Graph Databases

ANNA QUERALT, OSCAR ROMERO

(FACULTAT D’INFORMÀTICA DE BARCELONA)

Graph Databases

A (native) graph database:

- Provides means to efficiently **process** graph data
  - **Index-free adjacency**
  - Note this does not diminish the possibility to plug a storage system with an external processing system
- Provides means to **store** graph data
  - Each having potentially different physical graph data models
- Examples: Neo4j, Titan

(Distributed) Graph frameworks typically refer to processing frameworks. Thus, like MapReduce or Spark, provide means to extract data from databases **BUT DO NOT STORE GRAPHS**

- Examples: Pregel (Google), Giraph (MapReduce), GraphX (Spark), …

Implementation of Graphs

[Sakr and Pardede 2012]

Adjacency List

- For each node a list of neighbors.
- If the graph is directed, adjacency list of i contains only the outgoing nodes of i.
- Cheaper for obtaining the neighbors of a node.
- Not suitable for checking if there is an edge between two nodes.

Incidence List

- Vertices and edges are stored as records of objects.
- Each vertex stores incident edges.
- Each edge stores incident nodes.

Adjacency Matrix

- Bidimensional graph representation.
- Rows represent source vertices.
- Columns represent destination vertices.
- Each non-null entry represents that there is an edge from the source node to the destination node.

Incidence Matrix

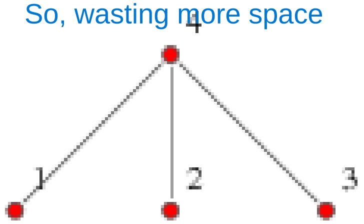
- Bidimensiona l graph representatio n.
- Rows represent Vertices.
- Columns represent edges
- A non-null entry represents that the source vertex is incident to the Edge.

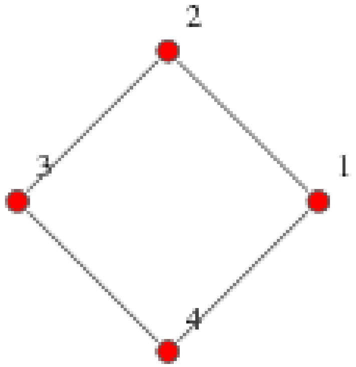
Implementation of Graphs

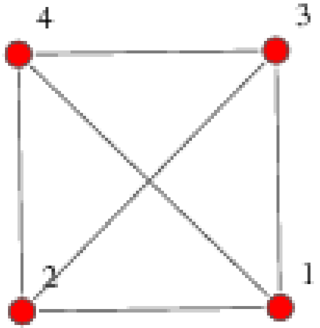
Cost of adjacency is as much as adjacent node

Adjacency matrix (baseline)

Incident matrix in pic
For each edge 2 1's and all 0
So, wasting more space







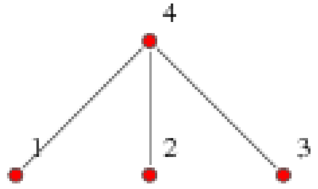
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

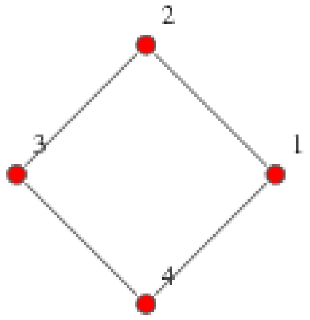
Activity

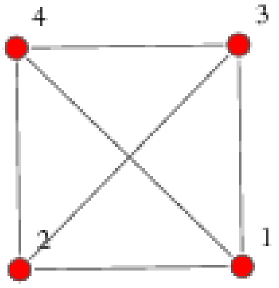
Objective: Understand the different structures needed to implement graphs following the main graph implementation strategies

Cannot add properties here

- *Implement the following graphs as an adjacency list **AND** as an incidence list*







$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Implementation of Graph Databases

NEO4J

Neo4J native graph storage

Based on **incidence lists**

- Implemented by means of singly and doubly linked list structures

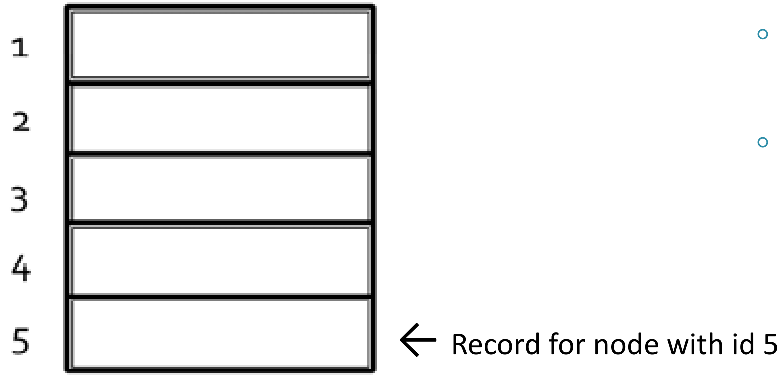
Separate files for each different part of the graph

- Nodes
- Relationships
- Properties
- Labels
- Values

Everything is stored in separated file

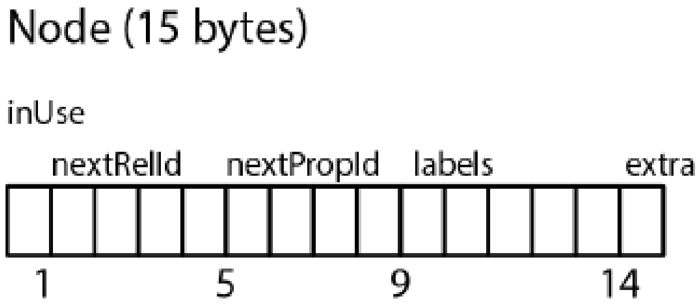
Least Frequently Used (LFU) Cache Policy:
The LFU cache policy is a cache eviction strategy that prioritizes removing the least frequently accessed items from the cache when space is needed. In the context of Neo4j, this means that nodes that are accessed less frequently are more likely to be evicted from the in-memory cache to make room for new or more frequently accessed nodes.

Incidence Lists – Neo4J



Nodes

- One physical file to store all nodes (in-memory, *Least Frequently Used* cache policy)
- **Fixed-size** record: 15 bytes in length easy to find required node in single operation
  - Fast look-up: O(1)



- Each record is as follows:
- Byte 1 (metadata; e.g., in-use?) whether node is in use or not
  instead of deleting node, it is marked here as not used
- Bytes 2-5: id first relationship first edge of this node
- Bytes 6-9: id first property first property of this node
- Bytes 10-14: labels labels of that node
- Byte 15: extra information

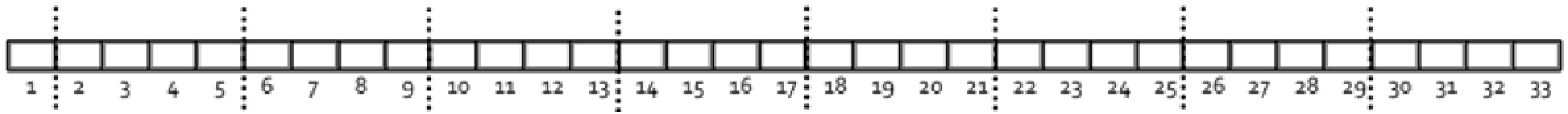
Incidence Lists – Neo4J

Relationship and property files.

- Both contain records of **fixed size**
- Cache with *Least Frequently Used* policy

Relationship file

- Contents of each record:
  - Metadata
  - id starting node, id end node
  - id label label of an edge in the form of id
  - ids of the previous and following relationship of the starting node and of the ending node
  - id first property id of first property of edge
- Doubly linked list



Incidence Lists – Neo4J

Relationship and property files.

- Both contain records of **fixed size**
- Cache with *Least Frequently Used* policy

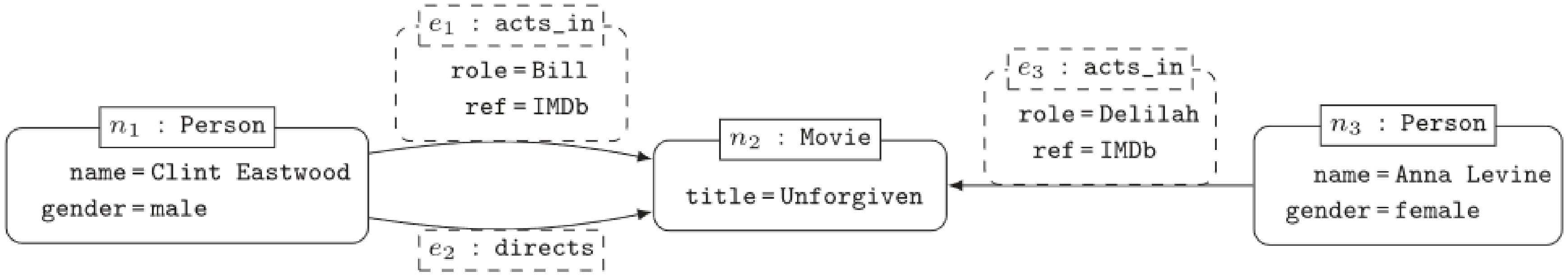
Property file

- A single file for all properties, regardless if they belong to nodes or to edges.
- Contents of each record:
  - Metadata (incl. a bit determining whether it belongs to an edge or node)
  - id node / edge the metadata field has a bit indicating whether the property belongs to a node or a relationship.
  - id of the following property of the node / edge previous property needed or not?
  - id property name
  - id property value
- Singly linked list

Activity

Objective: Understand how to implement a linked list to implement graphs

Consider the graph below. What would be the resulting data structures if you create such graph in Neo4J?



Types of graph databases

Types of Graph Databases

Some graph databases / processing frameworks are based on strong assumptions that are not explicit

- As a consequence of how they implement internal structures

Operational graphs:

- Map to the concept of a CRUD database
  - Nodes, edges can be deleted, updated, inserted and read
  - Example: Neo4j, Titan, OrientDB, Amazon Neptune, ...

Analytical graphs

- They are *snapshots* that cannot be modified by the final user
  - Equivalent to a data warehouse for graphs
  - Example: Sparksee, Giraph, GraphX, etc.

Summary

Unfortunately, there is no standard (yet) to implement graph databases

SQL on other hand has some standard like SQL99

They all follow the same principles, but the way to implement it really affects graph processing

When choosing a graph database, consider:

- Operational vs. Analytical graph database
- Internal data structures
- Impact of the internal data structures on the required graph processing for your project