DSA

Adithya Nair

July 29, 2024

Contents

1 Graphs

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.

1.1 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++;
}
}
```

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

1.2 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.

1.3 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

1.3.1 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 ${}^nC_2$.

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.

1.3.2 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>();
  }
}
```

2 Trees

A tree with at most two links from one node is known as a binary tree.

2.1 Properties

- Maximum number of nodes - 2^i
- Height is longest distance between root to leaf nodes (count the edges)
- Maximum number of nodes possible till height h is $(2^0+2^1+2^2+\cdots 2^h)$ $= 2^{h+1}-1$

2.2 Types Of Binary Trees

2.2.1 Full/Proper/Strict Binary Tree

It is a tree where each node must contain 2 children except the leaf node.

- There are either two children or no children.
- The number of leaf nodes is equal to number of internal nodes + 1
- The minimum number of nodes is equal to $2^h + 1$
- Maximum number of nodes is the same as number of nodes in binary tree, $2^{h+1} - 1$
- The minimum height of the full binary tree is $log_2(n + 1) - 1$

2.2.2 Complete Binary Tree

A binary tree where all the nodes are completely filled except the last level.

2.2.3 Degenerate Binary Tree

Exactly like a linked list.

2.2.4 Balanced Binary Tree

- The heights of left and right trees can have a maximum height difference of 1.

2.3 Binary Search Tree

Value of left node must be smaller than parent, and value of right node must be greater than the parent node.

2.3.1 Binary Search Tree Insertion

1. Deleting Nodes
    - It works the same as a linked list operation.
    - We use 2b to figure out the node's value, it takes the value of its in order successor.
2. Binary Search Tree Traversal. There are three types of traversal.

(a) Pre-order Traversal.

- Process the root
- Process left node subtree.
- Process right node subtree.
- Recursively repeat.

(b) In order Traversal

- Process the left subtree
- Process the root
- Process the right subtree
- Recursively repeat. **We use in order successors to figure out the value to copy.**

(c) Post order Traversal

- Process the left subtree
- Process the right subtree
- Process the root
- Recursively repeat.