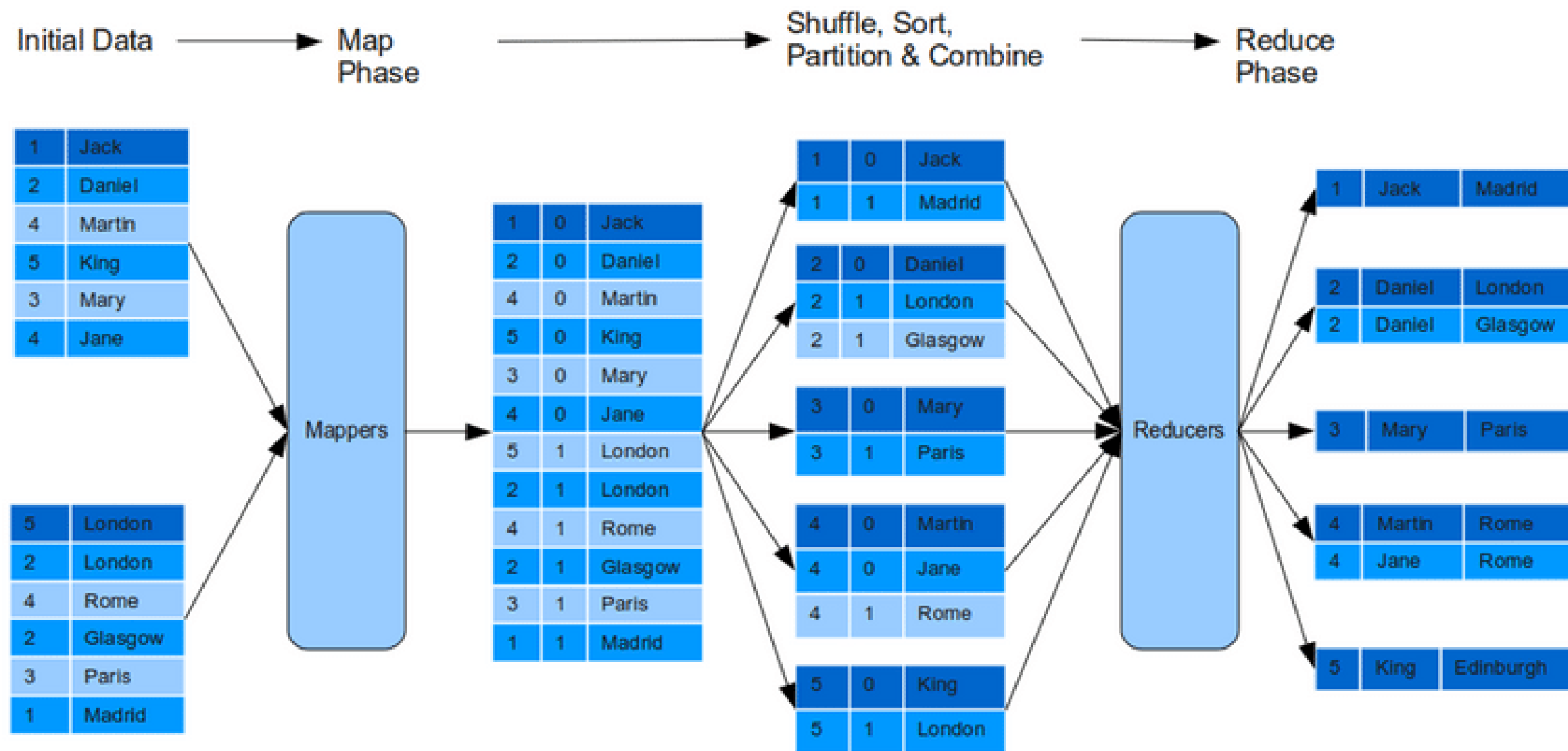


PAGERANK ET MAP REDUCE

MAP REDUCE



ALGORITHM



SQUELETTE DU PROGRAMME



```
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRank(MRJob):
    c = 0.15
    nIt = 10
    nodesInstances = set()

    def nodeInit(self, _, line):
        ...

    def rankInit(self, nodeId, AdjacencyList):
        ...

    def mapper(self, nodeId, node):
        ...

    def reducer(self, nodeId, values):
        ...

    def steps(self):
        return [MRStep(mapper=self.nodeInit, reducer=self.rankInit)] + \
            PageRank.nIt * [MRStep(mapper=self.mapper, reducer=self.reducer)]

if __name__ == '__main__':
    PageRank.run()
```

INITIALISATION

```
def nodeInit(self, _, line):  
    lineSplit = line.split('\t', maxsplit=1)  
    nodeFrom, nodeTo = lineSplit  
    PageRank.nodesInstances.add(nodeFrom)  
    PageRank.nodesInstances.add(nodeTo)  
    yield nodeFrom, nodeTo
```

```
def rankInit(self, nodeId, AdjacencyList):  
    node = {'rank': 1/len(PageRank.nodesInstances), 'AdjacencyList': list(AdjacencyList)}  
    yield nodeId, node
```



Si des nœuds ne citent aucun autre alors
il ne seront pas initialisés à cette étape.



**ensemble des nodeTo
groupés
sur les clés nodeFrom**

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRank(MRJob):
    c = 0.15
    nIt = 10
    nodesInstances = set()

    def nodeInit(self, _, line):
        ...

    def rankInit(self, nodeId, AdjacencyList):
        ...

    def mapper(self, nodeId, node):
        ...

    def reducer(self, nodeId, values):
        ...

    def steps(self):
        return [MRStep(mapper=self.nodeInit, reducer=self.rankInit)] + \
            PageRank.nIt * [MRStep(mapper=self.mapper, reducer=self.reducer)]

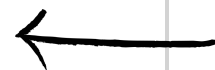
if __name__ == '__main__':
    PageRank.run()
```

ITÉRATIONS

```
def mapper(self, nodeId, node):
    yield nodeId, ('node', node)
    if node['AdjacencyList']:
        contribution = node['rank']/len(node['AdjacencyList'])
        for neighbourId in node['AdjacencyList']:
            yield neighbourId, ('contribution', contribution)

def reducer(self, nodeId, values):
    contributions = 0
    node = {'rank': 1/len(PageRank.nodesInstances), 'AdjacencyList': list()}
    for value in values:
        if value[0] == 'node':
            node = value[1]
        else:
            contributions += value[1]
    node['rank'] = PageRank.c * node['rank'] + (1 - PageRank.c) * contributions
    yield nodeId, node
```

Les nœuds non-initialisés
le sont à la
première itération



ITÉRATIONS

```
def mapper(self, nodeId, node):
    yield nodeId, ('node', node)
    if node['AdjacencyList']:
        contribution = node['rank']/len(node['AdjacencyList'])
        for neighbourId in node['AdjacencyList']:
            yield neighbourId, ('contribution', contribution)

def reducer(self, nodeId, values):
    contributions = 0
    node = {'rank': 1/len(PageRank.nodesInstances), 'AdjacencyList': list()}
    for value in values:
        if value[0] == 'node':
            node = value[1]
        else:
            contributions += value[1]
    node['rank'] = PageRank.c * node['rank'] + (1 - PageRank.c) * contributions
    yield nodeId, node
```

→ **values** = Iterator(("contrib.", contrib), ..., ("node", node), ..., ("contrib.", contrib))

ITÉRATIONS

```
def mapper(self, nodeId, node):
    yield nodeId, ('node', node)
    if node['AdjacencyList']:
        contribution = node['rank']/len(node['AdjacencyList'])
        for neighbourId in node['AdjacencyList']:
            yield neighbourId, ('contribution', contribution)

def reducer(self, nodeId, values):
    contributions = 0
    node = {'rank': 1/len(PageRank.nodesInstances), 'AdjacencyList': list()}
    for value in values:
        if value[0] == 'node':
            node = value[1]
        else:
            contributions += value[1]
    node['rank'] = PageRank.c * node['rank'] + (1 - PageRank.c) * contributions
    yield nodeId, node
```

Mise à jour du page rank



```
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRank(MRJob):
    c = 0.15
    nIt = 10
    nodesInstances = set()

    def nodeInit(self, _, line):
        ...

    def rankInit(self, nodeId, AdjacencyList):
        ...

    def mapper(self, nodeId, node):
        ...

    def reducer(self, nodeId, values):
        ...

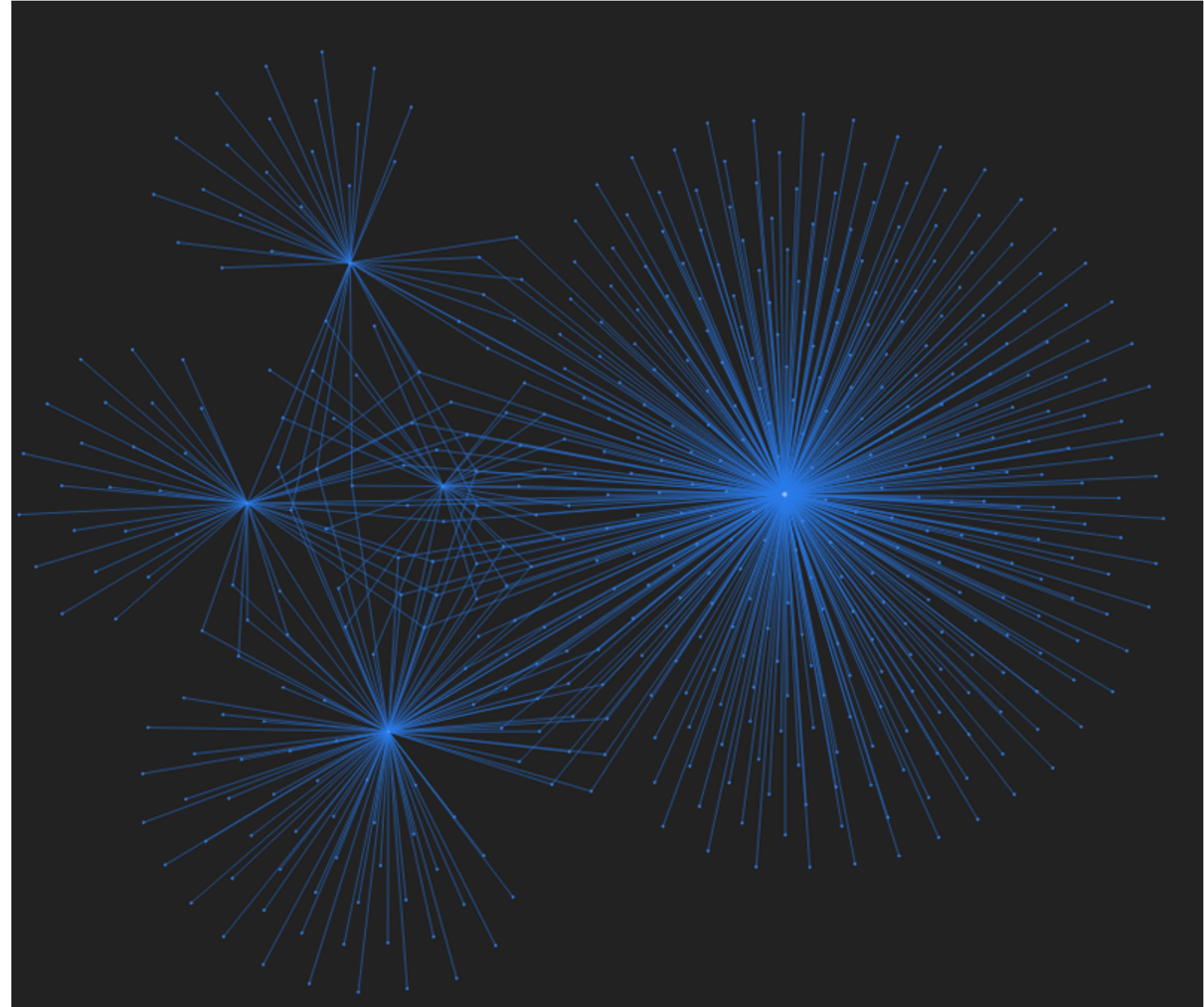
    def steps(self):
        return [MRStep(mapper=self.nodeInit, reducer=self.rankInit)] + \
            PageRank.nIt * [MRStep(mapper=self.mapper, reducer=self.reducer)]

if __name__ == '__main__':
    PageRank.run()
```

VISUALISATION

Top 5

Id.	Page rank
18	0.003234
4415	0.002902
737	0.001889
790	0.001852
1753	0.001582



PAGERANK ET MAP REDUCE