Gremlin Cheat Sheet 101

Read-Only Traversals

Initial Lookups

Steps	Meaning
`V()`	get all vertices in the graph
`E()`	get all edges in the graph
`V().hasLabel(label1, label2, ···)`	get all vertices with the specified labels
`V().has(label, key, value)`	get all vertices with the specified label and the property `key` matching the provided `value`
`V(1)`	get the vertex with the id `1`

Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V()
==>v[1]
==>v[2]
==>v[3]
==>v[4]
==>v[5]
==>v[6]
gremlin> g.V().hasLabel("person")
==>v[1]
==>v[2]
==>v[4]
==>v[6]
gremlin> g.V().has("person","name","marko")
==>v[1]
gremlin> g.V(1)
==>v[1]
```

Access Properties

Steps	Meaning
`properties(key1, key2, ···)`	get all specified properties for the current element
`values(key1, key2, ···)`	get all specified property values for the current element
`valueMap(key1, key2, ···)`	get all specified property values for the current element as a map

Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().hasLabel("person").properties("name")
==>vp[name->marko]
==>vp[name->vadas]
==>vp[name->josh]
==>vp[name->peter]
gremlin> g.V().hasLabel("person").values("name")
==>marko
==>vadas
==>josh
==>peter
gremlin> g.V().hasLabel("person").valueMap("name","age")
==>[name:[marko],age:[29]]
==>[name:[vadas],age:[27]]
==>[name:[josh],age:[32]]
==>[name:[peter],age:[35]]
```

Traversing the Graph

Steps	Meaning
`out(label1, label2, ···)`	get all adjacent vertices connected by outgoing edges with the specified labels
`in(label1, label2, ···)`	get all adjacent vertices connected by incoming edges with the specified labels
`outE(label1, label2, ···)`	get all outgoing edges with the specified labels
`inE(label1, label2, ···)`	get all incoming edges with the specified labels
`both(label1, label2, ···)`	get all adjacent vertices connected by an edge with the specified labels
`bothE(label1, label2, ···).otherV()`	traverse to all incident edges with the specified labels and then to the respective other vertices

Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V(1).outE("created")
==>e[9][1-created->3]
gremlin> g.V(1).out("created")
==>v[3]
gremlin> g.V().has("software","name","lop").in("created").values("name")
==>marko
==>josh
==>peter
```

Filters

Steps	Meaning
`has(key, value)`	keep the current element if the specified property has the given value
`has(key, predicate)`	keep the current element if the specified property matches the given predicate
`filter(traversal)`	keep the current element if the provided traversal emits a result
`not(traversal)`	keep the current element if the provided traversal doesn't emit a result
`where(predicate)`	keep the current element if it matches the predicate referencing another element

NOTE Predicates are used to compare values based on equality, ranges or certain patterns. All TinkerPop predicates are implemented as static methods; a full list of TinkerPop predicates can be found in the JavaDocs for `P` and `TextP`.

Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().has("age",29).valueMap("name","age")
==>[name:[marko],age:[29]]
gremlin> g.V().has("age",gt(30)).valueMap("name","age")
==>[name:[josh],age:[32]]
==>[name:[peter],age:[35]]
gremlin> g.V().filter(outE())
==>v[1]
==>v[4]
==>v[6]
gremlin> g.V().not(outE())
==>v[2]
==>v[3]
==>v[5]
gremlin> g.V(1).as("other").
......1>   out("knows").where(gt("other")).by("age").
......2>   valueMap()
==>[name:[josh],age:[32]]
```

Aggregations

Steps	Meaning
`store(key)`	store the current element in the side-effect with the provided key
`aggregate(key)`	store all elements held by all current traversers in the side-effect with the provided key
`group[{key}].by(keySelector)`	group all current elements by the provided `keySelector`; group into a side-effect if a side-effect `key` was provided, otherwise emit the result immediately
`fold()`	fold all current elements into a single list
`unfold()`	unfold the incoming list and continue processing each element individually
`count()`	count the number of current elements
`min()/max()`	find the min/max value
`sum()`	compute the sum of all current values
`mean()`	compute the mean value of all current values

Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().hasLabel("person").store("x").select("x")
==>v[1]
==>v[1],v[2]
==>v[1],v[2],v[4]
==>v[1],v[2],v[4],v[6]
gremlin> g.V().hasLabel("person").aggregate("x").select("x")
==>v[1],v[2],v[4],v[6]
==>v[1],v[2],v[4],v[6]
==>v[1],v[2],v[4],v[6]
==>v[1],v[2],v[4],v[6]
gremlin> g.V().group().by(label)
==>[software:[v[3],v[5]],person:[v[1],v[2],v[4],v[6]]]
gremlin> g.V().fold()
==>v[1],v[2],v[3],v[4],v[5],v[6]]
gremlin> g.V().count()
==>6
gremlin> g.V().fold().count(local)
==>6
```

Branches

Steps	Meaning
`union(branch1, branch2, ···)`	execute all branches and emit their results
`choose(condition, true-branch, false-branch)`	`if/then/else`-based traversal. If the condition matches (yields something), execute the `true-branch`, otherwise follow the `false-branch`.

`choose(selector).`
`  option(opt1, traversal).`
`  option(opt2, traversal).`
`  option(optN, traversal)`
	value-based traversal; If an option value matches the value emitted by the `selector` traversal, the respective option traversal will be executed.

Examples

```
gremlin> g = TinkerFactory.createModern().traversal()
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
gremlin> g.V().hasLabel("person").union(out("knows"), count())
==>v[2]
==>v[4]
==>4
gremlin> g.V().hasLabel("person").
......1>   choose(has("age",gt(30)), constant("senior"), constant("junior"))
==>junior
==>junior
==>senior
==>senior
gremlin> g.V().hasLabel("person").values("age").
......1>   union(min(), max(), sum(), mean(), count())
==>27
==>35
==>123
==>30.75
==>4
```

Mutating Traversals

Steps	Meaning
`addV(label)`	add a new vertex
`addE(label).from(source).to(target)`	adds a new edge between the two given vertices
`property(key, value)`	adds or updates the property with the given `key`

Examples

```
gremlin> g = TinkerGraph.open().traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> g.addV('company').
......1>   property('name','datastax').as('ds').
......2>   addV('software').
......3>   property('name','dse graph').as('dse').
......4>   addV('software').
......5>   property('name','tinkerpop').as('tp').
......6>   addE('develops').from('ds').to('dse').
......7>   addE('uses').from('dse').to('tp').
......8>   addE('likes').from('ds').to('tp').iterate()
gremlin> g.V().outE().inV().path().by('name').by(label)
==>[datastax,develops,dse graph]
==>[datastax,likes,tinkerpop]
==>[dse graph,uses,tinkerpop]
```