Case-Based Reasoning Implementation on Hadoop and MapReduce Frameworks

Done By: Soufiane Berouel
Supervised By: Dr Lily Liang

Independent Study – Advanced Case-Based Reasoning
Department of Computer Science and Information Technology
University of the District of Columbia
Washington, District of Columbia, 20008, USA
soufiane.berouel@yahoo.com

4/26/2010
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Design and Development</td>
<td>3</td>
</tr>
<tr>
<td>Methodology</td>
<td>3</td>
</tr>
<tr>
<td>Requirements specification</td>
<td>4</td>
</tr>
<tr>
<td>MapReduce implementation of a CBR program for a one node cluster</td>
<td>4</td>
</tr>
<tr>
<td>One and two node Hadoop cluster set up</td>
<td>5</td>
</tr>
<tr>
<td>MapReduce Implementation of a CBR program on a two node cluster</td>
<td>6</td>
</tr>
<tr>
<td>Conclusion</td>
<td>8</td>
</tr>
<tr>
<td>References</td>
<td>9</td>
</tr>
<tr>
<td>Appendices</td>
<td>10</td>
</tr>
</tbody>
</table>
Introduction

Apache Hadoop is a Java software framework that supports data-intensive distributed applications. The framework is extensively used by big companies such as Yahoo for their data and web applications. Moreover, IBM and Google have announced a major initiative to use Hadoop to support university courses in distributed computer programming [1].

One way to developing applications on Hadoop is MapReduce, a software framework introduced by Google to support distributed computing on large data sets on clusters of computers. MapReduce libraries have been written in different programming languages such as C++, Java, and Python [2].

Both Hadoop and MapReduce are very new frameworks. And due to their short age, their use is not very common except for big corporations, hardcore programmers, and hackers. Another reason why these frameworks are not very widespread is the fact that not many packages and libraries have been developed to help normal programmers take advantage of such powerful technologies.

Our project mainly revolves around Case-Based Reasoning (CBR) algorithms which aim to solve problems based on similar past solutions. So when we chose Hadoop and MapReduce as development environments, we had to look up past literature and research that have been done in the same area. Not surprisingly, we were unsuccessful in finding any past attempts to put MapReduce and CBR together. We believe that this is due to the fact that MapReduce and Hadoop are very recent technologies.

Thus, in our project, we developed a Case-Based Reasoning application using the Hadoop and MapReduce frameworks running on Linux Based machines.
Design and Development

1. Methodology

The first step in this project was to implement a Case-Based Reasoning program using MapReduce. As its name suggests, MapReduce programs are divided into two functional blocks: map and reduce. In the map stage, the master node in the cluster takes the input, chops it up into smaller pieces, and distributes the parts to the other nodes. In the reduce step however, the master machine collects the result to all the sub-problems and combines them to get the final output.

To develop programs in MapReduce, programmers need a software framework, such as Hadoop, that supports parallel computing. But since we had no experience using Hadoop or any similar technologies, we managed to find a learning environment that we could use to learn MapReduce separately. The environment is Cloudera’s Hadoop Training Virtual Machine. After going through many tutorials and sample codes, we managed to create our Case-Based Reasoning implementation, using Eclipse Integrated Development Environment and Java programming language as a basis. This code however will run on only one machine since Cloudera’s Training machine cannot emulate multiple node environments.

After implementing our program, the following chapter in our project was to configure the Hadoop framework on a Linux-OS based computer. We used an Ubuntu 9.1 desktop machine and a sample program “WordCount” that is offered by Hadoop to verify that our configuration is correct.

The final step was to follow the same configuration instructions on another computer. Then later connect both computers following the two nodes instructions, and run our CBR program on both of them.
The following table represents different phases in our project with respect to time:

<table>
<thead>
<tr>
<th>Period</th>
<th>Week 1 and 2</th>
<th>Week 3 and 4</th>
<th>Week 5 through 7</th>
<th>Week 8 through 10</th>
<th>Week 11 through 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>Requirements Specification</td>
<td>CBR, MapReduce Single Node Implementation</td>
<td>Set up Cluster - One Node Configuration</td>
<td>Set up Cluster - Two Node Configuration</td>
<td>CBR, MapReduce, Multiple Node Implementation</td>
</tr>
</tbody>
</table>

### 2. Requirements specification

When I first registered in Advanced Case-Based Reasoning course, there was a wide variety of topics that we could tackle using CBR and address them in an innovative way. Subjects included data-mining, mobile application development, and finally parallel computing. What made the last choice stand out was the availability of new, open source, and powerful environments that allow and encourage new advances in the field. Systems such as Hadoop and MapReduce, while used by few companies, are very new and full of potential. This gives a chance to a student like me to get familiar and do research on a new technology that could become the cornerstone of data handling in the upcoming few years.

For this project, we need to develop a diabetes detection program in MapReduce that uses CBR algorithms. The biggest challenge of a CBR is finding an accurate indexing function. In our dataset, there are five variables that influence the glycosylated hemoglobin (GH) level in blood: the stabilized glucose level, height, weight, waist, and hip sizes. It is medically known that whenever the level of GH in a person’s blood is over 7, the person is considered to have diabetes. These variables have various influences on GH thus they will have different weights in our function. The best indexing formula that we found is [6]:

\[
Index = \left(\frac{\text{glucose}}{100}\right) + \left(\frac{\text{waist}}{\text{hip}}\right) \times 2 + \left(\frac{\text{weight}}{\text{height}}\right) / 3
\]
We are also required to configure the Hadoop framework on a small cluster of Linux based computers and run our application on it.

3. MapReduce implementation of a CBR program for a one node cluster

To facilitate our first steps with MapReduce, we used a virtual machine that is pre-configured. The VM includes Cloudera’s Distribution for Hadoop and many MapReduce example codes, as well as Eclipse and other standard tools.

The most challenging part about MapReduce is to adapt already existing functions to do your own tasks. The code consists of two main blocks, or classes since we are using Java, which are map and reduce. In the map function, we get the data from Hadoop’s Distributed File System (HDFS) and divide it using mapping tools into several parts. The next step is to analyze that data and perform our Case-Based Reasoning algorithm to decide what outputs to keep. The reduce function accumulates all those outputs, compares them, then picks the best one. The final result can be found under a new folder in HDFS. Our full implementation is shown in Appendix 1.

4. One and two node Hadoop cluster set up

Hadoop configuration is quite complicated. The first challenge we faced was user permissions and access on Ubuntu. The newer versions of the operating system are very security oriented, so any file modification or folder creation has to be done from the terminal using special commands. To avoid that, it is preferable to use the root account since the other way around could get frustrating.

Every detail of our one and two node set ups could be found in Appendices 2-a and 2-b. For clusters, it is recommended to use the same folder names but different root passwords. The reason behind that is making sure that the public key of your master node is copied to your slave machines. When using the same password for all machines, users sometime don’t pay attention
that the Hadoop System is actually asking them for the slave password, instead of the master one, which means that access authorizations are not configured correctly.

5. **MapReduce Implementation of a CBR program on a two node cluster**

The merging of two nodes when our CBR application was already running on each one separately was quite simple. We just had to create a java executable file “jar” for our code and run it from the Ubuntu terminal as a normal Hadoop program. We also had to modify the code so that we get one final result in our program. The problem that could happen is that, while your MapReduce code specifies one output, when the input is divided into smaller sub-problems, the reducer will output more than one result. The solution is to change the number of reducers to one for the final step or add another reduce step for sub-outputs.

To make sure that our program is fully functional, we went through two testing stages. In the first stage, we test our CBR application and algorithm using cross-validation. In every round of this method, data is partitioned into two complementary subsets: training and validation. Cross validation requires multiple rounds on different partitions to be accurate, and the validation results are averaged over the rounds.

In our project, we have conducted three tests. For each one of them, we kept 70 cases as the training set and 30 cases as the validation set. For the different sets, different samples were chosen for better assessment. The hit scores for the tests were respectively: 72.3, 67.4, and 71.9 percent. The average score was 70.53 percent [6].

The second stage of testing involves the Hadoop and MapReduce frameworks. To verify that our configuration and implementation are both working, we ran trials for the same records on a Hadoop two node cluster, one node cluster, and a simple machine using Java only without MapReduce. Our results confirm that the system is fully functional in this area which means that
our Hadoop configuration is proper, and that our MapReduce code correctly translates our CBR goals.

The following figure represents a sample execution of our CBR application on a two node cluster. As you can see in the bottom, the program informs the user that he/she has diabetes and provides him/her with the index of closest case in our database.

Figure 1: Sample Execution of a CBR Application Running on Hadoop and MapReduce Frameworks
**Conclusion**

In this project, we implemented a CBR program using MapReduce that runs on a cloud of computers through the Hadoop framework that we configured ourselves. While, as far as we know, such projects have never been developed before, we believe that there is more we can do to help the cloud computing community. For the time being, we are planning to create libraries that will help future programmers with similar projects and write a publication that would document our experience.

Cloud computing is as a very simple concept that makes a lot of sense especially on the business side. And if there is anything we could count on during nowadays economy, it is that wherever there is a market, software companies will strongly compete to create the best applications and win it over. This puts frameworks like Hadoop and MapReduce in a tough spot because in few years, as cloud servers will become more popular, new technologies will be developed to handle data on such servers. We believe the whole environment should get richer in terms of libraries, tutorials, and documentations so that it could stand a chance when different industry giants get into the competition.
References

[4] MapReduce Tutorial,  
   http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
[5] Cloudera’s Hadoop Training Virtual Machine,  
Appendices

1 - a) MapReduce Single Node Implementation for a CBR program.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.StringTokenizer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class DiabetesCBR {

    public static class Person {

        public Person(Double glucose, Double cholesterol, Double height, Double weight,
                      Double waist, Double hip, Boolean decision) {
            this.glucose = glucose;
            this.cholesterol = cholesterol;
            this.height = height;
            this.weight = weight;
            this.waist = waist;
            this.hip = hip;
            this.decision = decision;
        }

        Double glucose, cholesterol, height, weight, waist, hip;
        Boolean decision;
    }
}
```
public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

    private Text word1 = new Text();
    private Text word2 = new Text();
    private Text word3 = new Text();
    private Text word4 = new Text();
    private Text word5 = new Text();
    private Text word6 = new Text();
    private Text word7 = new Text();
    private Text word8 = new Text();

    double currentCaseStabilizedGlucose = 0;
    double currentCaseCholesterolOverHdlRatio = 0;
    double currentCaseHeight = 0;
    double currentCaseWeight = 0;
    double currentCaseWaistSize = 0;
    double currentCaseHipSize = 0;

    LinkedList<Person> persons = new LinkedList<Person>();

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

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

        String currentCaseLine;
        int i;

        try {
            BufferedReader in = new BufferedReader(new FileReader("/usr/local/hadoop/DiabetesCurrentCaseData"));

            currentCaseLine = in.readLine();
            for (i = currentCaseLine.length() - 1; i > 0; i--)
                if (currentCaseLine.charAt(i) == ' ')
                    break;
            currentCaseStabilizedGlucose = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

            currentCaseLine = in.readLine();
            for (i = currentCaseLine.length() - 1; i > 0; i--)
                if (currentCaseLine.charAt(i) == ' ')
                    break;
            currentCaseCholesterolOverHdlRatio = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

            currentCaseLine = in.readLine();
            for (i = currentCaseLine.length() - 1; i > 0; i--)
                if (currentCaseLine.charAt(i) == ' ')
                    break;
            currentCaseHeight = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

            currentCaseLine = in.readLine();
            for (i = currentCaseLine.length() - 1; i > 0; i--)
                if (currentCaseLine.charAt(i) == ' ')
                    break;
            currentCaseWeight = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

            currentCaseLine = in.readLine();
            for (i = currentCaseLine.length() - 1; i > 0; i--)
                if (currentCaseLine.charAt(i) == ' ')
                    break;
            currentCaseWaistSize = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

            currentCaseLine = in.readLine();
            for (i = currentCaseLine.length() - 1; i > 0; i--)
                if (currentCaseLine.charAt(i) == ' ')
                    break;
            currentCaseHipSize = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

            output.collect(new Text(line), new IntWritable(1));
        }
    }
}
break;
currentCaseCholesterolOverHdlRatio =
Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

currentCaseLine = in.readLine();
for (i = currentCaseLine.length() - 1; i > 0; i--)
    if (currentCaseLine.charAt(i) == ' ')
        break;
currentCaseHeight = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

currentCaseLine = in.readLine();
for (i = currentCaseLine.length() - 1; i > 0; i--)
    if (currentCaseLine.charAt(i) == ' ')
        break;
currentCaseWeight = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

currentCaseLine = in.readLine();
for (i = currentCaseLine.length() - 1; i > 0; i--)
    if (currentCaseLine.charAt(i) == ' ')
        break;
currentCaseWaistSize = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

currentCaseLine = in.readLine();
for (i = currentCaseLine.length() - 1; i > 0; i--)
    if (currentCaseLine.charAt(i) == ' ')
        break;
currentCaseHipSize = Double.parseDouble(currentCaseLine.substring(i, currentCaseLine.length()));

} catch (Exception ee) {} 

while (tokenizer.hasMoreTokens()) {
    word1.set(tokenizer.nextToken());
    word2.set(tokenizer.nextToken());
    word3.set(tokenizer.nextToken());
    word4.set(tokenizer.nextToken());
    word5.set(tokenizer.nextToken());
    word6.set(tokenizer.nextToken());
    word7.set(tokenizer.nextToken());

    double w1 = Double.parseDouble(word1.toString());
    double w2 = Double.parseDouble(word2.toString());
double w3 = Double.parseDouble(word3.toString());
double w4 = Double.parseDouble(word4.toString());
double w5 = Double.parseDouble(word5.toString());
double w6 = Double.parseDouble(word6.toString());
int dec = Integer.parseInt(word7.toString());
Boolean w7 = Boolean.TRUE;
if (dec == 0)
    w7 = Boolean.FALSE;

persons.add(new Person(w1,w2,w3,w4,w5,w6,w7));
}

LinkedList<Double> indexValues = new LinkedList<Double>();
for (int j = 0; j < persons.size(); j++) {
    double indexValue = (persons.get(j).glucose / 100) + ((persons.get(j).waist / persons.get(j).hip) * 2) + ((persons.get(j).weight / persons.get(j).height) / 3);
    indexValues.add(indexValue);
}

double currentCaseIndexValue = (currentCaseStabilizedGlucose / 100) + ((currentCaseWaistSize / currentCaseHipSize) * 2) + ((currentCaseWeight / currentCaseHeight) / 3);

int index = -1;
double valueDifference;
if (indexValues.size() > 0) {
    index = 0;
    valueDifference = Math.abs(currentCaseIndexValue - indexValues.get(0));
    for (int j = 1; j < indexValues.size(); j++) {
        if (valueDifference > Math.abs(currentCaseIndexValue - indexValues.get(j))) {
            valueDifference = Math.abs(currentCaseIndexValue - indexValues.get(j));
            index = j;
        }
    }
}

if (persons.get(index).decision == false) {
    word8.set("This person doesn't have diabetes. The closest case index is: ");
} else {
    word8.set("This person has diabetes. The closest case index is: ");
}

output.collect(word8, new IntWritable(index + 1));
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        int closest = values.next().get();
        while (values.hasNext()) {
            if (closest > values.next().get())
                closest = values.next().get();
        }
        output.collect(key, new IntWritable(closest));
    }
}

public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(DiabetesCBR.class);
    conf.setJobName("perfectweight");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    conf.setNumMapTasks(2);
    conf.setNumReduceTasks(1);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));
    JobClient.runJob(conf);
}
2 – a) Hadoop configuration instructions for one node cluster

# website used

# install ubuntu
# connect as normal user
# go to terminal
sudo su -
# give normal password
passwd
# give new password for root
# switch user to root
# go to synaptic package manager
# search for jdk
# install the jdk package
# go to synaptic package manager
# search for ssh
# install the openssh-server package
# go to terminal
ssh-keygen -t rsa -P ""
# press enter
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
ssh localhost
yes
# in /etc/modprobe.d/blacklist add
      # disable IPv6
    blacklist ipv6
# reboot and log in to root
# download hadoop
# open folder /usr/local and move the hadoop package to it
# open terminal
cd /usr/local
sudo tar xzf hadoop-0.20.2.tar.gz
sudo mv hadoop-0.20.2 hadoop
# go to /usr/local/hadoop/conf/hadoop-env.sh and change line 1 to 2
      # export JAVA_HOME=/usr/lib/j2sdk1.5-sun
    export JAVA_HOME=/usr/lib/jvm/java-6-sun
# create folder hadoop-datastore in /usr/local
# add this between the configuration brackets in /usr/local/hadoop/conf/core-site.xml
<!-- In: conf/core-site.xml -->
<property>
  <name>hadoop.tmp.dir</name>
  <value>/usr/local/hadoop-datastore/hadoop-${user.name}</value>
  <description>A base for other temporary directories.</description>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.</description>
</property>

# add this between the configuration brackets in /usr/local/hadoop/conf/mapred-site.xml
<!-- In: conf/mapred-site.xml -->
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.</description>
</property>

# add this between the configuration brackets in /usr/local/hadoop/conf/hdfs-site.xml
<!-- In: conf/hdfs-site.xml -->
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication. The actual number of replications can be specified when the file is created. The default is used if replication is not specified in create time.</description>
</property>

# open terminal
/usr/local/hadoop/bin/hadoop namenode -format
/usr/local/hadoop/bin/start-all.sh
/usr/local/hadoop/bin/stop-all.sh
# download files from website links (outline of science, notebooks, ulysses)
# create folder gutenberg in /tmp and move the three files to it
/usr/local/hadoop/bin/start-all.sh
cd /usr/local/hadoop
bin/hadoop dfs -copyFromLocal /tmp/gutenberg gutenberg
# you should see 1 item found after this
bin/hadoop dfs -ls
# you should see 3 items found after this
bin/hadoop dfs -ls gutenberg
# this command will run the wordcount example
2 – b) Hadoop configuration instructions for two node cluster

# do the one node instructions for each computer
# make sure to stop all processes running in both computers
/usr/local/hadoop/bin/stop-all.sh
# go to /etc/hosts, remove the localhost ip, put instead the real ips on the network like this
   a.b.c.d master
e.f.g.h slave
# go to /usr/local/hadoop/config/core-site.xml in both computers. Change the word localhost to master in both of them
# go to /usr/local/hadoop/config/mapred-site.xml in both computers. Change the word localhost to master in both of them
# go to /usr/local/hadoop/config/hdfs-site.xml in both computers. Change the value from 1 to 2
# go to /usr/local/hadoop/config/masters in the master computer. Change localhost to master
# go to /usr/local/hadoop/config/slaves in the master computer. Change localhost to master
   slave
# copy the public key from the master computer to the slave one by typing this on the master's terminal
   scp $HOME/.ssh/id_rsa.pub root@slave:$HOME/.ssh/master-key.pub
# move the key to the authorized keys of the slave by typing this on the slave's terminal
cat $HOME/.ssh/master-key.pub >> $HOME/.ssh/authorized_keys
# On both computers, go to /usr/local/hadoop-datastore/root and delete everything inside
# on both computers, format the namenode by typing this in the terminal
/usr/local/hadoop/bin/hadoop namenode -format
# on master computer, start HDFS daemons. this should start the datanode on the slave computer.
You can check using the jps command
   /usr/local/hadoop/bin/start-dfs.sh
# on master computer, start MapReduce daemons. this should start the tasktracker on the slave computer.
You can check using the jps command
### 3) Table 1: Diabetes Dataset

<table>
<thead>
<tr>
<th>Stabilized Glucose</th>
<th>Cholesterol/HDL Ratio</th>
<th>Height in inches</th>
<th>Weight in pounds</th>
<th>Waist size in inches</th>
<th>Hip size in inches</th>
<th>Glycosolated Hemoglobin</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>3.5999999905</td>
<td>62</td>
<td>121</td>
<td>29</td>
<td>38</td>
<td>4.309999943</td>
</tr>
<tr>
<td>97</td>
<td>6.900000095</td>
<td>64</td>
<td>218</td>
<td>46</td>
<td>48</td>
<td>4.440000057</td>
</tr>
<tr>
<td>92</td>
<td>6.199999809</td>
<td>61</td>
<td>256</td>
<td>49</td>
<td>57</td>
<td>4.639999866</td>
</tr>
<tr>
<td>93</td>
<td>6.5</td>
<td>67</td>
<td>119</td>
<td>33</td>
<td>38</td>
<td>4.630000114</td>
</tr>
<tr>
<td>90</td>
<td>8.899999619</td>
<td>68</td>
<td>183</td>
<td>44</td>
<td>41</td>
<td>7.71999979</td>
</tr>
<tr>
<td>94</td>
<td>3.599999905</td>
<td>71</td>
<td>190</td>
<td>36</td>
<td>42</td>
<td>4.80999943</td>
</tr>
<tr>
<td>92</td>
<td>4.800000191</td>
<td>69</td>
<td>191</td>
<td>46</td>
<td>49</td>
<td>4.840000153</td>
</tr>
<tr>
<td>87</td>
<td>3.599999905</td>
<td>69</td>
<td>166</td>
<td>34</td>
<td>40</td>
<td>4.840000153</td>
</tr>
<tr>
<td>89</td>
<td>6.599999905</td>
<td>63</td>
<td>202</td>
<td>45</td>
<td>50</td>
<td>5.78000021</td>
</tr>
<tr>
<td>82</td>
<td>4.5</td>
<td>65</td>
<td>156</td>
<td>39</td>
<td>45</td>
<td>4.76999981</td>
</tr>
<tr>
<td>71</td>
<td>5</td>
<td>70</td>
<td>180</td>
<td>39</td>
<td>40</td>
<td>9.61999986</td>
</tr>
<tr>
<td>119</td>
<td>9.399999619</td>
<td>66</td>
<td>179</td>
<td>37</td>
<td>42</td>
<td>7.510000229</td>
</tr>
<tr>
<td>79</td>
<td>4</td>
<td>59</td>
<td>165</td>
<td>37</td>
<td>43</td>
<td>4.590000153</td>
</tr>
<tr>
<td>76</td>
<td>6.40000095</td>
<td>69</td>
<td>183</td>
<td>36</td>
<td>40</td>
<td>4.67000076</td>
</tr>
<tr>
<td>83</td>
<td>4.5</td>
<td>65</td>
<td>157</td>
<td>37</td>
<td>41</td>
<td>3.41000086</td>
</tr>
<tr>
<td>78</td>
<td>6.699999809</td>
<td>65</td>
<td>183</td>
<td>37</td>
<td>43</td>
<td>4.32999924</td>
</tr>
<tr>
<td>112</td>
<td>3.599999905</td>
<td>67</td>
<td>159</td>
<td>31</td>
<td>39</td>
<td>4.53000021</td>
</tr>
<tr>
<td>206</td>
<td>4.800000191</td>
<td>65</td>
<td>196</td>
<td>46</td>
<td>51</td>
<td>11.23999977</td>
</tr>
<tr>
<td>97</td>
<td>3.70000048</td>
<td>67</td>
<td>178</td>
<td>42</td>
<td>41</td>
<td>6.48999971</td>
</tr>
<tr>
<td>65</td>
<td>3.099999905</td>
<td>67</td>
<td>230</td>
<td>45</td>
<td>46</td>
<td>4.67000076</td>
</tr>
<tr>
<td>299</td>
<td>4.699999809</td>
<td>69</td>
<td>288</td>
<td>48</td>
<td>55</td>
<td>12.7399977</td>
</tr>
<tr>
<td>330</td>
<td>7.800000191</td>
<td>74</td>
<td>191</td>
<td>39</td>
<td>41</td>
<td>15.5200046</td>
</tr>
<tr>
<td>206</td>
<td>4.199999809</td>
<td>69</td>
<td>214</td>
<td>45</td>
<td>48</td>
<td>7.90999847</td>
</tr>
<tr>
<td>223</td>
<td>5.300000191</td>
<td>67</td>
<td>196</td>
<td>42</td>
<td>43</td>
<td>10.8999962</td>
</tr>
<tr>
<td>111</td>
<td>5.800000191</td>
<td>60</td>
<td>220</td>
<td>51</td>
<td>54</td>
<td>9.39000343</td>
</tr>
<tr>
<td>106</td>
<td>3.099999905</td>
<td>68</td>
<td>274</td>
<td>49</td>
<td>58</td>
<td>6.34999905</td>
</tr>
<tr>
<td>128</td>
<td>3.299999952</td>
<td>68</td>
<td>180</td>
<td>38</td>
<td>41</td>
<td>5.19999809</td>
</tr>
<tr>
<td>94</td>
<td>2.400000095</td>
<td>62</td>
<td>217</td>
<td>51</td>
<td>51</td>
<td>4.98000019</td>
</tr>
<tr>
<td>233</td>
<td>4.099999905</td>
<td>62</td>
<td>189</td>
<td>43</td>
<td>47</td>
<td>13.6999981</td>
</tr>
<tr>
<td>262</td>
<td>4.900000095</td>
<td>63</td>
<td>183</td>
<td>42</td>
<td>48</td>
<td>10.93000031</td>
</tr>
</tbody>
</table>
185 | 19.29999924 | 70 | 235 | 43 | 48 | 14.31000042
203 | 2.799999952 | 63 | 165 | 34 | 42 | 11.40999985
131 | 2.5 | 61 | 102 | 31 | 33 | 4.440000057
184 | 2 | 61 | 127 | 35 | 38 | 8.399999619
113 | 3.20000048 | 60 | 143 | 35 | 40 | 7.440000057
108 | 4.599999905 | 62 | 183 | 37 | 45 | 5.46999979
105 | 3.20000048 | 70 | 173 | 36 | 40 | 4.28999962
81 | 3.599999905 | 71 | 223 | 46 | 50 | 3.930000067
94 | 7 | 63 | 154 | 38 | 41 | 6.960000038
88 | 5.400000095 | 73 | 219 | 41 | 42 | 4.840000153
103 | 6.800000191 | 67 | 169 | 40 | 44 | 5.179999828
87 | 5.699999809 | 73 | 200 | 38 | 41 | 5.51999981
96 | 4 | 64 | 200 | 38 | 45 | 4.380000114
84 | 3.599999905 | 61 | 145 | 37 | 40 | 5.28000021
85 | 4.5 | 67 | 178 | 37 | 41 | 4.820000172
182 | 4 | 61 | 215 | 42 | 53 | 10.55000019
197 | 5.699999809 | 63 | 159 | 33 | 41 | 9.17000076
120 | 6.300000191 | 63 | 156 | 35 | 39 | 7.89999962
91 | 3.900000095 | 65 | 220 | 40 | 50 | 5.150000095
341 | 6.800000191 | 67 | 197 | 42 | 43 | 10.14999962
69 | 3.700000048 | 63 | 200 | 38 | 46 | 4.17000076
79 | 4.699999809 | 69 | 154 | 34 | 39 | 4.050000191
130 | 4.099999905 | 61 | 203 | 42 | 51 | 5.440000057
91 | 3.299999952 | 71 | 180 | 36 | 40 | 4.30999943
176 | 3.799999952 | 61 | 150 | 36 | 39 | 9.770000458
235 | 5.300000191 | 70 | 165 | 39 | 41 | 7.86999986
108 | 4.199999809 | 67 | 172 | 38 | 43 | 8.449999809
70 | 3.400000095 | 72 | 170 | 31 | 39 | 3.980000019
119 | 3.900000095 | 63 | 158 | 34 | 42 | 9.760000229
177 | 7.5 | 71 | 225 | 44 | 47 | 7.53000021
110 | 3.099999905 | 74 | 151 | 34 | 38 | 6.13000114
122 | 3.900000095 | 67 | 223 | 43 | 48 | 6.489999771
196 | 4.800000191 | 71 | 266 | 49 | 45 | 7.510000229
48 | 6.199999809 | 67 | 177 | 38 | 43 | 6.96999979
112 | 3 | 66 | 170 | 37 | 40 | 9.180000305
255 | 7 | 71 | 320 | 56 | 49 | 9.57999924
239 | 3.20000048 | 69 | 300 | 30 | 37 | 13.6000038
121 | 6.5 | 68 | 167 | 36 | 39 | 9.25
110 | 6 | 67 | 282 | 52 | 59 | 9.819999695
85 | 3.400000095 | 69 | 171 | 38 | 40 | 4.960000038
385 | 4.5 | 69 | 172 | 37 | 41 | 11.59000015
248 | 8 | 66 | 189 | 38 | 45 | 7.139999866
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>133</td>
<td>7.900000095</td>
<td>59</td>
<td>204</td>
<td>40</td>
<td>47</td>
</tr>
<tr>
<td>104</td>
<td>4</td>
<td>66</td>
<td>189</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>145</td>
<td>7.300000191</td>
<td>61</td>
<td>200</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>174</td>
<td>7.800000191</td>
<td>68</td>
<td>210</td>
<td>45</td>
<td>48</td>
</tr>
<tr>
<td>225</td>
<td>7.300000191</td>
<td>63</td>
<td>144</td>
<td>40</td>
<td>42</td>
</tr>
<tr>
<td>251</td>
<td>7.099999905</td>
<td>65</td>
<td>201</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>236</td>
<td>2.799999952</td>
<td>75</td>
<td>186</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>187</td>
<td>4.5</td>
<td>67</td>
<td>201</td>
<td>46</td>
<td>49</td>
</tr>
<tr>
<td>89</td>
<td>3.599999905</td>
<td>68</td>
<td>196</td>
<td>38</td>
<td>42</td>
</tr>
<tr>
<td>84</td>
<td>4.099999905</td>
<td>63</td>
<td>153</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>77</td>
<td>3.599999905</td>
<td>68</td>
<td>170</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>100</td>
<td>2.900000095</td>
<td>66</td>
<td>188</td>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>228</td>
<td>3.299999952</td>
<td>65</td>
<td>198</td>
<td>40</td>
<td>49</td>
</tr>
<tr>
<td>79</td>
<td>6.199999809</td>
<td>72</td>
<td>259</td>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>74</td>
<td>4.400000095</td>
<td>66</td>
<td>200</td>
<td>41</td>
<td>47</td>
</tr>
<tr>
<td>270</td>
<td>7</td>
<td>68</td>
<td>224</td>
<td>48</td>
<td>50</td>
</tr>
<tr>
<td>120</td>
<td>2.900000095</td>
<td>62</td>
<td>147</td>
<td>34</td>
<td>42</td>
</tr>
<tr>
<td>369</td>
<td>6.400000095</td>
<td>69</td>
<td>173</td>
<td>35</td>
<td>39</td>
</tr>
<tr>
<td>89</td>
<td>4.900000095</td>
<td>67</td>
<td>160</td>
<td>30</td>
<td>44</td>
</tr>
<tr>
<td>83</td>
<td>2.900000095</td>
<td>65</td>
<td>120</td>
<td>29</td>
<td>40</td>
</tr>
<tr>
<td>206</td>
<td>12.19999981</td>
<td>69</td>
<td>159</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>118</td>
<td>5.300000191</td>
<td>64</td>
<td>174</td>
<td>38</td>
<td>44</td>
</tr>
<tr>
<td>90</td>
<td>5.199999809</td>
<td>63</td>
<td>252</td>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>79</td>
<td>3.599999905</td>
<td>60</td>
<td>135</td>
<td>33</td>
<td>40</td>
</tr>
<tr>
<td>225</td>
<td>5.599999905</td>
<td>73</td>
<td>260</td>
<td>42</td>
<td>47</td>
</tr>
<tr>
<td>92</td>
<td>6</td>
<td>63</td>
<td>179</td>
<td>40</td>
<td>42</td>
</tr>
<tr>
<td>109</td>
<td>4</td>
<td>63</td>
<td>290</td>
<td>55</td>
<td>62</td>
</tr>
</tbody>
</table>