

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 康瑞 |
| 学号 | 1160300514 |
| 班号 | 1636101 |
| 电子邮件 | 110300514@stu.hit.edu.cn |
| 手机号码 |  |

**目录**

[1 实验目标概述 1](#_Toc507927440)

[2 实验环境配置 1](#_Toc507927441)

[3 实验过程 1](#_Toc507927442)

[3.1 Poetic Walks 1](#_Toc507927443)

[3.1.1 Get the code and prepare Git repository 1](#_Toc507927444)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc507927445)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc507927446)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc507927447)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc507927448)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc507927449)

[3.1.4.1 Make the implementations generic 2](#_Toc507927450)

[3.1.4.2 Implement Graph.empty() 2](#_Toc507927451)

[3.1.5 Problem 4: Poetic walks 2](#_Toc507927452)

[3.1.5.1 Test GraphPoet 2](#_Toc507927453)

[3.1.5.2 Implement GraphPoet 2](#_Toc507927454)

[3.1.5.3 Graph poetry slam 2](#_Toc507927455)

[3.1.6 Before you’re done 2](#_Toc507927456)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc507927457)

[3.2.1 FriendshipGraph类 2](#_Toc507927458)

[3.2.2 Person类 3](#_Toc507927459)

[3.2.3 客户端main() 3](#_Toc507927460)

[3.2.4 测试用例 3](#_Toc507927461)

[3.2.5 提交至Git仓库 3](#_Toc507927462)

[3.3 The Transit Route Planner（选做，额外给分） 3](#_Toc507927463)

[4 实验进度记录 3](#_Toc507927464)

[5 实验过程中遇到的困难与解决途径 3](#_Toc507927465)

[6 实验过程中收获的经验、教训、感想 4](#_Toc507927466)

# 实验目标概述

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现ADT

# 实验环境配置

App：Idea IntelliJ IDEA

# 实验过程

## Poetic Walks

构建图，节点为String，在概率允许的两点依靠原有数据加边，泛型设计

### Get the code and prepare Git repository

https://github.com/rainywang/Spring2018\_HITCS\_SC\_Lab2/tree/master/P1

### Problem 1: Test Graph <String>

Write Test for Instance test, which will be used for further testing after completed the insurance of Graph<L>.

### Problem 2: Implement Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

#### Implement ConcreteEdgesGraph

**Using the REP:**

private final Set<String> vertices = new HashSet<>();

private final List<Edge> edges = new ArrayList<>();

which means creating an Linked Edge Graph(neither matrix or list…, what is it..)and make functions for creating, adding, removing method to meet basic need of the graph using.

Write AF,RI,checkRep

#### Implement ConcreteVerticesGraph

Just like implemented before, in this case, using the Rep:

private final List<Vertex> vertices = new ArrayList<>();

which means using Linked List of the Graph implementation (Oh, that is good?.? A bit better than the former implementation,

also, develop the same function of the basic needs for Graph using, and checkRep(), specification.

### Problem 3: Implement generic Graph<L>

#### Make the implementations generic

Change the former implementation from concrete class to a abstract class-> Graph with parameter <L> which can get more widely used of the Graph.

#### Implement Graph.empty()

With a static method imported to Java from JDk1.7, static method is not well fit for the abstract data type, Graph.empty(), from the Doc saying: implement one possible Graph type…(So, used ConcreteEdgeGraph)

### Problem 4: Poetic walks

#### Test GraphPoet

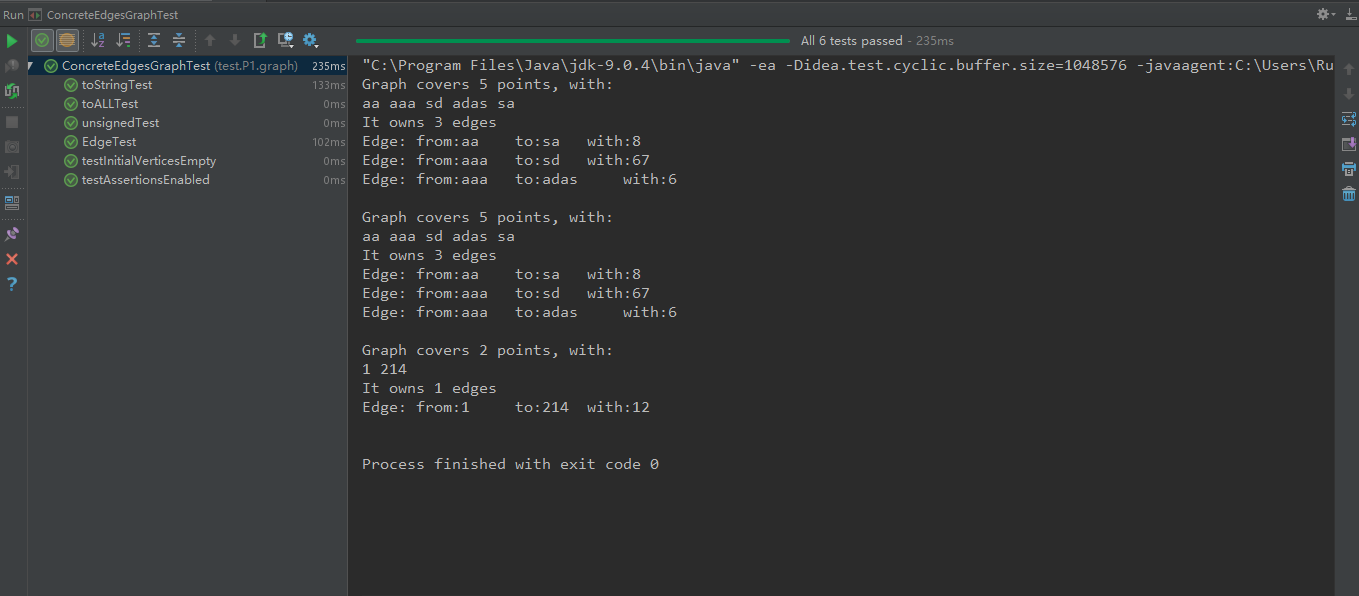
Just make test for the GraphPoet, and it relates the IO:File, Scanner to read from a file, and using the method of Graph<String>, Junit participated.

#### Implement GraphPoet

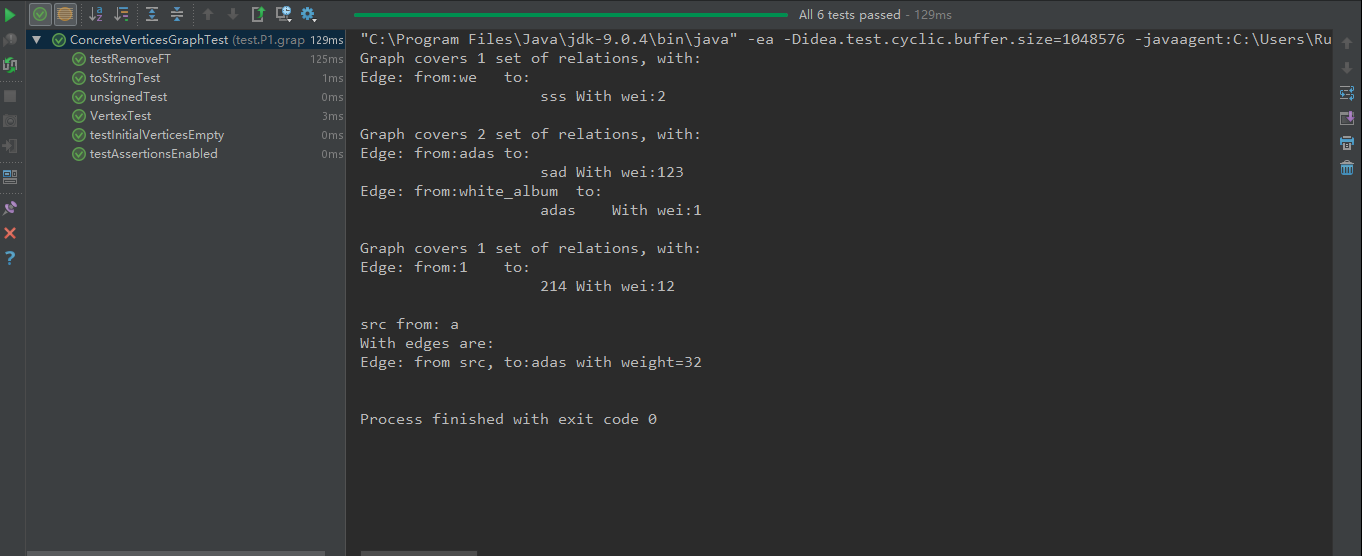
#### Graph poetry slam

Actually, the graph implementation can use Graph<String> to complete the need for developing a graph for the words which have occurred in the document provided, and it is able to develop

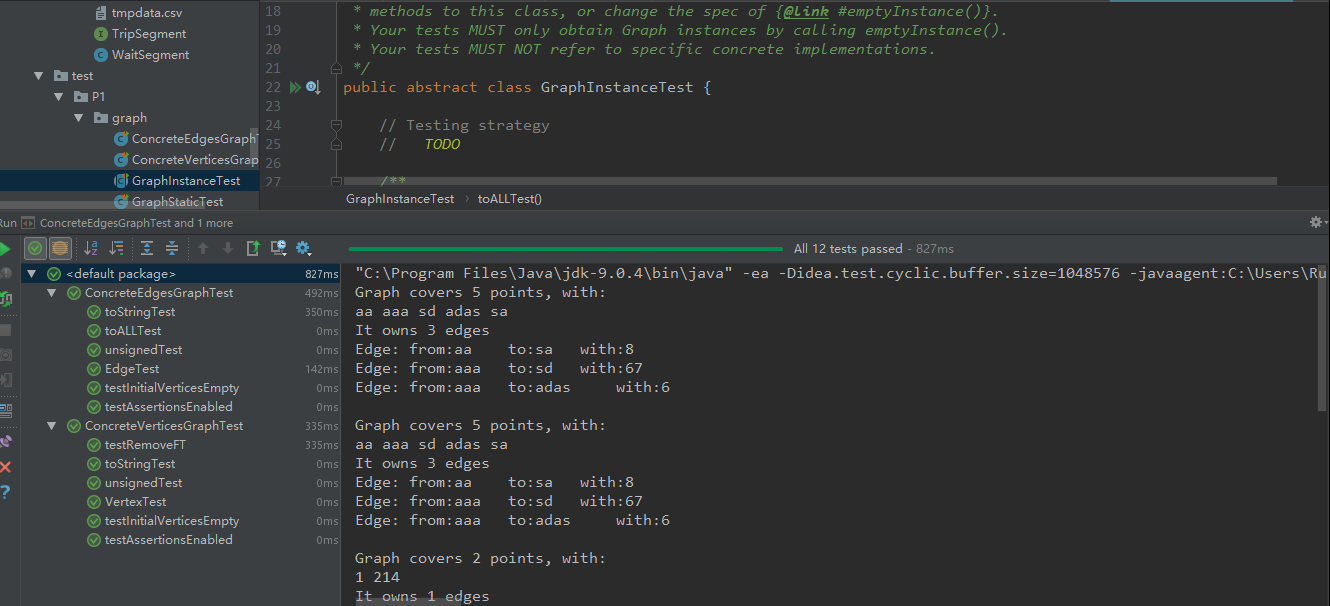
Final Tests:

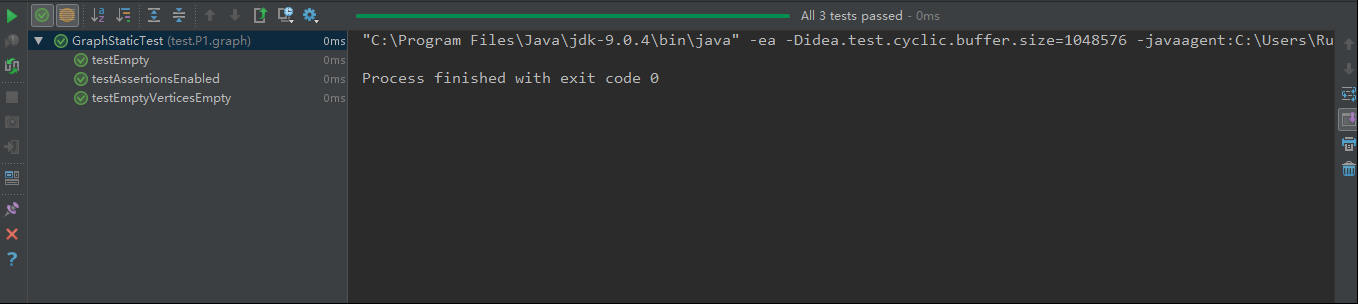


EdgeGraph Test

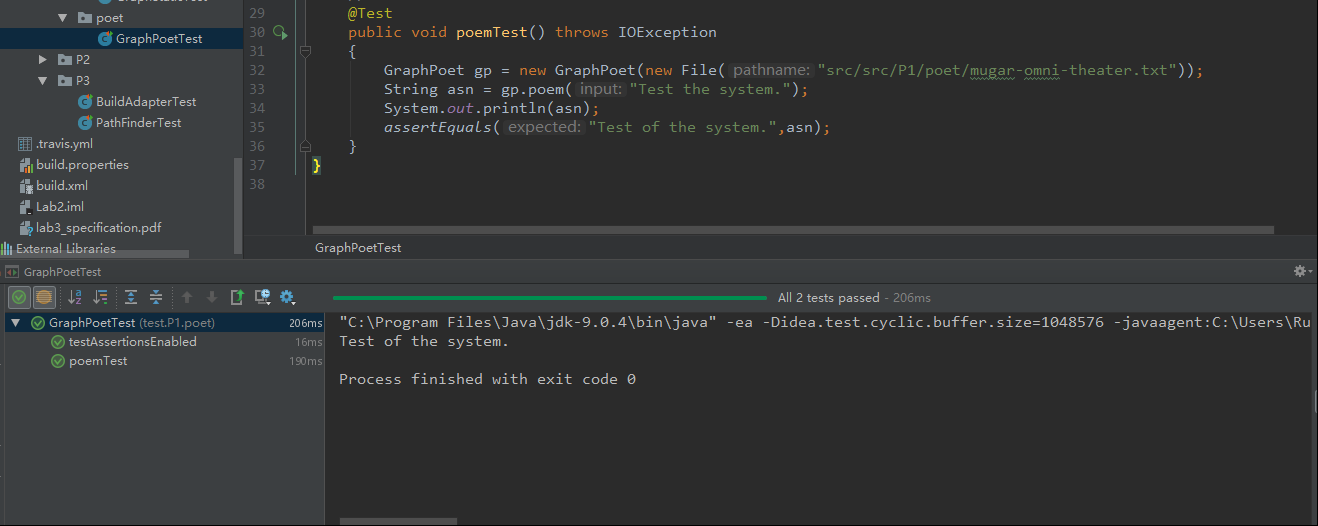


VertexGraph Test



(Instance Test)  


Empty Instance Test



Poet Test

### Before you’re done

请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "before_youre_done)的说明，检查你的程序。

如何通过Git提交当前版本到GitHub上你的Lab2仓库。

在这里给出你的项目的目录结构树状示意图。

src

src

P1

graph

….java

poet

….java

….txt

test

P1

graph

…Test.java

poet

…Test.java

….txt

## Re-implement the Social Network in Lab1

在本次实验中 Poetic Walks中定义的Graph<L>及其两种实现，重新实现Lab1中3.3节的FriendshipGraph类

### FriendshipGraph类

There only contains the using skill of the class: Graph, and I added a method called: getDistanceBetween(Person p1,Person p2), and it computes the distance between the Vertices of the Graph basis on the relationship defined in the Graph I had just feeded.

### Person类

“In Person”: there contains only the attribute of one point: String Name, & then develop method to return and set this value;

### 客户端main()

“In main()”: accept from the console, it can depart to several parts: names, relationships, get distance between relationship, with a Integer output. In FriendshipGraphTest

### 测试用例

“In Test”: calculateTest(), test examples: testBench():

Person rachel = new Person("Rachel");

Person ross = new Person("Ross");

Person ben = new Person("Ben");

Person kramer = new Person("Kramer");

gra.addVertex(rachel);

gra.addVertex(kramer);

gra.addVertex(ben);

gra.addVertex(ross);

gra.addEdge(rachel, ross);

gra.addEdge(ross, rachel);

gra.addEdge(ross, ben);

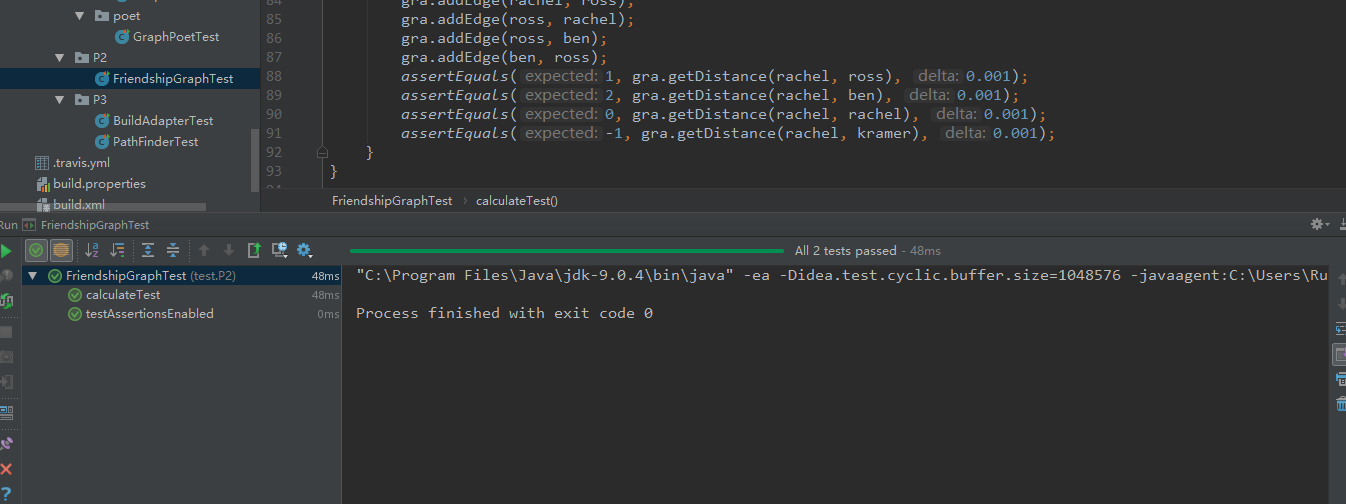
gra.addEdge(ben, ross);

assertEquals(1, gra.getDistance(rachel, ross), 0.001);

assertEquals(2, gra.getDistance(rachel, ben), 0.001);

assertEquals(0, gra.getDistance(rachel, rachel), 0.001);

assertEquals(-1, gra.getDistance(rachel, kramer), 0.001); Done.



### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab3仓库。

在这里给出你的项目的目录结构树状示意图。

“In Submitation”:

Src

src

P2

FriendshipGraph.java

Person.java ...

test

P2

FriendshipGraphTest.java

## The Transit Route Planner（选做，额外给分）

Documents: </p>

Caution： All my specification will depart into two parts: first part which is the final implementation with no bugs, and second part is an implementation with bugs due to the wrong choice while building graph. Therefore, the specification is not in the order of my implementation.

First part: NO BUGS

Files are listed below:

EnhancedPathPlanner.java

EnhancedBuildAdapter.java

PointVe.java

And other files are shared between the first and second part.

I will show the interface and details about those 3 files, and when you want to get details about those shared files, you can go to the second part.

1. EnhancedPathPlanner.java

Rep fields:

private ConcreteVerticesGraph<PointVe> graph;

private Map<String, Map<Integer, List<PointVe>>> busline;

private Map<String, Position> stopname;

private Map<String, List<Pair<PointVe,PointVe>>> station;

private int maxWaitTime;

Methods:

public List<Itinerary> computeRoute(Stop src, Stop tar,int Time)

private Itinerary dateback(Pair<Map<PointVe, Integer>, Map<PointVe, PointVe>> best, PointVe target)

public Pair<Map<PointVe, Integer>, Map<PointVe, PointVe>> Dijkstra(PointVe src)

the rep fields:

graph: hold the edges, vertices in it, and in the method of computeRoute(), I added the start point into the graph.

The implementation of the graph: I separated each bus into points, each represents one station, and in one station, the point representing one bus stop is separated into two points, one is in, the other is out, in the station, “in” points can add edge to the “out” points, however we cannot reverse it, and from “out” points can add edge to the next station’s “in” point of the same bus, inside the station: if two buses stops within the “maxWaitTime”, we can add edge from “in” point to “out” point with weight equals to the total time to wait, when it comes to the same bus(same start time which means the rider won’t get off the bus), I add edge from the “in” point to the “out” point, with the weight equals to 1, to avoid the “set” method in @Class Concrete Vertex Graph.

Method: ComputeRoute():

Get through and check whether there is a possible solution, from Start place and time to the target place. Using the method: DIJKSTRA. Return value is possible solution.

Dijstra:

running Algorithm: Dijkstra, based on graph provided as global value. From src, returns Map<String, Integer> defines the distance from src to key point, src is a generated point with Station, BusName, time

DateBack:

To generate the path from the result of the Dijkstra, we need the dateback method, it computes from the target point and get the src point of it, and repeat the same process until we meet the src or null point. Using Stack to reverse the arrange. If we meet the null point, and the former point is not one of the src point, this may be caused by some fault or the graph cannot generate from src to tar. Finally, returns the ans with Itinerary structure.

AF(RoutePlanner) = any possible solutions provided bt Dijkstra,

RI(RoutePlanner): judge whether a solution is possible.

Rep Exposure: using private, new ,final fields and return functions.

1. EnhancedBuildAdapter.java: implements from RoutePlannerBuilder which return a RoutePlanner variable while using the build method.

Uses the build graph method as I have illustrated.

AF(Adapter) = Possible file which can get the route plan.

RI(Adapter): Undefined, Just a kind of builder.

Rep Exposure: private , new , final.

1. PointVe.java: a class hold useful information, better than those class asked to implement.

Rep field:

private final Position Station\_Loc;

private final String BusName;

private final Integer Layin\_time;

private final Boolean INOUT;

methods are easy to think, just to get the unexposed varibles.

The INOUT variable which is Boolean class, is designed to judge the to separated point.

AF(PointVe) = {(bus, time, stop)| the bus stops @Stop @Time canmake sense}

RI(PointVe): true if this point really exists.

Rep Exposure: private, final, new.

Test Results: testfiles are in src:P4: \*.csv

time = 6100, from as we can, to how can it be

\*

\* results:

\* With Path Start: @Point Staying point: bus: Start Point Stay @Station: as we can switch:0.9902 90.0000

@Time: 6100

End @PointStaying point: bus: 78D Stay @Station: as we can switch:0.9902 90.0000

@Time: 7100

Costs Time=@Seconds 1000

With Path Start: @Point Staying point: bus: 78D Stay @Station: as we can switch:0.9902 90.0000

@Time: 7100

End @PointStaying point: bus: 78D Stay @Station: how are you switch:1.0002 91.0902

@Time: 8900

Costs Time=@Seconds 1800

With Path Start: @Point Staying point: bus: 78D Stay @Station: how are you switch:1.0002 91.0902

@Time: 8900

End @PointStaying point: bus: 67R Stay @Station: how are you switch:1.0002 91.0902

@Time: 10000

Costs Time=@Seconds 1100

With Path Start: @Point Staying point: bus: 67R Stay @Station: how are you switch:1.0002 91.0902

@Time: 10000

End @PointStaying point: bus: 67R Stay @Station: how can it be switch:0.2332 100.3333

@Time: 15900

Costs Time=@Seconds 5900

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Second Part: build graph:bugs

P1: Correct Interface: RoutePlanner, RoutePlannerBuilder, Stop, TripSegment:

Interface: Define basic method to be implemented.

P2: Concrete Methods (From base to top)

1. Stoper: implements Stop

Rep:

private final String StopName;

private final String Latitude;

private final String Longitude;

private final List< Pair<String, Integer> > BusTimeTable = new ArrayList<>();//bus -> time: in sec

private Boolean Updated;

private final Integer maxWaitTime;

These are rep in Stoper, in short, it gives a detailed info about one stop: its name, latitude, longitude, and the bustimetable of the bus coming to the stop and stop, Updated is a private value represents the arrange of whether the BusTimeTable is sorted, which uses Collections.sort() method.

1. Class: Liner

Which describe a line a specified bus runs, it controls with a starttime which makes one bus is specified to one “Liner”.

Rep:

private final String BusName;

private final Map<Integer, Set<Pair<Position, Integer>>> BusTimePlan;

Methods: addBusStopTime, toString(@Override)

1. Class: Itinerary

Rep:

Queue<String> que = new ArrayDeque<>();

Queue<Pair<Pair<String,String>, Integer>> TimeQue = new ArrayDeque<>();

private Set<Stoper> StopSet;

Integer wait;

Integer takebus;

Integer start;

Integer end;

String startPl;

String endPl;

This class uses “public void addPartTime(String From,String To,Integer time)” to add part time of the trip, and finally to the TimeQue: <Pair<Pair<String,String>, Integer>>which means:<Pair<Pair<Start ,End>, TimeCost>>. And toString: generate the Queue to a string, using the StringWriter.

1. Class: BusSegment, WaitSegment, TripSegment: implemented but not in use: they have not good attribute, and used other implementation instead.
2. Class: BuildAdapter implements RoutePlannerBuilder

Rep:

private Set<Stoper> stopList;

private Map<String, Liner> Buslines;

private final ConcreteVerticesGraph<String> trafficDesigner;

BuildAdapter reads from the filename provided, the interface is:

Build(String FileName, Integer maxWaitTime)

And it finally returns a implementation of RoutePlanner class, and can use this method to initialize the RoutePlanner.

Af(StopList, Buslines, trafficDesigner) = {graph:edge,vertices| from file: edge(v1, v2)}

Rep Exposure:

Private, new, final

1. Class: PathFinder implements RoutePlanner

Rep:

private ConcreteVerticesGraph<String> Graph;

private Map<String, String> pre;

private Set<Stoper> stop;

private Map<String, Liner> Buslines;

private Map<String, String> StationSep;//Station(NBus Seped)->Station

private Map<String, Stoper> StationParam;//Station(Speciality)->Stoper

private Map<String, Integer> SepedBusTime;//BusStaionseped->BusStartTime

private Integer maxWaitTime;

the first four value have been illustrated before, here explain the next four value:

StationSep: the Separated Bus Station mapped to the original bus Station. (Example: “start dash\_61C\_9200” it specified the station: “start dash” and the bus stopped: “61C”, and time: “9200”seconds. And this station mapped to the station: “start dash”(isay… hihihi)

StationParam: the Separated bus Station mapped to the Station info: Stoper, which contains the value of latitude, longitude and so on.

Pre: the private value used in the method Dijkstra to store the pre-visit point.

SepedBusTime: the Separated bus Station mapped to bus start time.

Method:

1. private void gener\_station\_map()

this method can provide the Rep value: StationSep, StationParam, SepedBusTime and is used while constructing the class.

1. public Itinerary computeRoute(Stop src, Stop tar,int Time)

It first compute the possible Station it can start with, due to I have separated the station into different points, connected with an edge between two points, which duration is less than the “max Wait Time”. And then, I run Dijkstra algorithm from each possible start points, and finally compare each result computed by the algorithm, get the legal answer with lowest time cost.

1. private Itinerary dateBack(Pair<Map<String, String>, Map<String, Integer>> minlist, Integer mintime, String target, Integer initiaqltime)

To generate the path from the result of the Dijkstra, we need the dateback method, it computes from the target point and get the src point of it, and repeat the same process until we meet the src or null point. Using Stack to reverse the arrange. If we meet the null point, and the former point is not one of the src point, this may be caused by some fault or the graph cannot generate from src to tar. Finally, returns the ans with Itinerary structure.

1. public Map<String, Integer> Dijkstra(String src)

running Algorithm: Dijkstra, based on graph provided as global value. From src, returns Map<String, Integer> defines the distance from src to key point, src is a generated point with Station, BusName, time.

1. Test of the whole Class:

Test Bench:

List<Itinerary> ans = pf.ProvidedForTest("as we can","how can it be",7000);

Starttime= 7000;

61C,5

as we can,0.9902,90.0000,6200

as we do,0.2133,92.3233,7100

start dash,0.0212,93.4344,7900

where is it,-0.9022,94.4444,8500

how can i do,-1.2332,100.3333,10900

78D,3

as you can see,1.7222,80.0999,6000

as we can,0.9902,90.0000,7100

how are you,1.0002,91.0902,8900

67R,4

how was it,0.0001,90.0000,6700

as we do,0.2133,92.3233,8900

how are you,1.0002,91.0902,10300

how can it be,0.2332,100.3333,15900

61C,5

as we can,0.9902,90.0000,7500

as we do,0.2133,92.3233,8000

start dash,0.0212,93.4344,9200

where is it,-0.9022,94.4444,9800

how can i do,-1.2332,100.3333,12200

Result:

Travel from @Station: Your Start Point to @Station: as we can\_61C\_7500 For @Seconds: 500

Travel from @Station: as we can\_61C\_7500 to @Station: as we do\_61C\_8000 For @Seconds: 500

Travel from @Station: as we do\_61C\_8000 to @Station:

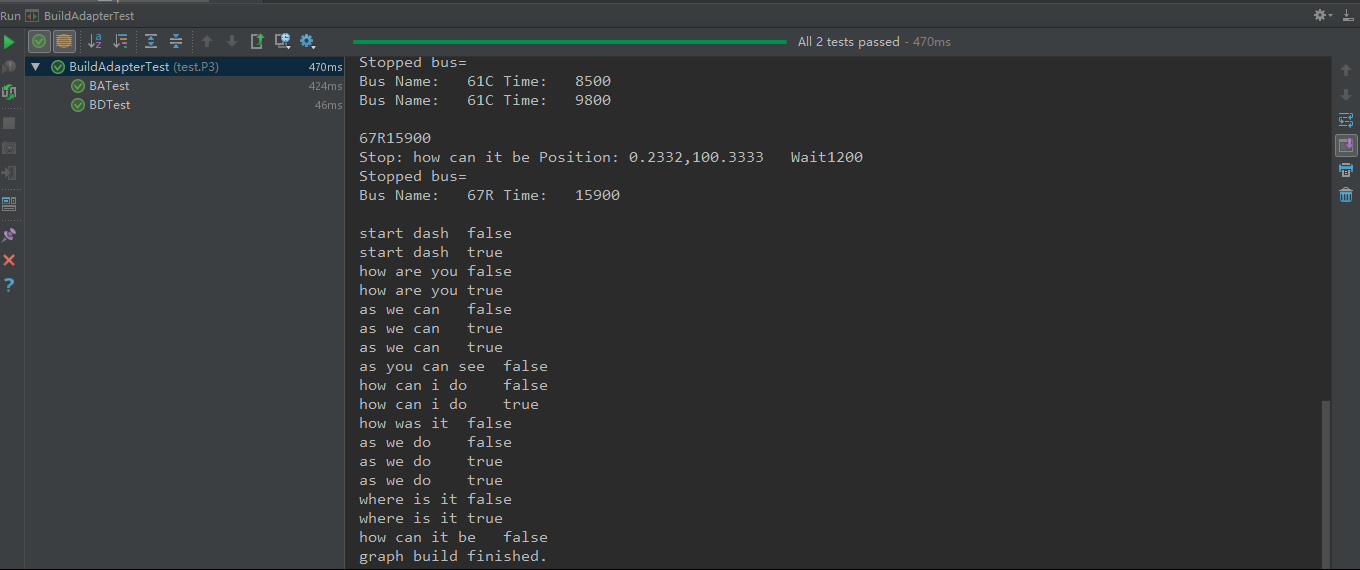
as we do\_67R\_8900 For @Seconds: 900

Travel from @Station: as we do\_67R\_8900 to @Station:

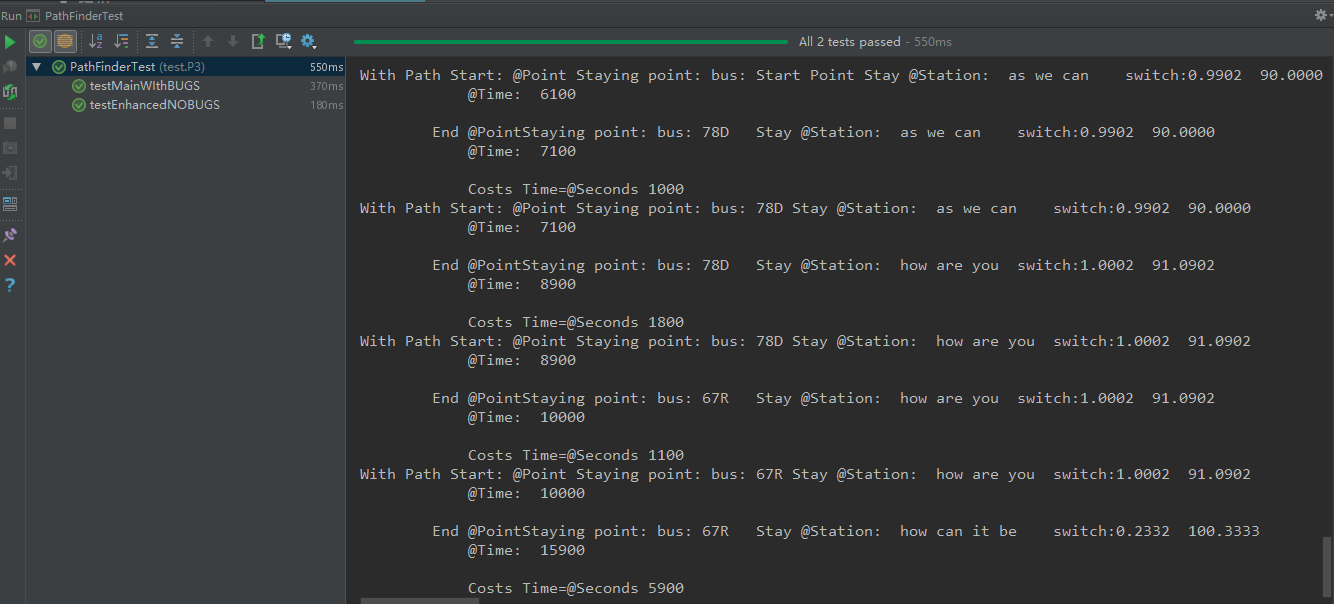
how are you\_67R\_10300 For @Seconds: 3600

Travel from @Station: how are you\_67R\_10300 to @Station: how can it be\_67R\_15900 For @Seconds: 9200

Screen Shot:



Builder Test



Route Plan Test( Solution A, no bugs)

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2018/4/3 | 0：00-24：00 | All | Finished |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

Java Core Volume Ⅰ

# 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？

ADT Programming uses interface and if the interface is well designed, it can be used in many areas, and adapting different Value type can make program more efficient. Programming towards application is always design classes to meet different needs, and can be easily read and modify, with less relay on previous code, it is more flexible, yet need more people or hours to coding for some fixed functions.

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

Just like what I have illustrated above, using generic paradigm can save time for some fixed functions, and easy to change some functions.

1. 在给出ADT的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？

Write code for some test can make programming more logical, but may cause some bugs due to the ignore of some circumstances.

This kind of design is not so good I think.

1. 本实验设计的ADT在三个应用场景下使用，这种复用带来什么好处？

Write less.

1. 为ADT撰写specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？

To avoid the re-visit from outside.

1. 关于本实验的工作量、难度、deadline。

P3:hard and the instructions may cause bugs, to be honest, the algorithm of Dijkstra is not fit for the problem, because when you stay in one station toi wait for one bus and then you can still wait for another bus in the same station, and the time may accumulate and in the end the time staying in one station may be over the max wait time. I think the only one method to get the whole graph work and produce a satisfying output is running the BFS algorithm , it can judge whether the time staying in a station is exceed.

However. P1,P2 is easy but the attach test to programming is not good.