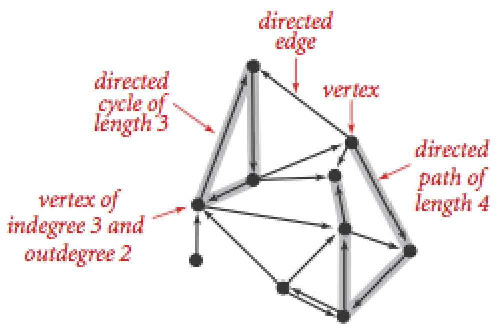
directed edge
vertex
directed cycle of length 3
directed path of length 4
vertex of indegree 3 and outdegree 2

Anatomy of a digraph

In [ ]:

```
import findspark
findspark.init()
import pyspark
```

In [ ]:

```
sc.stop()
```

In [ ]:

```
sc = pyspark.SparkContext(appName="graphs")
```

test on small rdd for finding a sink

In [ ]:

```
rdd = sc.parallelize([('1','4'), ('2','4'), ('3','4'), ('3','2') ])
rdd2 = rdd.map (lambda x : (x[1], x[0]))
rdd2 = rdd2.groupByKey()
rdd2 = rdd2.map (lambda x : (x[0], list(x[1])))
rdd3 = rdd.groupByKey()
rdd3 = rdd3.map (lambda x : (x[0], list(x[1])))
rdd4 = rdd2.subtractByKey(rdd3)
```

In [ ]:

```
rdd4.collect()
```

Find Sinks on big graph file

In [ ]:

```
rdd = sc.textFile("C:/Users/EASYFRONT/Documents/BD/requetessql/data/graph.txt")
rdd = rdd.map(lambda x : x.split('\t'))
#get the edges i.e eliminate the label and keep vertices only
rdd2 = rdd.map (lambda x : (x[2], x[0]))
#get the incoming edges for each node or vertex
rdd2 = rdd2.groupByKey()
rdd2 = rdd2.map (lambda x : (x[0], list(x[1])))
```

In [ ]:

```
#get the outcoming edges for each node
rdd3 = rdd.map (lambda x : (x[0], x[2]))
rdd3 = rdd3.groupByKey()
rdd3 = rdd3.map (lambda x : (x[0], list(x[1])))
```

In [ ]:

```
rdd3.collect()
```

In [ ]:

```
# filter nodes having only incoming edgesand having no outgoing edges
rdd4 = rdd2.subtractByKey(rdd3)
rdd4.collect()
```

Finding Universal Sinks

In [ ]:

```
#rddus = sc.parallelize([('1','3'), ('2','3'), ('3','4'),('2','4'),('1','4')])
rddus = sc.textFile("C:/Users/EASYFRONT/Documents/BD/requetessql/data/graph.txt")
rddus = rddus.map(lambda x : x.split('\t'))
rddus = rddus.map(lambda x : (x[0], x[2]))
#eliminate potential duplicate edges
rddus = rddus.distinct()
```

In [ ]:

```
rddus.collect()
```

In [ ]:

```
rddus.count()
```

In [ ]:

```
#We need to count how many nodes we have in the graph
nbnodes = rddus.flatMap(lambda x: [x[0]]+[x[1]]).distinct().count()
nbnodes
```

In [ ]:

```
#Compute graph sinks
rdds = rddus.map(lambda x : (x[1], x[0]))
rddsinks = rdds.subtractByKey(rddus).map(lambda x : (x[1],x[0]))
rdds.collect()
```

In [ ]:

```
#compute universal sinks
pre_sinks = rdds.groupByKey()
pre_sinks = pre_sinks.map(lambda x : (x[0], list(x[1])))
pre_sinks = pre_sinks.filter(lambda x : len(x[1])==nbnodes-1)
pre_sinks.collect()
```