Twitter Scenarios

Scenario T1

Returns all names occurring in the input.

Tree Pattern

```
def t1Twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val name = twig.createNode("name", on = true)
  twig = twig.createEdge(root, name, false)
  twig.validate().get
}
```

TPM Pipeline

```
def q1(iteration: Int): Unit = {
  val tweets = loadTweets()
  val res = tweets.matchTree(t1Twig())
  write(res)
}
```

Spark Pipeline

```
def q1(iteration: Int): Unit = {
  val tweets = loadTweets()
  val n1 = tweets.withColumn("muser",
explode($"entities.user_mentions")).select($"muser.name".alias("name"))
  val n2 = tweets.withColumn("muser",
explode($"extended_tweet.entities.user_mentions")).select($"muser.name".alias("name"))
  val n3 = tweets.select($"place.name".alias("name"))
  val n4 = tweets.withColumn("muser",
explode($"quoted_status.extended_tweet.entities.user_mentions")).select($"muser.name".alias("
name"))
  val n5 = tweets.select($"quoted_status.place.name".alias("name"))
  val n6 = tweets.select($"quoted_status.user.name".alias("name"))
  val n7 = tweets.withColumn("muser",
explode($"retweeted_status.extended_tweet.entities.user_mentions")).select($"muser.name".alia
s("name"))
  val n8 = tweets.withColumn("muser",
explode($"retweeted_status.entities.user_mentions")).select($"muser.name".alias("name"))
  val n9 = tweets.select($"retweeted_status.place.name".alias("name"))
  val n10 = tweets.select($"retweeted_status.user.name".alias("name"))
  val n11 = tweets.withColumn("muser",
explode($"retweeted_status.quoted_status.extended_tweet.entities.user_mentions")).select($"mu
ser.name".alias("name"))
  val n12 = tweets.select($"retweeted_status.quoted_status.place.name".alias("name"))
  val n13 = tweets.select($"retweeted_status.quoted_status.user.name".alias("name"))
  val n14 = tweets.select($"user.name".alias("name"))
  val res =
n1.union(n2).union(n3).union(n4).union(n5).union(n6).union(n7).union(n8).union(n9).union(n10)
.union(n11).union(n12).union(n13).union(n14)
  write(res)
}
```

Scenario T2

This scenario is the running example

Tree Pattern

```
def t2Twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val name = twig.createNode("screen_name", on = true)
  val entities = twig.createNode("entities")
  val user_mentions = twig.createNode("user_mentions", min=1, max=1000000)
  val id_str = twig.createNode("id_str")
  val retweet_count = twig.createNode("retweet_count", condition = "0")
  twig = twig.createEdge(root, entities, true)
  twig = twig.createEdge(entities, user_mentions, true)
  twig = twig.createEdge(user_mentions, id_str, false)
  twig = twig.createEdge(root, retweet_count, true)
  twig = twig.createEdge(root, name, false)
  twig.validate().get
}
```

TPM Pipeline

```
def q2(iteration: Int): Unit = {
  val tweets = loadTweets()
  var res = tweets.matchTree(t2Twig())
  res = res.distinct()
  write(res)
}
```

Spark Pipeline

```
def q2(iteration: Int): Unit = {
  var tweets = loadTweets()
  tweets = tweets.filter($"retweet_count" === 0)
  tweets = tweets.withColumn("umCount", size($"entities.user_mentions")).filter($"umcount" >
0)
  val n1 = tweets.withColumn("muser",
explode($"entities.user_mentions")).select($"muser.screen_name".alias("name"))
  val n2 = tweets.withColumn("muser",
explode($"extended_tweet.entities.user_mentions")).select($"muser.screen_name".alias("name"))
  val n3 = tweets.withColumn("muser",
explode($"quoted_status.extended_tweet.entities.user_mentions")).select($"muser.screen_name".
alias("name"))
  val n4 = tweets.select($"quoted_status.user.screen_name".alias("name"))
  val n5 = tweets.withColumn("muser",
explode($"retweeted_status.extended_tweet.entities.user_mentions")).select($"muser.screen_nam
e".alias("name"))
  val n6 = tweets.withColumn("muser",
explode($"retweeted_status.entities.user_mentions")).select($"muser.screen_name".alias("name"
))
  val n7 = tweets.select($"retweeted_status.user.screen_name".alias("name"))
  val n8 = tweets.withColumn("muser",
explode($"retweeted_status.quoted_status.extended_tweet.entities.user_mentions")).select($"mu
ser.screen_name".alias("name"))
  val n9 = tweets.select($"retweeted_status.quoted_status.user.screen_name".alias("name"))
  val n10 = tweets.select($"user.screen_name".alias("name"))
  var res =
n1.union(n2).union(n3).union(n4).union(n5).union(n6).union(n7).union(n8).union(n9).union(n10)
  res = res.distinct()
  write(res)
}
```

Scenario T3

Computes the user and retweet count of tweets with at least three hashtags longer than five characters.

Tree Pattern

```
def t3Twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val user = twig.createNode("user")
  val name = twig.createNode("screen_name", on = true)
  val entities = twig.createNode("entities")
  val hashtags = twig.createNode("hashtags", min=3, max=1000000)
  val text = twig.createNode("text", condition="lengthgt5")
  val retweet_count = twig.createNode("retweet_count", on = true)
  twig = twig.createEdge(root, entities, true)
  twig = twig.createEdge(entities, hashtags, true)
  twig = twig.createEdge(hashtags, text, false)
  twig = twig.createEdge(root, user, true)
  twig = twig.createEdge(user, name, true)
  twig = twig.createEdge(root, retweet_count, true)
  twig.validate().get
}
```

TPM Pipeline

```
def q3(iteration: Int): Unit = {
  val tweets = loadTweets()
  val res = tweets.matchTree(t3Twig())
  write(res)
}
```

Spark Pipeline

```
def q3(iteration: Int): Unit = {
  val tweets = loadTweets()
  var flattened = tweets.withColumn("hashtag",
explode($"entities.hashtags"))//.select($"hashtag")
  flattened = flattened.filter(length($"hashtag.text") > 5)
  val cols = flattened.columns.filter(x => x != "hashtag")
  var res = flattened.groupBy(cols.head, cols.tail:
_*).agg(count($"hashtag").as("cnt"))
  res = res.filter($"cnt" >= 3)
  res = res.select($"user.screen_name", $"retweet_count")
  write(res)
}
```

Scenario T4

Computes a nested list of hashtags for each user mentions, if the hashtag contains “BTS”.

Tree Pattern

```
def t4Twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val name = twig.createNode("screen_name", on = true)
  val entities = twig.createNode("entities")
  val user_mentions = twig.createNode("user_mentions")
  val hashtags = twig.createNode("hashtags", min=1, max=100000, on = true)
  val text = twig.createNode("text", condition="containsBTS")
  twig = twig.createEdge(root, entities, true)
  twig = twig.createEdge(entities, hashtags, true)
  twig = twig.createEdge(hashtags, text, false)
  twig = twig.createEdge(entities, user_mentions, true)
  twig = twig.createEdge(user_mentions, name, false)
  twig.validate().get
}
```

TPM Pipeline

```
def q4(iteration: Int): Unit = {
  val tweets = loadTweets()
  val res = tweets.matchTree(t4Twig())
  write(res)
}
```

Spark Pipeline

```
def q4(iteration: Int): Unit = {
  val tweets = loadTweets()
  var flattened = tweets.withColumn("hashtag",
explode($"entities.hashtags"))//.select($"hashtag")
  flattened = flattened.filter($"hashtag.text".contains("BTS"))
  val cols = flattened.columns.filter(x => x != "hashtag")
  val aggregated = flattened.groupBy(cols.head, cols.tail:
_*).agg(collect_list($"hashtag").as("hashtags_shortened"))
  val flattened2 = aggregated.withColumn("muser",
explode($"entities.user_mentions"))
  val res = flattened2.select($"muser.screen_name", $"hashtags_shortened")
  write(res)
}
```

Scenario T5

Selects tweets with at least two user mentions that have name longer than 5 characters and at least two hashtags containing “BTS”.

Tree Pattern

```
def t5Twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val name = twig.createNode("screen_name")
  val entities = twig.createNode("entities")
  val user_mentions = twig.createNode("user_mentions")
  val hashtags = twig.createNode("hashtags", min=1, max=1000000)
  val text = twig.createNode("text", condition="containsBTS")
  twig = twig.createEdge(root, entities, true)
  twig = twig.createEdge(entities, hashtags, true)
  twig = twig.createEdge(hashtags, text, false)
  twig = twig.createEdge(entities, user_mentions, true)
  twig = twig.createEdge(user_mentions, name, false)
  twig.validate().get
}
```

TPM Pipeline

```
def q5(iteration: Int): Unit = {
  val tweets = loadTweets()
  val res = tweets.matchTree(t5Twig())
  write(res)
}
```

Spark Pipeline

```
def q5(iteration: Int): Unit = {
  val tweets = loadTweets()
  var flattened = tweets.withColumn("hashtag",
explode($"entities.hashtags"))//.select($"hashtag")
  flattened = flattened.filter($"hashtag.text".contains("BTS"))
  val cols1 = flattened.columns.filter(x => x != "hashtag")
  val aggregated = flattened.groupBy(cols1.head, cols1.tail:
_*).agg(count($"hashtag").as("hashtags_count"))
  val hashtagsFiltered =
aggregated.filter($"hashtags_count">=2).drop("hashtags_count")
  var flattened2 = hashtagsFiltered.withColumn("muser",
explode($"entities.user_mentions"))
  flattened2 = flattened2.filter(length($"muser.screen_name") > 5)
  val cols2 = flattened2.columns.filter(x => x != "muser")
  val aggregated2 = flattened2.groupBy(cols2.head, cols2.tail:
_*).agg(count($"muser").as("muser_count"))
  val res = aggregated2.filter($"muser_count">=2).drop("muser_count")
  write(res)
}
```

Scenario D1

Selects all inproceedings with their proceedings from the last century that have more than 10 citations

Tree Pattern

```
def q1twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val year = twig.createNode("year", condition = "ltltltlt2000")
  val cite = twig.createNode("cite", 10, 100000)
  val element = twig.createNode("element")
  val value = twig.createNode("_VALUE")

  twig = twig.createEdge(root, year, false)
  twig = twig.createEdge(root, cite, false)
  twig = twig.createEdge(cite, element, true)
  twig = twig.createEdge(element, value, true)
  twig.validate.get
}
```

TPM Pipeline

```
def q1(iteration: Int): Unit = {
  val proceedings = loadProceedings()
  val inproceedings = loadInproceedings()
  val inproceedings_proceedings = proceedings.join(inproceedings,
proceedings("_key") === inproceedings ("crf"))
  val res = tweets.matchTree(t1Twig())
  write(res)
}
```

Spark Pipeline

```
def q1(iteration: Int = 0) : Unit = {
  val proceedings = loadProceedings()
  val inproceedings = loadInproceedings()
  var inproceedings_proceedings = proceedings.join(inproceedings,
proceedings("_key") === inproceedings("crf"))
  inproceedings_proceedings = inproceedings_proceedings.filter($"year" <
2000 || $"inproceeding.year" < 2000)
  var cite1Flattened = inproceedings_proceedings.withColumn("citeCnt",
size($"cite._VALUE"))
  cite1Flattened = cite1Flattened.filter($"citeCnt" > 10)
  var cite2Flattened = inproceedings_proceedings.withColumn("citeCnt",
size($"inproceeding.cite._VALUE"))
  cite2Flattened = cite2Flattened.filter($"citeCnt" > 10)
  val res = cite1Flattened.union(cite2Flattened)
  write(res)
}
```

Scenario D2

Selects all authors that have at least 2 aliases containing “Mill”.

Tree Pattern

```
def q2twig(): Twig = {
  var twig = new Twig()
  val root = twig.createNode("root")
  val author = twig.createNode("author", min=2, max=1000000)
  val element = twig.createNode("element")
  val value = twig.createNode("_VALUE", condition = "containsMill")

  twig = twig.createEdge(root, author, false)
  twig = twig.createEdge(author, element, false)
  twig = twig.createEdge(element, value, true)
  twig.validate.get
}
```

TPM Pipeline

```
def q5(iteration: Int): Unit = {
  val authors = loadAuthors()
  val res = tweets.matchTree(t2Twig)
  write(res)
}
```

Spark Pipeline

```
def q2(iteration: Int = 0) : Unit = {
  val authors = loadAuthors()
  var mills = authors.withColumn("alias", explode($"author"))
  mills = mills.filter($"alias._VALUE".contains("Mill"))
  val cols = mills.columns.filter(x => x != "alias")
  mills = mills.groupBy(cols.head, cols.tail:
_*).agg{count($"alias").as("aliasCnt")}
  val res = mills.filter($"aliasCnt" >= 2)
  write(res)
}
```