Assignment-3 : Click Distance

`MTCS-202(P) - Hadoop Assignment`

`Aryan Sai Arvapelly, Regd. No - 23352, I MTech CS`

Problem

The problem is to process a dataset containing web page links. A sample dataset is:

WS1,WS3
WS1,WS5
WS2,WS1
WS3,WS4
WS4,WS6

Each record in the dataset represents a link from one web page (source) to another (destination). The task is to determine the click distance between these web pages and output records based on the specified click distance, X.

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputForma
t;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFor
mat;
import java.util.ArrayList;
import java.util.List;

public class CD {
    public static class CDMapper extends Mapper<Object, Tex
t, Text, Text> {
```

```
        public void map(Object key, Text value, Context con
text) throws IOException, InterruptedException {
            String[] tokens = value.toString().split(",");
            String source = tokens[0].trim();
            String dest = tokens[1].trim();
            context.write(new Text(source), new Text(dest
+",0"));
            context.write(new Text(dest), new Text(source
+",1"));
        }
    }

    public static class CDReducer extends Reducer<Text, Tex
t, Text, Text> {
        public void reduce(Text key, Iterable<Text> values,
Context context) throws IOException, InterruptedException {
            int count0 = 0, count1 = 0;
            List<String> to = new ArrayList<>();
            List<String> from = new ArrayList<>();

            for (Text value : values) {
                String[] tokens = value.toString().split
(",");
                if(tokens[1].equals("1")){
                    count1++;
                    from.add(tokens[0]);
                } else{
                    count0++;
                    to.add(tokens[0]);
                }
            }
            if(count0 >= 1 && count1 >= 1){
                for(String w: from){
                    for(String ws: to){
                        context.write(new Text(w+","),new T
ext(ws));
                    }
                }
```

```
            }
        }
    }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("clickDistance", args[0]);
    int clickDistance = Integer.parseInt(args[0]);

    for (int i = 1; i < clickDistance; i++) {
        Job job = Job.getInstance(conf, "click distance " +
(i + 1));
        job.setJarByClass(CD.class);
        job.setMapperClass(CDMapper.class);
        job.setReducerClass(CDReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        if (i == 1) {
            FileInputFormat.addInputPath(job, new Path(args
[1]));
        } else {
            FileInputFormat.addInputPath(job, new Path(args
[2] + "_" + i));
        }
        FileOutputFormat.setOutputPath(job, new Path(args
[2] + "_" + (i + 1)));

        job.waitForCompletion(true);
    }
  }
}
```

Intuition

Mapper Class:

- The first key-value pair has the source web page as the key and the destination web page followed by ",0" as the value. Here, ",0" indicates that it is an outgoing link.

- The second key-value pair has the destination web page as the key and the source web page followed by ",1" as the value. Here, ",1" indicates that it is an incoming link.

Reducer Class:

- Inside the `reduce` method, two counters (`count0` and `count1`) are initialized to track the number of outgoing (count0) and incoming (count1) links associated with the source web page.
- Two lists (`to` and `from`) are initialized to store destination web pages reachable directly (to) and indirectly (from) from the source web page.
- The method iterates over the values associated with the key. For each value:
  - It splits the value into tokens using a comma (`,`) delimiter.
  - If the second token equals "1", it increments the count1 counter and adds the corresponding destination web page to the `from` list. Otherwise, it increments the count0 counter and adds the destination web page to the `to` list.
- If both outgoing and incoming links exist (`count0 >= 1 && count1 >= 1`), it generates output records representing connections between web pages. It iterates over the `from` and `to` lists, emitting key-value pairs where the key is the source web page followed by a comma (`,`) and an empty string (indicating a connection), and the value is the destination web page.
- The `context.write()` method is used to write the key-value pairs to the context, which sends them to the Hadoop framework for further processing.

Main Function

- It parses the command-line arguments. The first argument (`args[0]`) is assumed to be the click distance (X). It sets this value in the configuration using `conf.set("clickDistance", args[0])`.
- **Iterative Job Execution**: It iterates over click distance levels from 1 to X (`for (int i = 1; i < clickDistance; i++)`). For each level:
  - It creates a new Hadoop job instance (`Job job = Job.getInstance(conf, "click distance " + (i + 1))`).
  - It sets the main class for the job (`job.setJarByClass(CD.class)`).

- It sets the Mapper and Reducer classes for the job (`job.setMapperClass(CDMapper.class)` and `job.setReducerClass(CDReducer.class)`).
- It specifies the output key and value classes (`job.setOutputKeyClass(Text.class)` and `job.setOutputValueClass(Text.class)`).
- It sets the input path for the job. For the first level (i=1), it uses the input path specified in `args[1]`. For subsequent levels, it uses the output path of the previous level (`args[2] + "_" + i`).
- It sets the output path for the job to be `args[2] + "_" + (i + 1)`.
- It waits for the job to complete (`job.waitForCompletion(true)`).

- The loop iterates over click distance levels, executing MapReduce jobs to compute click distances iteratively. Each job processes the output of the previous job until the desired click distance level is reached.

Example:

Input:

WS1,WS2
WS2,WS3
WS2,WS4
WS4,WS6
WS5,WS6
WS6,WS7
WS6,WS8
WS4,WS8
WS3,WS9
WS6,WS9

Click Distance-2

Click Distance-3