Report

1. 首先計算初始向量 r_0，輸出格式為 **點 P 初始值**

2. 計算 Matrix：

Map：將 input 檔讀入，把其中連結的起點設為 key，終點為 value。

Reduce：看每個 key 有包含哪些 value，並計算其總和，將其平均後分配其機率。最後輸出為 **起點 M 機率 1 機率 2 .....** 以方便後面的計算。

```
Arrays.fill(M, (float) ((1 - d) / N));
float[] A = new float[N+10];

int sum = 0;
for(Text val : values) {
    int index = Integer.parseInt(val.toString());
    A[index] = 1;
    sum += 1;
}
if (sum == 0) {// 分母不能為0
    sum = 1;
}

StringBuilder sb = new StringBuilder();
for (int i = 0; i < N; i++) {
    sb.append(" " + (float) (M[i] + d * A[i] / sum));
}

Text sum_text = new Text(String.valueOf(sum));

//Text v = new Text(sb.toString().substring(1));
Text v = new Text( key.toString() + " M " + sb.toString().substring(1));
result.set(v);
//Text k = new Text(key.toString() + " M");
context.write(null,result);
```

3. 計算新的向量：

讀入先前所計算的向量及矩陣，計算出新的向量值。

Map：利用讀入內容的 M 及 P 判斷其為矩陣或舊向量的內容。

矩陣：將終點當成 key，起點與機率值當成 value。輸出為 **終點, M 起點 機率**

舊向量：將每個點輪流當成 key，將所有點與其舊的值當成 value。輸出為 **終點, P 起點 舊的機率**

```
if(words[1].equals("M"))
{
    int end = 0;
    while (end < N) {
        String start = words[0];
        k.set(Integer.toString(end));
        String p = words[end+2]; ///到第i個的機率
        v.set( "M " + start + " " + p );
        context.write(k, v);
        end += 1;
    }
}
else if(words[1].equals("P"))
{
    int end = 0;
    while (end < N) {
        k.set(Integer.toString(end));
        v.set("P " + words[0] + " " + words[2]);
        context.write(k, v);
        end += 1;
    }
}
```

```
0       P 3 0.25
0       P 2 0.25
0       P 0 0.25
0       P 1 0.25
0       M 3 0.037499994
0       M 2 0.037499994
0       M 0 0.037499994
0       M 1 0.037499994
1       M 3 0.88750005
1       M 0 0.32083333
1       M 2 0.037499994
1       M 1 0.037499994
1       P 3 0.25
1       P 0 0.25
1       P 2 0.25
1       P 1 0.25
2       P 0 0.25
2       P 3 0.25
2       P 2 0.25
2       P 1 0.25
2       M 0 0.32083333
2       M 3 0.037499994
2       M 2 0.037499994
2       M 1 0.4625
3       M 3 0.037499994
3       M 2 0.88750005
3       M 0 0.32083333
3       M 1 0.4625
3       P 3 0.25
3       P 2 0.25
3       P 0 0.25
3       P 1 0.25
```

(測試範例)

Reduce：利用 M 及 P 判斷其為矩陣或舊向量的內容，並以與 HW1 相似的方式相乘及相加，即可求出新的向量值。輸出時以 **點 P 新的值** 的格式輸出。

```
HashMap<Integer, Float> hashM = new HashMap<Integer, Float>();
HashMap<Integer, Float> hashP = new HashMap<Integer, Float>();
/*
for(Text val : values) {
    String line = val.toString();
    String[] words = line.split(" ");
    if(words[0].equals("M"))
    {
       hashM.put( Integer.parseInt(words[1]), Float.parseFloat(words[2]) ); // 從哪來的，機率**
       /*
       Text v = new Text(line );
       context.write(key, v);
       */
    }
    else if(words[0].equals("P"))
    {
        hashP.put( Integer.parseInt( words[1] ), Float.parseFloat(words[2]) );// **
        /*
        Text v = new Text(line);
       context.write(key, v);
       */
    }
}

float sum = 0;
float m, p;
for(int i=0; i<N; i++){
    m = hashM.containsKey(i) ? hashM.get(i) : 0;
    p = hashP.containsKey(i) ? hashP.get(i) : 0;
    sum += m * p;
}
```

4. Normalize：

用 1 減掉所有點的新的值總和，並將其差值平均加到各個點即可。輸出時以 **點 P 新的值** 的格式輸出，以利下一輪的計算。

```
float[] newP = new float[N];
float sum = 0.0f;
int idx = 0;
for(Text val:values){

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

    float p = Float.parseFloat(words[1].toString());
    idx = Integer.parseInt(words[0]);
    newP[idx] = p;
    sum += p;

}
float left = (float)(1.0-sum)/(float)N;

for(int i=0; i<N; i++){
    newP[i] += left;
    v.set( Integer.toString(i) + " P " + Float.toString(newP[i]) );
    context.write(null, v);
}
```

將第 3 跟第 4 步重複執行 20 次。

5. Top 10：

將最終的結果全部 scan 過一次，找出值最大的 10 個點即可。

```
int[] topNode = new int[10];
float[] topP = new float[10];
for(Text val:values){

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

    float p = Float.parseFloat(words[1].toString());
    int idx = Integer.parseInt(words[0]);
    for(int i=0; i<3; i++){ //******************
        if(topP[i] < p){
            topNode[i] = idx;
            topP[i] = p;
            break;
        }
    }
}
for(int i=0; i<3; i++){ //********************
    Text v = new Text(Integer.toString(topNode[i]) + " " + Float.toString(topP[i]));
    context.write(null, v);
}
```