**STAGE 1**

**MAP PHASE**

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.StringTokenizer;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class OnlMap2 extends MapReduceBase implements Mapper <LongWritable, Text, Text, Text >{

public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {

StringTokenizer st = new StringTokenizer(value.toString());

String t = st.nextToken(); %Token

String uid = st.nextToken(); %User ID

String uid\_size = st.nextToken(); % Size of that user ID

String frequency = st.nextToken(); %frequency of that user ID

String data = st.nextToken(); % Data in the UID

String str\_key = t+" "+data;

String str\_val = uid+" "+uid\_size+" "+frequency;

//System.out.println("frequency:" +uid\_size);

output.collect(new Text(str\_key),new Text(str\_val));

}

}

**REDUCE PHASE**

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class OnlReduce2StopWords extends MapReduceBase implements Reducer<Text, Text, Text, Text>{

public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException, NumberFormatException {

int count = 0;

int sum = 0;

ArrayList<CustomMapGI> alcmap = new ArrayList<CustomMapGI>();

StringTokenizer st = new StringTokenizer(key.toString());

String t = st.nextToken();

String data = st.nextToken();

while(values.hasNext())

{

String val = values.next().toString();

StringTokenizer stv = new StringTokenizer(val);

String uid = stv.nextToken();

String uid\_size = stv.nextToken();

System.out.println("uid\_size:" +uid\_size);

String frequency = stv.nextToken();

CustomMapGI cmap = new CustomMapGI(uid, uid\_size, frequency);

alcmap.add(cmap);

sum += Integer.parseInt(frequency);

count++;

}

// System.out.println("alcamp size:" +alcmap.size());

if(alcmap.size()<5000)

{

for(int i = 0; i<alcmap.size(); i++)

{

for(int j = 1; j<alcmap.size(); j++)

{

if((j>i)&&(!alcmap.get(i).equals(alcmap.get(j))))

{

String uidi = alcmap.get(i).getUid();

String uidj = alcmap.get(j).getUid();

ArrayList<String> alsort = new ArrayList<String>();

alsort.add(uidi);

alsort.add(uidj);

String i\_size = null;

String j\_size = null;

String freq\_i = null;

String freq\_j = null;

if(alcmap.get(i).getUid().equals(alsort.get(0))

{

i\_size = alcmap.get(i).getSize();

j\_size = alcmap.get(j).getSize();

freq\_i = alcmap.get(i).getFrequency();

freq\_j = alcmap.get(j).getFrequency();

}

Collections.sort(alsort);

int f\_i = Integer.parseInt(freq\_i);

int f\_j = Integer.parseInt(freq\_j);

int si = Integer.parseInt(i\_size);

int sj = Integer.parseInt(j\_size);

int small = 0;

int big = 0;

if(si<sj)

{

small = si;

big = sj;

}

else

{

small = sj;

big = si;

}

double small1 = (double) small;

double big1 = (double)big;

double filter = small1/big1;

System.out.println("filter:" +filter);

double checker = 0.7;

if(filter > checker)

{

String mul\_key = t+" "+uidi+" "+uidj+" "+

i\_size+" "+j\_size;

String str\_val = f\_i+" "+f\_j+" "+data;

output.collect(new Text(mul\_key),

new Text(str\_val));

}

}

}

}

}

**STAGE II**

**MAP PHASE**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class OnlMap3 extends MapReduceBase implements Mapper <LongWritable, Text, Text,Text> {

public void map(LongWritable key,Text value,OutputCollector<Text,Text> output, Reporter reporter) throws IOException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

String t = tokenizer.nextToken();

String uidi = tokenizer.nextToken();

String uidj = tokenizer.nextToken();

String i\_size = tokenizer.nextToken();

String j\_size = tokenizer.nextToken();

String freq\_i = tokenizer.nextToken();

String freq\_j = tokenizer.nextToken();

String data = tokenizer.nextToken();

String str\_key = t+" "+uidi+" "+uidj;

String str\_val= i\_size+" "+j\_size+" "+freq\_i+" "+freq\_j+" +data;

output.collect(new Text(str\_key), new Text(str\_val));

}

}

**REDUCE PHASE**

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class OnlReduce2StopWords extends MapReduceBase implements Reducer<Text, Text, Text, Text>{

public void reduce (Text key, Iterator<Text> values, OutputCollector <Text, Text> output, Reporter reporter) throws IOException, NumberFormatException {

int p = 0; % alpha p is p

int q = 0;

boolean passpf=false;

boolean passfc=false;

boolean passsf = false;

double z = 0;

int sum = 0;

int count = 0;

StringTokenizer st = new StringTokenizer(key.toString());

String t= st.nextToken();

String uidi = st.nextToken();

String uidj = st.nextToken();

int fi = 0;

int fj=0;

int s\_i = 0;

int s\_j = 0;

String data= "";

String i\_size = "";

String j\_size = "";

while(values.hasNext())

{

String val = values.next().toString();

StringTokenizer stv = new StringTokenizer(val);

i\_size = stv.nextToken();

j\_size = stv.nextToken();

String freq\_i = stv.nextToken();

String freq\_j = stv.nextToken();

data = stv.nextToken();

s\_i = Integer.parseInt(i\_size);

s\_j = Integer.parseInt(j\_size);

fi = Integer.parseInt(freq\_i);

fj = Integer.parseInt(freq\_j);

}

double sii = (double)s\_i;

double sjj = (double)s\_j;

int intersection = 0;

int maxi = 0;

if (fi < fj)

{

q = fi;

intersection = fi;

maxi = fj;

}

else

{

q = fj;

intersection = fj;

maxi = fi;

}

p += q; // alpha p is p

double alphap = (double)p;

z = (7/17)\*(sii+sjj); //z is alpha from stage -1 map phase

int pos\_i=0;

int pos\_j=0;

if (!values.hasNext())

{

pos\_i += fi;

pos\_j += fj;

int part\_i = pos\_i+fi;

int part\_j = pos\_j+fj;

int mini= s\_i-part\_i;

int minj = s\_j-part\_j;

double min\_i = (double)mini;

double min\_j= (double)minj;

if (mini<minj)

{

double overlap = min\_i;

double ubound = overlap + alphap;

if (ubound >= z)

{

passpf=true;

double suf\_mx = (((0.7)\*s\_i)-1);

double suf\_my = (((0.7)\*s\_j)-1);

int l\_x = 0;

l\_x += fi;

double r\_x = suf\_mx - (l\_x+fi);

int l\_y = 0;

l\_y += fj;

double r\_y = suf\_my - (l\_y+fj);

int diff = 0;

if (pos\_i == pos\_j)

{

diff = q;

}

else

{

diff = fi + fj;

}

double hs = Math.abs(l\_x-l\_y)+Math.abs(r\_x-r\_y)+diff;

double hpmin = (pos\_i+fi) + (pos\_j+fj) - 2\*p;

double hmax = s\_i + s\_j - (2\*z) - hpmin;

if (hs<hmax)

{

passsf = true;

String str\_key = t+" "+uidi+" "+uidj;

String str\_val = s\_i+" "+s\_j+" "+intersection+" "+maxi+" "+data;

output.collect(new Text(str\_key), new Text(str\_val));

}

}

}

}

**STAGE III**

**MAP PHASE**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class OnlMap4 extends MapReduceBase implements Mapper <LongWritable, Text, Text, Text> {

public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

String t = tokenizer.nextToken();

String uidi = tokenizer.nextToken();

String uidj = tokenizer.nextToken();

String i\_size = tokenizer.nextToken();

String j\_size = tokenizer.nextToken();

String intersection = tokenizer.nextToken();

String maxi = tokenizer.nextToken();

String data = tokenizer.nextToken();

String str\_key = t+" "+uidi+" "+uidj+" "+i\_size+" "+j\_size;

String str\_val= intersection+" "+maxi;

output.collect(new Text(str\_key), new Text(str\_val));

}

}

**REDUCE PHASE**

import java.io.IOException;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class OnlReduce4 extends MapReduceBase implements Reducer <Text, Text, Text, Text>{

public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {

StringTokenizer st = new StringTokenizer(key.toString());

String t = st.nextToken();

String uidi = st.nextToken();

String uidj = st.nextToken();

String i\_size = st.nextToken();

String j\_size = st.nextToken();

int inter=0;

int max = 0;

int interr=0;

int maxx=0;

while(values.hasNext())

{

String val = values.next().toString();

StringTokenizer stv = new StringTokenizer(val);

String intersection = stv.nextToken();

String maxi = stv.nextToken();

inter = Integer.parseInt(intersection);

max = Integer.parseInt(maxi);

interr = interr+inter;

maxx = maxx+max;

}

System.out.println("inter: "+inter);

System.out.println("max: "+max);

double interrr = (double) interr;

double maxxx = (double)maxx;

double similarity = ((interrr)/(maxxx));

if (similarity>0.7)

{

String str\_key = t+" "+uidi+" "+uidj;

String str\_val = String.valueOf(similarity);

output.collect(new Text(str\_key), new Text(str\_val));

}

}

}