Graphs

Adithya Nair

July 26, 2024

Contents

1 Storing Graph Data **1**

2 Adjacency Lists **2**

3 The method of traversal(visit all nodes). **2**
3.1 Breadth-First Search 2

4 Depth-First Search **3**

Graphs are nodes with links to each other. These links are defined by a matrix known as an **adjacency matrix**.

The best way to store such links is (under the assumption that the graphs in question are **bi-directional**) to store only the upper triangle of the adjacency matrix, while ignoring the diagonal.

1 Storing Graph Data

```
import java.util.*;
int c = 5;
int a[][] = new int[c][c];
int upperTriangleSize = (c*(c-1))/2;
int graph_data[] = new int[upperTriangleSize];
int k = 0;
for(int i = 0; i < c; i++){
for(int j = i+1;j < c;j++){
graph_data[k] =  a[i][j];
k++;
}
}
```

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

2 Adjacency Lists

This consists of a one-dimensional array, specifically of pointers. For an example graph,

$$\begin{aligned} & a \to b \to c \to d \\ & \downarrow \\ & b \to a \to c \\ & \downarrow \\ & c \to a \to b \\ & \downarrow \\ & d \to a \to e \\ & \downarrow \\ & e \to d \end{aligned}$$

We have a linked list of linked lists. The problem here is that links are represented twice.

This is an optimization for when graphs have very few links.

3 The method of traversal(visit all nodes).

There are two algorithms of traversal:

1. Breadth-First Search - which involves the usage of queue
2. Depth-First Search - which involves the usage of stack

3.1 Breadth-First Search

Breadth-first involves selecting a root node, and choosing a neighbour node.

Revising queues, they have a First-In, First Out approach, insertion at the rear and deletion at the front. The procedure follows:

- The queue starts by inserting any root node, say for example A.
- Now A is dequeued and stored in the result.
- Queue all neighbouring nodes of A.
- The first node is dequeued.
- Upon dequeueing, all neighbouring nodes are checked if it's in the queue. If not, they are queued in.
  - Let's say B is taken, and B is connected to C, and C is already connected to A and B.
  - A and C will not be queued.
- This iterates until there are no more nodes.

Clearly, this depends on the number of edges in the graph. The maximum number of edges in a graph is ${}^{n}C_{2}$.

The functions required in this queue are:

- 'Insert()'
- 'Delete()'
- 'Traverse()'

This will be covered in the next lab. NOTE - This will not work for nodes which are not connected to any node in the graph.

4 Depth-First Search

This involves traversing the nodes until an edge node is met, using a stack.

The procedure follows:

- A node is selected and pushed into the stack.
- A random node connected to A is pushed.
- A connected node is pushed.
- This procedure is met until no unvisited node exists
  - In which case the stack is full.

  - If the stack is still not full, it implies that there are nodes which haven't been visited.

- Now popping occurs
- The algorithm goes back and checks for unvisited nodes by popping the stack and checking the topmost elements's adjacent nodes for unvisited nodes.
- This is done until the stack is empty

```
class DFS {
  DFS(int V)  {
    adj = new LinkedList[V];
    visited = new boolean[V];
    for (int i = 0; i < V; i++)
      adj[i] = new LinkedList<Integer>();
  }
}
```