KSP问题算法JAVA版

包 *graphStructure*

类 *MyGraph*

- 由于`JGrpahT`中提供的图最多只能允许边有一种相关属性，所以本类将`JGraphT`提供的图进行包装，储存更多相关属性并且提供更多操作

成员

- *graph*:图
- *costMap*:每一条边对应的*cost*属性
- *delayMap*:每一条边对应的*delay*属性
- *CurentWeight*:记录目前图中的*weight*属性是取得哪一种属性

方法

```
CurentWeight getCurentWeight()
```

- 得到目前图中所采用的*weight*的信息，可以为*cost*或者*delay*

```
DefaultWeightedEdge addNewEdge(int src,int tar,int cost,int delay)
```

- 向图中添加新的一条边并且返回这条边
- *src*:边的起点
- *tar*:边的终点
- *cost*:边的*cost*
- *delay*:边的*delay*

```
void removeAllEdges(int src,int tar)
```

- 移除指定的所有边
- *src*:移除边的起点
- *tar*:移除边的终点

```
void setCurentWeight(CurentWeight curentWeight)
```

- 设置目前的图所用的*weight*的属性

```

```

```
MyGraph copyGraph()
```

- 将本图完全复制一份后返回

包 *graphIO*

类 *GraphWriter*

成员

- *graphDataFolder*:储存图文件的目录

方法

```
void checkFolder()
```

- 检查储存图的目录是否存在，不存在则创建

```
JSONObject MyGraphToJsonObj(MyGraph  myGraph)
```

- 将图转换成*JsonObject*并返回

```
void saveGraphToJson(MyGraph myGraph,String fileName)
```

- 将图转换为Json格式以后储存在指定的文件中
- *myGraph*:储存的图
- *fileName*:文件名

类 *GraphReader*

方法

```
String readJsonStr(String fileName)
```

- 读取Json文件并且返回文本

```
MyGraph readGraph(String fileName)
```

- 读取指定图文件并返回图

类 *GraphRandGen*

方法

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

- 根据指定的参数生成随机图并返回
- *nodeNum*:指定图的顶点数
- *edgeNum*:指定图的边数

类 *CSVRecorder*

成员

- *csvPath*:csv文件的存储目录

方法

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

- 将指定的数据存储到指定文件名的csv文件中
- *csvFileName*:文件名
- *data*:存储的数据

类 *csvCol*

- 用于记录csv文件中数据储存的具体格式

包 *algorithm*

类 *KRSPAlgBaseOnDelay*

- KRSP新算法的主类，但是是基于*delay*的

成员

- *INF*:用于表示无限

方法

```
List<List<Integer>> pathsXor(List<List<Integer>> paths, List<Integer> pathP)
```

- 本方法将一个路径的集合与另一条路径进行异或
- *paths*:路径集
- *pathP*:另一条路径

```
MyGraph getCostReverseGraph(MyGraph oriGraph, List<List<Integer>>paths)
```

- 输入一个图和一个路径的集合，将图中含有的路径的边全部反向，边的*cost*值也取反
- 后续的`getDelayReverseGraph`、`getAllReverseGraph`的功能类似，只是将*delay*取反，或者*cost*和*delay*全部取反

```
List<List<Integer>> getKSPWithCost(MyGraph myGraph,int startPoint,int desPoint,int spNum)
```

- 在*cost*属性设置为*weight*的情况下获得k条最短路径
- *myGraph*:图
- *startPoint*:起点
- *desPoint*:终点
- *spNum*:k的取值
- 后面的`getKSPWithDelay`功能类似，只是将*weight*设置为*delay*

```
int countAttr(MyGraph graph,List<List<Integer>>paths,Attr attr)
```

- 统计路径集合在一个图中某一种属性之和，这个属性可以是*delay*或者*cost*
- *graph*:图
- *paths*:路径集合
- *attr*:指示统计哪一种属性

```
int getSplitNode(int oriNode,int upperNum,int nodeNum)
```

- 算法需要用到的一个辅助图中点需要被拆分成有上标的点。考虑到方便处理，这些拆开的点仍然使用整数表示，但是与上标形式需要换算
- 本方法获得上标点的整数形式
- *oriNode*:点的下标
- *upperNum*:点的上标
- *nodeNum*:原图点的总数
- 后面的`getOriNode`作用相反

```
MyGraph getCycleAuxGraph(MyGraph myGraph,int delayBound,int desPoint)
```

- 获得用于求二分环的辅助图

```
List<Integer> getOriPath(List<Integer>path,int nodeNum)
```

- 将路径中的每一个被拆开的点还原成它的下标

```
List<Integer>findNegativeCycle(MyGraph graph,int startPoint)
```

- 寻找一个负环并且返回

```
List<Integer> getBestCycle(MyGraph myGraph,List<Integer>ori_cycle)
```

- 在找环时，找到的环可能不是简单环，这时提取出里面最好的环返回

```
List<Integer> getBicameralCycle(MyGraph reverseGraph,MyGraph
oriGraph,List<List<Integer>>ksp,int delayBound,int startPoint,int desPoint,int
spNum,int maxCost)
```

- 寻找二分环并且返回

```
List<List<Integer>>cyclePathXor(List<Integer>cycle,List<List<Integer>>paths,int
spNum)
```

- 用找到的二分环优化当前的最佳解的路径集合

```
List<List<Integer>> getKSP(MyGraph graph,int startPoint,int desPoint,int spNum,int
maxDelay)
```

- 新的kRSP算法的主入口，返回找到的k条不相交路径
- *graph*:原始图
- *startPoint*:起点
- *desPoint*:终点
- *spNum*:问题中k的取值
- *maxDelay*:问题中*delay*的上限

类 *ILPAlgorithm*

- 线性规划算法的主类

方法

```
List<List<Integer>> getPaths(MyGraph graph,List<DefaultWeightedEdge> edgeList,int
startPoint,int desPoint)
```

- 由于线性规划算法中每一条边的使用都是用变量表示了，本方法用于将得到的解中的变量取值还原为路径

```
kRSPResult write_lp_solution(glp_prob lp,MyGraph
graph,List<DefaultWeightedEdge>graphEdgeList,int startPoint,int desPoint)
```

- 将线性规划的解还原为kRSP问题的解

```
kRSPResult solveWithGLPK(MyGraph myGraph,int startPoint,int desPoint,int
pathNum,int maxDelay)
```

- 线性规划算法的主入口，返回最后找到的路径集合

包 *main*

类 *main*

- 整个问题算法的运行主文件，也是使用示例