

Querying graph databases

Graph mutating and pathfinding

NoSQL: Lecture 4

Piotr Fulmański

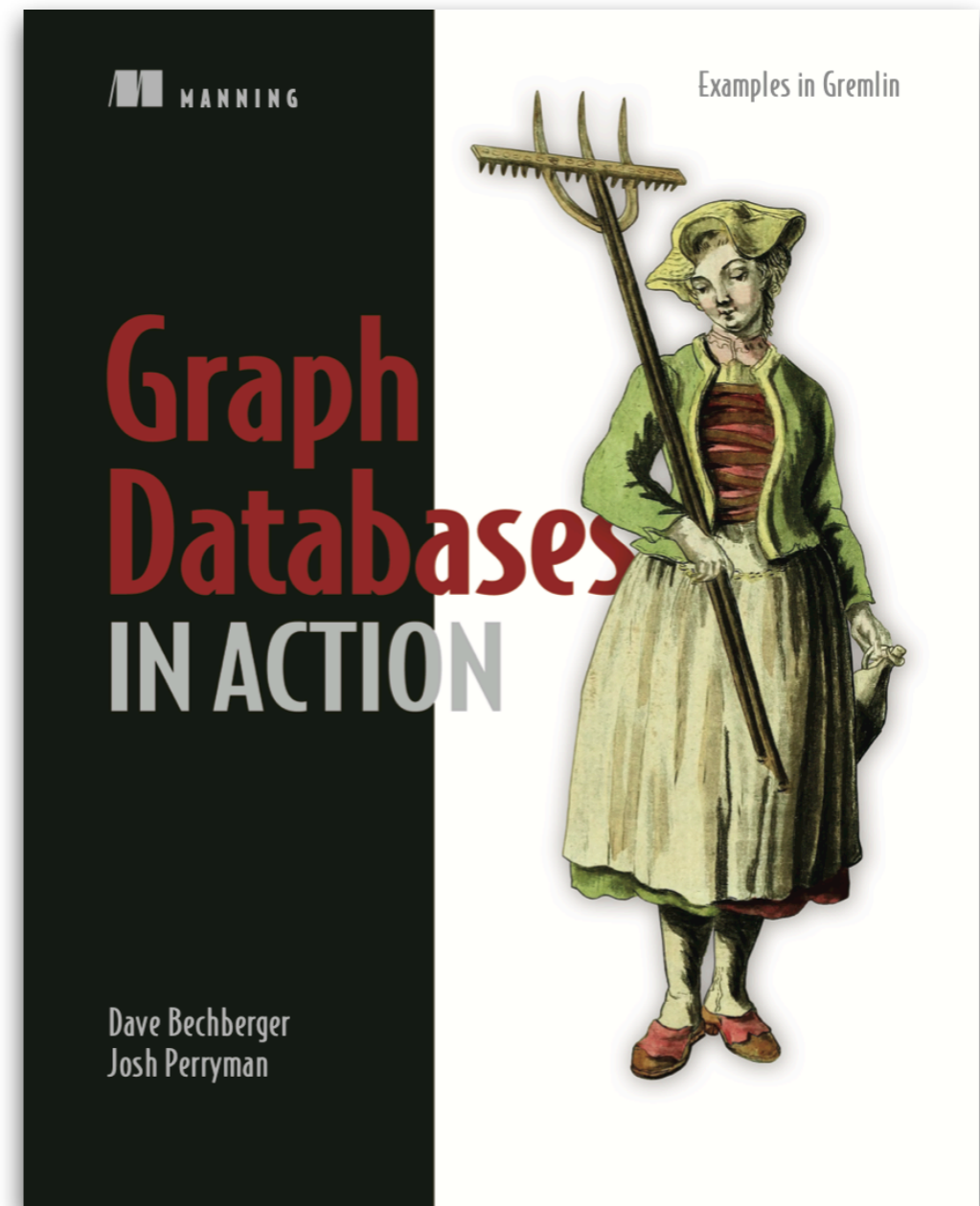


FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE
University of Lodz

Graph databases In Action

by Dave Bechberger
and Josh Perryman

Manning Publications, 2020



Graph mutating

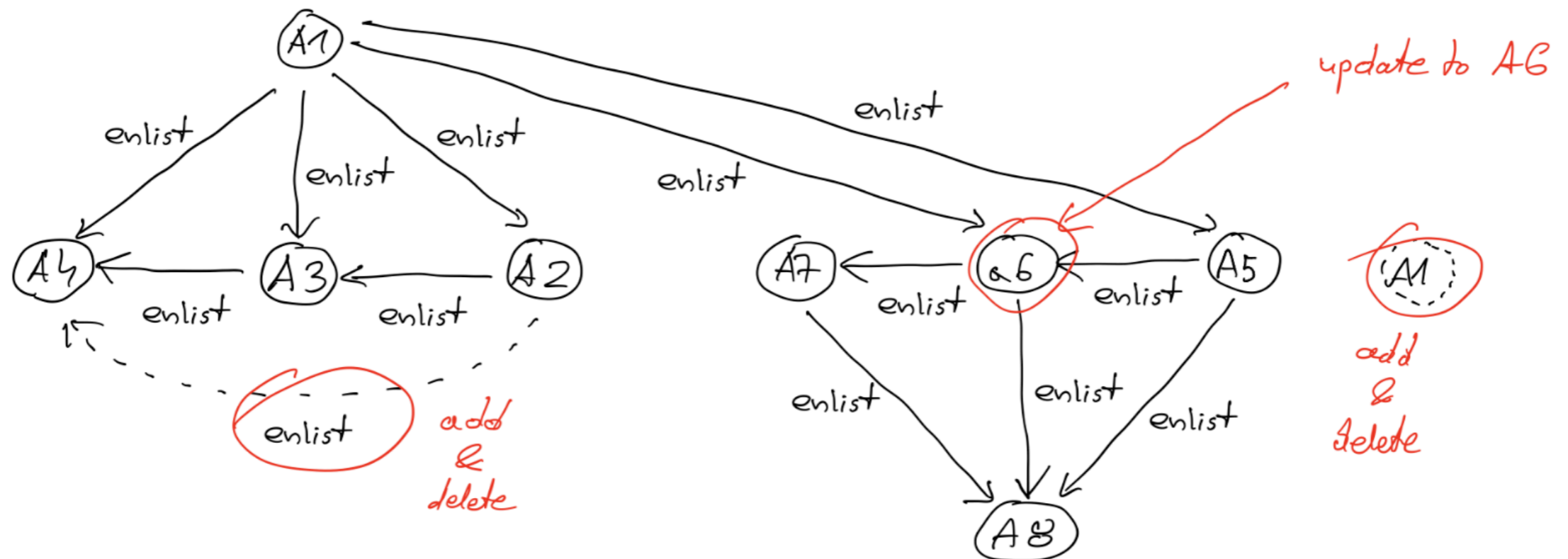
Graph mutating

Mutating simply means changing the graph by adding, modifying, or deleting vertices, edges, and/or properties.

So we will write traversals (yes, we still name it as traversal) to add, modify, and delete vertices, edges, and properties.

Graph mutating

Let's expand the network of secret agents to the following form (use agent2.groovy file to (re)create this data)



Graph mutating

Creating vertices and edges

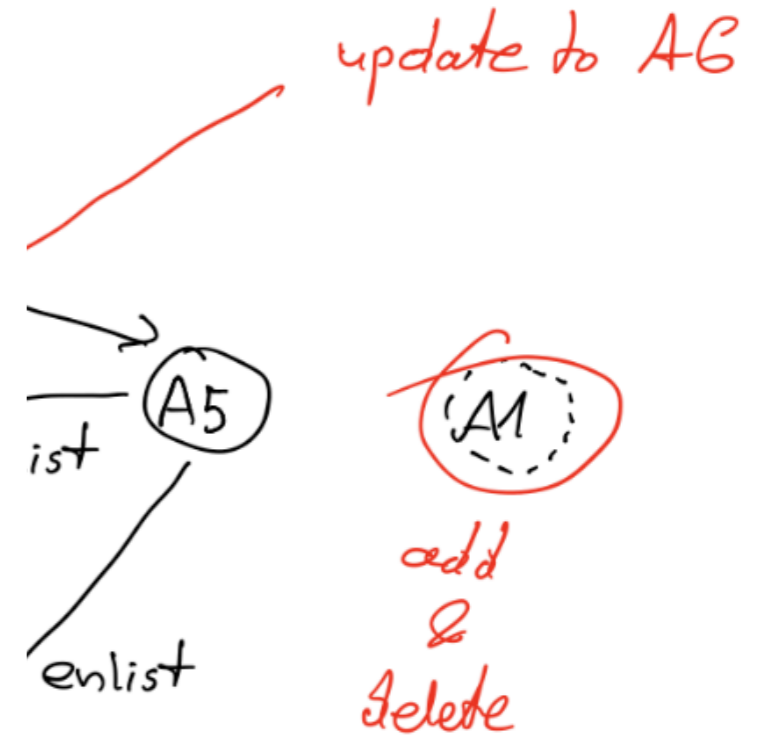
The fundamental concept of creating entities in graph databases isn't all that different from the relational database world. Creating new vertex entities involves adding the appropriate elements and properties. However, creating new edges is a bit more complicated because we need to specify the vertex that belongs at each end of the edge.

Other words, vertices can exist on its own, while every edge must have its endings "attached" to some vertex - we need to specify the inbound and the outbound vertex for the edge.

Graph mutating

Adding vertices

1. Given a traversal source g .
- 2.
- 3.
- g .

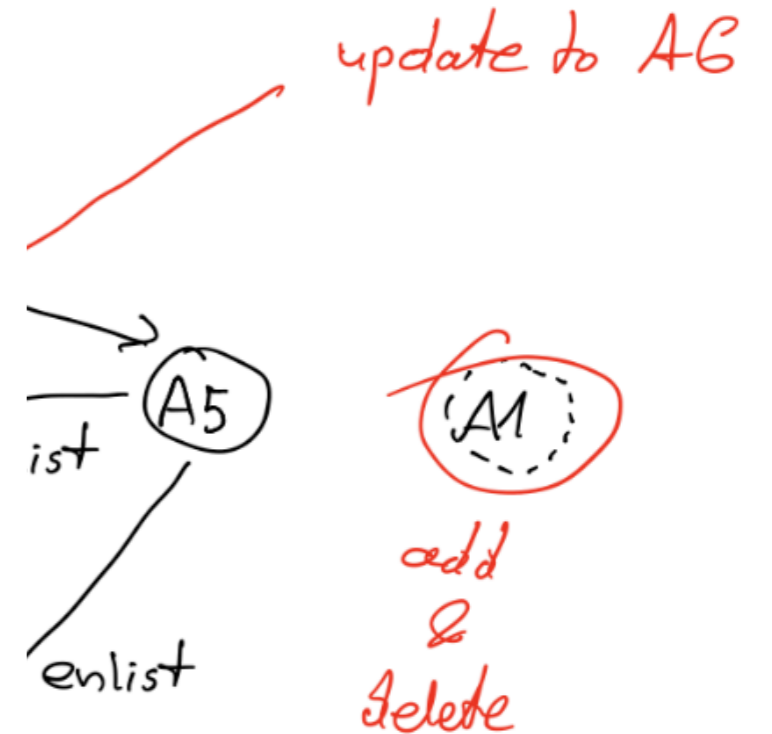


Graph mutating

Adding vertices

1. Given a traversal source `g`.
2. Add a new **vertex** of type *agent*.
- 3.

```
g.  
addV( 'agent' ).
```

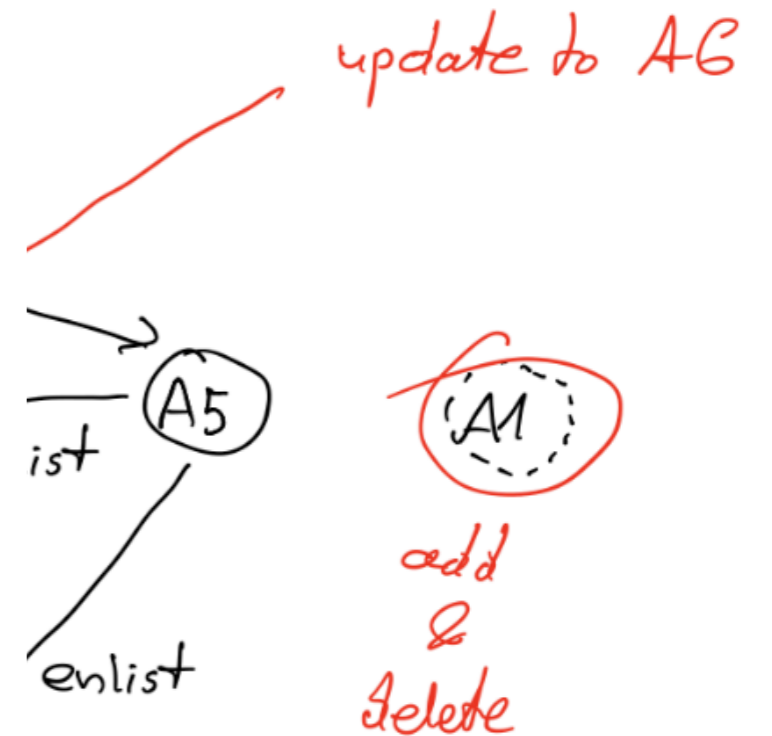


Graph mutating

Adding vertices

1. Given a traversal source `g`.
2. Add a new **vertex** of type *agent*.
3. Add a **property** to that **vertex** with the **key** *pseudonim* and the **value** *A1*.

```
g.  
addV( 'agent' ).  
property( 'pseudonim', 'A1' )
```



Graph mutating

Adding vertices

```
nosql@nosql:~/Desktop$ ./tinkerpop/gremlin-console/bin/gremlin.sh -i
~/Desktop/nosql_2/agents2.groovy
[ cut some lines here ]
      \,,,/
      (o o)
-----o00o-(3)-o00o-----
[ cut some lines here ]
gremlin> graph
==>tinkergraph[vertices:8 edges:12]
gremlin> g.V().elementMap()
==>[id:0,label:agent,pseudonim:A1]
==>[id:2,label:agent,pseudonim:A2]
==>[id:4,label:agent,pseudonim:A3]
==>[id:6,label:agent,pseudonim:A4]
==>[id:8,label:agent,pseudonim:A5]
==>[id:10,label:agent,pseudonim:a6]
==>[id:12,label:agent,pseudonim:A7]
==>[id:14,label:agent,pseudonim:A8]
```

Graph mutating

Adding vertices

```
gremlin> g.V().has('agent', 'pseudonim', 'A1')  
==>v[0]
```

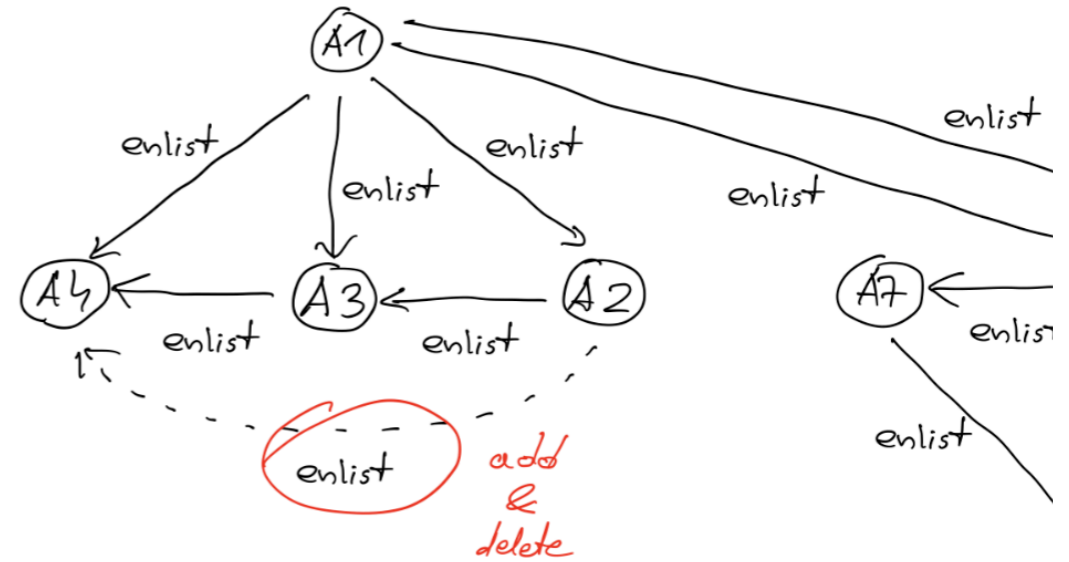
```
gremlin> g.addV('agent').property('pseudonim', 'A1')  
==>v[28]
```

```
gremlin> g.V().has('agent', 'pseudonim', 'A1')  
==>v[0]  
==>v[28]
```

Graph mutating

Adding edges

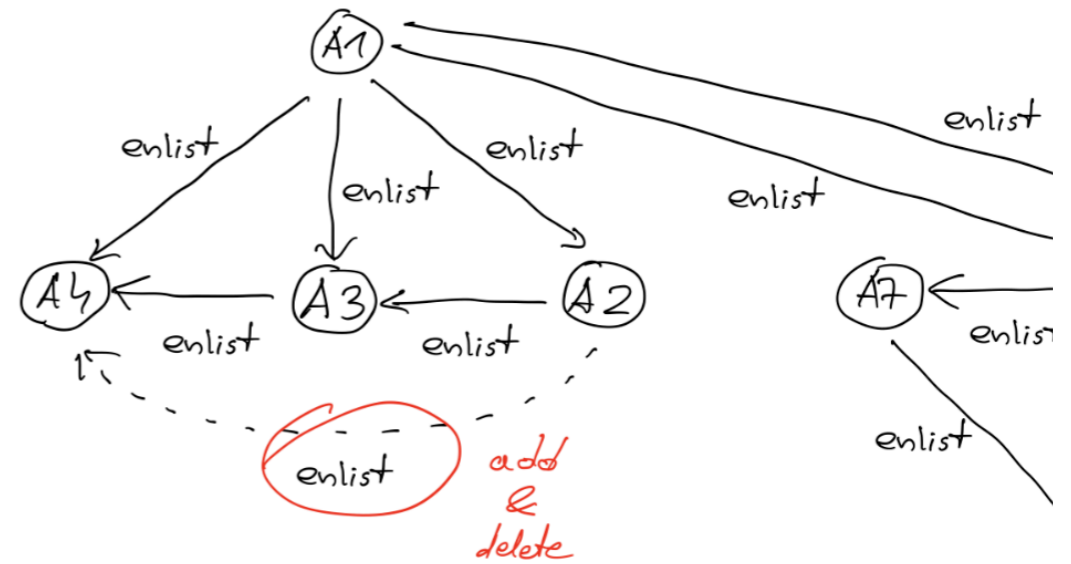
1. Given a traversal source g .
- 2.
- 3.
- 4.
- g .



Graph mutating

Adding edges

1. Given a traversal source `g`.
2. Add a new **edge** with a **label** `enlist`.
- 3.
- 4.
5. `g.addE('enlist')`.



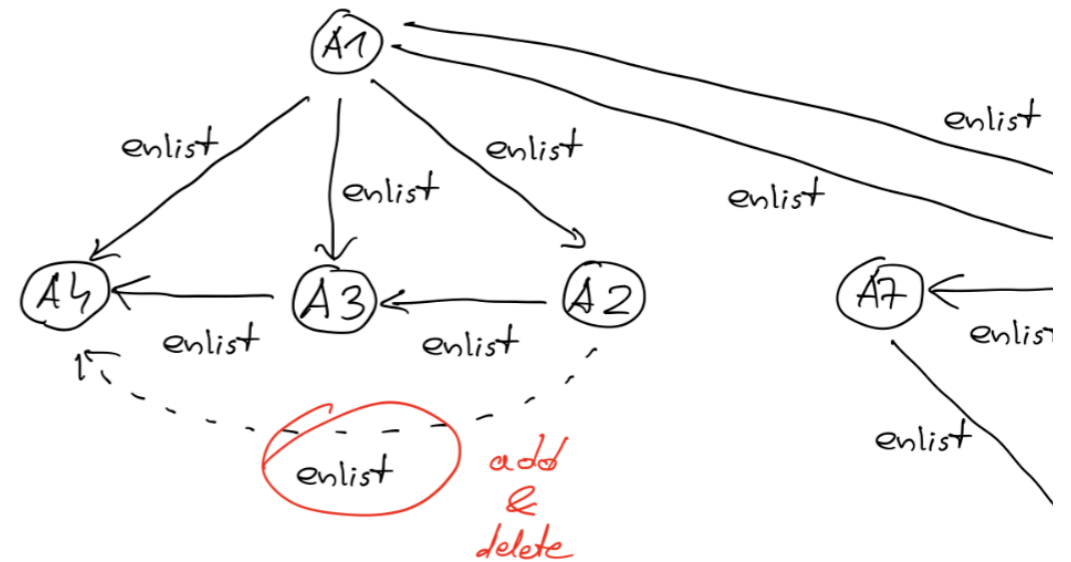
Graph mutating

Adding edges

1. Given a traversal source `g`.
2. Add a new **edge** with a **label** `enlist`.
3. Assign the **outbound vertex** of the **edge**

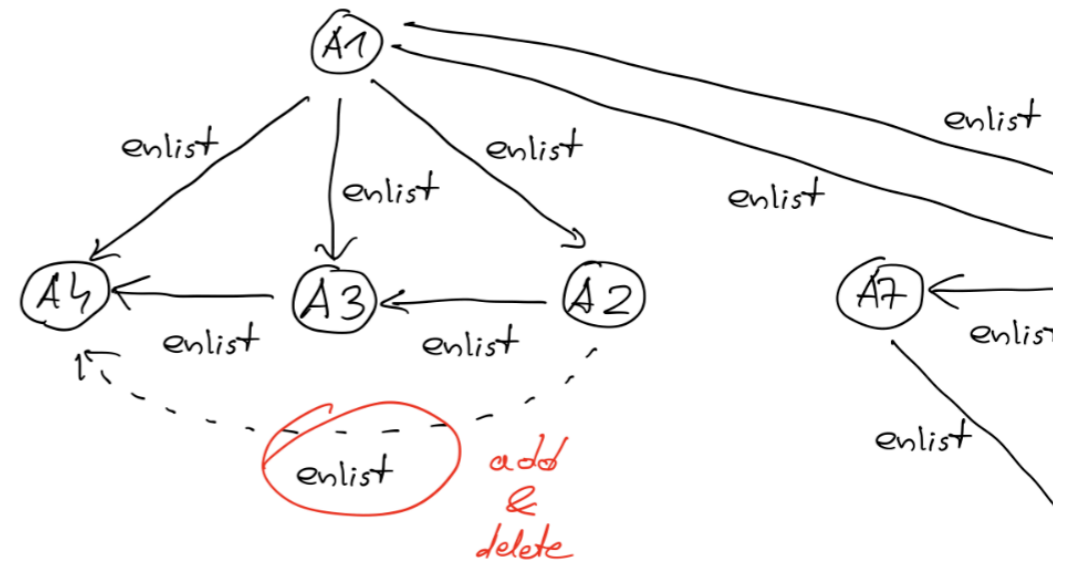
4.

```
g.  
addE( 'enlist' ).  
from(  
  
).
```



Graph mutating

Adding edges

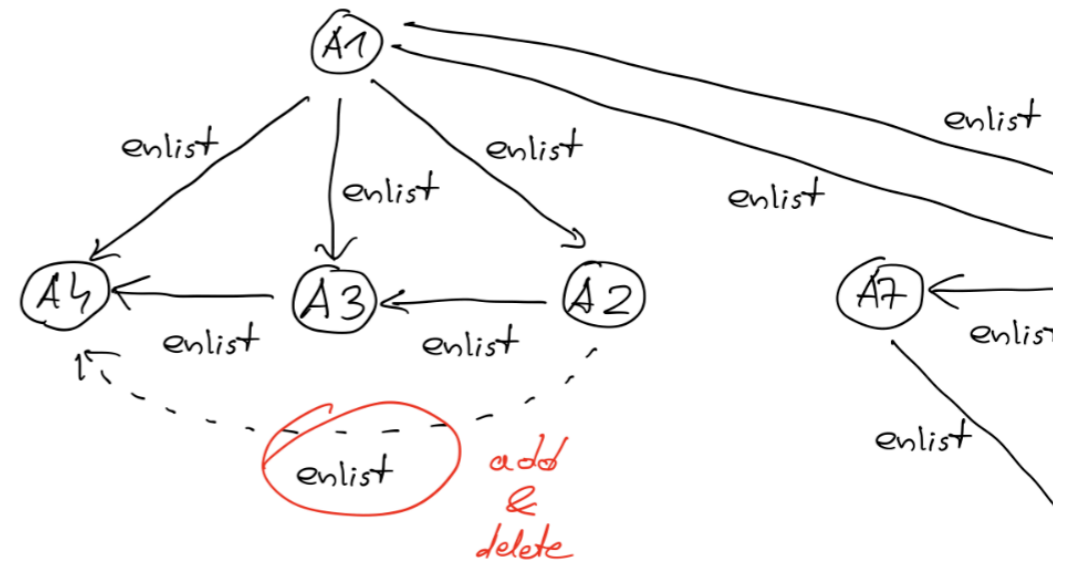


1. Given a traversal source `g`.
2. Add a new **edge** with a **label** `enlist`.
3. Assign the **outbound vertex** of the **edge** to the vertex with the **key** of `pseudonim` and the **value** of `A2`.
4. Assign the **inbound vertex** of the **edge**

```
g.  
addE('enlist').  
from(  
    V().has('agent', 'pseudonim', 'A2')  
).  
to(  
  
)
```


Graph mutating

Adding edges



1. Given a traversal source `g`.
2. Add a new **edge** with a **label** `enlist`.
3. Assign the **outbound vertex** of the **edge** to the vertex with the **key** of `pseudonim` and the **value** of `A2`.
4. Assign the **inbound vertex** of the **edge** to the vertex with the **key** of `pseudonim` and the **value** of `A4`.

```
g.  
addE('enlist').  
from(  
    V().has('agent', 'pseudonim', 'A2')  
).  
to(  
    V().has('agent', 'pseudonim', 'A4')  
)
```

Graph mutating

Adding edges

```
gremlin> g.E().elementMap()  
==>[id:16,label:enlist,IN:[id:2,label:agent],OUT:[id:0,label:agent]]  
==>[id:17,label:enlist,IN:[id:4,label:agent],OUT:[id:0,label:agent]]  
==>[id:18,label:enlist,IN:[id:6,label:agent],OUT:[id:0,label:agent]]  
==>[id:19,label:enlist,IN:[id:8,label:agent],OUT:[id:0,label:agent]]  
==>[id:20,label:enlist,IN:[id:10,label:agent],OUT:[id:0,label:agent]]  
==>[id:21,label:enlist,IN:[id:4,label:agent],OUT:[id:2,label:agent]]  
==>[id:22,label:enlist,IN:[id:6,label:agent],OUT:[id:4,label:agent]]  
==>[id:23,label:enlist,IN:[id:10,label:agent],OUT:[id:8,label:agent]]  
==>[id:24,label:enlist,IN:[id:14,label:agent],OUT:[id:8,label:agent]]  
==>[id:25,label:enlist,IN:[id:12,label:agent],OUT:  
[id:10,label:agent]]  
==>[id:26,label:enlist,IN:[id:14,label:agent],OUT:  
[id:10,label:agent]]  
==>[id:27,label:enlist,IN:[id:14,label:agent],OUT:  
[id:12,label:agent]]
```

Graph mutating

Adding edges

```
gremlin> g.addE('enlist').from(V().has('agent','pseudonim','A2')).  
to(V().has('agent','pseudonim','A4'))  
==>e[30][2-enlist->6]  
gremlin> g.E().elementMap()  
==>[id:16,label:enlist,IN:[id:2,label:agent],OUT:[id:0,label:agent]]  
==>[id:17,label:enlist,IN:[id:4,label:agent],OUT:[id:0,label:agent]]  
==>[id:18,label:enlist,IN:[id:6,label:agent],OUT:[id:0,label:agent]]  
==>[id:19,label:enlist,IN:[id:8,label:agent],OUT:[id:0,label:agent]]  
==>[id:20,label:enlist,IN:[id:10,label:agent],OUT:[id:0,label:agent]]  
==>[id:21,label:enlist,IN:[id:4,label:agent],OUT:[id:2,label:agent]]  
==>[id:22,label:enlist,IN:[id:6,label:agent],OUT:[id:4,label:agent]]  
==>[id:23,label:enlist,IN:[id:10,label:agent],OUT:[id:8,label:agent]]  
==>[id:24,label:enlist,IN:[id:14,label:agent],OUT:[id:8,label:agent]]  
==>[id:25,label:enlist,IN:[id:12,label:agent],OUT:[id:10,label:agent]]  
==>[id:26,label:enlist,IN:[id:14,label:agent],OUT:[id:10,label:agent]]  
==>[id:27,label:enlist,IN:[id:14,label:agent],OUT:[id:12,label:agent]]  
==>[id:30,label:enlist,IN:[id:6,label:agent],OUT:[id:2,label:agent]]
```

Graph mutating

Adding edges - multiple *from* and *to*

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

```
g.  
addE( 'enlist' ).  
from(  
    V().has( 'p', 'p1' )  
).  
to(  
    V().has( 'p', 'p2' )  
)
```

Graph mutating

Adding edges - multiple *from* and *to*

The screenshot shows the Gremlify web interface. The browser address bar displays "Brak ochrony — gremlify.com". The interface includes a top navigation bar with the Gremlify logo, "Powered By Apache Tinkerpop", and buttons for "Run", "Save", "Fork", "Clear", "Revert", "Untitled Workspace", and "Sign In".

The main workspace is divided into two panels. The left panel contains a query editor with the following code:

```
1 g.addE('enlist').from(V().
2 has('p','p1')).to(V().has('p','p2'))
```

Below the query editor is a "query" tab with a plus sign and a menu icon. The right panel displays a graph visualization with five red circular nodes. The nodes are labeled "Vertex (5149)", "Vertex (5155)", "Vertex (5163)", "Vertex (5166)", and "Vertex (5160)". An edge labeled "enlist-(5190)" connects the top-left node (5149) to the bottom node (5160).

The bottom panel shows a JSON response for the query:

```
[ 1 item
  0 : { 6 items
    "id" : 5190
    "label" : "enlist"
    "inVLabel" : "Vertex"
    "outVLabel" : "Vertex"
```

On the right side of the interface, there are icons for "GRAPH" and "SCHEMA".

Graph mutating

Adding edges - multiple *from* and *to*

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

```
g.  
V().  
has('p', 'p1').  
as('a').  
V().has('p', 'p2').  
addE('enlist').from('a')
```

Graph mutating

Adding edges - multiple *from* and *to*

The screenshot shows the Gremlify web interface. The browser address bar displays "Brak ochrony — gremlify.com". The interface includes a top navigation bar with buttons for "Run", "Save", "Fork", "Clear", "Revert", "Untitled Workspace", and "Sign In".

The main workspace is divided into two panels. The left panel contains a code editor with the following Gremlin query:

```
1 g.V().has('p', 'p1').as('a').
2 V().has('p', 'p2').addE('enlist').from('a')
```

Below the code editor, the "query" tab is active, showing a JSON response for the executed query:

```
[ 6 items
  0 : { 6 items
    "id" : 5184
    "label" : "enlist"
    "inVLabel" : "Vertex"
    "outVLabel" : "Vertex"
```

The right panel displays a graph visualization with five red circular vertices. The vertices are labeled "Vertex" with their IDs: 5149, 5155, 5160, 5163, and 5166. The edges are labeled "enlist" with their IDs: 5184, 5185, 5186, 5187, and 5189. The graph structure is as follows:

- Vertex 5149 is connected to Vertex 5160 (edge 5184) and Vertex 5163 (edge 5187).
- Vertex 5155 is connected to Vertex 5163 (edge 5185) and Vertex 5166 (edge 5186).
- Vertex 5160 is connected to Vertex 5163 (edge 5188).
- Vertex 5163 is connected to Vertex 5166 (edge 5189).

The interface also features a sidebar on the left with various icons and a "SCHEMA" tab on the right.

Graph mutating

Adding property to vertex

```
gremlin> g.V().elementMap()
==> [id:0,label:agent,pseudonim:A1]
==> [id:2,label:agent,pseudonim:A2]
==> [id:4,label:agent,pseudonim:A3]
==> [id:6,label:agent,pseudonim:A4]
==> [id:8,label:agent,pseudonim:A5]
==> [id:10,label:agent,pseudonim:a6]
==> [id:12,label:agent,pseudonim:A7]
==> [id:28,label:agent,pseudonim:A1] <-- we need this
==> [id:14,label:agent,pseudonim:A8]
gremlin> g.V(28).elementMap()
gremlin> <-- what's wrong?
```


Graph mutating

Adding property to vertex

```
gremlin> g.V(28L).elementMap()
```

```
==>[id:28,label:agent,pseudonim:A1]
```

```
gremlin> v = g.V(28L)
```

```
==>v[28]
```

```
gremlin> v.property('action','remove')
```

The traversal strategies are complete and the traversal can no longer be modulated

Type ':help' or ':h' for help.

Display stack trace? [yN]

```
gremlin> v.class
```

```
==>class org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.DefaultGraphTraversal
```

```
gremlin> v = g.V(28L).next()
```

```
==>v[28]
```

```
gremlin> v.class
```

```
==>class org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerVertex
```

```
gremlin> v.property('action','remove')
```

```
==>vp[action->remove]
```

```
gremlin> g.V(28L).elementMap()
```

```
==>[id:28,label:agent,action:remove,pseudonim:A1]
```

Graph mutating

Adding property to edge

```
gremlin> g.E(30)
```

```
gremlin> g.E(30L)
```

```
==>e[30][2-enlist->6]
```

```
gremlin> g.E(30L).elementMap()
```

```
==>[id:30,label:enlist,
```

```
IN:[id:6,label:agent],
```

```
OUT:[id:2,label:agent]]
```

```
gremlin> g.E(30L).property('status','new')
```

```
==>e[30][2-enlist->6]
```

```
gremlin> g.E(30L).elementMap()
```

```
==>[id:30,label:enlist,
```

```
IN:[id:6,label:agent],
```

```
OUT:[id:2,label:agent],status:new]
```

Graph mutating

Updating data: vertex

```
gremlin> g.V().has('agent','pseudonim','a6').  
elementMap()
```

```
==>[id:10,label:agent,pseudonim:a6]
```

```
gremlin> g.V(10L).elementMap()
```

```
==>[id:10,label:agent,pseudonim:a6]
```

```
gremlin> g.V().has('agent','pseudonim','a6').  
property('pseudonim','A6')
```

```
==>v[10]
```

```
gremlin> g.V(10L).elementMap()
```

```
==>[id:10,label:agent,pseudonim:A6]
```

Graph mutating

Updating data: edge

```
gremlin> g.E().has('enlist','status','new')
==>e[30][2-enlist->6]
gremlin> g.E().has('status','new')
==>e[30][2-enlist->6]
gremlin> g.E().has('status')
==>e[30][2-enlist->6]
gremlin> g.E().has('status').elementMap()
==>[id:30,label:enlist,
IN:[id:6,label:agent],
OUT:[id:2,label:agent],status:new]
gremlin> g.E().has('status').property('status','to_delete')
==>e[30][2-enlist->6]
gremlin> g.E().has('status').elementMap()
==>[id:30,label:enlist,
IN:[id:6,label:agent],
OUT:[id:2,label:agent],status:to_delete]
```

Graph mutating

Removing data: vertex

```
gremlin> g.V().has('action','remove').elementMap()  
==>[id:28,label:agent,action:remove,pseudonim:A1]  
gremlin> g.V(28L).elementMap()  
==>[id:28,label:agent,action:remove,pseudonim:A1]  
gremlin> g.V().has('action','remove').drop()  
gremlin> g.V(28L).elementMap()
```

Graph mutating

Removing data: edge

```
gremlin> g.E().has('status','to_delete').  
elementMap()  
==>[id:30,label:enlist,  
IN:[id:6,label:agent],  
OUT:[id:2,label:agent],status:to_delete]  
gremlin> g.E(30L).elementMap()  
==>[id:30,label:enlist,  
IN:[id:6,label:agent],  
OUT:[id:2,label:agent],status:to_delete]  
gremlin> g.E().has('status','to_delete').drop()  
gremlin> g.E(30L).elementMap()
```

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.

```
g.V().  
has('pseudonim', 'A2').  
until(  
    has('pseudonim', 'A8')  
).  
repeat(  
    both('enlist')  
).  
path()
```


Finding the simple path

Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim','A2').until(has('pseudonim','A8')).repeat(both('enlist')).path()
==>[v[2],v[0],v[8],v[14]]
==>[v[2],v[0],v[10],v[14]]
==>[v[2],v[4],v[0],v[8],v[14]]
==>[v[2],v[4],v[0],v[10],v[14]]
==>[v[2],v[0],v[8],v[10],v[14]]
==>[v[2],v[0],v[10],v[12],v[14]]
==>[v[2],v[0],v[10],v[8],v[14]]
==>[v[2],v[4],v[6],v[0],v[8],v[14]]
==>[v[2],v[4],v[6],v[0],v[10],v[14]]
==>[v[2],v[4],v[0],v[8],v[10],v[14]]
==>[v[2],v[4],v[0],v[10],v[12],v[14]]
==>[v[2],v[4],v[0],v[10],v[8],v[14]]
==>[v[2],v[4],v[2],v[0],v[8],v[14]]
==>[v[2],v[4],v[2],v[0],v[10],v[14]]
==>[v[2],v[0],v[2],v[0],v[8],v[14]]
==>[v[2],v[0],v[2],v[0],v[10],v[14]]
==>[v[2],v[0],v[4],v[0],v[8],v[14]]
==>[v[2],v[0],v[4],v[0],v[10],v[14]]
==>[v[2],v[0],v[6],v[0],v[8],v[14]]
==>[v[2],v[0],v[6],v[0],v[10],v[14]]
==>[v[2],v[0],v[8],v[10],v[12],v[14]]
==>[v[2],v[0],v[8],v[10],v[8],v[14]]
==>[v[2],v[0],v[8],v[0],v[8],v[14]]
==>[v[2],v[0],v[8],v[0],v[10],v[14]]
==>[v[2],v[0],v[10],v[12],v[10],v[14]]
==>[v[2],v[0],v[10],v[0],v[8],v[14]]
==>[v[2],v[0],v[10],v[0],v[10],v[14]]
==>[v[2],v[0],v[10],v[8],v[10],v[14]]
```

Finding the simple path

Finding the paths that connect two vertices

Be sure to avoid cycles

```
gremlin> g.V().has('pseudonim','A2').until(has('pseudonim','A8')).repeat(both('enlist')).path()
==>[v[2],v[0],v[8],v[14]]
==>[v[2],v[0],v[10],v[14]]
==>[v[2],v[4],v[0],v[8],v[14]]
==>[v[2],v[4],v[0],v[10],v[14]]
==>[v[2],v[0],v[8],v[10],v[14]]
==>[v[2],v[0],v[10],v[12],v[14]]
==>[v[2],v[0],v[10],v[8],v[14]]
==>[v[2],v[4],v[6],v[0],v[8],v[14]]
==>[v[2],v[4],v[6],v[0],v[10],v[14]]
==>[v[2],v[4],v[0],v[8],v[10],v[14]]
==>[v[2],v[4],v[0],v[10],v[12],v[14]]
==>[v[2],v[4],v[0],v[10],v[8],v[14]]
==>[v[2],v[4],v[2],v[0],v[8],v[14]]
==>[v[2],v[4],v[2],v[0],v[10],v[14]]
==>[v[2],v[0],v[2],v[0],v[8],v[14]]
==>[v[2],v[0],v[2],v[0],v[10],v[14]]
==>[v[2],v[0],v[4],v[0],v[8],v[14]]
==>[v[2],v[0],v[4],v[0],v[10],v[14]]
==>[v[2],v[0],v[6],v[0],v[8],v[14]]
==>[v[2],v[0],v[6],v[0],v[10],v[14]]
==>[v[2],v[0],v[8],v[10],v[12],v[14]]
==>[v[2],v[0],v[8],v[10],v[8],v[14]]
==>[v[2],v[0],v[8],v[0],v[8],v[14]]
==>[v[2],v[0],v[8],v[0],v[10],v[14]]
==>[v[2],v[0],v[10],v[12],v[10],v[14]]
==>[v[2],v[0],v[10],v[0],v[8],v[14]]
==>[v[2],v[0],v[10],v[0],v[10],v[14]]
==>[v[2],v[0],v[10],v[8],v[10],v[14]]
```

Finding the simple path

Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim', 'A2').  
until(has('pseudonim', 'A8')).  
repeat(both('enlist')).path()  
...  
==>[v[2], v[0], v[2], v[0], v[8], v[14]]  
...  
==>[v[2], v[0], v[10], v[8], v[10], v[14]]  
...  
==>[v[2], v[4], v[6], v[4], v[6], v[0], v[8], v[14]]  
...  
gremlin>
```

Be sure to avoid cycles

Finding the simple path

Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim', 'A2').  
until(has('pseudonim', 'A8')).  
repeat(both('enlist')).path()  
...  
==>[v[2], v[0], v[2], v[0], v[8], v[14]]  
...  
==>[v[2], v[0], v[10], v[8], v[10], v[14]]  
...  
==>[v[2], v[4], v[6], v[4], v[6], v[0], v[8], v[14]]  
...  
gremlin>
```

Finding the simple path

Finding the paths that connect two vertices

`simplePath()` - Filters out traversers that visit the same vertex more than once.

Using this step, we can update our traversal to find the simple path between node A and B.

```
g.V().has('pseudonim', 'A2').  
until(  
  has('pseudonim', 'A8')  
).  
repeat(  
  both('enlist').simplePath()  
).  
path()
```

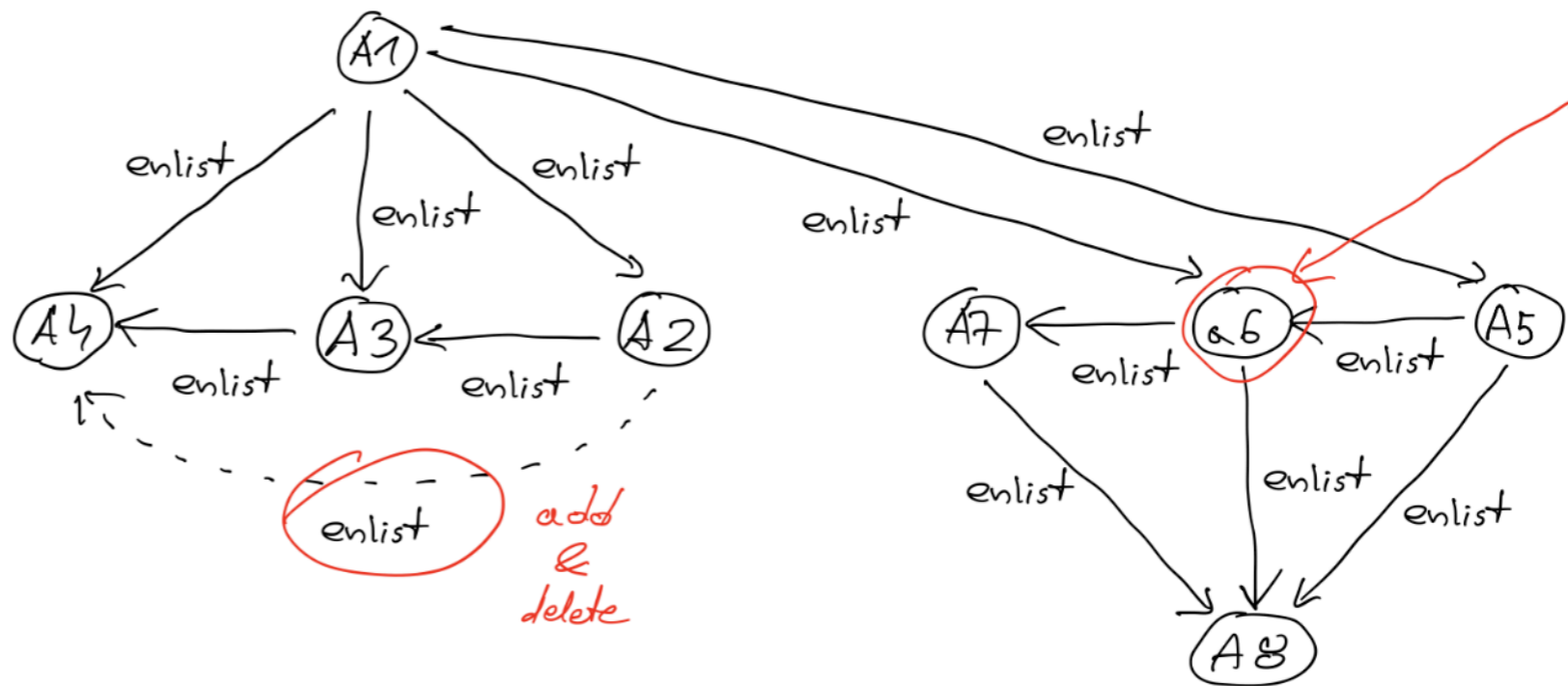
Finding the simple path

Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim','A2').until(has('pseudonim','A8')).repeat(both('enlist').simplePath()).path()
```

```
==> [v[2],v[0],v[8],v[14]]
==> [v[2],v[0],v[10],v[14]]
==> [v[2],v[4],v[0],v[8],v[14]]
==> [v[2],v[4],v[0],v[10],v[14]]
==> [v[2],v[0],v[8],v[10],v[14]]
==> [v[2],v[0],v[10],v[12],v[14]]
==> [v[2],v[0],v[10],v[8],v[14]]
==> [v[2],v[4],v[6],v[0],v[8],v[14]]
==> [v[2],v[4],v[6],v[0],v[10],v[14]]
==> [v[2],v[4],v[0],v[8],v[10],v[14]]
==> [v[2],v[4],v[0],v[10],v[12],v[14]]
==> [v[2],v[4],v[0],v[10],v[8],v[14]]
==> [v[2],v[0],v[8],v[10],v[12],v[14]]
==> [v[2],v[4],v[6],v[0],v[8],v[10],v[14]]
==> [v[2],v[4],v[6],v[0],v[10],v[12],v[14]]
==> [v[2],v[4],v[6],v[0],v[10],v[8],v[14]]
==> [v[2],v[4],v[0],v[8],v[10],v[12],v[14]]
==> [v[2],v[4],v[6],v[0],v[8],v[10],v[12],v[14]]
```

```
gremlin>
```



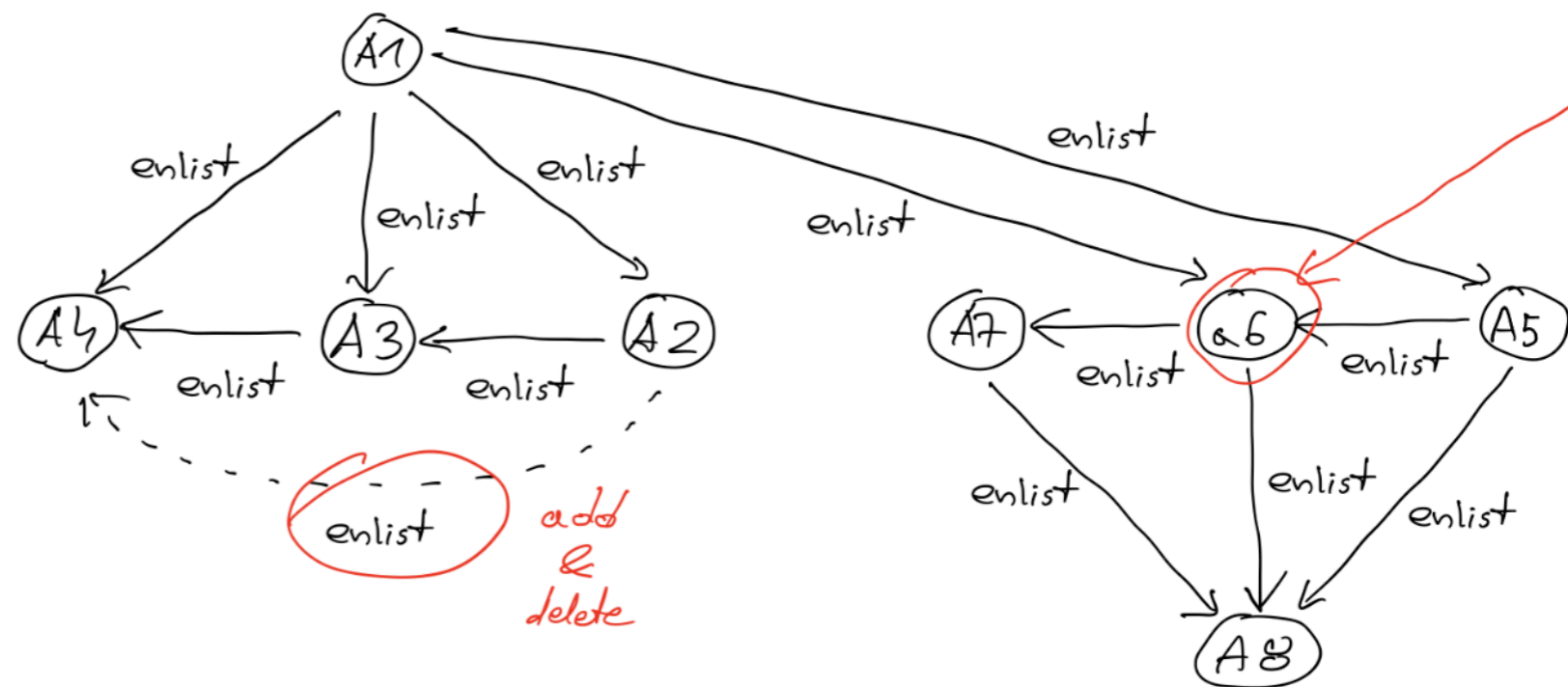
Finding the simple path

Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim', 'A2').until(has('pseudonim', 'A8')).repeat(both('enlist').simplePath()).path().unfold().elementMap()
```

```
==>[id:2,label:agent,pseudonim:A2]
==>[id:0,label:agent,pseudonim:A1]
==>[id:8,label:agent,pseudonim:A5]
==>[id:14,label:agent,pseudonim:A8]
==>[id:2,label:agent,pseudonim:A2]
==>[id:0,label:agent,pseudonim:A1]
==>[id:10,label:agent,pseudonim:A6]
==>[id:14,label:agent,pseudonim:A8]
==>[id:2,label:agent,pseudonim:A2]
==>[id:4,label:agent,pseudonim:A3]
==>[id:0,label:agent,pseudonim:A1]
==>[id:8,label:agent,pseudonim:A5]
==>[id:14,label:agent,pseudonim:A8]
```

...
gremlin>



Finding the simple path

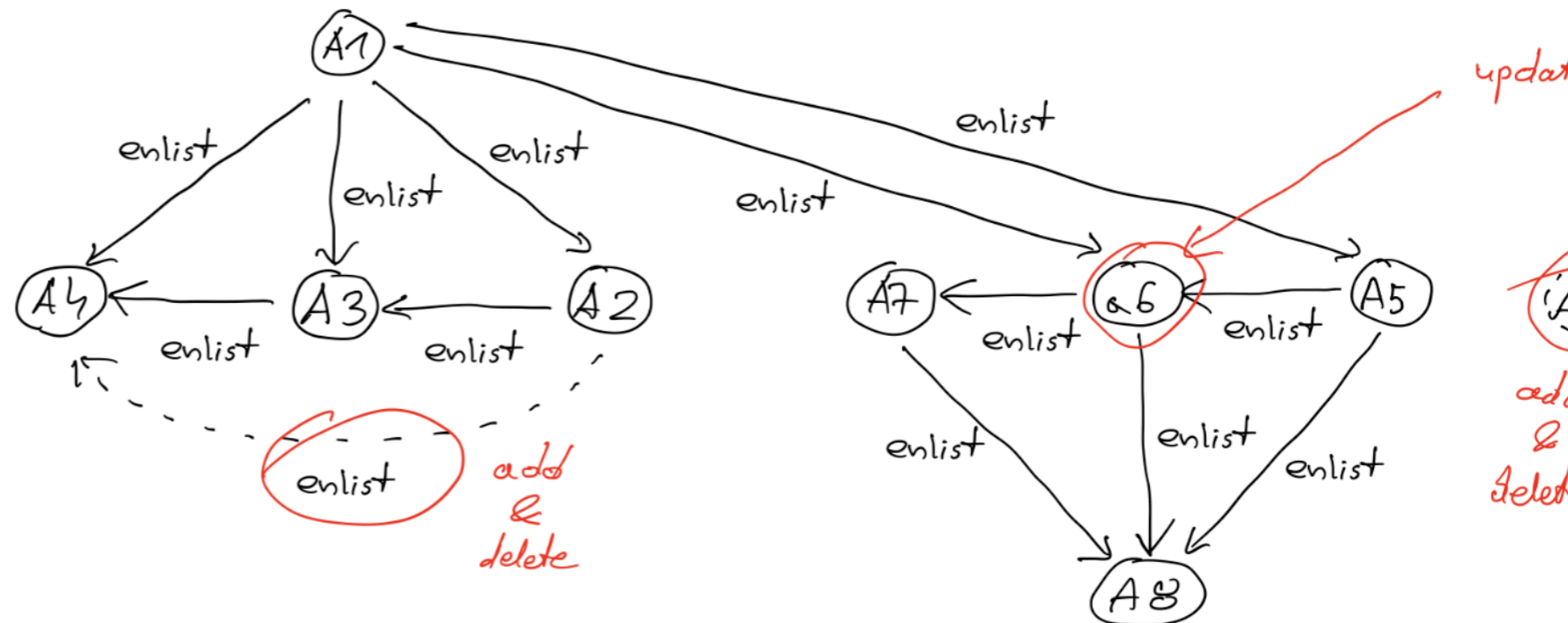
Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim','A2').until(has('pseudonim','A8')).  
repeat(both('enlist').simplePath()).path().unfold().  
values('pseudonim')
```

```
==>A2  
==>A1  
==>A5  
==>A8  
==>A2  
==>A1  
==>A6  
==>A8  
==>A2  
==>A3  
==>A1  
==>A5  
==>A8
```

...

```
gremlin>
```



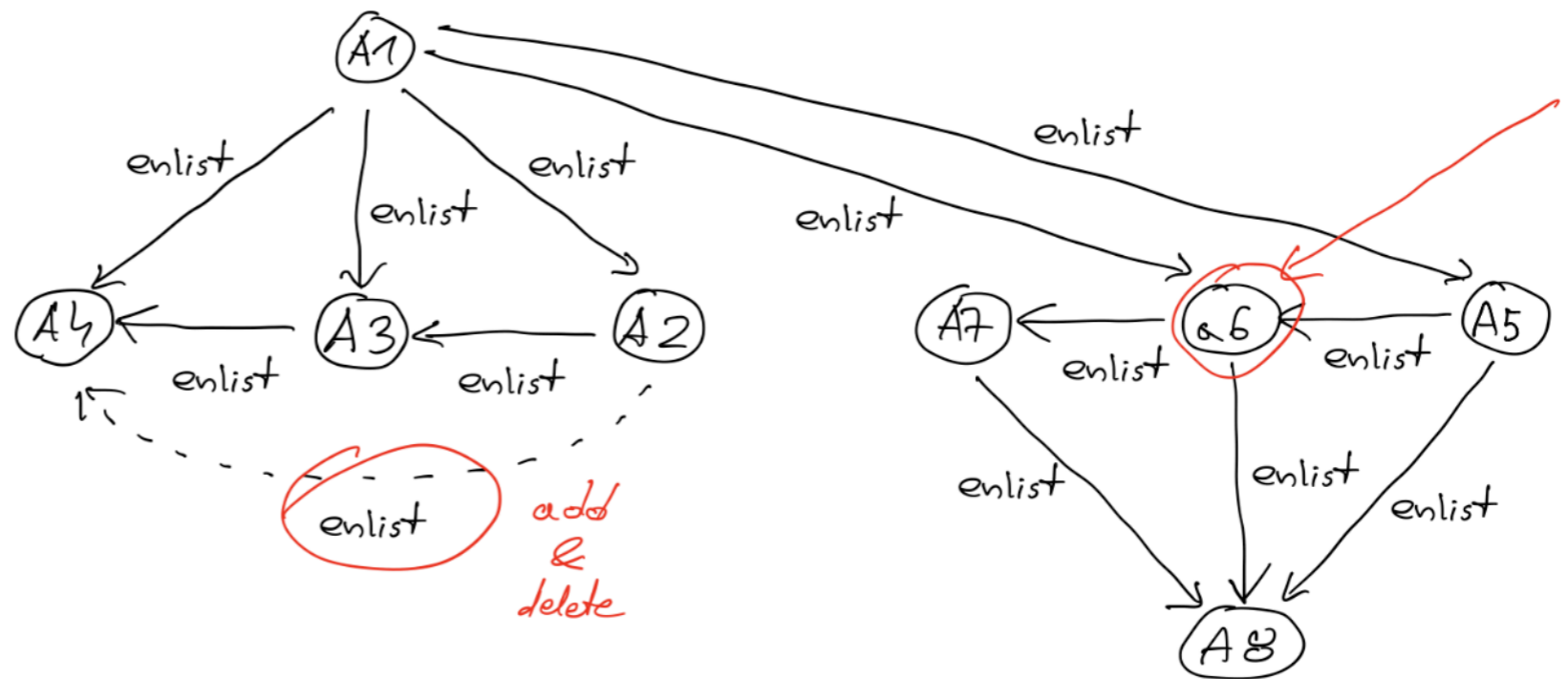
Finding the simple path

Finding the paths that connect two vertices

```
gremlin> g.V().has('pseudonim', 'A2').  
until(has('pseudonim', 'A8')).repeat(both('enlist')).  
simplePath().path().unfold().values('pseudonim').  
fold()
```

```
==>[A2, A1, A5, A8,  
A2, A1, A6, A8,  
A2, A3, A1, A5, A8,  
A2, A3, A1, A6, A8,  
A2, A1, A5, A6, A8,  
A2, A1, A6, A7, A8,  
A2, A3, A4, A1, A6, A8,  
A2, A3, A1, A5, A6, A8,  
A2, A3, A1, A6, A7, A8,  
A2, A3, A1, A6, A5, A8,  
A2, A1, A5, A6, A7, A8,  
A2, A3, A4, A1, A5, A6, A8,  
A2, A3, A4, A1, A6, A7, A8,  
A2, A3, A4, A1, A6, A5, A8,  
A2, A3, A1, A5, A6, A7, A8,  
A2, A3, A4, A1, A5, A6, A7, A8,  
A2, A1, A6, A5, A8,  
A2, A3, A4, A1, A5, A8]
```

```
gremlin>
```



Finding the simple path

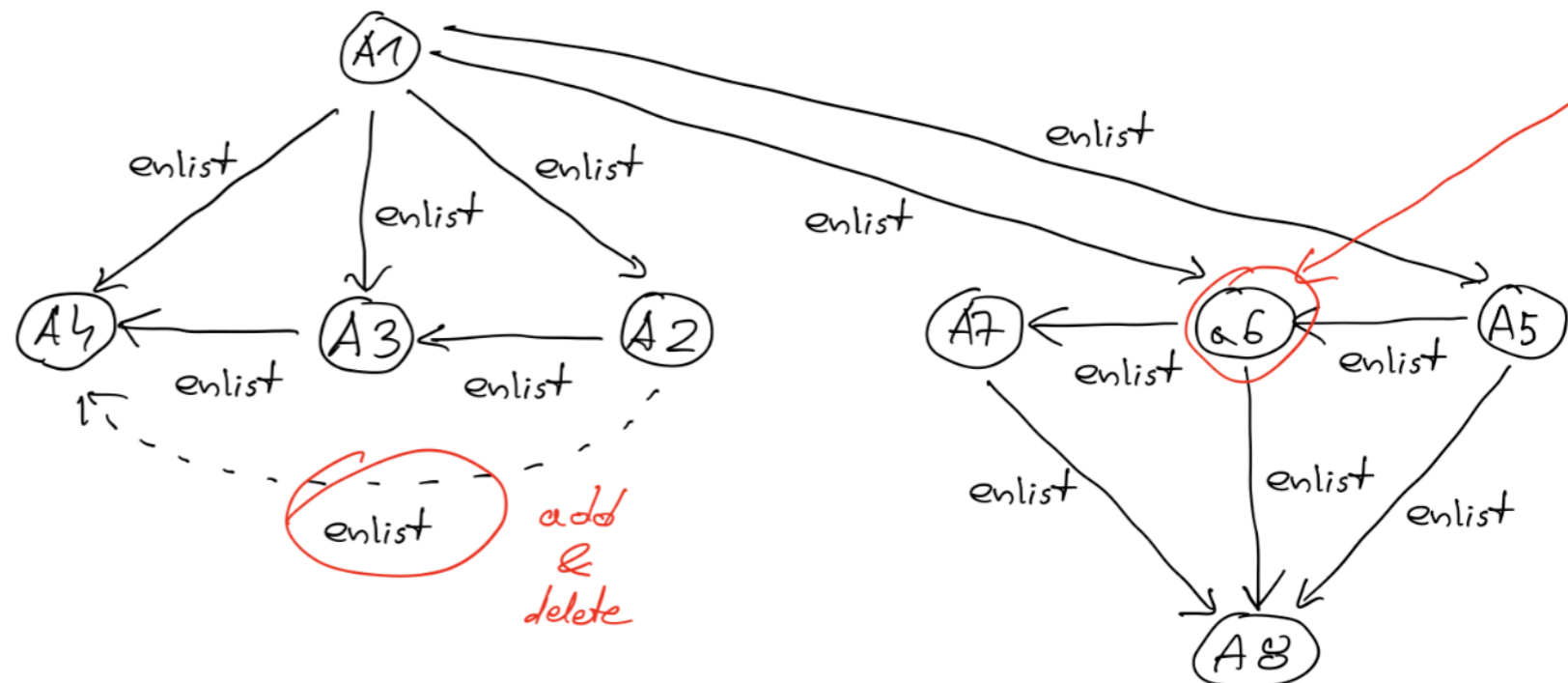
Finding the paths that connect two vertices: **task**

`gremlin> ???`

`==> [A2, A1, A5, A8]`

`==> [A2, A1, A6, A8]`

`==> [A2, A3, A1, A5, A8]`



Finding the simple path

Finding the paths that connect two vertices

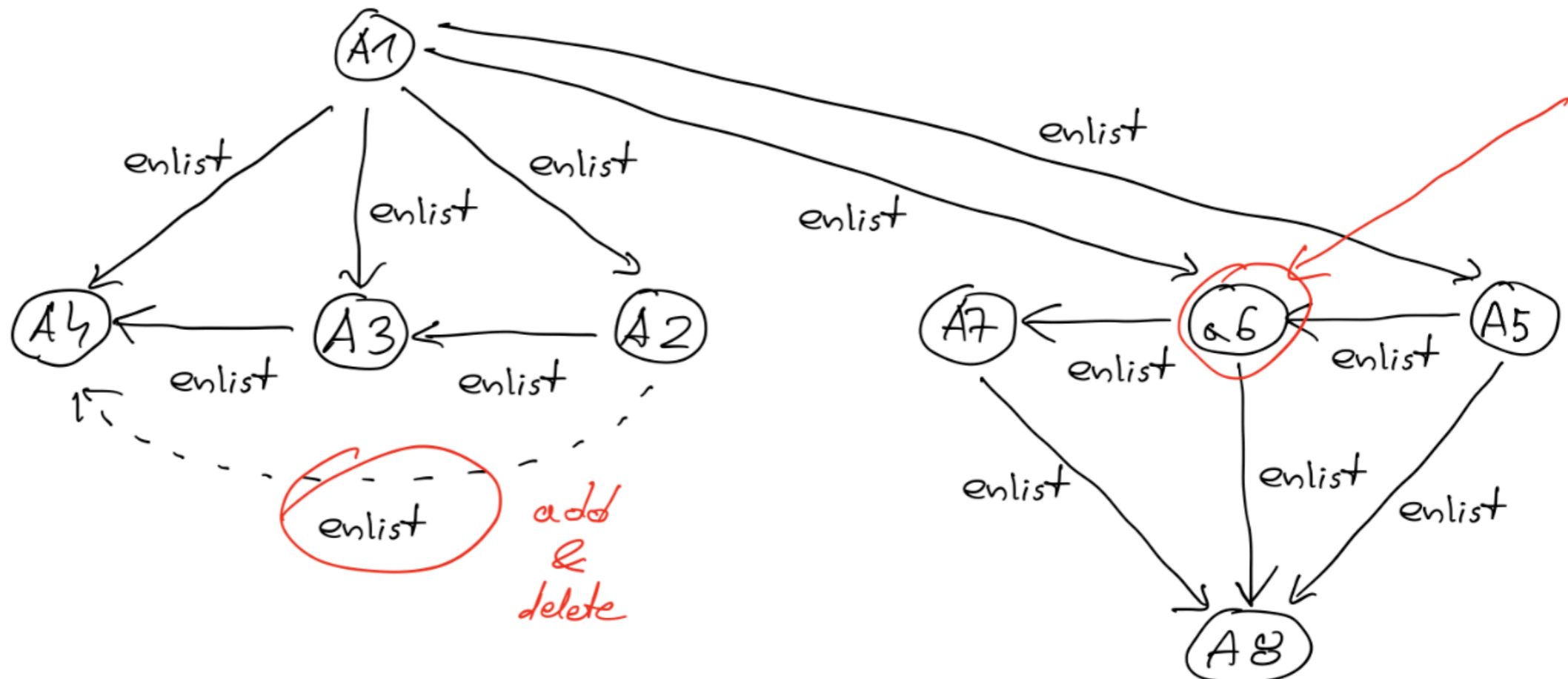
To be able to print full path info, with vertices and edges, we have to "touch" both edges and vertices during our traversal.

Recall the example of `out ()` and `outE ()` from the previous part of the lecture.

```

gremlin> g.V().has('pseudonim', 'A2').
until(has('pseudonim', 'A8')).
repeat(bothE('enlist').otherV().simplePath()).
path()
==>[v[2],e[16][0-enlist->2],v[0],e[19][0-enlist->8],v[8],e[24][8-enlist->14],v[14]]
==>[v[2],e[16][0-enlist->2],v[0],e[20][0-enlist->10],v[10],e[26][10-enlist->14],v[14]]
==>[v[2],e[21][2-enlist->4],v[4],e[17][0-enlist->4],v[0],e[19][0-enlist->8],v[8],e[24][8-enlist->14],v[14]]

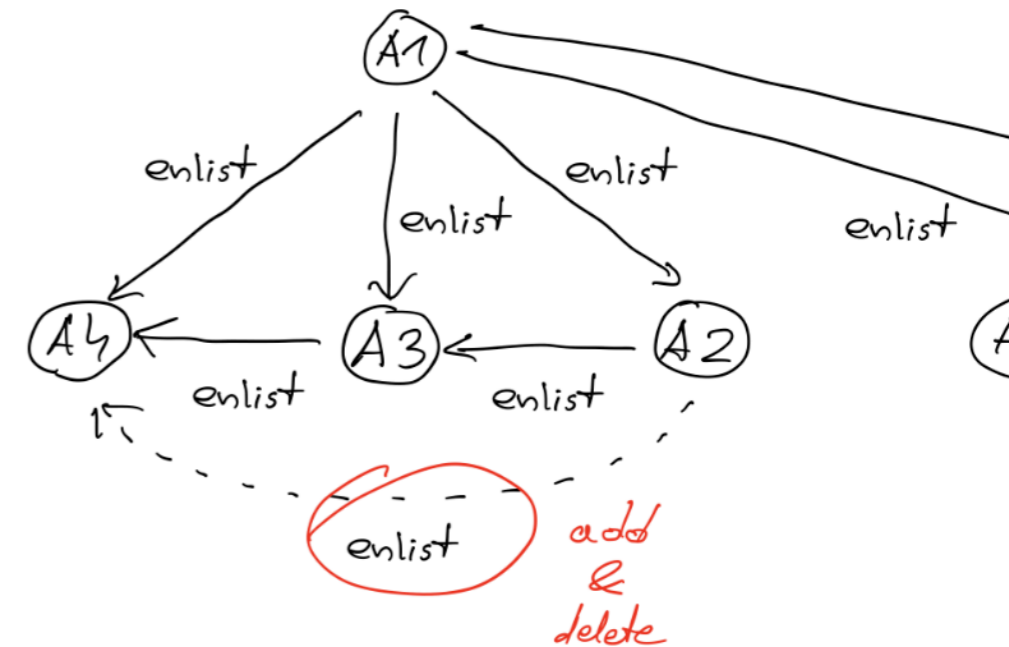
```



Working with subgraphs

Simple example

```
gremlin> subgraph = g.V().has('pseudonim','A3').bothE().subgraph('sg').
cap('sg').next()
==>tinkergraph[vertices:4 edges:3]
gremlin> sg=subgraph.traversal()
==>graphtraversalsource[tinkergraph[vertices:4 edges:3], standard]
gremlin> sg.V()
==>v[0]
==>v[2]
==>v[4]
==>v[6]
gremlin> sg.E()
==>e[17][0-enlist->4]
==>e[21][2-enlist->4]
==>e[22][4-enlist->6]
gremlin> sg.V().elementMap()
==>[id:0,label:agent,pseudonim:A1]
==>[id:2,label:agent,pseudonim:A2]
==>[id:4,label:agent,pseudonim:A3]
==>[id:6,label:agent,pseudonim:A4]
gremlin> sg.E().elementMap()
==>[id:17,label:enlist,IN:[id:4,label:agent],OUT:[id:0,label:agent]]
==>[id:21,label:enlist,IN:[id:4,label:agent],OUT:[id:2,label:agent]]
==>[id:22,label:enlist,IN:[id:6,label:agent],OUT:[id:4,label:agent]]
```



Bibliography

- [Bec] Dave Bechberger, Josh Perryman, *Graph Databases in Action*, Manning Publications, 2020