GRAPHS AND TREES

BY MNNIT COMPUTER CODING CLUB

INTRODUCTION TO GRAPHS

WHAT ARE GRAPHS

Graphs are non-linear data structures that are used to represent the relationships between various entities.

REAL-LIFE EXAMPLES OF GRAPHS.

GEOGRAPHY OF A COUNTRY

A country has various cities connected by roads. This type of information can be represented by graphs.

NETWORK TOPOLOGY

The information like how computers are connected in a network, and how should the data be transferred between two computers can be represented by graphs.

EXAM SCHEDULING

While conducting exams the college authorities try to finish them in the least amount of time while ensuring that two exams that occur at the same time should not be written by the same student. Such problems can be solved using graphs.

NEED TO LEARN GRAPHS.

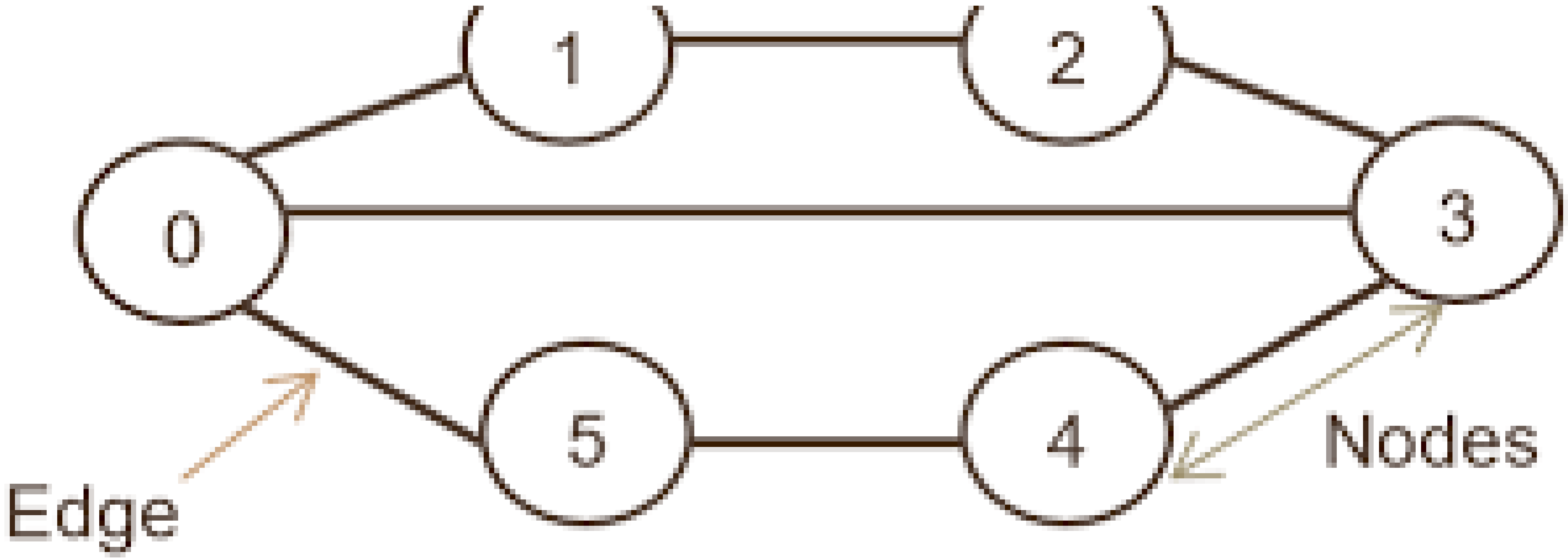
There are various problems in real life that require non-linear visualization of data. However, every data structure which we studied till now visualizes data linearly (eg., arrays, linked lists, stacks, queues, vectors, etc.). So the concept of graphs and trees will help us build the foundation for creating non-linear data structures that'll help solve much more complex problems.

FORMAL DEFINITION

A graph G can be defined as a pair (V,E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(u,v) \mid u, v \in V\}$. If the graph is undirected, the adjacency relation defined by the edges is symmetric, or $E \subseteq \{\{u,v\} \mid u, v \in V\}$ (sets of vertices rather than ordered pairs). If the graph does not allow self-loops, adjacency is irreflexive.

IN SIMPLER TERMS

A graph is a collection of nodes (also called vertices) that are connected by edges.



TYPES OF GRAPHS

- CONNECTED/UNCONNECTED GRAPH
- CYCLIC/ACYCLIC GRAPH
- DIRECTED/UNDIRECTED GRAPH
- WEIGHTED/UNWEIGHTED GRAPH
- BIPARTITE GRAPH
- SIMPLE GRAPH
- DENSE/SPARSE GRAPH

What are TREES?

Trees are nothing but acyclic connected graphs. Some of the properties of trees include:

- There are N-1 edges on a tree if the number of nodes = N.
- A tree is acyclic
- It is connected.

GRAPH/TREE REPRESENTATION

So far we have only discussed graphs theoretically. But when we'll write programs using them, then we need some way to represent them in the memory. For example, an array can be represented as a pointer pointing to the first element, a linked list can be represented as a pointer to the first node which in turn points to the next node, and so on. However representing non-linear data structures isn't as intuitive and straightforward as linear DS, so there are various ways to represent graphs.

Graph Representations

VECTOR OF EDGES

The most trivial way to represent graphs is using an array/vector of edges (pair of nodes which it connects). This type of representation is mostly used to give input in CP sites

ADJACENCY MATRIX

A very important way to represent graphs. It basically is a N*N matrix representing a graph containg N nodes, where matrix[i][j] = 0 if there is no edge between i , j otherwise it is 1 or sth else depending on whether the grpah is weighted or not.

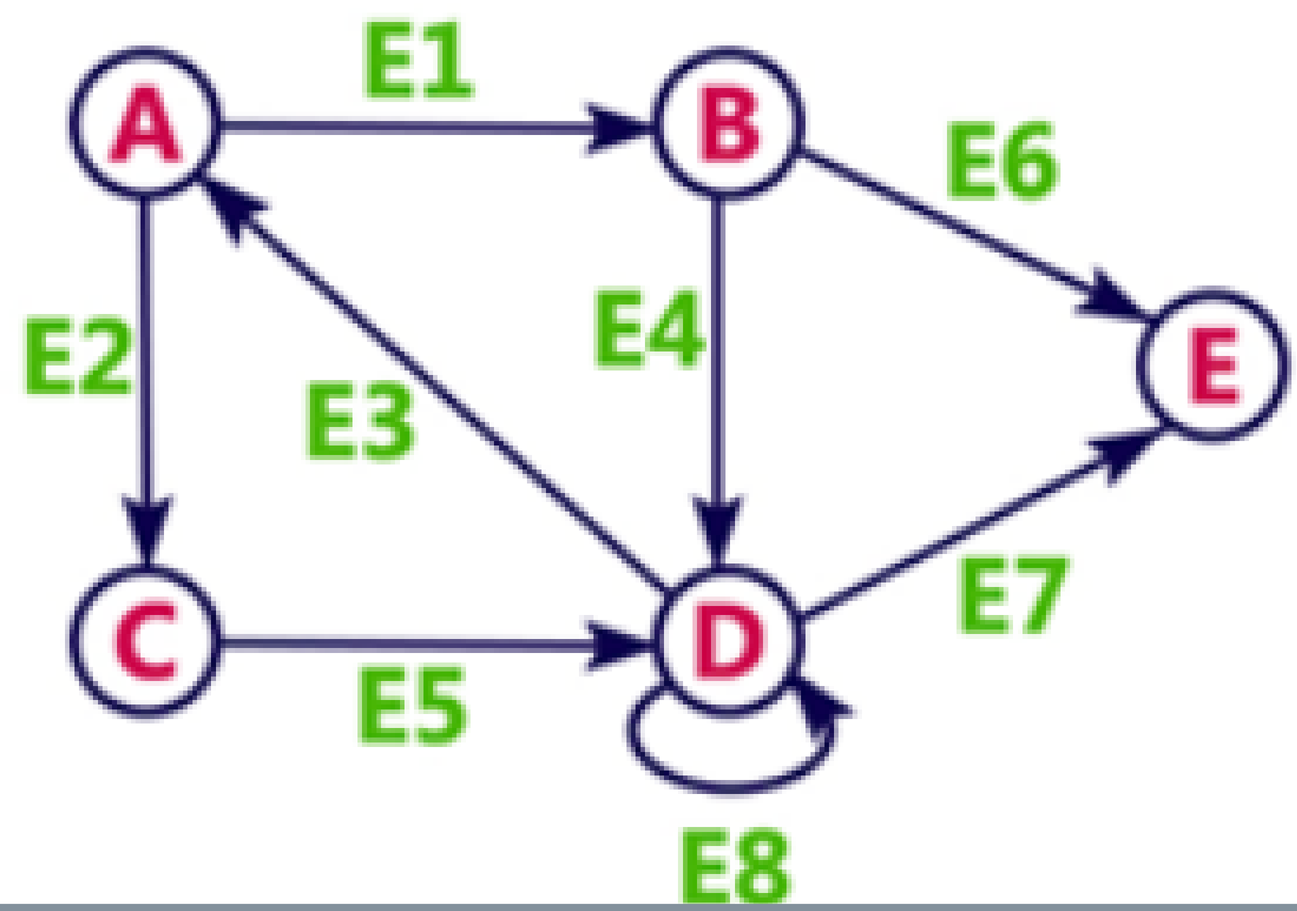
ADJACENCY LIST

Another important way to represent graphs. In this representation, every node has a list associated with it of the nodes which are connected to it. If the graph is weighted, the list is of pairs

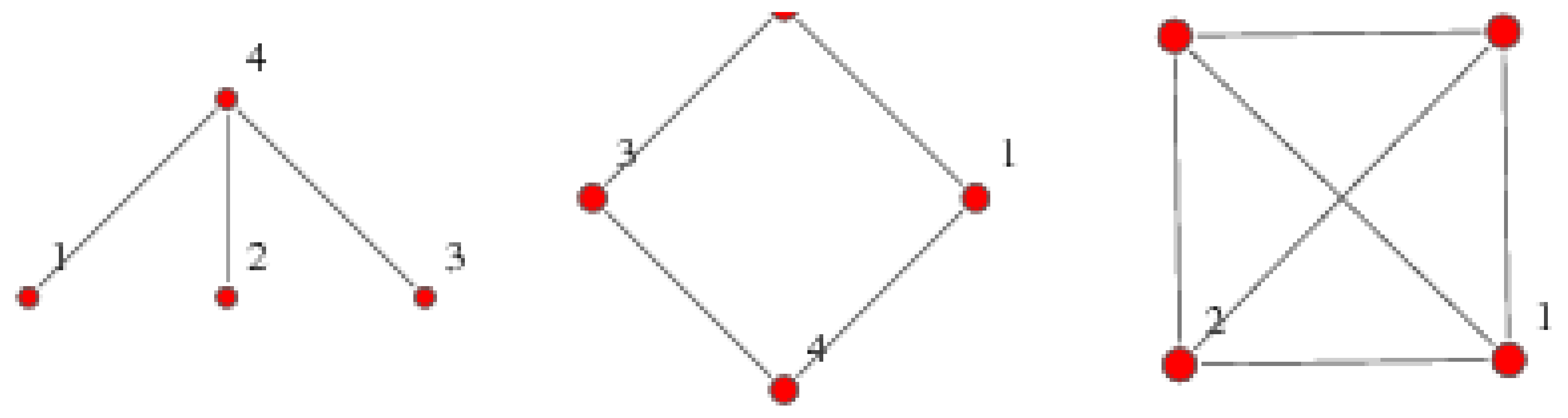
.

INCIDENCE MATRIX

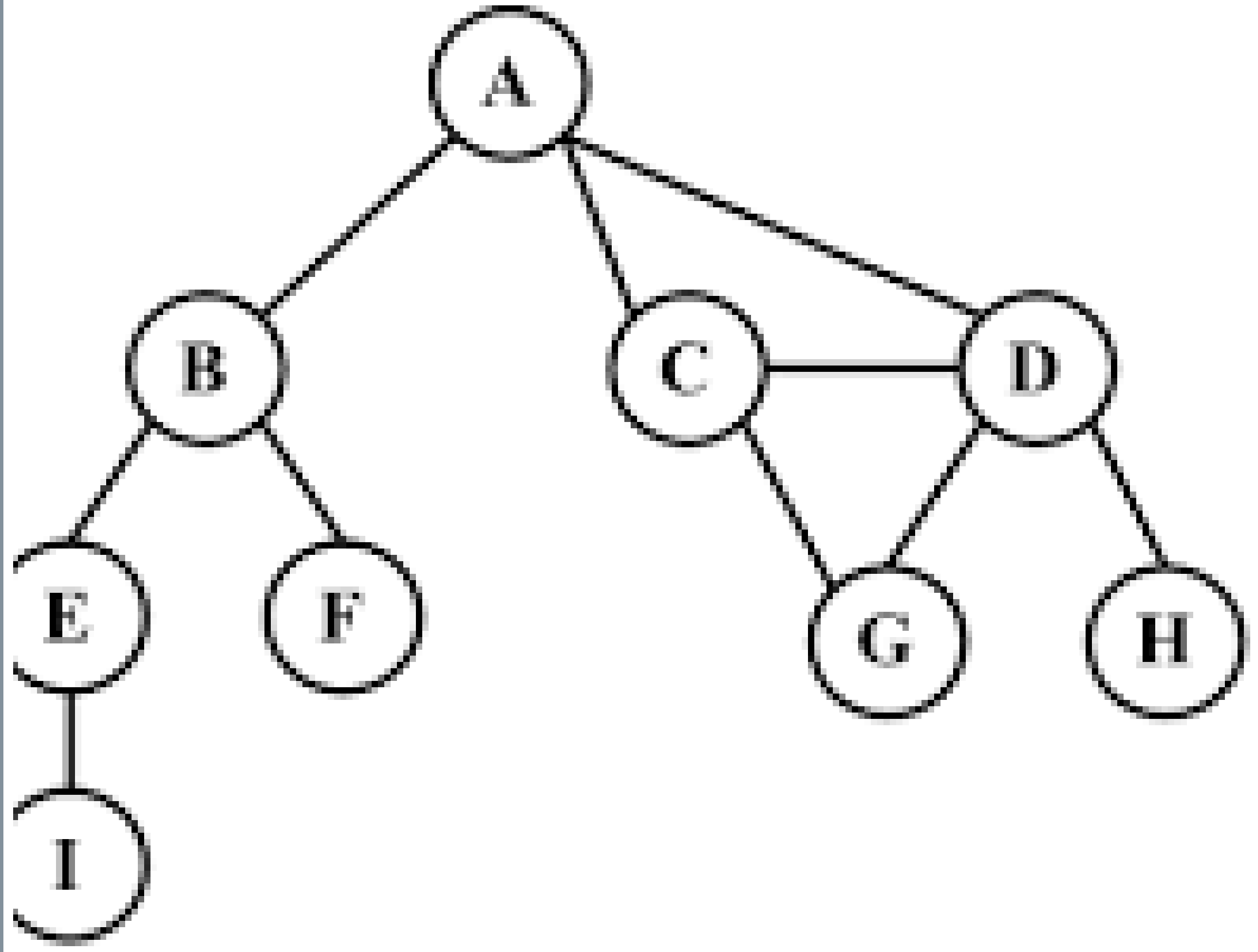
This is matrix of size N*E representing a graph of N nodes and E edges, where matrix[i][j] = 1 implies that jth edge is incident on the ith node.



$$\begin{array}{c|cccccccc}
 & E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 \\
\hline
A & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
B & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
C & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
D & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 \\
E & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0
\end{array}$$



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$



A	B	C	D	
B	A	E	F	
C	A	D	G	
D	A	C	G	H
E	B	I		
F	B			
G	C	D		
H	D			
I	E			

GRAPH TRAVERSALS.

How to traverse the graph we created?

DEPTH FIRST SEARCH (DFS)

It is a form of graph traversal in which we start from a node and keep exploring the depth of the graph as far below as we can go. Recursion uses this DFS.

BREADTH-FIRST DEARCH

It is a form of graph traversal in which we first explore all the immediate neighbours of a particular node and then move on to explore the other nodes.

DEPTH FIRST SEARCH

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Stacks are used for DFS.

Important terms for DFS

ENTRY/EXIT TIMES

These are self explanatory. They'll be very useful when studying about trees.

TREE EDGES

The edges traversed during the DFS traversal.

BACK EDGES

The remaining edges. Tree and back edges will be very useful when learning about bridges and articulation points.

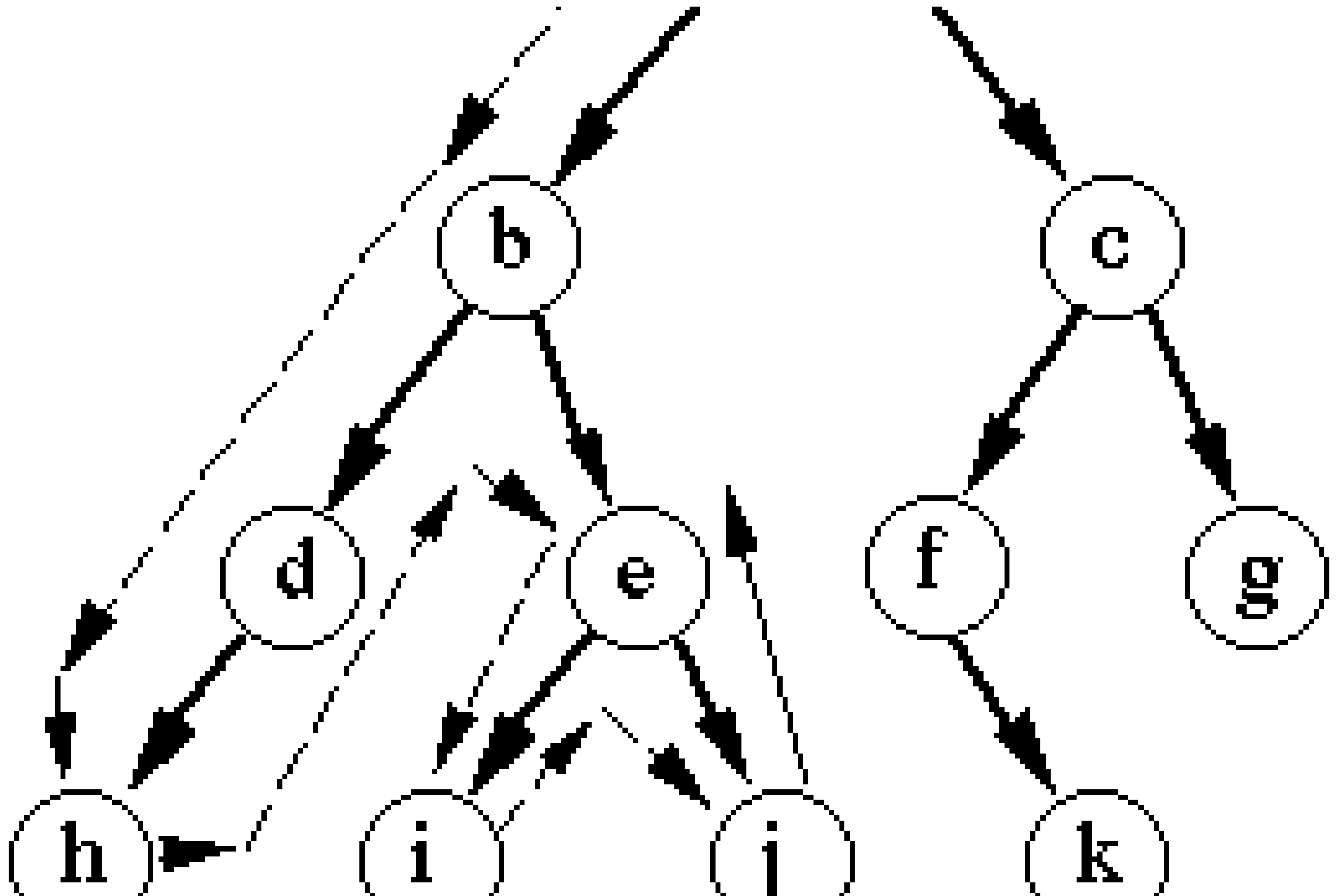
Breadth First Search

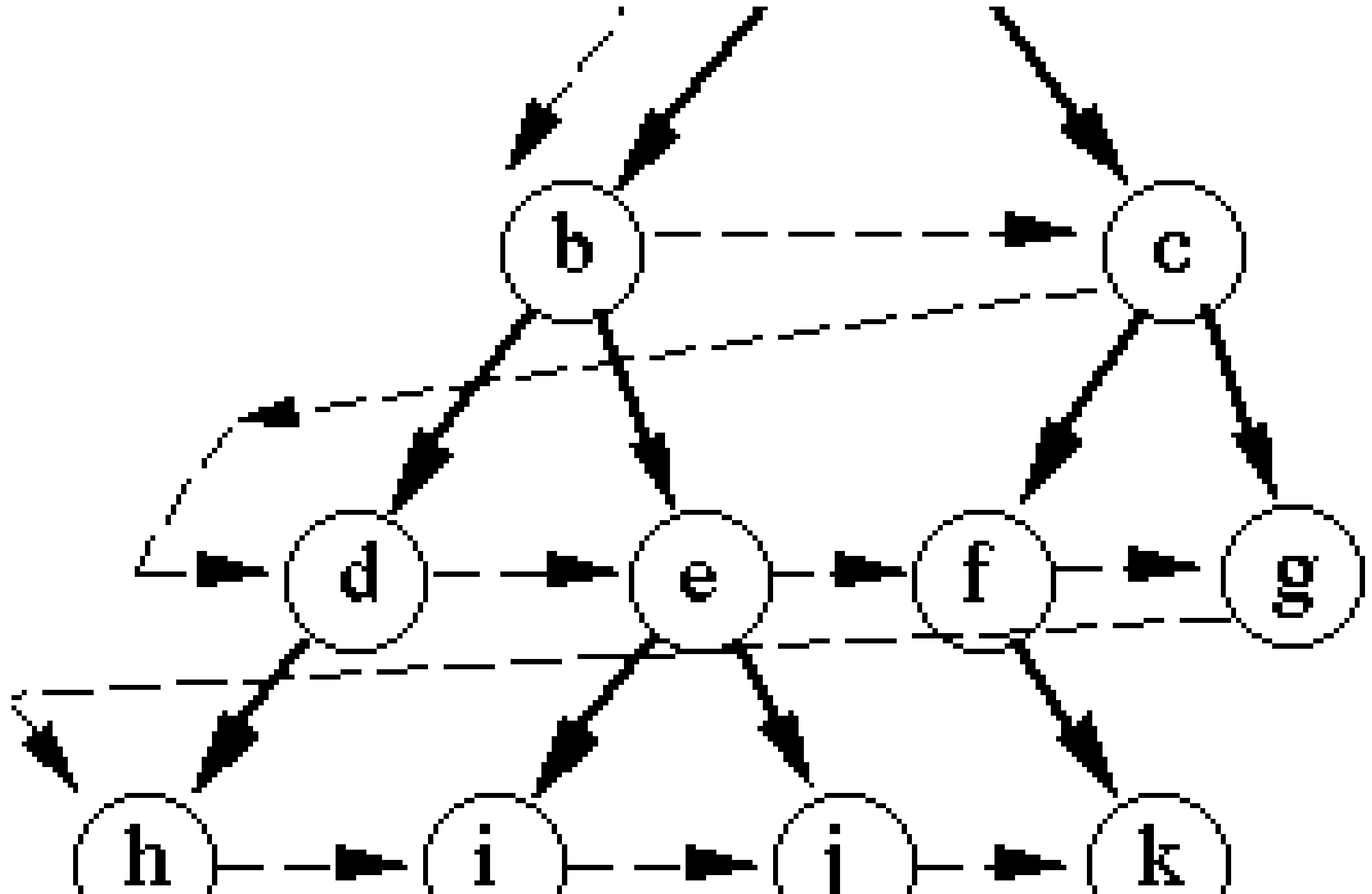
The algorithm can be visualized as a fire originating from some node and spreading throughout the graph using the edges.

Queues are used for the same.

WHEN TO USE WHAT?

Both the traversal mechanisms have their own advantages and disadvantages. The following set of problems will get you acquainted with what to use where.





Problem 1

Given an undirected graph determine the minimum distance each node has from the source node.

Problem 2

Given an undirected graph where each node has a letter associated with it. Determine the lexicographically minimum path of the graph from the root to each of the vertices.

Problem 3

Given a graph find if it contains any cycles.

Problem 4

Given a tree determine the minimum distance between the given root and any leaf.

Problem 5

Given an undirected graph, figure out if it's bi partite.

Problem 6 (0/1 BFS)

Given a graph such that each edge has either weight of 0 or 1. Find out the minimum distance each node has from the root, where distance refers to the sum of weights of the edges in the path from the root to the node.

Practice Problems

SPOJ ABCPATH

SPOJ KOZE