Querying graph databases

Neo4j and Cypher

NoSQL: Lecture 5

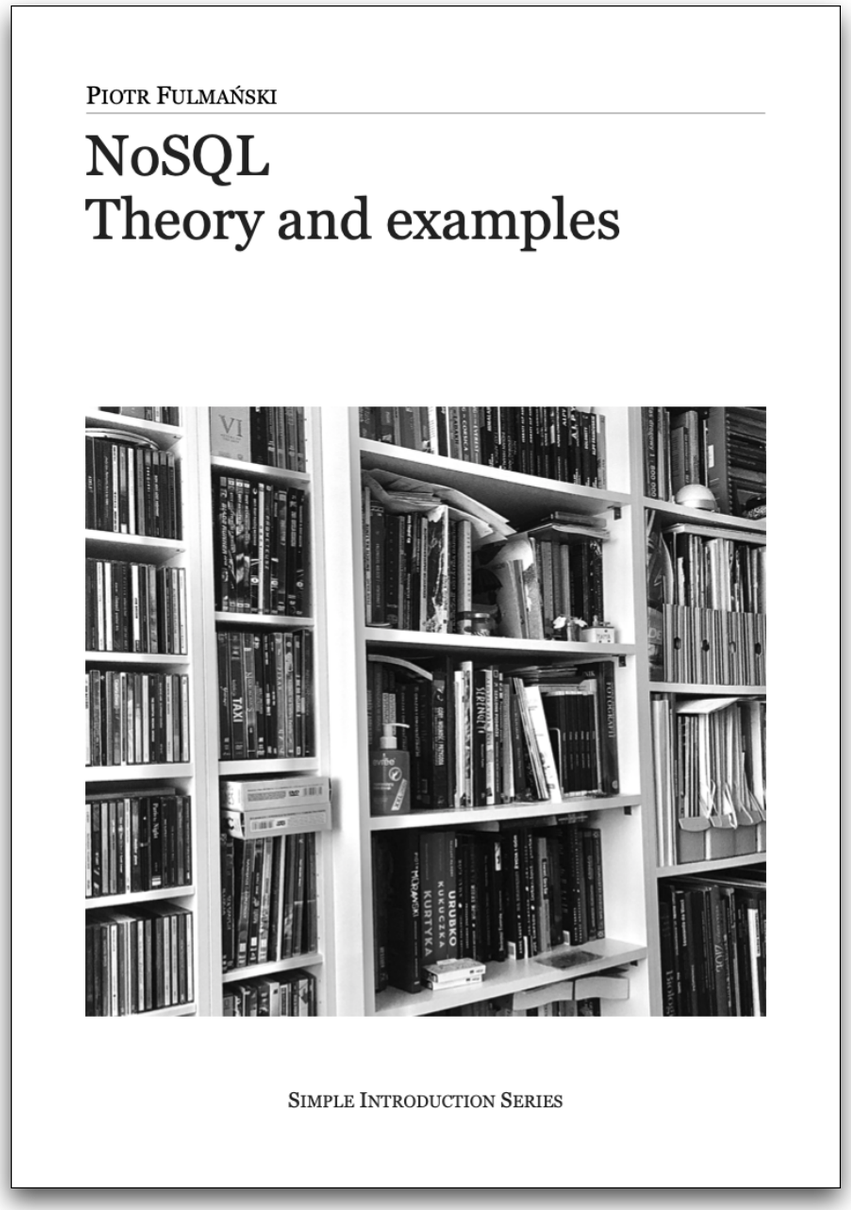
Piotr Fulmański

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
University of Lodz

NoSQL Theory and examples

by Piotr Fulmański

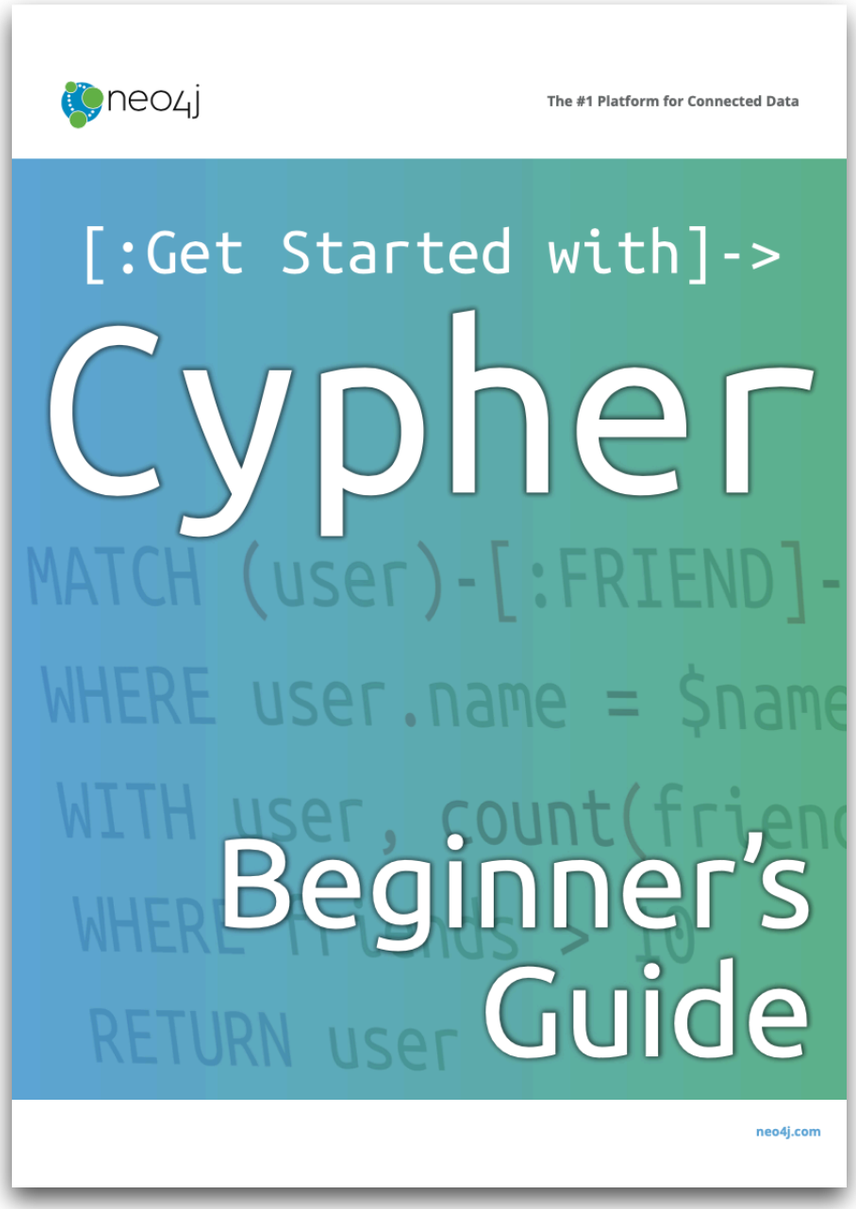
Piotr Fulmański, 2021



Get Started with Cypher: A Beginner's Guide

by Michael Hunger

https://neo4j.com/whitepapers/getting-started-with-cypher/



Cypher – Neo4j's query language

The art of making difficult things easy.

Cypher is a ***declarative***, ***pattern-matching*** query language that makes graph database management systems understandable and workable for any database user, even the less technical ones.

The true power of Cyper is its incredible ability to expressing ***graph patterns***. These patterns are what you would usually draw on a whiteboard, just converted into text using ASCII-art symbols.

Cypher – Neo4j's query language

The art of making difficult things easy.

The main components of the graph model are

- *nodes*,
- *relationships*
- and their *properties*.

In everyday life we express them with paper and pencil drawing

- *circles* for nodes,
- *arrows* or lines for relationships
- and *placing some text* close them for properties.

And we do exactly the same in Cypher, of course within the natural limits imposed by the text interface.

Cypher – Neo4j's query language

The art of making difficult things easy.

To depict nodes in Cypher, we use parentheses which, when empty, look similar to the circles: `()`. There are few possible options to do this:

- `()` – anonymous node (no label or variable); can refer to any node in the database;
- `(n)` – node identified by variable `n`;
- `(:Node)` – node labeled `Node`;
- `(n:Node)` – node labeled `Node` and identified by variable `n`.

Cypher – Neo4j's query language

The art of making difficult things easy.

To express the relationships between nodes we use an arrow `-->` or `<--`, and in case of undirected relationships, just two dashes `--`. Additional information, can be placed in square brackets inside of the arrow.

- `-->` – typical from-to relation;
- `<--` – backward relation;
- `--` – undirected relation;
- `-[r]->` – from-to relation identified by variable `r`;
- `-[:KNOWS]->` – from-to relation labeled `KNOWS`;
- `-[r:KNOWS]->` – from-to relation labeled `KNOWS` and identified by variable `r`.

Cypher – Neo4j's query language

The art of making difficult things easy.

The last part are properties. Properties are *name-value* pairs that provide additional details to nodes and relationships.

To represent these in Cypher, we can use curly braces within the parentheses of a node or the brackets of a relationship. The name and value of the property then go inside the curly braces. See the examples below:

- `(v:Node {propertyName: 'Property value'})` node property;

- `-[r:KNOWS {propertyName: 'Property value'}]->` relationship property.

Cypher – Neo4j's query language

The art of making difficult things easy.

To create a pattern we combine the node, relationship and properties syntaxes we have shown so far. Example of simple pattern would look like the code below.

```
(c:Customer {name:'C3PO'})-[rel:ORDERED]->(i:Item)
```

This snippet defines the pattern we want, but it does not tell what we want to do with it: whether we want to find that existing pattern or insert it as a new pattern. That's why need to use some *keywords* and both together will make a correct Cypher statement.

What we need

- `http://console.neo4j.org`
- `https://neo4j.com/sandbox`
- `https://neo4j.com/download`

Practical part 1

What we need

Sample data (sandbox)

We will use sample data of secret agents network. As they are a secret agents, we don't know their names. Instead we will use a short pseudonyms.

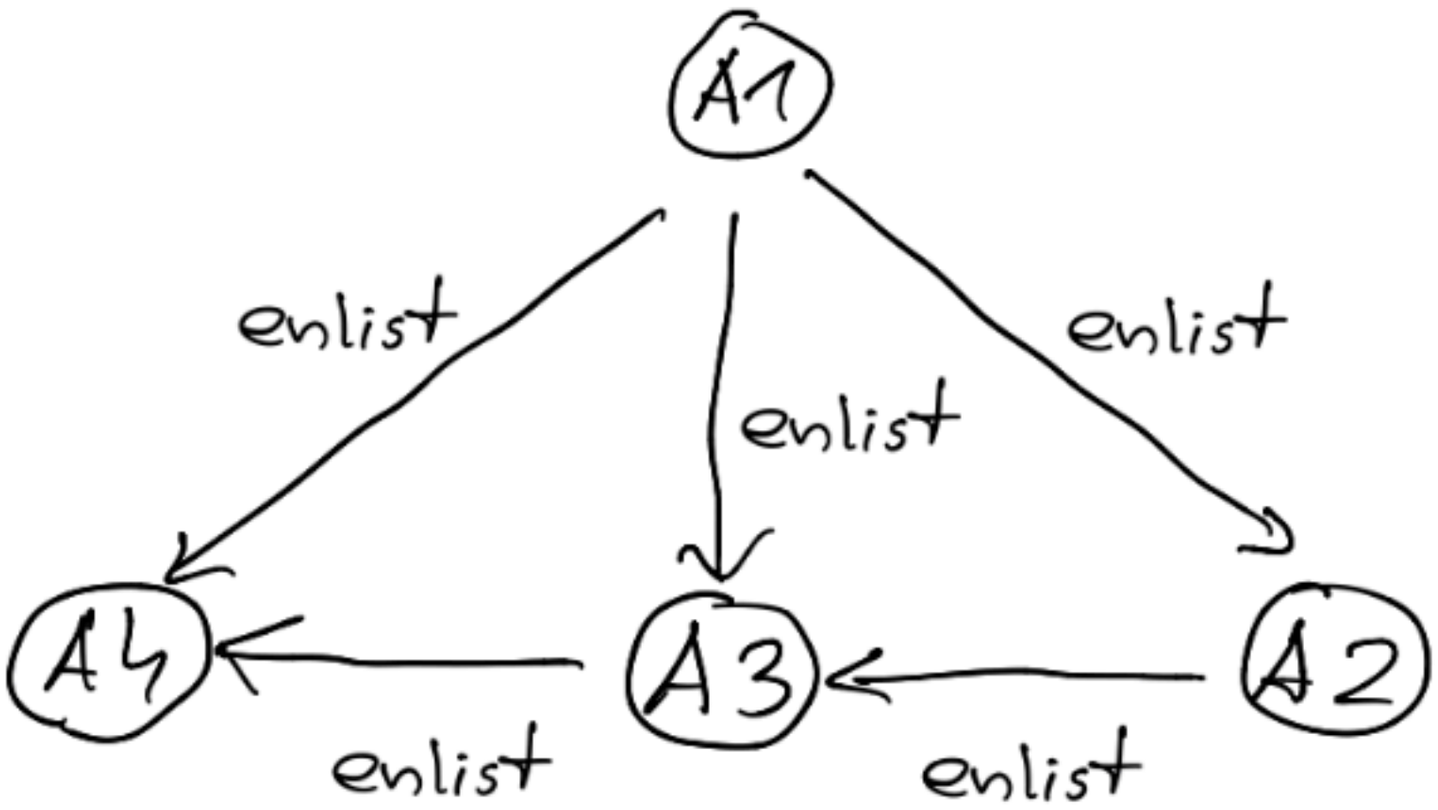
You can use a set of queries to create a sample data

`agents_cypher.txt`

In all presented examples we will use Gremlin Sandbox. In case of different tool, results may be presented in a different way.

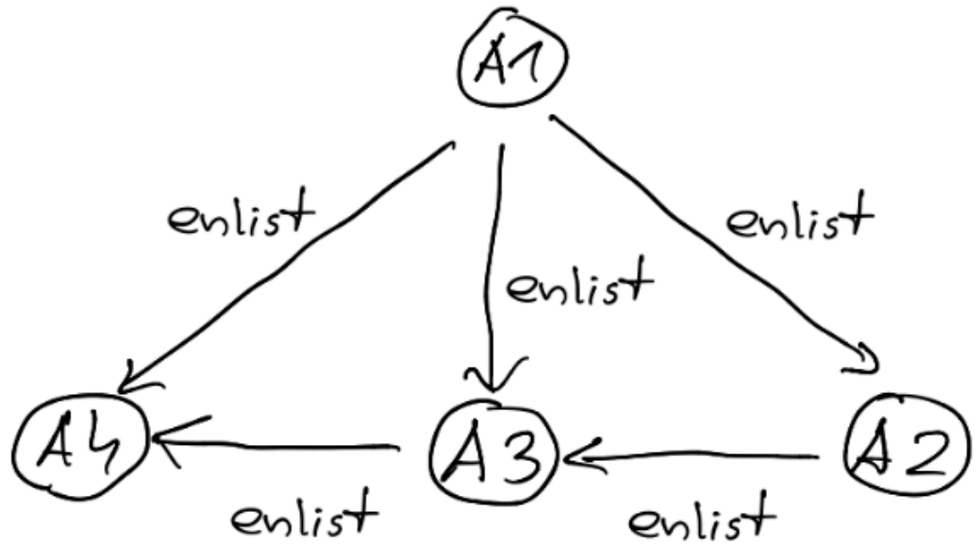
Sample data

Secret agents net



Graph traverse

Very basic "traversal"




```
MATCH (v:agent)
RETURN ID(v), v
```

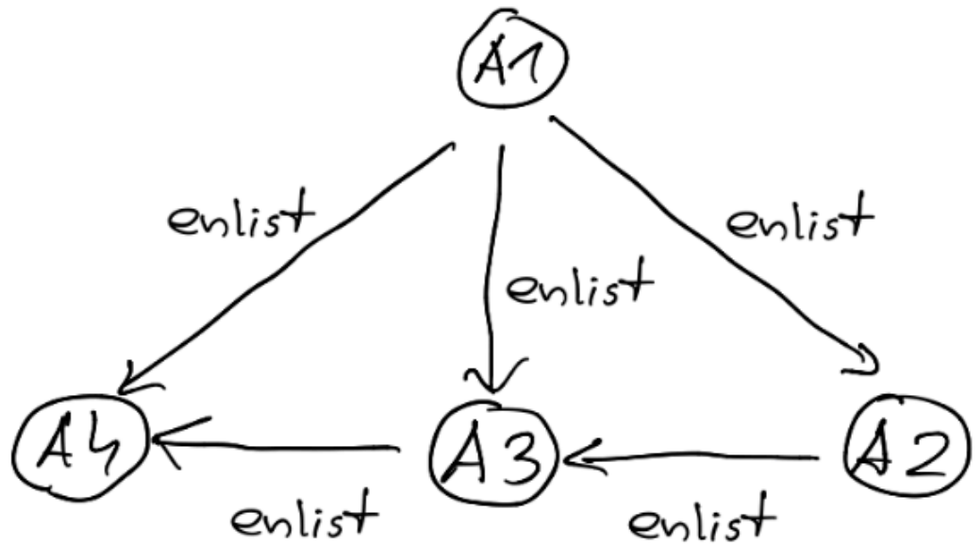
"ID(v)"	"v"
0	{"pseudonim":"A1"}
2	{"pseudonim":"A2"}
38	{"pseudonim":"A3"}
39	{"pseudonim":"A4"}

First row as JSON


```
{
  "identity": 0,
  "labels": [
    "agent"
  ],
  "properties": {
    "pseudonim": "A1"
  }
}
```

Graph traverse

Very basic "traversal"

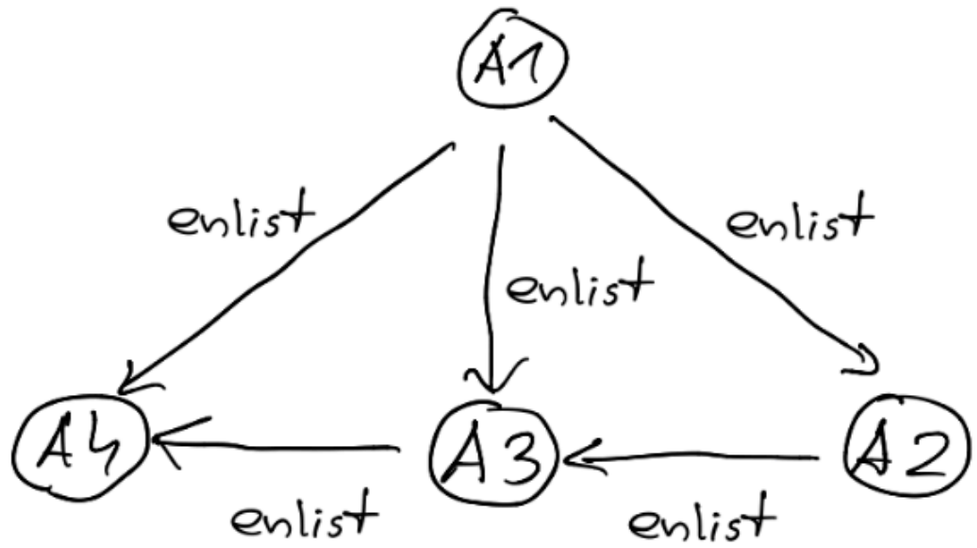



```
MATCH (v)
RETURN ID(v), labels(v), keys(v), v
```

"ID(v)"	"labels(v)"	"keys(v)"	"v"
0	["agent"]	["pseudonim"]	{"pseudonim":"A1"}
2	["agent"]	["pseudonim"]	{"pseudonim":"A2"}
38	["agent"]	["pseudonim"]	{"pseudonim":"A3"}
39	["agent"]	["pseudonim"]	{"pseudonim":"A4"}

Graph traverse

Very basic "traversal"




```
MATCH (from)-[e]->(to) RETURN ID(from), ID(to), ID(e), KEYS(e), e
```

"ID(from)"	"ID(to)"	"ID(e)"	"keys(e)"	"e"
0	39	51	["id"]	{"id":"A1_A4"}
0	38	72	["id"]	{"id":"A1_A3"}
0	2	74	["id"]	{"id":"A1_A2"}
2	38	89	["id"]	{"id":"A2_A3"}
38	39	92	["id"]	{"id":"A3_A4"}

First row as JSON


```
{
  "identity": 51,
  "start": 0,
  "end": 39,
  "type": "enlist",
  "properties": {
    "id": "A1_A4"
  }
}
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

We start by outlining the steps we will have to consider:

1. Use the following general pattern

   ```
   (agent)-[enlist]->(who)
   ```

2. Find all elements matched this pattern.
3. Vertex `agent` should have a *pseudonim* of *A2*.
4. Return the `who` vertex.

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

   ```
   (agent)-[enlist]->(who)
   ```

2.
3.
4.

```
(v:agent)-[:enlist]->(who)
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

   ```
   (agent)-[enlist]->(who)
   ```

2. Find all elements matched this pattern.
3.
4.

```
MATCH (v:agent)-[:enlist]->(who)
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

   `(agent)-[enlist]->(who)`

2. Find all elements matched this pattern.
3. Vertex `agent`
4.

```
MATCH (v:agent)-[:enlist]->(who)
      v
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

   `(agent)-[enlist]->(who)`

2. Find all elements matched this pattern.
3. Vertex `agent` should have
4.

```
MATCH (v:agent)-[:enlist]->(who)
WHERE v
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

   `(agent)-[enlist]->(who)`

2. Find all elements matched this pattern.
3. Vertex `agent` should have a *pseudonim* of *A2*.
4.

```
MATCH (v:agent)-[:enlist]->(who)
WHERE v.pseudonim = "A2"
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

   `(agent)-[enlist]->(who)`

2. Find all elements matched this pattern.
3. Vertex `agent` should have a *pseudonim* of *A2*.
4. Return

```
MATCH (v:agent)-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN
```

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]->??).

1. Use the following general pattern

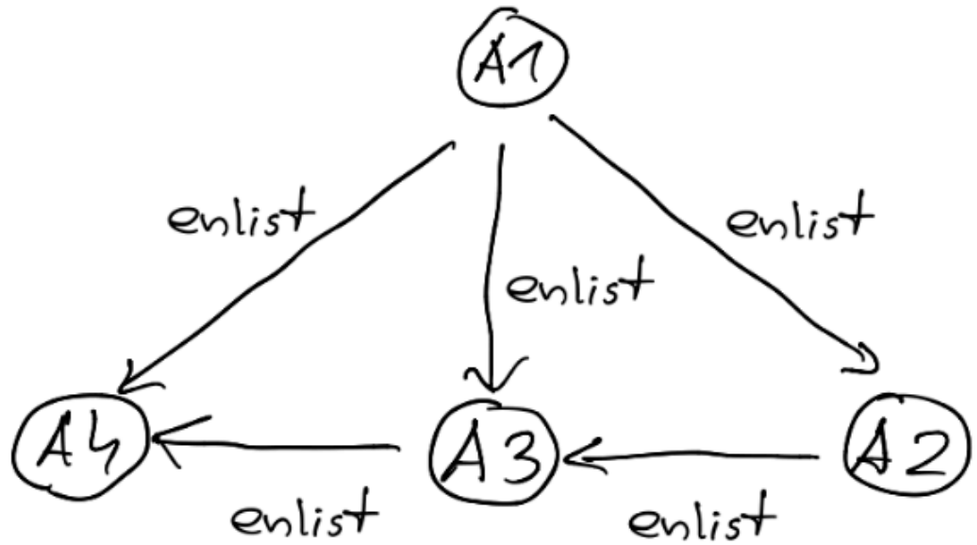
   `(agent)-[enlist]->(who)`

2. Find all elements matched this pattern.
3. Vertex `agent` should have a *pseudonim* of *A2*.
4. Return the `who` vertex.

```
MATCH (v:agent)-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

Graph traverse

Very basic "traversal"

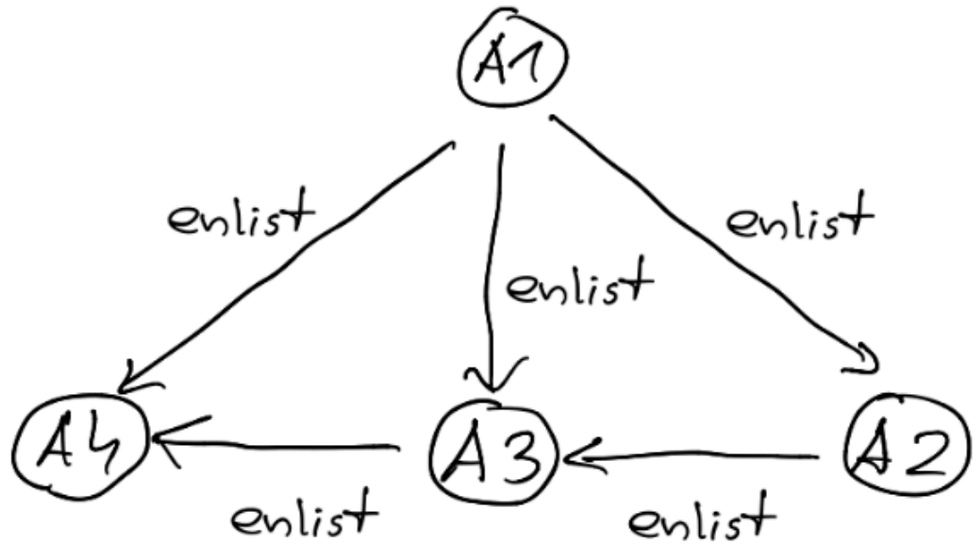



```
MATCH (v:agent)-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

"who"
{"pseudonim":"A3"}

Graph traverse

Very basic "traversal"



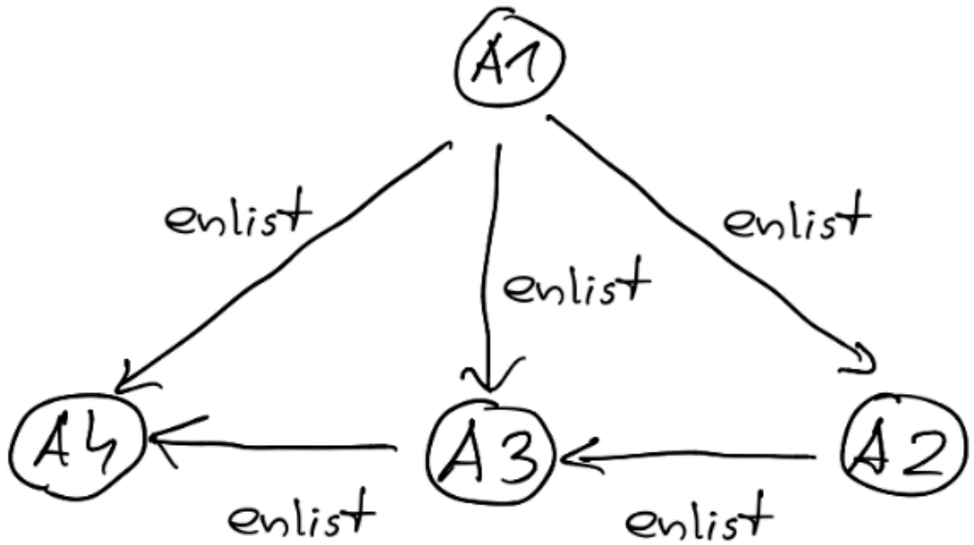

```
MATCH (v:agent)-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

"who"
{"pseudonim":"A3"}

Other variants:

Graph traverse

Very basic "traversal"




```
MATCH (v:agent)-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

"who"
{"pseudonim":"A3"}

Other variants:

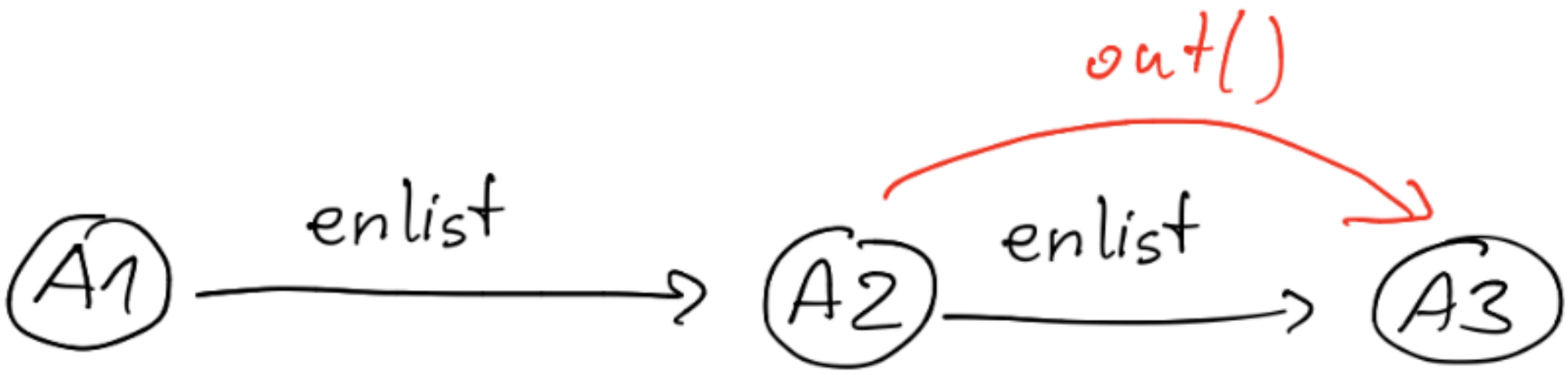
```
MATCH (v:agent {pseudonim: "A2"})-[:enlist]->(who)
RETURN who

MATCH (v:agent)-->(who)
WHERE v.pseudonim = "A2"
RETURN who

MATCH (v:agent {pseudonim: "A2"})-->(who)
RETURN who
```

Graph traverse

Traversal steps: *in*, *out*

```
MATCH (v:agent {pseudonim: "A2"})-->(who_out)
RETURN who_out
```

"who_out"
{"pseudonim":"A3"}

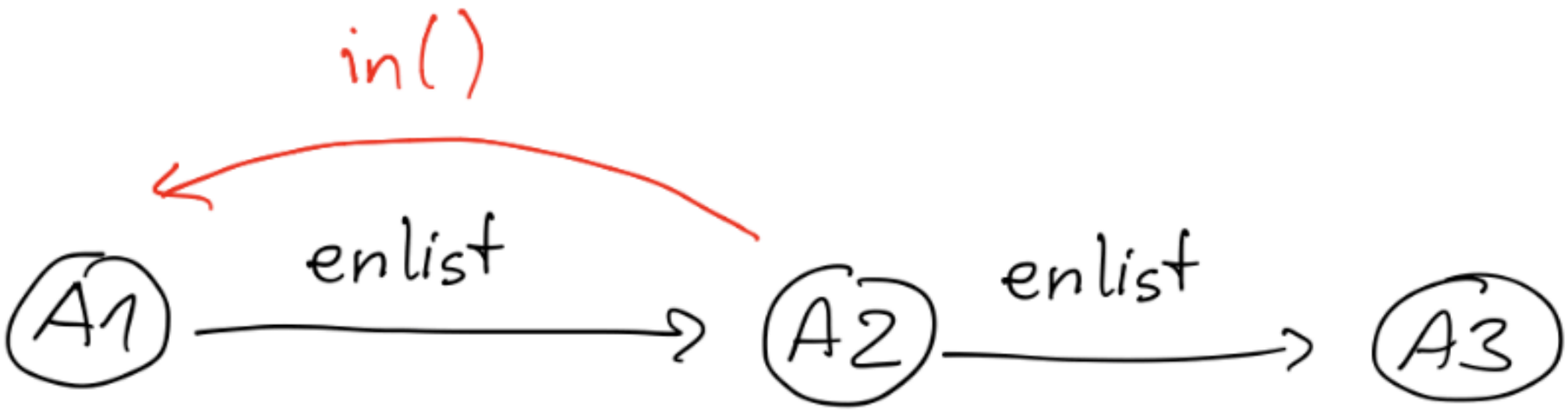


Graph traverse

Traversal steps: *in*, *out*

```
MATCH (who_in)-->(v:agent {pseudonim: "A2"})
RETURN who_in
```

"who_in"
{"pseudonim":"A1"}

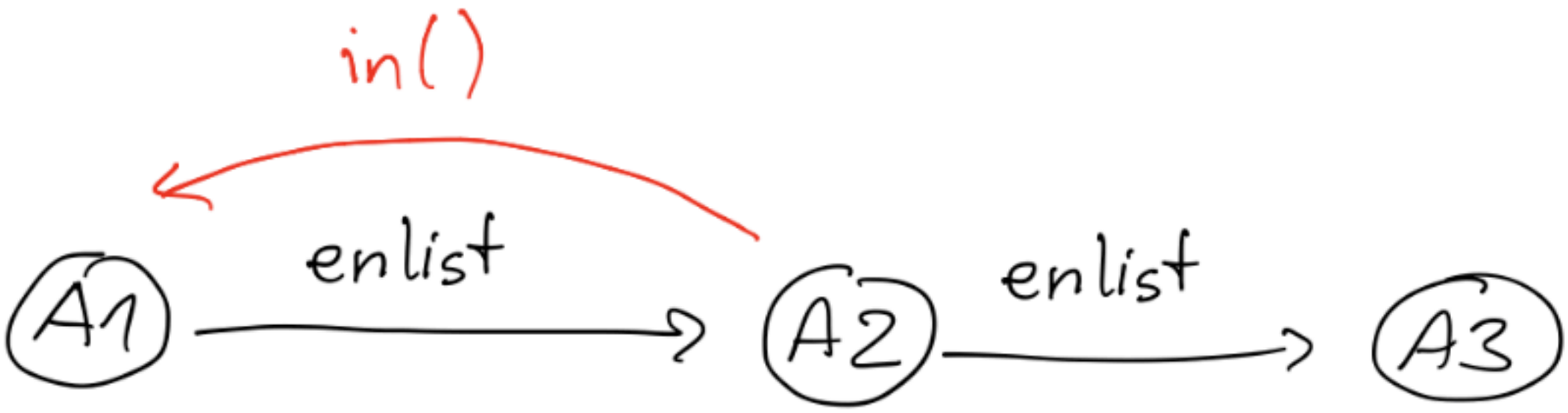


Graph traverse

Traversal steps: *in*, *out*

```
MATCH (v:agent {pseudonim: "A2"})<--(who_in)
RETURN who_in
```

"who_in"
{"pseudonim":"A1"}

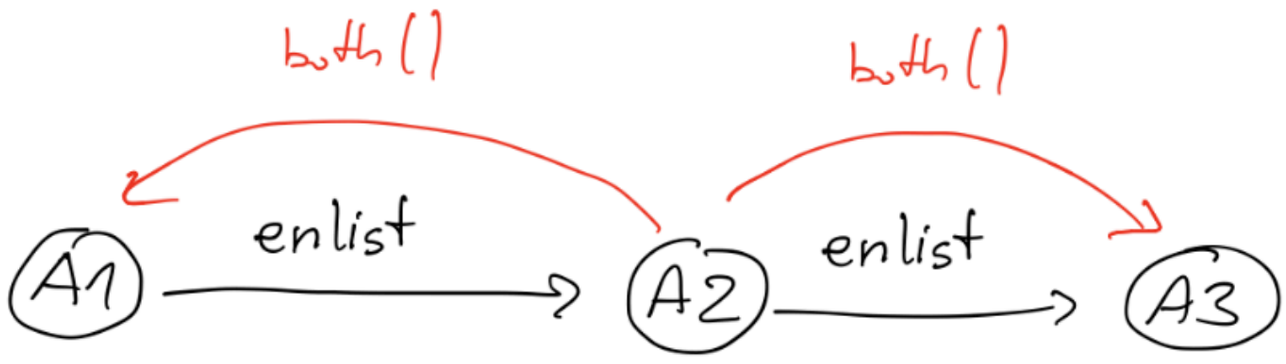


Graph traverse

Traversal steps: *in*, *out*

```
MATCH (who_in)-->(v:agent {pseudonim: "A2"})-->(who_out)
RETURN who_in, who_out
```

"who_in"	"who_out"
{"pseudonim":"A1"}	{"pseudonim":"A3"}



Graph traverse

Traversal steps: *in*, *out*

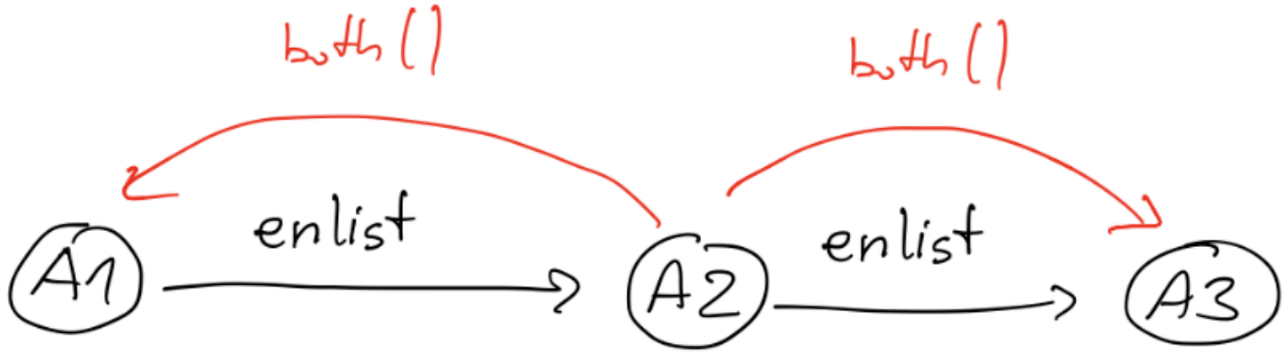
```
MATCH (who_in)-->(v:agent {pseudonim: "A2"})-->(who_out)
RETURN who_in, who_out
```

"who_in"	"who_out"
{"pseudonim":"A1"}	{"pseudonim":"A3"}

But this is not the same as

```
g.V().
has('agent', 'pseudonim', 'A2').
both('enlist').
values('pseudonim')
==>A3
==>A1
```

because we don't get a one consistent set that can be operated further.
Instead get `in` and `out` separately.

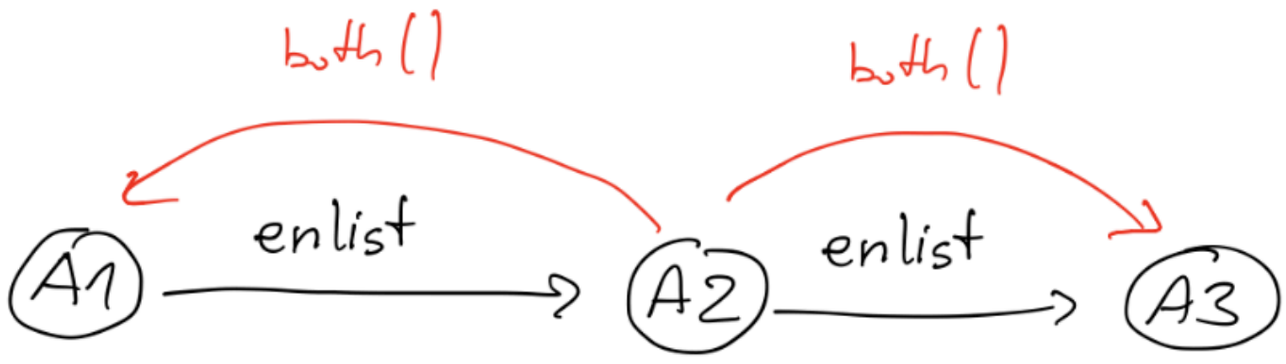


Graph traverse

Traversal steps: *in*, *out*

```
MATCH (v:agent {pseudonim: "A2"})--(who)
RETURN who
```

"who"
{"pseudonim":"A3"}
{"pseudonim":"A1"}

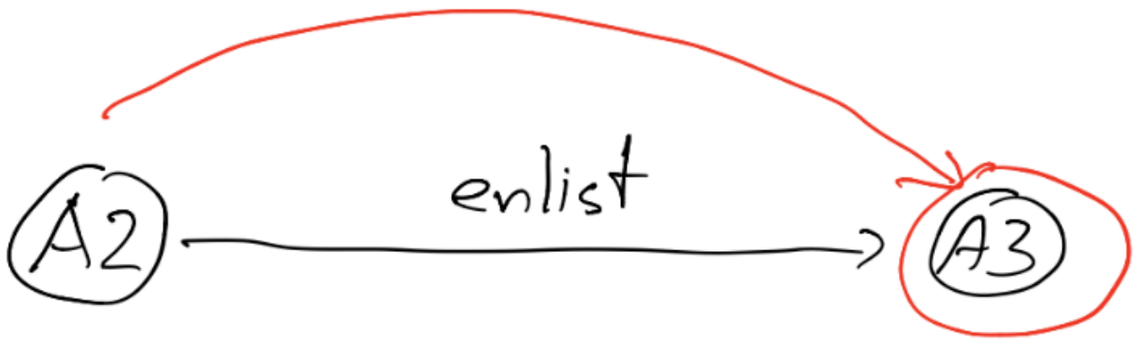


Graph traverse

Traversal steps: `out()` vs `outE()`

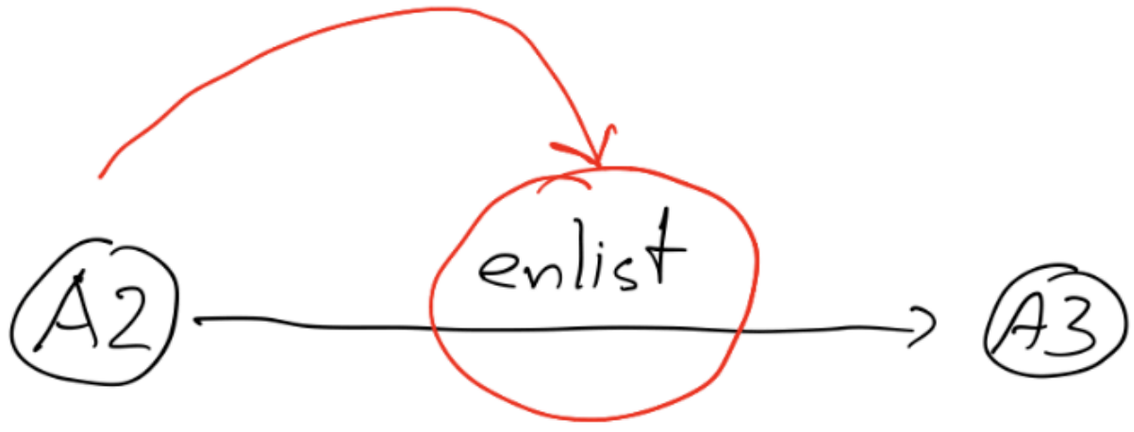
- `MATCH (v:agent {pseudonim: "A2"})-[]->(want_this)`
  `RETURN want_this`

"want_this"
{"pseudonim":"A3"}



- `MATCH (v:agent {pseudonim: "A2"})-[want_this]->()`
  `RETURN want_this`

"want_this"
{"id":"A2_A3"}



Graph traverse

First real traversal

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

We need to consider the following steps:

1. Use the following general pattern

   ```
   (agent)-[enlist]->()-[enlist]->(who)
   ```

2. Find all element matched this pattern.
3. Vertex `agent` should have a *pseudonim* of *A2*.
4. Return the `who` vertex.

As we can see, it's much simpler than in Gremlin case - the only difference is a different pattern.

Graph traverse

First real traversal

Task: Find all agents known by agent A2 (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   ```
   (agent)-[enlist]->()-[enlist]->(who)
   ```

2. Find all element matched this pattern.
3. Vertex `agent` should have a *pseudonim* of *A2*.
4. Return the `who` vertex.

```
MATCH (v:agent)-[:enlist]->()-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

Graph traverse

First real traversal

```
MATCH (v:agent)-[:enlist]->()-[:enlist]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

"who"
{"pseudonim":"A4"}

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

We need to consider the following steps:

1. Use the following general pattern

   ```
   (agent)-{2_steps_length_path}->(who)
   ```

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern
4. Vertex `agent` should have a *pseudonim* of *A2*.
5. Return the `who` vertex.

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   ```
   (agent)-{2_steps_length_path}->(who)
   ```

2.
3.
4.
5.

```
(v:agent)-{2_steps_length_path}->(who)
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   ```
   (agent)-{2_steps_length_path}->(who)
   ```

2. Define variable-length part of the pattern.
3.
4.
5.

```
(v:agent)-[:enlist*2]->(who)
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   ```
   (agent)-{2_steps_length_path}->(who)
   ```

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern.
4.
5.

```
MATCH (v:agent)-[:enlist*2]->(who)
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   `(agent)-{2_steps_length_path}->(who)`

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern.
4. Vertex `agent`
5.

```
MATCH (v:agent)-[:enlist*2]->(who)
      v
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   `(agent)-{2_steps_length_path}->(who)`

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern.
4. Vertex `agent` should have
5.

```
MATCH (v:agent)-[:enlist*2]->(who)
WHERE v
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   `(agent)-{2_steps_length_path}->(who)`

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern.
4. Vertex `agent` should have a *pseudonim* of *A2*.
5.

```
MATCH (v:agent)-[:enlist*2]->(who)
WHERE v.pseudonim = "A2"
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   `(agent)-{2_steps_length_path}->(who)`

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern.
4. Vertex `agent` should have a *pseudonim* of *A2*.
5. Return

```
MATCH (v:agent)-[:enlist*2]->(who)
WHERE v.pseudonim = "A2"
RETURN
```

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

1. Use the following general pattern

   `(agent)-{2_steps_length_path}->(who)`

2. Define variable-length part of the pattern.
3. Find all elements matched this pattern.
4. Vertex `agent` should have a *pseudonim* of *A2*.
5. Return the `who` vertex.

```
MATCH (v:agent)-[:enlist*2]->(who)
WHERE v.pseudonim = "A2"
RETURN who
```

Graph traverse

First real traversal - variable-length pattern matching

```
MATCH (v:agent {pseudonim: "A2"})-[:enlist*2]->(who)
RETURN who
```

"who"
{"pseudonim":"A4"}

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents known by agents who A2 knows (A2 -[enlist]-> A? -[enlist]-> ??).

This works and provides the correct answer, but it only works because we knew that we needed to make 2 steps. In many cases, we don’t know how many repetitions we’ll need.

```
MATCH (v:agent {pseudonim: "A2"})-[:enlist*1..2]->(who)
RETURN who
```

"who"
{"pseudonim":"A3"}
{"pseudonim":"A4"}

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents (known by agents) who A2 knows (A2 (-[enlist]-> A?)-[enlist]-> ??).

This works and provides the correct answer, but it only works because we knew that we needed to make 2 steps. In many cases, we don't know how many repetitions we'll need.

```
MATCH (v:agent {pseudonim: "A2"})-[:enlist*]->(who)
RETURN who
```

"who"
{"pseudonim":"A3"}
{"pseudonim":"A4"}

Graph traverse

First real traversal - variable-length pattern matching

Task: Find all agents (known by agents) who A2 knows (A2 (-[enlist]-> A?)-[enlist]-> ??).

This works and provides the correct answer, but it only works because we knew that we needed to make 2 steps. In many cases, we don't know how many repetitions we'll need.

```
MATCH (v:agent {pseudonim: "A2"})-[:enlist*]->(who)
RETURN who
```

"who"
{"pseudonim":"A3"}
{"pseudonim":"A4"}

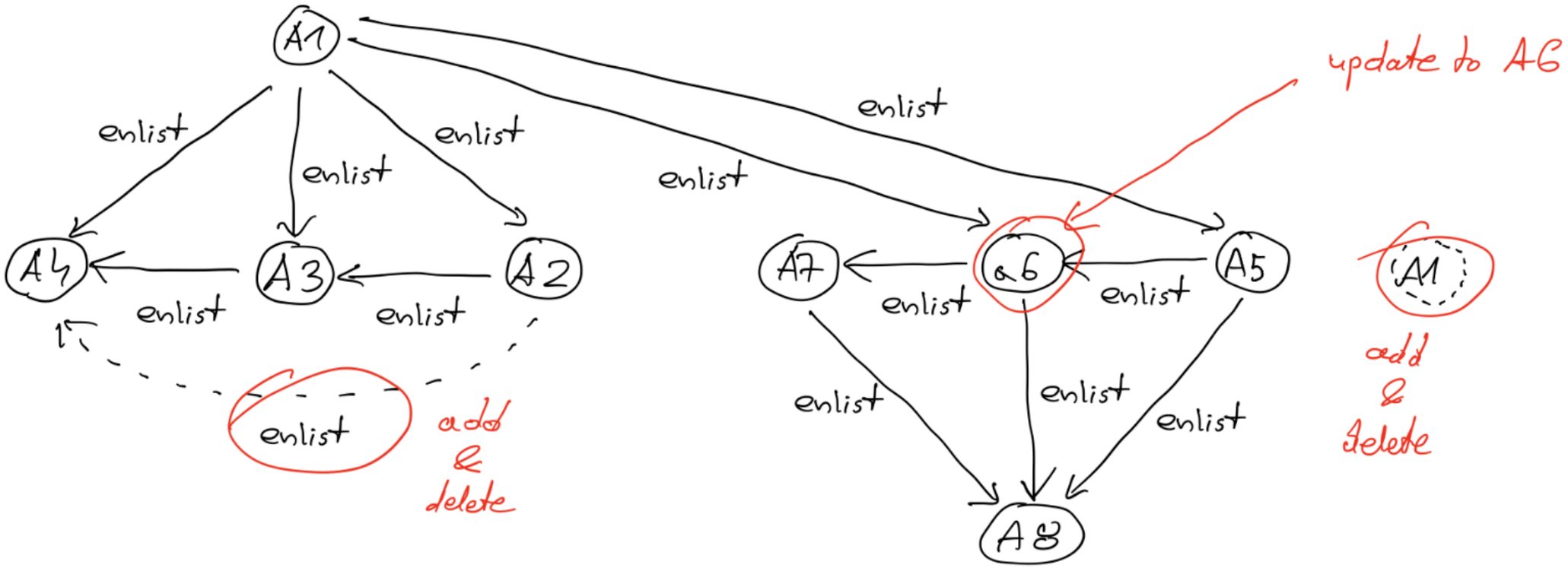
```
MATCH p=(v:agent {pseudonim: "A2"})-[:enlist*]->(who)
RETURN who, p
```

"who"	"p"
{"pseudonim":"A3"}	[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"}]
{"pseudonim":"A4"}	[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"}]

Practical part 2

Graph mutating

Let's expand the network of secret agents to the following form (use `agents2_cypher.txt` file to (re)create this data)



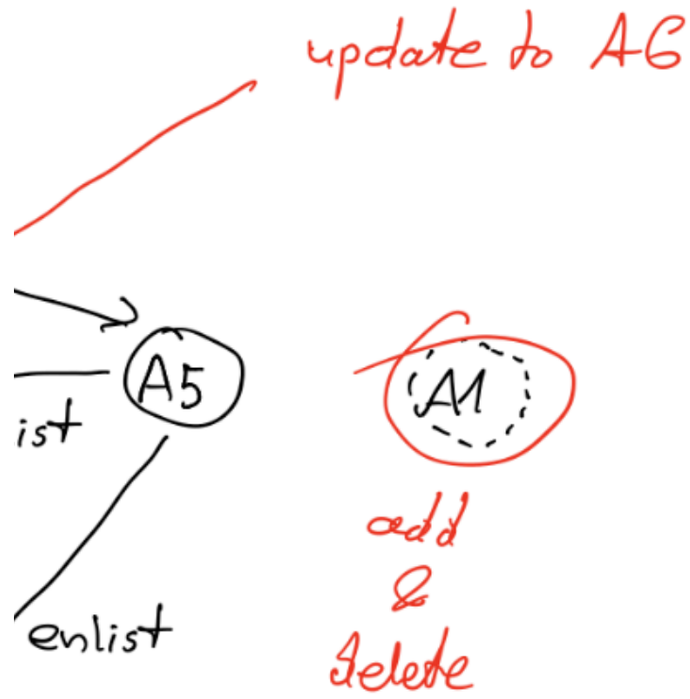
In all presented examples we will use Gremlin Sandbox. In case of different tool, results may be presented in a different way.

Graph mutating

Adding vertices

```
MATCH (v:agent)
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
1	{"pseudonim":"A7"}
2	{"pseudonim":"A2"}
20	{"pseudonim":"A8"}
38	{"pseudonim":"A3"}
74	{"pseudonim":"A4"}
94	{"pseudonim":"A5"}
95	{"pseudonim":"a6"}



Graph mutating

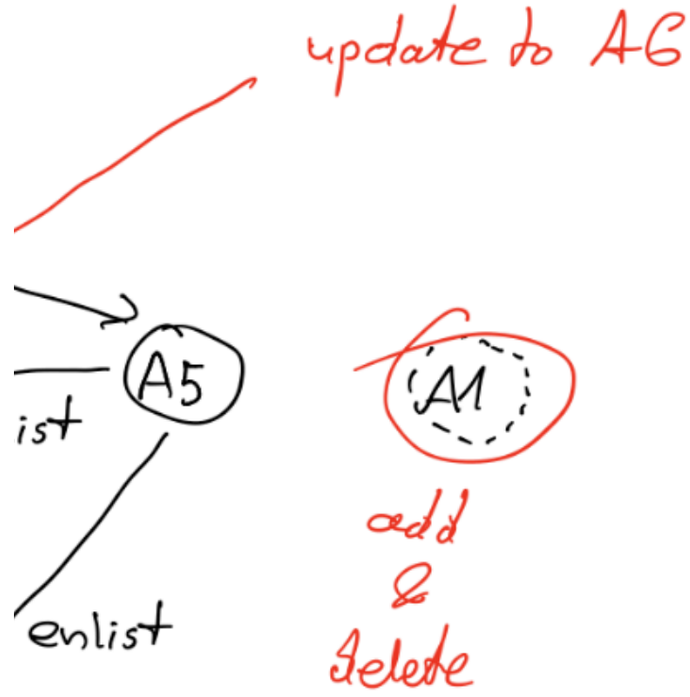
Adding vertices

```
MATCH (v:agent)
WHERE v.pseudonim = 'A1'
RETURN ID(v), v
```

or

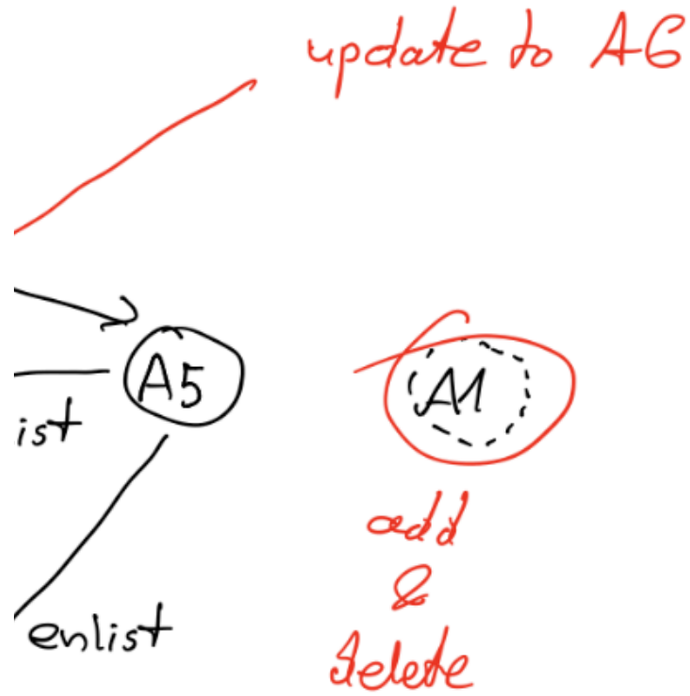
```
MATCH (v:agent {pseudonim: 'A1'})
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}



Graph mutating

Adding vertices




```
MATCH (v:agent {pseudonim: 'A1'})
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}

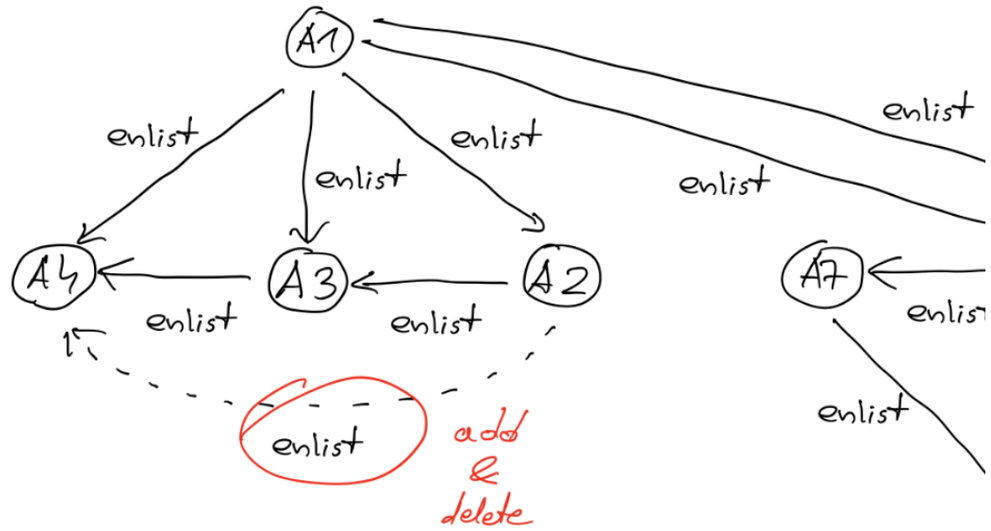
```
CREATE (:agent {pseudonim: 'A1'})
```

```
MATCH (v:agent {pseudonim: 'A1'})
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
21	{"pseudonim":"A1"}

Graph mutating

Adding edges




```
MATCH p=()-[e]->()
RETURN ID(e), p
```

"ID(e)"	"p"
3	[{"pseudonim":"a6"},{"id":"A6_A7"},{"pseudonim":"A7"}]
40	[{"pseudonim":"A1"},{"id":"A1_A2"},{"pseudonim":"A2"}]
39	[{"pseudonim":"A7"},{"id":"A7_A8"},{"pseudonim":"A8"}]
43	[{"pseudonim":"a6"},{"id":"A6_A8"},{"pseudonim":"A8"}]
42	[{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
74	[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"}]
0	[{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"}]
41	[{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"}]
1	[{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"}]
94	[{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"}]
2	[{"pseudonim":"A5"},{"id":"A5_A6"},{"pseudonim":"a6"}]
75	[{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"a6"}]

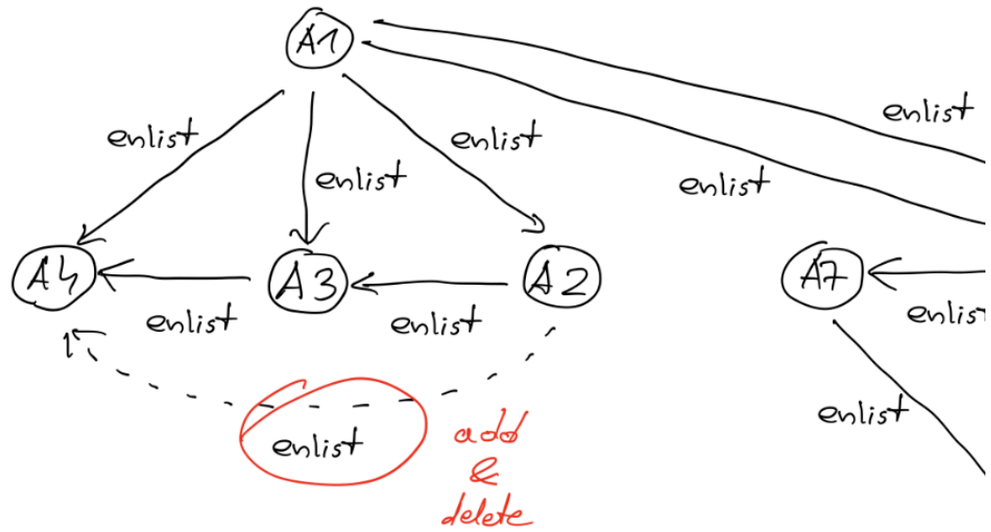
Graph mutating

Adding edges

```
MATCH (from:agent), (to:agent)
WHERE from.pseudonim = 'A2' AND to.pseudonim = 'A4'
CREATE (from)-[:enlist {id: 'A2_A4'}]->(to)
```

```
MATCH p=()-[e]->() RETURN ID(e), p
```

"ID(e)"	"p"
3	[{"pseudonim":"a6"},{"id":"A6_A7"},{"pseudonim":"A7"}]
40	[{"pseudonim":"A1"},{"id":"A1_A2"},{"pseudonim":"A2"}]
39	[{"pseudonim":"A7"},{"id":"A7_A8"},{"pseudonim":"A8"}]
43	[{"pseudonim":"a6"},{"id":"A6_A8"},{"pseudonim":"A8"}]
42	[{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
74	[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"}]
0	[{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"}]
95	[{"pseudonim":"A2"},{"id":"A2_A4"},{"pseudonim":"A4"}]
41	[{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"}]
1	[{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"}]
94	[{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"}]
2	[{"pseudonim":"A5"},{"id":"A5_A6"},{"pseudonim":"a6"}]
75	[{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"a6"}]



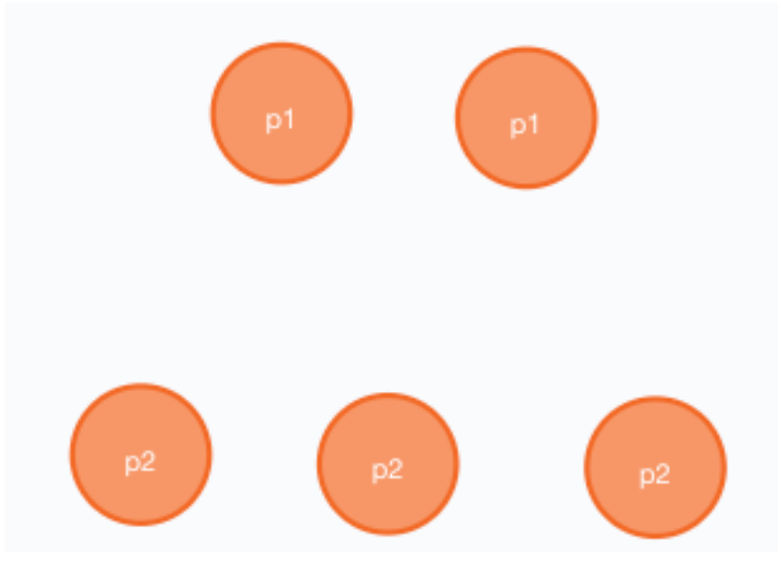
Graph mutating

Adding edges - multiple *from* and *to*

We have 2 x 'p1' and 3 x 'p2' vertex

```
MATCH (v)
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"p":"p1"}
1	{"p":"p1"}
2	{"p":"p2"}
20	{"p":"p2"}
21	{"p":"p2"}




```
MATCH ()-[e]->()
RETURN ID(e), e

(no changes, no records)
```

Graph mutating

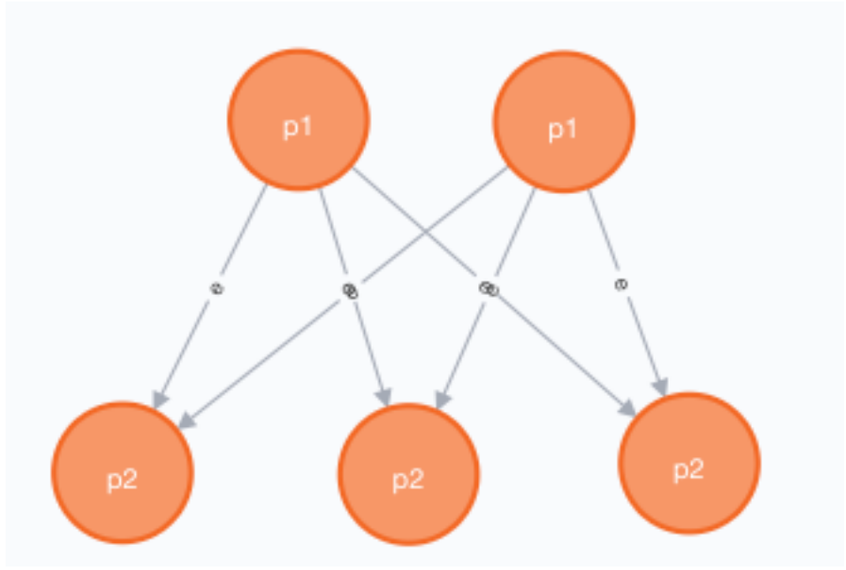
Adding edges - multiple *from* and *to*

We have 2 x 'p1' and 3 x 'p2' vertex

```
MATCH (from {p: 'p1'}), (to {p: 'p2'})
CREATE (from)-[:e]->(to)

Created 6 relationships, completed after 24 ms.
```

```
MATCH ()-[e]->()
RETURN ID(e), e
```

"ID(e)"	"e"
3	{}
0	{}
4	{}
1	{}
5	{}
2	{}



Graph mutating

Adding property to vertex

```
MATCH (v:agent {pseudonim: 'A1'})
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
21	{"pseudonim":"A1"}

```
MATCH (v:agent {pseudonim: 'A1'})
WHERE ID(v) = 21 SET v.action = 'remove'
RETURN ID(v), v
```

"ID(v)"	"v"
21	{"action":"remove","pseudonim":"A1"}

```
MATCH (v:agent {pseudonim: 'A1'})
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
21	{"action":"remove","pseudonim":"A1"}

Graph mutating

Adding property to edge

```
MATCH p=()-[e]->()
RETURN ID(e), p
```

"ID(e)"	"p"
3	[{"pseudonim":"a6"},{"id":"A6_A7"},{"pseudonim":"A7"}]
40	[{"pseudonim":"A1"},{"id":"A1_A2"},{"pseudonim":"A2"}]
39	[{"pseudonim":"A7"},{"id":"A7_A8"},{"pseudonim":"A8"}]
43	[{"pseudonim":"a6"},{"id":"A6_A8"},{"pseudonim":"A8"}]
42	[{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
74	[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"}]
0	[{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"}]
95	[{"pseudonim":"A2"},{"id":"A2_A4"},{"pseudonim":"A4"}]
41	[{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"}]
1	[{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"}]
94	[{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"}]
2	[{"pseudonim":"A5"},{"id":"A5_A6"},{"pseudonim":"a6"}]
75	[{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"a6"}]

```
MATCH p=()-[e]->()
WHERE ID(e) = 95
SET e.status = 'new'
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4","status":"new"},{"pseudonim":"A4"}]

Graph mutating

Updating data: vertex

```
MATCH (v:agent)
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
1	{"pseudonim":"A7"}
2	{"pseudonim":"A2"}
20	{"pseudonim":"A8"}
21	{"action":"remove","pseudonim":"A1"}
38	{"pseudonim":"A3"}
74	{"pseudonim":"A4"}
94	{"pseudonim":"A5"}
95	{"pseudonim":"a6"}

Graph mutating

Updating data: vertex

```
MATCH (v:agent)
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
1	{"pseudonim":"A7"}
2	{"pseudonim":"A2"}
20	{"pseudonim":"A8"}
21	{"action":"remove","pseudonim":"A1"}
38	{"pseudonim":"A3"}
74	{"pseudonim":"A4"}
94	{"pseudonim":"A5"}
95	{"pseudonim":"a6"}

```
MATCH (v)
WHERE v.pseudonim = 'a6'
SET v.pseudonim = 'A6'

MATCH (v:agent)
RETURN ID(v), v
```

"ID(v)"	"v"
0	{"pseudonim":"A1"}
1	{"pseudonim":"A7"}
2	{"pseudonim":"A2"}
20	{"pseudonim":"A8"}
21	{"action":"remove","pseudonim":"A1"}
38	{"pseudonim":"A3"}
74	{"pseudonim":"A4"}
94	{"pseudonim":"A5"}
95	{"pseudonim":"A6"}

Graph mutating

Updating data: edge

```
MATCH p=()-[e]->()
WHERE ID(e) = 95
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4","status":"new"},{"pseudonim":"A4"}]

```
MATCH p=()-[e]->()
WHERE ID(e) = 9
SET e.status = 'to_delete'
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4","status":"to_delete"},{"pseudonim":"A4"}]

Graph mutating

Updating data: edge

Different variants of update verification query

```
MATCH p=()-[e]->()
WHERE e.status = 'new
SET e.status = 'to_delete'
RETURN ID(e), p
```

```
MATCH p=()-[e]->()
WHERE e.id = 'A2_A4'
SET e.status = 'to_delete'
RETURN ID(e), p
```

```
MATCH p=()-[e]->()
WHERE EXISTS(e.status)
SET e.status = 'to_delete'
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4","status":"to_delete"},{"pseudonim":"A4"}]

Graph mutating

Removing data: vertex

```
MATCH (v)
WHERE v.action = 'remove'
RETURN ID(v), v
```

"ID(v)"	"v"
21	{"action":"remove","pseudonim":"A1"}

```
MATCH (v)
WHERE v.action = 'remove'
DELETE v

Deleted 1 node, completed after 1 ms.

MATCH (v)
WHERE v.action = 'remove'
RETURN ID(v), v

(no changes, no records)
```

Graph mutating

Removing data: property

```
MATCH p=()-[e]->()
WHERE e.id = 'A2_A4'
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4","status":"to_delete"},{"pseudonim":"A4"}]

```
MATCH p=()-[e]->()
WHERE e.id = 'A2_A4'
REMOVE e.status
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4"},{"pseudonim":"A4"}]

Graph mutating

Removing data: edge

```
MATCH p=()-[e]->()
WHERE e.id = 'A2_A4'
REMOVE e.status
RETURN ID(e), p
```

"ID(e)"	"p"
95	[{"pseudonim":"A2"},{"id":"A2_A4"},{"pseudonim":"A4"}]

```
MATCH p=()-[e]->()
WHERE e.id = 'A2_A4'
DELETE e
RETURN ID(e), p

Deleted 1 relationship, completed after 2 ms.

MATCH p=()-[e]->()
WHERE e.id = 'A2_A4'
RETURN ID(e), p

(no changes, no records)
```

Practical part 3: pathfinding

Finding the simple path

Finding the paths that connect two vertices

We are looking for the simple path between two vertices, no matter if it's from A to B or from B to A.

```
MATCH p=({pseudonim:'A2'})-[*]-({pseudonim:'A8'})
RETURN p
... CUT A LOT OF RESULTS ...
... HOW MANY? ...

MATCH p=({pseudonim:'A2'})-[*]-({pseudonim:'A8'})
RETURN count(*)

70
```

Finding the simple path

Finding the paths that connect two vertices

```
MATCH p=({pseudonim:'A2'})-[*]-({pseudonim:'A8'}RETURN p
```

"p"
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A1_A3"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A3_A4"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A1_A3"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A1_A4"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
...cut 65 results...
[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A1_A4"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A5_A6"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"},{"pseudonim":"A8"},{"id":"A6_A8"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A6_A7"},{"pseudonim":"A7"},{"pseudonim":"A7"},{"id":"A7_A8"},{"pseudonim":"A8"}]

Finding the simple path

Finding the paths that connect two vertices

```
MATCH p=({pseudonim:'A2'})-[*]-({pseudonim:'A8'}RETURN p
```

"p"
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A1_A3"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A3_A4"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A1_A3"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A1_A4"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
...cut 65 results...
[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A1_A4"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A5_A6"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"},{"pseudonim":"A8"},{"id":"A6_A8"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A6_A7"},{"pseudonim":"A7"},{"pseudonim":"A7"},{"id":"A7_A8"},{"pseudonim":"A8"}]

Finding the simple path

Finding the paths that connect two vertices

Be sure to avoid cycles

```
MATCH p=({pseudonim:'A2'})-[*]-({pseudonim:'A8'}RETURN p
```

"p"
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A1_A3"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A3_A4"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A1_A3"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A1_A4"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"}]
...cut 65 results...
[{"pseudonim":"A2"},{"id":"A2_A3"},{"pseudonim":"A3"},{"pseudonim":"A3"},{"id":"A3_A4"},{"pseudonim":"A4"},{"pseudonim":"A4"},{"id":"A1_A4"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A5_A6"},{"pseudonim":"A5"},{"pseudonim":"A5"},{"id":"A5_A8"},{"pseudonim":"A8"},{"pseudonim":"A8"},{"id":"A6_A8"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A6_A7"},{"pseudonim":"A7"},{"pseudonim":"A7"},{"id":"A7_A8"},{"pseudonim":"A8"}]

Finding the simple path

Finding the paths that connect two vertices

```
MATCH path = shortestPath(({pseudonim:'A2'})-[*]-({pseudonim:'A8'}))
RETURN path
```

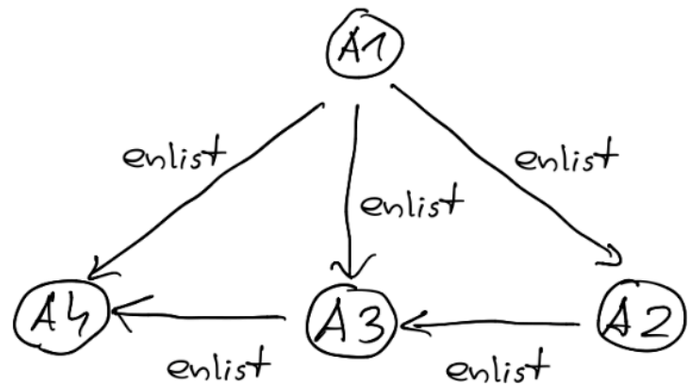
"path"
[{"pseudonim":"A2"},{"id":"A1_A2"},{"pseudonim":"A1"},{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"A6"},{"pseudonim":"A6"},{"id":"A6_A8"},{"pseudonim":"A8"}]

APOC

A Package Of Components or *Awesome Procedures On Cypher*

`subgraphNodes` This procedure expands to subgraph nodes reachable from the start node following relationships to max-level adhering to the label filters.

```
MATCH (n:agent {pseudonim:"A3"})
CALL apoc.path.subgraphNodes(
  n,
  {
    relationshipFilter: "enlist",
    minLevel: 0,
    maxLevel: 1
  }
)
YIELD node
RETURN node
```



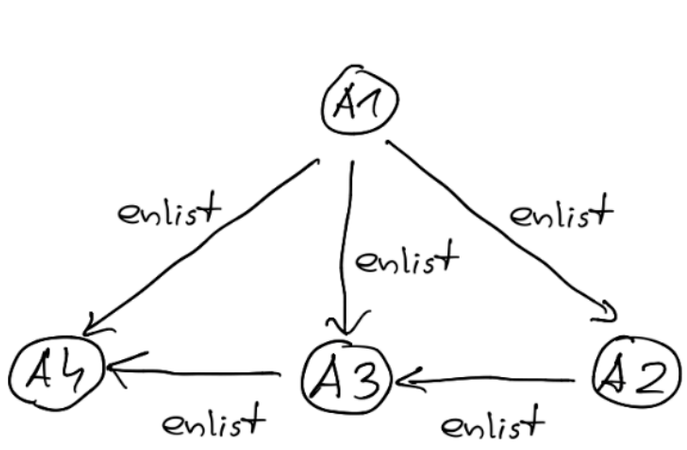
"node"
{"pseudonim":"A3"}
{"pseudonim":"A4"}
{"pseudonim":"A2"}
{"pseudonim":"A1"}

APOC

A Package Of Components or *Awesome Procedures On Cypher*

`subgraphAll` The expand to subgraph procedure expands to subgraph nodes reachable from the start node following relationships to max-level adhering to the label filters. Returns the collection of nodes in the subgraph, and the collection of relationships between all subgraph nodes.

```
MATCH (n:agent {pseudonim:"A3"})
CALL apoc.path.subgraphAll(
  n,
  {
    relationshipFilter: "enlist",
    minLevel: 0,
    maxLevel: 1
  }
)
YIELD nodes, relationships
RETURN nodes, relationships
```



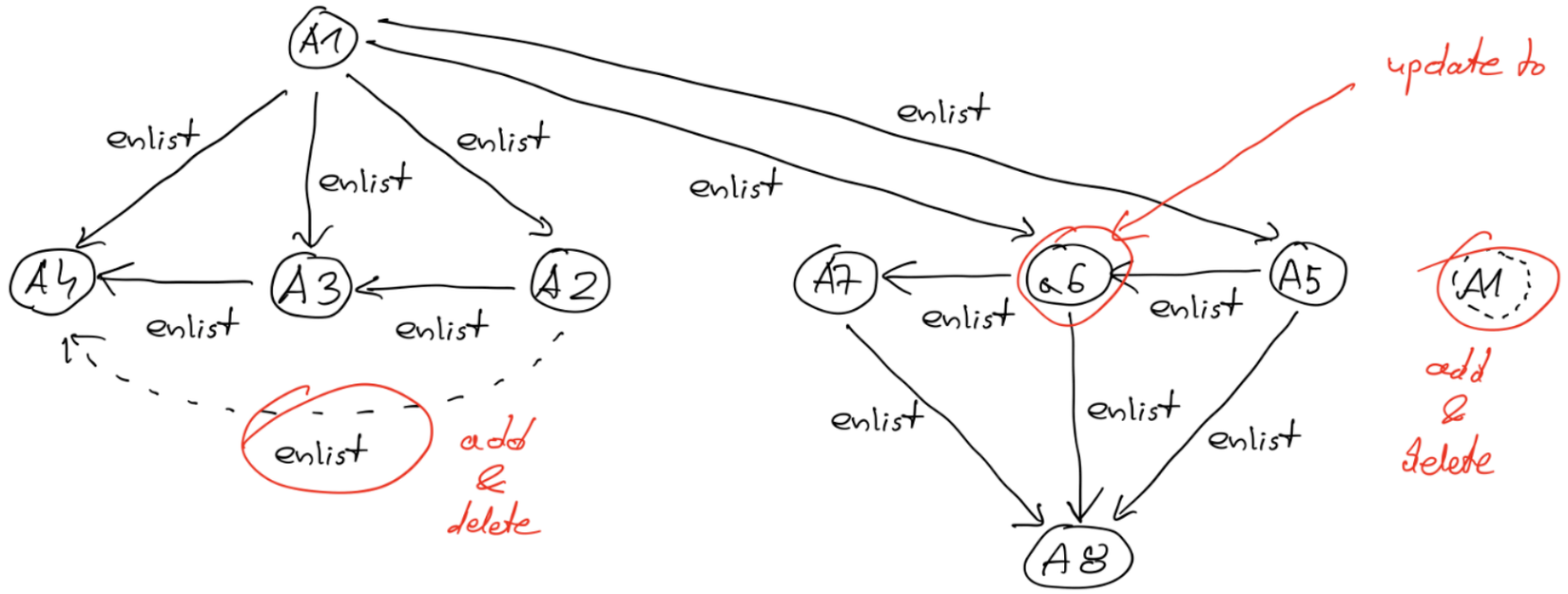
"nodes"	"relationships"
[{"pseudonim":"A3"},{"pseudonim":"A4"},{"pseudonim":"A2"},{"pseudonim":"A1"}]	[{"id":"A3_A4"},{"id":"A2_A3"},{"id":"A1_A4"},{"id":"A1_A3"},{"id":"A1_A2"}]

APOC

A Package Of Components or *Awesome Procedures On Cypher*

`spanningTree` Expands a spanning tree reachable from start node following relationships to max-level adhering to the label filters. The paths returned collectively form a spanning tree.

```
MATCH (n:agent {pseudonim:"A1"})
CALL apoc.path.spanningTree(
  n,
  {
    relationshipFilter: "enlist>",
    maxLevel: 2
  }
)
YIELD path
RETURN path;
```



"path"
[{"pseudonim":"A1"},null]
[{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"a6"}]
[{"pseudonim":"A1"},{"id":"A1_A5"},{"pseudonim":"A5"}]
[{"pseudonim":"A1"},{"id":"A1_A4"},{"pseudonim":"A4"}]
[{"pseudonim":"A1"},{"id":"A1_A3"},{"pseudonim":"A3"}]
[{"pseudonim":"A1"},{"id":"A1_A2"},{"pseudonim":"A2"}]
[{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"a6"},{"pseudonim":"a6"},{"id":"A6_A8"},{"pseudonim":"A8"}]
[{"pseudonim":"A1"},{"id":"A1_A6"},{"pseudonim":"a6"},{"pseudonim":"a6"},{"id":"A6_A7"},{"pseudonim":"A7"}]

Bibliography

- [Ful] Piotr Fulmański, *NoSQL. Theory and examples*, Piotr Fulmański, 2021
- [Hun] Michael Hunger, *Get Started with Cypher: A Beginner's Guide*, `https://neo4j.com/whitepapers/getting-started-with-cypher/`