δV-2EDSP算法java实现版

包 *myGraph*

类 *myGraph*

- JGraphT所提供的图最多只能在边上联系一种属性，本算法中图的边有多个属性，将JGraphT提供的图包装到*myGraph*类中来保存属性信息，同时也可以储存算法的相关参数以及结果信息，方便算法的编写.

成员

- *graph*：默认使用JGraphT中提供的DefaultDirectedWeightedGraph也就是带有weight的有向非重复图
- *costMap*:使用Map数据结构储存每一条边对应的cost
- *nodeNum*:图的顶点数
- *edgeNum*:图的边数
- *startPoint*:算法运行时给定的起点
- *sinkPoint*:算法运行给定的终点
- *maxComVertex*:算法给定的最大相交点数
- *mutiGraph*:标记图是否允许平行边
- *directed*:标记图是否有向
- *shortestPath*:原图中起点到终点的最短路径
- *restrictedShortestPaht*:运行RSP算法后得到的最短路径
- *pathPair*:算法最后的得到的最短路径对

方法

```
DefaultWeightedEdge addNewEdge(int source ,int target,double weight,int cost)
```

- 这个方法将在图中添加一条边，然后返回这条边
- *source*:边的起点
- *target*:边的终点
- *weight*:这条边的weight属性
- *cost*:这条边的cost属性

类 *ILPGraph*

- 线性规划算法需要输入允许平行边存在的图，这里除了*graph*成员与*myGraph*不同以外，其他类似.

包 *graphIO*

类 *GraphRandomGenerator*

方法

```
myGraph generateRandomGraph(int nodeNum, int edgeNum)
```

- 算法需要随机图进行测试,本方法返回生成的随机图，且边的属性也随机生成
- *nodeNum*:随机生成图的顶点数
- *edgeNum*:随机生成图边数

类 *GraphWriter*

成员

- *graphFolder*:用于储存图的文件的目录 方法

```
boolean saveGraphToJson(myGraph myGraph,String graphFileName)
```

- 这个方法将会把输入的图转化成json格式，储存到指定的文件名的文件中,返回表示操作是否成功
- *myGraph*:将要被储存的图
- *graphFileName*:指定的文件名

类 *CSVRecorder*

成员

- *csvPath*:用于储存记录文件的目录

方法

```
void saveToCSV(String csvFileName,String data[][])
```

- 这个方法将会将指定的数据转化为csv格式储存到指定的文件中
- *csvFileName*:储存的文件名
- *data*:将要储存的数据，二维数组，每一行表示一条记录，每一列表示一个记录属性

类 *CSVCol*

- 用于记录数据在转化为csv格式时的规则

成员

- *graphId*:图的编号在*data*数组中第二维中的位置
- *newAlgRunTime*:最新算法的运行时间,后面相同者省略
- *newAlgResult*:最新算法的运行结果,后面相同者省略
- *colNum*:csv中列的总数
- *csvHeader*:每一列的数据名称

包 *alg.NewAlg*

类 *NewAlg*

- 新算法的主类

成员

- *INFINITY*:用于表示无限大

方法

```
myGraph getResidualGraph(myGraph myGraph,int scale)
```

- 本方法将原图处理成余图并返回余图，但是不修改原图，处理的参数均存放于*myGraph*中
- *myGraph*:原图
- *scale*:用于成比例放缩边的cost属性，这里全部使用*scale*为1

```
double RSPNoRecrusive(myGraph myGraph)
```

- 本方法使用RSP算法寻找图中从起点到终点的受限制的最短路径，并返回这条路径的cost总和
- RSP算法所需要的三个额外参数起点、终点、最大共同点数都存储在*myGraph*中

类 *JavaLPAlg*

方法

```
String readJsonGraph(String fileName)
```

- 本方法读取指定的json文件，并将其中的文本返回
- *fileName*:读取的文件路径

```
myGraph parseJsonToGraph(String jsonStr)
```

- 本方法接受json文本，将其转化为图的数据结构，并返回图
- *jsonStr*:json图的文本

```
ILPGraph getGraphForILP(myGraph myGraph)
```

- 本方法接受原始的图，处理后返回线性规划算法所使用的图
- *myGraph*:原始的图以及其他参数

```
double solveWithGLPK(ILPGraph myGraph,int probId,LPSolver lpSolver)
```

- 本方法接受算法参数，并使用线性规划算法求解δV-2EDSP问题，并返回结果
- *myGraph*:算法所需要的参数以及处理后的图
- *probId*:问题ID
- *lpSolver*:指定使用哪一种工具求解，目前可选`GLPK`或者`CPLEX`

类 *MWLD*

- `MWLD`算法的主类

方法

```
double mwldALg(myGraph myGraph)
```

- `MWLD`算法的入口，返回算法结果

```
myGraph mwldGetAuxGraph(DefaultDirectedWeightedGraph graph,int sinkPoint)
```

- 将原图转化为算法需要的辅助图
- *graph*:原图
- *sinkPoint*:算法参数中的终点

```
mwldPathXor(List<Integer> pathP,List<Integer>pathQ)
```

- 本方法将两条路径进行异或操作，本质上是将两条路径的边结合，去除两者的反向边后重新获得两条路径
- *pathP*、*pathQ*:处理的两条路径
- 返回结果的两条边的数组

```
List<Integer>[] mwldGetAuxGraphEdge(DefaultDirectedGraph graph,int pointS,int
pointT)
```

- 获得辅助图中的边

```
DirectedWeightedMultigraph<Integer,DefaultWeightedEdge>
getSPReverseGraph(DefaultDirectedGraph graph,List<Integer> shortestPath)
```

- 将图中的最短路径上的边反向并且去除原来的边，返回得到图

包 *alg.Util*

类 *Util*

方法

```
double getSPWeight(DefaultDirectedWeightedGraph<Integer,DefaultWeightedEdge>
graph, List<Integer> path)
```

- 统计一条路径在一个图中所有边的weight的总和并返回