```java
//Problem Set 9 Problem 4.c
//programmed in Java
//using adjacency list as a linked list
public void BFS(Graph G, int s, int t)
{
    int nodes = G.vertices;//number of nodes in the graph
    boolean state[] = new boolean[nodes];//state array keeps track of visited nodes
    LinkedList<Integer> queue = new LinkedList<Integer>();
    int d = 0;
    int k = 0;
    Tuple[] result = new Tuple[d, k];//assuming Tuple object was created since Java does
                                    // not inherently support multiple return values.
    for(int i = 0; i < nodes; i++) //initialization
    {
        state[i] = false;
    }
    state[s]=true;
    queue.add(s);//Set s as visited and added to queue
    while (queue.size() != 0)
    {
        s = queue.poll();//pop off queue
        k++;

        Iterator<Integer> i = adjList[s].listIterator();
        while (i.hasNext())
        {
            d++;
            int n = i.next();//getting adjacent nodes of current node s
            if (!state[n])
            {
                state[n] = true;
                queue.add(n);
            }
            if(n == t)
            {
                return result;
            }
        }
    }
    d = -1;//if there is no path return d as -1
    return result;
}


//Problem 4.d
public int BidirectionalBFS(Graph G, int s, int t)
{
    int nodes = G.vertices;//number of nodes in the graph
    boolean sState[nodes], tState[nodes];
    int sParent[nodes], tParent[nodes];
    Tuple[] result = new Tuple[d, k];
    LinkedList<Integer> sQueue = new LinkedList<Integer>();
    LinkedList<Integer> tQueue = new LinkedList<Integer>();

    for(int i = 0; i < nodes; i++) //initialization
    {
        sState[i] = false;
        tState[i] = false;
    }

    sQueue.push_back(s);
    sState[s] = true;
    tQueue.push_back(t);
    tState[t] = true;

    while (!s_queue.empty() && !t_queue.empty())
    {
        result += BFS(G, s, t);
        result += BFS(G, t, s);
        for(int i = 0;i < nodes;i++)
        {
            if(sState[i] && tState[i])
                return result;
        }
        d = -1;
        return result;
    }
}
```