

RESEARCH PROJECT

DGPS: Towards Large-scale Distributed Graph Processing System

Name Dong Liu, Wentao Zhang
Email ldtenacity@gmail.com, wentao_zhang@pku.edu.cn
Institute Peking University DAIR Lab
Date October 18, 2022

Abstract

Analyzing large graphs provides valuable insights for social networking and web companies in content ranking and recommendations. However, the connectivity of nodes and edges in the graph structure makes it difficult to run many graph algorithms on large graphs in a distributed way. In this research project, we propose a new distributed graph data processing system: DGPS. DGPS is based on the distributed computing framework Hadoop, which implements parallel computing of graph algorithms. Specifically, in practice, we split the graph into several smaller graphs and put them on different machines for computation. Based on DGPS, we have implemented three graph algorithms: Small Dense Subgraphs Finding, PageRank and Graph Connected Components. We choose large scale graphs for our experiments and obtained efficient running speed and good performance no less than the single machine algorithm. These results confirm the efficiency and effectiveness of our framework. It also reveals the potential of distributed graph processing systems.

We provide the complete framework design, algorithm implementation, and experimental procedure and results. We use text descriptions, frame diagrams, pseudo-code and screenshots to show our results and workload in a variety of ways.

Keywords: Distributed system; Graph algorithm; Mapreduce

Contents

Abstract	ii
1 Introduction	1
1.1 Graph-Structure Data	1
1.2 Large Graph-Structure Data	1
1.3 Whole Graph Processing v.s. Distributed Graph Processing	1
1.4 Some Typical Distributed Graph Processing System and their frameworks	2
1.4.1 Pregel	2
1.4.2 Giraph	2
2 Basics of Math and Graph	3
2.1 Graph	3
2.2 Graph algorithms	4
2.2.1 Breadth-first search	4
2.2.2 Depth-first search	4
2.2.3 Shortest path	4
2.2.4 Cycle detection	4
2.2.5 Minimum spanning tree	4
2.2.6 Strongly connected components	5
2.2.7 Topological sorting	5
2.2.8 Graph colouring	5
3 Dilemmas and Motivation	6
3.1 The Dilemma of Processing on Large-scale Graph	6
3.2 Graph Processing Based on Existing Distributed Data Processing Systems. .	6
4 Framework of DGPS	7
4.1 Platform and Tools	7
4.1.1 Hadoop	7
4.1.2 Hadoop MapReduce	8
4.1.2.1 MapReduce Programming Model	8
4.2 Implementation	8
4.2.1 MapReduce Implementation	8
4.2.2 Graph Processing System Implementation	9

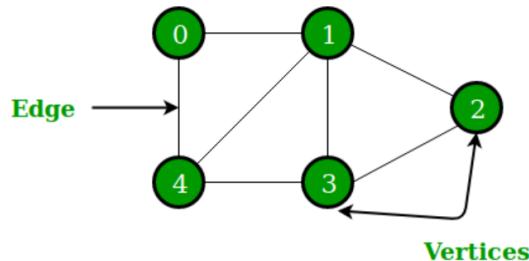
4.3	Architecture	10
4.3.1	Several Algorithms Application	10
4.3.2	Master-Slave Architecture	11
4.3.3	The details function of each step	11
4.3.4	Overview of the System	11
5	Experiments	13
5.1	Small Dense Subgraphs Finding	14
5.1.1	Setup and Code Analysis	14
5.1.2	Results	22
5.2	PageRank	26
5.2.1	Setup and Code Analysis	26
5.2.2	results	33
5.3	Graph Connected Components	37
5.3.1	Setup	37
5.3.2	Implementation	37
5.3.2.1	Input	38
5.3.2.2	Output	38
5.3.2.3	Design	39
5.3.3	Results	56
6	Difficulties and Solutions	60
6.1	Memory Allocation Problem	60
6.2	Java Heap Space is not enough	62
6.3	MapReduce Implementation in LargeStar Task	62
6.4	Shuffle Error of MapReduce	62
7	Conclusions	65

1. Introduction

1.1 Graph-Structure Data

A graph is an abstract data structure. It holds nodes that are usually related to each other. A node is a dataset, typically in the form of ordered pairs. Nodes are either connected or not connected to another node. The relation between nodes is usually defined as an Edge. Graphs are useful for their ability to associate nodes with other nodes.

A graph is often represented by an Adjacency matrix, A. If a graph has N nodes, then A has a dimension of ($N \times N$). People sometimes provide another feature matrix to describe the nodes in the graph. If each node has F numbers of features, then the feature matrix X has a dimension of ($N \times F$).



1.2 Large Graph-Structure Data

Nowadays the graph-structure data from the Internet is quite large. The web graph is a dramatic example of a large-scale graph. Google estimates that the total number of web pages exceeds 1 trillion; experimental graphs of the World Wide Web contain more than 20 billion nodes (pages) and 160 billion edges (hyperlinks). Graphs of social networks are another example. Facebook reportedly consists of more than a billion users (nodes) and more than 140 billion friendship relationships (edges) in 2012. The LinkedIn network contains almost 8 million nodes and 60 million edges. (Social network graphs are growing rapidly. Facebook went from roughly 1 million users in 2004 to 1 billion in 2012.) In the Semantic Web context, the ontology of DBpedia (derived from Wikipedia), currently contains 3.7 million objects (nodes) and 400 millions facts (edges).

1.3 Whole Graph Processing v.s. Distributed Graph Processing

As for whole graph processing, all computation will be completed in one server. The graph information will be stored on a single machine and all calculation will be executed in one

server.

As for distributed graph processing, a graph will be split into multiple partitions and the several worker nodes will process the partitions in parallel.

1.4 Some Typical Distributed Graph Processing System and their frameworks

1.4.1 Pregel

Pregel is a data flow paradigm and system for large-scale graph processing created at Google to solve problems that are hard or expensive to solve using only the MapReduce framework. While the system remains proprietary at Google, the computational paradigm was adopted by many graph-processing systems, and many popular graph algorithms have been converted to the Pregel framework.

Pregel is essentially a message-passing interface constrained to the edges of a graph. The idea is to "think like a vertex". A Pregel computation takes a graph and a corresponding set of vertex states as its inputs. At each iteration, referred to as a superstep, each vertex can send a message to its neighbors, process messages it received in a previous superstep, and update its state. Thus, each superstep consists of a round of messages being passed between neighbors and an update of the global vertex state.

1.4.2 Giraph

Giraph is a framework for performing offline batch processing of semi-structured graph data on massive scale. Giraph is loosely based upon Google's Pregel graph processing framework. Giraph performs iterative calculation on top of an existing Hadoop cluster. Giraph uses Apache Zookeeper to enforce atomic barrier waits and perform leader election

Apache Giraph is an iterative graph processing system built for high scalability. For example, it is currently used at Facebook to analyze the social graph formed by users and their connections. Giraph originated as the open-source counterpart to Pregel, the graph processing architecture developed at Google and described in a 2010 paper. Both systems are inspired by the Bulk Synchronous Parallel model of distributed computation introduced by Leslie Valiant. Giraph adds several features beyond the basic Pregel model, including master computation, sharded aggregators, edge-oriented input, out-of-core computation, and more. With a steady development cycle and a growing community of users worldwide, Giraph is a natural choice for unleashing the potential of structured datasets at a massive scale.

2. Basics of Math and Graph

2.1 Graph

A graph is often denoted by $\mathcal{G} = (V, E)$, where V is the set of vertices and E is the set of edges. An edge $e = u, v; v$ has two endpoints u and v , which are said to be joined by e . In this case, u is called a neighbor of v , or in other words, these two vertices are adjacent. Note that an edge can either be directed or undirected. A graph is called a directed graph if all edges are directed or undirected graph if all edges are undirected. The degree of vertex v , denoted by $d(v)$, is the number of edges connected with v . There are a few useful algebra representations for graphs, which are listed as follows.

- **Adjacency matrix:** for a simple graph $G = (V, E)$ with n -vertices, it can be described by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, where

$$A_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \text{ and } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

It is obvious that such matrix is a symmetric matrix when G is an undirected graph.

- **Degree matrix:** for a graph $G = (V, E)$ with n -vertices, its degree matrix $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix, where

$$D_{ii} = d(v_i).$$

- **Laplacian matrix:** for a simple graph $G = (V, E)$ with n -vertices, if we consider all edges in G to be undirected, then its Laplacian matrix $L \in \mathbb{R}^{n \times n}$ can be defined as

$$L = D - A.$$

Thus, we have the elements:

$$L_{ij} = \begin{cases} d(v_i) & \text{if } i = j, \\ -1 & \text{if } \{v_i, v_j\} \in E \text{ and } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the graph is considered to be an undirected graph for the adjacency matrix.

2.2 Graph algorithms

2.2.1 Breadth-first search

Traversing or searching is one of the fundamental operations which can be performed on graphs. In breadth-first search (BFS), we start at a particular vertex and explore all of its neighbours at the present depth before moving on to the vertices in the next level. Unlike trees, graphs can contain cycles (a path where the first and last vertices are the same). Hence, we have to keep track of the visited vertices. When implementing BFS, we use a queue data structure.

2.2.2 Depth-first search

In depth-first search (DFS) we start from a particular vertex and explore as far as possible along each branch before retracing back (backtracking). In DFS also we have to keep track of the visited vertices. When implementing DFS, we use a stack data structure to support backtracking.

2.2.3 Shortest path

The shortest path from one vertex to another vertex is a path in the graph such that the sum of the weights of the edges that should be travelled is minimum.

- Dijkstra's shortest path algorithm
- Bellman–Ford algorithm

2.2.4 Cycle detection

A cycle is a path in a graph where the first and last vertices are the same. If we start from one vertex, travel along a path and end up at the starting vertex, then this path is a cycle. Cycle detection is the process of detecting these cycles.

- Floyd cycle detection algorithm
- Brent's algorithm

2.2.5 Minimum spanning tree

A minimum spanning tree is a subset of the edges of a graph that connects all the vertices with the minimum sum of edge weights and consists of no cycles.

- Prim's algorithm
- Kruskal's algorithm

2.2.6 Strongly connected components

A graph is said to be strongly connected if every vertex in the graph is reachable from every other vertex.

- Kosaraju's algorithm
- Tarjan's strongly connected components algorithm

2.2.7 Topological sorting

Topological sorting of a graph is a linear ordering of its vertices so that for each directed edge (u, v) in the ordering, vertex u comes before v .

- Kahn's algorithm
- The algorithm based on depth-first search

2.2.8 Graph colouring

Graph colouring assigns colours to elements of a graph while ensuring certain conditions. Vertex colouring is the most commonly used graph colouring technique. In vertex colouring, we try to colour the vertices of a graph using k colours and any two adjacent vertices should not have the same colour. Other colouring techniques include edge colouring and face colouring. The chromatic number of a graph is the smallest number of colours needed to colour the graph.

- Algorithms using breadth-first search or depth-first search
- Greedy colouring

3. Dilemmas and Motivation

3.1 The Dilemma of Processing on Large-scale Graph

With explosion of data volume generated and collected from ubiquitous sensors, portable devices and the Internet, we are now moving into the “Big Data” era. There exist various modern Big Data applications relying on graph computing, including social networks, Internet of things, and neural networks. For example, Facebook has 1.44 billions monthly active users during the first quarter of 2015. Both user data and relationship among them are modeled by graphs for further exploration. To this end, it has become a very important problem to process, analyze, and understand these graphs.

To achieve high performance, existing graph systems aim to exploit massive parallelism using either distributed or shared memory models. Such graph systems process graphs in a repeated-relaxing manner (e.g., using Bellman-Ford algorithm variants) rather than in a sequential but work-optimal order. This introduces a fundamental trade-off between available parallelism and redundant computations.

3.2 Graph Processing Based on Existing Distributed Data Processing Systems.

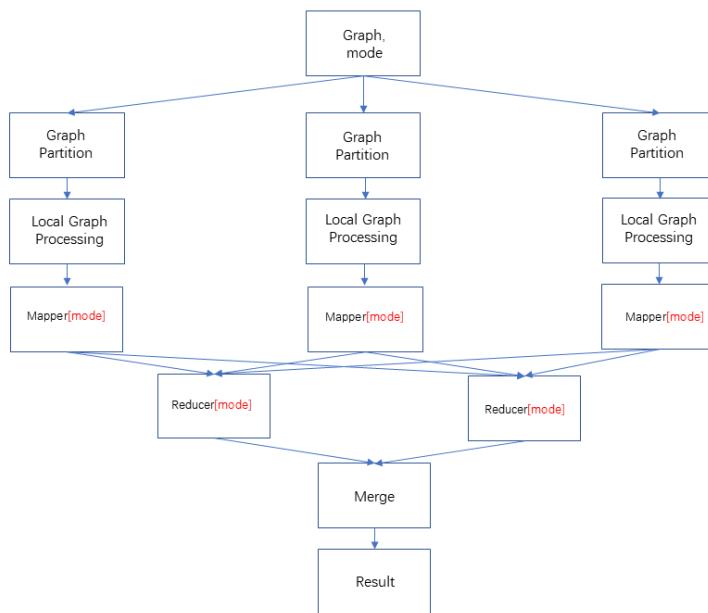
Building systems that process vast amounts of data has been made simpler by the introduction of the MapReduce framework, and its open-source implementation Hadoop. These systems offer automatic scalability to extreme volumes of data, automatic fault-tolerance, and a simple programming interface based around implementing a set of functions. However, it has been recognized that these systems are not always suitable when processing data in the form of a large graph. A framework similar to MapReduce—scalable, fault-tolerant, easy to program—but geared specifically towards graph data, would be of immense use. Google’s proprietary Pregel system was developed for this purpose. Pregel is a distributed message-passing system, in which the vertices of the graph are distributed across compute nodes and send each other messages to perform the computation.

4. Framework of DGPS

Our system provide a distributed processing system which can make queries to be split into several distributed jobs and to be executed on the set of machines in which the intermediate result is stored in the execution graphs.

In this system we provided three useful algorithms to execute the corresponding graph processing jobs on large graphs.

- Graph Partition
- Several mode (Several Algorithms)->Different Mapper & Reducer for different mode
- Parallel Processing



4.1 Platform and Tools

4.1.1 Hadoop

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.

It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

4.1.2 Hadoop MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a `_map_` function that processes a key/value pair to generate a set of intermediate key/value pairs, and a `_reduce_` function that merges all intermediate values associated with the same intermediate key.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

4.1.2.1 MapReduce Programming Model

The computation takes a set of `_input_` key/value pairs, and produces a set of `_output_` key/value pairs. The user of the MapReduce library expresses the computation as two functions: `_Map_` and `_Reduce_`.

`_Map_`, takes an input pair and produces a set of `_intermediate_` key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the `_Reduce_` function.

The `_Reduce_` function, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per `_Reduce_` invocation. The intermediate values are supplied to the user's reduce function via an iterator.

4.2 Implementation

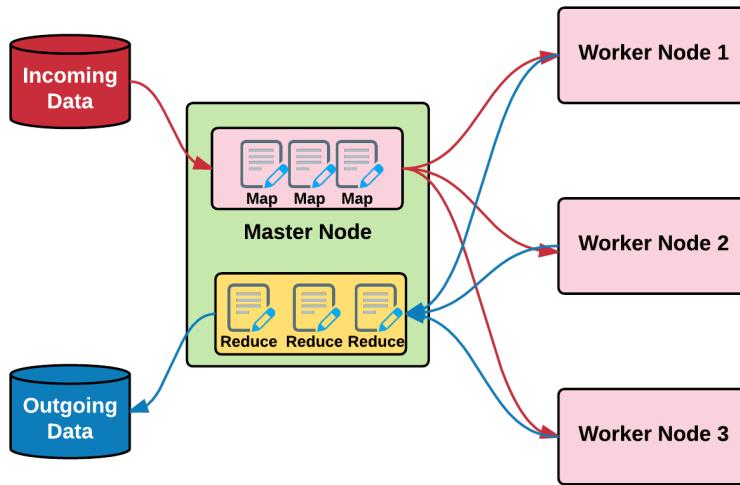
4.2.1 MapReduce Implementation

1. MapReduce library in the user program first splits the input file into M pieces. It then starts up many copies of the program on a cluster of machines.
2. One of copies of the program is master, and the rest are workers, which assigned by the master. There are M map tasks and R reduce tasks to assign. Master picks idle workers and assigns each one a map task or a reduce tasks.
3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.
4. The buffered pairs are written to local disk, partitioned into R regions by the user-defined partitioning function. The locations of these buffered pairs on the local disk are passed back to the master. Master is responsible for forwarding these locations to the reduce workers.

5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. Reduce worker then sorts it by the intermediate keys.

6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the reduce function is appended to a final output file for this reduce partition.

7. When all map tasks have been completed, master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.



4.2.2 Graph Processing System Implementation

Hadoop will distribute the entire graph onto multiple devices. When the graph size exceeds a single virtual machine memory, it will assign the extra part of graph to a new virtual machine.

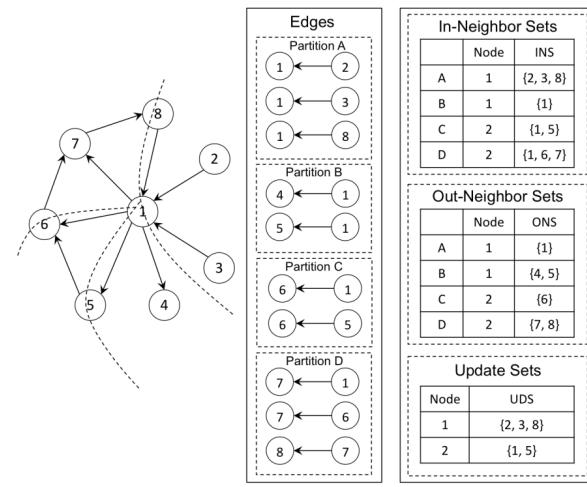
There are mainly two graph partitioning methods, some distributed graph frameworks like PowerGraph and GraphX use a vertex-cut partitioning to optimize for inter-node communication, while others use edge partitioning, an efficient graph partitioning strategy that assigns a roughly equal number of edges to each partition.

As for a typical implementation for graph partitioning (here we talk about directed graph), for each partition, the system will maintain two sets for it, the first one is the in-neighbor set, which contains all the outgoing edges from a specific node to the other nodes in the graph and the second one is out-neighbor set, which contains all the ingoing edges from nodes in the rest of the graph to a specific node.

$$INS(P_i) = u | (u, v) \in P_i$$

$$ONS(P_i) = v | (u, v) \in P_i$$

The figure below shows that a graph is divided into four partition, and assigned to 4 workers on 2 virtual machines.

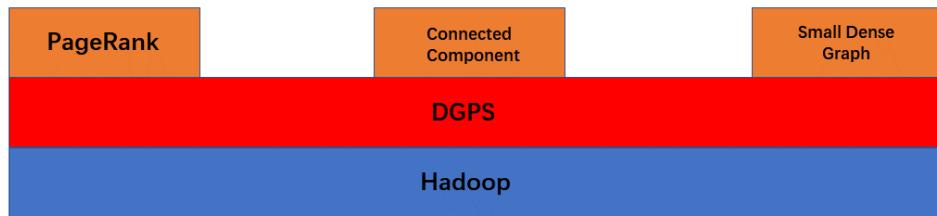


4.3 Architecture

4.3.1 Several Algorithms Application

- Algorithm I: Small Dense Subgraphs Finding
- Algorithm II: PageRank
- Algorithm III: Graph Connected Components

The details of implementation of these algorithms will be mentioned in the later chapter.

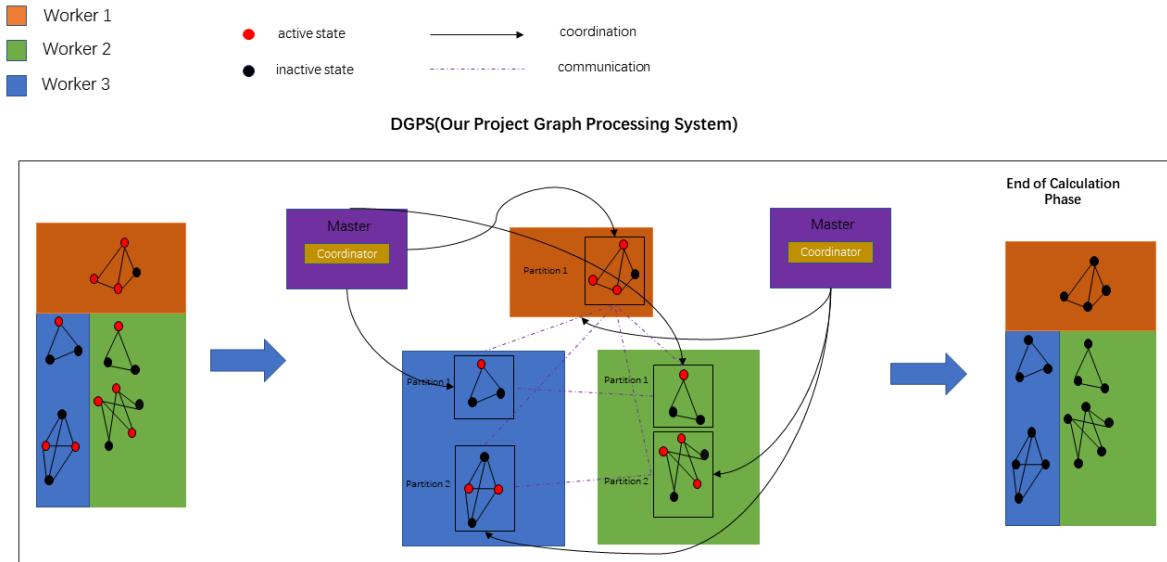


4.3.2 Master-Slave Architecture

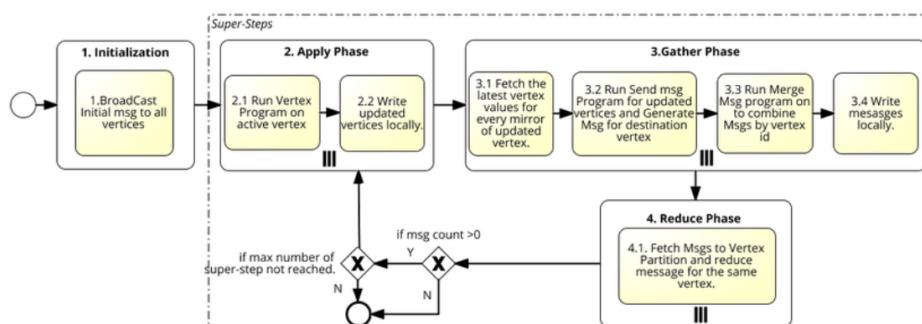
Our system is based on hadoop to realize the graph distributed computing. The figure below shows the architecture of DGPS.

There are several Key points in the System:

1. Graph is splitted into several partitions
2. Two states of nodes (active, inactive)
3. Several super-steps
4. Two states of nodes: active & inactivate
5. Master-Slave Architecture, two types of commands from master node: coordination & communication



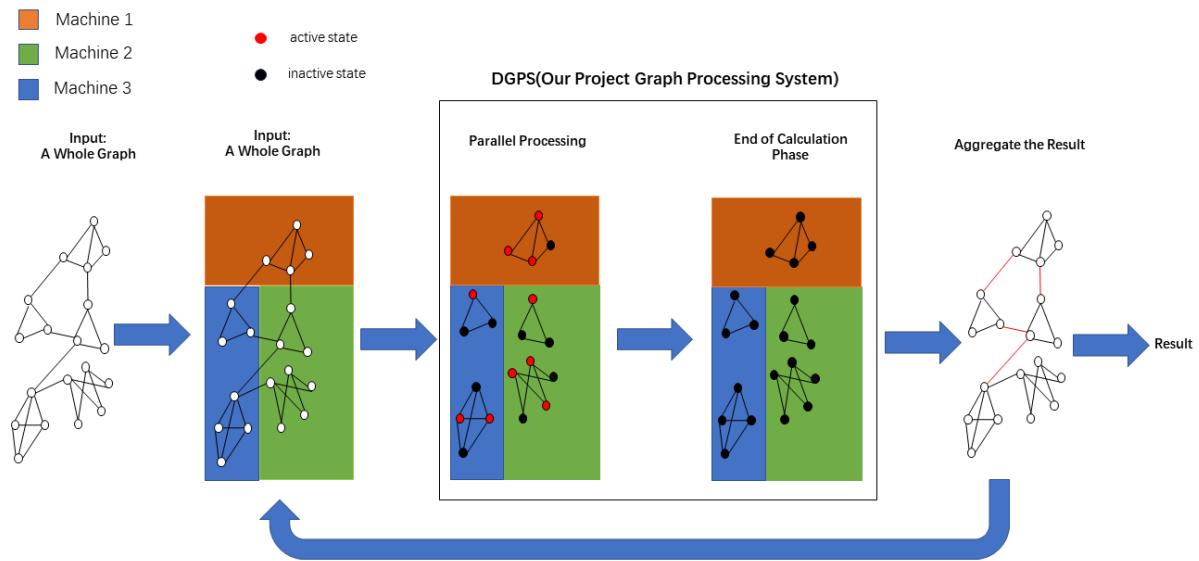
4.3.3 The details function of each step



4.3.4 Overview of the System

1. Input: A Whole Graph

2. Partitioning: Split the Graph into Several Partitions
3. Distributed Computing: Several Worker compute in parallel, each worker processes one or more partitions.
4. Supersteps: Each calculation iteration is referred to as a superstep, each vertex can send a message to its neighbors, process messages it received in a previous superstep, and update its state.
5. Aggregation: An aggregation of the results computed by different worker nodes.
6. Output: The result of aggregation step.



5. Experiments

Environment: IDEA:2021.2.3, hadoop:3.1.3, VMware:15, linux:CentOS7, VMs: Hadoop102 Hadoop103 Hadoop104.

```
[root@hadoop102 hadoop-3.1.3]# jpsall
=====
 hadoop102 =====
96386 NameNode
98097 Jps
96566 DataNode
96886 NodeManager
97064 JobHistoryServer
=====
 hadoop103 =====
79361 DataNode
79794 NodeManager
79573 ResourceManager
81514 Jps
=====
 hadoop104 =====
52066 SecondaryNameNode
53013 Jps
52155 NodeManager
51949 DataNode
[root@hadoop102 hadoop-3.1.3]#
```

Dataset: <http://snap.stanford.edu/biodata/datasets/10008/10008-PP-Decagon.html>

This is a protein-protein association network that includes direct (physical) protein-protein interactions, as well as indirect (functional) associations between human proteins. Nodes represent proteins and edges represent associations between them.

dataset	statistics
Nodes	19081
Edges	715612
Nodes in largest SCC	19065
Fraction of nodes in largest SCC	1.000000
Edges in largest SCC	715602
Fraction of edges in largest SCC	0.999986
Average clustering coefficient	0.233544
Number of triangles	157277164
Fraction of closed triangles	0.082030
Diameter (longest shortest path)	8
90-percentile effective diameter	3.199186

5.1 Small Dense Subgraphs Finding

5.1.1 Setup and Code Analysis

Let $\mathcal{G} = (V, E)$ be an undirected graph. For any subset of vertices V' from V , the density "rho" of the subgraph induced by V' is defined as: $\text{rho}(V') = |E[V']|/|V'|$, where $|E[V']|$ subset of E is the set of edges whose endpoints lie in V' . In this problem we are asked to design and implement a MapReduce algorithm that, given a density rho, finds a subgraph of \mathcal{G} with density $\geq \text{rho}$, i.e., a set V' such that $\text{rho}(V') \geq \text{rho}$. The challenge is to find a subgraph that is as small as possible. While the problem of finding the smallest dense-enough subgraph is hard, we should attempt at finding a subgraph with density $\geq \text{rho}$ as small as we can.

The problem is divided into 3 basic steps:

- Compute the density of the whole original graph, "ComputeGraphDensity" class.
- Use the just computed density as a threshold for both pruning and partitioning of the graph, "GraphPartitioning" class.
- Find the smallest subgraph with the corresponding degree threshold (rho value), "ComputeMinDensitySubgraph" class.

Each step is translated in a corresponding MapReduce round:

- 1. For the first round the Mapper just parses the input graph file, line by line (edge by edge). All the edges are being transferred to only one Reducer. The Reducer computes the density of the whole graph, iterating over the edges and keeping track of the nr of edges and nodes. The output file produced by this Reducer is of the following form:

```
#  
# [src]<tab>[dst]<tab>[graph_density] #  
# ... [src]<tab>[dst]<tab>[graph_density] ... #  
# ..... #  
  
#  
  
map phase  
**input**:  
““ [src]<tab>[dst] ““  
**output**:  
““ <1, [src]<tab>[dst]> ““  
**code:**  
1 public void map(LongWritable key, Text edge, Context  
2   context) throws IOException {  
3     if (edge.toString().isEmpty()) return;  
4     try {
```

```

4         context.write(new IntWritable(1), edge);
5     } catch (InterruptedException e) {
6         e.printStackTrace();
7     }
8 }
```

reduce phase

input:

“<1, list[src]<tab>[dst]>“

output

“<[src]<tab>[dst], [graph_density]>“

code:

```

1 public void reduce(IntWritable key, Iterable<Text> edges,
2 Context context) throws IOException {
3     HashSet<Integer> hashSet = new HashSet<>();
4     HashSet<String> edgeSet = new HashSet<>();
5
6     Iterator<Text> iter = edges.iterator();
7     Queue<String> queue = new LinkedList<>();
8     String hashEdge;
9     //while there are edges left
10    while (iter.hasNext()) {
11        Text line = (Text) iter.next();
12        String [] edge = line.toString().split("\t");
13        int src = Integer.parseInt(edge[0]);
14        int dst = Integer.parseInt(edge[1]);
15
16        if (src < dst) hashEdge = src + " - " + dst;
17        else hashEdge = dst + " - " + src;
18
19        if (edgeSet.contains(hashEdge)) continue;
20        edgeSet.add(hashEdge);
21
22        queue.add(line.toString());
23        hashSet.add(src);
24        hashSet.add(dst);
25    }
26
27    //computing density of the whole graph as |E|/|V|
28    int edgeCount = queue.size();
29    int nodeCount = hashSet.size();
30    double graph_density = (double) edgeCount / (double)
31    nodeCount;
32
33    Text K_out = new Text(), V_out = new Text();
34    while (!queue.isEmpty()) {
```

```

33     K_out.set(queue.poll());
34     V_out.set(String.valueOf(graph_density));
35     try {
36         context.write(K_out, V_out);
37     } catch (InterruptedException e) {
38         e.printStackTrace();
39     }
40 }
41 }
42 }
```

- 2. In the second round the Mapper parses the output file, line by line, produced by the previous MapReduce round. For each parsed line, emits a <subgraphID, edge> key-value pair, randomly choosing the “subgraphID” in [0 graph_density) range. The Reducer receives as input a <subgraphID, List<edge>> keyvalue pair, which corresponds to the specific “subgraphID” subgraph of the original graph. For each edge of the corresponding subgraph, it explores the degrees (in the subgraph’s context) of both edge’s endpoints, and emits the edge only if the degrees of the both endpoints are greater than graph_density. This round is very important to the success of the entire algorithm because: first, the algorithm takes advantage of parallelized computation (subgraph pruning is done in each graph partition) and second, it throws away edges which endpoints have very few links (endpoint_degree < graph_density), and thus nodes with small degree that don’t contribute as much to the final desired result, are pruned. In sum, the purpose of the round is to cross out those nodes that don’t contribute to the result, and, the output are the edges which endpoints are more suitable to help us in the task.

map phase

input:

“ same as the output of previous reduce phase “

output:

“ <[subgraphID], [edge]> subgraphID is randomly picked. “

code:

```

1 public void map(LongWritable key, Text record/*edge "[src
2 ]<tab>[dst]" */ , Context context) throws IOException {
3     if (record.toString().isEmpty()) return;
4
5     String[] line = record.toString().split("\t");
6     String edge = line[0] + "\t" + line[1];
7     double graphDensity = Double.parseDouble(line[2]);
8     int partitions = (int) Math.round(graphDensity);
9     int partition = new Random().nextInt(partitions);
10
11    System.out.println("Mapper: [graphDensity " +
12        graphDensity + "] [partitions " + partitions +
13        "] [subgraph " + partition + "] edge - " +
14        edge);
```

```

12
13     //<key, value> pair as <subgraphID, (edge, density)>
14     try {
15         context.write(new IntWritable(partition), new
16                     Text(edge + "\t" + graphDensity));
17     } catch (InterruptedException e) {
18         e.printStackTrace();
19     }

```

reduce phase

input:

““ <[subgraphID], List[edge]> ““

output:

““ <NULL, [src]<tab>[dst]> ““

code:

```

1 public void reduce(IntWritable key, Iterable<Text> edges,
2 Context context) throws IOException {
3     int graphDensity = 0;
4     HashMap<Integer, Integer> hashMap = new HashMap<>();
5
6     Iterator<Text> iter = edges.iterator();
7     Queue<String> queue = new LinkedList<>();
8     //while there are edges left
9     while (iter.hasNext()) {
10         Text line = iter.next();
11         String[] edge = line.toString().split("\\\\t");
12         queue.add(edge[0] + "\t" + edge[1]);
13
14         int src = Integer.parseInt(edge[0]);           //first
15         node of this edge
16         int dst = Integer.parseInt(edge[1]);           //second
17         node of this edge
18
19         if (hashMap.containsKey(src)) hashMap.put(src,
20             hashMap.get(src) + 1);      //update degree of this
21         node
22         else hashMap.put(src, 1);
23
24         if (hashMap.containsKey(dst)) hashMap.put(dst,
25             hashMap.get(dst) + 1);      //update degree of this
26         node
27         else hashMap.put(dst, 1);
28
29         if (!iter.hasNext()) graphDensity = (int) Math.
30             round(Double.parseDouble(edge[2]));
31     }

```

```

24
25     Text output_edge = new Text();
26     while (!queue.isEmpty()) {
27         String[] edge_ = queue.poll().split("\\\\t");
28         int src = Integer.parseInt(edge_[0]);
29         int dst = Integer.parseInt(edge_[1]);
30
31         if (hashMap.get(src) > graphDensity && hashMap.
32             get(dst) > graphDensity) {
33             output_edge.set(src + "\\t" + dst);
34             System.out.println("Reducer: [subgraph " +
35                 key.toString() + "] OUT(null, "
36                 + output_edge.toString() + ")");
37             try {
38                 context.write(null, output_edge);
39             } catch (InterruptedException e) {
40                 e.printStackTrace();
41             }
42         }
43     }

```

- 3. In this round the Mapper just parses the partitioned input graph file, produced by the previous MapReduce round, line by line (edge by edge), and outputs all the edges with the same key so that they are transferred to only one Reducer. The Reducer iterates over all the nodes, and at each iteration it removes from the graph the node with the minimum degree. Then it computes the new graph density and if the new graph is better, it stores the new best density of at least rho and the new best graph. This will continue until all nodes have been removed from the graph and in the end we will have stored the smallest graph of at least density rho the algorithm could find.

map phase

input:

“ same as the output of previous reduce phase “

output:

“ <1, [edge]> “

code:

```

1   public void map(LongWritable key, Text edge/*edge "[
2     src]<tab>[dst]" */ , Context context) throws
3     IOException {
4         if (edge.toString().isEmpty()) return;
5         //<key, value> pairs as <1, edge>
6         try {
7             context.write(new IntWritable(1), edge);
8         } catch (InterruptedException e) {
9             e.printStackTrace();
10        }

```

```
9 }
```

reduce phase

input:

““ <1, list[edge]> ““

output:

““ <NULL, [edge]> ““

code:

```
1 public void reduce(IntWritable key, Iterable<Text> edges,
2   Context context) throws IOException {
3   final int targetDensity = 4; //the corresponding rho value
4   int edgeCount = 0, nodeCount = 0;
5   Hashtable<Integer, LinkedList<Integer>> graph = new
6     Hashtable<>();
7   Hashtable<Integer, LinkedList<Integer>> minSubgraph =
8     null;
9   int maxIdxNode = 0;
10  for (Text text : edges) {
11    String [] edge = text.toString().split("\t");
12    int src = Integer.parseInt(edge[0]);
13    int dst = Integer.parseInt(edge[1]);
14
15    if (src > maxIdxNode) maxIdxNode = src;
16    if (dst > maxIdxNode) maxIdxNode = dst;
17
18    if (graph.containsKey(src)) graph.get(src).add(
19      dst);
20    else {
21      LinkedList<Integer> src_neighbors = new
22        LinkedList<>();
23      src_neighbors.add(dst);
24
25      graph.put(src, src_neighbors);
26    }
27
28    if (graph.containsKey(dst)) graph.get(dst).add(
29      src);
30    else {
31      LinkedList<Integer> dst_neighbors = new
32        LinkedList<>();
33      dst_neighbors.add(src);
34
35      graph.put(dst, dst_neighbors);
36    }
37
38    edgeCount++;
39  }
```

```

32 }
33
34 nodeCount = graph.size();
35 IndexMinPQ<Integer> minHeap = new IndexMinPQ<>(
36     maxIdxNode + 1);
37
38 for (int node : graph.keySet()) {
39     minHeap.insert(node, graph.get(node).size());
40     System.out.println("node " + node + "\tdegree " +
41         graph.get(node).size());
42 }
43
44 int minNode, minNodesSub = 100, minEdgeCount = 0;
45 while (!minHeap.isEmpty())
46 {
47     minNode = minHeap.delMin(); //extract the
48     // minimum degree node from the heap
49     System.out.println("minNode = " + minNode);
50     for (int neighbor : graph.get(minNode)) {
51         //remove the minimum degree node from the neighbor's
52         // adjacency list
53         graph.get(neighbor).remove(Integer.valueOf(
54             minNode));
55         minHeap.decreaseKey(neighbor, graph.get(
56             neighbor).size()); //update the neighbor node
57         // degree
58     }
59
60     edgeCount -= graph.get(minNode).size(); // update
61     // number of edges of the
62     // graph
63     nodeCount--; //update
64     // number of nodes of the graph
65     graph.remove(minNode); //remove
66     // the minimum degree node from the graph structure
67
68     if (((((float) edgeCount / (float) nodeCount) >=
69         4) && nodeCount < minNodesSub) {
70         minNodesSub = nodeCount;
71         minEdgeCount = edgeCount;
72         minSubgraph = new Hashtable<>(graph);
73
74         System.out.println("Reducer: minNodesSub = "
75             + minNodesSub);
76     }
77 }
78
79 if (minSubgraph == null) return;

```

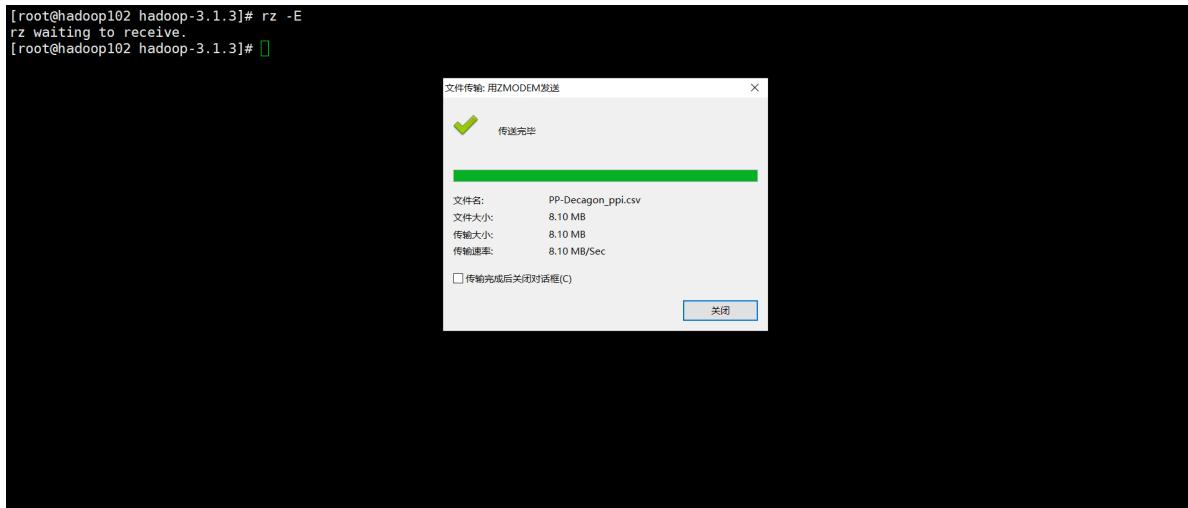
```

69   try {
70     float minDensity = (float) minEdgeCount / (float)
71     minSubgraph.size();
72     System.out.println("Reducer: density of the
73       minimum subgraph = " + minDensity);
74     context.write(null, new Text(String.valueOf(
75       minDensity)));
76   } catch (InterruptedException e) {
77     e.printStackTrace();
78   }
79
80   Text output_node = new Text();
81   for (int node : minSubgraph.keySet()) {
82     output_node.set(String.valueOf(node));
83     try {
84       System.out.println("Reducer: OUT(null, " +
85         output_node + ")");
86       context.write(null, output_node);
87     } catch (InterruptedException e) {
88       e.printStackTrace();
89     }
90   }
91
92   String hashEdge;
93   HashSet<String> edgeSet = new HashSet<>();
94   Text output_edge = new Text();
95   for (int src : minSubgraph.keySet()) {
96     for (int dst : minSubgraph.get(src)) {
97       if (src < dst) hashEdge = src + "\t" + dst;
98       else hashEdge = dst + "\t" + src;
99
100      if (!edgeSet.contains(hashEdge)) {
101        edgeSet.add(hashEdge);
102        output_edge.set(hashEdge);
103        try {
104          System.out.println("Reducer: OUT(null
105            , " + hashEdge + ")");
106          context.write(null, output_edge);
107        } catch (InterruptedException e) {
108          e.printStackTrace();
109        }
110      }
111    }
112  }
113}

```

5.1.2 Results

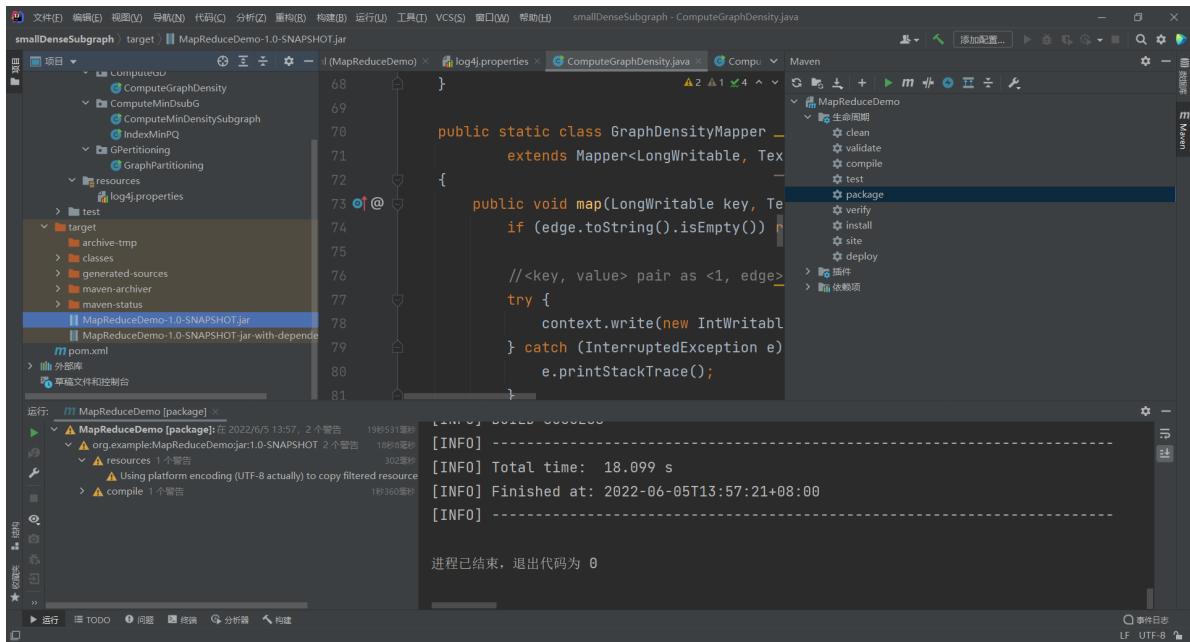
1. load dataset file(PP-Decagon_ppi.csv) to VM



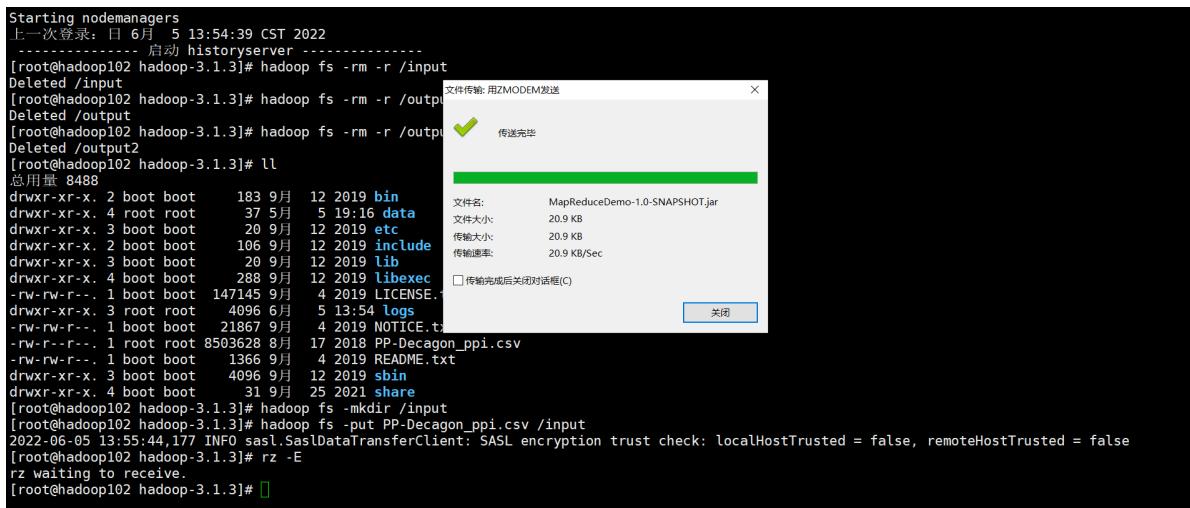
2. put dataset file(PP-Decagon_ppi.csv) to HDFS

```
[root@hadoop102 hadoop-3.1.3]# ll
总用量 8488
drwxr-xr-x. 2 root root    183 9月 12 2019 bin
drwxr-xr-x. 4 root root     37 5月 19 16 data
drwxr-xr-x. 3 root root     20 9月 12 2019 etc
drwxr-xr-x. 2 root root    106 9月 12 2019 include
drwxr-xr-x. 3 root root     20 9月 12 2019 lib
drwxr-xr-x. 4 root root    288 9月 12 2019 libexec
-rw-rw-r--. 1 root root 147145 9月 4 2019 LICENSE.txt
drwxr-xr-x. 3 root root    4096 6月 5 13:54 logs
-rw-rw-r--. 1 root root   21867 9月 4 2019 NOTICE.txt
-rw-rw-r--. 1 root root 8503628 8月 17 2018 PP-Decagon_ppi.csv
-rw-rw-r--. 1 root root   1366 9月 4 2019 README.txt
drwxr-xr-x. 3 root root    4096 9月 12 2019 sbin
drwxr-xr-x. 4 root root    31 9月 25 2021 share
[root@hadoop102 hadoop-3.1.3]# hadoop fs -mkdir /input
[root@hadoop102 hadoop-3.1.3]# hadoop fs -put PP-Decagon_ppi.csv /input
2022-06-05 13:55:44,177 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
[root@hadoop102 hadoop-3.1.3]#
```

3. compile program to Jar file



4. load Jar file to VM



5. run program in VM

task1: compute graph density

```

-rw-rw-r-- 1 boot boot 21867 9月 4 2019 NOTICE.txt
-rw-r--r-- 1 root root 8503628 8月 17 2018 PP-Decagon_ppi.csv
-rw-rw-r-- 1 boot boot 1366 9月 4 2019 README.txt
drwxr-xr-x 3 boot boot 4096 9月 12 2019 sbin
drwxr-xr-x 4 boot boot 31 9月 25 2021 share
[root@hadoop102 hadoop-3.1.3]# hadoop fs -mkdir /input
[root@hadoop102 hadoop-3.1.3]# hadoop fs -put PP-Decagon_ppi.csv /input
2022-06-05 13:55:44,177 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 14:00:24,257 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and ex
ecute your application with ToolRunner to remedy this
2022-06-05 14:00:24,275 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654408
482149_0001
2022-06-05 14:00:24,348 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 14:00:24,454 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 14:00:24,487 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 14:00:24,524 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 14:00:24,550 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-05 14:00:24,642 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 14:00:24,676 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1654408482149_0001
2022-06-05 14:00:24,814 INFO conf.Configuration: resource-types.xml not found
2022-06-05 14:00:24,815 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-06-05 14:00:24,973 INFO impl.YarnClientImpl: Submitted application application_1654408482149_0001
2022-06-05 14:00:25,002 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654408482149_0001/
2022-06-05 14:00:25,002 INFO mapreduce.Job: Running job: job_1654408482149_0001

```

result:

The screenshot shows the Hadoop Web UI interface. The main title is "Browse Directo" and the sub-path is "/output". A modal window titled "File information - part-r-00000" is open. Inside the modal, there's a "Block information" section showing "Block ID: 1073743168", "Block Pool ID: BP-1748348618-192.168.10.102-1651749363600", "Generation Stamp: 2347", "Size: 20669032", and "Availability" which lists three hosts: "hadoop102", "hadoop103", and "hadoop104". Below this is a "File contents" section showing a table of data with columns "8480", "8487", and "37.50390440752581". The table contains approximately 15 rows of data. To the right of the modal, there's a preview of the file showing its first few lines.

task2: graph partitioning

```

[root@hadoop102 hadoop-3.1.3]# hadoop jar MapReduceDemo-1.0-SNAPSHOT.jar GPartitioning.GraphPartitioning /output/part-r-00000 /output2
2022-06-05 16:54:40,006 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 16:54:40,338 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and ex
ecute your application with ToolRunner to remedy this.
2022-06-05 16:54:40,353 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654419
132285_0002
2022-06-05 16:54:40,415 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:54:40,510 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 16:54:40,532 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:54:40,561 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:54:40,582 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-05 16:54:40,670 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:54:40,697 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1654419132285_0002
2022-06-05 16:54:40,697 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-06-05 16:54:40,826 INFO conf.Configuration: resource-types.xml not found
2022-06-05 16:54:40,865 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-06-05 16:54:40,892 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654419132285_0002/
2022-06-05 16:54:40,892 INFO mapreduce.Job: Running job: job_1654419132285_0002
2022-06-05 16:54:46,013 INFO mapreduce.Job: Job job_1654419132285_0002 running in uber mode : false
2022-06-05 16:54:46,016 INFO mapreduce.Job: map 0% reduce 0%
2022-06-05 16:54:55,114 INFO mapreduce.Job: map 100% reduce 0%
2022-06-05 16:55:01,200 INFO mapreduce.Job: map 100% reduce 100%
2022-06-05 16:55:02,224 INFO mapreduce.Job: Job job_1654419132285_0002 completed successfully
2022-06-05 16:55:02,286 INFO mapreduce.Job: Counters: 53
    File System Counters
        FILE: Number of bytes read=24962710
        FILE: Number of bytes written=50361849
        FILE: Number of read operations=0
        FILE: Number of large read operations=0

```

result:

The screenshot shows the Hadoop Web UI interface. On the left, there's a sidebar with links like 'Hadoop', 'Overview', 'Datanodes', 'Datanode Volume Failures', 'Snapshot', 'Startup Progress', and 'Utilities'. Below this is a 'Browse Directories' section with a search bar and a dropdown menu set to '/output2'. It lists two entries: '_SUCCESS' and 'part-r-00000'. The '_SUCCESS' entry is 0 bytes and was modified at 2022-06-05 16:55:44.831. The 'part-r-00000' entry is 196 bytes and was modified at 2022-06-05 16:55:44.497. Both entries have 'root' as the owner and 'rwxr--r--' permissions. On the right, a modal window titled 'File information - part-r-00000' is open. It displays the file's size (196), block ID (1073743238), block pool ID (BP-1748348618-192.168.10.102-16517493600), generation stamp (2417), and availability (hadoop103, hadoop104, hadoop102). Below this, the 'File contents' section shows the file's binary data as a list of hex values: 1994 8452, 8452 4914, 7157 3312, 6233 8452, 1956 4914, 8452 7311, 6233 7314, 1994 7514.

task3: compute min subgraph density

```
[root@hadoop102 hadoop-3.1.3]# hadoop jar MapReduceDemo-1.0-SNAPSHOT.jar ComputeMinDsubG.ComputeMinDensitySubgraph /output2/part-r-00000 /output3
Bytes Written=196
2022-06-05 16:55:44.497 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 16:55:44.831 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2022-06-05 16:55:44.843 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654419132285_0003
2022-06-05 16:55:44.910 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:55:45.003 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 16:55:45.034 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:55:45.066 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:55:45.091 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-05 16:55:45.188 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 16:55:45.617 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1654419132285_0003
2022-06-05 16:55:45.617 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-06-05 16:55:45.743 INFO conf.Configuration: resource-types.xml not found
2022-06-05 16:55:45.744 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-06-05 16:55:45.781 INFO impl.YarnClientImpl: Submitted application application_1654419132285_0003
2022-06-05 16:55:45.807 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654419132285_0003/
2022-06-05 16:55:45.808 INFO mapreduce.Job: Running job: job_1654419132285_0003
2022-06-05 16:55:49.889 INFO mapreduce.Job: Job job_1654419132285_0003 running in uber mode : false
2022-06-05 16:55:49.890 INFO mapreduce.Job: map 0% reduce 0%
2022-06-05 16:55:49.940 INFO mapreduce.Job: map 100% reduce 0%
2022-06-05 16:55:59.005 INFO mapreduce.Job: map 100% reduce 100%
2022-06-05 16:55:59.021 INFO mapreduce.Job: Job job_1654419132285_0003 completed successfully
2022-06-05 16:55:59.085 INFO mapreduce.Job: Counters: 53
  File System Counters
    FILE: Number of bytes read=316
    FILE: Number of bytes written=437155
```

result:

The screenshot shows the Hadoop Web UI interface. On the left, there's a sidebar titled "Browse Directories" showing a directory structure with "/output3" selected. The main content area has a title "File information - part-r-00000". It contains several tabs: "Download", "Head the file (first 32K)", and "Tail the file (last 32K)". Below these tabs, there's a "Block information" section with a dropdown menu set to "Block 0". The "Block 0" section displays the following details:

- Block ID: 1073743249
- Block Pool ID: BP-1748348618-192.168.10.102-165174936300
- Generation Stamp: 2428
- Size: 9
- Availability: (Listed as SUCCESS)

On the right side of the main content area, there's a table titled "File contents" showing two rows of data:

	1994	8452	
8452	4914	7157	3312
6233	8452	1956	4914
8452	7311	6233	7314
1994	7514		

5.2 PageRank

5.2.1 Setup and Code Analysis

We can state the Page Rank algorithm in a more general way. We want to assign scores so that

$$x_i = \sum_{j \in L_i} \frac{x_j}{n_j}$$

where L_i are the indices such that v_j links to v_i if $j \in L_i$, and n_j is equal to outdegree of v_j . Note that L_i contains $i \deg(v_i)$ elements. The set L_i is called the set of *backlinks* of vertex v_i . This can be rewritten as a vector equation

$$x = Ax,$$

where A is the matrix $A = (a_{ij})$ given by

$$a_{ij} = \begin{cases} \frac{1}{n_j} & \text{if } j \in L_i \\ 0 & \text{otherwise} \end{cases}.$$

This matrix is called the *link matrix*. We note that in the example, the matrix A was

$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}.$$

The problem of solving for the scores x then amounts to finding an eigenvector with eigenvalue 1 for the matrix A .

We can consider the link matrix as giving the probabilities of traversing a link from the page represented by the column to the page representing the row. Thus it makes sense that the sum of the values of the columns are equal to one.

The basic idea is that we can try to compute an eigenvector iteratively like

$$x_{k+1} = Mx_k = M^k x_0.$$

Certainly, if $Mx_0 = x_0$, then this procedure fixes x_0 . In general, if we replace this method with

$$x_{k+1} = \frac{Mx_k}{\|Mx_k\|}$$

for any vector norm, we will generally find an eigenvector for the largest eigenvalue.

In this research project, for each graph, we list the top ten nodes in order of descending PageRank value. In addition, list a summary of each graph property, e.g., number of nodes, number of edges, and (min, max, avg) of out-degree for each node; list the total number of MapReduce iterations for convergence.

The MapReduce method for PageRank we use is shown below:

Map:

- Input:
 - key: index.html
 - value: <pagerank> 1.html 2.html...
- Output for each outlink:
 - key: "1.html"
 - value: "index.html <pagerank> <number of outlinks>"

Reduce

- Input:
 - Key: "1.html"
 - Value: "index.html 0.5 23"
 - Value: "2.html 2.4 2"
 - Value: ...
- Output:
 - Key: "1.html"
 - Value: "<new pagerank> index.html 2.html..."

When the ranks converge, we use also MapReduce to sort the ranks and outputs only the top 10 nodes. The MapReduce method is shown below: Reduce is simply outputting them. After that, the graph information is output using another simple MapReduce.

1. preprocess the dataset "Driver" class.
2. iteratively rank the pages until they converge, "PageRank" class.
3. sort the ranks and outputs only the top 10 nodes, "SortRank" class.
4. output graph information, "AddInfo" class.

1. Preprocess

In this phase, dataset will be transformed to target format, using simple mapreduce operation.

Mapper phase does not do any thing, reducer phase will gather all the adjacent nodes into one simple Text, in the format of following.

```
**input:**  
“ [src]<tab>[dst] “  
**output:**
```

Map:

- Input:
 - Key: "index.html"
 - Value: "<pagerank> <outlinks>"
- Output:
 - Key: "<pagerank>"
 - Value: "index.html"

““ <[node], [adjacent node]<tab>[adjacent node]...> ““
code:

```
1  protected void map(LongWritable key , Text value , Context
2      context)
3          throws IOException , InterruptedException {
4
5      context.write(new Text(key.toString()),value);
6
7  protected void reduce(Text key , Iterable<Text> values ,
8      Context
9          context) throws IOException , InterruptedException
10         {
11             StringBuilder v= new StringBuilder();
12             for (Text value : values) {
13                 v.append(value).append('\t');
14             }
15             context.write(key,new Text(v.toString()));
16         }
17 }
```

2. PageRank

The PageRank.java will use MapReduce to iteratively rank the pages until they converge.
How many iterations are needed to reach convergence

For me, convergence is defined as **1) the top 10 ranks don't change for 3 consecutive iterations and 2) The rank values don't change more than 0.01 in 3 consecutive iterations.**

For current dataset, it takes 18 iterations to converge.

map phase

input:

““ same as the output of preprocessing phase ““

output:

“<[node], info [degree] [adjacent node] [adjacent node]...> <[adjacent node], [node]
[rank] [degree]> rank = 1 if first time <[adjacent node], [node] [rank] [degree]> same
as up ... <[node], info 0> if node alone “

code:

```
1  public void map(LongWritable key, Text value, OutputCollector
2   <Text, Text> output, Reporter reporter)
3   throws IOException {
4
5   String line = value.toString();
6   String[] parts = line.split("[\t]");
7
8   thisLink.set(parts[0]);
9   double rank = 0.0;
10  int degree = 0;
11  if (parts.length >= 2) {
12      if (parts[1].length() < 10) { // A rank b c...
13          rank = Double.parseDouble(parts[1]);
14          degree = parts.length - 2;
15
16          StringBuilder builder = new StringBuilder();
17          if (parts.length >= 3) {
18              for (int i = 2; i < parts.length; i++)
19                  {
20                      builder.append(parts[i]).
21                         append(" ");
22                  }
23
24          outputValue.set("info " + degree + " "
25                          + builder);
26          output.collect(thisLink, outputValue);
27
28          outputValue.set(thisLink + " "
29                          + rank + " "
30                          + degree);
31          if (parts.length >= 3) {
32              for (int i = 2; i < parts.length; i++)
33                  {
34                      outputKey.set(parts[i]);
35                      output.collect(outputKey,
36                                     outputValue);
37                      reporter.incrCounter(Counters
38                                     .INPUT_WORDS, 1);
39                  }
40          }
41      }
42  }
43  else { // A (no rank) b c ...
44      rank = 1.0;
45      degree = parts.length - 1;
46  }
```

```

38
39     StringBuilder builder = new StringBuilder();
40     for (int i = 1; i < parts.length; i++) {
41         builder.append(parts[i]).append(" ");
42     }
43     outputValue.set("info " + degree + " "
44                     + builder);
45     output.collect(thisLink, outputValue);
46
47     outputValue.set(thisLink + " "
48                     + rank + " "
49                     + degree);
50     for (int i = 1; i < parts.length; i++) {
51         outputKey.set(parts[i]);
52         output.collect(outputKey, outputValue
53                     );
53     reporter.incrCounter(Counters.
54                         INPUT_WORDS, 1);
55 }
56 } else { // only A
57     rank = 1.0;
58
59     outputValue.set("info 0");
60     output.collect(thisLink, outputValue);
61 }
62
63 }
```

reduce phase

input:

“<[node], listinfo... [node]... info 0> the output of map phase combined together.“

output:

“<[node], [rank] [adjacent node] [adjacent node]...>“

code:

```

1 public void reduce(Text key, Iterator<Text> values,
2                   OutputCollector<Text, Text> output, Reporter reporter)
3                         throws IOException {
4
5     double sum = 0.00;
6     String outlinks = "";
7
8     while (values.hasNext()) {
9         String line = values.next().toString();
10        String[] parts = line.split("[\t]");
11
11         if (!parts[0].equals("info")) { //Not info message
12             sum += Double.parseDouble(parts[1]);
13             outlinks += parts[0] + "\t";
14         }
15     }
16
17     output.collect(key, new Text(outlinks));
18 }
```

```

12             sum += Double.parseDouble(parts[1]) / 
13             Double.parseDouble(parts[2]);
14         } else {
15             if (parts.length > 2) {
16                 for (int i = 2; i < parts.
17                     length; i++) {
18                     outlinks = outlinks.
19                     concat(parts[i] +
20                           " ");
21                 }
22             }
23             double factor = 0.85;
24             sum = factor * sum + 1 - factor;
25             sum = Math.round(sum * 10000.0) / 10000.0;
26             outputValue.set(sum + " " + outlinks);
             output.collect(key, outputValue);
         }

```

3. sortRank

sort the nodes according to its rank.

map phase

input:

““ same as the last time of the reduce phase ““

output:

““ <[rank], [node]> rank = 100 - rank, in order to find the top 10 ““

code:

```

1  public static class Map extends MapReduceBase implements
2   Mapper<LongWritable, Text, Text, Text> {
3
4      private final Text outputKey = new Text();
5      private final Text outputValue = new Text();
6
7      public void map(LongWritable key, Text value,
8                      OutputCollector<Text, Text> output,
9                      Reporter reporter)
10                     throws IOException {
11
12             String line = value.toString();
13             String[] parts = line.split("[\t]");
14             outputValue.set(parts[0]);
15             outputKey.set(String.valueOf(100.0 -
16               Double.parseDouble(parts[1])));
17             output.collect(outputKey, outputValue);
18         }

```

15 }

reduce phase

input:

““ <[rank], list[node]> ““

output:

““ <[node], [rank]> ““

code:

```
1  public void reduce(Text key, Iterator<Text> values,
2                     OutputCollector<Text, Text> output, Reporter reporter)
3                     throws IOException {
4             while (values.hasNext()) {
5                 double thekey = Math.round((100.0 - Double.parseDouble(
6                     key.toString())) * 10000.0) / 10000.0;
7                 Text newkey = new Text(String.valueOf(thekey));
8                 if (numOutput < 10) { //Number of displayed records
9                     output.collect(newkey, values.next());
10                    numOutput++;
11                } else {
12                    values.next();
13                }
14            }
}
```

4. AddInfo

this class will output the related infomation about the dataset and the output.

input:

““ same as the output of preprocess ““

output:

““ Info: Number of Node: [numNodes] Number of Edges: [numEdges] Average Degree:
[numEdges/numNodes] Max Degree: [maxDegree] Min Degree: [minDegree] ““

code:

```
1  public void map(LongWritable key, Text value, OutputCollector
2                  <Text, Text> output, Reporter reporter)
3                  throws IOException {
4
5                 String line = value.toString();
6                 String[] parts = line.split("[ \t]");
7
8                 int degree = parts.length - 1;
9
10                StringBuilder builder = new StringBuilder();
11                if (parts.length >= 3) {
12                    for (int i = 2; i < parts.length; i++) {
13                        builder.append(parts[i]).append(" ");
14                    }
15                }
16                output.collect(key, builder.toString());
17            }
18        }
19    }
20 }
```

```

13         }
14     }
15     outputKey.set("Info:");
16     outputValue.set(String.valueOf(degree));
17     output.collect(outputKey, outputValue);
18 }

19
20 public void reduce(Text key, Iterator<Text> values,
21   OutputCollector<Text, Text> output, Reporter reporter)
22   throws IOException {
23   while (values.hasNext()) { // key.toString().equals("Info:")
24     String line = values.next().toString();
25     String[] parts = line.split("[ \t]");
26     numNodes++;
27
28     int degree = Integer.parseInt(parts[0]);
29     numEdges += degree;
30     if (degree < minDegree)
31       minDegree = degree;
32     if (degree > maxDegree)
33       maxDegree = degree;
34   }
35   if (key.toString().equals("Info")){
36     Text out = new Text();
37     out.set("\nNumber of Node: "+numNodes+
38           "\nNumber of Edges: "+ numEdges +
39           "\nAverage Degree: " + numEdges/numNodes +
40           "\nMax Degree: "+ maxDegree +
41           "\nMin Degree: "+ minDegree);
42     output.collect(key, out);
43   } else {
44     output.collect(key, values.next());
45   }
46 }
```

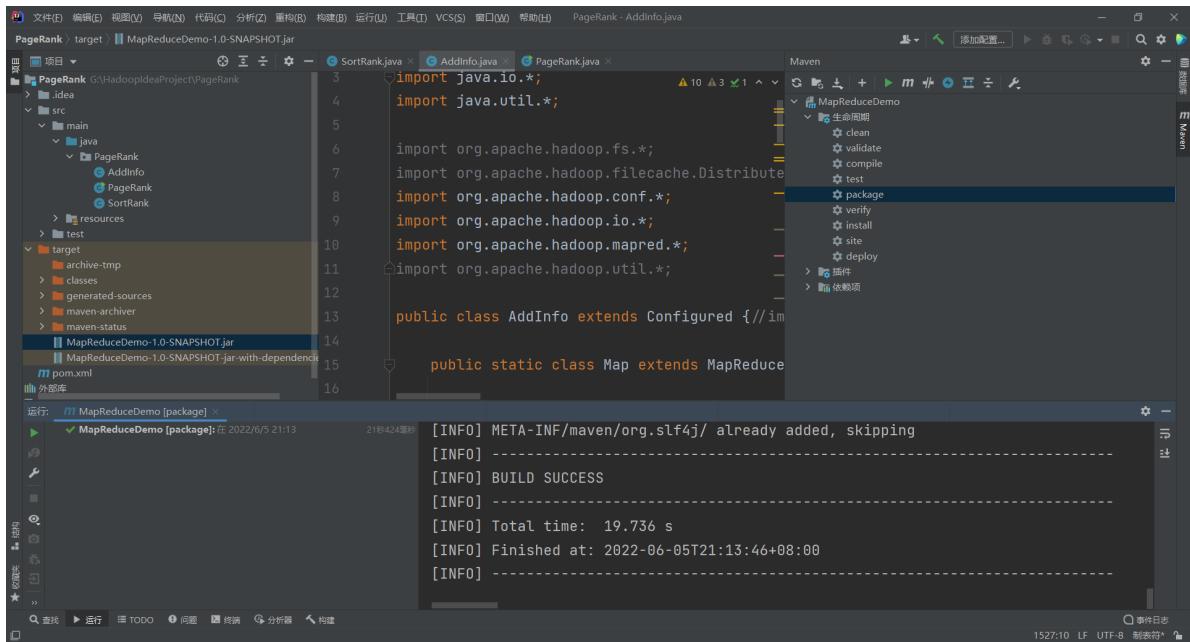
5.2.2 results

1. preprocess the dataset

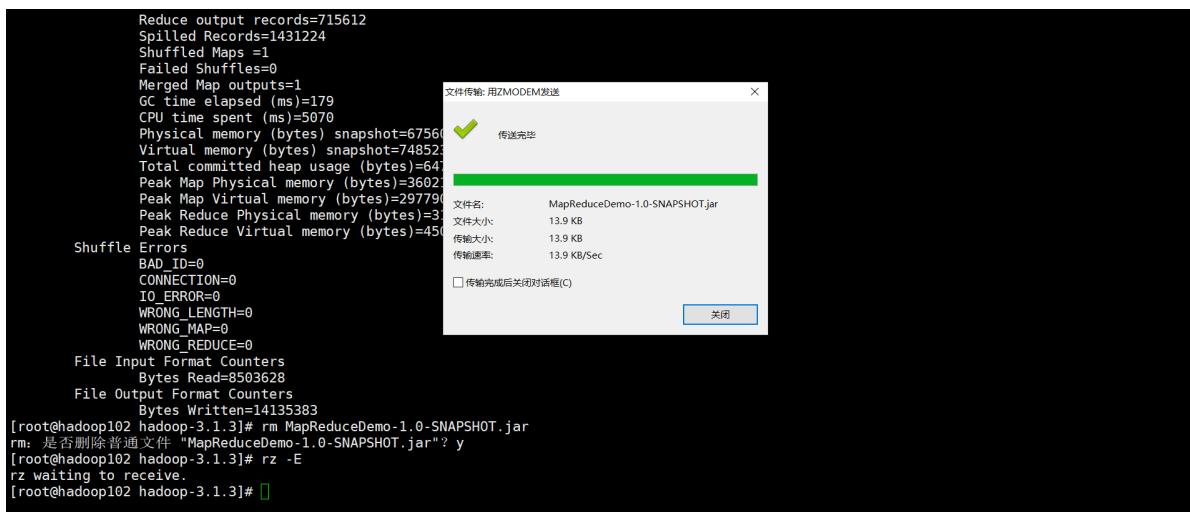
```
[root@hadoop102 hadoop-3.1.3]# hadoop jar MapReduceDemo-1.0-SNAPSHOT.jar example.org.mapreduce.lab.driver /input/PP-Decagon_ppi.csv /inputTask2
2022-06-05 21:22:06,495 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 21:22:06,884 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2022-06-05 21:22:06,897 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654434395401_0002
2022-06-05 21:22:06,970 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:22:07,074 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 21:22:07,107 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:22:07,147 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:22:07,168 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-05 21:22:07,264 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:22:07,288 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1654434395401_0002
2022-06-05 21:22:07,289 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-06-05 21:22:07,434 INFO conf.Configuration: resource-types.xml not found
2022-06-05 21:22:07,435 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-06-05 21:22:07,891 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0002
2022-06-05 21:22:07,953 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654434395401_0002/
2022-06-05 21:22:07,953 INFO mapreduce.Job: Running job: job_1654434395401_0002
```

The screenshot shows the Hadoop Web UI interface. The main navigation bar includes links for Overview, Datanodes, DataNode Volume Failures, Snapshot, Startup Progress, and Utilities. A sub-menu is open under Utilities, showing 'File information - part-r-00000'. This sub-menu has tabs for Download, Head the file (first 32K), and Tail the file (last 32K). The 'Block information' tab is selected, showing details for Block 0: Block ID 1073744170, Block Pool ID BP-1748348618-192.168.10.102-1651749363600, Generation Stamp 3349, Size 14135383, and Availability (hadoop103, hadoop104, hadoop102). The 'File contents' tab shows the first few lines of the file, which are numerical values. The right side of the screen lists all blocks for the file, showing Block Size (8 MB), Name (_SUCCESS and part-r-00000), and a trash bin icon for each.

2. compile program to Jar file



3. load Jar file to VM



4. run program in VM

```
[root@hadoop102 hadoop-3.1.3]# hadoop jar MapReduceDemo-1.0-SNAPSHOT.jar PageRank.PageRank /inputTask2/part-r-00000 /output
2022-06-05 21:24:29,395 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 21:24:29,515 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 21:24:29,769 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2022-06-05 21:24:29,786 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654434395401_0003
2022-06-05 21:24:29,848 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:24:29,947 INFO mapred.FileInputFormat: Total input files to process : 1
2022-06-05 21:24:29,972 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:24:30,008 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:24:30,029 INFO mapreduce.JobsSubmitter: number of splits:2
2022-06-05 21:24:30,121 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 21:24:30,146 INFO mapreduce.JobsSubmitter: Submitting tokens for job: job_1654434395401_0003
2022-06-05 21:24:30,146 INFO mapreduce.JobsSubmitter: Executing with tokens: []
2022-06-05 21:24:30,275 INFO conf.Configuration: resource-types.xml not found
2022-06-05 21:24:30,318 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0003
2022-06-05 21:24:30,343 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654434395401_0003/
2022-06-05 21:24:30,344 INFO mapreduce.Job: Running job: job_1654434395401_0003
```

****result:****

File information - part-00000

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073744373
Block Pool ID: BP-1748348618-192.168.10.102-1651749363600
Generation Stamp: 3552
Size: 164
Availability:

- hadoop103
- hadoop102
- hadoop104

File contents

0	114787	375519
1000006	6232	6222
1000017	6232	6223
1000028	6232	6227
100003	84126	4171
1000039	6232	6224
1000050	6232	55173
1000062	6232	6228

File information - part-00000

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073744385
Block Pool ID: BP-1748348618-192.168.10.102-1651749363600
Generation Stamp: 3564
Size: 101
Availability:

- hadoop103
- hadoop104
- hadoop102

File contents

Info:
Number of Node: 715612
Number of Edges: 1431224
Average Degree: 2
Max Degree: 2
Min Degree: 2

5.3 Graph Connected Components

5.3.1 Setup

Given a graph, this algorithm identifies the connected components. In the following, a connected component will be called as "cluster". We have tried to implement the "The Alternating Algorithm" proposed in the paper "Connected Components in MapReduce and Beyond: <https://dl.acm.org/doi/10.1145/2670979.2670997>". Below, we have reported the pseudo-code of the algorithm.

Algorithm 1 The Alternating Algorithm

INPUT: Edges (u, v) as a set of key-value pairs $\langle u; v \rangle$.

INPUT: A unique label ℓ_v for every node $v \in V$.

```

1: repeat
2:   Large-Star
3:   Small-Star
4: until Convergence

```

Algorithm 2 The Large-Star operation

```

1: procedure MAP(  $u; v$  )
2:   Emit  $\langle u; v \rangle$ 
3:   Emit  $\langle v; u \rangle$ 
4: end procedure
5: procedure REDUCE(  $u; \Gamma(u)$  )
6:    $m \leftarrow \arg \min_{v \in \Gamma^+(u)} \ell_v$ 
7:   Emit  $\langle v; m \rangle \forall v \text{ where } \ell_v > \ell_u$ 
8: end procedure

```

Algorithm 3 The Small-Star operation

```

1: procedure MAP(  $u; v$  )
2:   if  $\ell_v \leq \ell_u$  then
3:     Emit  $\langle u; v \rangle$ 
4:   else
5:     Emit  $\langle v; u \rangle$ 
6:   end if
7: end procedure
8: procedure REDUCE(  $u; N \subseteq \Gamma(u)$  )
9:    $m \leftarrow \arg \min_{v \in N \cup \{u\}} \ell_v$ 
10:  Emit  $\langle v; m \rangle \forall v \in N$ 
11: end procedure

```

1. Initialization, to transform data into targeting format.
2. Set lists of comparable class in order to sort custom-made key.
3. Perform large star and small star operations.
4. Terminate when convergence
5. Termination, gather nodes of each connected components.
6. check if duplicated.
7. translate cluster type to text type.

5.3.2 Implementation

Below, it is shown a demonstration of usage of the `ConnectedComponents` class that allows us to run the connected component algorithm on graph. This code simply creates a

ConnectedComponents object, taking as input the graph and the output folder. Invoking the run method, the algorithm will produce an hdfs file for each Reducer task, invoked by MapReduce framework, into the output folder. These files will contain the clusters found in the input graph.

```

1 package app;
2 import pad.ConnectedComponents;
3 public class App
4 {
5     public static void main( String[] args ) throws
6         Exception
7     {
8         // Create ConnectedComponents object giving an
9         // adjacency/cluster
10        // list rappresenting the graph and the output folder as input
11        ConnectedComponents cc = new
12            ConnectedComponents( new Path("graph.txt")
13                , new Path("out") );
14
15        // Run all the Jobs necessary to compute the result
16        if ( !cc.run() )
17            System.exit( 1 );
18        System.exit( 0 );
19    }
20 }
```

5.3.2.1 Input

The input file can contain:

- the adjacency list of the graph, i.e. multiple lines in the following format:
 $<NodeID><TAB><Neighborhood>$
 $<NodeID>$: is a unique integer ID corresponding to an unique node of the graph.
 $<Neighborhood>$: is a comma separated list of increasing unique IDs corresponding to the nodes of the graph that are linked to $<NodeID>$.
- the clique list of the graph, i.e. multiple lines in the following format:
 $<NodeID1><SPACE><NodeID2><SPACE> ... <SPACE><NodeIDN>$
indicating that all the nodes, $<NodeID1> ... <NodeIDN>$, are linked to each other.

5.3.2.2 Output

The program will create a folder where the clusters found are stored in a star list format. In particular the output files produced by the Reducer tasks are formatted by the *SequenceFileOutputFormat* $<pad.ClusterWritable, org.apache.hadoop.io.NullWritable>$,

where `pad.ClusterWritable` represents an array of integers and is serialized writing on the output file its size and then its elements. Therefore, each output file has the following format:

$<Cluster1><Cluster2>\dots<ClusterK>$

where each Cluster is a sequence of increasing numbers which they compound a connected component of the input graph.

5.3.2.3 Design

Here it is shown the algorithm of our implementation of Alternating Algorithm.

Algorithm 4 Our implementation of the Alternating Algorithm

INPUT: $G = (V, E)$ represented with an adjacency/cluster list.

INPUT: A unique and positive label ℓ_v for every node $v \in V$.

- 1: Initialization_Phase
 - 2: **repeat**
 - 3: Large-Star
 - 4: Small-Star
 - 5: **until** Convergence
 - 6: Termination_Phase
 - 7: Check_Phase
-

Where:

- Initialization_Phase → Transform the adjacency/clique list in a list of edges $<NodeID><TAB><NeighborID>$
- Termination_Phase → Transform the list of edges into sets of nodes (cluster files, i.e. star list)
- Check_Phase → Verify that no clusters is malformed

When we are talking about huge graph, usually they are represented with an adjacency list or a clique list. In fact, both of these format can be represented in a more compacted way than the sparse matrix format.

Since Algorithm 1 expects as input a set of key-values pairs $<u; v>$, for what we have said above, we have decided to introduce an Initialization Phase that transforms the adjacency/cliques list into a edges list $<NodeID><TAB><NeighbourID>$. In this phase, we also capture the nodes and clique number of the input.

As you can see from Algorithm 4, after the Initialization Phase, takes place the cycle suggested by Algorithm 1 where the Large-Star and Small-Star operations are called sequentially until the graph does not reach a convergent configuration.

When we finally reach this point, we still have the edges list $<NodeID><TAB><NeighbourID>$. So we need to identify the clusters and print them in a usable format. This operation is performed during the Termination Phase. The counting of nodes and clusters numbers is performed again in order to compare these values with the initial ones.

Finally, thanks to the Check Phase, we verify that no clusters is malformed, i.e. every node belonging to a cluster cannot be present in another one, but must be unique.

1. Initialization

Thanks to this format, the nodes number is easily calculated counting the input file rows. We use InitializationMapperAdjacent.java as Mapper and zero Reducer. In the Mapper, we read a line and we split it by the <TAB> character. Then, for each adjacent node, we produce the pair <NodeID, AdjacentNodeID> only if NodeID > AdjacentNodeID. In fact, whether the graph is directed or not, with this input format, we assume that every link is an undirected link. Mainly the reason is that, in the following operations, we only need the edge direction from the larger to smaller node.

if a node has no neighbours, it is emitted the pair <NodeID, -1>. In this way, when in the following phases we recognize this information, we just bring it forward, until the Termination Phase where we emit a cluster formed only by that NodeID.

input:

““ [src]<tab>[dst] ““

output:

““ <[node], [adjacent node]> while NodeID > AdjacentNodeID <[node], -1> while node alone ““

code:

```
1 public void map( LongWritable , Text value , Context context )
2     throws IOException , InterruptedException
3     {
4         // Read line.
5         String line = value.toString();
6
7         // Increment the number of nodes, since the input file presents a
8         // new node in each line.
9         context.getCounter( UtilCounters .
10             NUM_INITIAL_NODES ).increment( 1 );
11
12         // Split the line on the tab character.
13         String [] userID_neighbourhood = line.split( " \t" );
14
15         // Extract the nodeID.
16         nodeID.set( Integer.parseInt(
17             userID_neighbourhood[0] ) );
18
19         // If the node is alone.
20         if ( userID_neighbourhood.length == 1 )
21         {
22             // NeighbourID is set to minus one, to indicate that the
23             // node is alone.
24             context.write( nodeID , MINUS_ONE );
25             return;
26         }
27
28         // Split by "," to find the list of neighbours of nodeID.
29         String [] neighbours = userID_neighbourhood
30             [1].split( "," );
```

```

25
26         // Emit the pair <nodeID, neighbourID> for each neighbours.
27         for (String neighbour : neighbours) {
28             neighbourID.set(Integer.parseInt(
29                 neighbour));
30             // only if nodeID > neighbourID
31             if (nodeID.get() > neighbourID.get())
32                 context.write(nodeID,
33                               neighbourID);
34         }
35     }

```

2. Set lists of comparable class in order to sort custom-made key.

During the implementation of the algorithm, we heavily use a technique that allows the programmer to control the order in which the values show up during the Reducer phase, i.e. the comparable interface is designed to do so. In fact in the Hadoop's implementation of MapReduce, this feature is not provided by the framework. So we need several custom-made class in order to exploit this feature.

First of all, we need a composite key that contains the necessary information to perform the sorting and we have to know how compare those composite keys during the sorting phase. NodesPairWritable.java class is a writable and comparable data structure used to wrap two nodes into a key:

- NodeID: is the natural key used for joining purposes;
 - NeighbourID: is the value for which we want to sort;
- and contains the compareTo method that use both nodes during the sorting phase.

Second, we need a custom partitioner: NodePartitioner.java. Thanks to this class, the composite key will be sent to a Reducer only considering the NodeID, i.e. a given NodeID will be sent to only one Reducer.

Finally, we have to ensure that when the Reducer read the map output records from local disk, those records are combined by NodeID. The NodeGroupingComparator.java class specify how to preform this grouping process.

The most important usage of this technique is during the Small-Star and Large-Star operations. In fact, as we can see from Algorithms 2 and 3 respectively at line 6 and 9, we need to obtain the minimum neighbour. Thanks to the secondary sort technique, the Reducer will receive for a given NodeID all its neighbours sorting in ascending order. So, it is sufficient to compare this NodeID with its first neighbour in order to identify the minimum label. Then, we use it during the Termination Phase, since we want to store the set of nodes that compose a cluster in ascending order.

Related classes:

(Not every details are shown here)

```

1  ** NodesPairWritable . java **
2
3  package GCC;
4  public class NodesPairWritable implements WritableComparable<
5      NodesPairWritable>
6  {

```

```

6   public Integer NodeID = -1;
7   public Integer NeighbourID = -1; // default to -1 if node alone.
8
9   // Deserializes the array. Read the data out in the order it is written.
10  public void readFields( DataInput in ) throws
11    IOException { ... }
12
13  // Serializes this array. Write the data out in the order it is read.
14  public void write( DataOutput out ) throws
15    IOException { ... }
16
17  // Convert the object into a string.
18  public String toString() { ... }
19
20  // Compare this object with other one of its kind.
21  // First compare NodeID. if same, compare NeighbourID.
22  // Return: 0 if identical, -1 if this smaller, 1 if this greater
23  public int compareTo( NodesPairWritable other ) { ... }
24
25  // calculate the hash code using NodeID and NeighbourID
26  public int hashCode() { ... }
27
28  // Check if two objects are equals.
29  // Return: True if equal, False if not equal.
30  public boolean equals( Object other ) { ... }
31
32  }
33
34  /**
35   * Java code starts here
36   */
37
38  public class NodePartitioner extends Partitioner<
39    NodesPairWritable , IntWritable >
40  {
41    // Choose the Reducer identifier to which send the record using only the
42    // NodeID information
43    public int getPartition( NodesPairWritable pair ,
44      IntWritable _, int numPartitions ){
45      return pair.NodeID % numPartitions ;
46    }
47
48  /**
49   * Java code ends here
50   */
51
52  /**
53   * Java code starts here
54   */
55
56  public class NodeGroupingComparator extends WritableComparator {
57    // ...
58  }
59
60  /**
61   * Java code ends here
62   */
63
64  /**
65   * Java code starts here
66   */
67
68  public class NodeGroupingKey extends WritableComparable {
69    // ...
70  }
71
72  /**
73   * Java code ends here
74   */
75
76  /**
77   * Java code starts here
78   */
79
80  public class NodeGroupingValue extends WritableComparable {
81    // ...
82  }
83
84  /**
85   * Java code ends here
86   */
87
88  /**
89   * Java code starts here
90   */
91
92  public class NodeGroupingKeyAndValue extends WritableComparable {
93    // ...
94  }
95
96  /**
97   * Java code ends here
98   */
99
100 /**
101  * Java code starts here
102 */
103
104 /**
105  * Java code ends here
106 */
107
108 /**
109  * Java code starts here
110 */
111
112 /**
113  * Java code ends here
114 */
115
116 /**
117  * Java code starts here
118 */
119
120 /**
121  * Java code ends here
122 */
123
124 /**
125  * Java code starts here
126 */
127
128 /**
129  * Java code ends here
130 */
131
132 /**
133  * Java code starts here
134 */
135
136 /**
137  * Java code ends here
138 */
139
140 /**
141  * Java code starts here
142 */
143
144 /**
145  * Java code ends here
146 */
147
148 /**
149  * Java code starts here
150 */
151
152 /**
153  * Java code ends here
154 */
155
156 /**
157  * Java code starts here
158 */
159
160 /**
161  * Java code ends here
162 */
163
164 /**
165  * Java code starts here
166 */
167
168 /**
169  * Java code ends here
170 */
171
172 /**
173  * Java code starts here
174 */
175
176 /**
177  * Java code ends here
178 */
179
180 /**
181  * Java code starts here
182 */
183
184 /**
185  * Java code ends here
186 */
187
188 /**
189  * Java code starts here
190 */
191
192 /**
193  * Java code ends here
194 */
195
196 /**
197  * Java code starts here
198 */
199
200 /**
201  * Java code ends here
202 */
203
204 /**
205  * Java code starts here
206 */
207
208 /**
209  * Java code ends here
210 */
211
212 /**
213  * Java code starts here
214 */
215
216 /**
217  * Java code ends here
218 */
219
220 /**
221  * Java code starts here
222 */
223
224 /**
225  * Java code ends here
226 */
227
228 /**
229  * Java code starts here
230 */
231
232 /**
233  * Java code ends here
234 */
235
236 /**
237  * Java code starts here
238 */
239
240 /**
241  * Java code ends here
242 */
243
244 /**
245  * Java code starts here
246 */
247
248 /**
249  * Java code ends here
250 */
251
252 /**
253  * Java code starts here
254 */
255
256 /**
257  * Java code ends here
258 */
259
260 /**
261  * Java code starts here
262 */
263
264 /**
265  * Java code ends here
266 */
267
268 /**
269  * Java code starts here
270 */
271
272 /**
273  * Java code ends here
274 */
275
276 /**
277  * Java code starts here
278 */
279
280 /**
281  * Java code ends here
282 */
283
284 /**
285  * Java code starts here
286 */
287
288 /**
289  * Java code ends here
290 */
291
292 /**
293  * Java code starts here
294 */
295
296 /**
297  * Java code ends here
298 */
299
300 /**
301  * Java code starts here
302 */
303
304 /**
305  * Java code ends here
306 */
307
308 /**
309  * Java code starts here
310 */
311
312 /**
313  * Java code ends here
314 */
315
316 /**
317  * Java code starts here
318 */
319
320 /**
321  * Java code ends here
322 */
323
324 /**
325  * Java code starts here
326 */
327
328 /**
329  * Java code ends here
330 */
331
332 /**
333  * Java code starts here
334 */
335
336 /**
337  * Java code ends here
338 */
339
340 /**
341  * Java code starts here
342 */
343
344 /**
345  * Java code ends here
346 */
347
348 /**
349  * Java code starts here
350 */
351
352 /**
353  * Java code ends here
354 */
355
356 /**
357  * Java code starts here
358 */
359
360 /**
359  * Java code ends here
360 */
361
362 /**
363  * Java code starts here
364 */
365
366 /**
367  * Java code ends here
368 */
369
370 /**
371  * Java code starts here
372 */
373
374 /**
375  * Java code ends here
376 */
377
378 /**
379  * Java code starts here
380 */
381
382 /**
383  * Java code ends here
384 */
385
386 /**
387  * Java code starts here
388 */
389
390 /**
389  * Java code ends here
390 */
391
392 /**
393  * Java code starts here
394 */
395
396 /**
397  * Java code ends here
398 */
399
400 /**
401  * Java code starts here
402 */
403
404 /**
405  * Java code ends here
406 */
407
408 /**
409  * Java code starts here
410 */
411
412 /**
413  * Java code ends here
414 */
415
416 /**
417  * Java code starts here
418 */
419
420 /**
421  * Java code ends here
422 */
423
424 /**
425  * Java code starts here
426 */
427
428 /**
429  * Java code ends here
430 */
431
432 /**
433  * Java code starts here
434 */
435
436 /**
437  * Java code ends here
438 */
439
440 /**
441  * Java code starts here
442 */
443
444 /**
445  * Java code ends here
446 */
447
448 /**
449  * Java code starts here
450 */
451
452 /**
453  * Java code ends here
454 */
455
456 /**
457  * Java code starts here
458 */
459
460 /**
459  * Java code ends here
460 */
461
462 /**
463  * Java code starts here
464 */
465
466 /**
467  * Java code ends here
468 */
469
470 /**
471  * Java code starts here
472 */
473
474 /**
475  * Java code ends here
476 */
477
478 /**
479  * Java code starts here
480 */
481
482 /**
483  * Java code ends here
484 */
485
486 /**
487  * Java code starts here
488 */
489
490 /**
489  * Java code ends here
490 */
491
492 /**
493  * Java code starts here
494 */
495
496 /**
497  * Java code ends here
498 */
499
500 /**
501  * Java code starts here
502 */
503
504 /**
505  * Java code ends here
506 */
507
508 /**
509  * Java code starts here
510 */
511
512 /**
513  * Java code ends here
514 */
515
516 /**
517  * Java code starts here
518 */
519
520 /**
519  * Java code ends here
520 */
521
522 /**
523  * Java code starts here
524 */
525
526 /**
527  * Java code ends here
528 */
529
530 /**
531  * Java code starts here
532 */
533
534 /**
535  * Java code ends here
536 */
537
538 /**
539  * Java code starts here
540 */
541
542 /**
543  * Java code ends here
544 */
545
546 /**
547  * Java code starts here
548 */
549
550 /**
549  * Java code ends here
550 */
551
552 /**
553  * Java code starts here
554 */
555
556 /**
557  * Java code ends here
558 */
559
560 /**
561  * Java code starts here
562 */
563
564 /**
565  * Java code ends here
566 */
567
568 /**
569  * Java code starts here
570 */
571
572 /**
573  * Java code ends here
574 */
575
576 /**
577  * Java code starts here
578 */
579
580 /**
579  * Java code ends here
580 */
581
582 /**
583  * Java code starts here
584 */
585
586 /**
587  * Java code ends here
588 */
589
590 /**
591  * Java code starts here
592 */
593
594 /**
595  * Java code ends here
596 */
597
598 /**
599  * Java code starts here
600 */
601
602 /**
603  * Java code ends here
604 */
605
606 /**
607  * Java code starts here
608 */
609
610 /**
609  * Java code ends here
610 */
611
612 /**
613  * Java code starts here
614 */
615
616 /**
617  * Java code ends here
618 */
619
620 /**
621  * Java code starts here
622 */
623
624 /**
625  * Java code ends here
626 */
627
628 /**
629  * Java code starts here
630 */
631
632 /**
633  * Java code ends here
634 */
635
636 /**
637  * Java code starts here
638 */
639
640 /**
639  * Java code ends here
640 */
641
642 /**
643  * Java code starts here
644 */
645
646 /**
647  * Java code ends here
648 */
649
650 /**
651  * Java code starts here
652 */
653
654 /**
655  * Java code ends here
656 */
657
658 /**
659  * Java code starts here
660 */
661
662 /**
663  * Java code ends here
664 */
665
666 /**
667  * Java code starts here
668 */
669
670 /**
669  * Java code ends here
670 */
671
672 /**
673  * Java code starts here
674 */
675
676 /**
677  * Java code ends here
678 */
679
680 /**
681  * Java code starts here
682 */
683
684 /**
685  * Java code ends here
686 */
687
688 /**
689  * Java code starts here
690 */
691
692 /**
693  * Java code ends here
694 */
695
696 /**
697  * Java code starts here
698 */
699
699 /**
699  * Java code ends here
700 */
700
701 /**
702  * Java code starts here
703 */
704
705 /**
706  * Java code ends here
707 */
708
709 /**
709  * Java code starts here
710 */
711
710 /**
711  * Java code ends here
712 */
713
714 /**
715  * Java code starts here
716 */
717
718 /**
719  * Java code ends here
720 */
721
722 /**
723  * Java code starts here
724 */
725
726 /**
727  * Java code ends here
728 */
729
730 /**
731  * Java code starts here
732 */
733
734 /**
735  * Java code ends here
736 */
737
738 /**
739  * Java code starts here
740 */
741
742 /**
743  * Java code ends here
744 */
745
746 /**
747  * Java code starts here
748 */
749
749 /**
749  * Java code ends here
750 */
750
751 /**
752  * Java code starts here
753 */
754
755 /**
756  * Java code ends here
757 */
758
759 /**
759  * Java code starts here
760 */
761
760 /**
761  * Java code ends here
762 */
763
764 /**
765  * Java code starts here
766 */
767
768 /**
769  * Java code ends here
770 */
771
772 /**
773  * Java code starts here
774 */
775
776 /**
777  * Java code ends here
778 */
779
779 /**
779  * Java code starts here
780 */
781
780 /**
781  * Java code ends here
782 */
783
784 /**
785  * Java code starts here
786 */
787
788 /**
789  * Java code ends here
790 */
791
792 /**
793  * Java code starts here
794 */
795
795 /**
795  * Java code ends here
796 */
796
797 /**
798  * Java code starts here
799 */
800
801 /**
802  * Java code ends here
803 */
804
805 /**
806  * Java code starts here
807 */
808
809 /**
809  * Java code ends here
810 */
811
810 /**
811  * Java code starts here
812 */
813
814 /**
815  * Java code ends here
816 */
817
818 /**
819  * Java code starts here
820 */
821
822 /**
823  * Java code ends here
824 */
825
826 /**
827  * Java code starts here
828 */
829
829 /**
829  * Java code ends here
830 */
830
831 /**
832  * Java code starts here
833 */
834
835 /**
836  * Java code ends here
837 */
838
839 /**
839  * Java code starts here
840 */
841
840 /**
841  * Java code ends here
842 */
843
844 /**
845  * Java code starts here
846 */
847
848 /**
849  * Java code ends here
850 */
851
852 /**
853  * Java code starts here
854 */
855
856 /**
857  * Java code ends here
858 */
859
859 /**
859  * Java code starts here
860 */
860
861 /**
862  * Java code starts here
863 */
864
865 /**
866  * Java code ends here
867 */
868
869 /**
869  * Java code starts here
870 */
871
870 /**
871  * Java code ends here
872 */
873
874 /**
875  * Java code starts here
876 */
877
878 /**
879  * Java code ends here
880 */
881
882 /**
883  * Java code starts here
884 */
885
886 /**
887  * Java code ends here
888 */
889
889 /**
889  * Java code starts here
890 */
891
890 /**
891  * Java code ends here
892 */
893
894 /**
895  * Java code starts here
896 */
897
898 /**
899  * Java code ends here
900 */
901
902 /**
903  * Java code starts here
904 */
905
906 /**
907  * Java code ends here
908 */
909
910 /**
911  * Java code starts here
912 */
913
914 /**
915  * Java code ends here
916 */
917
918 /**
919  * Java code starts here
920 */
921
922 /**
923  * Java code ends here
924 */
925
926 /**
927  * Java code starts here
928 */
929
929 /**
929  * Java code ends here
930 */
930
931 /**
932  * Java code starts here
933 */
934
935 /**
936  * Java code ends here
937 */
938
939 /**
939  * Java code starts here
940 */
941
940 /**
941  * Java code ends here
942 */
943
944 /**
945  * Java code starts here
946 */
947
948 /**
949  * Java code ends here
950 */
951
952 /**
953  * Java code starts here
954 */
955
956 /**
957  * Java code ends here
958 */
959
959 /**
959  * Java code starts here
960 */
960
961 /**
962  * Java code starts here
963 */
964
965 /**
966  * Java code ends here
967 */
968
969 /**
969  * Java code starts here
970 */
971
970 /**
971  * Java code ends here
972 */
973
974 /**
975  * Java code starts here
976 */
977
978 /**
979  * Java code ends here
980 */
981
982 /**
983  * Java code starts here
984 */
985
986 /**
987  * Java code ends here
988 */
989
989 /**
989  * Java code starts here
990 */
991
990 /**
991  * Java code ends here
992 */
993
994 /**
995  * Java code starts here
996 */
997
998 /**
999  * Java code ends here
1000 */
1001
1002 /**
1003  * Java code starts here
1004 */
1005
1006 /**
1007  * Java code ends here
1008 */
1009
1010 /**
1011  * Java code starts here
1012 */
1013
1014 /**
1015  * Java code ends here
1016 */
1017
1018 /**
1019  * Java code starts here
1020 */
1021
1022 /**
1023  * Java code ends here
1024 */
1025
1026 /**
1027  * Java code starts here
1028 */
1029
1029 /**
1029  * Java code ends here
1030 */
1030
1031 /**
1032  * Java code starts here
1033 */
1034
1035 /**
1036  * Java code ends here
1037 */
1038
1039 /**
1039  * Java code starts here
1040 */
1041
1040 /**
1041  * Java code ends here
1042 */
1043
1044 /**
1045  * Java code starts here
1046 */
1047
1048 /**
1049  * Java code ends here
1050 */
1051
1052 /**
1053  * Java code starts here
1054 */
1055
1056 /**
1057  * Java code ends here
1058 */
1059
1059 /**
1059  * Java code starts here
1060 */
1060
1061 /**
1062  * Java code starts here
1063 */
1064
1065 /**
1066  * Java code ends here
1067 */
1068
1069 /**
1069  * Java code starts here
1070 */
1071
1070 /**
1071  * Java code ends here
1072 */
1073
1074 /**
1075  * Java code starts here
1076 */
1077
1078 /**
1079  * Java code ends here
1080 */
1081
1082 /**
1083  * Java code starts here
1084 */
1085
1086 /**
1087  * Java code ends here
1088 */
1089
1089 /**
1089  * Java code starts here
1090 */
1091
1090 /**
1091  * Java code ends here
1092 */
1093
1094 /**
1095  * Java code starts here
1096 */
1097
1098 /**
1099  * Java code ends here
1100 */
1101
1102 /**
1103  * Java code starts here
1104 */
1105
1106 /**
1107  * Java code ends here
1108 */
1109
1110 /**
1111  * Java code starts here
1112 */
1113
1114 /**
1115  * Java code ends here
1116 */
1117
1118 /**
1119  * Java code starts here
1120 */
1121
1122 /**
1123  * Java code ends here
1124 */
1125
1126 /**
1127  * Java code starts here
1128 */
1129
1129 /**
1129  * Java code ends here
1130 */
1130
1131 /**
1132  * Java code starts here
1133 */
1134
1135 /**
1136  * Java code ends here
1137 */
1138
1139 /**
1139  * Java code starts here
1140 */
1141
1140 /**
1141  * Java code ends here
1142 */
1143
1144 /**
1145  * Java code starts here
1146 */
1147
1148 /**
1149  * Java code ends here
1150 */
1151
1152 /**
1153  * Java code starts here
1154 */
1155
1156 /**
1157  * Java code ends here
1158 */
1159
1159 /**
1159  * Java code starts here
1160 */
1160
1161 /**
1162  * Java code starts here
1163 */
1164
1165 /**
1166  * Java code ends here
1167 */
1168
1169 /**
1169  * Java code starts here
1170 */
1171
1170 /**
1171  * Java code ends here
1172 */
1173
1174 /**
1175  * Java code starts here
1176 */
1177
1178 /**
1179  * Java code ends here
1180 */
1181
1182 /**
1183  * Java code starts here
1184 */
1185
1186 /**
1187  * Java code ends here
1188 */
1189
1189 /**
1189  * Java code starts here
1190 */
1191
1190 /**
1191  * Java code ends here
1192 */
1193
1194 /**
1195  * Java code starts here
1196 */
1197
1198 /**
1199  * Java code ends here
1200 */
1201
1202 /**
1203  * Java code starts here
1204 */
1205
1206 /**
1207  * Java code ends here
1208 */
1209
1210 /**
1211  * Java code starts here
1212 */
1213
1214 /**
1215  * Java code ends here
1216 */
1217
1218 /**
1219  * Java code starts here
1220 */
1221
1222 /**
1223  * Java code ends here
1224 */
1225
1226 /**
1227  * Java code starts here
1228 */
1229
1229 /**
1229  * Java code ends here
1230 */
1230
1231 /**
1232  * Java code starts here
1233 */
1234
1235 /**
1236  * Java code ends here
1237 */
1238
1239 /**
1239  * Java code starts here
1240 */
1241
1240 /**
1241  * Java code ends here
1242 */
1243
1244 /**
1245  * Java code starts here
1246 */
1247
1248 /**
1249  * Java code ends here
1250 */
1251
1252 /**
1253  * Java code starts here
1254 */
1255
1256 /**
1257  * Java code ends here
1258 */
1259
1259 /**
1259  * Java code starts here
1260 */
1260
1261 /**
1262  * Java code starts here
1263 */
1264
1265 /**
1266  * Java code ends here
1267 */
1268
1269 /**
1269  * Java code starts here
1270 */
1271
1270 /**
1271  * Java code ends here
1272 */
1273
1274 /**
1275  * Java code starts here
1276 */
1277
1278 /**
1279  * Java code ends here
1280 */
1281
1282 /**
1283  * Java code starts here
1284 */
1285
1286 /**
1287  * Java code ends here
1288 */
1289
1289 /**
1289  * Java code starts here
1290 */
1291
1290 /**
1291  * Java code ends here
1292 */
1293
1294 /**
1295  * Java code starts here
1296 */
1297
1298 /**
1299  * Java code ends here
1300 */
1301
1302 /**
1303  * Java code starts here
1304 */
1305
1306 /**
1307  * Java code ends here
1308 */
1309
1310 /**
1311  * Java code starts here
1312 */
1313
1314 /**
1315  * Java code ends here
1316 */
1317
1318 /**
1319  * Java code starts here
1320 */
1321
1322 /**
1323  * Java code ends here
1324 */
1325
1326 /**
1327  * Java code starts here
1328 */
1329
1329 /**
1329  * Java code ends here
1330 */
1330
1331 /**
1332  * Java code starts here
1333 */
1334
1335 /**
1336  * Java code ends here
1337 */
1338
1339 /**
1339  * Java code starts here
1340 */
1341
1340 /**
1341  * Java code ends here
1342 */
1343
1344 /**
1345  * Java code starts here
1346 */
1347
1348 /**
1349  * Java code ends here
1350 */
1351
1352 /**
1353  * Java code starts here
1354 */
1355
1356 /**
1357  * Java code ends here
1358 */
1359
1359 /**
1359  * Java code starts here
1360 */
1360
1361 /**
1362  * Java code starts here
1363 */
1364
1365 /**
1366  * Java code ends here
1367 */
1368
1369 /**
1369  * Java code starts here
1370 */
1371
1370 /**
1371  * Java code ends here
1372 */
1373
1374 /**
1375  * Java code starts here
1376 */
1377
1378 /**
1379  * Java code ends here
1380 */
1381
1382 /**
1383  * Java code starts here
1384 */
1385
1386 /**
1387  * Java code ends here
1388 */
1389
1389 /**
1389  * Java code starts here
1390 */
1391
1390 /**
1391  * Java code ends here
1392 */
1393
1394 /**
1395  * Java code starts here
1396 */
1397
1398 /**
1399  * Java code ends here
1400 */
1401
1402 /**
1403  * Java code starts here
1404 */
1405
1406 /**
1407  * Java code ends here
1408 */
1409
1410 /**
1411  * Java code starts here
1412 */
1413
1414 /**
1415  * Java code ends here
1416 */
1417
1418 /**
1419  * Java code starts here
1420 */
1421
1422 /**
1423  * Java code ends here
1424 */
1425
1426 /**
1427  * Java code starts here
1428 */
1429
1429 /**
1429  * Java code ends here
1430 */
1430
1431 /**
1432  * Java code starts here
1433 */
1434
1435 /**
1436  * Java code ends here
1437 */
1438
1439 /**
1439  * Java code starts here
1440 */
1441
1440 /**
1441  * Java code ends here
1442 */
1443
1444 /**
1445  * Java code starts here
1446 */
1447
1448 /**
1449  * Java code ends here
1450 */
1451
1452 /**
1453  * Java code starts here
1454 */
1455
1456 /**
1457  * Java code ends here
1458 */
1459
1459 /**
1459  * Java code starts here
1460 */
1460
1461 /**
1462  * Java code starts here
1463 */
1464
1465 /**
1466  * Java code ends here
1467 */
1468
1469 /**
1469  * Java code starts here
1470 */
1471
1470 /**
1471  * Java code ends here
1472 */
1473
1474 /**
1475  * Java code starts here
1476 */
1477
1478 /**
1479  * Java code ends here
1480 */
1481
1482 /**
1483  * Java code starts here
1484 */
1485
1486 /**
1487  * Java code ends here
1488 */
1489
1489 /**
1489  * Java code starts here
1490 */
1491
1490 /**
1491  * Java code ends here
1492 */
1493
1494 /**
1495  * Java code starts here
1496 */
1497
1498 /**
1499  * Java code ends here
1500 */
1501
1502 /**
1503  * Java code starts here
1504 */
1505
1506 /**
1507  * Java code ends here
1508 */
1509
1510 /**
1511  * Java code starts here
1512 */
1513
1514 /**
1515  * Java code ends here
1516 */
1517
1518 /**
1519  * Java code starts here
1520 */
1521
1522 /**
1523  * Java code ends here
1524 */
1525
1526 /**
1527  * Java code starts here
1528 */
1529
1529 /**
1529  * Java code ends here
1530 */
1530
1531 /**
1532  * Java code starts here
1533 */
1534
1535 /**
1536  * Java code ends here
1537 */
1538
1539 /**
1539  * Java code starts here
1540 */
1541
1540 /**
1541  * Java code ends here
1542 */
1543
1544 /**
1545  * Java code starts here
1546 */
1547
1548 /**
1549  * Java code ends here
1550 */
1551
1552 /**
1553  * Java code starts here
1554 */
1555
1556 /**
1557  * Java code ends here
1558 */
1559
1559 /**
1559  * Java code starts here
1560 */
1560
1561 /**
1562  * Java code starts here
1563 */
1564
1565 /**
1566  * Java code ends here
1567 */
1568
1569 /**
1569  * Java code starts here
1570 */
1571
1570 /**
1571  * Java code ends here
1572 */
1573
1574 /**
1575  * Java code starts here
1576 */
1577
1578 /**
1579  * Java code ends here
1580 */
1581
1582 /**
1583  * Java code starts here
1584 */
1585
1586 /**
1587  * Java code ends here
1588 */
1589
1589 /**
1589  * Java code starts here
1590 */
1591
1590 /**
1591  * Java code ends here
1592 */
1593
1594 /**
1595  * Java code starts here
1596 */
1597
1598 /**
1599  * Java code ends here
1600 */
1601
1602 /**
1603  * Java code starts here
1604 */
1605
1606 /**
1607  * Java code ends here
1608 */
1609
1610 /**
1611  * Java code starts here
1612 */
1613
1614 /**
1615  * Java code ends here
1616 */
1617
1618 /**
1619  * Java code starts here
1620 */
1621
1622 /**
1623  * Java code ends here
1624 */
1625
1626 /**
1627  * Java code starts here
1628 */
1629
1629 /**
1629  * Java code ends here
1630 */
1630
1631 /**
1632  * Java code starts here
1633 */
1634
1635 /**
1636  * Java code ends here
1637 */
1638
1639 /**
1639  * Java code starts here
1640 */
1641
1640 /**
1641  * Java code ends here
1642 */
1643
1644 /**
1645  * Java code starts here
1646 */
1647
1648 /**
1649  * Java code ends here
1650 */
1651
1652 /**
1653  * Java code starts here
1654 */
1655
1656 /**
1657  * Java code ends here
1658 */
1659
1659 /**
1659  * Java code starts here
1660 */
1660
1661 /**
1662  * Java code starts here
1663 */
1664
1665 /**
1666  * Java code ends here
1667 */
1668
1669 /**
1669  * Java code starts here
1670 */
1671
1670 /**
1671  * Java code ends here
1672 */
1673
1674 /**
1675  * Java code starts here
1676 */
1677
1678 /**
1679  * Java code ends here
1680 */
1681
1682 /**
1683  * Java code starts here
1684 */
1685
1686 /**
1687  * Java code ends here
1688 */
1689
1689 /**
1689  * Java code starts here
1690 */
1691
1690 /**
1691  * Java code ends here
1692 */
1693
1694 /**
1695  * Java code starts here
1696 */
1697
1698 /**
1699  * Java code ends here
1700 */
1701
1702 /**
1703  * Java code starts here
1704 */
1705
1706 /**
1707  * Java code ends here
1708 */
1709
1710 /**
1711  * Java code starts here
1712 */
1713
1714 /**
1715  * Java code ends here
1716 */
1717
1718 /**
1719  * Java code starts here
1720 */
1721
1722 /**
1723  * Java code ends here
1724 */
1725
1726 /**
1727  * Java code starts here
1728 */
1729
1729 /**
1729  * Java code ends here
1730 */
1730
1731 /**
1732  * Java code starts here
1733 */
1734
1735 /**
1736  * Java code ends here
1737 */
1738
1739 /**
1739  * Java code starts here
1740 */
1741
1740 /**
1741  * Java code ends here
1742 */
1743
1744 /**
1745  * Java code starts here
1746 */
1747
1748 /**
1749  * Java code ends here
1750 */
1751
1752 /**
1753  * Java code starts here
1754 */
1755
1756 /**
1757  * Java code ends here
1758 */
1759
1759 /**
1759  * Java code starts here
1760 */
1760
1761 /**
1762  * Java code starts here
1763 */
1764
1765 /**
1766  * Java code ends here
1767 */
1768
1769 /**
1769  * Java code starts here
1770 */
1771
1770 /**
1771  * Java code ends here
1772 */
1773
1774 /**
1775  * Java code starts here
1776 */
1777
1778 /**
1779  * Java code ends here
1780 */
1781
1782 /**
1783  * Java code starts here
1784 */
1785
1786 /**
1787  * Java code ends here
1788 */
1789
1789 /**
1789  * Java code starts here
1790 */
1791
1790 /**
1791  * Java code ends here
1792 */
1793
1794 /**
1795  * Java code starts here
1796 */
1797
1798 /**
1799  * Java code ends here
1800 */
1801
1802 /**
1803  * Java code starts here
1804 */
1805
1806 /**
1807  * Java code ends here
1808 */
1809
1810 /**
1811  * Java code starts here
1812 */
1813
1814 /**
1815  * Java code ends here
1816 */
1817
1818 /**
1819  * Java code starts here
1
```

```

50 package GCC;
51
52 public class NodeGroupingComparator extends
53     WritableComparator
54 {
55     // Initializes a new instance of the NodeGroupingComparator class.
56     protected NodeGroupingComparator() {
57         super( NodesPairWritable.class, true );
58     }
59
60     // Compare two keys read from the mapper output records, only looking to the
61     // first component, i.e NodeID.
62     // Return: 0 if identical, -1 if this smaller, 1 if this greater
63     public int compare( WritableComparable key1,
64                         WritableComparable key2 ) { ... }
65 }
```

3. Large-star Small Star

Since the structure of two operations is very similar, we have decided to use the following java class files: StarDriver.java, StarMapper.java, StarCombiner.java, StarReducer.java and change their behaviour using an external variable, i.e. the Mapper and Reducer extract this variable during the setup and than they act accordingly depending on its value. Then, we just follow the exactly behaviour described in Algorithms 2 and 3, as you can see from the code we wrote for the Mapper and Reducer.

After a Large-Star or Small-Star operation, it is possible that some links are replicated.

For this reason, we added a Combiner phase (StarCombiner.java) that is responsible of reducing the amount of data written to local storage, merged, sorted and sent over the network.

The Hadoop framework does not ensure how many times a Combiner is called in action, but we know that, if it happens, is after the data sorting:

- after in-memory sort and before writing data to disk;
- after the merging and sorting phases.

Therefore, knowing that the data is sorted, we can exploit the secondary sort and delete the duplicates simply storing the last NeighbourId emitted and skipping all the following neighbours with the same identifier.

Related classes:

(Not every details are shown here)

```

1 ** StarDriver.java **
2
3 /**
4  * package GCC;
5  * // Driver of the Job responsible for executing the Small-Star or Large-Star operation on the
6  * // input graph.
7  * public class StarDriver extends Configured implements Tool
8  {
9      // The StarDriver can be of type Large-StarDriver or Small-StarDriver
10     public enum StarDriverType { LARGE, SMALL };
```

```

10
11     public StarDriver( StarDriverType type , Path input ,
12         Path output , int iteration , boolean verbose )
13     {
14         this .type = type ;
15         this .title = type .equals( StarDriverType .
16             SMALL ) ? "Small-Star" + iteration : "
17             Large-Star" + iteration ;
18         ...
19     }
20
21     public int run( String [] args ) throws Exception
22     {
23         //set different job accordingly.
24         conf .set( "type" , this .type .toString () );
25         Job job = new Job( conf , this .title );
26         ...
27
28         // Set up the private variable looking to the counter value
29         this .numChanges = job .getCounters () .
30             findCounter( UtilCounters .NUM_CHANGES ) .
31             getValue ();
32     }
33
34     // Return the number of changes occurred during the operation Small-Star or
35     // Large-Star.
36     public long getNumChanges (){
37         return this .numChanges ;
38     }
39
40     /**
41      * [node] , [adjacent node]
42      */
43
44     /**
45      * [node] , [adjacent node]
46      */
47
48     /**
49      * pair {[node] , [adjacent node]} , [node]
50      */
51

```

```

52
53 code:
54
55   '''java
56 package GCC;
57
58 public class StarMapper extends Mapper<IntWritable,
59   IntWritable, NodesPairWritable, IntWritable>
60 {
61     // Extract type variable from the context configuration.
62     // Based on this value, this Mapper will behave as a Small-Star Mapper or
63     // Large-Star Mapper
64     public void setup( Context context ){
65       smallStar = context.getConfiguration().get( "type" )
66         .equals( "SMALL" );
67     }
68
69     // If Large-Star, emits the pairs <u,v> and <v,u>.
70     // If Small-Star, emits the pair <max(u,v), min(u,v)>.
71     public void map( IntWritable nodeID, IntWritable
72       neighbourID, Context context ) throws IOException,
73       InterruptedException
74     {
75       // if the node is alone, emit it like is it in order to keep that
76       // information
77       if ( neighbourID.get() == -1 ){ context.write(
78         ( pair, neighbourID ) );
79
80       }
81
82     ** StarReducer.java **
83
84 input:
85
86   '''
87 <pair{[node], [adjacent node]}, list{[node]}>
88 <pair{[node], [adjacent node]}, list{[adjacent node]}>
89 # the value depends on the type of star operation
90
91 output:

```

```

93      """
94
95      <[node] , [minNode]>
96      <[adjacent node] , [minNode]>
97      # the value depends on the type of star operation and the
98      # comparation of NeighbourID and the NodeID
99
100     code:
101
102     """java
103     package GCC;
104
105     public class StarReducer extends Reducer<NodesPairWritable ,
106         IntWritable , IntWritable , IntWritable >
107     {
108         // same setup as StarMapper.java
109         public void setup( Context context )
110         {
111             smallStar = context.getConfiguration().get(
112                 "type" ).equals( "SMALL" );
113
114             /* The neighbours are sorted, thanks to the
115                NodesPairWritable.java, we know that the
116                minimum node is either the NodeID or the first neighbour. We
117                call it MinNodeID.
118                For each neighbour, we produce the pairs <NeighbourID
119                , MinNodeID> and <MinNodeID , NeighbourID> :
120                Small-Star: - always;
121                Large-Star: - only when NeighbourID is greater than
122                NodeID */
123         public void reduce( NodesPairWritable pair , Iterable<
124             IntWritable> neighbourhood , Context context )
125             throws IOException , InterruptedException
126         {
127             // This means that the nodeID is isolated, so we emit it unchanged
128             if ( pair.NeighbourID == -1 ){...}
129
130             // We know that the first element contains the neighbour node with
131             // the minimum label. We just need to compare it with the node id.
132             minNodeID.set( Math.min( pair.NodeID , pair.
133                 NeighbourID ) );
134
135             // If we are running Small-Star, we need to connect this node to
136             // the minimum neighbours
137             if ( smallStar && ( pair.NodeID != minNodeID.
138                 get() ) ){...}
139
140

```

```

129         // same impossible value
130         int lastNodeSeen = -2;
131         for ( IntWritable neighbour : neighbourhood )
132         {
133             // Skip the duplicate nodes.
134             if ( neighbour.get() == lastNodeSeen
135                 )
136                 continue;
137
138             // If we are running Small-Star, we always emit the
139             // neighbours except when it is the minNodeID
140             // If we are running Large-Star, we emit only when the
141             // neighbourID is greater than nodeID
142             boolean cond = ( smallStar ? (
143                 neighbour.get() != minNodeID.get()
144                 ) : ( neighbour.get() > pair.
145                     NodeID ) );
146             if ( cond ){ ... }
147
148             // Store the last neighbourId that we have processed.
149             lastNodeSeen = neighbour.get();
150         }
151
152         // If the NodeID has not the minimum label means that the
153         // produced pairs will be different,
154         // so we increment the number of changes by the number of
155         // produced pairs
156         if ( pair.NodeID != minNodeID.get() ){ ... }
157     }
158
159     /**
160      * StarCombiner.java
161      */
162
163     /**
164      * Input:
165      *
166      * <[node], [minNode]>
167      * <[adjacent node], [minNode]>
168      * # the value depends on the type of star operation and the
169      *   comparation of NeighbourID and the NodeID
170      */
171
172     /**
173      * Output:
174      *
175      * # same, but not duplicated.
176      */
177
178 
```

```

169 code:
170
171   '''java
172 package GCC;
173
174 /**
175 * Combiner task of the \see StarDriver Job. */
176 public class StarCombiner extends Reducer<NodesPairWritable,
177   IntWritable, NodesPairWritable, IntWritable>
178 {
179
180   // It reduces the number of duplicates that are emit by the StarMapper.
181   public void reduce( NodesPairWritable pair, Iterable<
182     IntWritable> neighbourhood, Context context )
183     throws IOException, InterruptedException
184   {
185     // set a impossible value to check if seen.
186     int lastNodeSeen = -2;
187     for ( IntWritable neighbour : neighbourhood )
188     {
189       // 1. Skip the duplicate nodes.
190       // 2. Emit the pair
191       // 3. Store the last neighbourId that we have
192       // processed.
193     }
194   }
195 }
```

4. Convergence

The number of modified edges can be counted in this kind of situation that whether the minimum neighbour is the same node as the node we are analyzing.

If it's not the same, the number of modified edges is equal to the number of emitted edges.

In conclusion, the Convergence condition can be presented in an equation as following:

$$\text{LargeStar.NumChanges} + \text{SmallStar.NumChanges} = 0$$

This means that only when both the operations: Large-Star and Small-Star do not modify the graph, then we can conclude that the graph has reached its convergence point.

5. Termination phase

In order to realize this phase, we used three java class files: TerminationDriver.java, TerminationMapper.java, and TerminationReducer.java. The first one is used to configure the Termination Job that has the other two class files respectively as Mapper and Reducer. This Job expects a pairs list of integers nodes as input, and produces an output formatted as following:

“ Cluster_1 Cluster_2 ... Cluster_K Each cluster is a list of increasing NodeID, forming a connected component of the input graph. “

For each pair $\langle u, v \rangle$, this Mapper emits the pair $\langle u, v \rangle$ if $u < v$, otherwise $\langle v, u \rangle$. In this way, the Reducer will receive a cluster for each key. In fact, if a connected component has reached its convergence point, all nodes belonging to this sub-graph are connected to the minimum node.

Therefore, in the Reducer phase, we receive the minimum NodeID of a cluster and all the nodes belonging to it (sorted in ascending order thanks to the comparable technique). What remains to be done is just adding these nodes to a ClusterWritable object and emitting it, so producing a star list. As we said, during this phase, the counting of nodes and clusters number is performed again in order to compare these values with the initial ones.

Related classes

(Not every details are shown here)

```

1  ** StarDriver.java **
2
3  `` `java
4  package GCC;
5  // Driver of the Job responsible for executing the Small-Star or Large-Star operation on the
6  // input graph.
7  public class StarDriver extends Configured implements Tool
8
9  {
10
11     // The StarDriver can be of type Large-StarDriver or Small-StarDriver
12     public enum StarDriverType { LARGE, SMALL };
13
14
15     public StarDriver( StarDriverType type , Path input ,
16                       Path output , int iteration , boolean verbose )
17     {
18         this.type = type;
19         this.title = type.equals( StarDriverType .
20                               SMALL ) ? "Small-Star" + iteration : "
21                               Large-Star" + iteration ;
22
23         ...
24     }
25
26
27     public int run( String[] args ) throws Exception
28     {
29         //set different job accordingly.
30         conf.set( "type" , this.type.toString() );
31         Job job = new Job( conf , this.title );
32
33         ...
34
35         // Set up the private variable looking to the counter value
36         this.numChanges = job.getCounter( UtilCounters.NUM_CHANGES ) .
37                           findCounter( UtilCounters.NUM_CHANGES ) .
38                           getValue();
39     }
40
41
42     // Return the number of changes occurred during the operation Small-Star or
43     // Large-Star.
44     public long getNumChanges(){
45         return this.numChanges;
46     }
47 }
```

```

35   ‘ ‘ ‘
36
37 ** StarMapper . java **
38
39 input :
40
41   ‘ ‘ ‘
42 <[ node ] , [ adjacent node]>
43   ‘ ‘ ‘
44
45 output :
46
47   ‘ ‘ ‘
48 <pair {[ node ] , [ adjacent node]} , [ node]>
49 <pair {[ node ] , [ adjacent node]} , [ adjacent node]>
50 # the value depends on the type of star operation and the
      comparation of NodeID and NeighbourID
51   ‘ ‘ ‘
52
53 code :
54
55   ‘ ‘ ‘ java
56 package GCC;
57
58 public class StarMapper extends Mapper<IntWritable ,
      IntWritable , NodesPairWritable , IntWritable >
59 {
60     // Extract type variable from the context configuration.
61     // Based on this value, this Mapper will behave as a Small-Star Mapper or
          Large-Star Mapper
62     public void setup( Context context ){
63         smallStar = context.getConfiguration().get( "type" ) .
              equals( "SMALL" );
64     }
65
66     // If Large-Star, emits the pairs <u,v> and <v,u>.
67     // If Small-Star, emits the pair <max(u,v), min(u,v)>.
68     public void map( IntWritable nodeID , IntWritable
          neighbourID , Context context ) throws IOException ,
              InterruptedException
69     {
70         // if the node is alone, emit it like is it in order to keep that
              information
71         if ( neighbourID.get() == -1 ){ context.write
              ( pair , neighbourID ); }
72
73         // If we are running Small-Star, we emit only when the
              neighbourID is smaller than nodeID
74

```

```

75     if ( smallStar ) { ... }
76     // If we are running Large-Star, we always emit: <NodeID;
77     // NeighbourID> and <NeighbourID; NodeID>
78     else { ... }
79   }
80   /**
81
82 ** StarReducer.java**
83
84 input:
85
86   """
87 <pair {[node], [adjacent node]}, list {[node]}>
88 <pair {[node], [adjacent node]}, list {[adjacent node]}>
89 # the value depends on the type of star operation
90 """
91
92 output:
93
94   """
95 <[node], [minNode]>
96 <[adjacent node], [minNode]>
97 # the value depends on the type of star operation and the
98 # comparation of NeighbourID and the NodeID
99 """
100
101 code:
102 """
103 package GCC;
104
105 public class StarReducer extends Reducer<NodesPairWritable,
106                                         IntWritable, IntWritable, IntWritable>
107 {
108     // same setup as StarMapper.java
109     public void setup( Context context )
110     {
111         smallStar = context.getConfiguration().get(
112             "type" ).equals( "SMALL" );
113     }
114
115     /* The neighbours are sorted, thanks to the
116      NodesPairWritable.java, we know that the
117      minimum node is either the NodeID or the first neighbour. We
118      call it MinNodeID.
119      For each neighbour, we produce the pairs <NeighbourID
120      , MinNodeID> and <MinNodeID, NeighbourID> :
121      Small-Star: - always;

```

```

117      Large - Star : - only when NeighbourID is greater than
118      NodeID */
119
120  public void reduce( NodesPairWritable pair , Iterable<
121      IntWritable> neighbourhood , Context context )
122      throws IOException , InterruptedException
123  {
124
125      // This means that the nodeID is isolated, so we emit it unchanged
126      if ( pair . NeighbourID == -1 ) { ... }
127
128      // We know that the first element contains the neighbour node with
129      // the minimum label. We just need to compare it with the node id.
130      minNodeID . set( Math . min( pair . NodeID , pair .
131          NeighbourID ) );
132
133      // If we are running Small-Star, we need to connect this node to
134      // the minimum neighbours
135      if ( smallStar && ( pair . NodeID != minNodeID .
136          get() ) ) { ... }
137
138      // same impossible value
139      int lastNodeSeen = -2;
140      for ( IntWritable neighbour : neighbourhood )
141      {
142
143          // Skip the duplicate nodes.
144          if ( neighbour . get() == lastNodeSeen
145              )
146              continue ;
147
148          // If we are running Small-Star, we always emit the
149          // neighbours except when it is the minNodeID
150          // If we are running Large-Star, we emit only when the
151          // neighbourID is greater than nodeID
152          boolean cond = ( smallStar ? (
153              neighbour . get() != minNodeID . get()
154                  ) : ( neighbour . get() > pair .
155                      NodeID ) );
156          if ( cond ) { ... }
157
158          // Store the last neighbourId that we have processed.
159          lastNodeSeen = neighbour . get();
160      }
161
162      // If the NodeID has not the minimum label means that the
163      // produced pairs will be different,
164      // so we increment the number of changes by the number of
165      // produced pairs
166      if ( pair . NodeID != minNodeID . get() ) { ... }
167  }
168
169  " "
170
171

```

```

152
153 ** StarCombiner.java **
154
155 input:
156
157 """
158 <[node] , [minNode]>
159 <[adjacent node] , [minNode]>
160 # the value depends on the type of star operation and the
161 # comparation of NeighbourID and the NodeID
162 """
163
164
165 output:
166 """
167 # same, but not duplicated.
168 """
169
170 code:
171 """
172 package GCC;
173
174 /**
175 * Combiner task of the \see StarDriver Job. */
176 public class StarCombiner extends Reducer<NodesPairWritable,
177 IntWritable , NodesPairWritable , IntWritable >
178 {
179
180     // It reduces the number of duplicates that are emit by the StarMapper.
181     public void reduce( NodesPairWritable pair , Iterable<
182                         IntWritable> neighbourhood , Context context )
183             throws IOException , InterruptedException
184     {
185         // set a impossible value to check if seen.
186         int lastNodeSeen = -2;
187         for ( IntWritable neighbour : neighbourhood )
188         {
189             // 1. Skip the duplicate nodes.
190             // 2. Emit the pair
191             // 3. Store the last neighbourId that we have
192             // processed.
193         }
194     }
195 }
```

6. Check phase

In the Termination Phase, we have made sure to not store duplicate nodes within a cluster, exploiting the secondary sorting property. As we said, a cluster is not legal if one of its node is present in another cluster. So in order to verify if all the resulting clusters are well-formed,

it is sufficient to check that all nodes are unique. If a node is found twice, for what said above, it is surely present in two different clusters.

As usual, we need a class to configure the Job: CheckDriver.java. The Mapper (CheckMapper.java) emits all nodes of the clusters. The Reducer (CheckReducer.java) increments an error counter if receives a given node identifier more than one time.

To be precise, we should check if all the nodes of input graph are present in the final result. During the steps of the algorithm, a node is never added. Besides, as we said, we ensure that there are no duplicated nodes.

Therefore, in order to verify the property, it is necessary to check that the nodes number of the input graph is equal to the one of the final results. These two counting are calculated respectively in the Initialization and Termination Phases. We have chosen to simply report this two values to the final user that will judge if to keep or not the obtained result.

Related classes:

(Not every details are shown here)

```

1  ** CheckDriver . java **
2  ‘ ‘ ‘ java
3  package GCC;
4
5  public class CheckDriver extends Configured implements Tool
6  {
7      public CheckDriver( Path input , boolean verbose ){
8          ...
9      }
10
11     public int run( String [] args ) throws Exception
12     {
13         ...
14
15         // Set up the private variable looking to the counter value
16         this . testOk = ( job . getCounters () . findCounter
17             ( UtilCounters .NUM_ERRORS ) . getValue () ==
18             0 );
19
19         // Delete the output folder ( we did not write on it )
20         FileSystem . get( conf ) . delete( input . suffix (" _check " ) , true );
21
22         return 0;
23     }
24
25     // Return False if the checking found malformed, True otherwise.
26     public boolean isTestOk (){
27         return this . testOk ;
28     }
29     ‘ ‘ ‘
30

```

```

31 **CheckMapper.java**
32
33 input:
34
35   " "
36 <[ cluster ] , NULL>
37   " "
38
39 output:
40
41   " "
42 <[ node ] , NULL>
43   " "
44
45 code:
46
47   /** java
48 package GCC;
49
50 public class CheckMapper extends Mapper<ClusterWritable ,
51   NullWritable , IntWritable , NullWritable >
52 {
53     // For each cluster, it emits all its nodes.
54     public void map( ClusterWritable cluster ,
55       NullWritable _, Context context ) throws
56       IOException , InterruptedException
57     {
58       for ( Integer node : cluster ){...}
59     }
60   }
61   " "

```

7. Translate phase

Because the output is in the format of sequence file, which is binary, so we need another step to translate the output into something human can read.

TranslatorDriver.java makes it possible to translate the result files in text and look which nodes form the connected components.

Related class:

```

1 ** TranslatorDriver**
2
3   /** java
4 package GCC;
5
6 public class TranslatorDriver extends Configured implements
7   Tool
8 {
9   public TranslatorDriver( TranslationType type , Path
10     input , Path output ){...}

```

```

9
10    public int run( String[] args ) throws Exception
11    {
12        ...
13        // simply set the Mapper as the initial Mapper.class
14        job.setMapperClass( Mapper.class );
15        ...
16    }
17    /**
18 */

```

5.3.3 Results

1. preprocess the dataset

```

[root@hadoop102 hadoop-3.1.3]# hadoop jar MapReduceDemo-1.0-SNAPSHOT.jar example.org.mapreduce.lab.driver /input/PP-Decagon_ppi.csv /preprocess
2022-06-05 23:08:16,178 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 23:08:16,555 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this
2022-06-05 23:08:16,571 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654434395401_0030
2022-06-05 23:08:16,647 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:08:16,747 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 23:08:16,775 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:08:16,800 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:08:16,815 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-05 23:08:16,907 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:08:16,928 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1654434395401_0030
2022-06-05 23:08:17,064 INFO conf.Configuration: resource-types.xml not found
2022-06-05 23:08:17,064 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-06-05 23:08:17,107 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0030
2022-06-05 23:08:17,134 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654434395401_0030/
2022-06-05 23:08:17,135 INFO mapreduce.Job: Running job: job_1654434395401_0030

```

The screenshot shows the Hadoop Web UI interface. The main page displays a list of entries under the '/preprocess' directory, including file names and their permissions. A modal window is open over this list, providing detailed information about a specific file block. The modal header is 'File information - part-r-00000'. It contains tabs for 'Download', 'Head the file (first 32K)', and 'Tail the file (last 32K)'. The 'Block information' tab is selected, showing the Block ID (1073744515), Block Pool ID (BP-1748348618-192.168.10.102-1651749363600), Generation Stamp (3694), Size (4009748), and Availability (listing three hosts: hadoop102, hadoop104, and hadoop103). Below the modal, the 'File contents' section shows the beginning of the file's data.

1. compile program to Jar file

The screenshot shows the IntelliJ IDEA interface. The code editor displays Java code for a `CheckMapper` class. The Maven tool window on the right shows the `GCC` project with the `package` goal selected. The terminal window at the bottom shows the build logs for the `GCC` package.

```

    /**
     * Map method of the this CheckMapper class
     * For each cluster, it emits all its node
     * @param cluster the cluster.
     * @param _ not used.
     * @param context context of this Job.
     * @throws IOException, InterruptedException
     */
    public void map(ClusterWritable cluster,
    {
        for (Integer node : cluster)
        {
            nodeID.set(node);
        }
    }

```

```

[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 16.557 s
[INFO] Finished at: 2022-06-05T23:38:06+08:00
[INFO] -----

```

2. load Jar file to VM

The terminal window shows the command `rz -E` being run, followed by a message indicating it is waiting to receive a file. A file transfer progress dialog box is displayed, showing the file `GCC-1.0-SNAPSHOT.jar` has been successfully transferred.

文件名:	GCC-1.0-SNAPSHOT.jar
文件大小:	37.3 KB
传输大小:	37.3 KB
传输速率:	37.3 KB/Sec

3. run program in VM

task1: ConnectedComponents.class

```
[root@hadoop102 hadoop-3.1.3]# rz -E
rz waiting to receive.
[root@hadoop102 hadoop-3.1.3]# hadoop jar GCC-1.0-SNAPSHOT.jar GCC.ConnectedComponents /preprocess/part-r-00000 /output
Start ConnectedComponents.
2022-06-05 23:39:41.246 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:39:41.359 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 23:39:41.690 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654434395401_0032
2022-06-05 23:39:41.740 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:39:41.808 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 23:39:41.838 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:39:41.872 INFO mapreduce.JobsSubmitter: number of splits:1
2022-06-05 23:39:41.961 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:39:41.982 INFO mapreduce.JobsSubmitter: Submitting tokens for job: job_1654434395401_0032
2022-06-05 23:39:41.982 INFO mapreduce.JobsSubmitter: Executing with tokens: []
2022-06-05 23:39:42.097 INFO conf.Configuration: resource-types.xml not found
2022-06-05 23:39:42.138 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0032
2022-06-05 23:39:42.172 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654434395401_0032/
[
```

```
2022-06-05 23:42:00.341 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 23:42:00.349 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:00.370 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:00.387 INFO mapreduce.JobsSubmitter: number of splits:1
2022-06-05 23:42:00.398 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:00.417 INFO mapreduce.JobsSubmitter: Submitting tokens for job: job_1654434395401_0041
2022-06-05 23:42:00.417 INFO mapreduce.JobsSubmitter: Executing with tokens: []
2022-06-05 23:42:00.436 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0041
2022-06-05 23:42:15.555 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 23:42:15.572 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654434395401_0042
2022-06-05 23:42:15.582 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:15.606 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 23:42:15.620 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:15.641 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:15.658 INFO mapreduce.JobsSubmitter: number of splits:1
2022-06-05 23:42:15.672 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:42:15.691 INFO mapreduce.JobsSubmitter: Submitting tokens for job: job_1654434395401_0042
2022-06-05 23:42:15.691 INFO mapreduce.JobsSubmitter: Executing with tokens: []
2022-06-05 23:42:15.708 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0042
2022-06-05 23:42:15.714 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654434395401_0042/
End ConnectedComponents.
Input file format: ADJACENCY_LIST.
Number of initial nodes: 17544.
Number of Cliques: 0.
Number of final nodes: 18055.
Number of Clusters: 30.
TestOK: true.
[root@hadoop102 hadoop-3.1.3]# [
```

result:

The screenshot shows the Hadoop Web UI interface. The top navigation bar includes tabs for Hadoop, Overview, Datanodes, DataNode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled "File information - part-r-00000". It contains three tabs: Download, Head the file (first 32K), and Tail the file (last 32K). The "Download" tab is active. Below it, there's a "Block information" section with a dropdown menu set to "Block 0". This section shows the Block ID (1073744634), Block Pool ID (BP-1748348618-192.168.10.102-1651749363600), Generation Stamp (3813), Size (72660), and Availability (listing three hosts: hadoop103, hadoop104, and hadoop102). To the right of this is a table showing file statistics: Block Size (1 MB), Name (SUCCESS), and another row for part-r-00000. At the bottom of the page, there are links for "File contents" and "File preview". The footer of the page reads "Hadoop, 2019."

this is hadoop sequence file, its binary so cannot be directly read

task2: Cluster2Text

```
[root@hadoop102 hadoop-3.1.3]# hadoop jar GCC-1.0-SNAPSHOT.jar GCC.TranslatorDriver Cluster2Text /output/part-r-00000 /output2
Start TranslatorDriver Cluster2Text.
2022-06-05 23:51:11,685 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
2022-06-05 23:51:12,047 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1654434395401_0043
2022-06-05 23:51:12,112 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:51:12,201 INFO input.FileInputFormat: Total input files to process : 1
2022-06-05 23:51:12,218 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:51:12,242 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:51:12,256 INFO mapreduce.JobsSubmitter: number of splits:1
2022-06-05 23:51:12,355 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2022-06-05 23:51:12,373 INFO mapreduce.JobsSubmitter: Submitting tokens for job: job_1654434395401_0043
2022-06-05 23:51:12,373 INFO mapreduce.JobsSubmitter: Executing with tokens: []
2022-06-05 23:51:12,501 INFO conf.Configuration: resource-types.xml not found
2022-06-05 23:51:12,501 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-06-05 23:51:12,540 INFO impl.YarnClientImpl: Submitted application application_1654434395401_0043
2022-06-05 23:51:12,565 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application_1654434395401_0043/
2022-06-05 23:51:12,565 INFO mapreduce.Job: Running job: job_1654434395401_0043
```

The screenshot shows the Hadoop Web UI interface. At the top, there's a navigation bar with links like Overview, Datanodes, DataNode Volume Failures, Snapshot, Startup Progress, and Utilities. Below the navigation, there's a search bar and a breadcrumb trail showing the path: /output2/part-m-00000.

The main area has two tabs: "File information" and "File contents". The "File information" tab is currently active, displaying details about the file:

- Block ID: 1073744657
- Block Pool: BP-1748348618-192.168.10.102-1651749363600
- Generation Stamp: 3836
- Size: 105446
- Availability: hadoop102, hadoop103, hadoop104

The "File contents" tab shows the binary data of the sequence file. The data starts with:
1 87,5155,81,
10 8260,165,2952,
100
51251,51987,28378,6229,23038,6223,1812,151531,2630,9615,8833,139596,2027,54963
,16,4171,1025,54,386,11,331,1005,05,03,846,18,11,5024,43,309,163,4,340,60,3615,23,326,
7378,902,105,104,4706,1655,9547,7,433,26,155,9,034,3673,8685,23,308,134,12,322,136,
345651,52,53,1499,11273,10521,4860,26277,56953,3251,1446,7301,51176,60487,155,1
32,64080,1803,

6. Difficulties and Solutions

6.1 Memory Allocation Problem

Background description: When running MapReduce: Container killed on request. Exit code is 143

Problem analysis: Insufficient memory allocation, the xml file parameters of MapReduce and yarn need to be adjusted

```
[master@hadoop102 hadoop-3.1.3]$ cd etc/hadoop/
[master@hadoop102 hadoop]$ ll
总用量 172
-rw-r--r--. 1 master master 8260 9月 12 2019 capacity-scheduler.xml
-rw-r--r--. 1 master master 1335 9月 12 2019 configuration.xsl
-rw-r--r--. 1 master master 1940 9月 12 2019 container-executor.cfg
-rw-r--r--. 1 master master 1060 7月 25 19:39 core-site.xml
-rw-r--r--. 1 master master 3999 9月 12 2019 hadoop-env.cmd
-rw-r--r--. 1 master master 15903 9月 12 2019 hadoop-env.sh
-rw-r--r--. 1 master master 3323 9月 12 2019 hadoop-metrics2.properties
-rw-r--r--. 1 master master 11392 9月 12 2019 hadoop-policy.xml
-rw-r--r--. 1 master master 3414 9月 12 2019 hadoop-user-functions.sh.example
-rw-r--r--. 1 master master 1055 7月 25 19:42 hdfs-site.xml
-rw-r--r--. 1 master master 1484 9月 12 2019 httpfs-env.sh
-rw-r--r--. 1 master master 1657 9月 12 2019 httpfs-log4j.properties
-rw-r--r--. 1 master master 21 9月 12 2019 httpfs-signature.secret
-rw-r--r--. 1 master master 620 9月 12 2019 httpfs-site.xml
-rw-r--r--. 1 master master 3518 9月 12 2019 kms-acls.xml
-rw-r--r--. 1 master master 1351 9月 12 2019 kms-env.sh
-rw-r--r--. 1 master master 1747 9月 12 2019 kms-log4j.properties
-rw-r--r--. 1 master master 682 9月 12 2019 kms-site.xml
-rw-r--r--. 1 master master 13326 9月 12 2019 log4j.properties
-rw-r--r--. 1 master master 951 9月 12 2019 mapred-env.cmd
-rw-r--r--. 1 master master 1764 9月 12 2019 mapred-env.sh
-rw-r--r--. 1 master master 4113 9月 12 2019 mapred-queues.xml.template
-rw-r--r--. 1 master master 1314 7月 26 11:58 mapred-site.xml
drwxr-xr-x. 2 master master 24 9月 12 2019 shellprofile.d
-rw-r--r--. 1 master master 2316 9月 12 2019 ssl-client.xml.example
-rw-r--r--. 1 master master 2697 9月 12 2019 ssl-server.xml.example
-rw-r--r--. 1 master master 2642 9月 12 2019 user_ec_policies.xml.template
-rw-r--r--. 1 master master 30 7月 25 20:28 workers
-rw-r--r--. 1 master master 2250 9月 12 2019 yarn-env.cmd
-rw-r--r--. 1 master master 6056 9月 12 2019 yarn-env.sh
-rw-r--r--. 1 master master 2591 9月 12 2019 yarnservice-log4j.properties
-rw-r--r--. 1 master master 1694 7月 26 11:55 yarn-site.xml
```

Change mapred-site.xml file.

```
1 vim mapred-site.xml
```

```

1 <property>
2   <name>mapreduce.map.memory.mb</name>
3   <value>1500</value>
4   <description>The Physical Memory Upper Bound of Map Task
      </description>
5 </property>
6
7 <property>
8   <name>mapreduce.reduce.memory.mb</name>
9   <value>3000</value>
10  <description>The Physical Memory Upper Bound of Reduce
     Task</description>
11 </property>
12
13 <property>
14   <name>mapreduce.map.java.opts</name>
15   <value>-Xmx1200m</value>
16 </property>
17
18 <property>
19   <name>mapreduce.reduce.java.opts</name>
20   <value>-Xmx2600m</value>
21 </property>
22 <property>
23   <name>mapreduce.framework.name</name>
24   <value>yarn</value>
25 </property>

```

Change yarn-site.xml file.

```

1 vim yarn-site.xml

```

```

1 <property>
2   <name>yarn.nodemanager.resource.memory-mb</name>
3   <value>22528</value>
4   <description>The Available Memory for Each Node (MB) </
      description>
5 </property>
6
7 <property>
8   <name>yarn.scheduler.minimum-allocation-mb</name>
9   <value>1500</value>
10  <description>The Smallest Memory for each Task: 1024MB</
      description>
11 </property>
12
13 <property>
14   <name>yarn.scheduler.maximum-allocation-mb</name>
15   <value>16384</value>

```

```

16 <description>The Largest Memory for each Task</
17   description>
</property>
```

```

1 stop-all.sh
2 start-all.sh
```

6.2 Java Heap Space is not enough

This is a kind of heap error. It is caused by a too small heap size.

We solve this problem by change the hadoop configuration,

```

1 cd hadoop/etc/hadoop/
2 vim hadoop-env.sh
```

```

1 export HADOOP_HEAPSIZE=2000
```

```

1 cd hadoop/etc/hadoop/
2 vim mapred-site.xml
```

```

1 <property>
2   <name>mapred.child.java.opts</name>
3   <value>-Xmx2000m</value>
4 </property>
```

6.3 MapReduce Implementation in LargeStar Task

The specific implementation of mapreduce for large and small stars is difficult, and the specific input and output form is very vague. We solve this problem by redefining the pairable class and implementing the compareTo interface.

```

1 public int compareTo(NodesPairWritable other) {
2     int result = this.NodeID - other.NodeID;
3     if (result == 0)
4         result = this.NeighbourID - other.NeighbourID;
5     return result;
6 }
```

6.4 Shuffle Error of MapReduce

```

1 Error: org.apache.hadoop.mapreduce.task.reduce.
2   Shuffle$ShuffleError: error in shuffle in fetcher#1
3   at org.apache.hadoop.mapreduce.task.reduce.Shuffle.
4     run(Shuffle.java:134)
```

```

3      at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask
4          .java:376)
5      at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild
6          .java:167)
7      at java.security.AccessController.doPrivileged(Native
8          Method)
9      at javax.security.auth.Subject.doAs(Subject.java:396)
10     at org.apache.hadoop.security.UserGroupInformation.
11         doAs(UserGroupInformation.java:1556)
12     at org.apache.hadoop.mapred.YarnChild.main(YarnChild.
13         java:162)
14 Caused by: java.lang.OutOfMemoryError: Java heap space
15     at org.apache.hadoop.io.BoundedByteArrayOutputStream
16         .<init>(BoundedByteArrayOutputStream.java:56)
17     at org.apache.hadoop.io.BoundedByteArrayOutputStream
18         .<init>(BoundedByteArrayOutputStream.java:46)
19     at org.apache.hadoop.mapreduce.task.reduce.
20         InMemoryMapOutput.<init>(InMemoryMapOutput.java
21             :63)
22     at org.apache.hadoop.mapreduce.task.reduce.
23         MergeManagerImpl.unconditionalReserve(
24             MergeManagerImpl.java:297)
25     at org.apache.hadoop.mapreduce.task.reduce.
26         MergeManagerImpl.reserve(MergeManagerImpl.java
27             :287)
28     at org.apache.hadoop.mapreduce.task.reduce.Fetcher.
29         copyMapOutput(Fetcher.java:411)
30     at org.apache.hadoop.mapreduce.task.reduce.Fetcher.
31         copyFromHost(Fetcher.java:341)
32     at org.apache.hadoop.mapreduce.task.reduce.Fetcher.
33         run(Fetcher.java:165)

```

According to "Hadoop: The Definitive Guide 4th Edition" (P203-219), a shuffle process is required between the map task and the reduce task, which copies the output of the map task as the input of the reduce task.

Specifically, the input of the shuffle process is: the output file of the map task, its output receiver is: the memory buffer on the machine running the reduce task, and the shuffle process runs in parallel.

The parameter `mapreduce.reduce.shuffle.input.buffer.percent` controls how much of the memory on the machine running the reduce task is used as the above buffer (the default value is 0.70), and the parameter `mapreduce.reduce.shuffle.parallelcopies` controls the parallelism of the shuffle process (the default value is 0.70). value is 5)

Then "`mapreduce.reduce.shuffle.input.buffer.percent`" * "`mapreduce.reduce.shuffle.parallelcopies`" must be less than or equal to 1, otherwise the above error will occur.

Therefore, I set `mapreduce.reduce.shuffle.input.buffer.percent` to a value of 0.1 and it works fine (set to 0.2, it still throws the same error)

Another solution is to solve this problem by modifying the `mapred.xml` file.

```
1 <property>
2   <name>mapreduce.reduce.input.buffer.percent</name>
3   <value>0.0</value>
4 </property>
```

In addition, it can be found that if the default values of the two parameters are used, the product of the two is 3.5, which is much larger than 1. Why are the above errors not often thrown?

1) First, set the default values to be relatively large, mainly based on performance considerations. Setting them to relatively large can greatly speed up the speed of copying data from the map

2) Secondly, to throw the above exception, another condition needs to be met, that is, the data of the map task is ready to be copied by shuffle. In this case, the "number of threads" and "Memory buffer usage" is a full load value, which naturally causes an insufficient memory error; and if the data of the map task is completed intermittently, there is no "thread number" and "memory buffer usage" of the shuffle process at any time. It is a full load value, so naturally it will not throw the above error

In addition, if an error still occurs after setting the above parameters, it may be that the total memory of the process running the Reduce task is insufficient, which can be adjusted by the mapred.child.java.opts parameter, such as setting mapred.child.java.opts =-Xmx2048m.

7. Conclusions

In this research project, we propose a new distributed graph data processing system: DGPS. DGPS is based on the distributed computing framework Hadoop, which implements parallel computing of graph algorithms. Specifically, in practice, we split the graph into several smaller graphs and put them on different machines for computation. Based on DGPS, we have implemented three graph algorithms: Small Dense Subgraphs Finding, PageRank and Graph Connected Components. We choose a large scale graph for our experiments and obtained efficient running speed and good performance no less than the single machine algorithm. These results confirm the efficiency and effectiveness of our framework. It also reveals the potential of distributed graph processing systems.

We provide the complete framework design, algorithm implementation, and experimental procedure and results. We use text descriptions, frame diagrams, pseudo-code and screenshots to show our results and workload in a variety of ways.