**#AWK Program**

#!/usr/bin/awk/env bash

for year in all/\*

do

echo -ne `basename $year .gz`"\t"

gunzip -c $year | \

awk '{ temp = substr($0, 88, 5) + 0;

q = substr($0, 93, 1);

if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }

END { print max }'

done

**#Python Program**

import re

import sys

for line in sys.stdin:

val = line.strip()

(year,temp, q) = (val[15:19], val[87:92], val[31:32])

temp=int(temp)

if (temp != 9999 and re.match("[01459]", q)):

print "%s\t%s" % (year, temp)

import sys

(last\_key, max\_val) = (None, -9999)

for line in sys.stdin:

(key, val) = line.strip().split("\t")

if last\_key and last\_key != key:

print "%s\t%s" % (last\_key, max\_val)

(last\_key, max\_val) = (key, int(val))

else:

(last\_key, max\_val) = (key, max(max\_val, int(val)))

if last\_key:

print "%s\t%s" % (last\_key, max\_val)

**/\* Word count**

package com.sample;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.io.Text;

public class WordCountCombiner extends Reducer<Text,LongWritable,Text,LongWritable>

{

@Override

protected void reduce(Text key,Iterable<LongWritable> value,Context context)throws IOException,InterruptedException

{

long sum=0;

while(value.iterator().hasNext())

{

sum+=value.iterator().next().get();

}

context.write(key,new LongWritable(sum));

}

}

package com.sample;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

//import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.util.Tool;

public class WordCountJob implements Tool{

private Configuration conf;

@Override

public Configuration getConf()

{

return conf;

}

@Override

public void setConf(Configuration conf)

{

this.conf=conf;

}

@Override

public int run(String []args)throws Exception

{

Job wordcountjob=new Job(getConf());

wordcountjob.setJobName("mat word count");

wordcountjob.setJarByClass(this.getClass());

wordcountjob.setMapperClass(WordCountMapper.class);

wordcountjob.setReducerClass(WordCountReducer.class);

wordcountjob.setCombinerClass(WordCountCombiner.class);

wordcountjob.setMapOutputKeyClass(Text.class);

wordcountjob.setMapOutputValueClass(LongWritable.class);

wordcountjob.setOutputKeyClass(Text.class);

wordcountjob.setOutputValueClass(LongWritable.class);

FileInputFormat.setInputPaths(wordcountjob,new Path(args[0]));

FileOutputFormat.setOutputPath(wordcountjob,new Path(args[1]));

wordcountjob.setNumReduceTasks(2);

return wordcountjob.waitForCompletion(true)==true? 0:1;

}

public static void main(String []args)throws Exception

{

ToolRunner.run(new Configuration(),new WordCountJob(),args);

}

}

package com.sample;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

//import org.apache.hadoop.mapreduce.Counter;

public class WordCountMapper extends

Mapper<LongWritable, Text, Text, LongWritable> {

private final static LongWritable one = new LongWritable(1);

@Override

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

String[] words = line.split(" ");

for (int i = 0; i < words.length; i++) {

context.write(new Text(words[i]), one);

}

/\*StringTokenizer strtock = new StringTokenizer(str);

while (strtock.hasMoreTokens()) {

temp.set(strtock.nextToken());

context.write(temp, one);

}\*/

}

}

package com.sample;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.io.Text;

public class WordCountReducer extends Reducer<Text,LongWritable,Text,LongWritable>

{

@Override

protected void reduce(Text key,Iterable<LongWritable> value,Context context)throws IOException,InterruptedException

{

long sum=0;

while(value.iterator().hasNext())

{

sum+=value.iterator().next().get();

}

context.write(key,new LongWritable(sum));

}

}

/\*[cloudera@localhost ~]$ hadoop jar wo1.jar com.sample.WordCountJob /user/cloudera/bhaskar/t.txt /user/cloudera/bhaskar/w10

**/\* MAXTEMPERATURE**

package com.tem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

//import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

public class MaxTemperature {

public static void main(String[] args) throws Exception {

if (args.length != 2) {

System.err.println("Usage: MaxTemperature <input path> <output path>");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(MaxTemperature.class);

job.setJobName("Max temperature");

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setMapperClass(MaxTemperatureMapper.class);

job.setReducerClass(MaxTemperatureReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

#!/usr/bin/env python

#import sys because we need to read and write data to STDIN and STDOUT

import sys

# reading entire line from STDIN (standard input)

for line in sys.stdin:

# to remove leading and trailing whitespace

line = line.strip()

# split the line into words

words = line.split()

# we are looping over the words array and printing the word

# with the count of 1 to the STDOUT

for word in words:

# write the results to STDOUT (standard output);

# what we output here will be the input for the

# Reduce step, i.e. the input for reducer.py

print '%s\t%s' % (word, 1)\

#!/usr/bin/env python

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

# read the entire line from STDIN

for line in sys.stdin:

# remove leading and trailing whitespace

line = line.strip()

# slpiting the data on the basis of tab we have provided in mapper.py

word, count = line.split('\t', 1)

# convert count (currently a string) to int

try:

count = int(count)

except ValueError:

# count was not a number, so silently

# ignore/discard this line

continue

# this IF-switch only works because Hadoop sorts map output

# by key (here: word) before it is passed to the reducer

if current\_word == word:

current\_count += count

else:

if current\_word:

# write result to STDOUT

print '%s\t%s' % (current\_word, current\_count)

current\_count = count

current\_word = word

# do not forget to output the last word if needed!

if current\_word == word:

print '%s\t%s' % (current\_word, current\_count)

package com.tem;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper

extends Mapper<LongWritable, Text, Text, IntWritable> {

private static final int MISSING = 9999;

@Override

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

String year = line.substring(15, 19);

int airTemperature;

if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs

airTemperature = Integer.parseInt(line.substring(88, 92));

} else {

airTemperature = Integer.parseInt(line.substring(87, 92));

}

String quality = line.substring(92, 93);

if (airTemperature != MISSING && quality.matches("[01459]")) {

context.write(new Text(year), new IntWritable(airTemperature));

}

}

}

package com.tem;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer

extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<IntWritable> values,

Context context)

throws IOException, InterruptedException {

int maxValue = Integer.MIN\_VALUE;

for (IntWritable value : values) {

maxValue = Math.max(maxValue, value.get());

}

context.write(key, new IntWritable(maxValue));

}

}

$ hadoop jar temp.jar com.tem.MaxTemperature /user/cloudera/bhaskar/tt2.txt /user/cloudera/bhaskar/w8

**// MatrixMultiplication Program**

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiplication {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

// A is an m-by-n matrix; B is an n-by-p matrix.

conf.set("m", "2");

conf.set("n", "5");

conf.set("p", "3");

Job job = new Job(conf, "MatrixMultiplication");

job.setJarByClass(MatrixMultiplication.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(MatrixMapper.class);

job.setReducerClass(MatrixReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);

}

}

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MatrixMapper extends Mapper<LongWritable, Text,

Text, Text> {

public void map(LongWritable key, Text value, Context

context) throws IOException, InterruptedException {

Configuration conf = context.getConfiguration();

int m = Integer.parseInt(conf.get("m"));

int p = Integer.parseInt(conf.get("p"));

String line = value.toString();

String[] indicesAndValue = line.split(",");

Text outputKey = new Text();

Text outputValue = new Text();

if (indicesAndValue[0].equals("A")) {

for (int k = 0; k < p; k++) {

outputKey.set(indicesAndValue[1] + "," + k);

outputValue.set("A," + indicesAndValue[2] + "," +

indicesAndValue[3]);

context.write(outputKey, outputValue);

}

} else {

for (int i = 0; i < m; i++) {

outputKey.set(i + "," + indicesAndValue[2]);

outputValue.set("B," + indicesAndValue[1] + "," +

indicesAndValue[3]);

context.write(outputKey, outputValue);

}

}

}

}

import java.io.IOException;

import java.util.HashMap;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class MatrixReducer extends Reducer<Text, Text, Text, Text> {

public void reduce(Text key, Iterable<Text> values, Context

context) throws IOException, InterruptedException {

String[] value;

HashMap<Integer, Float> hashA = new HashMap<Integer,

Float>();

HashMap<Integer, Float> hashB = new HashMap<Integer,

Float>();

for (Text val : values) {

value = val.toString().split(",");

if (value[0].equals("A")) {

hashA.put(Integer.parseInt(value[1]),

Float.parseFloat(value[2]));

} else {

hashB.put(Integer.parseInt(value[1]),

Float.parseFloat(value[2]));

}

}

int n =

Integer.parseInt(context.getConfiguration().get("n"));

float result = 0.0f;

float a\_ij;

float b\_jk;

for (int j = 0; j < n; j++) {

a\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;

b\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;

result += a\_ij \* b\_jk;

}

if (result != 0.0f) {

context.write(null, new Text(key.toString() + "," +

Float.toString(result)));

}

}

}

4 5 6 7 8

1 3 4 5 6

6 3 4

1 2 7

1 2 7

8 6 5

2 3 1

7 4 2

A,0,0,4

A,0,1,5

A,0,2,6

A,0,3,7

A,0,4,8

A,1,0,1

A,1,1,3

A,1,2,4

A,1,3,5

A,1,4,6

B,0,0,6

B,0,1,3

B,0,2,4

B,1,0,1

B,1,1,2

B,1,2,7

B,2,0,8

B,2,1,6

B,2,2,5

B,3,0,2

B,3,1,3

B,3,2,1

B,4,0,7

B,4,1,4

B,4,2,2

0,0,147.0

0,1,111.0

0,2,104.0

1,0,93.0

1,1,72.0

1,2,62.0