Yandex

Shortest path

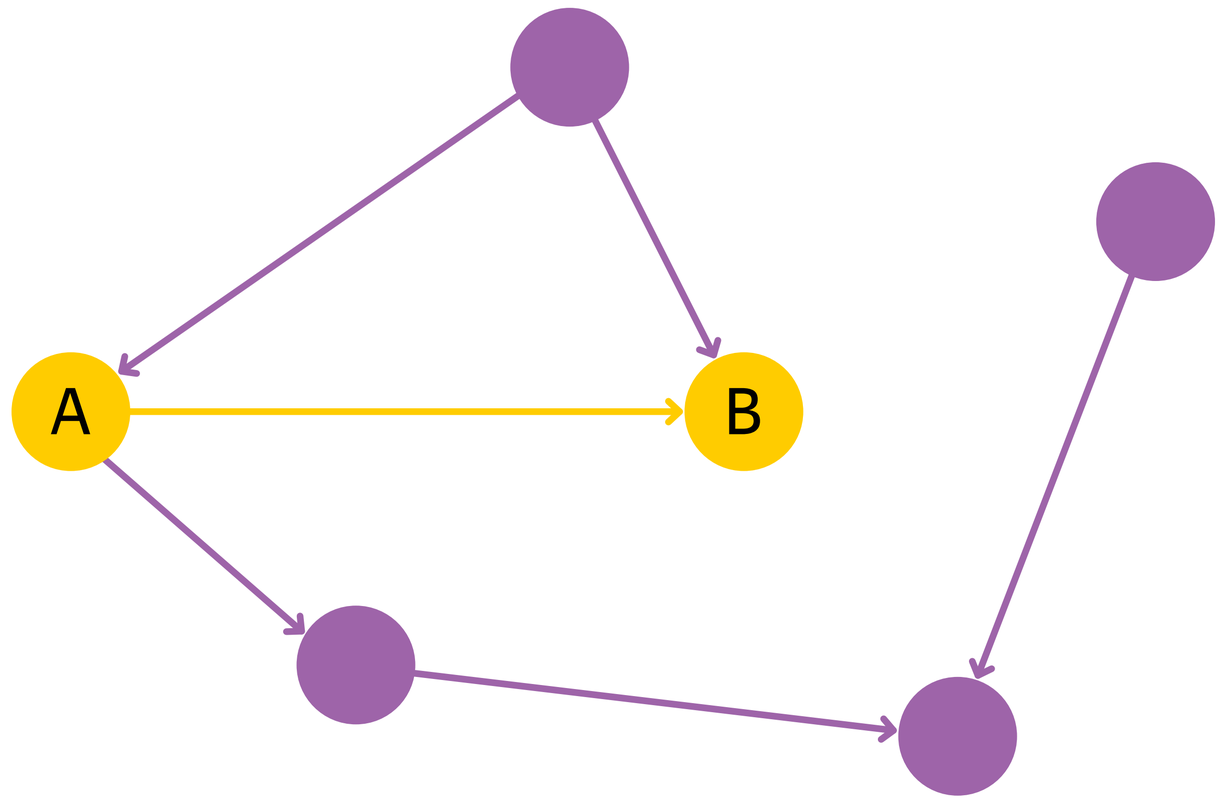
Social graph

› Vertexes — users
(41'652'230)

› Edges – "follows" relation
(1'468'365'182)

› 6G compressed,
25G uncompressed

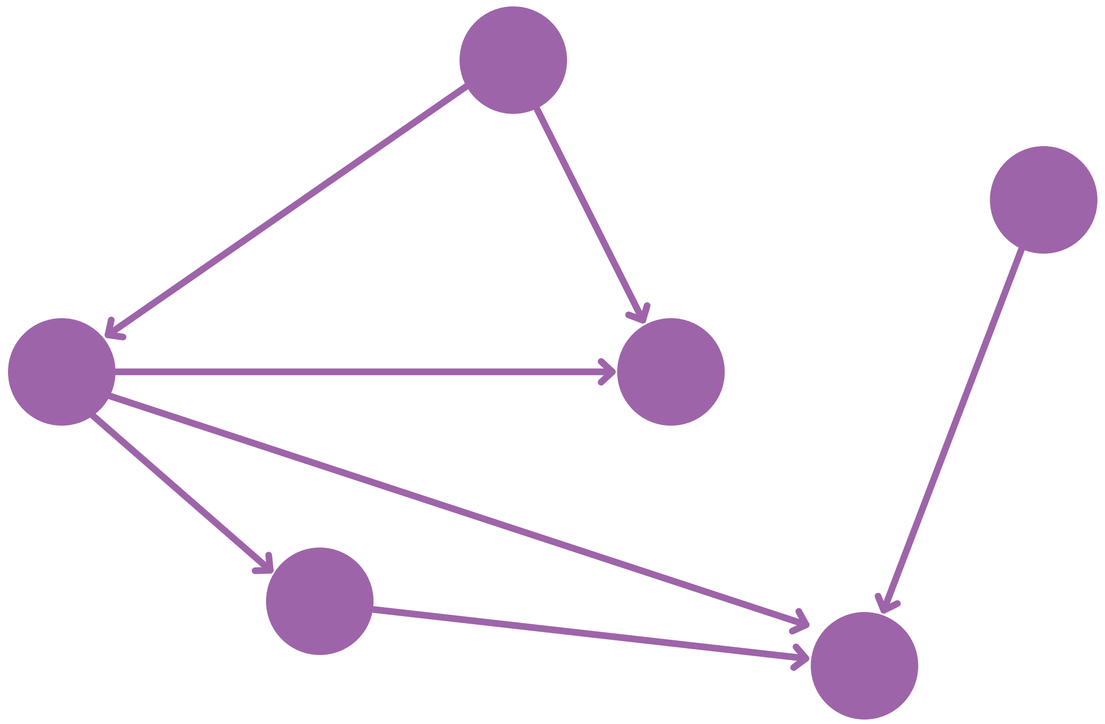
› Edge (A, B) =
= User A follows B =
= User B is followed by A



$ pyspark

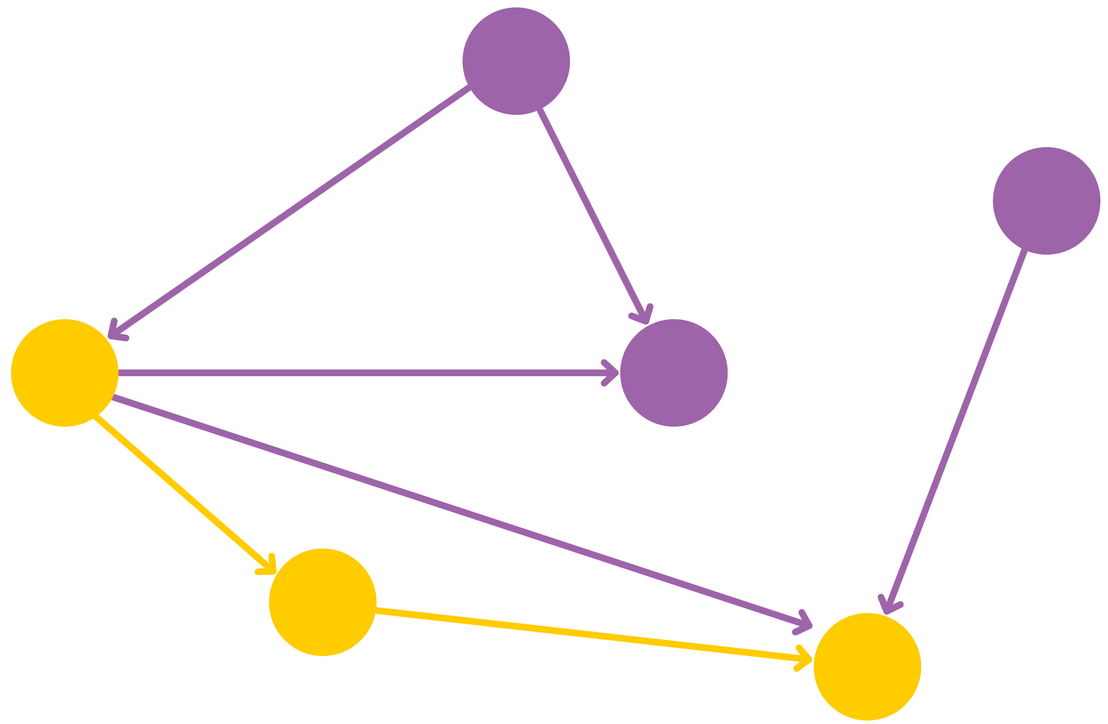
```
...
>>> def parse_edge(s):
...   user, follower = s.split("\t")
...   return (int(user), int(follower))
...
>>> edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
>>>
```

Definitions



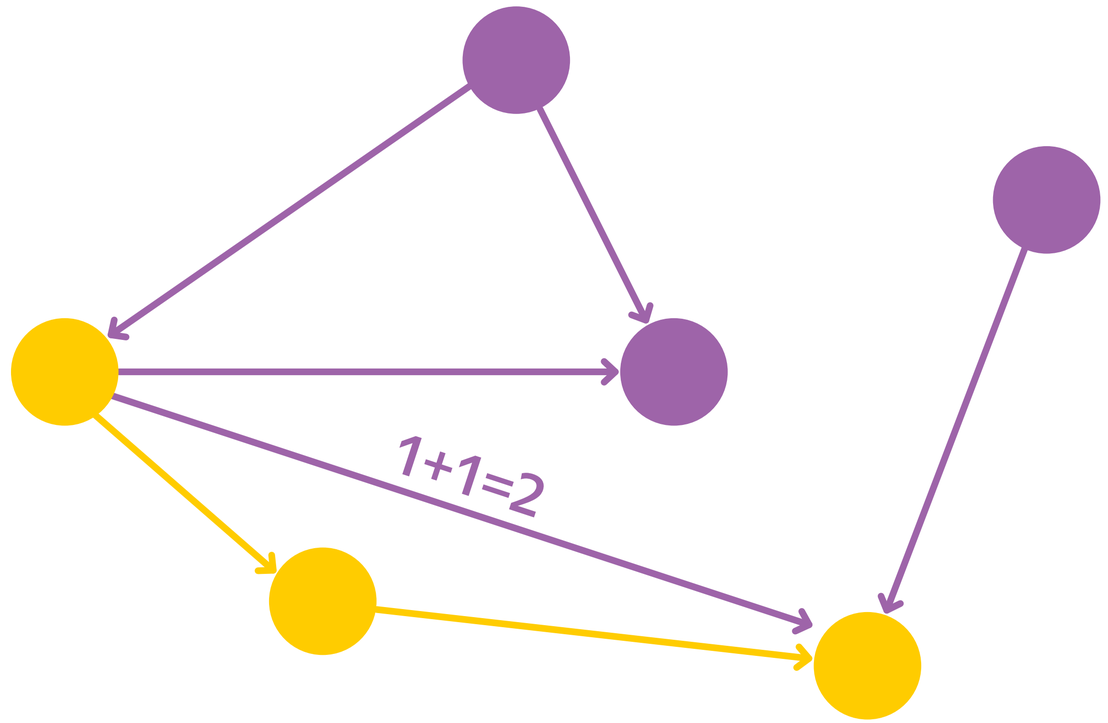
Definitions

- **(Vertex) Path** — a sequence of vertices where every two consequent vertices are connected by an edge



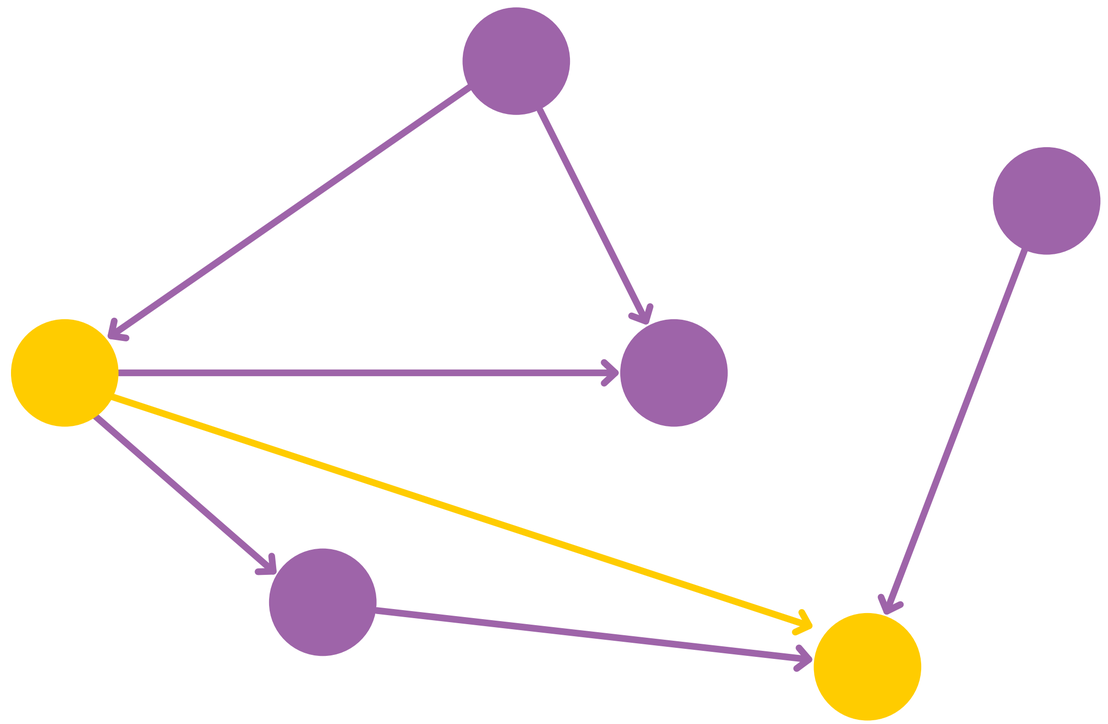
Definitions

- **(Vertex) Path** — a sequence of vertices where every two consequent vertices are connected by an edge
- **Path Length** — number of edges in the path



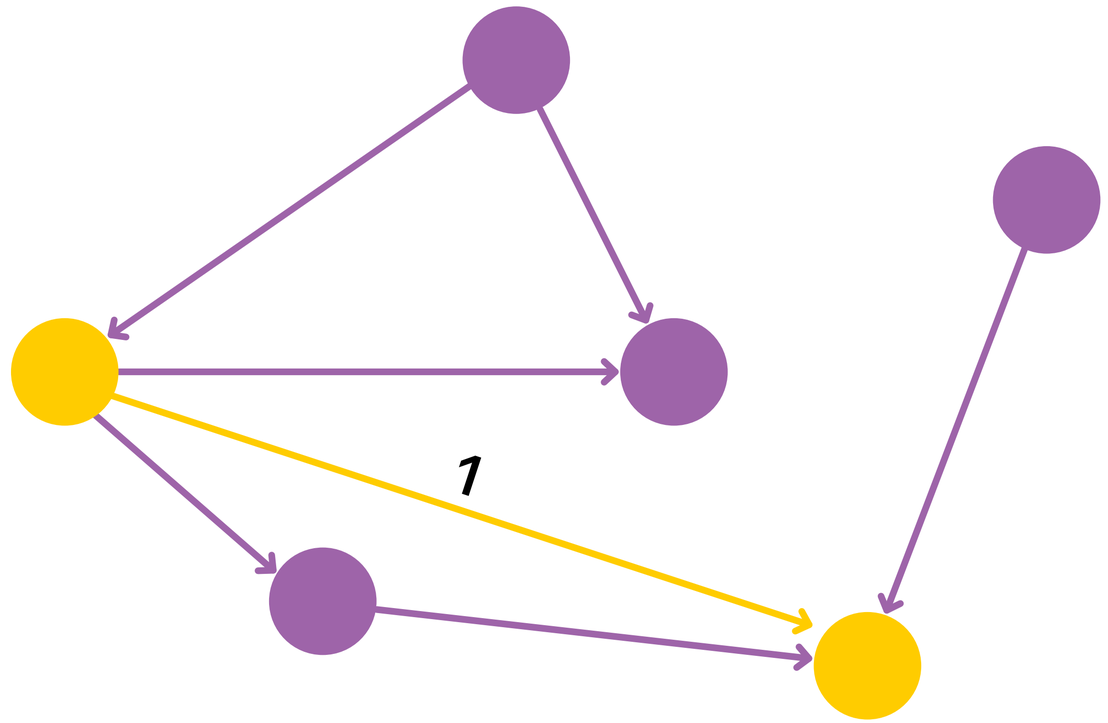
Definitions

- **(Vertex) Path** — a sequence of vertices where every two consequent vertices are connected by an edge
- **Path Length** — number of edges in the path
- **Shortest Path** (between two vertices) — one with a minimal length

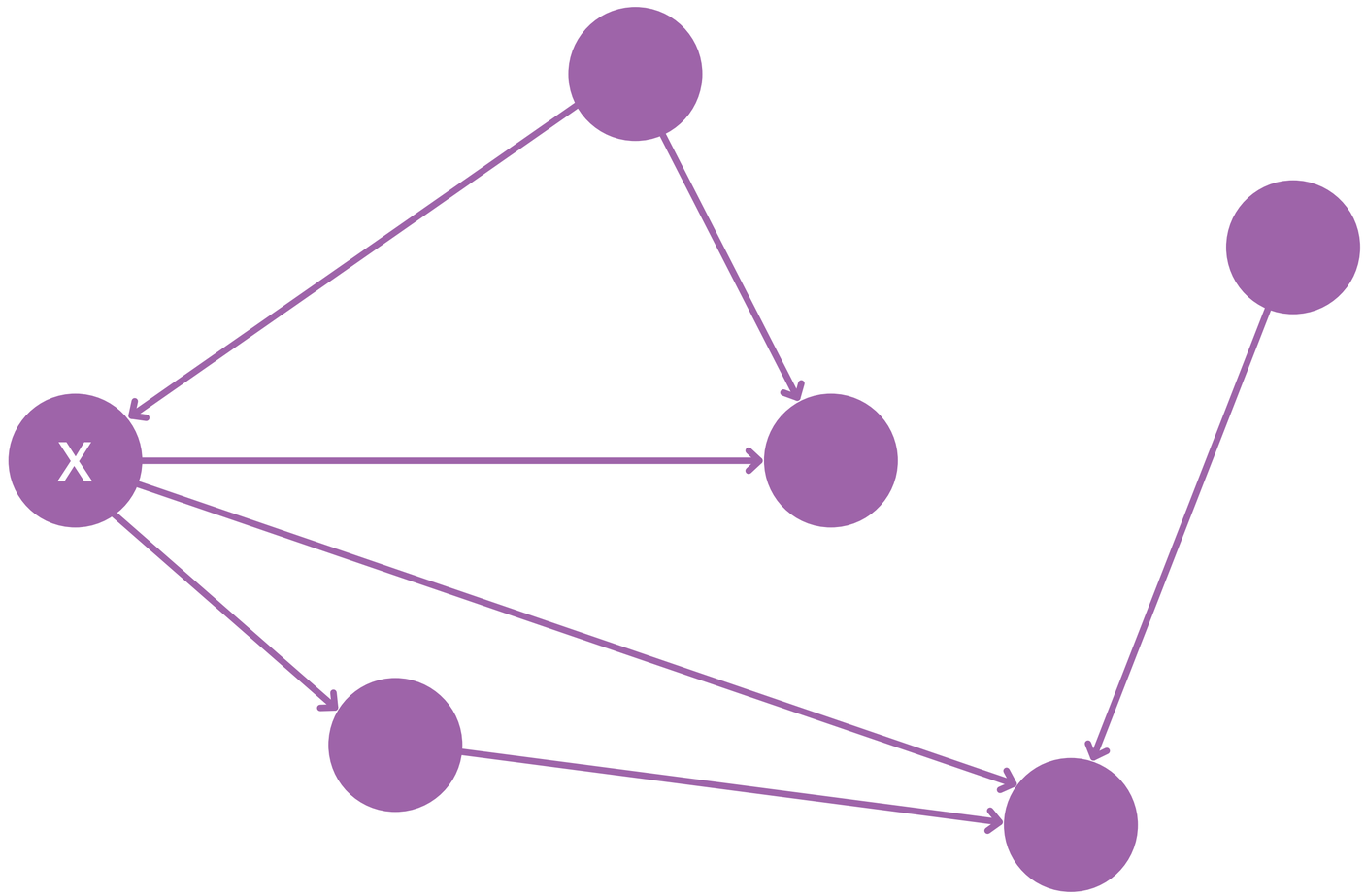


Definitions

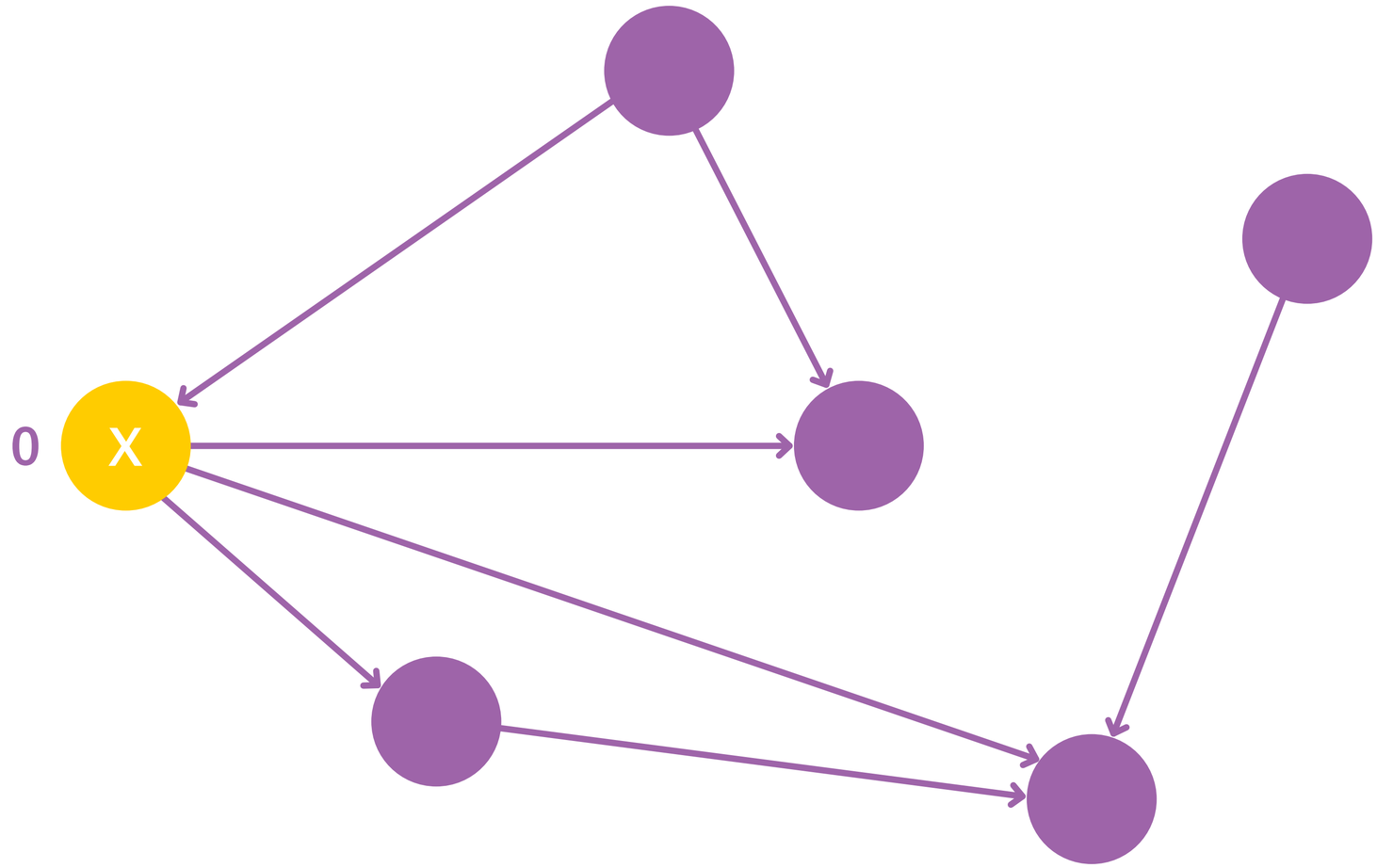
- **(Vertex) Path** — a sequence of vertices where every two consequent vertices are connected by an edge
- **Path Length** — number of edges in the path
- **Shortest Path** (between two vertices) — one with a minimal length
- **Distance** (between two vertices) — length of the shortest path



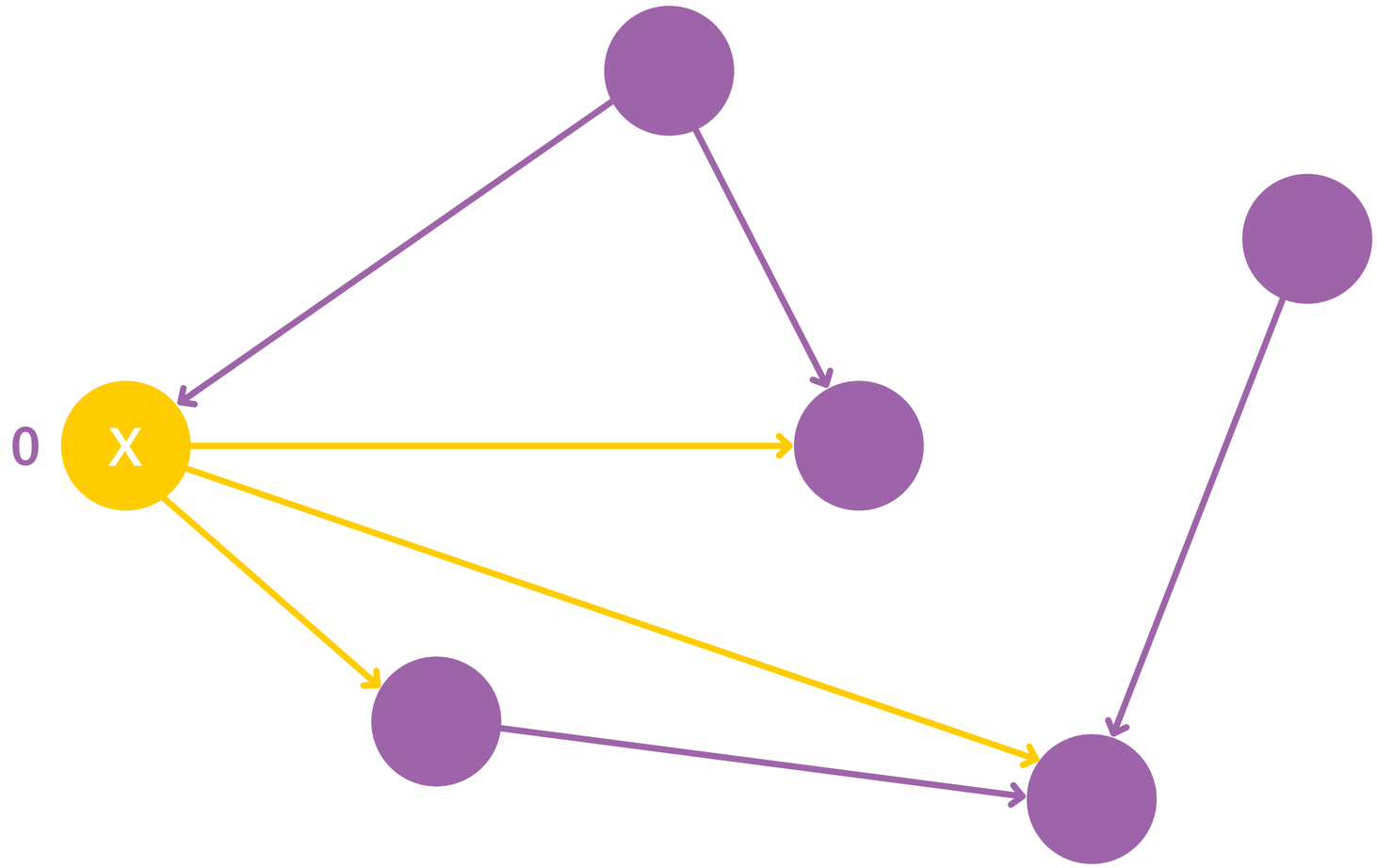
Computing distances



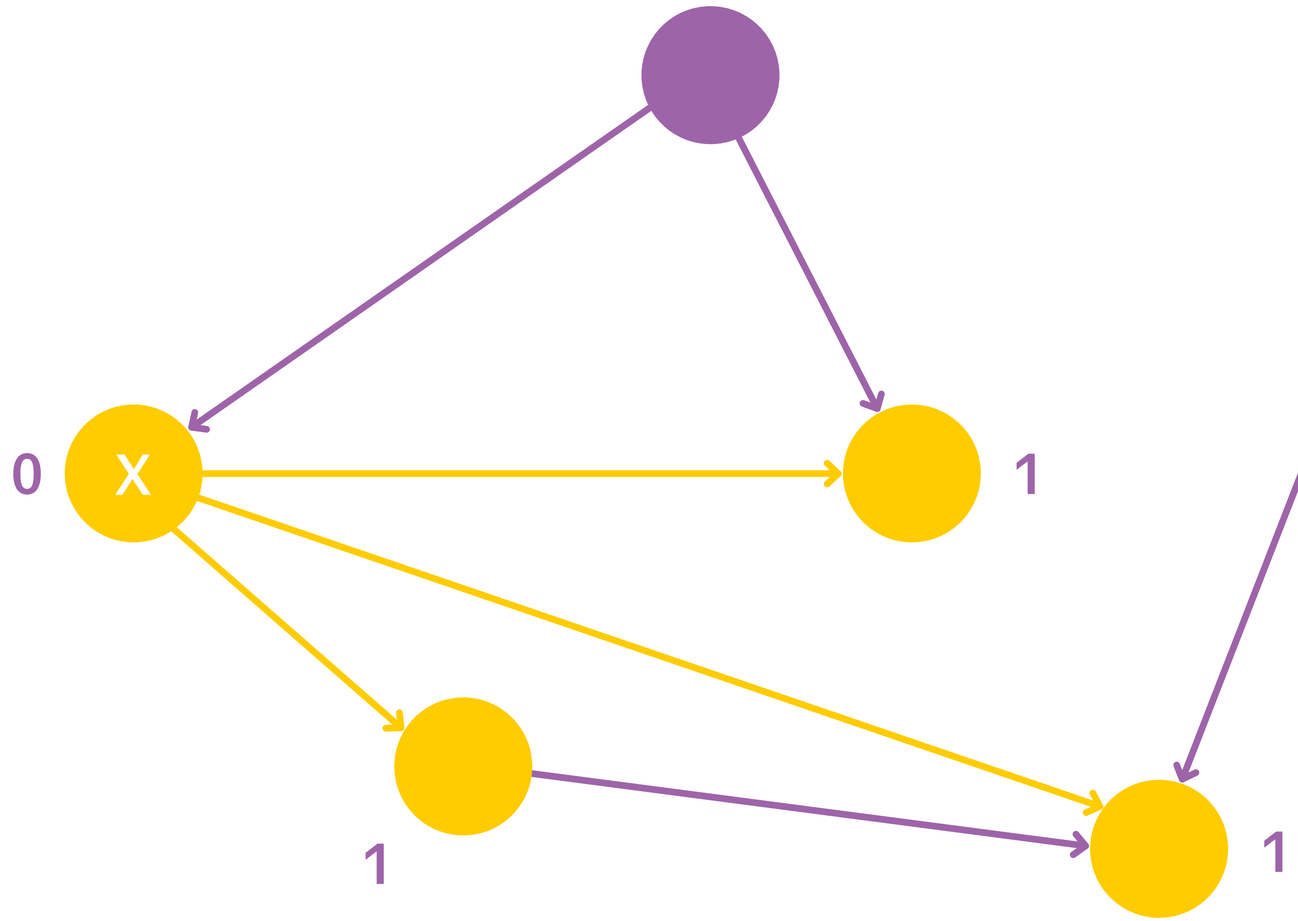
Computing distances



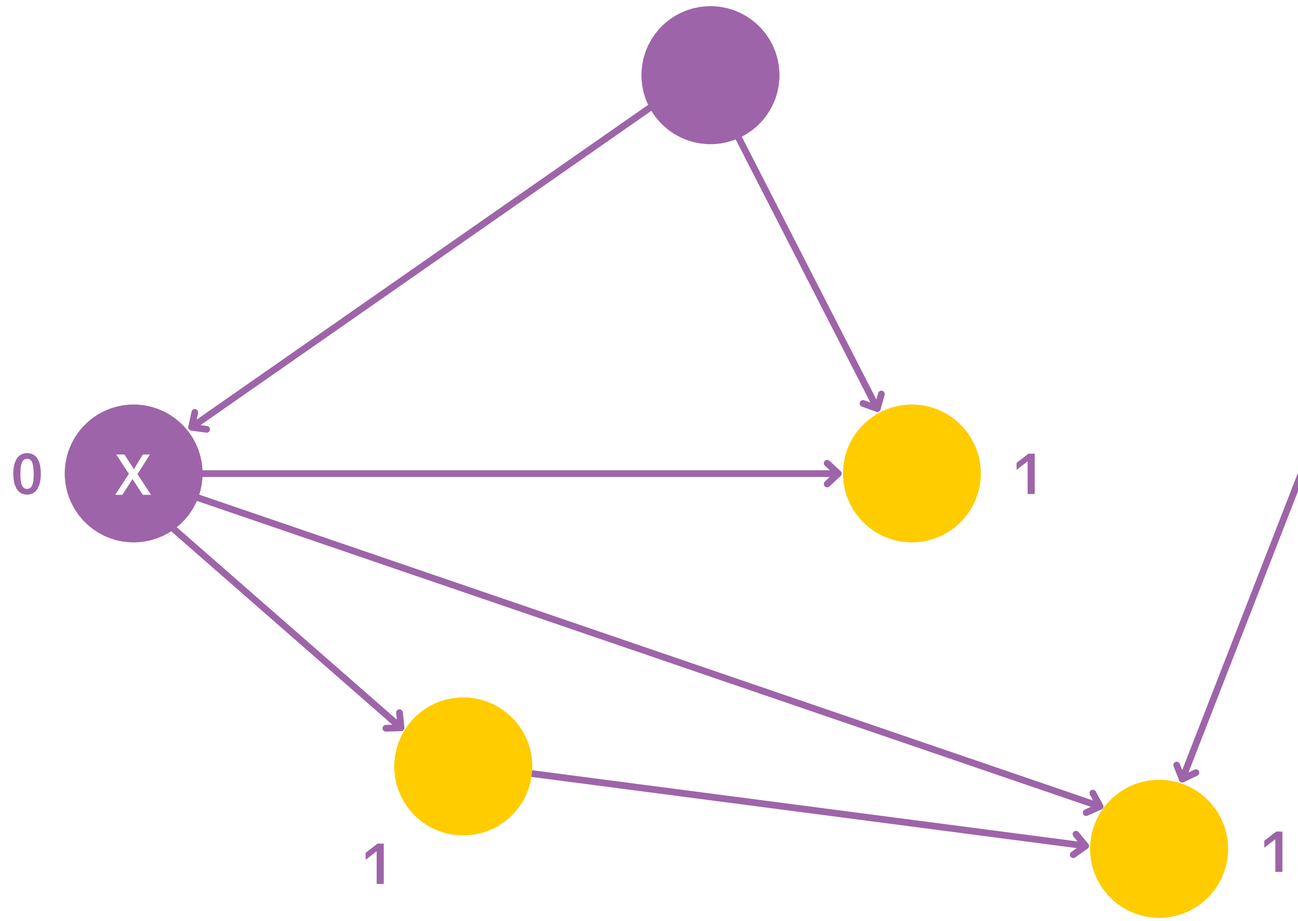
Computing distances



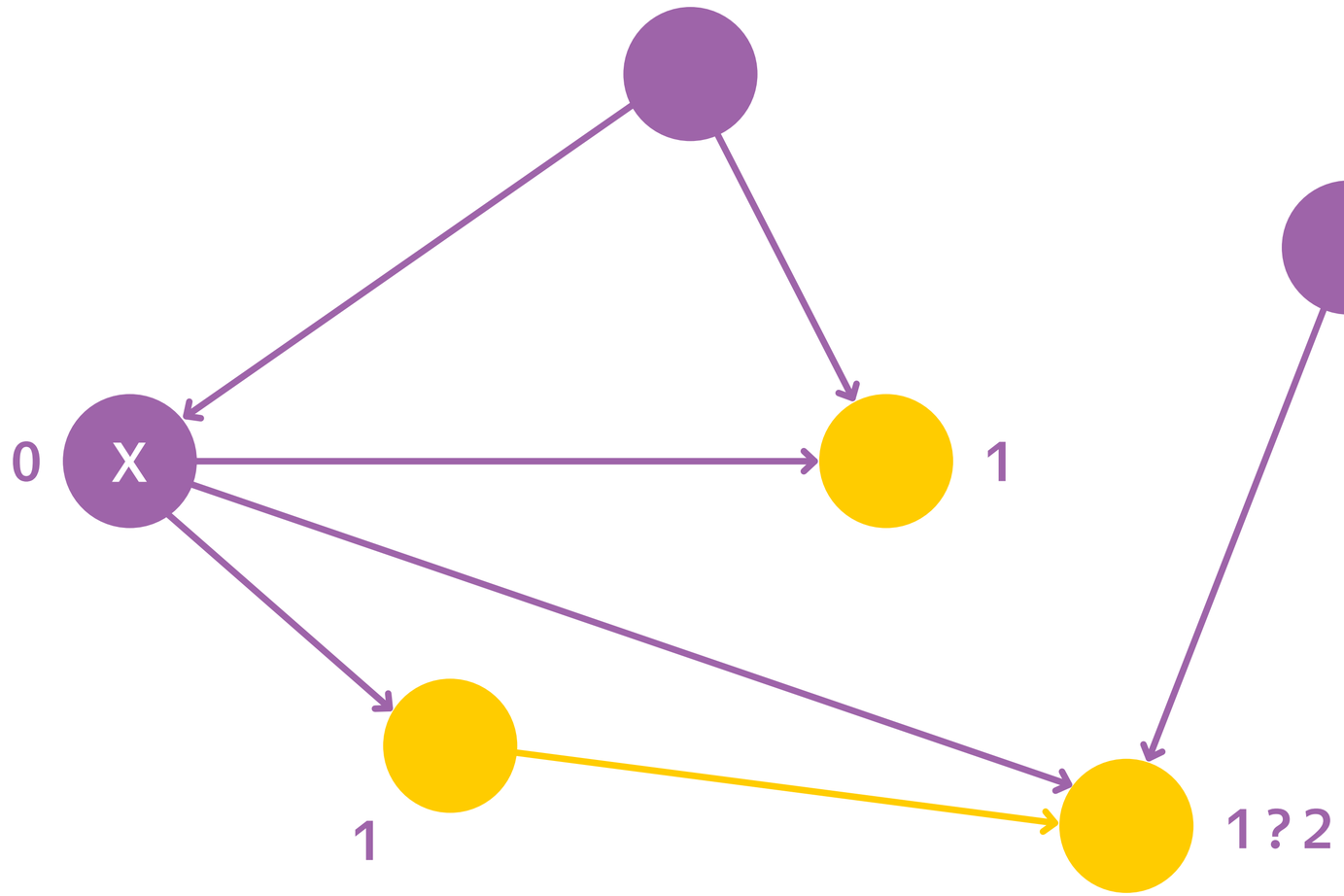
Computing distances



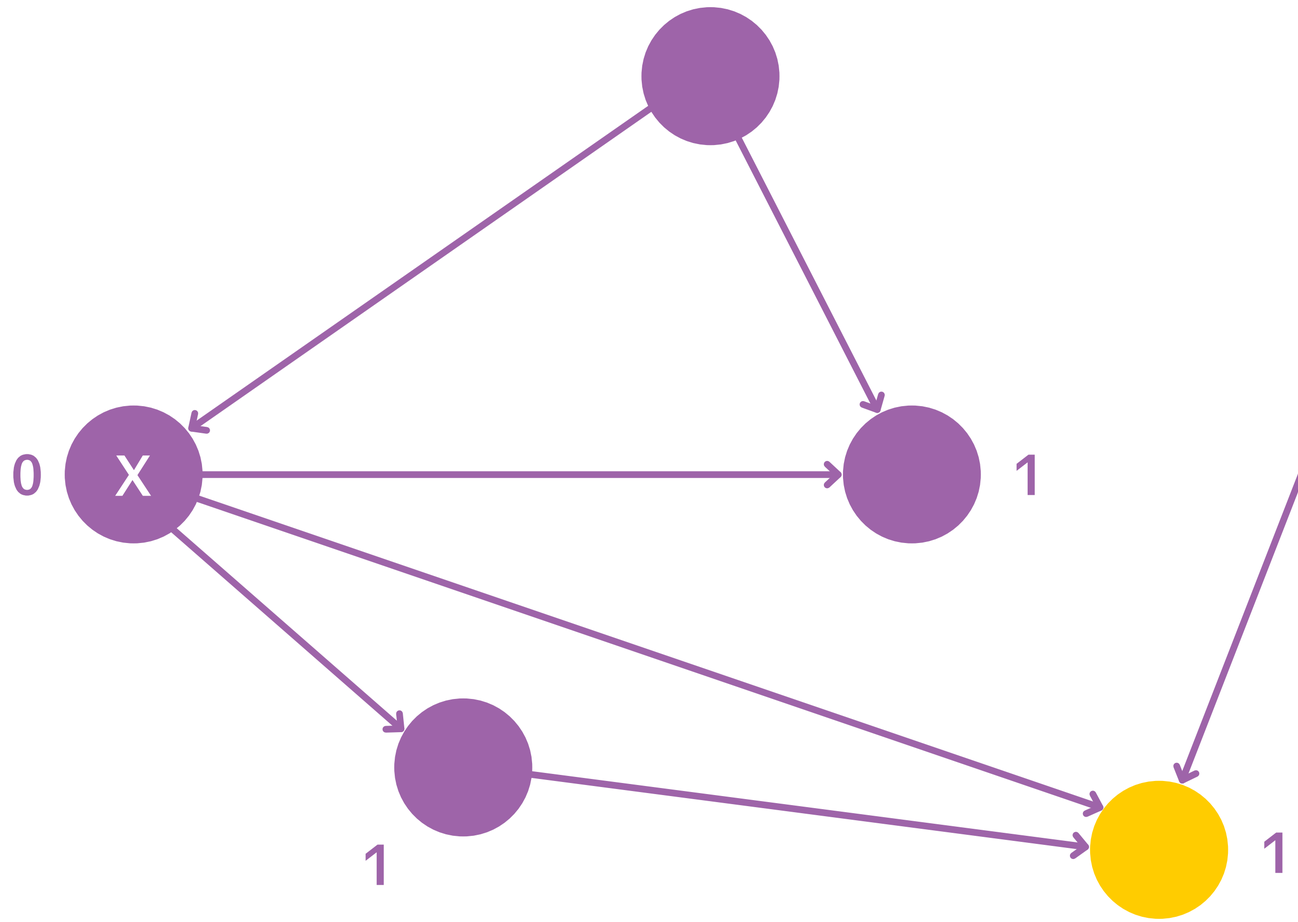
Computing distances



Computing distances



Computing distances



$ pyspark

```
...
>>> def parse_edge(s):
...   user, follower = s.split("\t")
...   return (int(user), int(follower))
...
>>> edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
>>>
```

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
```

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()
```

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()
x = 12
distances = sc.parallelize([(x, 0)])
```

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()
x = 12
distances = sc.parallelize([(x, 0)])
```

```
distances.join(forward_edges)
```

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()
x = 12
distances = sc.parallelize([(x, 0)])
```

```
distances.join(forward_edges)
```

L = (v 12, d 0)
R = (v 12, v 13)

J = (v 12, (d 0, v 13))

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()
x = 12
distances = sc.parallelize([(x, 0)])
def step(item):
  prev_v, prev_d, next_v = item[0], item[1][0], item[1][1]
  return (next_v, prev_d + 1)
distances.join(forward_edges).map(step)
```

L = (v 12, d 0)
R = (v 12, v 13)

J = (v 12, (d 0, v 13))
M = (v 13, d 1)

```
def parse_edge(s):
  user, follower = s.split("\t")
  return (int(user), int(follower))

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").
map(parse_edge).cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()
x = 12
distances = sc.parallelize([(x, 0)])
def step(item):
  prev_v, prev_d, next_v = item[0], item[1][0], item[1][1]
  return (next_v, prev_d + 1)
```

```
candidates = distances.join(forward_edges).map(step)
def complete(item):
  v, old_d, new_d = item[0], item[1][0], item[1][1]
  return (v, old_d if old_d is not None else new_d)

new_distances = distances.fullOuterJoin(candidates).map(complete)
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()

x = 12
distances = sc.parallelize([(x, 0)])
candidates = distances.join(forward_edges).map(step)
new_distances = distances.fullOuterJoin(candidates).map(complete)
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()

x = 12
d = 0
distances = sc.parallelize([(x, d)])
while True:
    candidates = distances.join(forward_edges).map(step)
    new_distances = distances.fullOuterJoin(candidates).map(complete)
    count = new_distances.filter(lambda i: i[1] == d + 1).count()
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()

x = 12
d = 0
distances = sc.parallelize([(x, d)])
while True:
  candidates = distances.join(forward_edges).map(step)
  new_distances = distances.fullOuterJoin(candidates).map(complete)
  count = new_distances.filter(lambda i: i[1] == d + 1).count()
  if count > 0:



  else:
    break
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()

x = 12
d = 0
distances = sc.parallelize([(x, d)])
while True:
  candidates = distances.join(forward_edges).map(step)
  new_distances = distances.fullOuterJoin(candidates).map(complete)
  count = new_distances.filter(lambda i: i[1] == d + 1).count()
  if count > 0:
    d += 1
    distances = new_distances
    print("d = ", d, "count = ", count)
  else:
    break
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()

x = 12
d = 0
distances = sc.parallelize([(x, d)])
while True:
  candidates = distances.join(forward_edges).map(step)
  new_distances = distances.fullOuterJoin(candidates).map(complete)
  count = new_distances.filter(lambda i: i[1] == d + 1).count()
  if count > 0:
    d += 1
    distances = new_distances
    print("d = ", d, "count = ", count)
  else:
    break
```

Pause the video; why does the iteration time increases?

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)

edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).cache()

x = 12
d = 0
distances = sc.parallelize([(x, d)])
while True:
  candidates = distances.join(forward_edges).map(step)
  new_distances = distances.fullOuterJoin(candidates)
.map(complete).persist()
  count = new_distances.filter(lambda i: i[1] == d + 1).count()
  if count > 0:
    d += 1
    distances = new_distances
    print("d = ", d, "count = ", count)
  else:
    break
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)
n = 400  # number of partitions
edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).partitionBy(n).persist()

x = 12
d = 0
distances = sc.parallelize([(x, d)]).partitionBy(n)
while True:
  candidates = distances.join(forward_edges, n).map(step)
  new_distances = distances.fullOuterJoin(candidates, n).map(complete,
True).persist()
  count = new_distances.filter(lambda i: i[1] == d + 1).count()
  if count > 0:
    d += 1
    distances = new_distances
    print("d = ", d, "count = ", count)
  else:
    break
```

```
def parse_edge(s): u, f = s.split("\t"); return (int(u), int(f))
def step(i): pv, pd, nv = i[0], i[1][0], i[1][1]; return (nv, pd + 1)
def complete(i): v, od, nd = i[0], i[1][0], i[1][1]; return (v, od if od is
not None else nd)
n = 400  # number of partitions
edges = sc.textFile("hdfs:///data/twitter/twitter_rv.net").map(parse_edge).
cache()
forward_edges = edges.map(lambda e: (e[1], e[0])).partitionBy(n).persist()

x = 12
d = 0
distances = sc.paralle
while True:
  candidates = distanc                                          ep)
  new_distances = dist                                          ).map(complete,
True).persist()
  count = new_distances.filter(lambda i: i[1] == d + 1).count()
  if count > 0:
    d += 1
    distances = new_distances
    print("d = ", d, "count = ", count)
  else:
    break
```

Exercise: how to reconstruct the shortest path?

Summary

- You have learned how to:
  - write iterative algorithms in Spark
  - tune persistence and partitioning
  - implement a simple BFS graph algorithm

BigDATAteam