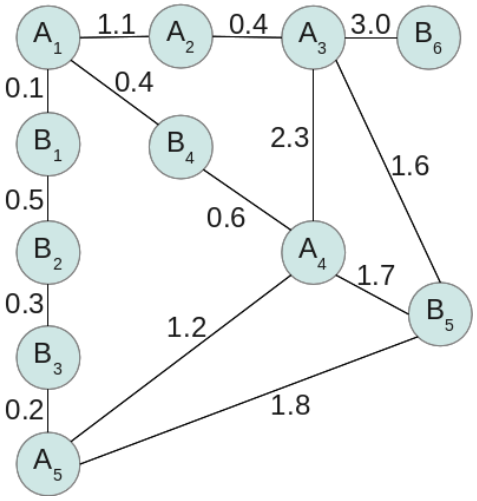
Graph Databases

Bayan Berri, Terry Guan, Shannon Lau

April 11, 2018

1 Graphs

A graph database is merely a representation of a mathematical graph. There are two parts of a graph, the node and the edge. Edges connect nodes. Edges can also be weighted, meaning they have a value. In the diagram below, the letters represent nodes and the lines represent edges. The weighted edges have numbers ranging from as low as 0.1 up to 3.0.



There are two types of graphs:

- Directed Graphs: Directed graphs are a special type of graph in which edges have a direction associated with them. An example of a directed graph is if person A has romantic feelings for person B without their knowledge. It's a one way road. It's a directed graph.
- Undirected Graphs: Undirected graphs are cases in which the edges just link nodes without any direction associated. An example of an undirected graph is if two people shake hands at a party. The handshake connects the two equally. There is no direction.

Other examples of graphs include a transit map or a family tree. A transit map connects different locations (nodes) with a train route (edges). Similarly members of the family are nodes of a family tree while an edge represents their relationship. The edges can have weights such as age difference which can help us gather more information from the graph.

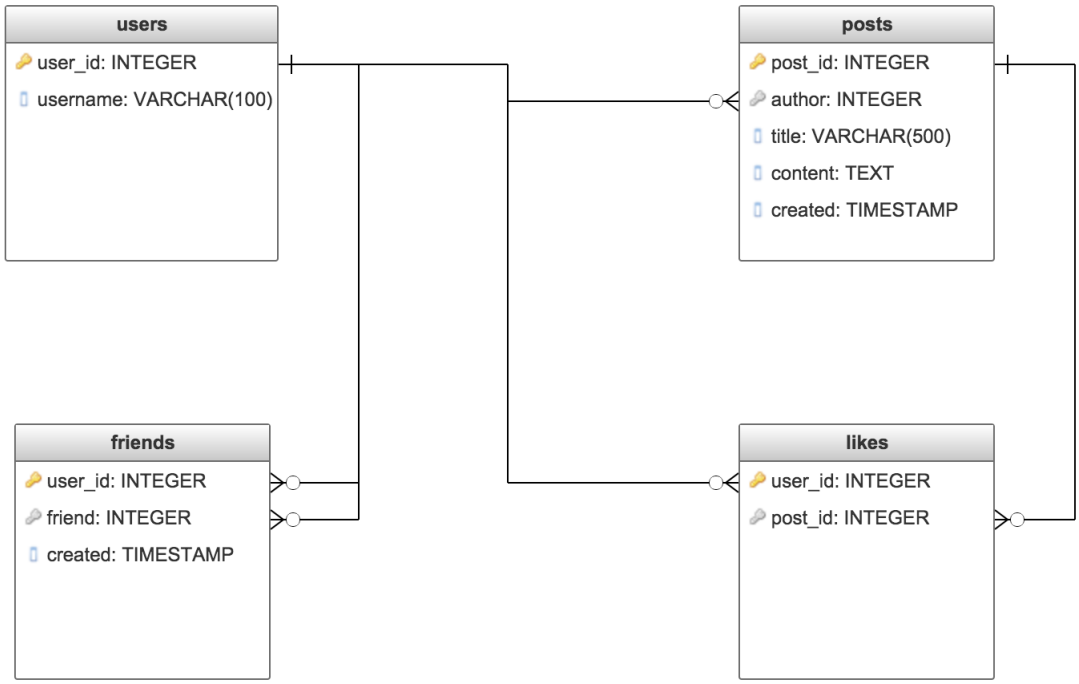
2 Other Databases

To fully understand the use and purpose of graph databases, it is important to acknowledge other types of graphs and their unique features.

2.1 Relational Database

A relational database highlights relationships of data through tables. SQLite, for example, is a relational database. Let's propose that we are creating a social media platform and we use a relational database to store any necessary information

Example. A relational database schema to show users and posts and their connections.



Note that there are four different tables in the diagram. Only two of thee are actual data (users and posts), while the other two represent connections in our data (friends and likes).

2.2 Document Store Database

A document store database can model data in many more ways using a JSON or XML format. MongoDB is an example of a document store database. Nested information gives the database structure, but it can be difficult to draw connections between different elements.

Example. A sample JSON to show users and posts and their connections.

```
// Users
{
  "user_id": "u1",
  "username": "grahamcox",
  "friends": {
    "u2": "2017-04-25T06:41:11Z",
    "u3": "2017-04-25T06:41:11Z"
  }
}

// Posts
{
  "post_id": "p1",
  "author": "u1",
  "title": "My first post",
  "content": "This is my first post",
  "created": "2017-04-25T06:41:11Z",
  "likes": [
    "u2"
  ]
}
```

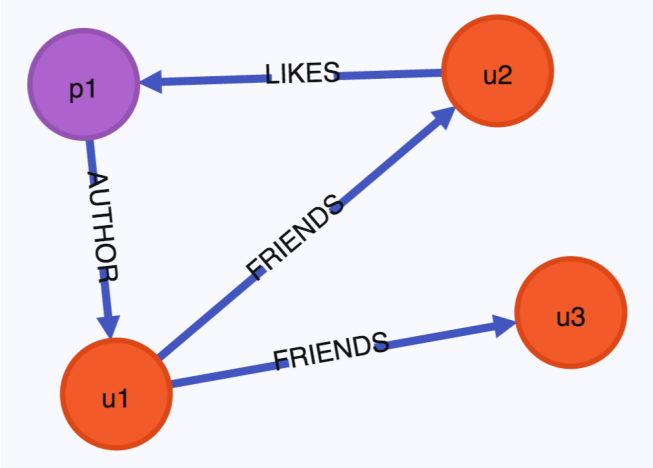
With a document database, we've reduced the number of entities to two, and some of the connections are a lot clearer. However, some key relationships, like post to post or user to user,

are still difficult to access and visualize. It would require parsing through both datasets with an unnecessarily long runtime.

3 Graph Database

Graph databases models data using nodes and edges, allowing you to represent complex interactions in much more natural form. They give the flexibility of a document store database but supports relationships seen in a relational database. Cypher is a common graph database querying language to implement this type of database.
We can choose nodes to represent our entities and edges to represent those relationships where we can easily traverse the links inside the database itself.

Example. A graph database diagram to show users and posts and their connections.



Although it is not shown, each node and edge contain specific data. For example, a "FRIENDS" edge would contain the relationship's time of creation. By simply traversing through a graph database, it is much more efficient to access and process data.

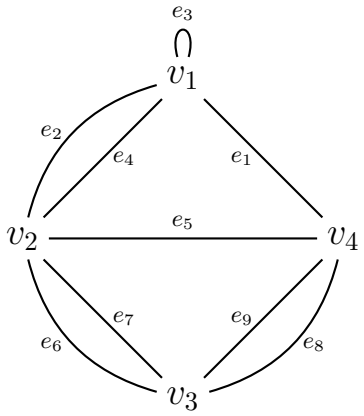
4 Dijkstra's Algorithm

Many of graph databases utilities are based on the mathematics of graph theory. One example is Dijkstra's Algorithm which finds the shortest path between two vertices in a graph– it is used in real life applications such as finding the shortest distance between two locations. Below we'll go through a rigorous derivation of Dijkstra's Algorithm using graph theory to demonstrate the power of graph theory and subsequently graph databases.

4.1 Notation

First a review of our notation. Let V be the set of vertices, E be the set of edges, and $G(V, E)$ be the graph given vertices V and edges E.

Example.



V: $v_1, v_2, v_3, \ldots$
E: $e_1, e_2, e_3, \ldots$
Thus the graph represented can be denoted as $G(V, E)$

4.2 Definitions

These are a few definitions you'll need to understand for our derivation of Dijkstra's Algorithm. There are more terms in graph theory but these are the one's you'll need.

Definition 1 (Walk)**.** A walk is a finite sequence of alternating vertices and edges

Example. Using the graph above, a possible walk is

$$v_1, e_1, v_4, e_8, v_3, e_7, v_4, e_8, v_3$$

Definition 2 (Trail)**.** A trail is a walk where no edges repeat

Example. Using the graph above, a possible trail is

$$v_1, e_1, v_4, e_8, v_3, e_7, v_4$$

Notice that v_4 repeated. Vertices can repeat in a trail– only edges cannot

Definition 3 (Path)**.** A path is a trail where no vertices repeat or a walk where no edges or vertices repeat

Example. Using the graph above, a possible path is

$$v_1, e_1, v_4, e_8, v_3$$

Definition 4 (Parallel)**.** Edges are considered parallel if the edges end vertices are the same

Example. Referring to the the graph above, e_2 and e_4 are parallel

Definition 5 (Loop)**.** A loop is an edge where the beginning and ending vertex are the same

Example. Referring to the graph above, e_3 is a loop

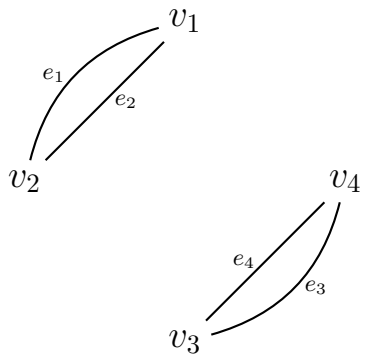
Definition 6 (trivial)**.** A graph is trivial if it is one vertex

Definition 7 (Connected)**.** A graph is considered connected if there exists a path between every vertex.

Example. the above graph is considered connected because there exists a walk between every vertex. In other words, every vertex is reachable by every other vertex.

Definition 8 (Component)**.** A subgraph G_1 of graph G is a component if G_1 is connected and is connected to any additional vertices to supergraph G

Example. Graph G



$G_1(v_1, v_2, e_1, e_2)$ is a component because it is a subgraph of supergraph G and is not connected to any other vertices in supergraph G. In other words, it is isolated from all other vertices.

4.3 Derivation

For our derivation we'll assume that the graph is connected or contains components and that it has no loops. This is a justified assumption because in the real world data is mostly connected and not completely isolated; assuming no loops is also justified because there is no need for relational data that relates to itself in the real world.

Lemma 1. *The subpath $v_b \dots v_c$ of the shortest path $v_a \dots v_n$ is itself the shortest path between v_b and v_c*

The proof is trivial and is left as an exercise for the reader. (It really is quite intuitive)

Corollary 1. *The shortest path S is the addition of all its shortest subpaths s*

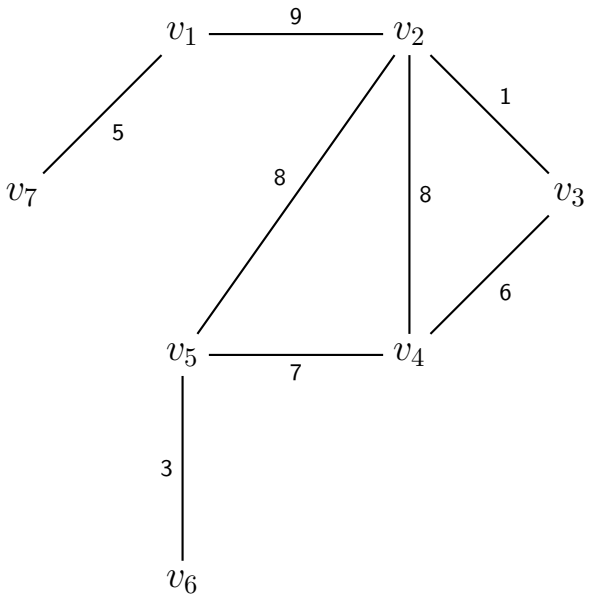
This corollary is merely the converse of the previous lemma

Using our corollary we define function δ to be the shortest path between v_0 and v_n

$$\delta(v_0, v_n) \coloneqq \delta(v_0, v_u) + \delta(v_u, v_n)$$

where v_u is a set of all vertices adjacent to v_0 Intuitively, this is saying that the shortest path between vertex v_0 and vertex v_n is the addition of all the shortest paths formed by vertices connected to v_0 and v_n.

Example.



In a trivial calculation such as $\delta(v_5, v_4)$ we can immediately see that the shortest path is v_5, v_4 and not v_5, v_2, v_4. However in a more complex calculation such as $\delta(v_5, v_3)$, it becomes hard to figure out how to determine that v_2 is a better choice than v_4 algorithmically. This issue can be solved by keeping track of the shortest path between v_5 and **all** other vertices. Given our corollary, if we find all subpaths s, we can add them up to get the desired path S. This is well visualized in the following link: https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

One of the beauty of Dijkstra's Algorithm is it can compute the shortest path between all vertices in $O(n^2)$. However, it can also be a drawback since you cannot calculate the shortest path between two specific vertices.

Sources

https://www.compose.com/articles/introduction-to-graph-databases
http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf