Graph Databases

Francisco Moreno

(with extracts from different sources)

Introduction

- What is it that makes graphs not only relevant, but necessary in today's world? → Its focus on relationships.
- We are gathering more data than ever before, but more and more frequently it is how that data is related that is truly important.
- Obviously, graph databases (GDBs) are particularly suited to model situations in which the information is somehow "natively" in the form of a graph.

- GDBs are useful in understanding big datasets in scenarios as diverse as logistics route optimization, retail suggestion engines (recommendation systems), fraud detection, social networks, authorization and access control systems, among others.
- The success key of GDBs in these contexts is that they provide **native** means to represent relationships.
- Relational databases (RDBs) instead simulate relationships through the help of foreign keys, thus adding additional development and maintenance overhead, and “discover” them require costly join operations.

Why We Should Care about GDBs?

- Performance: with RDBs, relationship queries (that is *joins*) will come to a grinding halt (so-called *join pain*) as the number and depth of relationships increase. In contrast, GDBs performance stays consistent even as our data grows year over year (but performance depends on the kind of queries!!!).
- Flexibility (schema): the big data structure may change at a high rate over time. The schema of a RDB is static and has to be understood from the beginning of a database design.
- Agility (development).

What Is a GDB? (An Informal Definition)

- A graph is composed of two elements: a node (vertex) and a relationship (edge).
- Each node represents an entity (a person, place, thing, category or other piece of data), and each relationship represents how two nodes are associated. Nodes and relationships have properties (attributes).
- Twitter is a perfect example of a GDB connecting millions of monthly active users.

Graph Storage

Some graph DBMS use “native” graph storage, while others use non-native storage (e.g., RDBs or object-oriented DBs). Non-native storage is often slower than a native approach, this is because in native graph processing connected nodes physically “point” to each other in the database.

Why Data Relationships Matter?

- The greatest weakness of RDBs is that their schema is too inflexible.
- To compensate, our development team can try to leave certain columns empty (tech. lingo: nullable).
- Even worse, as our data multiplies in complexity and diversity, our RDB becomes burdened with large tables (relations) that disrupt performance (join) and hinder further development.

Why Other NoSQL Databases Do not Fix the Problem Either?

Graphs Put Data Relationships **at the Center**:

- GDBs explicitly storage relationship data.
- The flexibility of a graph model allows us to add new nodes and relationships without compromising our existing network or expensively migrating our data.
- With data relationships at their center, graphs are incredibly efficient at handling connected data

- Other non-relational databases (NRDBs) store sets of disconnected documents, values, and columns, which in some ways gives them a performance advantage over RDBs: NRDBs avoid joins organizing the data in such a way that the join is already computed, but no precomputed joins are expensive. In addition, their construction makes it harder to harness data relationships properly.
- This is not to say other NRDBs or RDBs do not have a role to play (they certainly do!), but they fall short when it comes to connected data relationships.

Why a Database Query Language Matters?

- Just as GDBs have made the modeling process more understandable for the uninitiated, so has a GDB query language made it easier than ever for the common person to understand and create their own queries.
- A single query in SQL can be many lines longer than the same query in a GDB query language like Cypher (Neo4j's language).

- Lengthy (of many lines) queries usually take more time to run, are difficult to understand, and they also are more likely to include human coding mistakes because of their complexity.
- In addition, shorter queries increase the ease of understanding and maintenance across a team of developers.

- A query language represents its model closely. That is why SQL is all about tables and joins while Cypher is all about relationships between entities.
- GDBs models, on the other hand, not only communicate how our data is related, but they also help us clearly communicate the kinds of queries we want to ask of our data model.

GDBs vs RDBs Queries

In RDBs (querying is through joins):

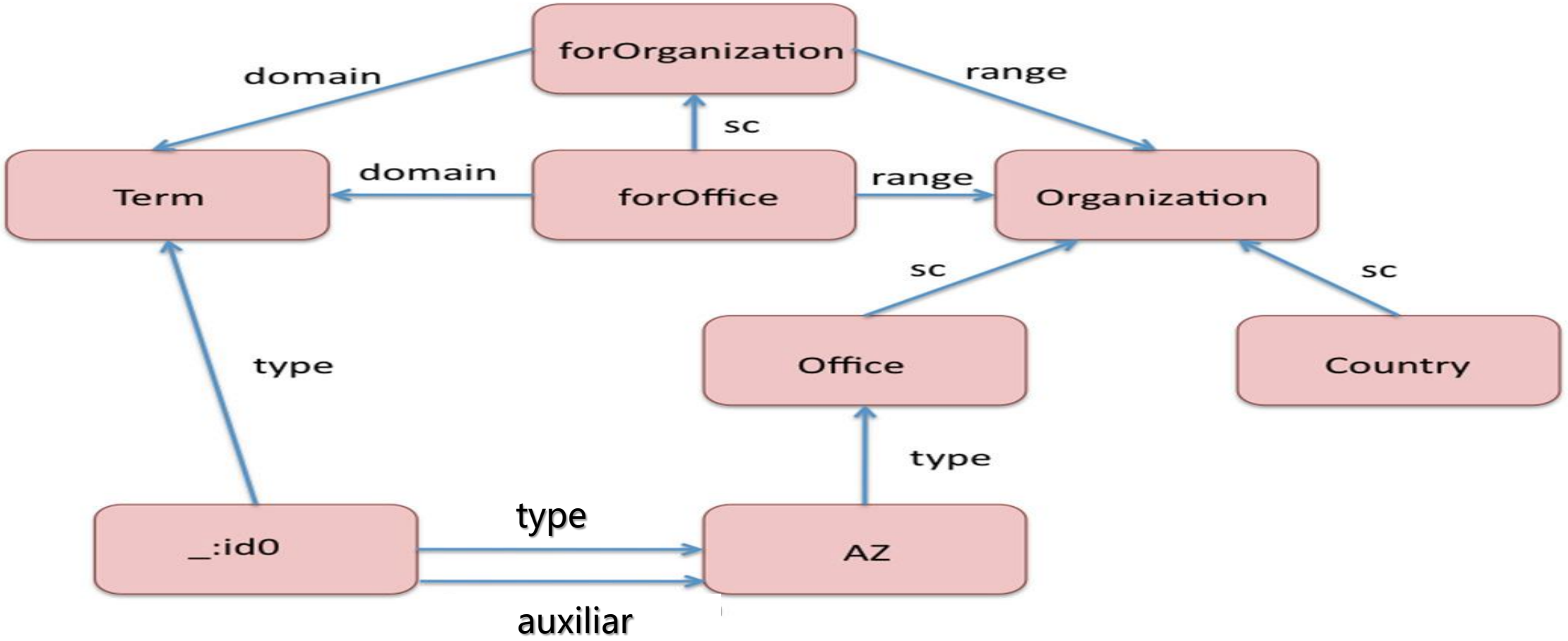
- In effect, the join operation forms a graph *that is dynamically constructed as one table is joined to another table*. While having the benefit of dynamically construct graphs, this graph ***is not explicit*** in the relational structure, but instead it must be inferred through a series of join operations.
- Moreover, while only a particular subset of the data in the DB may be desired (e.g., only Alice's friends), all data in the queried tables must be examined to extract the desired subset (although indexes may help to avoid this…)

GDBs vs RDBs Queries

In GDBs (querying is through traversal paths):

- There is no explicit join operation because nodes maintain direct references to their adjacent edges. In many ways, the edges of the graph serve as explicit, "hard-wired" join structures (i.e., structures that **are not** computed at query time as in a RDB).
- What makes this more efficient in a GDB is that traversing from one node to another is a constant time operation.

What Is a GDB? (A Formal Definition)
Abstract Data Type Multi-Graph*



*In a multi-graph two nodes can be connected by more than one edge.

What Is a GDB? (A Formal Definition) Abstract Data Type Multi-Graph

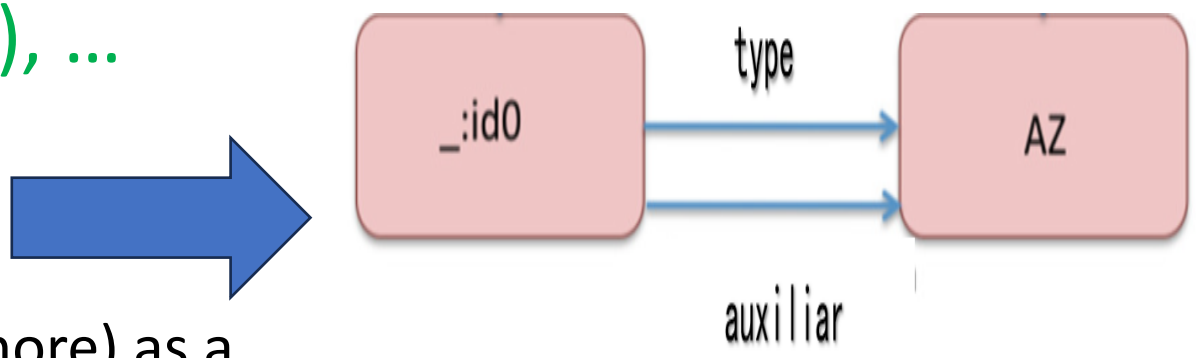
- A GDB is a set of multi-graphs.
- G = (N, E, Σ, L) is a multi-graph where:
- N is a set of nodes (vertices), e.g., N = {Term, forOffice, Organization, …}.
- E ⊆ N x N is a set of edges representing binary relationships between elements in N, e.g., E = {(forOffice, Term), (forOffice, Organization), (_id0, AZ), …}.
- A node or an edge can be associated with a set of (key, value) pairs, called properties (attributes).

What Is a GDB? (A Formal Definition) Abstract Data Type Multi-Graph

- Σ is a set of labels, e.g., Σ = {domain, range, auxiliar, type, …}.
- L is a set of labeled edges*, for the labels of an edge we have a function with signature: E → PowerSet(Σ). For example:

L = { ((forOffice, Term), {domain}),
((forOffice, Organization), {range}),
((_id0, AZ), {type, auxiliar}), …
}.

An example of two nodes connected by two edges.



*In Neo4j nodes can also have labels (zero or more) as a way of semantically categorizing the nodes in a graph.

What Is a GDB? (A Formal Definition) Abstract Data Type Multi-Graph

Given a graph G, the following are operations over G:

- AddNode(G, x): adds node x to the graph G.
- DeleteNode(G, x): deletes the node x from graph G.
- Adjacent(G, x, y): tests if there is an edge from node x to node y.
- Neighbors(G, x): nodes y s.t. there is an edge from x to y.
- AdjacentEdges(G, x, y): set of labels of edges from x to y.

What Is a GDB? (A Formal Definition)
Abstract Data Type Multi-Graph

- Add(G, x, y, l): adds an edge between x and y with label l.
- Delete(G, x, y, l): deletes an edge between x and y with label l.
- Reach(G, x, y): tests if there is (at least) a path from x to y.
- Path(G, x, y): a (the shortest) path from x to y.
- 2-hop(G, x): set of nodes y s.t. there is a path of length 2 from x to y, or from y to x.
- n-hop(G, x): set of nodes y s.t. there is a path of length n from x to y, or from y to x.

Querying GDBs

- A traversal refers to visiting elements (i.e., nodes and edges) in a graph in some algorithmic fashion.
- A path π in G from node n_0 to node n_m is a sequence of the form $((n_0, n_1), l_1), ((n_1, n_2), l_2), \ldots, ((n_{m-1}, n_m), l_m)$, where $((n_{i-1}, n_i), l_i)$ is an edge in L, for each $1 \leq i \leq m$ (l_i is a label of the set of labels associated to (n_{i-1}, n_i) in G).
- The label of π, denoted $\lambda(\pi)$, is the string $l_1 l_2 \ldots l_m \in \Sigma^*$, where Σ^* is the set of all strings composed from elements (labels) in Σ. In the following, for simplicity, labels are just letters.

Querying GDBs

- A **regular path query** (RPQ) is a **regular expression** over Σ (the vocabulary of edge labels). The evaluation RPQ(G) of RPQ over G is the set of pairs (u, v) of nodes in N for which there is a path π in G from u to v such that $\lambda(\pi)$ satisfies RPQ.

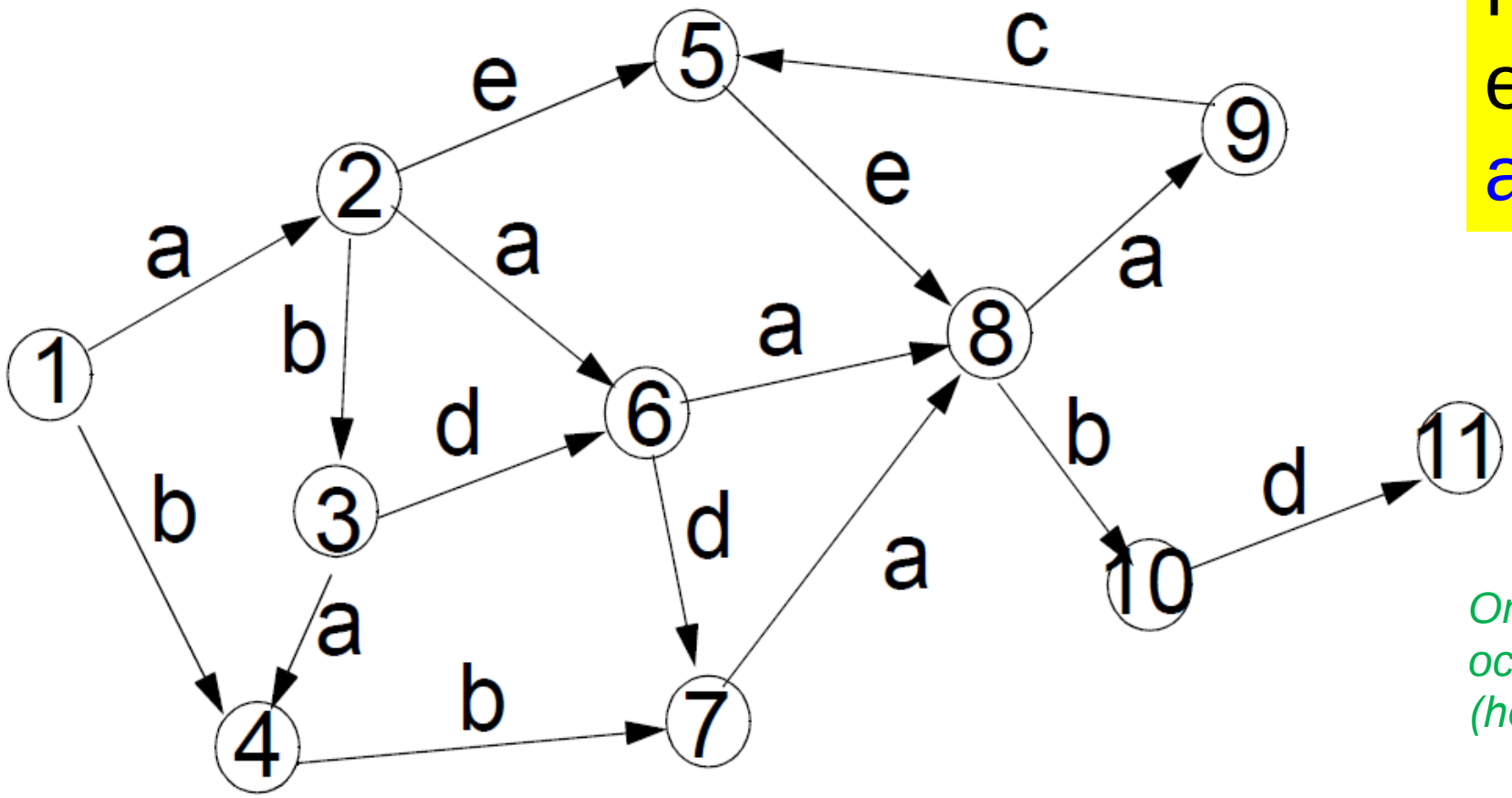
Note: RPQs do not allow to access data stored at the nodes and the edges (i.e., properties), but tailored graph query languages (as the Neo4j Cipher) exist that enable property retrieval (usually, some properties are indexed using a tree structure, B+, analogous to those used by RDBs).

- We often refer to these types of queries as pattern matching queries: We specify a pattern, **we anchor that pattern to one or more starting points**, and start looking for matching occurrences of that pattern.
- As we can see, the graph database will be an excellent tool to spin around the anchor node and figure out whether there are matching patterns connected to it.

- This, actually, is one of the key performance characteristics of a GDB: as soon as it "grabs" a starting node, the GDB will only explore the vicinity of that starting node and will be completely oblivious to anything that is not connected to the starting node.

Local search!

Querying GDBs: RPQ example

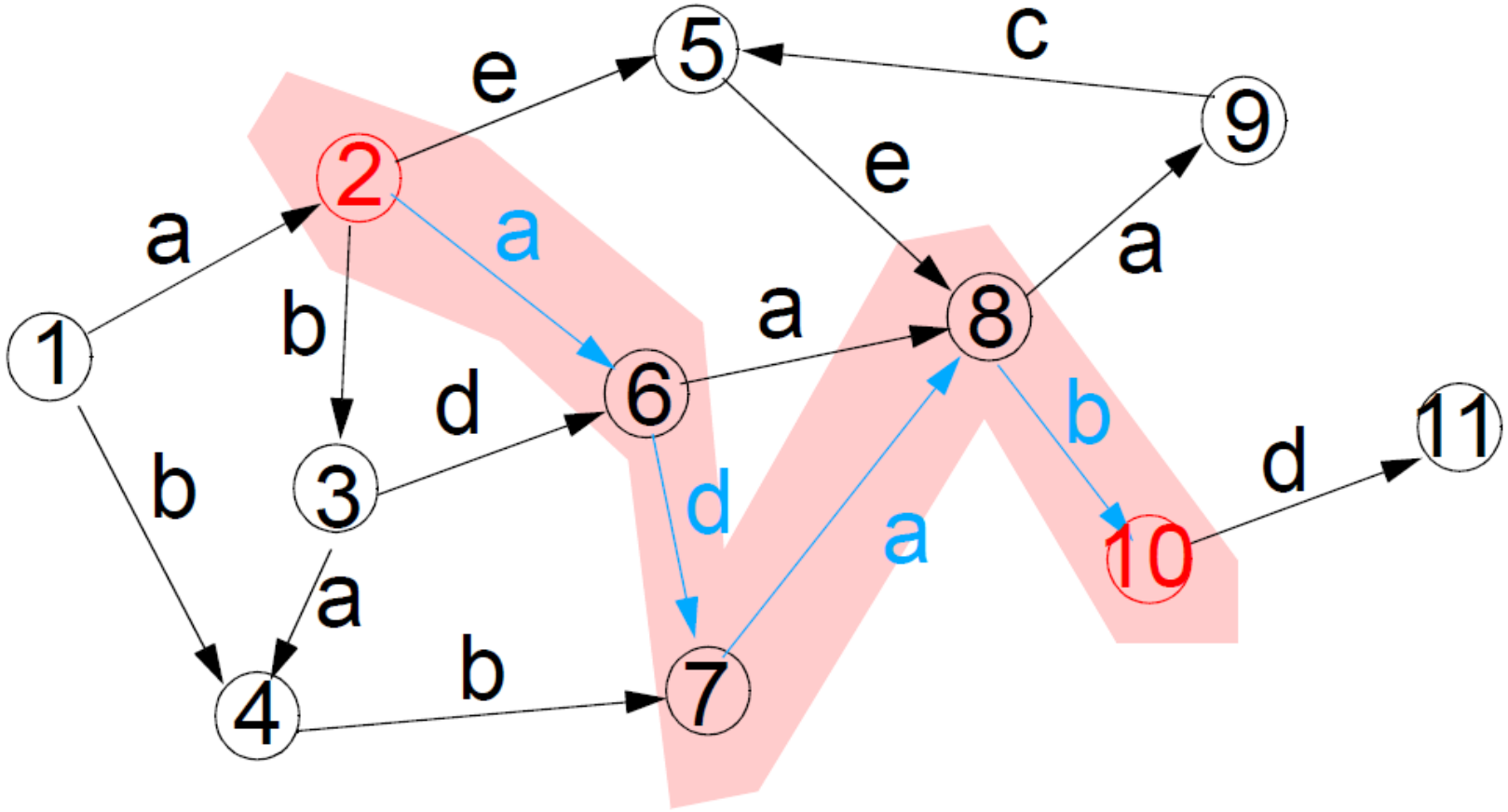


Regular expression = a+(d|b)ab

Or

One or more occurrences (here of "a")

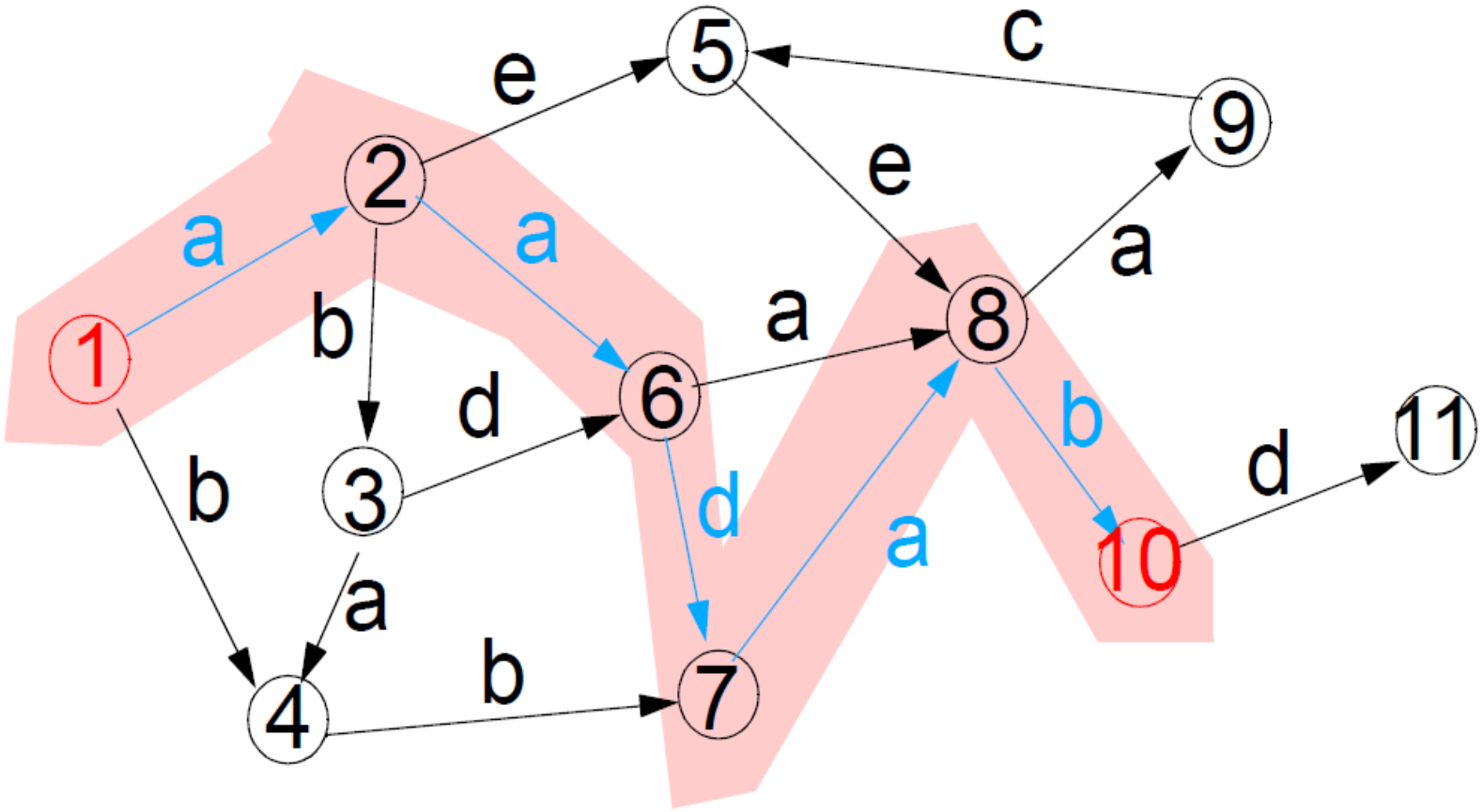
Querying GDBs: RPQ example



Regular expression = a+(d|b)ab

A result:
adab: (2,10)

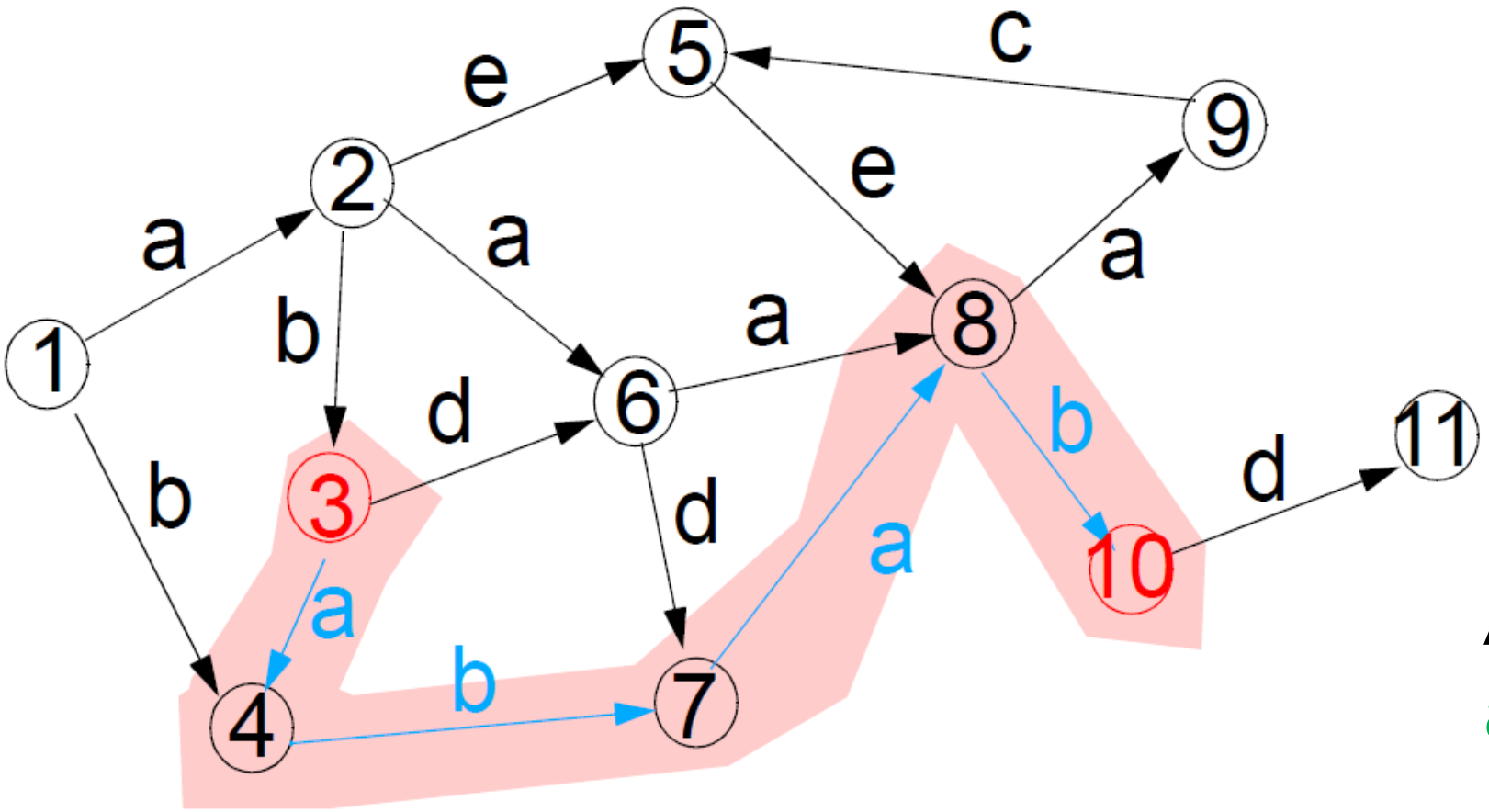
Querying GDBs: RPQ example



Regular expression = a+(d|b)ab

Another result:
aadab: (1, 10)

Querying GDBs: RPQ example



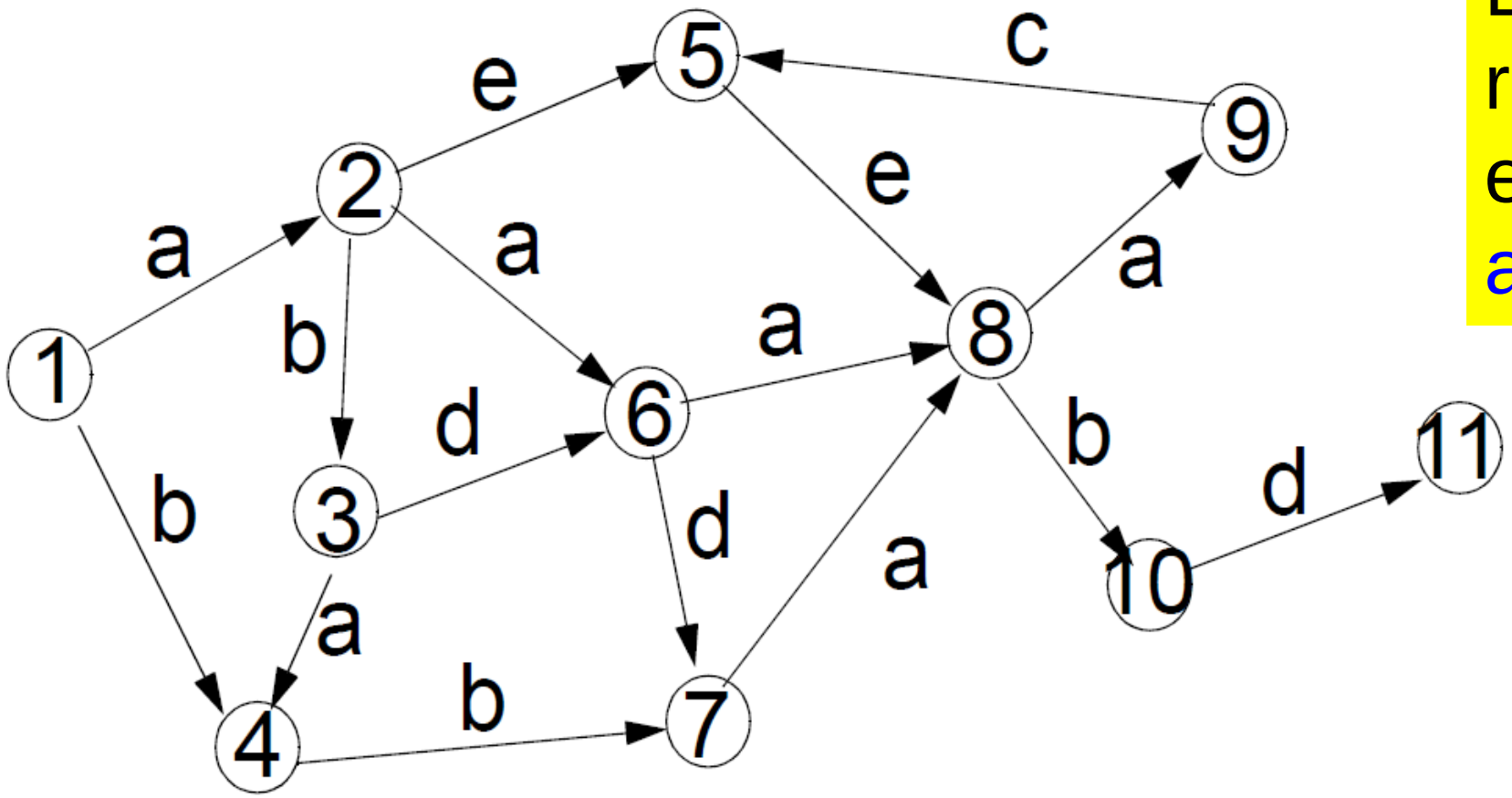
Regular expression = a+(d|b)ab

Another result:
abab: (3, 10)

Querying GDBs:
Two-way Regular Path Queries (2RQP)

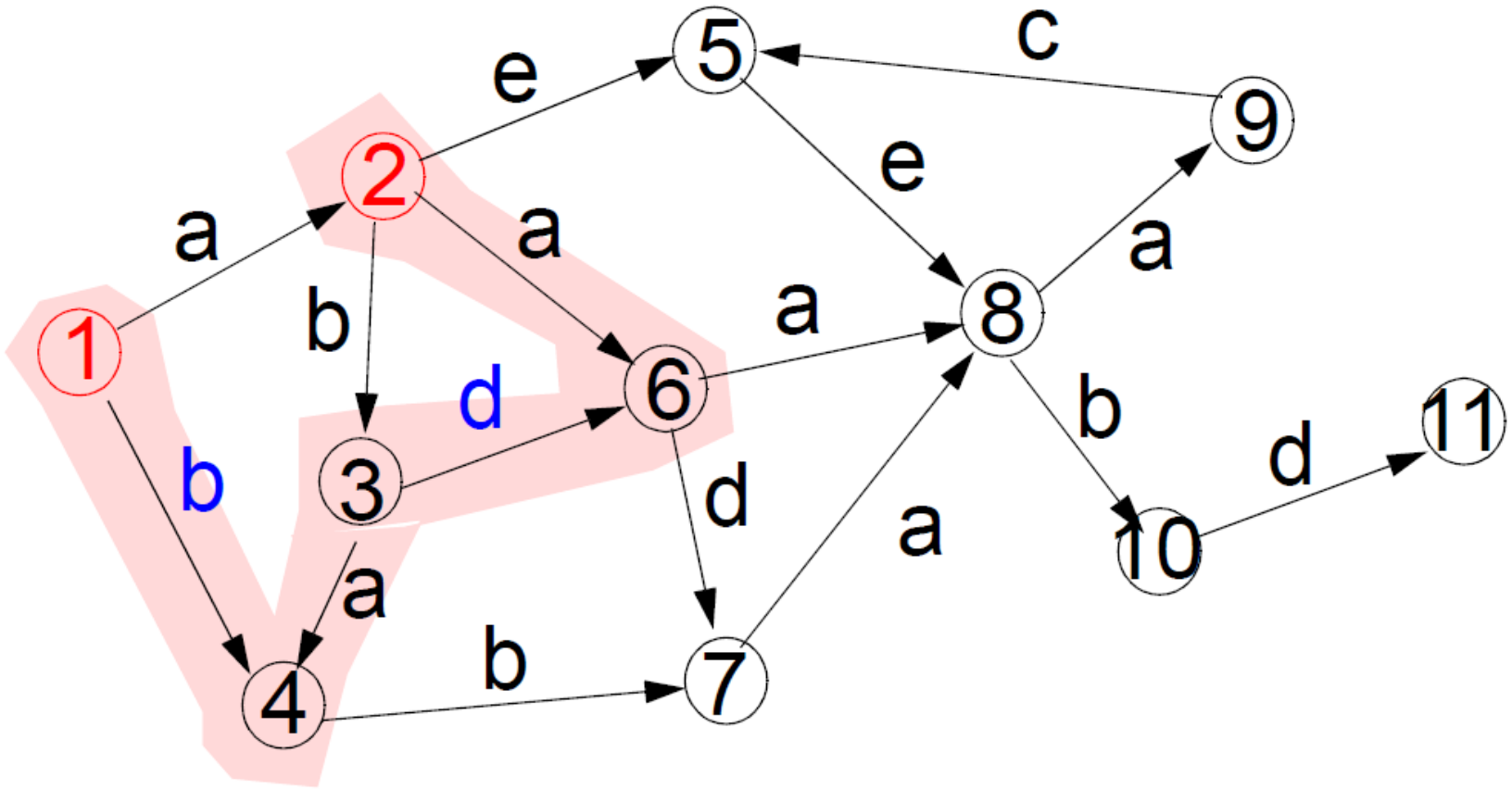
- 2RPQ extend the vocabulary of RPQ by the inverse of each label.
- Foreach label l in Σ there exist a symbol l^-.

Querying GDBs: 2RPQ example



Extended regular expression = $a+d^{-}ab^{-}$

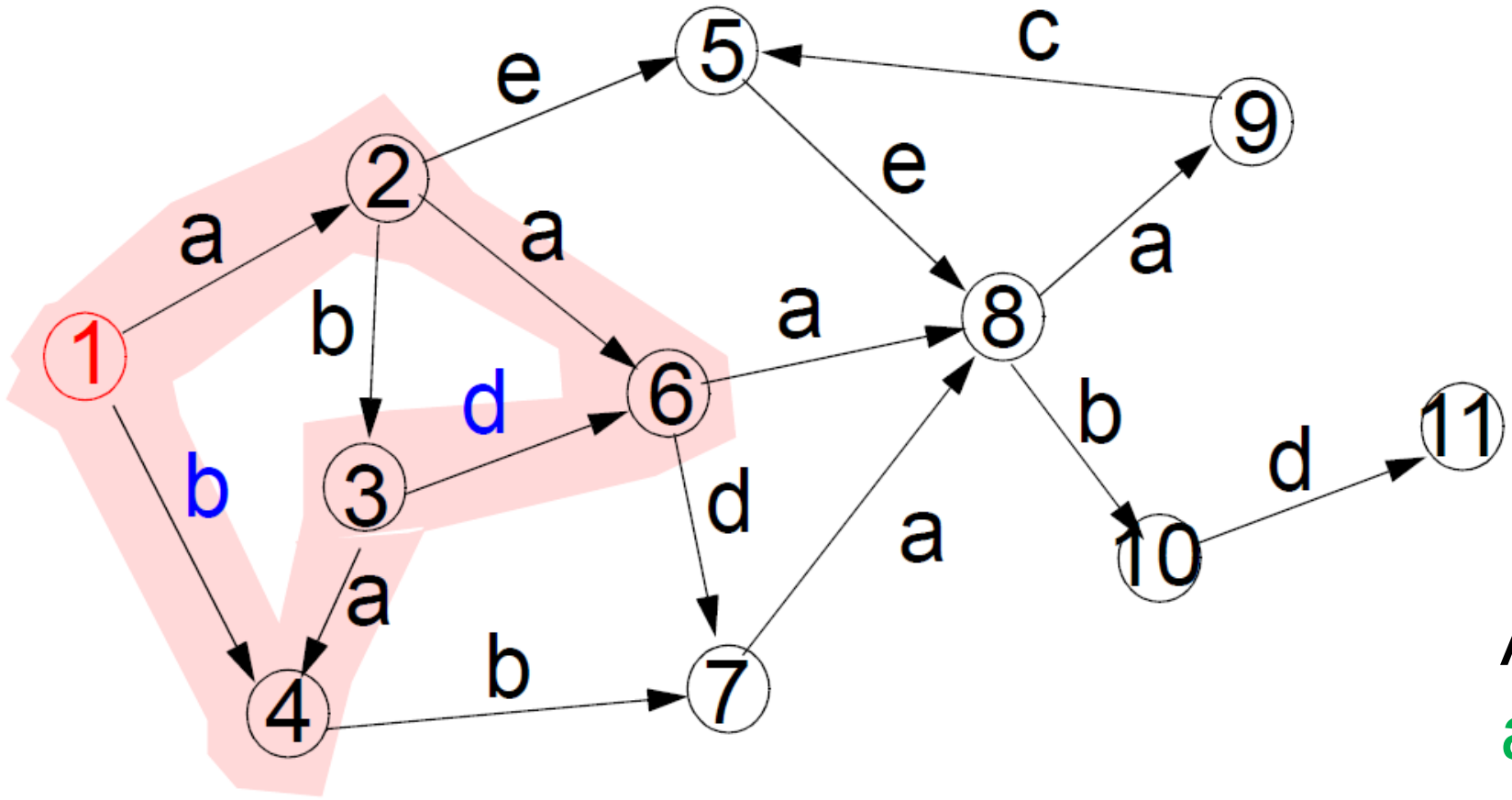
Querying GDBs: 2RPQ example



Extended regular expression = $a+d^{-}ab^{-}$

A result:
adab: (2,1)

Querying GDBs: 2RPQ example



Extended regular expression = $a+d^{-}ab^{-}$

Another result:
aadab: (1, 1)

GDB Management Systems

A GDB management system (GDBMS) is a system that manages GDBs. Some GDBMS are:

When not to use a GDB?

- The whole concept of the category of NRDBs is all about task-orientation:

Use the right tool for the job

- This means that there are certain use cases that GDBs are not as perfectly suited for.
- So, let us briefly touch on a couple of categories of operations that we would probably want to separate from the GDB category.

When not to use a graph database?

- **Large, set-oriented queries**

If we are trying to put together large sets of things (nodes), that do not require a lot of joining or require a lot of aggregation (summing, counting, averaging, and so on) on these sets, then the performance of a GDB compared to other DBMSs will be not as favorable.

The problem in an GDBMS is that the nodes are **scattered**, they are like single-table rows!

When not to use a graph database?

- **Graph global operations**

While GDBs are powerful at answering "graph local" queries, they are not suited for answering graph global operations, i.e., operations that apply over the whole graph (e.g., centrality, in-betweeness measures).

Cypher, the GDB Query Language of Neo4j

- Cypher is designed to be easily read and understood by developers, database professionals and business stakeholders alike.
- It is easy to use because it matches the way we intuitively describe graphs using diagrams.
- The basic notion of Cypher is that it allows us to ask the database to find data that matches a specific pattern.
- Colloquially, we might ask the database to "*find things like this*" and the way we describe what "*things like this*" look like is to draw them using ASCII art.

See next presentation for details

References

- **Domenico Lembo and Riccardo Rosati**, "*Data Management for Data Science*", Corso di laurea magistrale in Data Science, Sapienza Università di Roma, Dipartimento di Ingegneria Informatica Automatica e Gestionale.
- **Andreas Schmidt and Iztok Savnik**, "*Overview of Regular Path Queries in Graphs*", The Seventh International Conference on Advances in Databases, Knowledge, and Data Applications, Rome, Italy.