NoSQL: Graph Database

Presenter: Longbin Lai

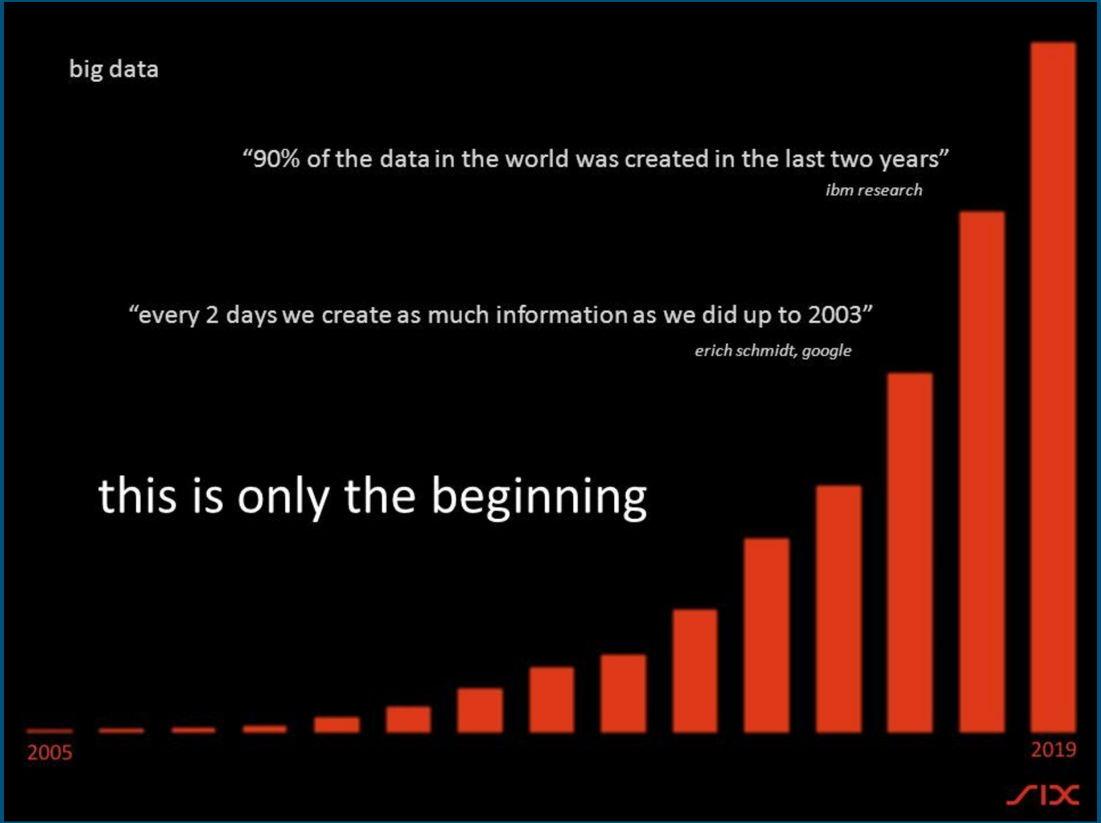
Background

From SQL Database to NoSQL Database

- What is SQL Database?
    - Simply: What you have studied prior to this course, a.k.a. relational database
    - Features:
        - The data is well-structured (schema)
        - Querying using SQL
- What is NoSQL Database?
    - No standard definition, “Not only SQL”
    - Features:
        - The data can be stored in any format according to applications: Text file, KeyValue, XML, …
        - Querying is often ad-hoc, but more often borrow from SQL

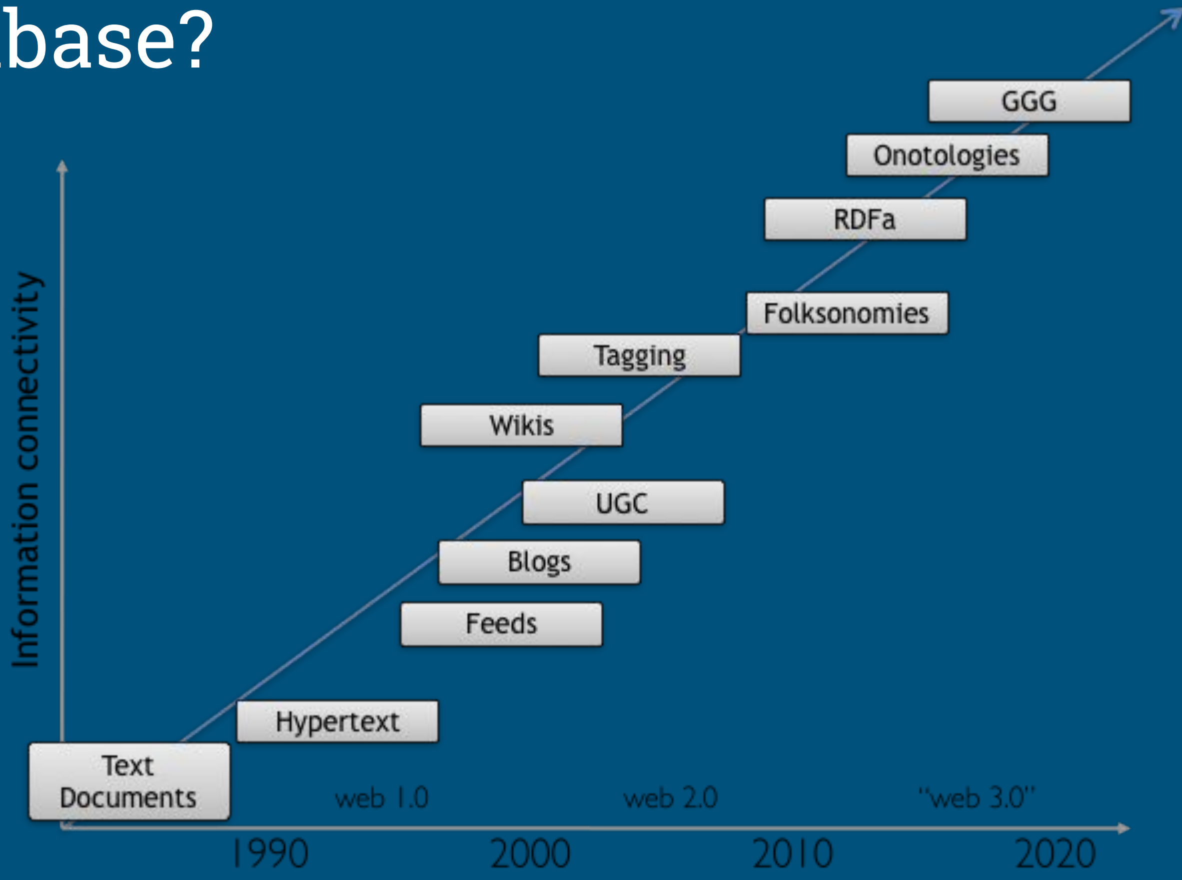
Why NoSQL Databases?

- Data is getting bigger
- How much you are using in you postgresql?

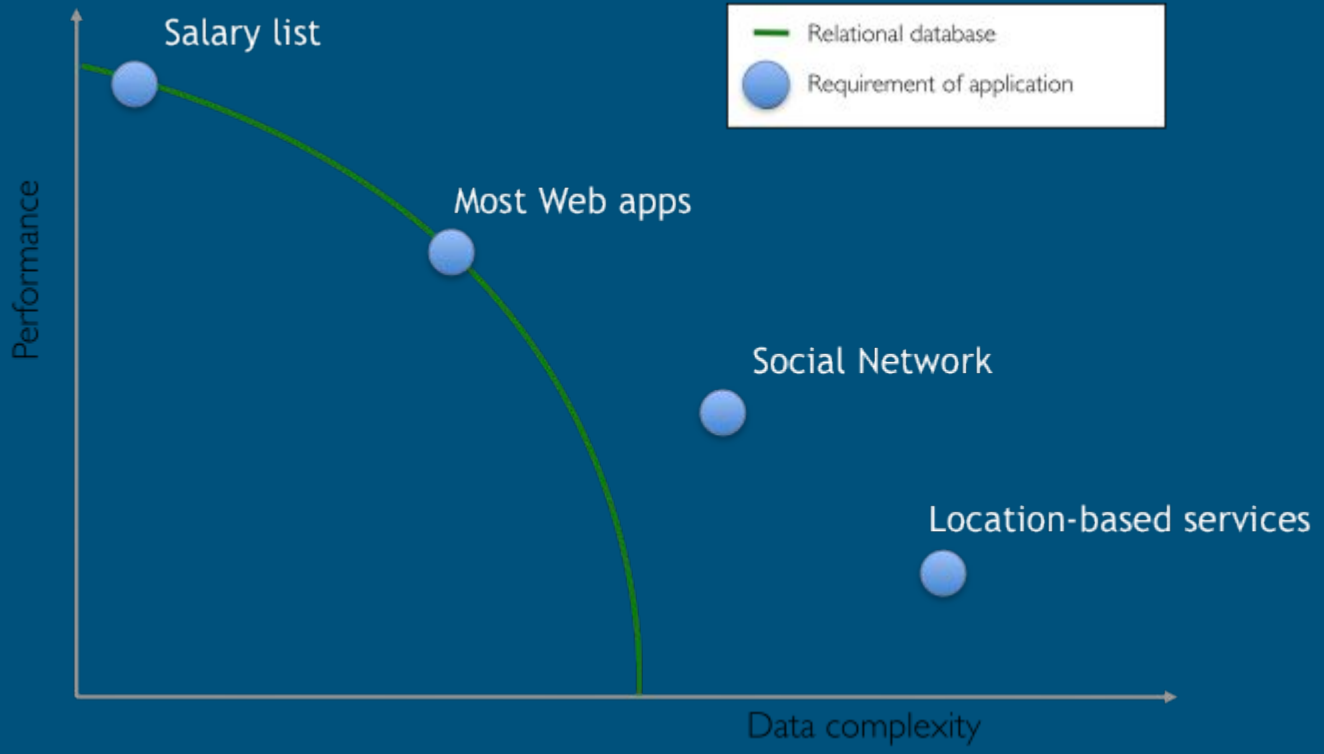


Why NoSQL Database?

- Data is more connected

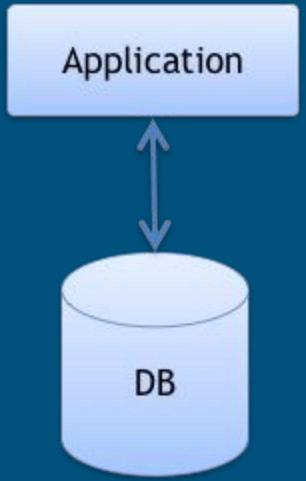


RDBMS performance



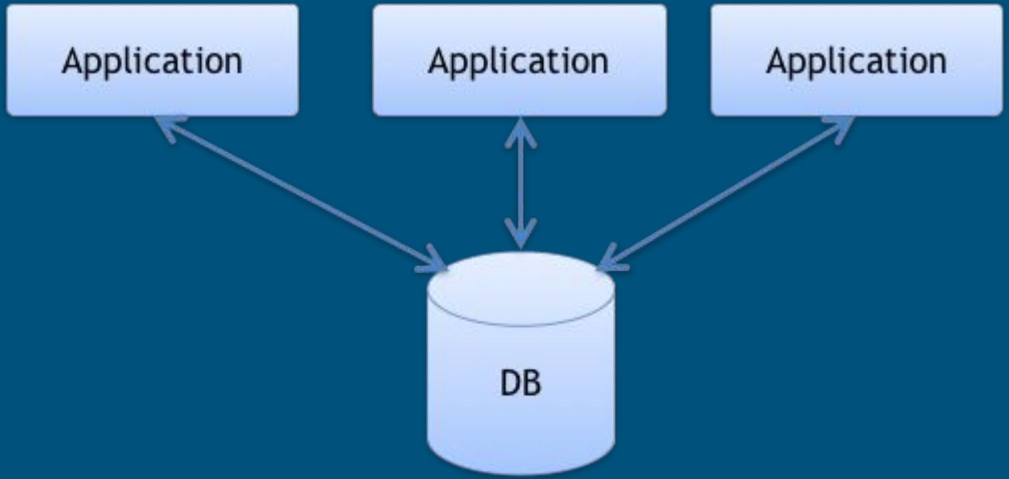
Why NoSQL Database

- 1980’: Single Application



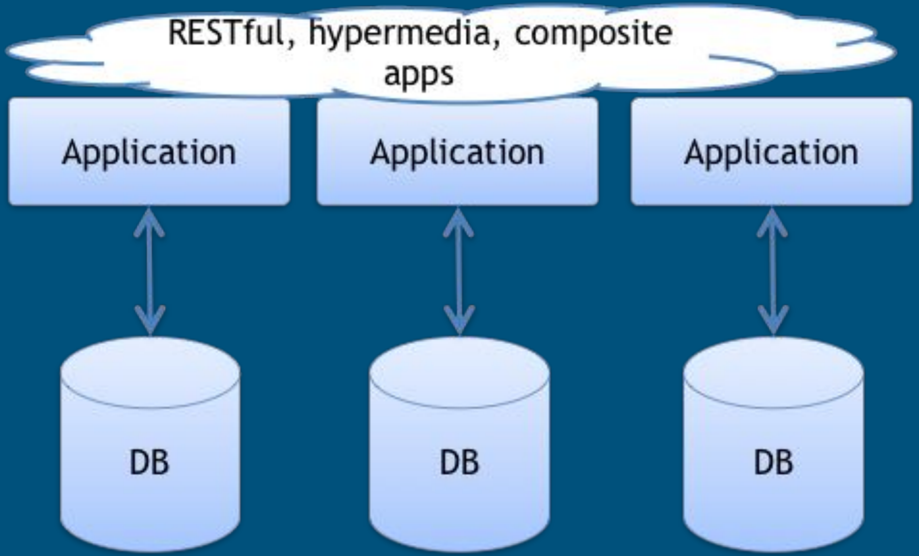
Why NoSQL Database

- 1990's: Integration Database



Why NoSQL Database

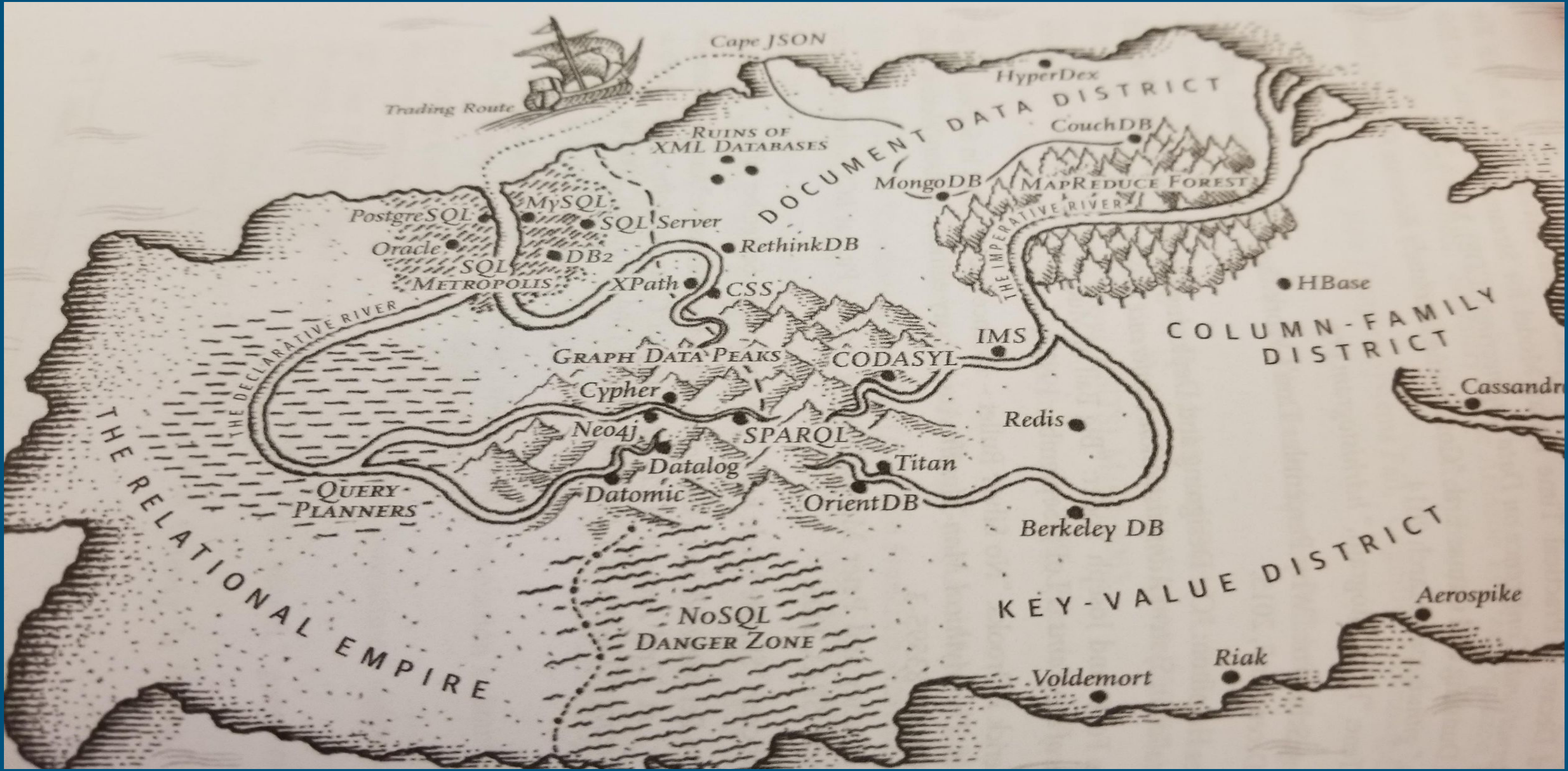
- 2000’ ~ Service-based



We start to have multi-sourced data, which means impossibility of maintain one common structure

The DB World Map

Book: Design Data-Intensive Applications



NoSQL

Not Only SQL

NoSQL Vendors

Key-Value Stores

- Come from a research article by Amazon (Dynamo)
  - Global Distributed Hash Table (Key-Value Stores)
- Popular Vendors
  - Redis (Open Sourced)
  - Amazon's DynamoDB (the inventor)
  - Microsoft Azure Cosmos DB

Everything in Key-Value

- Friendship of Facebook
  - Relational Database:
    - People (Id, Name, ...)
    - Friend (Pid1, Pid2, Time)
  - Key-Value Store:
    - Profile KV: <Id; Name>
    - Friendship KV: <Id; Set{(Id1, Time1), (Id2, Time2), ...}>
- When we want to get all friends
  - Relational Database: Join People with Friend (costly)
  - Key-Value Store: Get directly from the Friendship KV (O(1))

Key-Value Store

- Why
  - Simple Data Model: Hash Table is mature data structure
  - Good Scalability: Small System Cost, via good look-up locality and caching
- Why not
  - Poor to complex (interconnected) data
    - many KV pairs needed to be maintained for each data
    - hash-table-like structure tends to performance poorly for large data

Column Family

- Origin from Google's BigTable
- Main Idea
  - Each table tends to have many attributes (thousands ~ millions)
  - In most applications (analytics) we are only interested in a few (10s)
  - Traditional raw-based
    - Store the each record in a sequential file
    - Even just read one attribute, we should read the whole record
  - Column-based
    - Store the data by putting the same attribute in a sequential file
    - Faster access to a few attributes

Column Family

- Google’s BigTable
  - Drives MapReduce
  - Apache Hadoop, Hadoop File System (HDFS), HBase
  - Apache Cassandra

Column Family

- Why?
  - Optimized for data analytics
  - Semi-Structured Data: Each column can define its own schema
  - Big Data
- Why not?
  - Nightmare for interconnected data
    - Suppose I have an Friendship table (pID1, pID2, ...) stored based on column
    - Want to find how many steps from me to “Donald Trump”
    - Sequentially scan the “pID1“ and “pID2” column again and again

Others

- Document-based DB
  - XML
  - MangoDB (one of most popular DBs)
  - CouchDB
- Specific-purpose
  - RDF: 3-tuple (obj1, <action>, obj2), e.g. (John, add_friend, Emily)
  - To resolve the problem of maintaining interconnected data.
  - But it does not give the ultimate solution

Graph Database

- Data Model
  - Nodes (Vertices) -> Entities
  - Edges -> Relations
  - Are we going to learn ER model again?
- Main Vendors
  - Neo4j
  - JanusGraph
  - OrientDB
  - Gremlin

Graph Database

- Why
  - Natural fitness for data with complex connections
  - Powerful data model, as general as relational model
  - Flexible query: based on many graph algorithms
- Why not
  - Not easy to scale, because of **poor locality**
  - Most algorithms are computationally intensive

Graph is not just everywhere, it is $$$

Recommendations

Social Computing

Geospatial (Google map)

Internet of things

Web Analytics

Bioinformatics ..

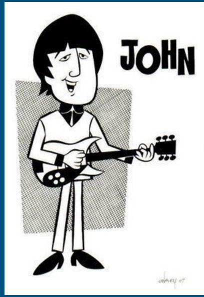
Market Cap: US$2 Trillion

Graph Database

What is a Graph Data Structure

- Node / Vertex (id, name, age) -> Entity
- Edge / Link / Arc (srcId, dstId, timestamp) -> Relation

srcId: 1
dstId: 2
married: 1969

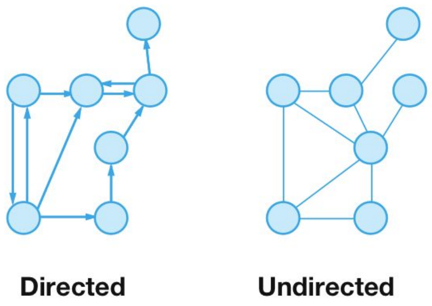


id: 1,
name: John Lennon
prof: Musician

id: 2,
name: Yoko Ono
prof: Musician

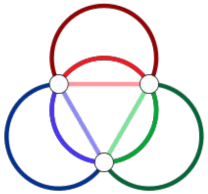
Graph Types (Edge Types)

- **Directed Graph**
- **Undirected Graph**



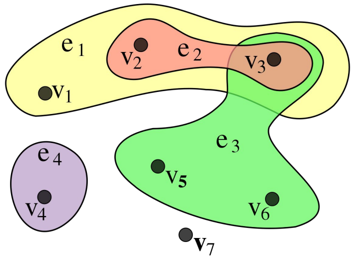
Edge: Two nodes
Two nodes: at most one edge

- Multi Graph



Edge: Two nodes
Two nodes: >= one edge

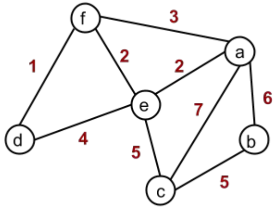
- HyperGraph



Edge: >= two nodes
Two nodes: >= one edge

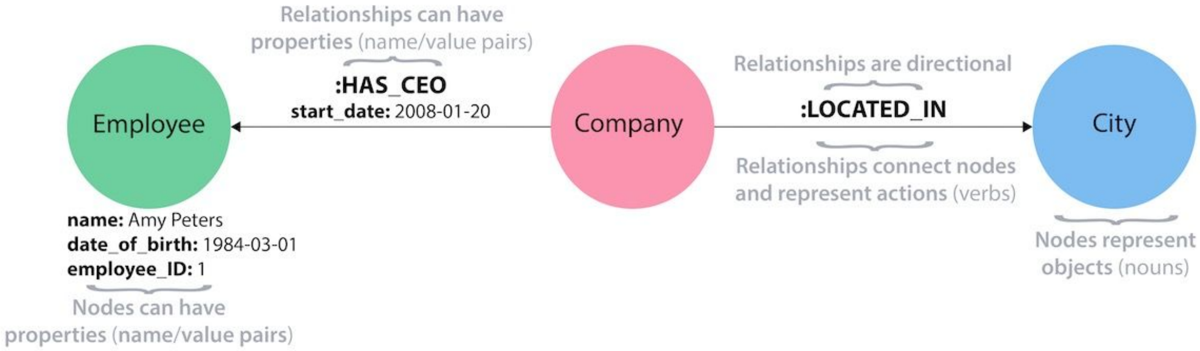
Graph Types (Attributes)

- Weighted Graph



Every edge has a number as its "**weight**"

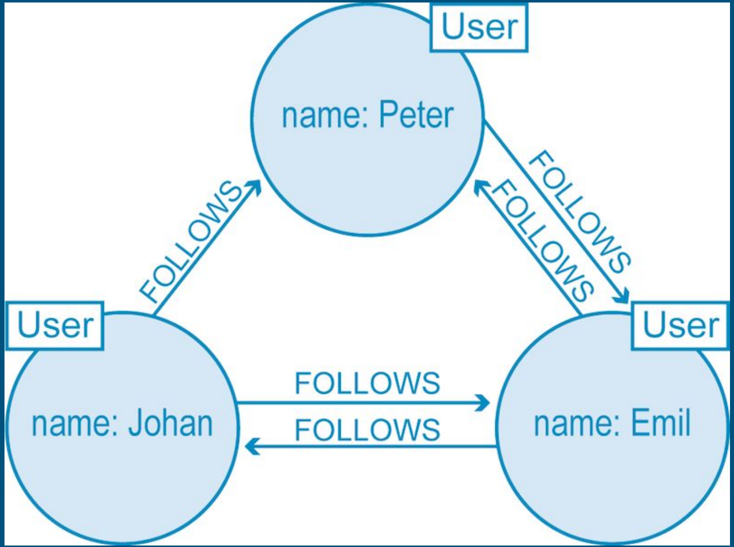
- **Property Graph**

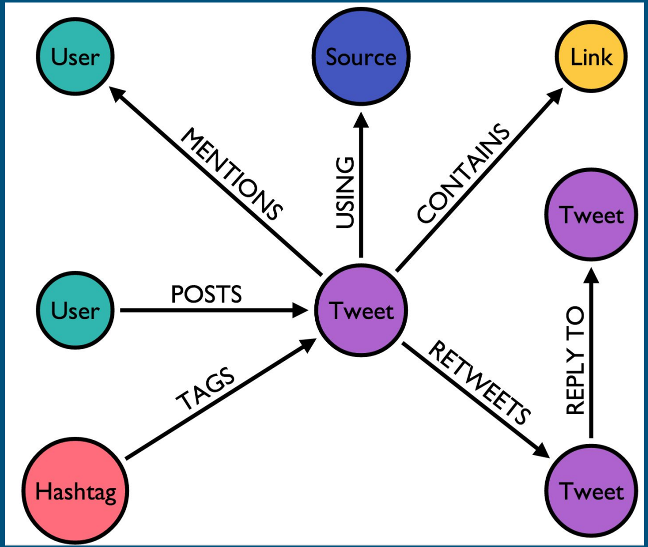


Properties

- Analogous to Attributes in relational table
- Both Nodes and Edges can have properties
- Properties are key-value pairs
  - e.g. “name”: John, “prof”: musician, “join_day”: 09/04/2019
- Properties are more flexible than Attributes
  - Primitive type: String, Integer, Char, Boolean, ...
  - Array: int[ ], String[ ]
  - Set: int{ }, String{ } // Set can not contain duplicate elements

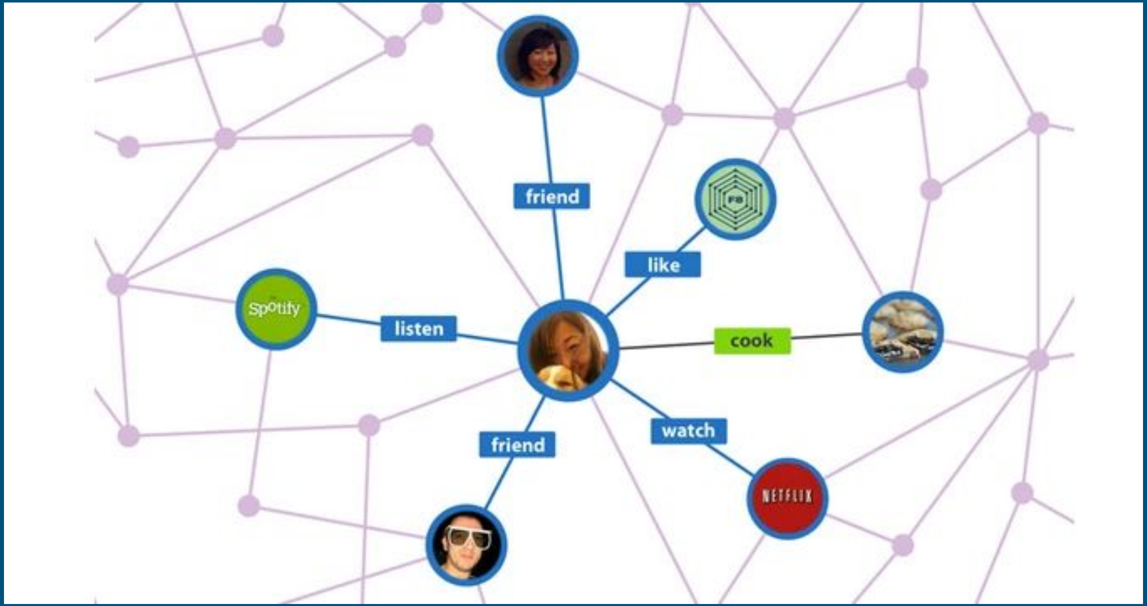
Twitter Graph





“following”/”followed by” indicates directions

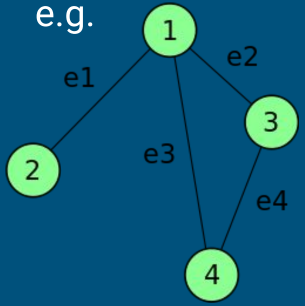
Facebook Graph - undirected



“friend” is a mutual relationship, which indicates undirection

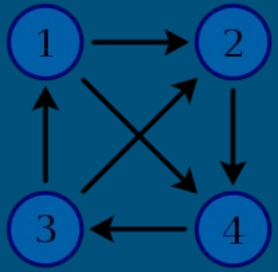
Graph database

- A database based on an explicit graph structure
- Every node maintains not only its properties, but its adjacent (neighbor) nodes
  - Adjacent nodes are nodes that I connect with
  - e.g.



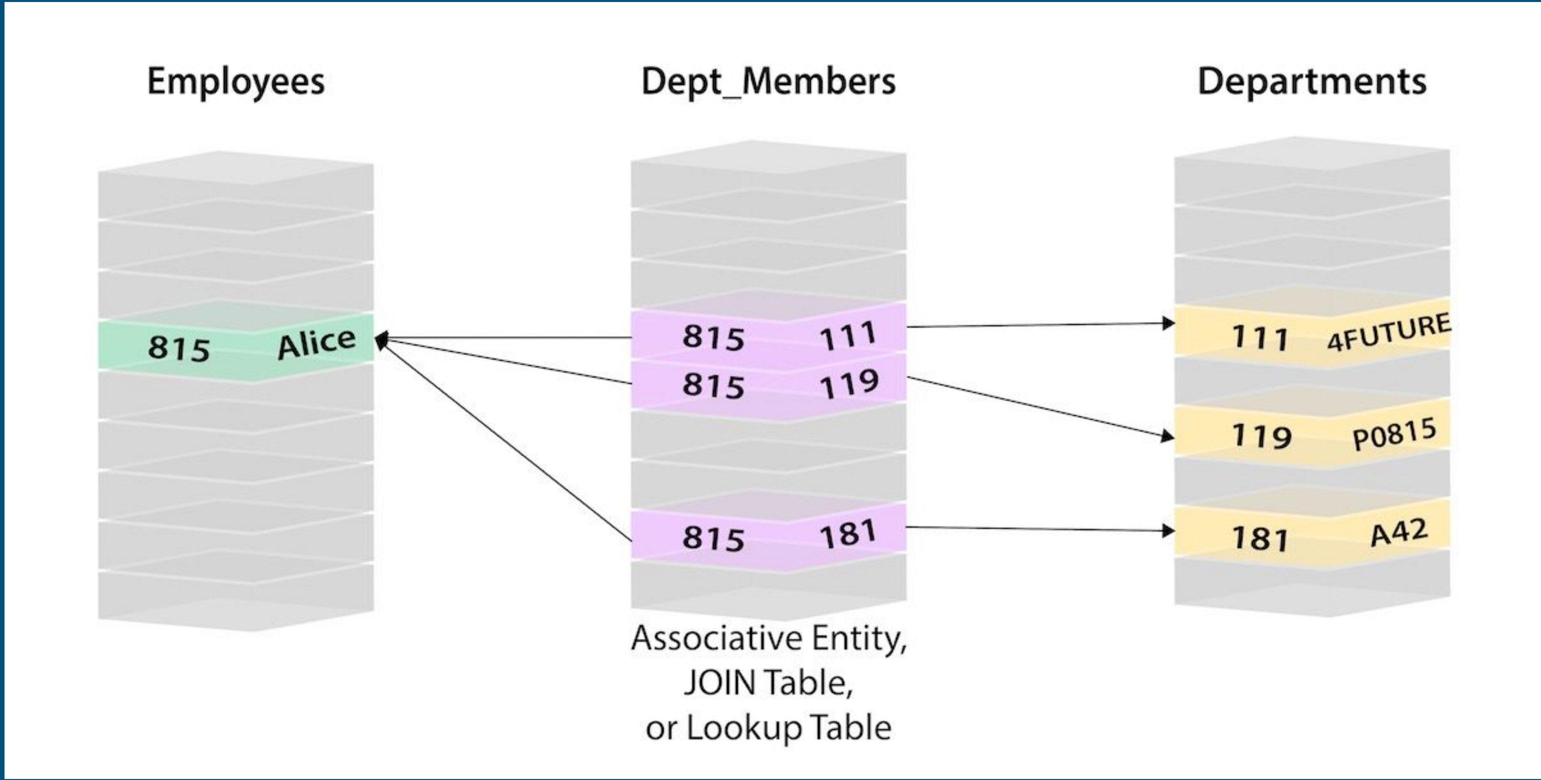
1: {2, 3}
2: {1}
3: {1, 4}
4: {1, 3}

Directed Graph:
In neighbors and out neighbors

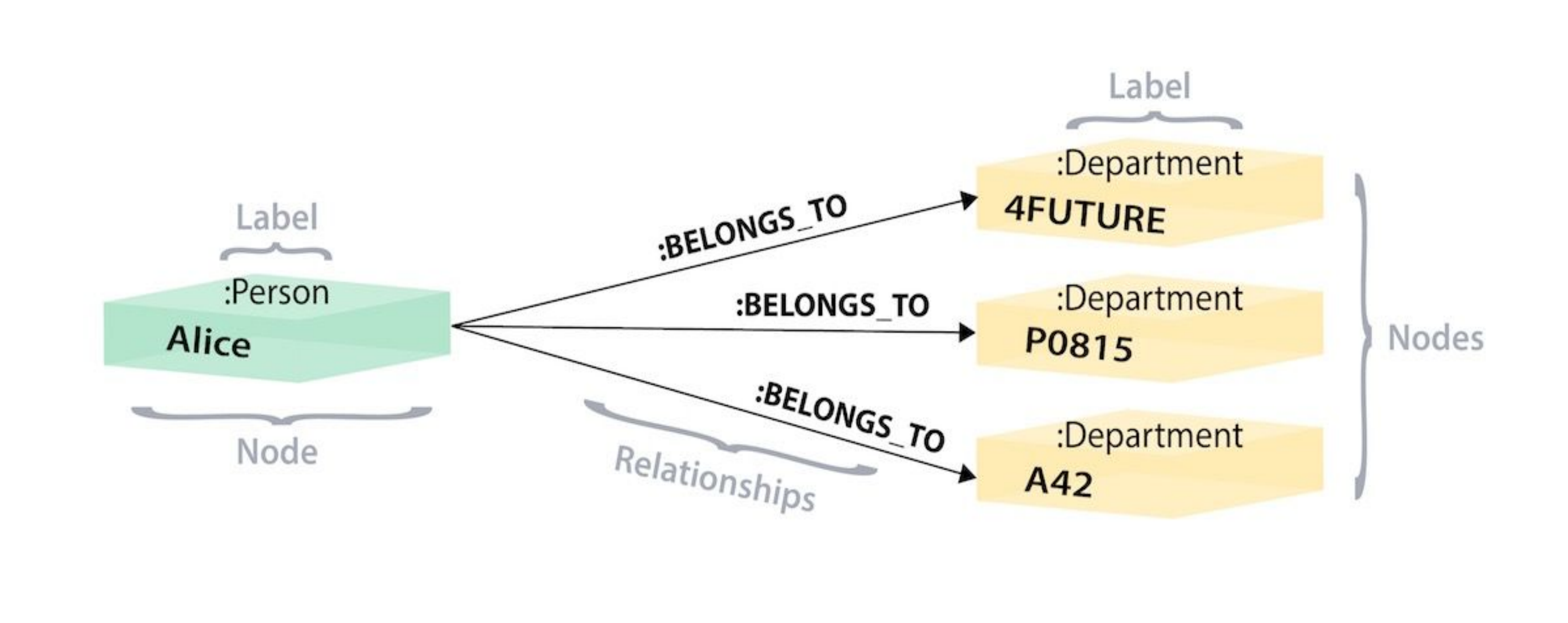


1: Out {2, 4}, In {3}
2: Out {4}, In {1, 3}
3: Out {2}, In {4}
4: Out {3}, In {1, 2}

Relational Databases



Graph Database



Graph DB for Connection Analytics

- Find out among my friends' friends, who are also my friends?
- Solving with relational database
  - `People (Id, name)`
  - `Friend (srcId, dstId, ...)`
  -
  - `SELECT Distinct(p2.name)`
  - `FROM People p1, People p2, Friend f1, Friend f2, Friend f3`
  - `WHERE  f1.dstId = f2.srcId  AND              # Get friend`
  - `                 f2.dstId = f3.dstId  AND    # Get friend's friends`
  - `                 f1.srcId = f3.srcId  AND    # Check also my friend`
  - `                 f1.srcId = p1.id     AND`
  - `                 p1.name = "MY_NAME"         # check it is me`
  - `                 f3.dstId = p2.id     AND    # get friend's name`
  -

Graph DB for Connection Analytics

- Find out among my friends' friends, who are also my friends?
- Solving with graph database
    - `Node (srcId, name, nbrs(friends): [nbr1, nbr2, nbr3, ...])`
    -
    - `Set {}             # Initialize an empty set`
    - `GET my_node from Graph DB       # Get the node from Id`
    - `FOR EACH nbr IN my_node.nbrs:  # Outer: Check all my friends`
    - `    FOR EACH nbr_nbr IN nbr.nbrs: # Inner: Check my friend's friends`
    - `        IF nbr_nbr IS IN my_node.nbrs:  # Check also my friend`
    - `            Set.insert(nbr_nbr)`
    - `RETURN Set`

Graph DB for Connection Analytics

- Find out the shortest path from me to Donald Trump in the Twitter network
  - A Path: A sequence of nodes where there is an edge in between
    - e.g. A -> B -> C -> D -> E (path of length 4)
  - A shorted path: For every such path, the one with shortest length
    - e.g. A -> B -> C -> D -> E -> F
    - A -> B -> C -> F **Shorter one**
-
- Use everyday
  - Navigation (Road)
  - Routing (Network)
  -

Graph DB for Connection Analytics

- Find out the shortest path from me to Donald Trump
- Solving with Relational DB
  - STEP 1:
  - SELECT COUNT(*)
  - FROM friend f1
  - WHERE f1.srcId = myId AND f1.dstId = TrumpId
  -
  - IF COUNT = 0 -> STEP 2:
  - SELECT COUNT(*)
  - FROM friend f1, friend f2
  - WHERE f1.srcId = myId AND
  -         f1.dstId = f2.srcId
  -         f2.dstId = TrumpId

Goes until COUNT != 0,
When completes?

Graph DB for Connection Analytics

- Single-Source Shortest Path Problem
  - Given a source node of the graph, find the shortest path of **all** nodes to this source node
  - Nightmare to solve using relational DB
  - Well-defined graph algorithm
    - **Breadth-First Search (BFS)**: When all edges have the same weight value
    - **Dijkstra Algorithm**: When all edges have some **positive** weight
    - Bellman-Ford Algorithm: General case
  - Assume Directed Graph G
    - Each node {id: ID, dist: FLOAT, out_nbrs: [nodes], is_visited: BOOLEAN }
    - Each edge { srcId: ID, dstId: ID, weight: FLOAT }

Breadth-First Search (BFS)

- Queue
  - First In First Out (FIFO)
  - EnQueue(v): Put an element into the queue
  - DeQuene(): Return and remove the top element in the queue
  - e.g.

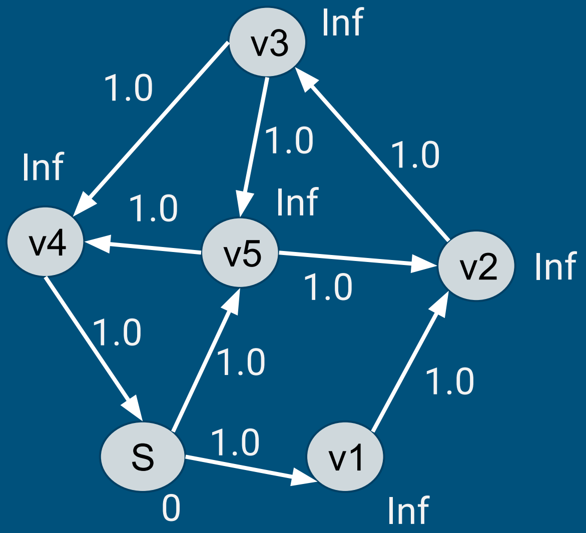
1. Q.EnQueue(v1), Q.EnQueue(v2), Q.Enqueue(v3) — v3 v2 v1
2. Q.DeQueue -> v1 — v3 v2 v1
3. Q.DeQueue -> v2 — v3 v2
4. Q.EnQueue(v4) — v4 v3
5. Q.DeQueue -> v3 — v4 v3
6. Q.DeQueue -> v4 — v4

Breadth-First Search

- Initialize every node’s distance as **Infinity**
- Make the source node’s distance as 0
- Q.EnQueue(s) // s is the source node
- Loop until Q is empty
  - v = Q.DeQueue()
  - for each nbr in v.out_nbrs():
    - if nbr.dist > v.dist + e(v, nbr).weight:
      - nbr.dist = v.dist + e(v, nbr).weight (=1)
      - if not nbr.is_visited: Q.EnQueue(nbr)
  - Make v as visited: v.is_visited = true
- EndLoop

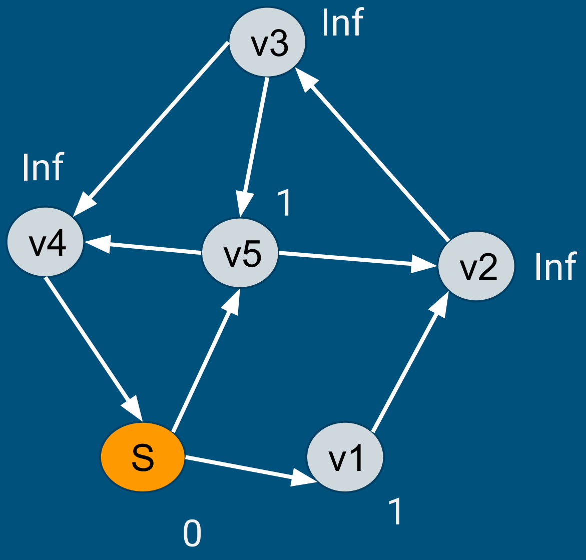
Breadth-First Search

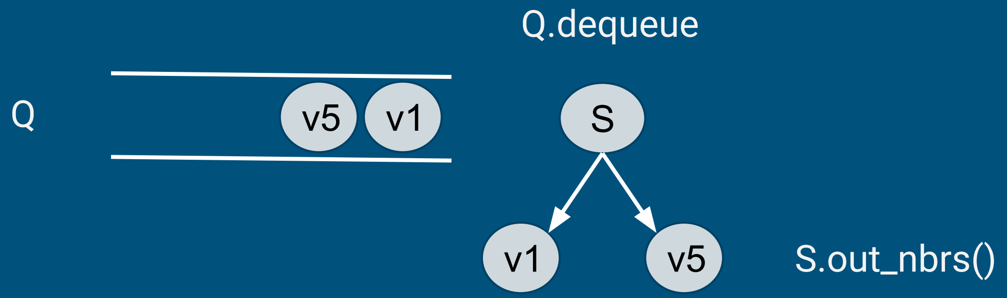
Initialization



Breadth-First Search

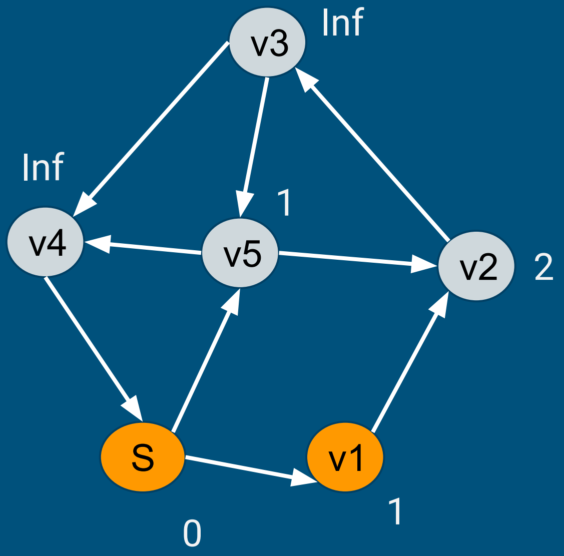
Loop1

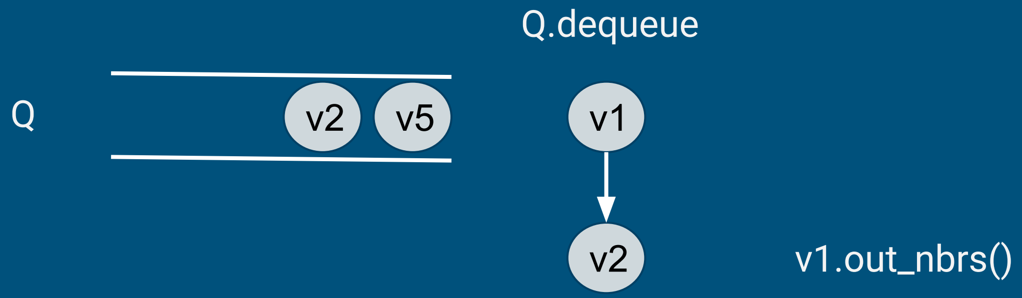




Breadth-First Search

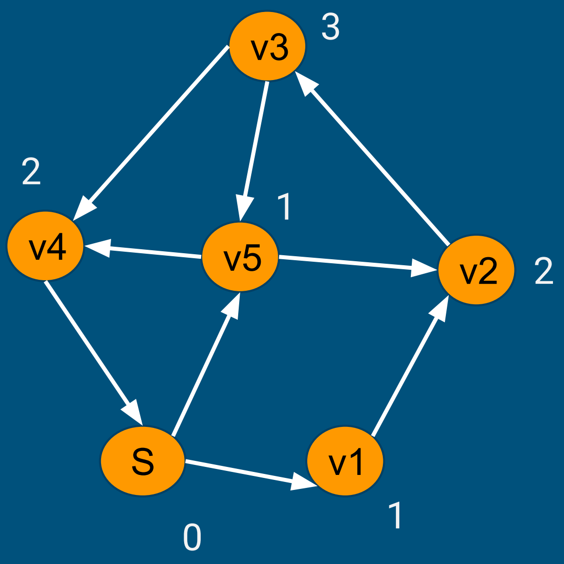
Loop2





Breadth-First Search

Loop3~6



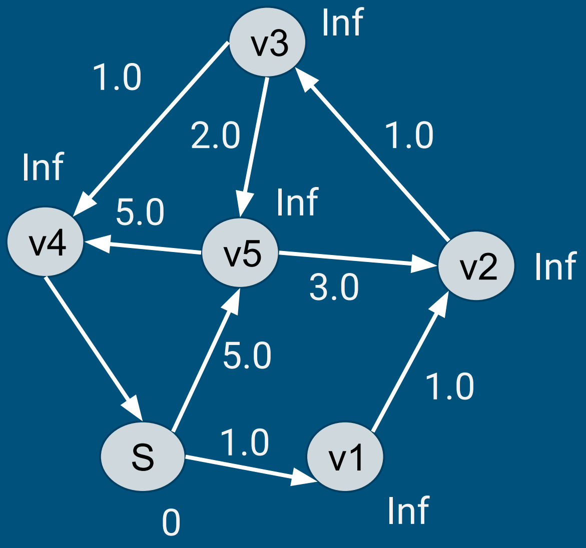
Loop3: DeQueue v5, update v4 {2}, EnQueue v4

Loop4: DeQueue v2, update v3 {3}, EnQueue v3

Loop5: DeQueue v4, no ACT

Loop6: DeQueue v3, no ACT

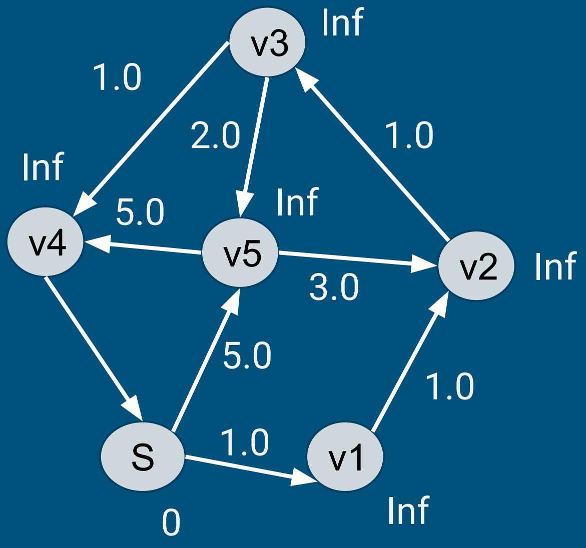
Breadth-First Search



If Edge does not have the same weight value,
BFS will compute **wrong** results?

Practice !!

Dijkstra Algorithm



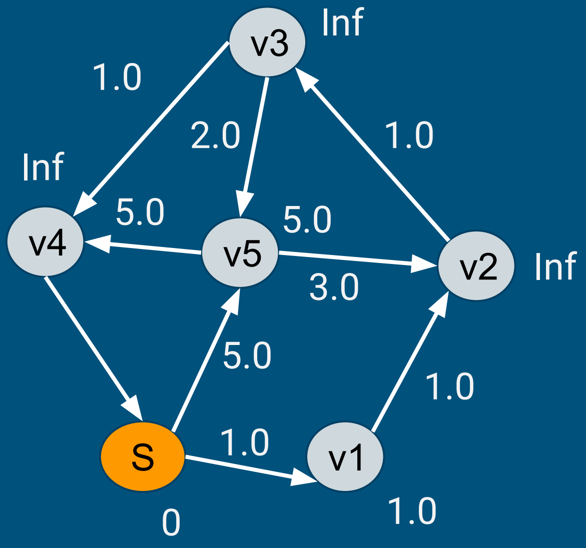
Dijkstra Algorithm: Queue -> PriorityQueue

Priority Queue

- Each element has a key
- DeQueue the element with the **smallest** key, rather than FIFO

Dijkstra Algorithm

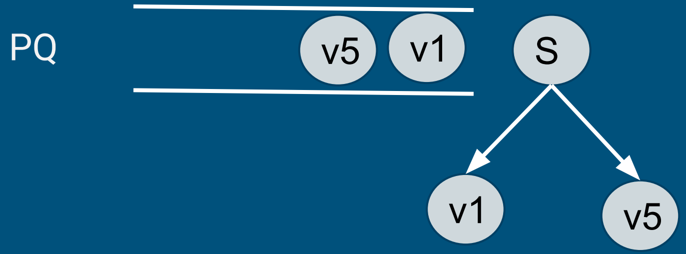
Loop1



Dijkstra Algorithm: Queue -> PriorityQueue

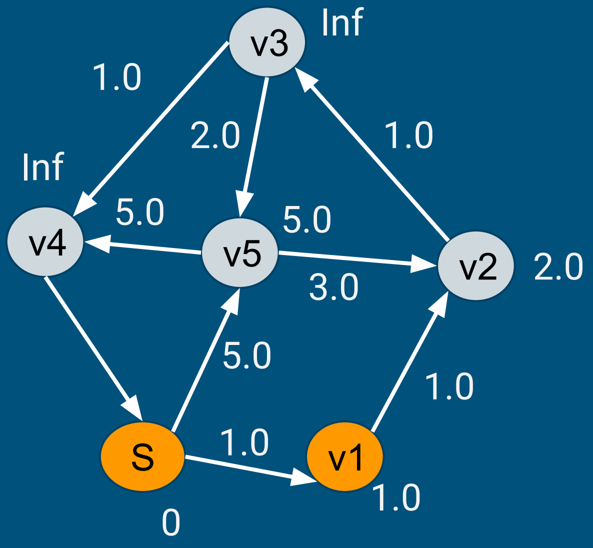
Priority Queue

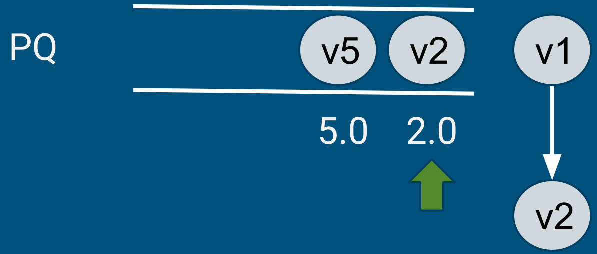
- Each element has a key
- Dequeue the element with the **smallest** key, rather than FIFO



Dijkstra Algorithm

Loop2





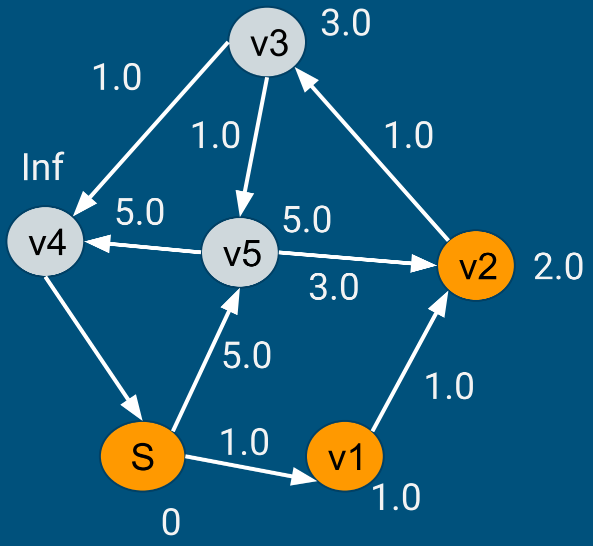
Dijkstra Algorithm: Queue -> PriorityQueue

Priority Queue

- Each element has a key
- Dequeue the element with the **smallest** key, rather than FIFO

Dijkstra Algorithm

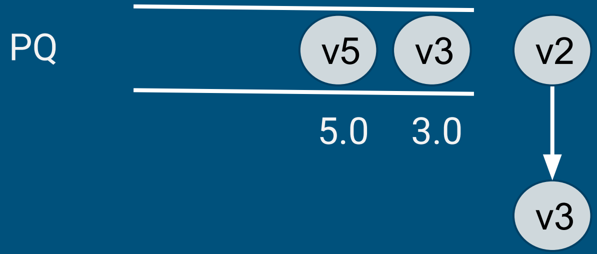
Loop3



Dijkstra Algorithm: Queue -> PriorityQueue

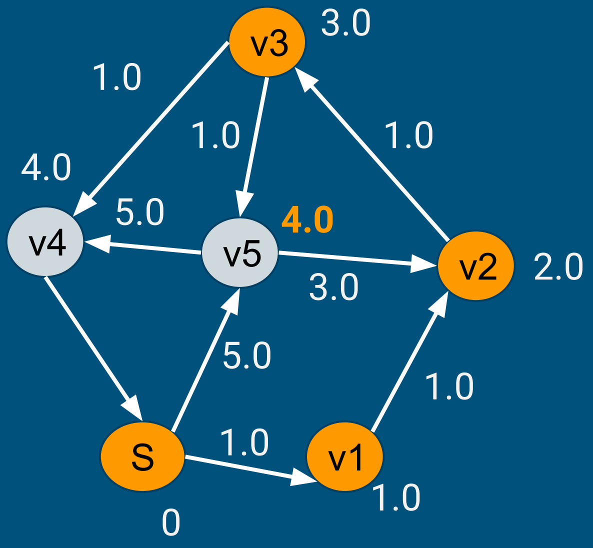
Priority Queue

- Each element has a key
- Dequeue the element with the **smallest** key, rather than FIFO



Dijkstra Algorithm

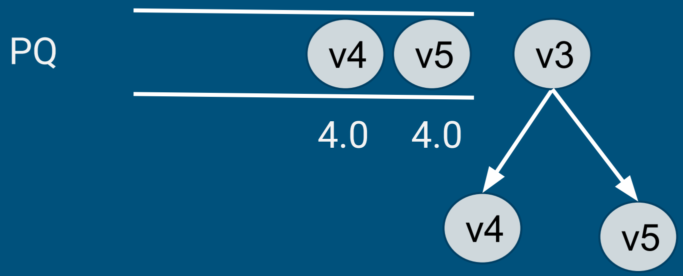
Loop3



Dijkstra Algorithm: Queue -> PriorityQueue

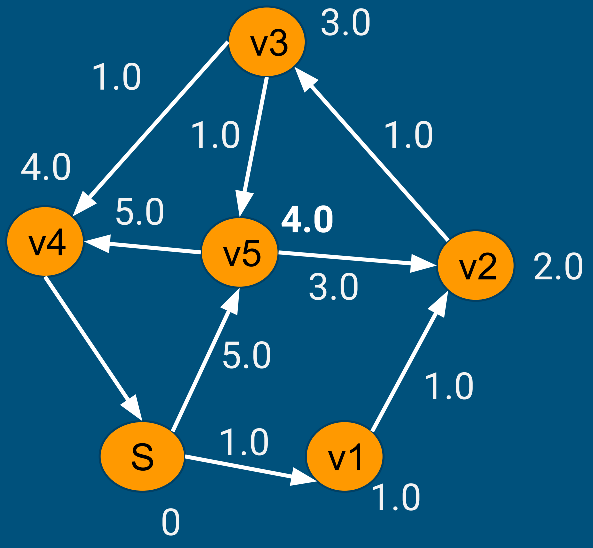
Priority Queue

- Each element has a key
- Dequeue the element with the **smallest** key, rather than FIFO



Dijkstra Algorithm

LoopEnd



Dijkstra Algorithm: Queue -> PriorityQueue

Priority Queue

- Each element has a key
- Dequeue the element with the **smallest** key, rather than FIFO

Summarizations

Summarizations

- Graph DB is NoSQL DB
- Graph DB's main structure
  - Node: Define entity, with properties (attributes), **nbrs (in_nbrs, out_nbrs)**
  - Edge: Define relationships (connections), directed or undirected
- Graph DB is specially designed for Connection Analytics
  - Friend's friends are friends (Triangle Listing)
  - Single-source shortest path (SSSP)

Next Class

- Cypher Query Language in Neo4J
- Distributed Graph Processing
  - Pregel: Vertex-centric Computation Model
  - Some typical algorithms: Triangle Listing, PageRank, SSSP, etc.