

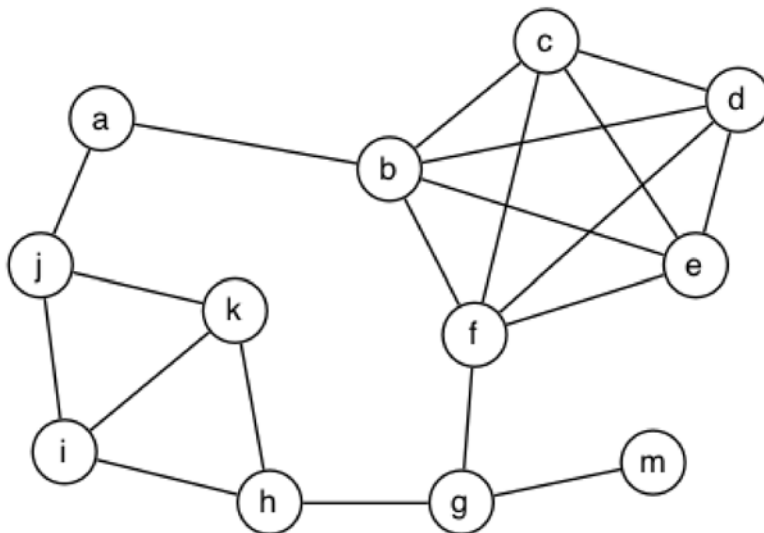
## 目录

1. Week 8 Exercises
  1. GraphFundamentals.txt
  2. GraphRepresentations.txt
  3. GraphStorageCosts.txt
  4. DepthAndBreadthFirst.txt
  5. VisitedArray.txt
  6. Palindrome.c

## Week 8 Exercises

### GraphFundamentals.txt

For the graph:

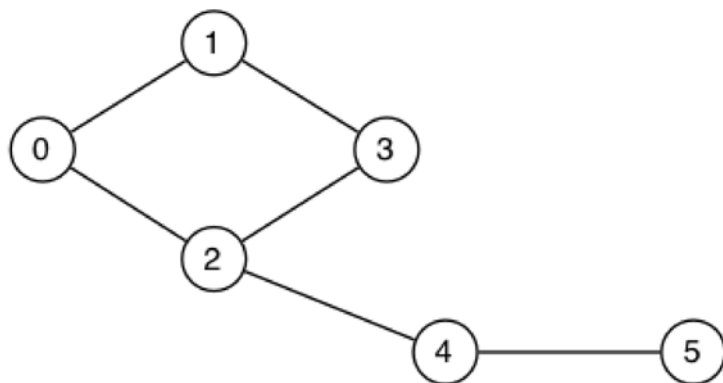


give examples of the **1.** smallest (non-zero) and the **2.** largest of each of the following:

- a. path
- b. cycle
- c. spanning tree
- d. vertex degree
- e. clique

### GraphRepresentations.txt

For the graph:



show how it 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  $w$ )

## GraphStorageCosts.txt

For the purposes of this exercise you may assume:

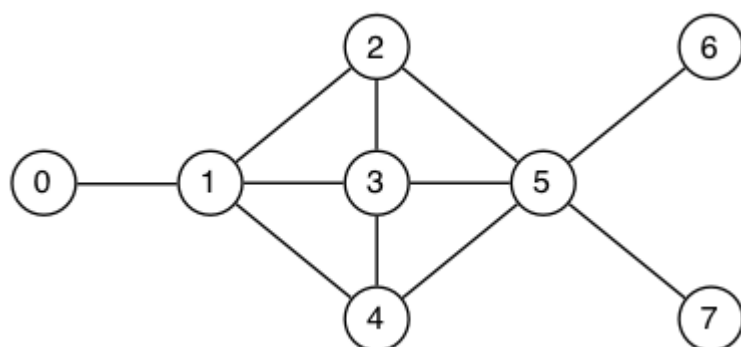
- a pointer is 8 bytes long, an integer is 4 bytes
- a vertex is an integer
- a linked list node stores an integer
- an adjacency matrix element is an integer

For a given graph:

- Analyse the precise storage cost for the adjacency matrix (AM) representation of a graph of  $V$  vertices and  $E$  edges.
- Do the same analysis for the adjacency list (AL) representation.
- Determine the approximate  $V:E$  ratio at which point it is more storage efficient to use an AM representation than an AL representation.
  - If a graph has 100 vertices, how many edges should it have for AM to be more storage efficient than AL?
- You can save space by making the adjacency matrix an array of bytes instead of integers. What difference does that make to the ratio?

## DepthAndBreadthFirst.txt

Consider the following graph:



Show the contents of the stack or queue in a 'search' traversal of the graph using:

- depth-first search* starting at vertex 0
- depth-first search* starting at vertex 3
- breadth-first search* starting at vertex 0
- breadth-first search* starting at vertex 3

## VisitedArray.txt

Consider the following 2 graphs (the right graph is missing edge 7-8):

0--1--2	0--1--2
3--4--5	3--4--5
6--7--8	6--7 8

Starting at vertex **0**, and using a *depth-first search* (assume the smallest vertex is selected):

- compute the visited array for the graph on the left
  - can you deduce a path from vertex **0** to vertex **8** from the visited array?
- compute the visited array for the graph on the right
  - can you deduce a path from vertex **0** to vertex **8** from the visited array?

Can you draw any conclusions?

## Palindrome.c

To finish off, some 'easy' programming and a complexity analysis.

- Write an algorithm in pseudo code to determine if an input character array of length  $n$  is a palindrome. A palindrome is a word that reads the same forward and backward. For example, "racecar" is a palindrome.
- What is the complexity of the algorithm. Justify your answer.
- Implement your algorithm in C. Your program should accept a single command line argument and check whether it is a palindrome. If there are less or more arguments the program simply returns. Examples of the program executing are:

```
prompt$ ./palindrome racecar
yes
prompt$ ./palindrome cat
no
prompt$ ./palindrome
prompt$
prompt$ ./palindrome cats dogs
prompt$
```

*Hint: use strlen().*