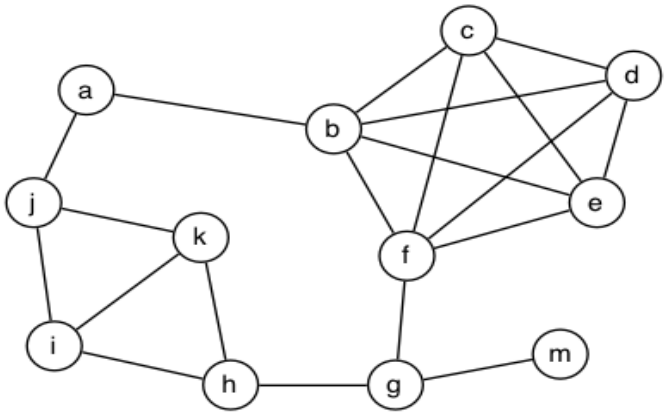
Problem Set 7

Problem 1. For the following graph,

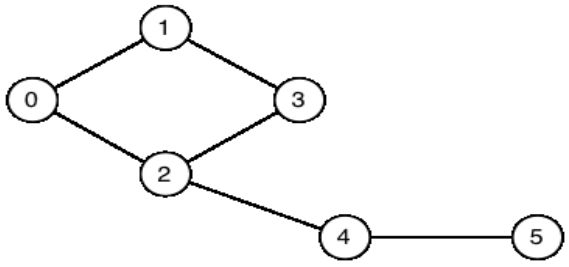


give examples of the smallest (but not of size/length 0) and largest of each of the following:

1. path
2. cycle
3. spanning tree
4. vertex degree
5. clique

Problem 2. Show how the following graph would be represented by

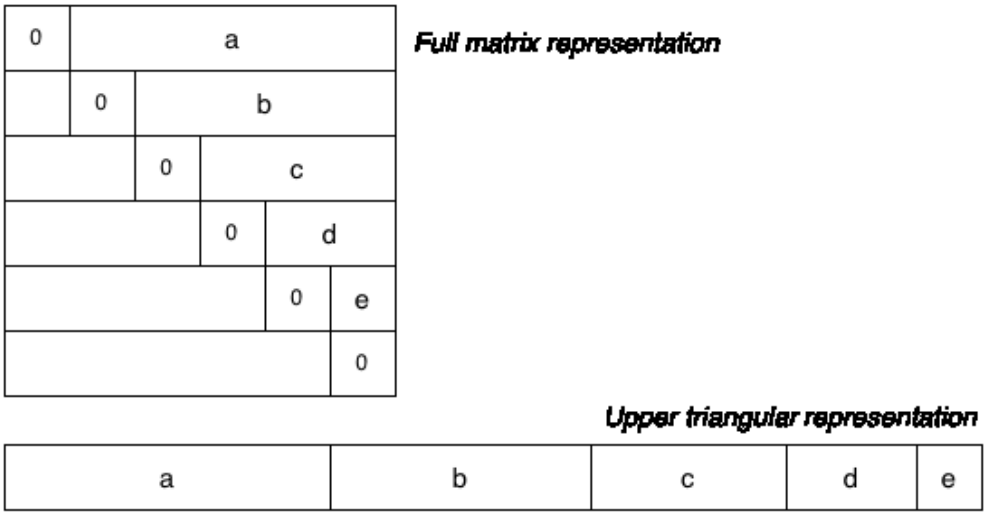
1. an adjacency matrix representation (V×V matrix with each edge represented twice)
2. an adjacency list representation (where each edge appears in two lists, one for v and one for w)



Problem 3. 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×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.)

Problem 4. The standard adjacency matrix representation for a graph uses a full n×n 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:



The n×n 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.

Problem 5. Write an algorithm in pseudocode for computing the minimum and maximum vertex degree. Your algorithm should be representation-independent; the only function you should use is to check if two vertices v,w∈{0,⋯n-1} are adjacent in g. Determine the time complexity of your algorithm. Assume that the adjacency check takes constant time, O(1).

Problem 6. Write an algorithm in pseudocode for computing all 3-cliques ("triangles") of a graph g with n vertices. Your algorithm should be representation-independent; the only function you should use is to check if two vertices v,w∈{0,⋯n-1} are adjacent in g. Determine the time complexity of your algorithm. Assume that the adjacency check takes constant time, O(1).