Evaluating Cypher queries as algebraic expressions within RedisGraph

Roi Lipman
RedisLabs
RedisGraph property graph representation
4 types of matrices

- THE adjacency matrix
- Label matrices
- Relation matrices
- Relationship mapping matrices

All matrices share the same dimensions
#rows = #columns = #nodes
THE Adjacency matrix

```
 1 . . . . 
. . . . . . 
. . . . . . 
1 . . . . . 
. . . . . . 
1 . . . . . 
1 . . . . . 
```

THE Adjacency matrix

• Boolean
THE Adjacency matrix

- Boolean
- Relation type agnostic
THE Adjacency matrix

- Boolean
- Relation type agnostic
- Label agnostic
THE Adjacency matrix

- Boolean
- Relation type agnostic
- Label agnostic
- Directional, $M[i, j] = 1$

Node $i$ connected to Node $j$
*might be multiple times with different relationship types*
Label matrix
Label matrix

- Boolean
Label matrix

- Boolean
- One for each node type
Label matrix

- Boolean
- One for each node type
- Diagonal
  \[ L[i, i] = 1 \]
  Node with ID i is labeled as L
Relation matrix
Relation matrix

- Boolean
- Label agnostic
- Directional, $R[i, j] = 1$

Node $i$ connected to Node $j$
*might be multiple times with relationship type $R$
Relation mapping matrix

\[
\begin{pmatrix}
5 & . & . & . & . \\
. & . & . & . & . \\
. & . & . & . & . \\
. & . & . & . & . \\
0xac804a & . & . & . & . \\
56 & . & 3 & . & . \\
\end{pmatrix}
\]
Relation mapping matrix

- 64bit entries

\[ \text{RM}[i, j] - \text{either edge ID if node } i \text{ is connected to node } j \text{ with a single edge of type } R \]

\[ \text{Pointer to edge IDs array if node } i \text{ is connected to node } j \text{ with multiple edges of type } R \]
Query Execution

Traversals

Chain:
A path where each node on the path appears only once and the sum of its In/Out degrees < 3
MATCH (a:L)-[:R]->(c)<-[[]]-(b:L)
RETURN b
Algebraic expression

\[ \text{THE adj} \]
Matrix multiplication is associative
Motivation keep sparsity
Evaluation

\[ a \begin{bmatrix} LR \\ c \end{bmatrix} \quad c \begin{bmatrix} \text{THE} \\ \text{adj} \end{bmatrix} \quad b \begin{bmatrix} L \\ b \end{bmatrix} \]
Evaluation

\[
\begin{bmatrix}
a & \color{red}{\text{X}} & b \\
\end{bmatrix} * \begin{bmatrix}
\color{red}{\text{b}} \\

\end{bmatrix} \begin{bmatrix}
\color{red}{\text{c}} \\
\end{bmatrix} = \begin{bmatrix}
a & \color{red}{\text{X}} & c \\
\end{bmatrix}
\]

Row domain is “Sticky”
Evaluation

\[
\begin{bmatrix}
a & b \\
X & Y \\
c & c
\end{bmatrix}
= 
\begin{bmatrix}
a & z
\end{bmatrix}
\]

Column domain changes with every “hop”
“2D”
$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ c \end{bmatrix} = \begin{bmatrix} Z \end{bmatrix}$
a is connected to b
But how?
MATCH (a:L)-[:R]->(c)<--[]-(b:L)
RETURN c
Intermediate entities

- Referred node/edge

  RETURN n,e
Intermediate entities

- Variable length edges

MATCH (a)-[e:*2..4]->(b)
Intermediate entities

- Filtered entities

WHERE n.v = 34
Break on intermediate
Break on intermediate
Traverse L*R (a, c)

Traverse c*Bt*L (a, c, b)
High degree nodes & Cycles

MATCH (a)-[:]->(b)-[:]->(a)
RETURN a
Cycle
Cycle
Improved Algebraic expressions construction
Exps = []
QG = QueryGraph(AST)
CCS = ConnectedComponents(QG)

For CC in CCS
    while(CC not empty)
        P = LongestPath(CC)
        E = AlgebricExpressionFromPath(P)
        Es = AlgebricExpressionBreakInter(E)
        Exps += Es
        CC -= P

Pseudo code
MATCH (a)-[]->(b)-[]->(a), (a)-[]->(c), (d)-[]->(e)
Connected Components

- C
- A
- B
- D
- E
Longest path

$P_0 = \{C, A, B\}$

$P_1 = \{B, A, C\}$
Algebraic expression from path (chain)
Break on intermediates
Remove path from Query Graph

\[
\begin{align*}
\text{C} & \rightarrow \text{A} \rightarrow \text{B} \\
\text{C} & \rightarrow \text{A} \rightarrow \text{B} \\
\text{C} & \rightarrow \text{A} \rightarrow \text{B}
\end{align*}
\]
Algebraic expression from path (chain)
Expressions

Exp 0

Exp 1

Exp 2
Arrangements

Permuted \( ([\text{Exp0}, \text{Exp1}, \text{Exp2}]), \ N! \)

Pick best arrangement to be executed in order

Valid arrangement:
1. Expi either src or dest node is already resolved by previous expression.
2. Early filters
Expand into (a, b)

Traverse (b, a)

Expand into (a, b)

Traverse (a, c)
Thank you

Roi Lipman
roi@redislabs.com