#1 Hadoop

#2 Multi join

#3 Mapper(Pattern Matcher)

```java

public class MultiJoinMapperImpl extends MultiJoinMapper{

    @Override
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException{
//        System.out.println("the key is :" + key);
        System.out.println(" the value is : " + value);
        String _key = "";
        String _value = "";
        String flg_1 = ""; // flag of whether this file is file 1,
        // if yes, then this attribute is 1, otherwise is 0.
        String[] str = value.toString().split("\\s+");
        System.out.println(str[0]);
        System.out.println(str[1]);
//        StringTokenizer itr = new StringTokenizer(value.toString());
//        List<String> str = new ArrayList<>();
//        // Since we already known that the itr has 2 part: [companyname, addressid]/[addressid, addressname]
//        //str.add(itr.nextToken());
//        while(itr.hasMoreTokens()) {
//            String tempt_str = itr.nextToken();
//            str.add(Pattern.compile("\\W+").matcher(tempt_str).replaceAll(""));
//        }
//        str[1] = itr.nextToken();
//        str[1] = Pattern.compile("\\W+").matcher(str[1]).replaceAll("");
        //if(Character.isDigit(str[0])  || str[0].equals("addressid")){
        Pattern pattern = Pattern.compile("^[-\\+]?[\\d]*$");
        if(str[0].equals("addressid") | pattern.matcher(str[0]).matches()){
            flg_1 = "0";
            _key = str[0];
            _value = str[1];
            //}else if(Character.isDigit(str[1].CharAt(1)) || str[1].equals("addressid")){
        }else if(str[1].equals("addressid") | pattern.matcher(str[1]).matches()){
            flg_1 = "1";
            _key = str[1];
            _value = str[0];
        }
        System.out.println(_key + ":" + _value + ":" + flg_1);
        context.write(new Text(_key) , new Text(_value + "\t" + flg_1));

    }

}
```

#3 Reducer(list array)

```java
public class MultiJoinReducerImpl extends MultiJoinReducer {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException{
        Iterator<Text> itr = values.iterator();
        List<String> companyname = new ArrayList<String>();
        int i = 0;
        String placename = "";
        while(itr.hasNext()){
            String[] tempt = itr.next().toString().split("\\W+");
            if(tempt[1].equals("1")){
                companyname.add(tempt[0]);
            }else if(tempt[1].equals("0")){
                placename = tempt[0];
            }
        }
        if (placename.equals("")){
            return;
        }
        for(int j = 0; j < companyname.size(); j = j + 1){
            context.write(null, new Text(companyname.get(j) + "\t" +
placename));
        }

    }

}
```

Select

Mapper(Pattern.compile().matcher())

```
public class SelectMapperImpl extends SelectMapper{
    @Override
    public void map(Object key, Text value, Mapper.Context context)
            throws IOException, InterruptedException{
        System.out.println(key);
        System.out.println(value);
        StringTokenizer itr = new StringTokenizer(value.toString());
        String[] str = null;
        str[0] = itr.nextToken();
        str[0] = Pattern.compile("\\W+").matcher(str[0]).replaceAll("");
        str[1] = itr.nextToken();
        str[1] = Pattern.compile("\\W+").matcher(str[1]).replaceAll("");
        str[2] = itr.nextToken();
        str[2] = Pattern.compile("\\W+").matcher(str[2]).replaceAll("");
        // id, name, city
        String _key = str[2];
        String _value = str[0] + "\t" + str[1];
        context.write(new Text(_key), new Text(_value));
    }
}

```

Reducer

```
public class SelectReducerImpl extends SelectReducer{
    @Override
    public void reduce(Text key, Iterable<NullWritable> values, Context
context)
            throws IOException, InterruptedException{
        if(key.equals("shanghai")){
            Iterator<NullWritable> itr = values.iterator();
            while(itr.hasNext()){
                //System.out.println(itr.next().toString() + "\t" + key);
                context.write(null,NullWritable.get());
            }
        }else{
            return;
        }

    }
}

```

ssp(context)

Mapper

```
public class SimpleShortestPathsMapperImpl extends SimpleShortestPathsMapper {

    /**
     * TODO 请完成该函数
     * -
     * 1. 填写默认最短路径距离
     * 2. 计算当前节点经过 所有已有临时最短路径的节点 到A节点的 所有路径距离
     */
    @Override
    public void map(Text key, Text value, Context context)
            throws IOException, InterruptedException{
        System.out.println("the key is :" + key);
        System.out.println("The value is :" + value);
        Text _key = new Text(key);
        Text _value = new Text();

        String[] str = value.toString().split("\\t+");
//        Pattern pattern = Pattern.compile("^[-\\+]?[\\d]*$");
        int length = str.length;
//        String[] neighbours = new String[length - 1];
//        System.arraycopy(str, 1, neighbours, 0, length -1);
        Node node = new Node();
//        Boolean flg = isInteger(0);
        if(!(StringUtils.isNumeric(str[0]) | str[0].equals("inf"))){
            if(key.toString().equals("A")){
                context.write(_key, new Text("0"));
                _value.set("inf" + "\t" + value.toString());
            }else{
                _value.set("inf" + "\t" + value.toString());
            }
            context.write(_key, _value);
        }else{
            node.FormatNode(value.toString());
            if(node.getDistance().equals("inf")){
                _value.set(value);
                System.out.println("111");
                context.write(_key, _value);
                return;
            }
            int nodeNum = node.getNodeNum();
            for(int i = 0; i < nodeNum; i++){
                String target = node.getNodeKey(i);
                int distance = Integer.parseInt(node.getDistance()) + Integer.parseInt(node.getNodeValue(i));
                System.out.println(target);
                System.out.println(distance);
                context.write(new Text(target), new
Text(String.valueOf(distance)));
            }
            _value.set(value);
            context.write(_key, _value);
        }



    }

}
```

#3 **Reducer**

```
public class SimpleShortestPathsReducerImpl extends
SimpleShortestPathsReducer {

    /**
     * TODO 请完成该函数
     * -
     * 修改每个节点的最短路径距离
     * 每次迭代都要修改，直到所有节点的最短路径距离不再发生改变
     * {B, {10 (C,1) (D,2)}, {8}, {12}}   =>  B, 8 (C,1) (D,2)
     * isChange: Node node, String min, Context context => void
     */
    @Override
     public void reduce(Text nodeKey, Iterable<Text> values, Context
context)
            throws IOException, InterruptedException{
        System.out.println("the reducer's key is :" + nodeKey.toString());
        System.out.println("2222");
        //System.out.println("the reducer's value is :" + values);
        Iterator itr = values.iterator();
        //String target_node = itr.next().toString();
        //String value_inf = itr.next().toString();
        String min = INF;
        String dis = INF;
        Node node = new Node();
        //node.FormatNode(value_inf);
        while(itr.hasNext()){

            dis = itr.next().toString();
            String[] flg = dis.split("\\t+");
            if(flg.length > 1){
                node.FormatNode(dis);
            }else if(dis.equals(INF)){
                ;
            }else if(min.equals(INF)){
                min = dis;
            }
            else if(Integer.parseInt(dis) < Integer.parseInt(min)){
                min = dis;
            }
        }
        isChange(node, min, context);
        if(min.equals(INF)){
            ;
        }else if(node.getDistance().equals(INF)) {
            node.setDistance(min);
        }else if(Integer.parseInt(min) <
Integer.parseInt(node.getDistance())){
            node.setDistance(min);
        }


        context.write(nodeKey, new Text(node.toString()));



    }

}

```

Spark

Broadcast join

Broadcast

```
Broadcast<Map<Long, String>> persons;

public class BroadcastJoinMapperImpl extends BroadcastJoinMapper {

    /**
     * 用于存储广播变量. Map 中的键是 Person 的 Id_P, 值是对应的 LastName 和 FirstName, 由 "," 分隔
     * (如 键: 1, 值: "Adams,John")
     */
//    Broadcast<Map<Long, String>> persons;

//    public void setPersons(Broadcast<Map<Long, String>> persons) {
//        this.persons = persons;
//    }

    /**
     * TODO 请完成该函数
     *
     * 根据输入变量 order 和广播变量 persons, 计算有关该 order 的所有连接结果
     *
     * @param order 一个 Order 记录, 各字段由 "," 分隔 (如 "1,77895,3")
     * @return 返回该条 Order 记录的所有连接结果, 其中每条字符串代表一个连接记录, 各字段由 "," 分隔 (如 "Adams,John,24562")
     */
    @Override
    public Iterator<String> call(String order){
        String[] order_infs = order.split("\\W+");
        Long order_idx = Long.parseLong(order_infs[2]);
        String order_inf = order_infs[1];
        String person_inf = persons.getValue().get(order_idx);
        String inf = null;
        List<String> list = new ArrayList<>();
        System.out.println(inf);
        if(person_inf==null){
            System.out.println("wsnb");
        }else{
             inf = person_inf + "," + order_inf;
             list.add(inf);
        }
        return list.iterator();
    }

}

```

#2 Shufflejoin(map、join)

```
public class ShuffleJoinImpl extends ShuffleJoin  {

    /**
     * TODO 请完成该函数
     *
     * 连接 Persons 表和 Orders 表
     *
     * @param personRdd Person 数据，键为 Id_P，值为 LastName 和 FirstName，由
"," 分隔 (如 键: 1, 值: "Adams,John")
     * @param orderRdd Order 数据，键为 Id_P，值为 OrderNo (如 键: 1, 值:
"22456")
     * @return 返回代表连接结果的 RDD，字段间由 "," 分隔 (如 "Adams,John,24562")
     */
    public JavaRDD join(JavaPairRDD<Long, String> personRdd,
JavaPairRDD<Long, String> orderRdd){
//          List<Tuple2<Long, String>> personlist = personRdd.collect();
//          List<Tuple2<Long, String>> orderlist = orderRdd.collect();
//          for(int i = 0; i < orderlist.size() ;i++){
//              personlist.add(orderlist[i]);
//          }
//          JavaPairRDD<Long, String> inf_ = personRdd.mapToPair((Tuple2<Long,
String> person)->{
        JavaPairRDD<Long, Tuple2<String, String>> joinResult =
personRdd.join(orderRdd);
        JavaRDD<String> result = joinResult.map((Tuple2<Long, Tuple2<String,
String>> element) -> {
            return element._2._1 + "," + element._2._2;
        });
        return result;
    }

}

```

#2 Pagerank(iterator -> list)

```
public class CalculateRankImpl extends CalculateRank {

    /**
     * 公式中的 q
     * final static Double FACTOR = 0.85;
     */


    /**
     * TODO 请完成该函数
     *
     * 计算新的 rank 值
     *
     * @param weight (节点 ID, 该节点所有入边传递来的权值) 键值对
```

```java
     * @return (节点 ID, 该节点新的 rank 值) 键值对
     */
    @Override
    public Tuple2<String, Double> call(Tuple2<String, Iterable<Double>> weight) throws Exception{
        Iterator itr = weight._2.iterator();
        List<Double> weight_list = IteratorUtils.toList(itr);
        Double sum = 0.0;
        for(int i = 0; i < weight_list.size(); i++){
            sum += weight_list.get(i);
        }
        return new Tuple2<String, Double>(weight._1, sum*FACTOR + (1 - FACTOR));
    }

}

```

```java
public class FlatMapToPairImpl extends FlatMapToPair {

    /**
     * TODO 请完成该函数
     *
     * 生成 (节点 ID, 某一出边对其影响) 键值对
     *
     * @param outsideWeight (一个节点所有出边指向的节点 ID, 该节点当前的 rank 值) 键值对
     * @return (出边指向的节点 ID, 出边传递出去的 rank 值) 键值对
     */
    @Override
    public Iterator<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> outsideWeight) throws Exception{
        System.out.println(outsideWeight);
        Iterator<String> itr = outsideWeight._1.iterator();
        List<String> inf = IteratorUtils.toList(itr);
        Double rank = outsideWeight._2 / inf.size();
        List<Tuple2<String, Double>> out = new ArrayList();
        for(int i = 0; i < inf.size(); i++){
            out.add(i, new Tuple2<String, Double>(inf.get(i), rank));
        }
        return out.iterator();
    }

}

```

Storm

Window join(write into file)

```java
public class PrinterBoltImpl extends PrinterBolt{


    public PrinterBoltImpl(String outputFile) {
        super(outputFile);
    }

    @Override
    public String parseTuple(Tuple tuple){
//        System.out.println("[" + tuple.getInteger(0) + ", " + tuple.getString(1) + ", " + tuple.getInteger(2) + "]");
        return "[" + tuple.getInteger(0) + ", " + tuple.getString(1) + ", " + tuple.getInteger(2) + "]\n";
    }

    @Override
    public void saveResult(String outputFile, String result){
        BufferedWriter bw = FileProcess.getWriter(outputFile);
        FileProcess.write(result, bw);
        FileProcess.close(bw);
    }
}

```

```java
public class StormJoinBoltImpl extends StormJoinBolt {
    public void  setJoinBolt(){

    }

    public JoinBolt getJoinBolt(){
        JoinBolt joinBolt = new JoinBolt("ageSpout", "id")
                .join("genderSpout", "id", "ageSpout")
                .select("id, gender, age")   // chose output fields
                .withTumblingWindow(new BaseWindowedBolt.Duration(2, TimeUnit.SECONDS));
        return joinBolt;
    }
}
```

#2 SlidesCountWindow

```java
public class SlideCountWindowBoltImpl extends SlideCountWindowBolt {
    /**
     * TODO: 实现此方法每次接收一个Tuple e.g. (a 1)将此tuple放入相应得窗口
     *       同一个key的Tuple每出现两次，对此key最近出现的三个元素进行一次计算 这里为append计算即
     *       (a 1) + (a 2) + (a 3) = (a 123)
     *  注意:emit操作使用outputFormat简化操作 e.g:
     *  collect.emit(new Value(outputFormat(key, value, windowNum))
     *
     **/
    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector){
        if(words == null){
            words = new HashMap<>();
            words.put(tuple.getString(0), new ArrayList<String>(){{add("1");add("1");add(tuple.getString(1));}});
            return;
        }


        String word = tuple.getString(0);
        String tmp_val = tuple.getString(1);
        if(words.get(word) == null){
            words.put(word, new ArrayList<String>(){{add("1");add("1");add(tmp_val);}});
            return;
        }
//        if()
        ArrayList<String> tmp_values = words.get(word);

//        if(Integer.parseInt(tmp_values.get(1)) < 2){
            tmp_values.add(tmp_val);
            int window_count = Integer.parseInt(tmp_values.get(1));
            tmp_values.remove(1);
            tmp_values.add(1, String.valueOf(window_count + 1));
//        }else{
            int length = tmp_values.size();
            if(Integer.parseInt(tmp_values.get(1)) == 2) {
                String out_val = "";
                if (Integer.parseInt(tmp_values.get(0)) > 1) {
                    out_val += tmp_values.get(length - 3) + tmp_values.get(length - 2) + tmp_values.get(length - 1);
                } else {
                    out_val += tmp_values.get(length - 2) + tmp_values.get(length - 1);
                }
                basicOutputCollector.emit(new Values(outputFormat(word, out_val, tmp_values.get(0))));

                int window_idx = Integer.parseInt(tmp_values.get(0));
                tmp_values.remove(0);
                tmp_values.add(0, String.valueOf(window_idx + 1));
                window_count = Integer.parseInt(tmp_values.get(1));
                tmp_values.remove(1);
                tmp_values.add(1, String.valueOf(0));
            }
    }
}
```

#1 Flink

#2 K-Means

#3 声明

```
public FilterOperator<Tuple2<Tuple3<Integer, Double, Double>, Tuple3<Integer, Double, Double>>> getTerminatedDataSet(DataSet<Centroid> newCentroids, DataSet<Centroid> oldCentroids){
        DataSet<Tuple2<Tuple3<Integer, Double, Double>, Tuple3<Integer, Double, Double>>> ds = newCentroids.join(oldCentroids).where("id").equalTo("id")
                .map(new MapFunction<Tuple2<Centroid, Centroid>, Tuple2<Tuple3<Integer, Double, Double>, Tuple3<Integer, Double, Double>>>() {
                    @Override
                    public Tuple2<Tuple3<Integer, Double, Double>, Tuple3<Integer, Double, Double>> map(Tuple2<Centroid, Centroid> value) throws Exception {
                        return Tuple2.of(Tuple3.of(value.f0.id, value.f0.x, value.f0.y), Tuple3.of(value.f1.id, value.f1.x, value.f1.y));
                    }
                });
        return ds.filter(new FilterFunction<Tuple2<Tuple3<Integer, Double, Double>, Tuple3<Integer, Double, Double>>>() {
            @Override
            public boolean filter(Tuple2<Tuple3<Integer, Double, Double>, Tuple3<Integer, Double, Double>> value) throws Exception {
//                Double delta = value.f0.euclideanDistance(value.f1);
                Double delta = Math.sqrt(Math.pow(value.f0.f1 - value.f1.f1, 2) + Math.pow(value.f0.f2 - value.f1.f2, 2));
                if(delta <= EPSILON){
                    return false;
                }
                return true;
            }
        });
    }
```

#3 使用

```
public class IterationStepImpl extends IterationStep {
    /**
     * TODO://利用已有工具类（k_means->util）实现kmeans运算迭代步
     * @return 返回迭代一次后的中心点坐标
     * @param points 数据点 <x,y> e.g. (32.05 -32.08)
     * @param centroids  中心点 <id, x, y> e.g. (1 30.01 -30.02)
     * */
    public DataSet<Centroid> runStep(DataSet<Point> points, DataSet<Centroid> centroids){
        DataSet<Centroid> tmp_c = points.map(new SelectNearestCenter())
                .withBroadcastSet(centroids, "centroids")
                .map(new CountAppender())
                .groupBy(0)
                .reduce(new CentroidAccumulator())
                .map(new CentroidAverager());
        return tmp_c;
    }

}
```

#2 watermark

```
 public static void run(SourceFunction<Tuple2<Long, Integer>> source, String outputFile) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment().setParallelism(1);
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

        TimestampWithWatermarkAssigner.setMaxOutOfOrder(1000 * 60 * 2);

        env.addSource(source)
                .assignTimestampsAndWatermarks(new TimestampWithWatermarkAssignerImpl())
                .timeWindowAll(Time.minutes(3), Time.minutes(1))
                .apply(new AllWindowFunction<Tuple2<Long, Integer>, Tuple2<String, Integer>, TimeWindow>() {
                    @Override
                    public void apply(TimeWindow window, Iterable<Tuple2<Long, Integer>> tuples,
                                      Collector<Tuple2<String, Integer>> collector) {
                        int sum = 0;
                        for (Tuple2<Long, Integer> tuple : tuples) {
                            sum += tuple.f1;
                        }
                        collector.collect(new Tuple2<>(
                FORMAT.format(window.getStart()) + "-" + FORMAT.format(window.getEnd()), sum));
          }
        })
        .writeAsText(outputFile);

    env.execute();
  }
```

#2 flapMap

```
public void flatMap(Tuple2<String, Integer> tuple, Collector<Tuple2<String, Boolean>> collector)
      throws Exception{
    int original = 0;
    int now = 0;
    if(map.containsKey(tuple.f0)){
      original = map.get(tuple.f0);
      now = original + tuple.f1;
      map.put(tuple.f0, now);
    }else{
      map.put(tuple.f0, tuple.f1);
    }
    System.out.println(tuple.f0 + " : " + map.get(tuple.f0));
    if (original <= THRESHOLD && now > THRESHOLD){
      collector.collect(new Tuple2<String, Boolean>(tuple.f0, true));
    }
    if(original > THRESHOLD && now <= THRESHOLD){
      collector.collect(new Tuple2<String, Boolean>(tuple.f0, false));
    }
  }
```

#2 jsonParser

```
public void flatMap(String value, Collector<Tuple2<String, Integer>> collector) throws Exception{
    jsonParser = new ObjectMapper();
    JsonNode jnode = jsonParser.readTree(value);
    Iterator<Map.Entry<String, JsonNode>> itr = jnode.fields();
    String text_value = "";
    Boolean lang_en = false;

    while(itr.hasNext()){
      Map.Entry<String, JsonNode> t = itr.next();
      String key = t.getKey();
      if(key.equals("text")){
        text_value = t.getValue().asText();
      } else if(key.equals("user")){
        JsonNode user_value = t.getValue();
        Iterator<Map.Entry<String, JsonNode>> user_itr = user_value.fields();
        while(user_itr.hasNext()){
          Map.Entry<String, JsonNode> user_t = user_itr.next();
          if(user_t.getKey().equals("lang") && user_t.getValue().asText().equals("en")){
```

```java
                        lang_en = true;
                    }
                }
            }
        }
        if(lang_en == true && !text_value.isEmpty()){
            String[] texts = text_value.split("\\s+");
            for(String text : texts){
                collector.collect(new Tuple2<String, Integer>(text.toLowerCase(), 1));
            }
        }
        collector.close();
    }
}
```