Chapter 2
The Graph Traversal Pattern

Marko A. Rodriguez
AT&T Interactive, USA

Peter Neubauer
Neo Technology, Sweden

ABSTRACT
A graph is a structure composed of a set of vertices (i.e. nodes, dots) connected to one another by a set of edges (i.e. links, lines). The concept of a graph has been around since the late 19th century, however, only in recent decades has there been a strong resurgence in both theoretical and applied graph research in mathematics, physics, and computer science. In applied computing, since the late 1960s, the interlinked table structure of the relational database has been the predominant information storage and retrieval model. With the growth of graph/network-based data and the need to efficiently process such data, new data management systems have been developed. In contrast to the index-intensive, set-theoretic operations of relational databases, graph databases make use of index-free, local traversals. This chapter discusses the graph traversal pattern and its use in computing. (Angles & Guiterrez, 2008)

INTRODUCTION
The first paragraph of any publication on graphs usually contains the iconic \( G = (V,E) \) definition of a graph. This definition states that a graph is composed of a set of vertices \( V \) and a set of edges \( E \). Normally following this definition is the definition of the set \( E \). For directed graphs, \( E \subseteq (V \times V) \) and for undirected graphs, \( E \subseteq \{V \times V\} \). That is, \( E \) is a subset of all ordered or unordered permutations of \( V \) element pairings. From a purely theoretical standpoint, such definitions are usually sufficient for deriving theorems. However, in applied research, where the graph is required to be embedded in reality, this definition says little about a graph’s realization. The structure a graph takes in the real-world determines the efficiency of the operations that are applied to it. It is exactly those efficient graph operations that
yield an unconventional problem-solving style. This style of interaction is dubbed the graph traversal pattern and forms the primary point of discussion for this chapter\(^1\).

**THE REALIZATION OF GRAPHS**

Relational databases have been around since the late 1960s (Codd, 1970) and are today’s most predominate data management tool. Relational databases maintain a collection of tables. Each table can be defined by a set of rows and a set of columns. Semantically, rows denote objects and columns denote properties/attributes. Thus, the datum at a particular row/column-entry is the value of the column property for that row object. Usually, a problem domain is modeled over multiple tables in order to avoid data duplication. This process is known as data normalization. In order to unify data in disparate tables, a “join” is used. A join combines two tables when columns of one table refer to columns of another table.\(^2\) This is the classic relational database design which affords them their flexibility (Mishra & Eich, 1992).

In stark contrast, graph databases do not store data in disparate tables. Instead there is a single data structure – the graph (Angles & Guiterrez, 2008). Moreover, there is no concept of a “join” operation as every vertex and edge has a direct reference to its adjacent vertex or edge. The data structure is already “joined” by the edges that are defined. There are benefits and drawbacks to this model. First, the primary drawback is that it’s difficult to shard a graph (a difficulty also encountered with relational databases that maintain referential integrity).

Sharding is the process of partitioning data across multiple machines in order to scale a system horizontally.\(^3\) In a graph, with unconstrained, direct references between vertices and edges, there usually does not exist a clean data partition. Thus, it becomes difficult to scale graph databases beyond the confines of a single machine and at the same time, maintain the speed of a traversal across sharded borders. However, at the expense of this drawback there is a significant advantage: there is a constant time cost for retrieving an adjacent vertex or edge. That is, regardless of the size of the graph as a whole, the cost of a local read operation at a vertex or edge remains constant. This benefit is so important that it creates the primary means by which users interact with graph databases – traversals. Graphs offer a unique vantage point on data, where the solution to a problem is seen as abstractly defined traversals through its vertices and edges.\(^4\)

**The Indices of Relational Tables**

Imagine that there is a gremlin who is holding a number between 1 and 100 in memory. Moreover, assume that when guessing the number, the gremlin will only reply by saying whether the guessed number is greater than, less than, or equal to the number in memory. What is the best strategy for determining the number in the fewest guesses? On average, the quickest way to determine the number is to partition the space of guesses into equal size chunks. For example, ask if the number is 50. If the gremlin states that its less than 50, then ask, is the number 25? If greater than 25, then ask, is the number 37? Follow this partition scheme until the number is converged upon. The structure that these guesses form over the sequence from 1 to 100 is a binary search tree. On average, this tree structure is more efficient in time than guessing each number starting from 1 and going to 100. This is ultimately the difference between an index-based search and a linear search. If there were no indices for a set, every element of the set would have to be examined to determine if it has a particular property of interest.\(^5\) For \(n\) elements, a linear scan of this nature runs in \(O(n)\). When elements are indexed, there exists two structures – the original set of elements and an index of those.