Parallel BFS Code in PySpark - With Path Tracking

SINGLE-SOURCE SHORTEST PATH

```
#Running on Colab
!pip install pyspark
!pip install -U -q PyDrive
!apt install openjdk-8-jdk-headless -qq
import os
os.environ['JAVA_HOME'] = '/usr/lib/jvm/java-8-openjdk-amd64'
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 2.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=77be91e8172b9661efaa9f873410083c70d9c4707ab9be892973d1173eb2df87
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
The following additional packages will be installed:
  libxtst6 openjdk-8-jre-headless
Suggested packages:
  openjdk-8-demo openjdk-8-source libnss-mdns fonts-dejavu-extra fonts-nanum fonts-ipafont-gothic
  fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  libxtst6 openjdk-8-jdk-headless openjdk-8-jre-headless
0 upgraded, 3 newly installed, 0 to remove and 45 not upgraded.
Need to get 39.7 MB of archives.
After this operation, 144 MB of additional disk space will be used.
Selecting previously unselected package libxtst6:amd64.
(Reading database ... 121918 files and directories currently installed.)
Preparing to unpack .../libxtst6_2%3a1.2.3-1build4_amd64.deb ...
Unpacking libxtst6:amd64 (2:1.2.3-1build4) ...
Selecting previously unselected package openjdk-8-jre-headless:amd64.
Preparing to unpack .../openjdk-8-jre-headless_8u402-ga-2ubuntu1~22.04_amd64.deb ...
Unpacking openjdk-8-jre-headless:amd64 (8u402-ga-2ubuntu1~22.04) ...
Selecting previously unselected package openjdk-8-jdk-headless:amd64.
Preparing to unpack .../openjdk-8-jdk-headless_8u402-ga-2ubuntu1~22.04_amd64.deb ...
Unpacking openjdk-8-jdk-headless:amd64 (8u402-ga-2ubuntu1~22.04) ...
Setting up libxtst6:amd64 (2:1.2.3-1build4) ...
Setting up openjdk-8-jre-headless:amd64 (8u402-ga-2ubuntu1~22.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/orbd to provide /usr/bin/orbd (orbd) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/servertool to provide /usr/bin/servertool (servertool) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/tnameserv to provide /usr/bin/tnameserv (tnameserv) in auto mode
Setting up openjdk-8-jdk-headless:amd64 (8u402-ga-2ubuntu1~22.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/clhsdb to provide /usr/bin/clhsdb (clhsdb) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/extcheck to provide /usr/bin/extcheck (extcheck) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/hsdb to provide /usr/bin/hsdb (hsdb) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/idlj to provide /usr/bin/idlj (idlj) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/javah to provide /usr/bin/javah (javah) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jhat to provide /usr/bin/jhat (jhat) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jsadebugd to provide /usr/bin/jsadebugd (jsadebugd) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/native2ascii to provide /usr/bin/native2ascii (native2ascii) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/schemagen to provide /usr/bin/schemagen (schemagen) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsgen to provide /usr/bin/wsgen (wsgen) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsimport to provide /usr/bin/wsimport (wsimport) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/xjc to provide /usr/bin/xjc (xjc) in auto mode
Processing triggers for libc-bin (2.35-0ubuntu3.4) ...
/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link
```

```
#  Importing Required Libraries
import pyspark
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf

# Create Spark session and ContextRun PySpark.
# create the session
conf = SparkConf().set("spark.ui.port","4050")
# create the context
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession.builder.appName("DataFrame").config('spark.ui.port', '4050').getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
```
v3.5.1
```
Master
```
local[*]
```
AppName
```
pyspark-shell
```

```
textFile = sc.textFile("input.dat")

count = sc.accumulator(0)

def customSplitNodesTextFile(node):
    if len(node.split(' ')) < 3:
        nid, distance = node.split(' ')
        neighbors = None
    else:
        nid, distance, neighbors = node.split(' ')
        neighbors = neighbors.split(':')
        neighbors = neighbors[:len(neighbors) - 1]
    path = nid
    return (nid , (int(distance), neighbors, path))

def customSplitNodesIterative(node):
    nid = node[0]
    distance = node[1][0]
    neighbors = node[1][1]
    path = node[1][2]
    elements = path.split('->')
    if elements[len(elements) - 1] != nid:
        path = path + '->' + nid;
    return (nid , (int(distance), neighbors, path))

def customSplitNeighbor(parentPath, parentDistance, neighbor):
    if neighbor!=None:
        nid, distance = neighbor.split(',')
        distance = parentDistance + int(distance)
        path = parentPath + '->' + nid
        return (nid, (int(distance), 'None', path))

def minDistance(nodeValue1, nodeValue2):
    neighbors = None
    distance = 0
    path = ''
    if nodeValue1[1] != 'None':
        neighbors = nodeValue1[1]
    else:
        neighbors = nodeValue2[1]
    dist1 = nodeValue1[0]
    dist2 = nodeValue2[0]
    if dist1 <= dist2:
        distance = dist1
        path = nodeValue1[2]
    else:
        count.add(1);
        distance = dist2
        path = nodeValue2[2]
    return (distance, neighbors, path)

def formatResult(node):
    nid = node[0]
    minDistance = node[1][0]
    path = node[1][2]
    return nid, minDistance, path

nodes = textFile.map(lambda node: customSplitNodesTextFile(node))

oldCount = 0
iterations = 0
while True:
    iterations += 1
    nodesValues = nodes.map(lambda x: x[1])
    neighbors = nodesValues.filter(lambda nodeDataFilter: nodeDataFilter[1]!=None).map(
        lambda nodeData: map(
            lambda neighbor: customSplitNeighbor(
                nodeData[2], nodeData[0], neighbor
            ), nodeData[1]
        )
    ).flatMap(lambda x: x)
    mapper = nodes.union(neighbors)
    reducer = mapper.reduceByKey(lambda x, y: minDistance(x, y))
    nodes = reducer.map(lambda node: customSplitNodesIterative(node))
    nodes.count() # We call the count to execute all the RDD transformations
    if oldCount == count.value:
        break
    oldCount=count.value

print('Finished after: ' + str(iterations) + ' iterations')
result = reducer.map(lambda node: formatResult(node))
```

```
Finished after: 5 iterations
```

```
result.collect()
```

```
[('5', 7, '1->3->5'),
 ('2', 8, '1->3->2'),
 ('1', 0, '1'),
 ('3', 5, '1->3'),
 ('4', 9, '1->3->2->4')]
```

My Own **Try**

```
# Note 999 means Infinity
# We have to store node name, distance from source, neigbours, neighbors edge weight, shortest path parent node
nodes=sc.parallelize([('1',('0',['2,10','3,5'],'1')), #Initially, parent node equals to itself and distance from source node (except for source itself) infinity
                      ('2',('999',['3,2','4,1'],'2')),
                      ('3',('999',['2,3','4,9','5,2'],'3')),
                      ('4',('999',['5,4'],'4')),
                      ('5',('999',['1,7','4,6'],'5'))])
nodes.collect()
```

```
[('1', ('0', ['2,10', '3,5'], '1')),
 ('2', ('999', ['3,2', '4,1'], '2')),
 ('3', ('999', ['2,3', '4,9', '5,2'], '3')),
 ('4', ('999', ['5,4'], '4')),
 ('5', ('999', ['1,7', '4,6'], '5'))]
```

```
nodes.values().collect()
```

```
[('0', ['2,10', '3,5'], '1'),
 ('999', ['3,2', '4,1'], '2'),
 ('999', ['2,3', '4,9', '5,2'], '3'),
 ('999', ['5,4'], '4'),
 ('999', ['1,7', '4,6'], '5')]
```

```
# Algorithm is 4 steps
#Step1 Use Map to get node values only i.e distance from source, neigbours, neighbors edge weight, shortest path parent node
#Step2 Use flatMap to Seperate neigbours and edge weights
#Step3 Take Union of org and step2 ans
#Step4 Use Reduce By Key to Find Minimum Distance from source node

nodes=sc.parallelize([('1',(0,['2,10','3,5'],'1')), #Initially, parent node equals to itself and distance from source node (except for source itself) infinity
                      ('2',(999,['3,2','4,1'],'2')),
                      ('3',(999,['2,3','4,9','5,2'],'3')),
                      ('4',(999,['5,4'],'4')),
                      ('5',(999,['1,7','4,6'],'5'))])
# Step 1
i=0

#Function for Neighbour Split
def NeighbourSplit(value):
  answer=[]
  dist,neighbours,parentNode = value
  for neighbourInfo in neighbours:
    neighbour,weight=neighbourInfo.split(',')
    answer.append((neighbour,(int(weight)+int(dist),None,parentNode+'->'+neighbour)))
  return answer

def MinDistance(value,value2):
  dist,structure,parentNode=value
  dist2,structure2,parentNode2=value2
  if(int(dist)<int(dist2)):
    return (dist,structure if structure != 'None' else structure2,parentNode)
  else:
   return (dist2,structure if structure != 'None' else structure2,parentNode2)

while i<4: #This should be while True but for testing Purposes keep i<4
  nodeValues=nodes.map(lambda x:x[1]) # get all values using MAP
  neighboursNode=nodeValues.flatMap(lambda value: NeighbourSplit(value))
  allInformation=nodes.union(neighboursNode)
  nodes=allInformation.reduceByKey(lambda x,y: MinDistance(x,y))
  i=i+1

nodes.collect()
```

```
[('5', (7, ['1,7', '4,6'], '1->3->5')),
 ('3', (5, ['2,3', '4,9', '5,2'], '1->3')),
 ('2', (8, ['3,2', '4,1'], '1->3->2')),
 ('4', (9, ['5,4'], '1->3->2->4')),
 ('1', (0, ['2,10', '3,5'], '1'))]
```

Code with Accumulators used For Convergence

```
# Function for Neighbour Split
def NeighbourSplit(value):
    dist, neighbours, parentNode = value
    answer = []
    if neighbours:
        for neighbourInfo in neighbours:
            neighbour, weight = neighbourInfo.split(',')
            answer.append((neighbour, (int(weight) + int(dist), None, parentNode + '->' + neighbour)))
    return answer

# Function to find minimum distance and update accumulator
def MinDistance(value1, value2):
    dist1, structure1, parentNode1 = value1
    dist2, structure2, parentNode2 = value2

    if int(dist1) < int(dist2):
        if int(dist2) != int(dist1):
            count.add(1)  # Increment accumulator when a shorter path is found
        return (dist1, structure1 if structure1 != 'None' else structure2, parentNode1)
    else:
        if int(dist1) != int(dist2):
            count.add(1)  # Increment accumulator when a shorter path is found
        return (dist2, structure1 if structure1 != 'None' else structure2, parentNode2)


# Function to format result
def formatResult(node):
    nid = node[0]
    minDistance = node[1][0]
    path = node[1][2]
    return nid, minDistance, path

nodes = sc.parallelize([('1', (0, ['2,10', '3,5'], '1')),
                        ('2', (999, ['3,2', '4,1'], '2')),
                        ('3', (999, ['2,3', '4,9', '5,2'], '3')),
                        ('4', (999, ['5,4'], '4')),
                        ('5', (999, ['1,7', '4,6'], '5'))])

count = sc.accumulator(0)
oldCount = 0
iterations = 0
converged = False

while not converged and iterations < 100:  # Large number for practical purposes
    iterations += 1
    count = sc.accumulator(0)

    # Step 1: Get node values
    nodeValues = nodes.map(lambda x: x[1])

    # Step 2: Separate neighbours and edge weights
    neighboursNode = nodeValues.flatMap(lambda value: NeighbourSplit(value))

    # Step 3: Take union of original and neighbours
    allInformation = nodes.union(neighboursNode)

    # Step 4: Reduce by key to find minimum distance
    nodes = allInformation.reduceByKey(lambda x, y: MinDistance(x, y))

    # Trigger the execution of transformations
    nodes.count()

    # Check for convergence
    if oldCount == count.value:
        converged = True
    else:
        oldCount = count.value

    print(f"Iteration {iterations}: {count.value} updates")

print(f'Finished after: {iterations} iterations')
result = nodes.map(lambda node: formatResult(node)).collect()
print(result)
#  By triggering execution, we can accurately update and check the value of the accumulator count.
[illegible]
```

```
nodes.collect()
```

```
[('4', (9, ['5,4'], '1->3->2->4')),
 ('1', (0, ['2,10', '3,5'], '1')),
 ('5', (7, ['1,7', '4,6'], '1->3->5')),
 ('3', (5, ['2,3', '4,9', '5,2'], '1->3')),
 ('2', (8, ['3,2', '4,1'], '1->3->2'))]
```