## 7.1 Graph definitions:

**Multigraph -** may have multiple edges. When m different edges connect the vertices u and v, we say that the edge {u,v} is an edge of multiplicity m

**Pseudograph** – has both multiple edges and a loop

**Order –** number of its vertices

**Size –** number of edges

**Adjacent or neighbour** – two vertices are called adjacent if there is an edge between them.

Such an edge is called **incident** with those vertices. Or **connects** those vertices

**Degree** ( indegree and out degree)

**Sparse graph** – generally more efficient to store edges by recording between which vertices they occur

**Dense graph** - generally more efficient to store edges by listing vertex pairs and recording between which pairs there are edges

**Source** - vertex with in-degree 0

**Sink –** vertex with out-degree 0

**Walk** – sequence of edges, and the same vertex can be passed twice

**Path** – a walk which does contain a same vertex more than once.

**Cycle** – a walk begins and ends at the same vertex

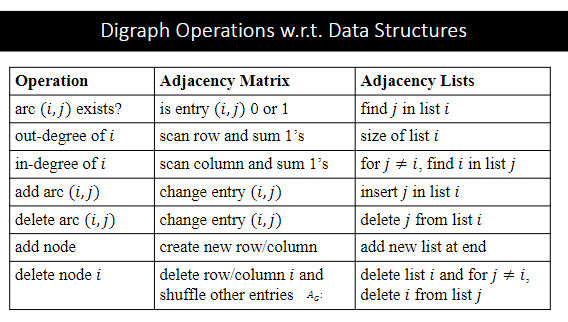
## 7.2 Graph data structures.

### Graph problems:

* List all nodes in order of their distance to a fixed node v
* What is the nearest node to v with some property?
* Is there a path from u to v?
* What is the shortest path between u and v?
* How many different paths are there between u and v?

### Adjacency Matrix vs Adjacency List

Storage space required for an adjacency **matrix** is , use for small dense graphs which we want to test for the existence of arcs, find the in degree of vertices/ delete arcs.

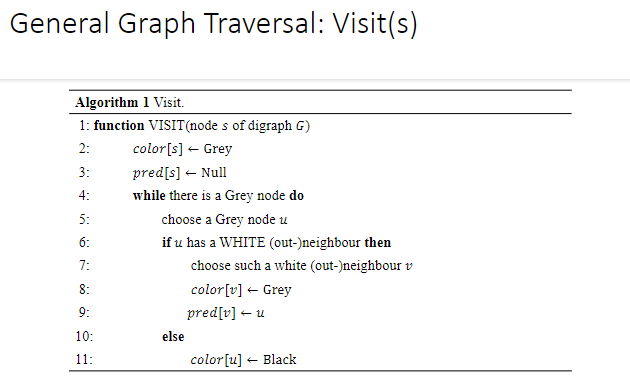
****Storage space required for an adjacency **list** is also however, usually graphs are sparse and space requirement is only . Use for large, sparse graphs, where we need to comput out degree and add/delete vertices.

### Adjacency list/matrix operation time complexity

A table with numbers and letters

Description automatically generated

## 7.3 General graph traversal



A screenshot of a computer program

Description automatically generated

### Runtime analysis

* The initialisation of the array colour takes time so traverse is where t is the total time taken by all the calls to visit.
* We must execute the while-loop of visit in total times since every node must eventually move from white through to grey to black.
  + Time taken to choose grey nodes is .
  + Time taken to find white neighbour involves checking each neighbour of u:
    - Adjacency matrix, we need to scan the whole row,
    - Adjacency list,
* Runtime of traverse:
  + Adjacency list =
  + Adjacency matrix =

### Traversal arc classifications:

Suppose (u,v) is an arc in E. Digraph is G, and search forest is F

* **Tree arc :** arc belongs to one of the trees of F.
* **Forward arc :** u is an ancestor of v in F
* **Back arc** : u is a descendant of v in F
* **Cross arc :** u nor v is an ancestor of the other in F.

## 8.1 DFS Graph traversal.

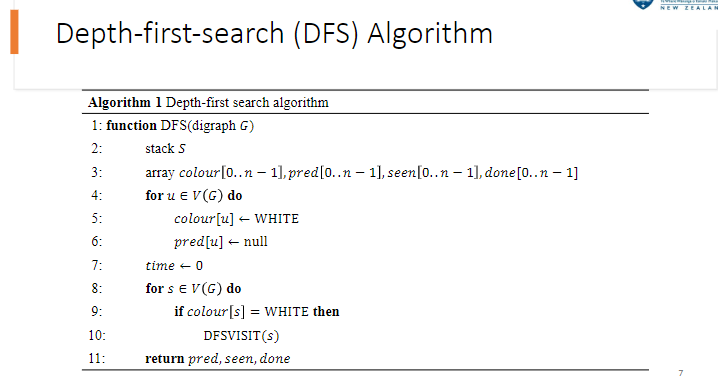
The traversal strategy chosen, will determine the order that grey frontier nodes are chosen.

DFS – Stack. LIFO. The next grey vertex to pick is the youngest remaining grey vertex

BFS – queue

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

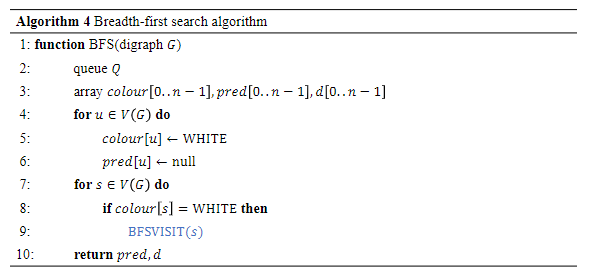
Description automatically generatedPFS – priority queue

### Basic properties of DFS

Each call to DFSVIST(v) terminates only when all nodes reachable from v via a path of white nodes have been seen

## 8.2 BFS

A screenshot of a computer program

Description automatically generatedThe next grey vertex to pick is the oldest remaining grey vertex

### BFS property: There are no forward arcs when performing BFS on a digraph

Proof:

* Suppose there exist a forward arc. Then we have the following in the search forest.
* There is a directed path of tree arcs(at least with two arcs) from 𝑢 to 𝑣, and there is  
  an arc (𝑢,𝑣) that is not in any tree of the search forest.
* But then BFS explores 𝑣 as a neighbour of 𝑢 when 𝑣 is white. So (𝑢,𝑣) is a tree arc, a  
  contradiction

### Time complexity DFS vs BFS

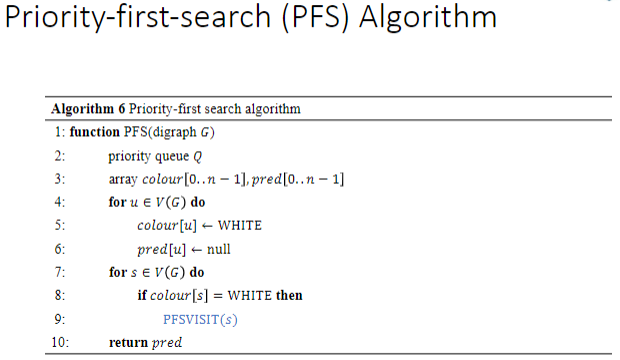
Both BFS and DFS have same complexity

* the next grey/white node has constant complexity
* With adjacency matrix: Need to find out-neighbours for n vertices. With Θ(𝑛)  
  per vertex, this means Θ 𝑛 V .
* With adjacency list: the sequence for each node is only visited once and we  
  need to visit all the edges. Thus, the complexity is Θ(𝑛 + 𝑒) .

### PFS

In PFS, the next grey vertex is selected according to a priority value

* Typically, this is an integer, such that the grey vertex with the lowest value is  
  selected first
* Priority value is assigned (often computed) no later than when the vertex turns  
  grey



A screenshot of a computer program

Description automatically generated

In simple PFS, the priority value of a vertex does not change

In advanced PFS, the priority value of a vertex may be updated again later

BFS and DFS are special cases of simple PFS

* + In BFS the priority values are the order in which the vertices turn grey
  + In DFS the priority values are the negative order in which the vertices turn grey

### PFS time complexity

Not very good as we need to find the minimum priority value for each time we need to select eh next grey node. If we are search up to n vertices, we need

This can be improved using priority queue, however still slower that pure BFS or DFS

## 8.3 Cycles girth