## 第一章 系统分析

描述系统背景，系统的功能需求，分析系统可能的解决方案，介绍自己的主要工作等内容。

### 系统背景

本次课程设计题目主要是设计一个大型景区的管理系统。系统用户包括管理员和游客两类，管理员负责管理景区的景点维护；游客可以根据自己的需求对景区进行各种信息查询，及路线规划等。

### 解决方案

系统需要大量的算法实现，以及较为美观的可视化界面展示，由此我可以想出的三种解决方案如下：

1. 使用C++实现后端算法，使用Qt实现可视化界面展示。
2. 使用Java实现后端算法，使用swing实现可视化界面展示。
3. 使用Java实现后端算法，使用网页实现可视化界面展示，使用Servlet技术在后端接收前端的请求。

由于我只在半年到一年之前学习过Qt和Swing，现在已经比较生疏。所以我选用了上面列出的第三种解决方案。

### 工作成果

在这次课程设计中，我的成果如下。

#### 算法设计成果

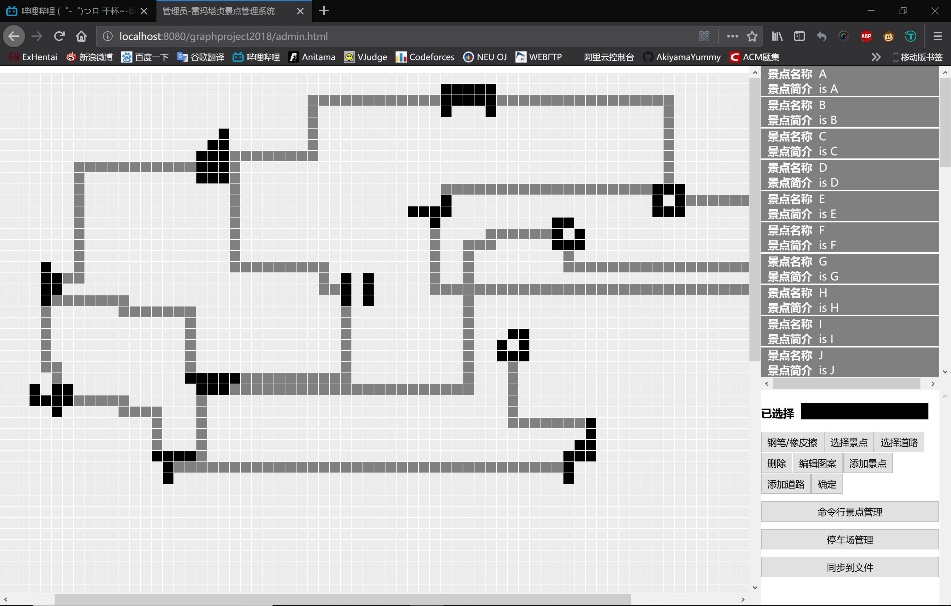
算法设计方面，我作为一名算法竞赛选手，做出了我认为比较优美高效的算法设计：

1. 图采用邻接表来表示，但一般的邻接表表示图，对于删点和删边的支持不佳，复杂度较高，所以我**自行设计了一套邻接表实现方法，通过自行设计的方法配合散列表，使得邻接表可以以低复杂度增删点和边。**加点、加边、删边的复杂度均为O(h)，删点的复杂度为O(h+m)。（其中h为散列表存取的平均复杂度，接近O(1)，m为所删的点的度）
2. **图的最短路算法采用了迪杰斯特拉算法的堆优化**，使得时间复杂度从O(e+v^2)降低到O(e+vlogv)。（其中e为图中边数，v为图中点数）
3. 系统的搜索功能中需要串匹配算法，**实现了Boyer-Moore算法用于串匹配。**该算法预处理复杂度为O(n)，串匹配的复杂度为O(m/n)~O(n)。（其中m为待匹配串的长度，n为模式串的长度）是比KMP算法复杂度更低的算法。
4. 对于系统要求的排序功能，使用了**堆排序**，复杂度为O(nlogn)，其中n为待排序序列长度，相比于插入排序等方式的O(n^2)，快速排序的最坏情况下O(n^2)，堆排序的最坏时间复杂度为O(nlogn)，达到了以比较为基础的排序算法的最优。
5. 系统要求实现一个近似求旅行商问题的算法，采用了任务书中给出的方法，经过修正后实现在了项目中。对于其中需要的最小生成树算法，采用了Kruskal算法
6. 为了支持算法的实现，**自行实现了一系列的容器如向量、双向链表、堆、散列表等，实现了并查集来支持最小生成树算法。**
7. 停车场管理页面采用纯前端实现，使用JavaScript实现了功能完整的栈和队列的数据结构。

#### 可视化界面成果

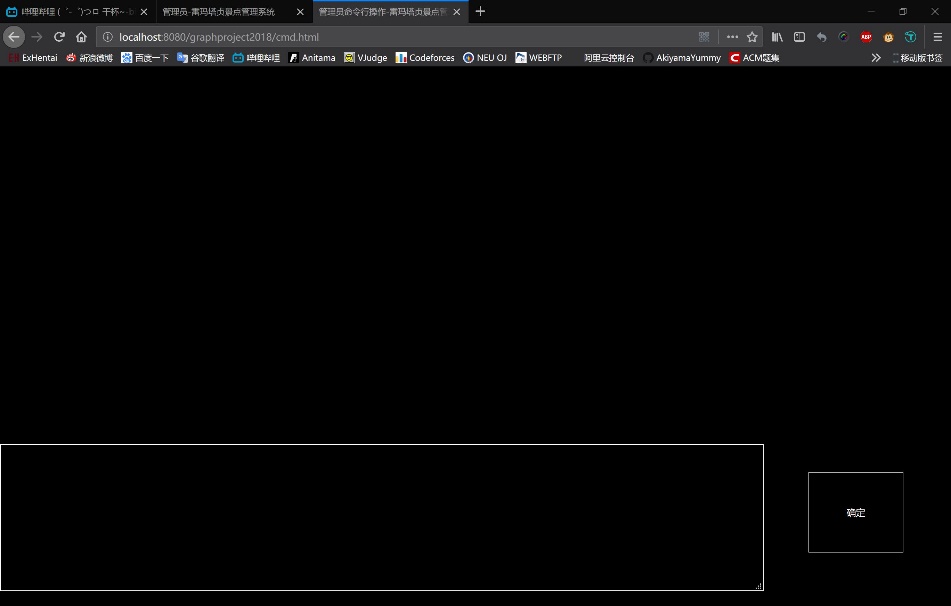
在可视化界面设计的方面，由于我此前已经积累了一些前端开发的经验，所以做出了较为美观的界面

1. 管理员可视化管理页面。管理员可以通过页面增删图中的点和边，并且**可以在该页面自行绘制像素图中的图案，编辑景点和道路的图像**



管理员可视化管理页面

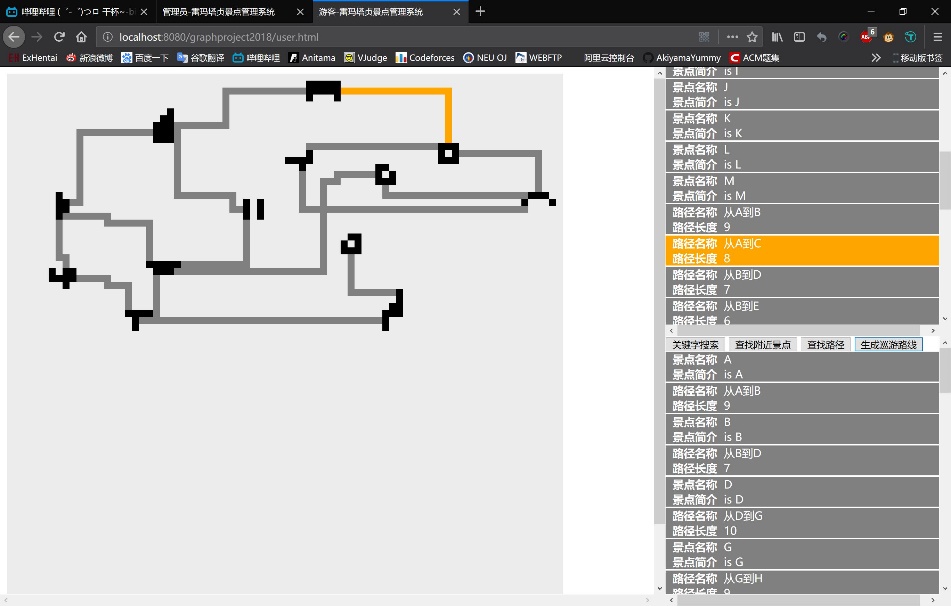
1. 管理员命令行管理页面。由于在算法设计的过程中，我所自行设计的邻接表实现方式，对于点和边的增删复杂度较低，但在可视化管理页面，每次进行图的修改后，后端都会将整个图的信息返回给前端，仍然需要遍历整个图。所以我又额外涉及了命令行管理页面，管理员可以输入一组多条命令，对图进行快速修改，以此可以更好体现出低复杂度的优势。



管理员命令行管理页面

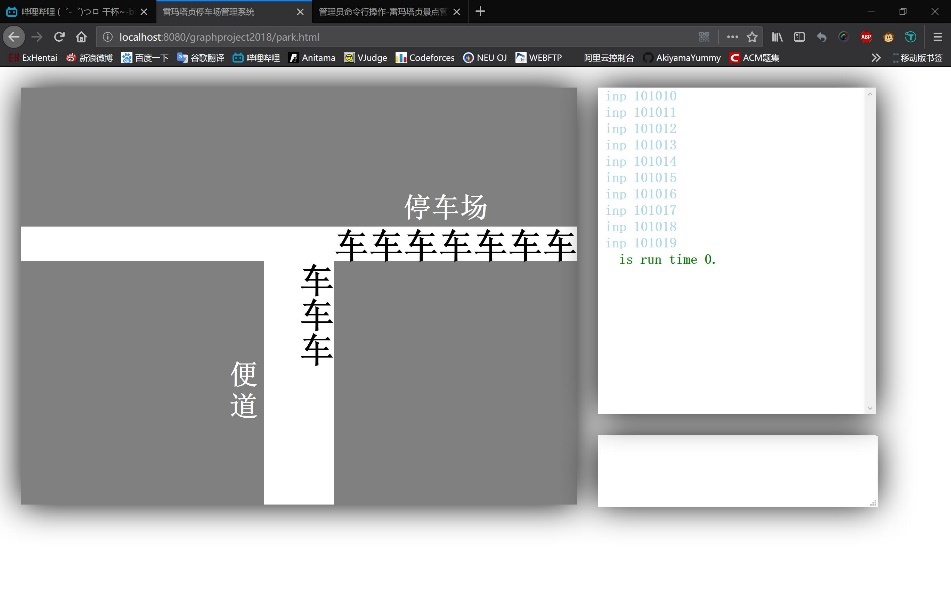
1. 用户页面。用户在鼠标浮动到右侧某节点或道路的模块的位置时，图像中对应位置会有高亮的效果，十分美观。执行每一种功能后，下方搜索结果面板中会出现一系列与执行结果对应的节点或道路的模块。

如图中展示的“巡游路线”功能执行的结果，所展示的模块就依次为“起始景点-到下一个景点的道路-下一个景点-……-起始景点”。



用户页面

1. **停车场管理页面。对于各种情况都考虑周全且设计了连贯优美的动画效果模拟车的运行。**管理员可以在右下方的命令输入栏使用“inp 车牌号”“out 车牌号”这两条命令来模拟停车场的活动。（输入shift+回车换行，输入回车提交命令）



停车场管理页面

## 第二章 系统设计

描述系统中自己的数据结构与算法的设计方案。

### 系统整体设计方案

#### 方案背景

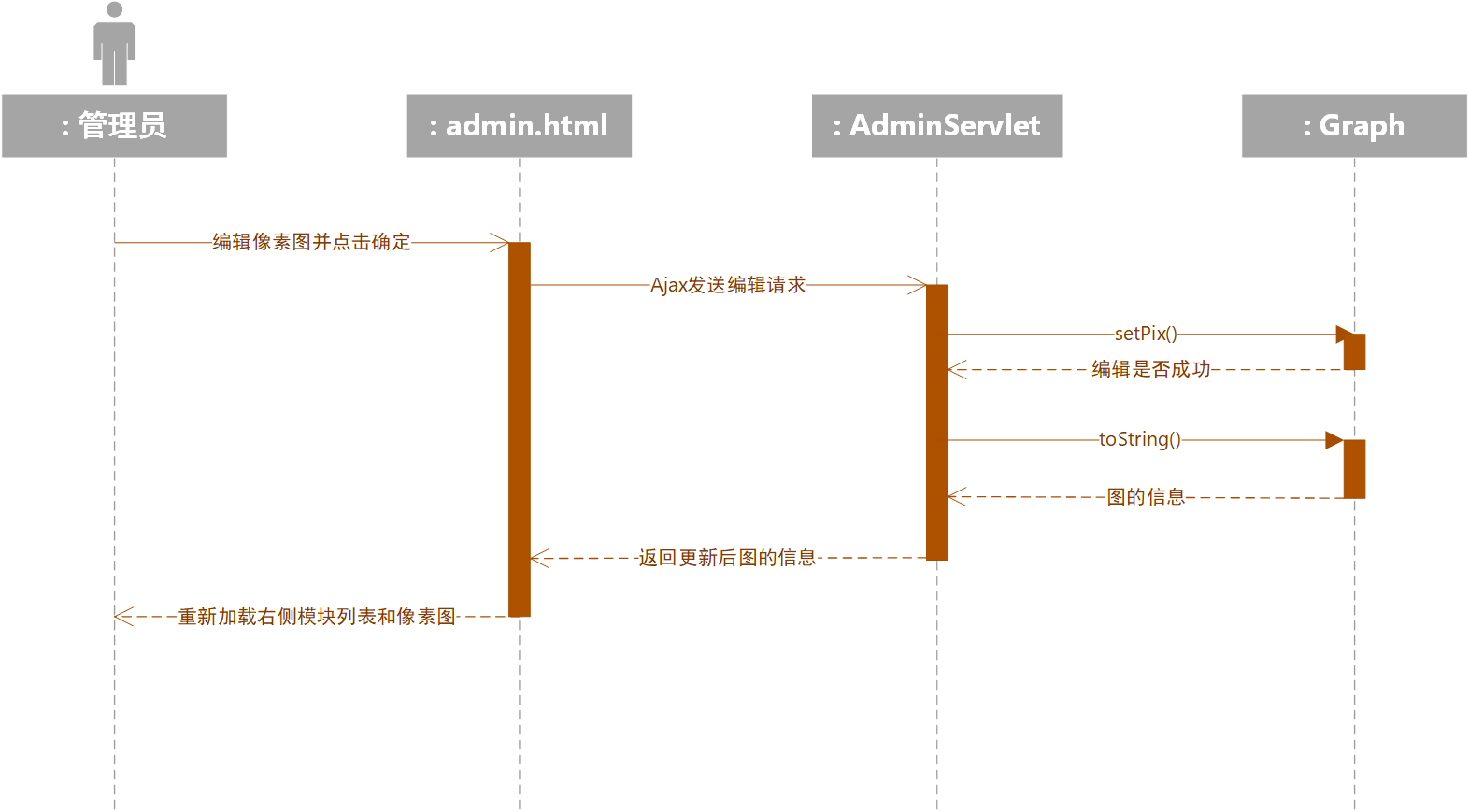
出于我对于Qt和Swing已经比较生疏，所以可视化页面采用网页实现。在此基础上由于我只了解使用Php和Java来进行Web后端开发，不了解使用C++等其他语言的开发方式。所以选择了使用我认为实现算法更加优美和方便的Java来进行后端算法的实现。

#### 方案概述

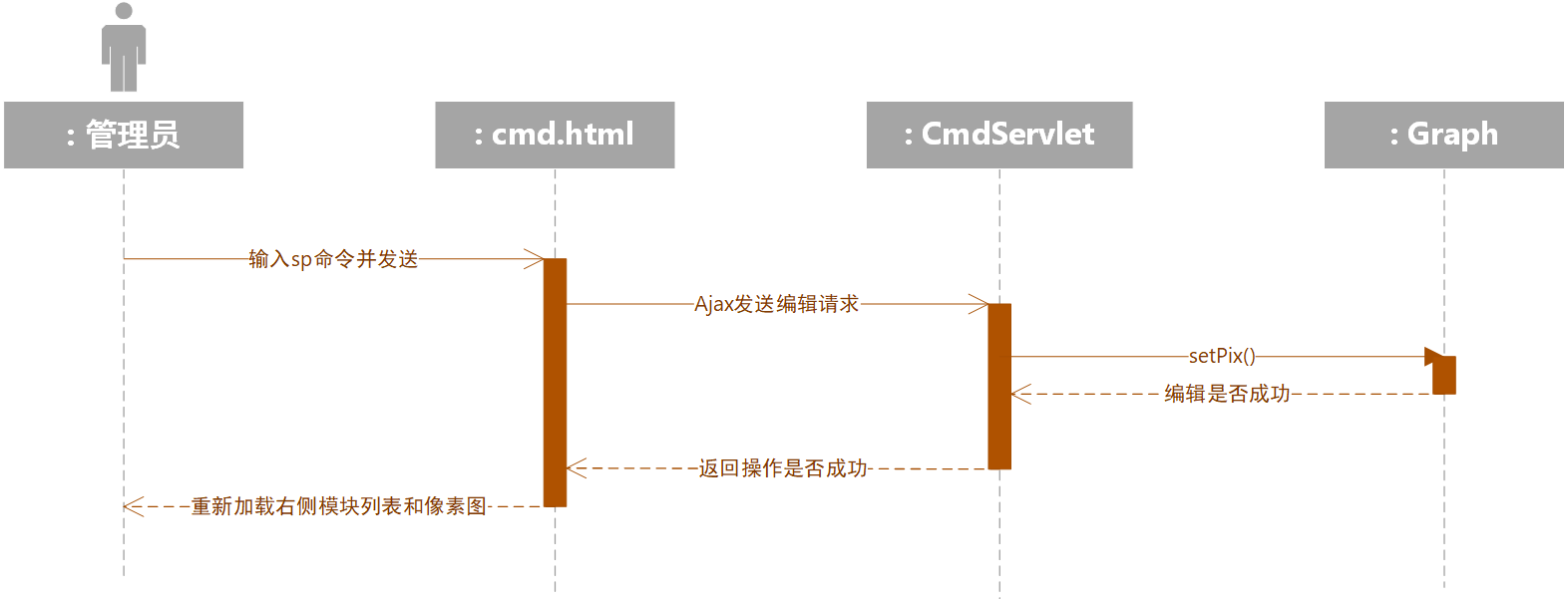
前端的组成为html网页文件，完全使用jQuery库中提供的Ajax方法向后端发送请求。

采用Servlet接收前端的请求，后端含有一个静态的Graph类的对象，Servlet类收到请求后，调用Graph类对象中的对应方法，进行图的编辑和求最短路、求旅行商问题近似解等的操作。将执行结果或前端请求的内容返回给前端。

下面给出两个页面，admin.html以及cmd.html在执行像素图编辑时的时序图。可以看出cmd.html无需调用toString()方法来给前端返回全图的信息，而toString()方法需要遍历全图，复杂度较高，所以说使用cmd.html的命令行管理页面可以提升程序运行的效率，而且可以一次性向后端批量发送多条命令，执行多个编辑操作。



使用admin.html的可视化界面修改像素图



使用cmd.html的命令行界面修改像素图

#### 多线程同步方面的处理

**问题一**

由于系统中涉及文件操作以及静态对象的操作，考虑如何保障多线程同步的安全又成了问题。而我认为该系统的工作重点在于算法的设计，所以做了一刀切的处理， Servlet中代码所有的执行部分，都被同一个对象的同步块所包裹。由此节省时间，把精力都用在了算法设计上。

**问题二**

管理员与用户之间，又极有可能会产生另一种问题。如果用户打开用户界面、管理员编辑图的内容、用户发出诸如求取某两节点之间最短路的请求，这三个事件顺序发生，那么用户端前端收到的响应，和前端的信息不符，可能会发生不可预料的错误。

所以我针对此问题，自行设计出了一个简单的解决方案：

后端存有一个“管理员最后更新时间”的静态变量，每当收到管理员发出的编辑图的请求时，更新该变量为当期时间。

而用户端的前端存有一个“最后得到图信息”的变量，每次执行获得全图信息的请求时，更新为当期时间。每次执行其他请求时，会在请求内容中附上该变量，后端接到该变量，会将收到的该变量，与后端的“管理员最后更新时间”进行比较。如果该前端“最后得到图的信息”的时间，早于“管理员最后更新时间”，不执行前端的请求，直接返回提示信息，提示前端刷新其页面，更新图的信息。

我认为这样的简单设计，也是算法设计能力的某种体现。

### 高效邻接表设计方案

#### 方案背景

本次课程设计中的，对邻接表的实现方式是我自行设计的高效邻接表实现方案。

我考虑到我们在算法竞赛中常用的，数组表示点集，链表表示与点相关的边的方式，这种方式的优点是实现起来十分简单（竞赛选手们写出一份这样的邻接表不会超过十分钟），但一般适用于不需要删点和删边的情况，涉及到删点和删边，复杂度会变得相当之高。

在删点时，自然需要对与该点相关的边进行删除，虽然可以通过遍历待删除的点对应的边，对该点的出边进行删除，但该点的入边，则需要遍历所有其他节点，找到从其他点出发到该待删除点的边对其进行删除，遍历了整个图，复杂度更高。

在删边时，由于该课设中的图为双向图，需要首先遍历该边一端的节点所对应的边，然后遍历另一端节点所对应的边，对边进行删除，才可以完成删除。

而且由于景点可以被删除，也难以给景点编号后存至数组中。

而由我设计的邻接表实现方案，解决了这一系列问题。

#### 方案概述

首先，该方案是基于景点名均不重复而建立的，无法添加重复名字的点。

点集以及和点相关的边，均使用双向链式结构表示。使用散列表建立从景点名字符串到对应点对象的映射。对于边，讲 “景点名1”+某不可能出现于景点名中的字符+“景点名2”字符串作为边名，从边名到边对象的映射，保证景点名1的字典序小于景点名2。

对应于删点的处理，可以用景点名通过散列表找到点对应的对象，遍历该点的出边并删除。对于入边的删除，我做了以下处理，使得复杂度下降到O(h+m)。

在边的数据结构中加入一个指针域，指向该边的反边。在每次加边的过程中都对该指针进行赋值。在删边的时候，可以直接通过指针找到该边的反边并删除。

这样的设计下，从景点名找到对应点的对象以及从边名找到对应边的复杂度仅为散列表存取的平均复杂度O(h)，加点、加边、删边复杂度均为O(h)。删点的复杂度为O(h+m)，流程为先通过散列表找到点的对象并从链表中删除，复杂度为O(h)，然后遍历该点的出边并从链表中删除，遍历过程中通过出边中的反边指针域找到入边对象并从链表中删除，复杂度为O(m)。（其中h为散列表存取的平均复杂度，接近O(1)，m为所删的点的度）

以下为该方法的数据结构示意图。

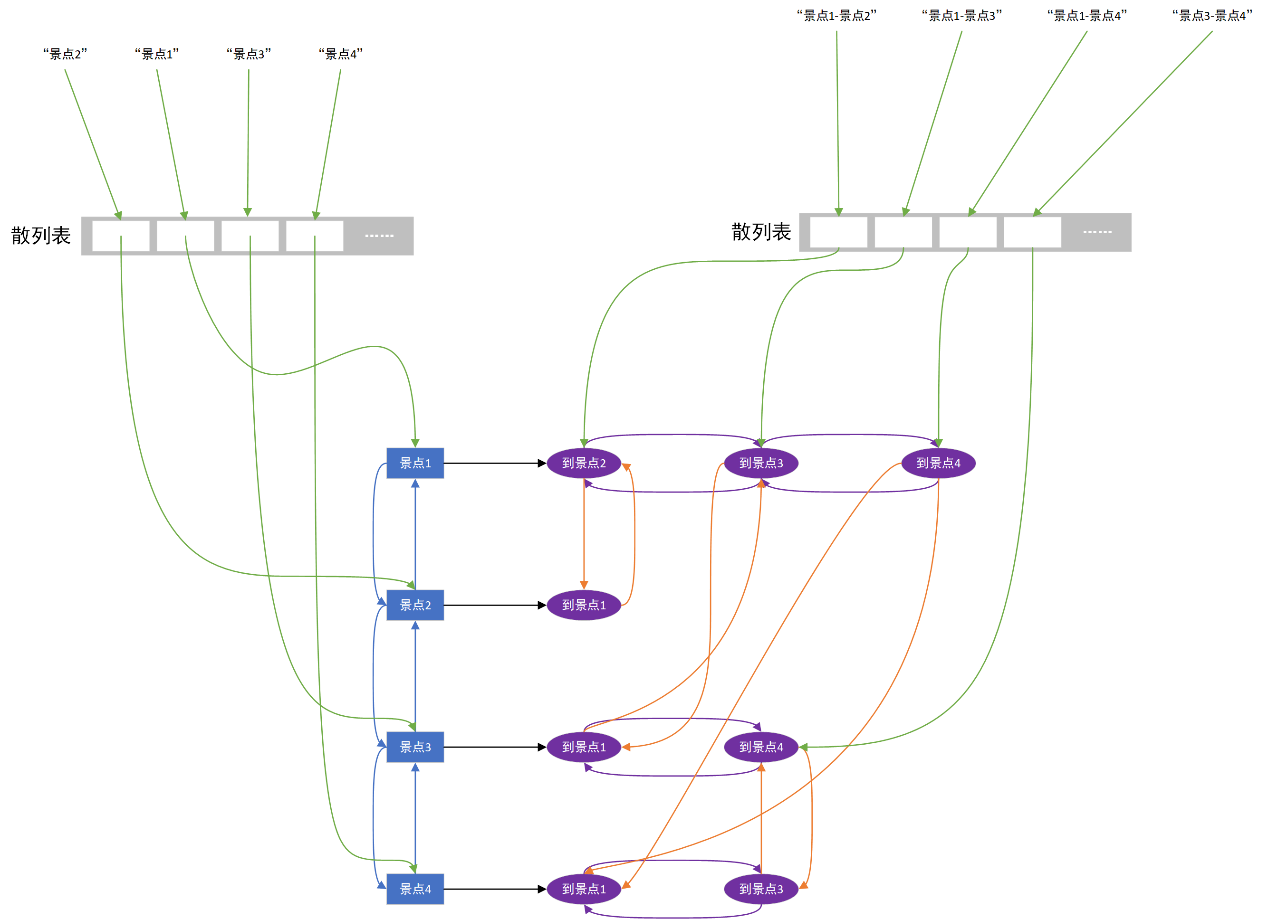
图中灰色框图代表散列表、蓝色矩形代表景点对象、紫色椭圆代表边对象。

图中绿色箭头代表使用散列表从字符串找到目标对象，注意边的对象边名存入散列表的，在一堆户为反的边中只有字典序较小的一个。

图中橙色箭头表示户为反的两个边之间的连接。

图中黑色箭头代表景点对象中含有边对象的一个链表。

图中蓝色箭头意为点集使用了链表表示。紫色箭头为边链表节点之间的连接。



数据结构示意图

#### 复杂度对比

与单纯的数组+链表的经典邻接表实现方式进行时间复杂度对比，结果见下：

表中N为图中景点数，M为图中边数，m为节点的平均度数，h为散列表存取的平均复杂度，在许多情况下h都被认为是一个常数。

需要注意的是，对于经典方式的时间复杂度讨论仅限于理想状况，如果需要防止景点名重复，经典方式的增加景点时间复杂度为O(N)，如果需要防止增加重边，经典方式增加边的时间复杂度为O(N+M)。

|  |  |  |
| --- | --- | --- |
| 执行操作\方式 | 经典方式时间复杂度 | 我设计的新方式时间复杂度 |
| 由数组编号读取点的对象 | O(1) | 无此操作 |
| 由数组编号读取边的对象 | O(m) | 无此操作 |
| 由景点名读取点的对象 | O(N) | O(h) |
| 由起止景点名读取边的对象 | O(N+m) | O(h) |
| 增加景点 | O(1) | O(h) |
| 增加边 | O(1) | O(h) |
| 删除景点 | O(N+M)（M为遍历全图寻找该景点入边的复杂度） | O(h+m) |
| 删除边 | O(N+M)（M为遍历全图寻找该边反边的复杂度） | O(h) |

很明显，我所设计的新方式，在时间复杂度方面远远优于经典的邻接表实现方式。

### 散列表的设计方案

#### 方案背景

在我所设计的邻接表实现方法中，需要散列表的参与。所以我实现了一个散列表。

#### 方案概述

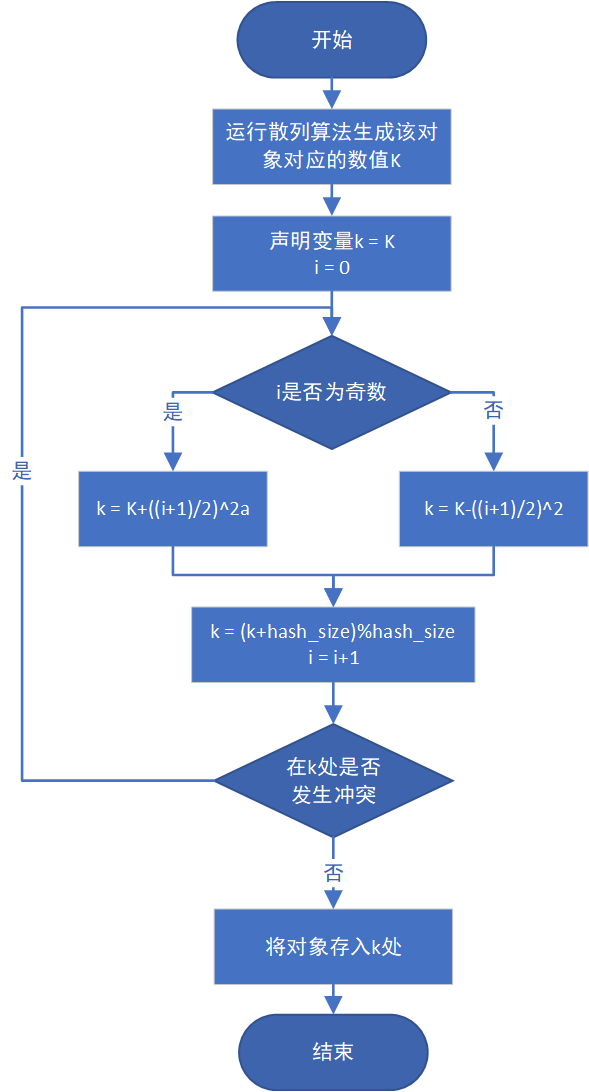
**散列算法**

借用了Java自带的hashCode()函数，因为项目中只需要键值为String类型的散列表，所以也参看了String中hashCode()函数的实现，满足对逻辑的要求后才决定了借用该函数。对hashCode()的返回值进行多次位运算，最后做取模操作。使得获得的数值分散均匀。

**冲突处理算法**

使用了双向平方试探法。方法简述如下。

散列算法得到值为k，此时在k位置发生冲突，则按照如下流程试探。



冲突处理算法流程图

使用这样的试探方法，当散列表的长度为一个对4取模余3的质数时，可以保证当i为hash\_size时，试探算法试探到所有的散列表位置，数学证明略。

**删除元素算法**

对元素的删除选用了惰性删除的方法，该方法实现简单，并且使得算法编写的思路更加清晰，当删除某位置的元素时，不实际改动元素，而将该位置对应的删除标记改变，对应的插入新元素时，也需要改动删除标记。

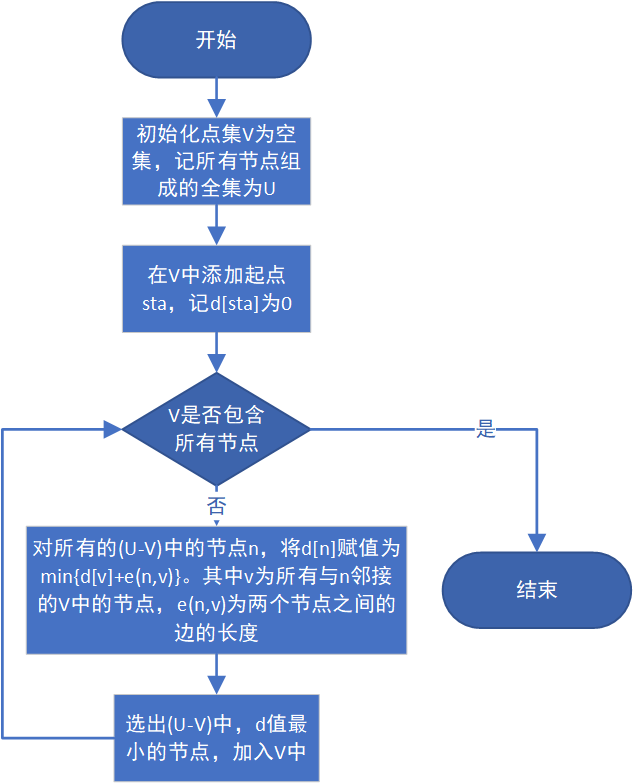
### 迪杰斯特拉算法的堆优化设计方案

#### 方案背景

迪杰斯特拉算法的堆优化，也是在算法竞赛中常见的算法实现方案。此次课程设计中我自行实现的堆（优先级队列）容器，同时用于迪杰斯特拉算法堆优化和堆排序。

#### 方案概述

**迪杰斯特拉算法流程：**



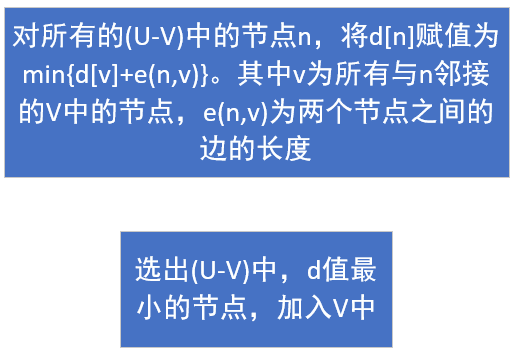
迪杰斯特拉算法流程图

该流程所示程序运行结束后，每个节点的d值，即为起点到该节点的最短路径长度。

**迪杰斯特拉算法证明：**

在上方流程中，算法维护了一个点集V，保证V点集中的节点，其d值等于起点到该节点的最短路径长度。

则我们现在只需证以下两步完成之后，V节点扩充，且依然使得V中的所有节点d值等于起点到该节点最短路径长度，算法的正确性即可得到证明。



需要重点论证的流程

首先我们说明一个引理，加入从起点到某节点n，最短路径途经节点m，则该最短路径的从sta到m的部分，也是从sta到m的最短路径。该引理易证。

则当前情况，(U-V)中节点可以分为两种。

·从起点到该节点的最短路径中该节点的前一个节点，属于V。

·从起点到该节点的最短路径中该节点的前一个节点，属于(U-V)。

由引理易证，当前d值，对于第一种节点来说，等于最短路径长度。对于第二种节点来说，d值大于等于最短路径长度。

记被选中的，(U-V)中d值最小的节点为p。

现在假设p属于第二种节点。

则明显，在从起点到p的最短路径上，存在节点n属于V，节点m属于(U-V)，且n与m邻接。该路径可表示为sta-…-n-m-…-p。

明显可得，p的最短路长度大于m的最短路长度。

由引理得，该路径的一部分，sta-…-n-m，为从起点到m的最短路径。

则m的上一个节点n属于V，则m的最短路径长度为d[m]。

则此时，m的最短路径长度=d[m]<p的最短路径长度<=dp[p]与d[m]>d[p]同时成立。

产生了冲突，所以p属于第二种节点的假设不成立。

则p属于第一种节点，此时d[p]等于sta到p的最短路径长度。

所以此时将p加入V中，V仍满足其性质，算法得证。

**迪杰斯特拉算法（堆优化）数据结构设计：**

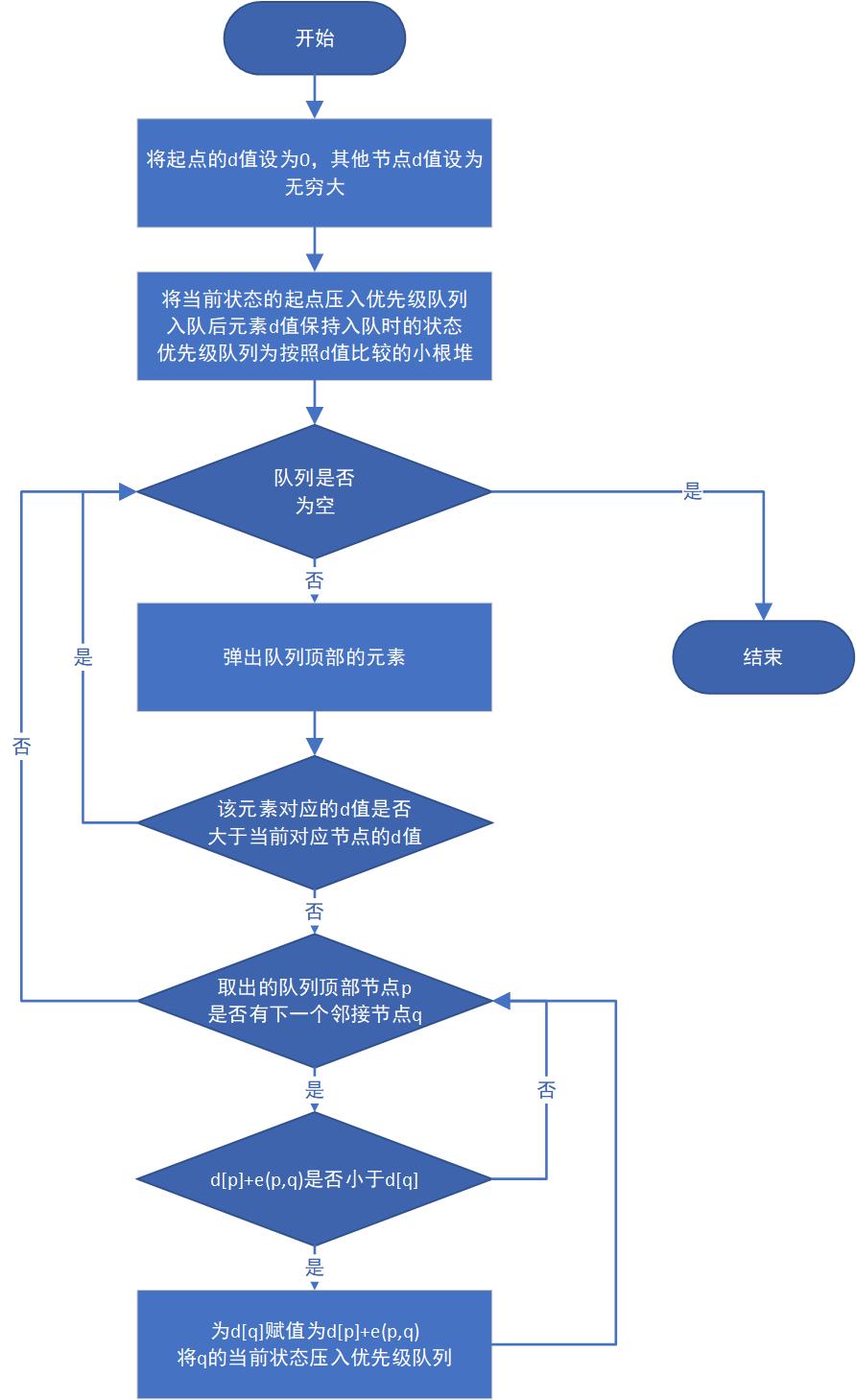
由于上文流程图中，每次循环后，(U-V)中的节点的d值，需要被更新的值仅为与上一轮循环中新加入的节点p’相关的一些，所以算法实际设计中，该业务逻辑可以具体化为：

遍历与p’邻接的所有节点q’，使得d[q’] = min(d[q’],d[p’]+e(q’,p’))。

而其中被更新d值的q’节点，成为该轮选择中待选择的节点。

选择d值最小的节点，该过程可以使用优先级队列

由此，可以得到bfs+堆优化的程序流程。



迪杰斯特拉算法（堆优化）详细流程

#### 复杂度对比

一般的迪杰斯特拉算法实现方法，时间复杂度为O(e+v^2)，而堆优化的实现方式，时间复杂度为O(e+vlogv)。（其中e为图中边数，v为图中点数）

### Boyer-Moore算法设计方案

#### 方案背景

这个算法是我在大一上学期学习邓俊辉老师的数据结构课程中学到的。在邓俊辉老师的《数据结构》中有着详细的论证和讲解。由于原理较为复杂以及期末时间比较紧张，在这里不多做原理方面的论述。

#### 与其他串匹配方案的对比

记待匹配文本串长度为t，模式串长度为p，串中可能出现的字符种数为c。

对比于我们熟知的KMP算法，Boyer-Moore算法突出的优点，就在于极低的时间复杂度。其预处理模式串的复杂度为O(p+c)，进行匹配的时间复杂度为O(t/p)~O(t)，这是一个十分吸引算法学习者的数值。

但KMP算法对比于该算法，也有一定的优势。

比如KMP算法只需要元素之间可以判等，就可以实现，所以对于任何可以判等的对象的序列，都可以使用KMP算法。而BM算法则由于c的限制，只能用于类似ASCII范围内的字符等的匹配。比如对于两个数列进行串匹配，数列当中数字的范围是0~10^9，那么对应于BM算法，c即为10^9，时间复杂度和空间复杂度都无法承受，而KMP算法可以达成。

再比如KMP算法中，预处理生成的next数组，不仅仅可以用来高效串匹配，还有用于判断串的循环节等其他用处。

但应用于该项目中的英文、以及被转码的其他语言组成的景点名和景点简介，而且仅需要进行串匹配，并没有其他要求，我实现了BM算法来进行串匹配，是有其优越性的。

### 停车场模拟程序的设计

#### 方案背景

对于停车场模拟程序，由于较为简单，此次完全前端实现，不经过Java后端的方式。

#### 方案概述

在前端用JavaScript实现了一个循环队列和一个栈，两个数据结构，其中队列用于模拟便道，栈用于模拟停车场和停车场对面用于调车的道路。

在算法流程中，此次我考虑比较周全，考虑到了一辆想要进入停车场的汽车，在尚未进入停车场，在便道等待时就希望离开的情况，以及车辆在同一时间单位内同时发出希望进入和希望离开停车场信号的情况。以下描述的算法对这两种情况均有比较好的解决。

算法在每个时间单位中，都会接到若干条“某牌号汽车希望进入”和若干条“某牌号汽车希望离开”的指令，并对其进行处理。

算法在每个时间单位内执行以下几个部分：

1. 处理进入请求。让所有希望进入的汽车驶入便道。对应数据结构操作，即为汽车的对象进入便道对应的队列。
2. 标记希望驶离的汽车。遍历便道和队列中的所有汽车对象，为该时间单位内法出离开命令的汽车的离开标记赋值。
3. 处理栈中汽车。从栈底遍历栈，找到第一个遇到的希望驶离的汽车，即得到该部分算法中需要离开或暂时离开停车场栈的汽车的数量。

将该数量的汽车弹出停车场栈，如果该汽车的离开标记被赋值，直接通过离开的道路离开，如果并未被赋值，驶入停车场对面的调车道路，即为汽车对象被压入调车道路对应的栈中。

1. 调车道路中所有的汽车驶入停车场中，对应的数据结构操作即为调车道路对应的栈中所有元素弹栈，压入停车场栈中。

处理便道上的汽车，将便道中的汽车循环弹出队列。循环弹出直至以下两个条件至少有一条成立：

* 1. 便道队列为空。
  2. 停车场栈为满，且便道队列队首的汽车，离开标记尚未赋值。

如果被弹出的汽车的离开标记被赋值，直接通过离开的道路离开，如果并未赋值，进入停车场栈。

通过以上算法，可以十分圆满地处理停车场的模拟问题。

### 求旅行商问题近似解的算法设计

#### 方案背景

课设任务书中给出了一种旅行商问题近似算法的设计。引用如下：

void Traceablepath (Graph g)

{

(1)选择g的任一顶点r；

(2)用最小生成树算法找出带权图g的一棵以r为根的最小生成树T；

(3)前序遍历树T得到的顶点表L；

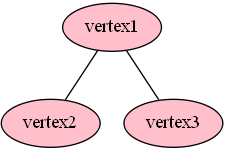
(4)将r加到表L的末尾，按表L中顶点次序组成回路H，作为计算结果返回；

}

但我认为该方法并不完整，存在一个重要缺陷需要实现者自行完成。那就是，该方案生成的巡回路线，路线中相邻的两个节点，之间不一定有边。

我可以举出一个反例。

如下图，



图例

下图中，以vertex1为根节点r，生成的最小生成树包含全图。前序遍历得到的顶点表L为vertex1-vertex2-vertex3，最终获得的返回值H为vertex1-vertex2-vertex3-vertex1。

然后我们注意到，vertex2与vertex3之间并没有路，伪代码中的回路H，并不是一个回路。

#### 方案概述

由于我的确对NP难问题如何求近似解缺乏研究，短时间内无法给出一个旅行商问题近似解的更好的解法，只能在任务书中方法的基础上加以完善。完善过后的伪代码如下：

void Traceablepath (Graph g)

{

(1)选择g的任一顶点r；

(2)用最小生成树算法找出带权图g的一棵以r为根的最小生成树T；

(3)前序遍历树T得到的顶点表L；

(4)将r加到表L的末尾，按表L中顶点次序组成回路H；

(5)遍历H中的第二个到最后一个顶点，循环用迪杰斯特拉算法求出该顶点到H中上一个顶点的最短路，将最短路路径连入H列表的该点和上一个点之间；

(4)将H作为计算结果返回；

}

这样，保证了该算法返回值是一个确实能为游客提供参考的回路。

但该算法不但复杂度较高，而且在实际测试中，出现了获得的路径严重偏离旅行商问题正确解的现象，不能称之为一个好算法。但时间紧任务重，又缺乏重新设计新算法的能力，只能将该算法实现在系统中。

### 其他容器的设计

除了以上这些，系统中我设计实现了向量（可变长数组，用于系统各处，并成为了堆的实现基础）、双向链表（用于高效邻接表的实现基础）、堆（用于迪杰斯特拉算法堆优化以及堆排序）等容器或数据结构。

除了在Java后端中，在前端，我使用JavaScript实现了一个循环队列和一个栈。两个数据结构。

这些容器的实现，均为经典算法中的方式，在此不详细描述。

## 第三章 系统实现

针对自己所设计的方案，描述具体的实现开发内容。

### 关键代码分析

由于系统涉及的实现过程较为复杂，而且我已经在上一章中提及了许多实现部分的内容，所以该部分主要为我所实现的关键代码的分析。

首先给出Vector<T>类中的swap()方法的代码。

**public** **void** swap(**int** ra,**int** rb) {

**if**(ra >= size() || rb >= size())**return**;

T nt = get(ra);

a[ra] = a[rb];

a[rb] = nt;

}

由于Java与我平时用于算法实践的C++不同，不存在传引用到函数中，来实现交换两个对象的方式。而在Heap类的实现中，又需要频繁地对其中的元素进行交换，所以我首先在Vector中实现了swap()方法，传入整数ra和rb，交换Vector中ra位置和rb位置的元素。

然后是因此得到便捷实现方法的Heap<T>中的push()方法。其中v为Heap中的一个属性，顺序储存了该堆对象中应当储存的元素。

**public** **void** push(T nt) {

v.pushBack(nt);

**for**(**int** r=v.size()-1;r>1;) {

Comparable<T> tn=(Comparable<T>)v.get(r/2);

T ts = v.get(r);

**if**(tn.compareTo(ts) != 1)**break**;

v.swap(r,r/2);

r /= 2;

}

}

除了push()方法之外，Vector的swap()方法还在Heap中的pop()方法中得到调用，而且使得代码思路更加清晰，可读性也更强。

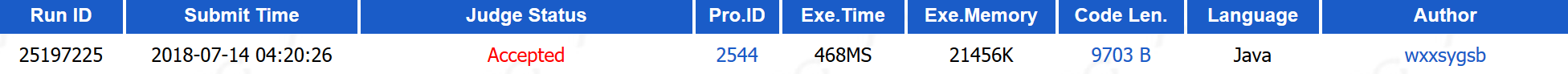
## 第四章 系统测试

系统前后端整体，我使用与任务书中相同规模的数据集进行了一系列测试，测试过程比较琐碎，结果已经在验收时得以展现，系统运行正确流畅，容错性较强。所以该部分主要描述关键算法的测试。

由于个人能力有限，无法编出较为庞大的数据集进行测试，所以我在算法实现过程中想到了使用Online Judge的题目测试我实现的算法。

### 迪杰斯特拉算法的测试

我选用了[HDU2544](http://acm.hdu.edu.cn/showproblem.php?pid=2544)，一道最短路题目，来测试我实现的迪杰斯特拉算法的正确性和效率。[过题代码](https://paste.ubuntu.com/p/3mt7hskcPX/)中用到了几乎所有容器类，以及Graph类，直接证明了我实现的这一系列算法的正确性。



Online Judge网站给出的结果

### Boyer-Moore算法的测试

上文当中也分析过，Boyer-Moore算法相比于KMP等算法，缺陷在于功能单一，所以找到一道可以测试该算法的算法题目比较困难。我记得去年下半年打过一场网络赛，一道伪装成AC自动机题目的算法题可以用BM算法来解决，但想不起具体是哪道题了。

寻找了半天，找到一道十分基础的题目，[SDUT2125](http://acm.sdut.edu.cn/onlinejudge2/index.php/Home/Index/problemdetail/pid/2125.html)，来测试该算法，[过题代码](https://paste.ubuntu.com/p/bHSqsDQspM/)我也贴在了链接里。这道题的通过，的确可以证明我实现的该算法的正确性。但我认为无法说明我实现的该算法在效率上的优越性。不得不说是一个遗憾。



Online Judge网站给出的结果

## 第五章 结论

这次课程设计，虽然有很多遗憾，但能成功完成我也很高兴。是对我算法方面和前端设计方面实力的一个证明。完成了一系列高效优美的算法，实现了富有创意的美观界面，尤其是管理员可以手绘地图展示给用户的这个想法，我认为这是重要的创新之处。

## 参考文献

列出实验中所查阅的相关参考文献。

[1] 数据结构（C++语言版）（第3版）,邓俊辉

[2] 算法导论（原书第3版）,Thomas H.Cormen Charles E,leiserson Ronald L,Rivest Clifford Stein

[3] 数据结构与算法分析:java语言描述（原书第3版）,Mark Allen Weiss

[4] 算法竞赛入门经典（第2版）,刘汝佳

**附录：**

**《数据结构课程设计》实验成绩评定表**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **评价内容** | **具 体 要 求** | **分值** | **得分** |
|
| **平时表现** | **课程设计过程中，无缺勤、迟到、早退现象，学习态度积极。能够主动查阅文献，积极分析系统中数据结构与算法的多种可能的设计方案，并认真地对所选择方案进行实现、测试、分析与总结。** | **20** |  |
| **分析与解决问题的能力** | **能够理解复杂数据结构及算法的设计思路和基本原理；能够应用所学数据结构与算法等相关知识和技能去解决实验系统中要求的各个题目；设计或实现思路有独特见解。** | **20** |  |
| **实验结果与工作量** | **能够按实验要求完成系统的开发与测试，并达到实验要求的预期结果；能够认真记录实验数据，并对实验结果分析准确，归纳总结充分；工作量饱满。** | **20** |  |
| **报告质量** | **实验报告文字通顺、格式规范，体例符合要求；报告内容充实、正确，实验目的归纳合理到位。** | **40** |  |
| **总 分** | | |  |