1. LinkedLIst Floyd’s Finding Algorithm

class LinkedList {

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public static boolean hasCycle(Node head) {

Node slow = head;

Node fast = head;

while (fast != null && fast.next != null) {

slow = slow.next; // Move by 1 step

fast = fast.next.next; // Move by 2 steps

if (slow == fast) {

return true; // Cycle detected

}

}

return false; // No cycle

}

public static void main(String[] args) {

Node head = new Node(1);

head.next = new Node(2);

head.next.next = new Node(3);

head.next.next.next = new Node(4);

head.next.next.next.next = new Node(5);

head.next.next.next.next.next = head.next;

if (hasCycle(head)) {

System.out.println("Cycle detected in the LinkedList.");

} else {

System.out.println("No cycle in the LinkedList.");

}

}

}

1. BFS Algorithm

import java.util.\*;

public class GraphBFS {

private int vertices

private LinkedList<Integer>[] adj;

public GraphBFS(int v) {

vertices = v;

adj = new LinkedList[v];

for (int i = 0; i < v; i++) {

adj[i] = new LinkedList<>();

}

}

public void addEdge(int u, int v) {

adj[u].add(v);

}

public void BFS(int start) {

boolean[] visited = new boolean[vertices];

Queue<Integer> queue = new LinkedList<>();

visited[start] = true;

queue.add(start);

System.out.println("BFS Traversal starting from vertex " + start + ":");

while (!queue.isEmpty()) {

int current = queue.poll();

System.out.print(current + " ");

for (int neighbor : adj[current]) {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.add(neighbor);

}

}

}

}

public static void main(String[] args) {

GraphBFS graph = new GraphBFS(6);

graph.addEdge(0, 1);

graph.addEdge(0, 2);

graph.addEdge(1, 3);

graph.addEdge(1, 4);

graph.addEdge(2, 4);

graph.addEdge(3, 5);

graph.addEdge(4, 5)

graph.BFS(0);

}

}

1. TSP Shortest path problem

public class SimpleTSP {

static int[][] distance = {

{ 0, 10, 15, 20 },

{ 10, 0, 35, 25 },

{ 15, 35, 0, 30 },

{ 20, 25, 30, 0 }

};

static boolean[] visited = new boolean[4];

static int minCost = Integer.MAX\_VALUE;

public static void tsp(int currentCity, int count, int cost, int startCity) {

if (count == 4 && distance[currentCity][startCity] > 0) {

cost += distance[currentCity][startCity];

minCost = Math.min(minCost, cost);

return;

}

for (int i = 0; i < 4; i++) {

if (!visited[i] && distance[currentCity][i] > 0) {

visited[i] = true;

tsp(i, count + 1, cost + distance[currentCity][i], startCity);

visited[i] = false;

}

}

}

public static void main(String[] args) {

int startCity = 0;

visited[startCity] = true;

tsp(startCity, 1, 0, startCity);

System.out.println("Minimum tour cost: " + minCost);

}

}