未来城市物流系统——代码思路

数据结构

```
CombileItem：组合货物类
    ArrayList<Item> items;//组合货物集合
    Node srcNode;//货物起始站点
    Node dstNode; //货物终止站点
    double itemTotalWeight; //组合货物总重量
    ArrayList<Node> incNode;//货物必经站点
    ArrayList<Link> linkRoute;//路径
    ArrayList<Integer> cars;//路径对应的列车号
Node：站点类
    int nodeId;//站点id
    ArrayList<Link> relatedLink;//与站点相连的轨道
    int totalWorkerNum;//站点拣货员数量
    int availWorkerNum; //站点可用拣货员数量
Link：轨道类
    int linkId;//轨道id
    Node srcNode;//轨道连接的站点1
    Node dstNode; //轨道连接的站点2
    ArrayList<Car> linkCars;// 轨道上的车
    double linkCost;// 轨道权重 可设成1或其他
    double totalWeight;// 轨道总载重
    double availWeight;// 轨道可用载重
Car：列车类
    int linkId;//列车所在的轨道id
    int carNo;//列车号
    double maxWeight;//列车最大载重
    double availWeight; //列车可用载重
Item：货物类
    int itemId; //货物id
    Node srcNode; //货物起始站点
    Node dstNode; //货物终止站点
    double itemWeight;//货物重量
    ArrayList<Node> incNode; //货物必经站点
```

算法思路

1. 读取输入的数据，封装成站点、轨道、货物类的对象，其中货物分为有必经站点的货物和没有必经站点的货物。
2. 将货物按照“起始站点-必经站点-终止站点”相同的原则组合成组合货物，存入组合货物集合。

   2.1 将货物存入map，map的key是按照“起始站点-必经站点-终止站点”的顺序将站点id拼合成的字符串，value为对应的货物组成的集合；

   2.2 遍历map，得到同源同宿货物的集合。组合集合中的货物，封装成组合货物，存入组合货物集合。

为了使组合之后的组合货物尽可能的少，将同源同宿的货物按照重量降序排列，先取得集合中最重的货物，使其与最轻的若干货物组合，使组合货物的总重量等于或尽可能接近列车容量，并将已组合的货物从集合中移除，直到所有货物都被组合。

3. 将组合货物集合按照组合货物的重量降序排列，并遍历集合分别求出最短路径并保存。

   3.1 判断组合货物的起始站点和终止站点有无拣货员，若无，直接返回失败；

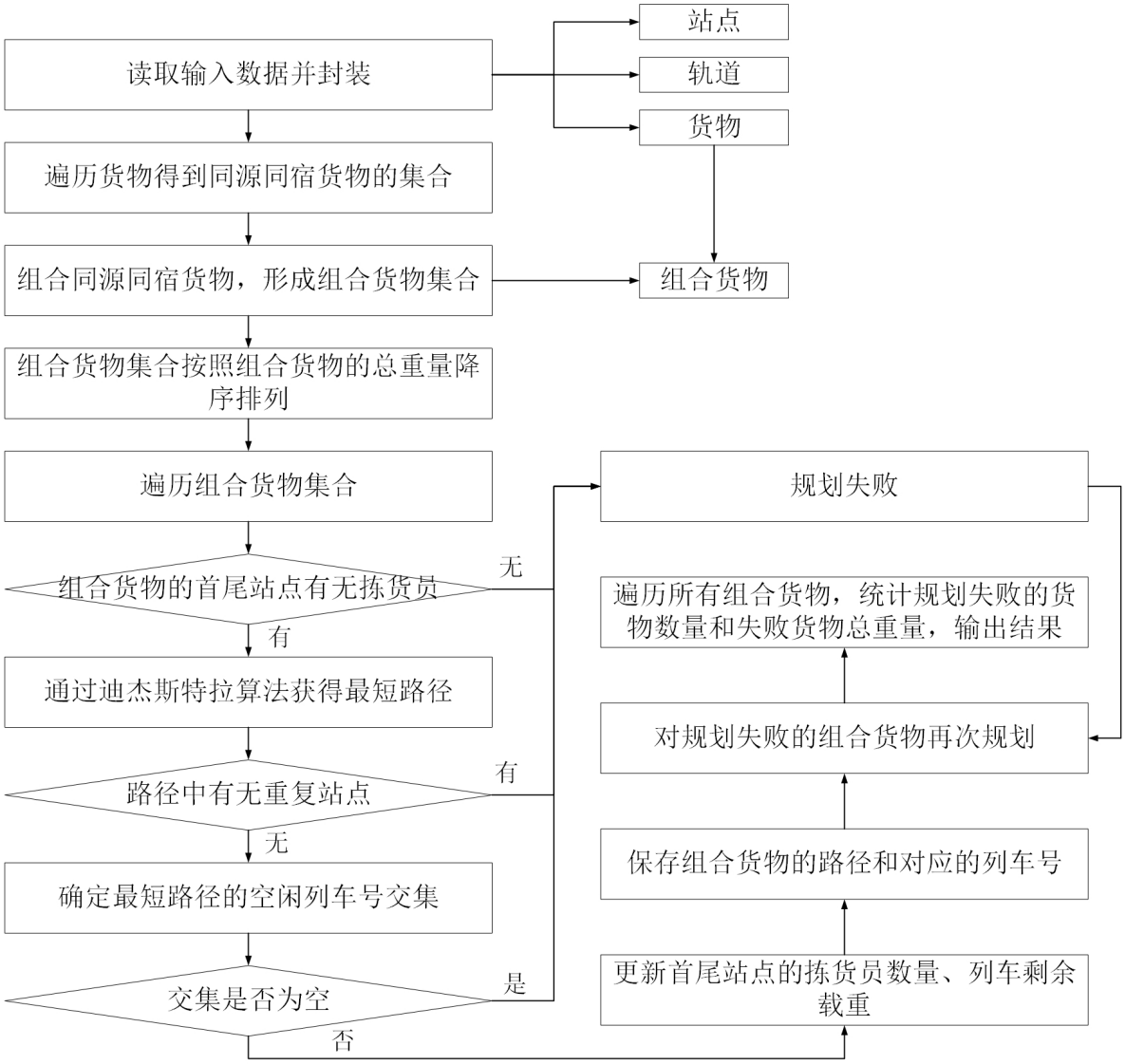
   3.2 利用迪杰斯特拉算法获得最短路径。对于含有必经站点的组合货物，将其看成多个路径的组合，利用迪杰斯特拉算法获得各个最短路径并组合，若路径中含有重复站点，直接返回失败；

   3.3 根据求得的最短路径中的站点确定经过的轨道，并对路径上的各个轨道的空闲列车号求交集，若交集为空，直接返回失败；

   3.4 取交集中的第一个列车号作为组合货物路径所用的列车号，更新首尾站点的拣货员数量，更新列车的剩余载重，保存组合货物对应的路径轨道和轨道对应的列车号。

4. 对规划失败的组合货物再次规划，将原来规划失败的路径的权重设为最大。

5. 遍历所有的组合货物，统计规划失败的货物数量和失败货物总重量，并按要求输出结果。



创新点

- 使用CombileItem组合货物类保存组合货物信息及其路径信息

  使用CombileItem的items属性保存被组合的货物，itemTotalWeight属性保存组合货物总重量，linkRoute属性保存路径轨道，cars属性保存轨道对应的列车号。

```
CombileItem：组合货物类
    ArrayList<Item> items;//组合货物集合
    Node srcNode;//货物起始站点
    Node dstNode; //货物终止站点
    double itemTotalWeight; //组合货物总重量
    ArrayList<Node> incNode;//货物必经站点
    ArrayList<Link> linkRoute;//路径
    ArrayList<Integer> cars;//路径对应的列车号
```

- 合并同源同宿货物时尽可能考虑最优组合方案

  将同源同宿的货物组合可以大量减少需要规划的货物数量。为了使组合之后的组合货物尽可能的少，将同源同宿的货物按照重量降序排列后，组合最重和最轻的货物，先取得集合中最重的货物，使其与最轻的若干货物组合，使组合货物的总重量等于或尽可能接近列车容量，并将已组合的货物从同源同宿货物集合中移除，直到所有货物都被组合。

```
            //将同源同宿货物按照重量不同存入map key:重量 value：重量相同的货物集合
             HashMap<Double, ArrayList<Item>> map = new HashMap<>();
             ArrayList<Double> weights = new ArrayList<>();//同宿货物的重量组成
的集合
             for (Item item : list.get(i)) {
                 weights.add(item.itemWeight);
                 if (map.get(item.itemWeight) == null) {
                     ArrayList<Item> items = new ArrayList<>();
                     items.add(item);
                     map.put(item.itemWeight, items);
                 } else {
                     ArrayList<Item> items = map.get(item.itemWeight);
                     items.add(item);
                 }
             }
             //整合重量 至等于或尽可能接近列车载重
             while (weights.size() > 0) {
                 ArrayList<Double> weightCombile = new ArrayList<>();//保存被
组合的货物重量
                 double lastWeight = carCapacity;
                 //先选取最重的货物
                 lastWeight -= weights.get(0);
                 weightCombile.add(weights.get(0));
                 weights.remove(0);
                 //然后选取若干个最轻的货物
                 if (weights.size() > 0 && lastWeight > 0) {
                     int endIndex = weights.size() - 1;
                     int startIndex = endIndex;
                     do {
                         lastWeight -= weights.get(startIndex);
                         weightCombile.add(weights.get(startIndex));
                         startIndex--;
                     } while (lastWeight >= 0 && startIndex >= 0);
                     //组合后列车剩余载重<0?
                     if (lastWeight < 0) {
                         //移除weights中被整合的重量
                         startIndex++;
                         weightCombile.remove(weightCombile.size() - 1);
                         for (int j = endIndex; j >= startIndex + 1; j--) {
                             weights.remove(j);
                         }
```

```
                    } else {
                        weights.clear();
                    }
                }
                //根据weightCombile集合从map中取出货物，创建CombileItem并存入
                ....
            }
```

路径规划失败原因

- 起始站点和终止站点无拣货员。
- 求出的最短路径中的各个轨道可用列车号无交集。
- 带有必经站点的组合货物规划出的路径包含重复站点。

改进思路

在求组合货物的路径时，组合货物独占一个列车，可以在此基础上分析更换列车和货物汇聚分流的情况，能更好的利用资源，规划更多货物。