $[v,w]$ and $[w,v]$). This consumes a lot of space, and wastes a lot of space when the graph is sparse. One way to use less space is to store just the upper (or lower) triangular part of the matrix, as shown in the diagram below:

0	a		
	0	b	
		0	c
			0

Full matrix representation

a	b	c	d	e
---	---	---	---	---

Upper triangular representation

The $V \times V$ matrix has been replaced by a single 1-dimensional array `g.edges []` containing just the "useful" parts of the matrix.

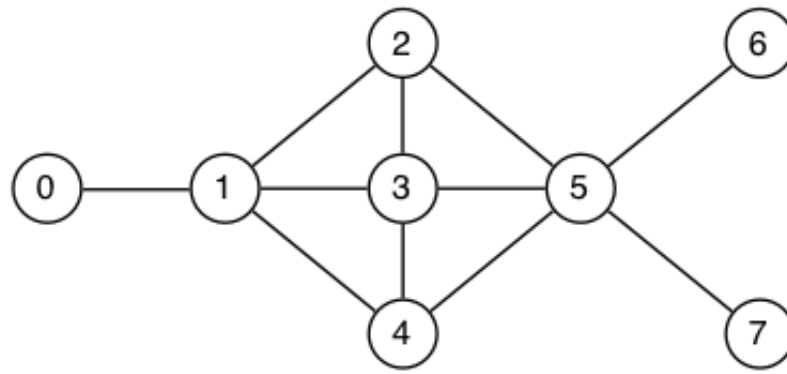
Accessing the elements is no longer as simple as `g.edges [v] [w]`. Write pseudocode for a method to check whether two vertices v and w are adjacent under the upper-triangle matrix representation of a graph g .

[\[show answer\]](#)

5. (Graph traversal: DFS and BFS)

Show the order in which the nodes of the graph depicted below are visited by

- DFS starting at node 0
- DFS starting at node 3
- BFS starting at node 0
- BFS starting at node 3



Assume the use of a stack for depth-first search (DFS) and a queue for breadth-first search (BFS), respectively. Show the state of the stack or queue explicitly in each step. When choosing which neighbour to visit next, always choose the smallest unvisited neighbour.

[\[show answer\]](#)

6. Challenge Exercise

Extend your program from exercise 2 to compute the *largest* size of a clique in a graph. For example, if the input happens to be the complete graph K_5 but with any one edge missing, then the output should be 4.

Hint: Computing the maximum size of a clique in a graph is known to be an *NP-hard problem*. Try a generate-and-test strategy.

[\[show answer\]](#)