Algorithms

Lecture : Graphs - Definition and Representation

Imdad ullah Khan

Contents

Many of the problems that we will see later on often involve representations of data other

than the usual array of integers. We will now look at one such representation, the graph, and will explore its properties, as well as the situations that are usually associated with the use of graphs as a primary means of representation.

1 Introduction

1.1 Definition of Graph

A graph, in the formal sense, is defined as a pair G of two sets, V and E. Set V is defined as a set of *vertices*, whereas set E is defined as a set of *edges*. Vertices are typically represented by letters, and edges are represented as unordered pairs of these letters. In more mathematical terms, we say:

$$G = (V, E), where$$

$$V : \text{set of elements}$$

$$E \subseteq \binom{V}{2}$$

For a graph that contains n vertices, we can say:

$$|V| = n$$

There are some properties of a graph and its components that arise of this definition, we will be now looking at those properties.

1.2 Adjacency:

With the presence of the V and E sets, we can ask ourselves 'What are the properties of the elements of these sets?' We now define one such property, that defines the relationship between two vertices $a, b \in V$, which is the adjacency property. Two vertices $a, b \in V$ are defined to be adjacent if:

$$(a, b) \in E$$

i.e. the unordered pair (a, b) is a member of the set E

1.3 Neighborhood:

With the adjacency property defined, we can now define the neighborhood of a vertex v in G. The neighborhood of v is defined as the set of vertices in V that are adjacent to v in the graph G i.e.:

$$N(v) = \{u : (v, u) \in E\}$$

1.4 Degrees:

Finally, we can define the degree property of a vertex v, which is simply the number of vertices that are adjacent to v, i.e. the size of the set $N(v)$:

$$deg(v) = |N(v)|$$

A key lemma that is defined using the degree property is the **Handshaking lemma**, which states:

$$\sum_{v \in V} deg(v) = 2|E|$$

2 Representation of graphs

While we may represent a graph in our discussion in terms of sets and pairs, we find that these representations are not intuitive when we seek to define graphs in code. Hence we look into forms of representing graphs which are easier to code around.

2.1 Adjacency lists:

As the name implies, adjacency lists are lists, where each list represents the neighborhood of that vertex, i.e., it holds the list of vertices which are adjacent of that particular vertex. These lists themselves may be held in a list or an array, ordered by a standard vertex order. As an example, we use the graph in Figure 1 to build our adjacency list.

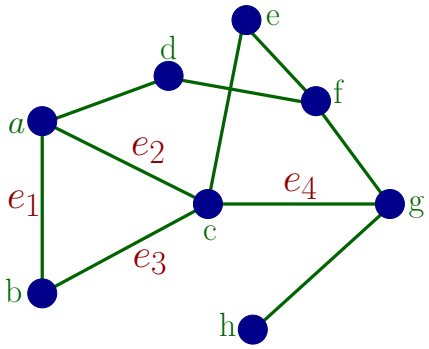


Figure 1: Sample Graph

The adjacency list for this graph will look as follows:

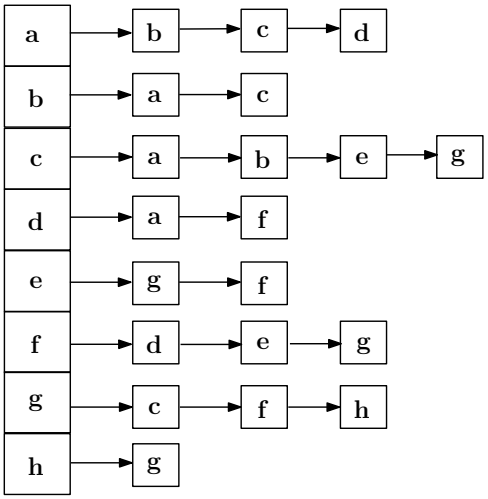
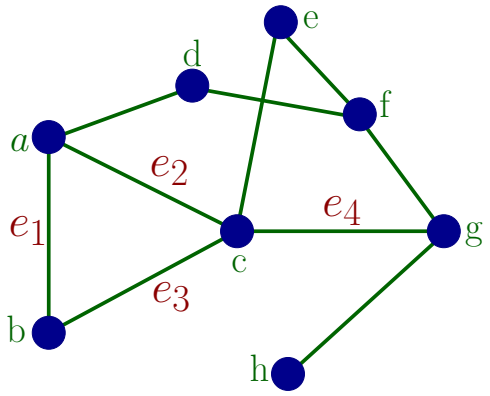


Figure 2: Adjacency List for Graph in Figure 1

2.2 Adjacency matrix:

Another form of graph representation is the adjacency matrix, which is simply a matrix with row and column indices as the vertices of the graph, and the value at a particular pair of indices indicating whether an edge exists between the two vertices. As an example, here is a graph and its adjacency matrix representation.



$A_G =$

	a	b	c	d	e	f	g	h
a	0	1	1	1	0	0	0	0
b	1	1	0	0	0	0	0	0
c	1	1	0	0	1	0	1	0
d	1	0	0	0	0	1	0	0
e	0	0	1	0	0	1	0	0
f	0	0	0	1	1	0	1	0
g	0	0	1	0	0	1	0	1
h	0	0	0	0	0	0	1	0

A key point to observe is that the matrix is symmetric along the diagonal. This means that access to the matrix can simply be limited to one of the triangular areas to reduce computations.

2.3 Importance of graph representation:

With these two representations, we can now ask 'Which representation is more efficient for use in our algorithm?'. It turns out that it depends on the properties of the graph to represent. More specifically, it depends of the number of edges of the graphs. Generally, it is more efficient to use the adjacency list when:

$$0 \leq |E| \leq \binom{n}{2}$$

for an n-vertex graph. If the graph has edges to the order n^2, it is better to use an adjacency matrix.

3 Paths and graph connectivity

With defining the initial properties and the means of representation of a graph, we can now look at some important items relating to graphs, such as paths and connectivity.

3.1 Path:

A path is a sequence of vertices $v_1, v_2, \ldots, v_k$, with no repetition such every two consecutive pair of vertices is an edge in the graph. We say the path is between v_1 and v_k. As an example, we can use Figure 1, where a path between 1 and 5 exists in the form $Path(1, 5) = 1, 2, 5$ where there is an edge between 1 and 2 and an edge between 2 and 5. Another path for the same two vertices is 1,3,4,5.

3.2 Length of a path:

The length of a path is the number of edges involved, in the path or the number of vertices minus one. In the above two examples the paths have lengths 2 and 3 respectively.

3.3 Reachability and connectivity:

One of the most important questions when dealing with graphs is whether a vertex is reachable from another vertex i.e. is it possible to get to vertex b starting from vertex a. There are two ways in which we can answer this question. If a is adjacent to b or there exists a path from a to b, then b is reachable from a. Otherwise b is not reachable from a. An algorithmic method of finding whether b is reachable from a is as follows:

1. If $a = b$, then b is reachable.
2. if $b \in N(a)$ then b is reachable.
3. Repeat step 2 for neighbors of a, then the neighbors of the neighbors of a, and so on, till all possible options are exhausted.

From this concept, we can now define another property of the graph, which is its connectivity. A graph G is defined as connected when every vertex is reachable from every other vertex. In more formal terms:

$$\text{G is connected when: } \forall a \neq b \in V \ni Path(a, b)$$

If this condition is not met, then the graph is said to be *disconnected*. Note that a graph consisting of a single vertex and no edges is considered to be connected. An edge e in a graph G is called a **cut edge** if removing e from G disconnects the graph.

3.4 Connected Components:

Previously, we defined the conditions for a graph to be considered as connected, but within a disconnected graph, we may be able to find connected subgraphs. A subgraph is simply a subset of the graphs vertices and edges, such the edges only exist between the vertex subset. Hence for a graph that is disconnected, we can say that there are *connected components*, that taken as subgraphs are connected.

3.5 Cycles:

There may exist graphs where there are vertices that are reachable to themselves via other vertices i.e there exists a path for some vertex v to itself, which includes vertices other than itself. Such a path is called a *cycle*, and a graph that contains a cycle is *cyclic*.

3.6 Tree and Forest

If a graph is connected and it contains no cycles, it is called a **tree**. If the graph does not contain any cycles, but is not connected, it is called a **forest**. The connected components of a forest are trees, since they are connected and have no cycles, hence the name. There are many important characterizations of trees (and forests). See Problem Set.

4 Directed Graphs

While the type of graphs defined above is suitable for many applications, some problems require a definition that is more stringent. This requirement is defined by directional graphs, or digraphs for short.

4.1 Definition

As the name suggests, a directed graph incorporates a sense of direction. This is defined by the direction of an edge in the graph, which defines the order in which its vertices are

accessed. Using this notion, we now mathematically define the directional graph:

$$G = (V, E), \text{ where } E \subseteq V \times V, (a, b) \neq (b, a)$$

That is, G is a graph where E is a subset of the cross product of V with itself, and the edge (a, b) is not equal to (b, a), in that one must go from a to b along that particular edge, not the other way around.

4.2 Adjacency in digraphs:

Following the definition of the directed graph, we now need to modify our previous definitions of the properties of the graphs, starting with adjacency. Adjacency in directed graphs is defined as follows, vertex a is adjacent to vertex b if there exists an edge that goes from b to a i.e. $(b, a) \in E$. Note that a being adjacent to b does not imply that b is adjacent to a.

4.3 In/out neighborhood:

With the addition of direction, neighborhoods in digraphs must now be defined with respect to the direction from the particular vertex. We use the following graph to illustrate this point:

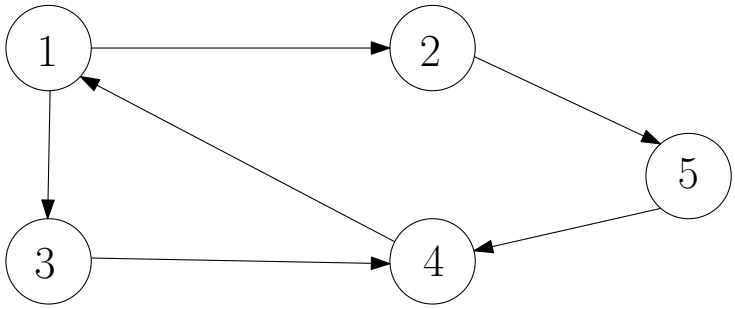


Figure 3: Sample directed graph

We now define two kinds of neighborhoods The *in-neighborhood* of a vertex a is the set of vertices that have edges directed towards a i.e. which have a adjacent to them , whereas the *out-neighborhood* of a vertex is the set of vertices to whom a has edges directed to i.e.

that are adjacent to a. We use $N^-(a)$ and $N^+(a)$ to represent the in/out neighborhood respectively. So in our example directed graph, we have:

$$N^-(1) = \{4\}$$

$$N^+(1) = \{2, 3\}$$

$$N^-(2) = \{1\}$$

$$N^+(2) = \{5\}$$

$$N^-(4) = \{3, 5\}$$

4.4 In/out degree:

With the in/out neighborhood defined, we now define the in/out degrees for a vertex, which are simply the size of their respective neighborhoods With the previous example, we get:

$$deg^-(1) = |N^-(1)| = 1$$

4.5 Directed paths and directed reachability:

With our neighborhoods and degrees defined, we can now talk about paths in directed graphs. We previously defined paths in undirected graphs to be ordered sets of vertices which have edges between adjacent vertices in the list. With directed paths, we slightly modify the definition to include another constraint, which is that any vertex in the path must be in the out-neighborhood of the preceding vertex. We take Figure 3 as our example graph, and take a path between 1 and 5 as an example:

$$Path(1, 5) = \{1, 2, 5\}$$

Note that each vertex is in the out neighborhood of the preceding vertex, where there is one.

4.6 Strongly, weakly and simply connected graphs:

We now go on to define connectivity in directed graphs. In undirected graphs, we saw that connectivity meant that each vertex is reachable from every other vertex i.e. there exists a path from every vertex to every other vertex. Due to the nature of directed graphs, we define connectivity in terms of three types. A **strongly connected** graph is one where there exists a path from every vertex to every other vertex. Note that this means that for any pair of vertices a and b, $Path(a, b)$ and $Path(b, a)$ exist. A **connected** graph has a weaker constraint, in that it requires either a path from a to b or a path from b to a for all vertices. Finally, a **weakly connected** directional graph will give a connected undirected graph if its edges are replaced with undirected edges.