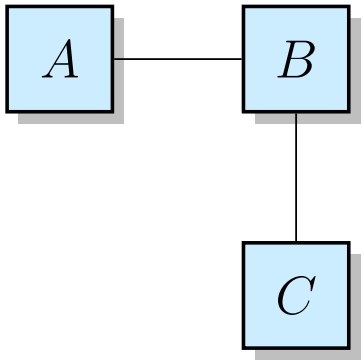
Graphs

Sujit Chakrabarti

In several of the next videos we are going to learn about graphs, arguably the most important and versatile abstract data types.

1 Graphs in Real Life

We often like to represent the presence of a relation or the lack thereof between two entities in the following manner:



The general common-sense interpretation of this picture is that A is related to B, B is related to C. But there seems to exist no relation between A and C. In this Internet age, the best example we can quote of such a scenario is FaceBook, or LinkedIn. A, B and C represent the users, and the existence of a line between any two elements of this type indicates that they are 'friends' of each other. There are n number of cases where entities and relations between them are shown pictorially using something like the above, e.g. flight/train/bus routes (transportation engineers), electric circuits (electrical engineers), class diagrams, flowcharts (computer scientists), finite state machines add more examples etc .

All these pictures are examples of graphs. Graphs are mathematical objects with *nodes* (representing things) and *edges* (representing relations). Nodes are often called *vertices*; edges are also called *arcs*.

2 Taxonomy of Graphs

Graphs can be of various types depending on what they are being used to represent. For example, consider the 'Friendship' graph in FaceBook. Here, two users are either friends of each other, or they are not. If A is B's friend, it implies that B is A's friend. That is, the *friendship* relation is symmetric. Graphs used to represent such relations would typically be drawn with edges being line segments. Such graphs are called undirected graphs.

On the other hand, consider a flowchart representing an algorithm. Here, an edge from A (representing an instruction in the algorithm) to B (representing another instruction) means that it is possible for control to flow directly from A to B in at least one of the executions of the algorithm. Here, an edge (called a control flow edge)from A to B defines some sort of successor relation, wherein B is A's successor. Edges in such graphs are typically arrow to indicate the directionality of the relation. These relations (like successor) are assymmetric. Therefore, an arrow flowing from B to A would mean exactly the opposite (i.e. A is successor of B). Such graphs are called *directed graphs* or *digraphs*.

There are graphs which are used to represent multiple types of relations between entities, some of them symmetric and some assymmetric. Take, for example, a graph that represents friendship between two Facebook users using an undirected edge, and 'follows' relation using an arrow or directed edge. Such graphs, with both directed and undirected edges are called *mixed graphs*.

Graphs may represent relations which are such that two entities may have only one relation, e.g. Facebook friendship. Such a graph will always have at the most one edge between two nodes. Also, a person can't be his own friend, i.e., there can't be an edge starting and ending on the same node. Such graphs are called *simple* graphs. On the contrary, there are graphs which represent relations multiple instances of which may exist between two entities. Again take the example of a flowchart, in which multiple control flow edges may exist between the same pair of nodes. Similarly, there may be edges starting and ending on the same node. Such graphs are often called *multigraphs*.

For the purpose of this class, we will often be dealing with undirected simple graphs. When we wish to deal with other types of graphs, we will make it explicit.

Two nodes are called neighbours if there exists an edge between them. Mathematically, predicate $neighbour(A, B)$ holds true if node A and node B are neighbours. A path is the sequence of nodes such that each pair of consecutive nodes in it are neighbours in the graph.

Example

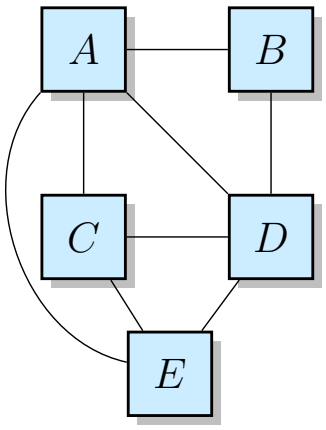


Figure 1: An undirected graph

Fig. 1 shows an undirected graph. Nodes are A, B, C, D and E. A and B are neighbours as there's an edge between them. So are B and D. ABD is a path since A and B are neighbours and B and D are neighbours. In contrast, ABC is not a path because B and C are not neighbours.

3 Graph ADT

4 Traversals

Most algorithms which work with graphs involves visiting the nodes in certain order and doing 'something'. For instance, you may want to print the graph, find the shortest path between two nodes, check whether one graph is same as the other etc. etc.

Most – if not all – of these algorithms are built on top of or are variations of two prototypical traversal techniques: the depth first search and breadth first search. Studying DFS and BFS also gives us a very good way of understanding how the graph abstract data type can be used to implement algorithms.

We now look at these two search/traversal techniques.

4.1 DFS

4.1.1 Recursive DFS

4.1.2 DFS using a Stack

4.2 Programming Problems

Use DFS to:

1. Find if a graph is connected (i.e. there exists a path between all pairs of nodes).
2. Find if a graph has a cycle (i.e. there exists a node which has a path to itself).

4.3 BFS

4.4 Generic Traversals

5 Implementations

5.1 Edge List

5.2 Adjacency Matrix

5.3 Adjacency List

6 Directed Graph

7 Directed Acyclic Graphs

7.1 Topological Sort

8 Dijkstra