**北京邮电大学课程设计报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课 程 设**  **计 名 称** | **数 据 结 构** | | **学 院** | **计 算 机** | **指导教师** | **王尊亮** |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成 绩** | |
| 2020211313 |  | 2020212895 | | **简旭锋** |  | |
| 2020211313 |  | 2020212900 | | **凌梓超** |  | |
| 2020211313 |  | 2020212910 | | **严云鹤** |  | |
| **课**  **程**  **设**  **计**  **内**  **容** |  | | | | | |
| **学 生**  **课程设计**  **报 告**  （附 页） |  | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩:**  指导教师签名：  年 月 日 | | | | | |

**课程设计报告表**

|  |  |  |
| --- | --- | --- |
| 组别 | | 6 |
| 系统架构（采用的编程语言、是否多线程、是BS/CS/Android哪种架构） | | 架构：BS 编程语言：C++17、Javascript 支持多线程 |
| 课程信息管理 | 完成的基本功能描述 | 主要由简旭锋同学实现 （1） 课程数目要求 （2） 由课程名称/表查询课程信息/资料信息/作业信息 （3） 上传和更新课程资料与完成的作业内容（凌梓超帮助实现） （4） 使用效果较好的算法进行相关资料文件的压缩/查重/去重（凌梓超、严云鹤帮助实现） （5） 统一安排考试与上课的时间、地点 （6） 使用课表图形界面方式进行课程管理和查询 （7） **没写的：排序、版本管理** |
| 核心算法描述、算法优缺点及性能 | （1） 教务系统所有课程/资料/用户信息管理：由自主实现的轻量级数据库系统支持。底层算法依赖数据结构B+树存储文件并支持高效查找。提供了较为优秀的性能。 （2） 资料的上传／下载：由自主实现的后端服务器与浏览器的通信实现，通信依赖HTTP协议。通过BS架构将用户上传的文件统一保存到服务器端并在数据库处记录 （3） 文件的查重/去重/压缩：服务器接收文件后使用系统自带指令对其进行压缩，再通过MD5信息摘要算法实现文件的唯一标号并存入数据库。去重通过比较各文件的MD5编码是否相同实现。 |
| 课外信息管理 | 完成的基本功能描述 | 主要由简旭锋同学实现 （1） 课外活动数目要求 （2） 以不同方式查询课外活动信息 （3） 设定不同的活动闹钟用于提醒 （4） 检测时间冲突并给出提示 （5） **没写的：排序** |
| 核心算法描述、算法优缺点及性能 | （1） 课外活动信息的保存与查询：将相关信息保存在后端服务器的数据库模块中，相关信息由后端提供，由数据库提供较好支持 （2） 活动闹钟提醒：通过Vue框架提供前端的原生支持，依赖watch命令与前后端时间的统一。 （3） **检测时间冲突：** |
| 课程导航 | 完成的基本功能描述 | 主要由严云鹤、凌梓超同学实现 （1） 以更真实的地图实现建筑物/服务设施等要求 （2） 根据课表与指定地址、导航策略以图示的可视化形式输出导航线路 （3） 通过性能更好的寻路算法实现三种要求的导航策略 （4） 实现校区间的路线规划，包括班车时刻表等信息 |
| 核心算法描述、算法优缺点及性能 | （1）  寻路算法/不同导航策略：A\*，综合性能被认为是常见场景下最优秀  （2）  班车等校区间通信方式：与虚拟时钟系统联动，由后端统一的虚拟时间和保存的班车时刻表返回最近的一班车 |
| 模拟系统架构、模拟时钟推进思想 | 完成的基本功能描述 | 主要由凌梓超同学实现   1. 实现系统时钟的推进 2. 系统时间精确到分钟 3. 独立时钟下的多用户并行推进 |
| 核心算法描述、算法优缺点及性能 | 1. 虚拟系统时钟：后端实现以事件为核心的时钟系统，以广播的形式实时统一修改所有用户的时钟速率 |
| 日志功能 | 完成的基本功能描述 | 主要由凌梓超、简旭锋同学实现   1. 日志文件记录用户状态变化 2. 展示和查询日志信息 |
| 核心算法描述、算法优缺点及性能 | 1. 核心算法：在后端服务器中统一记录并保存于数据库中，在查询时由数据库提供支持返回所有文件 |
| 选做功能一 | 完成的基本功能描述 | 各种功能的图形界面  主要由简旭锋、严云鹤同学实现 |
| 核心算法描述、算法优缺点及性能 | 浏览器前端：使用Vue作为前端框架整理展示信息  地图文件：由软件与贴图素材生成还原度较真实的版本 |
| 选做功能二 | 完成的基本功能描述 | 课表图形界面方式进行课程管理和查询  主要由简旭锋同学实现 |
| 核心算法描述、算法优缺点及性能 | 使用Vue作为前端框架整理展示信息 |
| 选做功能三 | 完成的基本功能描述 | 自主实现的数据库模块  主要由简旭锋同学实现 |
| 核心算法描述、算法优缺点及性能 | 核心算法：借鉴sqlite等轻量数据库，以B+树为数据结构独立实现一个可正常工作的数据库，底层算法依赖数据结构B+树存储文件并支持高效查找。提供了较为优秀的性能。 |

**更具体的分工：**

简旭锋：主要负责数据库的自主实现，教务系统/日程表的模块、文件相关操作模块的实现和前端可视化的设计

凌梓超：主要负责前后端的分离和相关通信，设计并维护了服务器的架构和通信的过程，为各后端模块设计上层的调用和实现，帮助实现了诸如文件操作、压缩去重、寻路业务逻辑与可视化等模块。

严云鹤：主要负责如文件查重模块、地图寻路模块及其底层算法等一些后端模块的底层实现和地图的UI设计与可视化等部分。同时主要负责撰写、积累开发过程中的相关文档。

**功能需求报告**

1. **课程信息管理查询**

* 学生可以输入课程名称，或者通过课程表查询课程的上课时间、上课地点、课程资料、当前进度、已交作业、待交作业、课程群、考试时间和考试地点等信息。
* 学生可以上传和更新课程资料、上传完成的作业内容。对于课程资料和作业可以进行查询、压缩和去重。
* 系统管理员统一发布考试时间和考试地点，并可以发布和更改课程上课的时间和上课地点。

**需求分析**：

* 1. 不同的学生当然会有不同的课程安排以及作业、考试等信息，这些信息应当能够作为文件长久保留。
  2. 学生（以及教师）自己的账户信息（学号等）和各课程的资料、进度、考试等信息彼此之间相对独立但又有联系。同时这些信息预计在底层也会随着用户操作被频繁地增删查改。故在底层以一个关系型数据库对这些需固化信息进行维护是理想的。在此基础上，用户账户的信息和选课的信息会以不同的主键索引被各自组建成表，需要时可以通过数据库充分实现对相关资料的查询需求。因此本小组决定实现一个自己的数据库模块以满足底层信息维护的需求。
  3. 由使用北邮教务系统的体验出发，小组决定完整实现一个web项目的架构。由此一个自然的总体设计就是BS架构，用户通过浏览器进行所有的操作，前端则通过网络通信与后端的多线程服务器交互数据。由此随后也确定了学生上传、更新资料、作业等需求的基本实现方式：以HTTP协议传输文件等所有必要内容。
  4. 在有效组织信息的基础上，可以把作业、资料等内容以合适的文件夹目录组织起来，比如 课程id/教师id/班级id/作业id/学生id/文件。在前端需要时先从数据库中查询信息，若被记录存在则在对应的目录中找到文件并回送给前端即可实现对资料文件传输的需求。
  5. 文件的压缩与去重：压缩文件本小组决定调用后台系统的自带命令实现，对每个收到的文件进行自动压缩处理。文件去重（不保存同样的文件）本小组在查阅资料后了解到：可以使用MD5算法对所有文件生成唯一的标识，不同文件的MD5值可以认为是不同的。通过对接收到的文件进行扫描得到编码，并与已保存的所有文件编码进行比较，可以实现检测重复文件的需求。
  6. 系统管理员统一发布考试信息并维护课程信息：如果实现数据库的支持，则这一需求只需要为管理员开设一个独立的后台，允许管理员更直接地修改后端数据库保存的信息即可大体实现。所有用户的信息都依赖于后端数据，管理员的改动是可以传达到所有用户的。

1. **课外信息管理查询**

* 学生可以输入课外活动信息，课外活动包括个人活动和集体活动；个人活动可以包括：自习、锻炼、外出等，集体活动可以包括：班会、小组作业、创新创业、聚餐等。
* 学生可以设定活动闹钟，闹钟可以是一次性的和周期性的，用于活动提醒。
* 系统可以检测个人活动、集体活动和课程的时间冲突，并给出提示。

**需求分析**：

* 1. 课外活动与课程其实在“日程”的角度区别没那么大，都是占用一段具体时间的一件事，而一天的时间总量是固定的。同时课程的占用时间也是相对固定的。如果日程划分相对统一或者比较接近，那么对课外活动进行增删查改跟对课程进行增删查改的业务逻辑差别不是很大，而后者已在前一块需求中实现了
  2. 个人活动与集体活动的区别应该是：后者在一位用户添加新课外活动时影响到所有参与集体活动的用户。同时检测时也应当检测所有用户的相关冲突情况，但这只是在业务逻辑上的差别，不涉及数据结构与算法。
  3. 检测活动的时间冲突（没写）
  4. 闹钟应与整体的虚拟时间模块联动，后端统一时间并将信息送给前端时，若检测到用户设定的时间值后只需交给前端提醒即可。（没写多少）

1. **课程导航**

* 学生向系统输入课程名称、上课时间或者上课地点，系统自动进行路径规划。
  + 课程名称可以是“数据结构”，系统会自动根据学生的班级信息和最近的上课时间（未开始）查询上课地点
  + 上课时间可以是“周五10点”系统会自动更据学生的班级信息和最近的上课时间（未开始）查询上课地点
  + 上课地点是上课的物理位置，例如 “教学楼N204”
  + 起点和终点可以在不同校区
* 学生可以在系统中选择导航策略，系统提供最短距离策略、最短时间策略以及包含交通工具的最短时间策略。系统会根据所选策略输出展示线路。

**需求分析**：

* 1. 这一模块与之前的日程／课程模块的需求相对独立，联系仅来自于“课程地点”与“上课时间”，故这些信息可以被统一地规范，体现在地图寻路上。时间的统一是比较容易理解的，而地点则相对固定，可以将各类不同的地点文字说明（如“教学楼N204”）映射为更底层的信息（比如地图坐标）
  2. 寻路得先有路。需要在底层规定整个地图的数据结构，在此基础上才能实现寻路算法，并给出较好的显示效果。以游戏中的寻路机制为灵感，查阅资料后本小组了解到了游戏中使用的寻路机制和地图呈现方式。在基本的“点——边”的图结构上可以更进一步以贴近真实效果：把2D的地图整理成以坐标为联系的各基本节点的二维数组集合。坐标上邻近的节点即是彼此连通且边权统一的。在此之上可以用不同的节点类型来区分“路”和“地点”。 找到的路径则可以通过输出所有经过节点的坐标来实现。
  3. 在寻路算法方面，本小组查阅资料了解到实际应用场景中性能优秀的A\*算法。其是常见寻路算法的改进，可以整合进前述的地图中，并可以通过参数的调整实现需求中不同导航策略的功能。由此基本的寻路模块可以正常工作
  4. 在此基础上，多校区的导航可以通过指定各校区的转换地点实现，比如走到校门位置则进行校区跳转。班车则结合时钟模块，查询后台服务器保存的班车时刻表与当前时间给出班车时间等相关信息。如此则所有需求都可以实现。

1. **模拟系统时间**

* 系统依据时钟向前推进，时间精度为小时，可以自定义以计算机的n秒数作为模拟系统的1小时，支持快进。

**需求分析**：

* 1. 因为支持快进，故“时间”应当为虚拟时间。前述的各模块都涉及到时间的操作，而且其他信息都需通过后端处理。故为了统一各模块的时间操作，虚拟时间也应该由后端服务器统一安排。
  2. 前端查询时应当可以由后端返回时间协调，但不能在这上面消耗太多的数据交互资源，前端拿到时间后也应可以自己独立控制时间的向前推进，这可以在前端架构中实现。
  3. 同时在多线程的BS架构下，所有用户的操作都需要同一服务器的支持。故其中一个用户（管理员）修改前端时钟的流速时，服务器也应当调整其他用户的时间流速。这意味着服务器主动向前端发送信息。查询资料后，本小组复现了以事件为核心的时钟系统，通过comet技术实现服务器对所有用户的实时广播以修改时钟流速。

1. **建立日志文件**

* 记录学生课程和活动状态变化，系统提醒的信息，输出的导航信息，以及学生输入的信息和各种查询操作。

**需求分析**：

* 1. 因为所有的操作多需要通过后端的数据交互，故在规定了日志的格式后，只要每次交互后令后端保存一条信息进文件中存储起来即可。可以划分不同的用户日志文件以实现相对更清楚的划分
  2. 如果有少数的仅需前端处理的信息，只需要此时设置主动给后端发送日志的相关信息，将日志文件的维护交给后端即可。

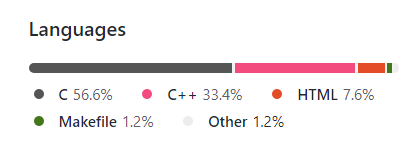
**总体方案设计报告**

### 开发环境

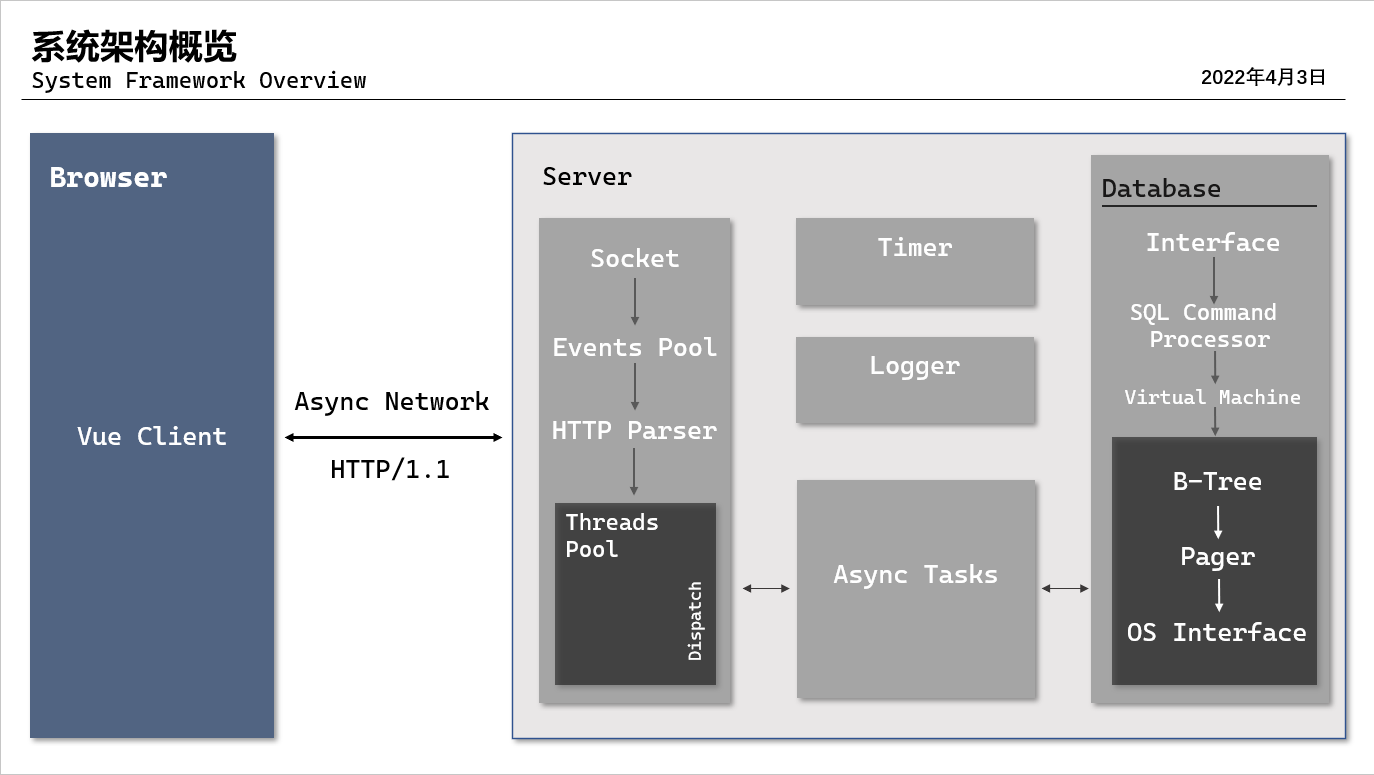
本项目为BS架构。主要开发环境为GNU/Linux。服务器端主要使用Linux下提供的库函数和系统调用实现后端服务器的稳定运行，网页端则以常见的web开发模式设计了一个功能完整的网页，支持在常见平台的浏览器上正常工作。

### 开发语言

本项目实现后端各模块使用的语言主要为C/C++（C++17），前端则以Vue为框架，即以HTML等前端语言来实现效果。



### 总体框架与模块划分



#### 前端

本项目采用网络编程模式，以Vue为主要的开发框架实现客户端网页的呈现。

#### 后端

项目后段将负责对课设所有主要需求的实现，并通过HTTP协议形成服务器——客户端交互的设计模式，将用户在客户端中请求的资源经过后端的调用发送回给用户，包括时间推进、日志等需求分析中的主要功能都在后端实现。

#### 数据库模块

项目自行编码设计了小型数据库**NEDB**(Not Enough DataBase)并设计sql语句，完成基于数据库的数据存储、删除、修改和查询操作，通过后端模块完成前端与底层数据库的数据通信。

项目的教务系统及相当一部分的主力需求的实现依赖于一个合适的数据库系统。本项目首先通过研究SQL语句，具体地指sqlite软件的接口实现与数据库的交互，并会通过进一步的研究，自行编程实现一个小型的数据库**NEDB**，并完全通过这个自主设计的数据库实现项目的所有需求，支持后端的各类功能。

#### 地图寻路模块

项目在基本的寻路算法之上进行了升级，以重现常在游戏软件中（如红色警戒等）呈现的地图视觉效果和寻路的可视化为出发点进行模块设计。为此采用了A\*算法，在节点——边的寻路模式上进行升级，更好地模拟了完整的两校区地图，实现了更清晰的地图和路径的可视化。在此基础上较好地实现了需求的各类导航策略

#### 文件查重/压缩模块

项目在查阅了有关资料后自行实现了MD5算法作为底层信息摘要算法支持包括用户密码加密、文件查重等各模块的需求。在此基础上实现了文件的唯一标号识别，进而可以通过比较数据库内文件的MD5编码判断是否出现重复。文件压缩模块则调用了Linux系统下自带的gzip命令实现快速的压缩。

#### 时钟模块

时钟模块主要基于“服务器推”的Comet技术，后台可以令服务器通过HTTP长连接的形式实现对所有客户的时间速率调整，进而实现了对虚拟时钟的需求。

#### 教务系统/日程模块

略

**数据结构说明报告**

#### 数据库模块

**性能分析**

本小组数据库模块的底层依赖于B＋树，故在不考虑常数项的前提下，性能分析的重点也是B＋树。

查阅资料后我们知道，B＋树因较为契合实际IO中的局部性原理，被认为可以有效减少对外存的访问次数，故性能相对优异，被广泛使用于各类文件系统、数据库中作为底层的算法。

具体来说，相关的复杂度为：

算法 平均 最坏的情况下

空间 O( n ) O( n )

搜索 O(log n ) O(log n )

插入 O(log n ) O(log n )

删除 O(log n ) O(log n )

#### 寻路模块

**数据结构**

我们小组在实现地图模块和寻路算法中采用的数据结构为：

新的地图模块的基本单位（数据结构）是“节点”，通过节点识别地图中为“可走”(walkable，即路)和“不可走”(unwalkable，即障碍)的两种坐标类型

节点结构体

struct Point2D {

  int x   = 0;

  int y   = 0;

  int dis = INF;

  Point2D\* parent = nullptr;

};

节点结构体保存着对应的地图坐标，以及为最后寻路需要而设计的指向最短路径前一节点的指针。

地图结构

所有的节点将组成支持寻路的模拟地图，模拟地图是一个大量节点组成的二维数组，数组的每个元素代表对应坐标的相关信息，生成节点地图的所有必要信息将被提前保存进头文件中。

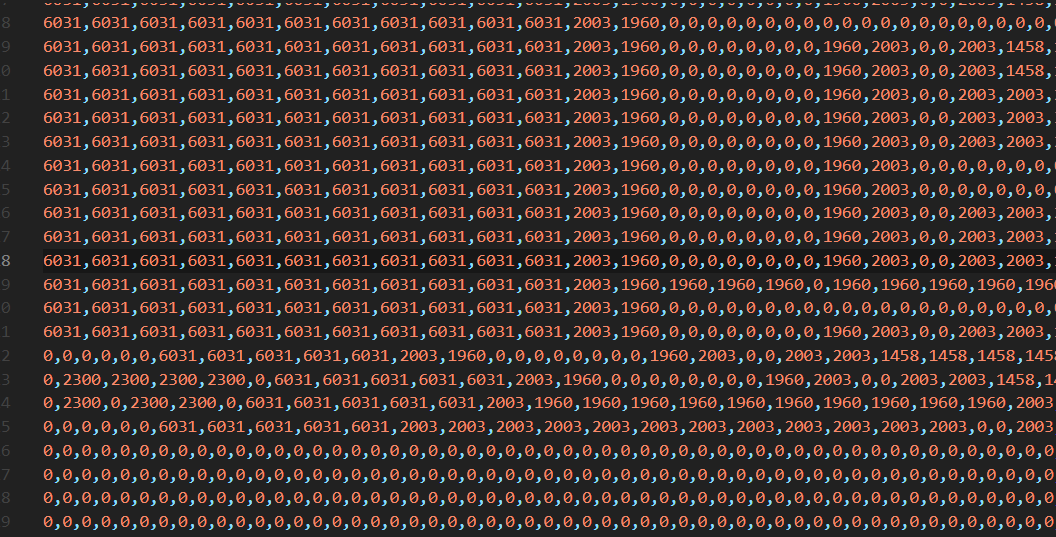


图 1 截自shahe\_walking.h

在shahe\_walking.h中保存的地图中，所有的可走路径均为0，以此区分可走/不可走的节点



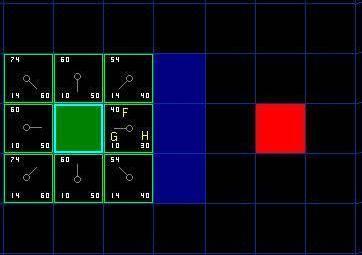
图 2 截自shahe\_walking\_busy.h

在shahe\_walking\_busy.h中保存的地图中，上一图中标为0的（路径）节点将保存所有路径节点的拥挤度信息，可以用于对下文中寻路代价的调整。

**寻路算法**

在地图数据结构的基础上本小组独立实现了A\*算法的设计。其的核心思想是：

* 1. 确定起点和终点
  2. 维护两个节点堆，open list和close list，其中open list主要包含所有未被处理的路径节点，close list主要包含已被处理的和不可走的节点
  3. 计算出组成路径的方格的核心是计算“代价”。其中对于特定可走节点，其的代价包括：
     1. 从起点经过已知最短的路径（最小代价）到达该特定节点的移动代价，记为G
     2. 无视障碍物的前提下该特定节点到达终点的估计代价（比如通过曼哈顿距离实现），记为H
     3. 总代价即为二者之和，记为F=G+H
     4. 代价的示例：



对于这个演示用的例子，深绿色为起点节点，其周围的八个基本节点有不同的移动代价。在这里，与起点直接相邻的上下左右四节点，G值被记为10，斜方向节点的G值则是14（对根号2=1.414的比例放大近似处理）。不同的代价代表了基本移动距离的长短。而对于H值，以曼哈顿距离为例，起点右边方格的H值为30，右边节点的总代价即为F=10+30=40，其代表了“如果路径经过这个节点，从起点到终点的预计总移动路程/时间”。

* 1. 在这样的基础上，寻路的步骤可以被描述为：
     1. 将起点加入open list
     2. 循环：从open list堆中取出现在总代价F最小的节点，将其加入close list中
        1. 检查它的所有相邻方格，忽略不可走的方格和已被处理的方格
        2. 如被检查的相邻方格节点未被加入open list，在当前处理节点的代价基础上，计算出它的G、H、F值，记录到达它的路径（即，在节点数据类型中保存一个指针，在此步令指针指向当前处理节点，使其成为父亲，在上面的演示图中，所有的八个节点都将指向起点）

具体来说，当前节点主要影响新加入节点的G值：以演示的代价计算为例，假设当前节点的G＝14，而新加入节点在当前节点的斜方向，则它的G值便是14+14=28，G值起到了“记录从起点经过某特定节点到达这一新节点的总移动代价和”的作用。而H值则是新节点自己的曼哈顿距离估计，相对孤立。

* + - 1. 如被检查的相邻节点已被加入open list，意味着它已经保存了一条经起点到达它的路径，这时候要检查是否经过当前节点到达它的路径会更短（即代价更小，当前节点的G值+处理相邻节点的G值<该节点已经保存了的G值），如果是，则更新该相邻节点的G值（进而影响总代价F）和路径（令该节点内保存的路径指针改为指向当前节点），使得每一个在未加入新节点时，open list中节点存下的路径和移动代价总是局部最短/小的。
    1. 如此重复，将地图内的节点一直加入openlist中生成路径。停止的条件是：
       1. 终点已经被加入到了open list中，此时已经找到了我们需要的最短路径
       2. 查找终点失败，open list为空，此时没有任何路径
    2. 当找到到达节点的最短路径后，我们可以输出它：自终点开始，经由指针一步步地遍历途径的路径节点直到起点，便有了可用的路径。

具体在我们小组的实现中，没有使用斜方向的节点惩罚计算，仅使用了在上下左右四个方向移动的代价。

**性能分析**

由对A\*算法的分析我们可以知道，其属于一种启发式搜索算法，其“通过更新已知的最短路信息得到最终需要的最短路径”也是一种动态规划思想的体现，能够保证找到最优路径。可以把Dijkstra算法视为其H值恒为0的特例。且其每次重复操作中“取出list中F值最小的节点”的步骤不难看出，其典型的实现依赖于堆排序。

分析并查阅相关资料后可以知道，A\*的时间复杂度取决于我们使用的启发式。在无界搜索空间中的最坏条件下，如果终点是可达的，则扩展的节点数在解的深度（最短路径）d : O()中呈指数增长，其中b是分支因子（每个状态的平均后继数），否则算法将不会终止。

最坏情况下时间复杂度：

最坏情况下空间复杂度：

更好的启发式（即我们估计节点到终点路径的值H）则可以允许A\*修剪掉更多不必要的枝。在有效分支因子b\*最低（等于1）时启发式性能也会最优。最低有效分支因子可以通过扩展生成的节点数N和解的深度来确定，可解方程(d是最短路径或解的深度)：

最优状态下，当搜索空间为一棵树，终点单一，启发式函数h满足以下条件时，时间复杂度为多项式：

其中h \*是最优启发式，即从x到目标的确切成本。换句话说，h的误差不会比返回从x到目标的真实距离的“完美启发式” h \*的对数增长得更快。

而A\*的空间复杂度则与其他的图搜索算法大致相同，因为我们使用时仍需要将其的所有节点都保存在内存中，这一点是可以改进的。

查阅资料与相关的讨论中我们知道，A\*在相当多的场合中已经是最优的算法，在一些常见问题中也优于Dijkstra等常见算法，被广泛地用于各类游戏程序中的寻路问题。因此可以认为小组成员对课程知识的拓展估计是比较成功的。

查阅其他文献中提到了一些在真实场景实际应用中提升速度的方法，如提前划分孤岛、改用更小的地图、预载入一些现有路径等方式，在实际使用中本小组的A\*算法工作效率仍算满意、地图的大小也不算特别大，这些措施可以作为日后改进的方向。

**各模块设计报告**

#### 前端

#### 后端服务器通信

#### 数据库模块

**概述**

本小组的数据库模块为基于C++自主实现的超轻量数据库NEDB(Not Enough Database)。参考常在嵌入式系统使用的sqlite数据库的底层原理与InnoDB引擎复现关系型数据库底层使用的B+树数据结构，并在独立实现B＋树后设计复用了查询用的相关sql语句，在sql语句的机制下实现了数据库[增-删-查-改]的基本操作，以此为上层教务系统等模块提供接口和较高的性能支持。

源代码可见于： <https://github.com/Jianxff/NEDB>

在编写时参考的资料有：

作为简易的数据库，NEDB的底层在B＋树结构数据处理的基础上，以行为最小单位进行数据存储，默认大小为0.4KB的页为操作的基本单位，树的叶子节点存有数据页起始位置在数据文件中的偏移量，一次I/O操作可将整页数据读入内存，在操作完毕后重新写入。

**模块划分**

其设计的各子模块可以被分为：

| **模块** | **内容** |
| --- | --- |
| Console | 控制台运行控制 |
| Process | 输入输出流, 命令解析与执行 |
| Data | 数据表, 数据页, 数据行基础操作 |
| Operation | 增, 删, 查, 改基本操作 |
| IO | 文件读写存储 |
| NEexception | 异常处理 |
| BalanceTree | 底层索引B+树 |
| BasicUtils | 快排、日志等小工具集 |

**项目实现功能**

在设计结束时，数据库模块主要实现的功能包括：

1. 基本表结构：规定了数据库操作的相关数据表格式。

具体地，一张数据表将包括以下两个文件

table\_name.nef 表结构文件, 包含表名和字段信息等

具体的文件格式为：

|  |
| --- |
|  |
|  |
|  | ---------------------- |
|  | FileHead 表信息 |
|  | 文件指针头 |
|  | ---------------------- |
|  | TableHead 表数据数 |
|  | 表数据类型:ID+Type |
|  | 表主键 |
|  | 总页数 |
|  | ---------------------- |
|  | TableBody B+树 |
|  | 页偏移量+页索引 |
|  | ---------------------- |
|  | FileTail TAIL标识 |
|  | ---------------------- |

table\_name.ned 数据文件, 包含写入表格的所有数据

具体的文件格式为：

----------------------

|  |
| --- |
|  |
|  | FileHead 页索引 |
|  | ---------------------- |
|  | PageHead 页已满信息 |
|  | 页数据最小值 |
|  | 页数据最大值 |
|  | 页数据量 |
|  | 页数据光标位置 |
|  | ---------------------- |
|  | PageBody 行数据 |
|  | ---------------------- |
|  | FileTail TAIL标识 |
|  | 下页文件指针偏移量 |
|  | ---------------------- |

1. 终端控制机制：

实现对已保存的数据库文件的终端形式控制。//这段不知道怎么表达

1. 命令解析器：

对元命令以及外部的基本SQL语句进行解析的相关机制，主要参考了sqlite的类似结构。

具体地，可以分为对终端操作的元命令和对数据库文件操作的一般命令。

元命令以’.’开头，支持的元命令包括（需严格小写）：

| **命令** | **参数** | **效果** |
| --- | --- | --- |
| .help | - | 显示命令帮助信息 |
| .debug | 调试等级 | 设置调试等级(参数为空则显示当前调试等级) |
| .dir | - | 显示当前数据加载目录 |
| .setdir | 文件夹路径(-d:default) | 设置数据加载目录 |
| .dirinit | - | 自动检查及创建当前加载目录 |
| .open | 数据表绝对路径(无后缀) | 从相应目录加载相应数据表至内存 |
| .mount | 数据表名称 | 从当前加载目录加载相应名称数据表 |
| .mountall | - | 从数据加载目录加载所有数据表 |
| .unmount | 数据表名称 | 取消挂挂载相应数据表 |
| .size | - | 显示当前单数据页最大字节数 |
| .setsize | 数据页字节数(-d:default) | 设置单数据页最大字节数 |
| .exit | - | 退出 **NEDB** 控制台 |

支持识别的SQL语句包括：

* 1. 创建数据表

语句格式为:

create table 'table\_name' (

'Parm\_Name' 'Parm\_Type',

...

);

其中：

* + 1. 将默认输入的首个元素为主键，并可以在相应的元素类型后加key关键字以指定主键
    2. 不支持将longtext类型的数据设置为主键
    3. 创建指令时若有同名数据表存在于数据库文件中将创建失败
  1. 删除数据表

drop table 'table\_name';

其将会直接删除整个数据文件

* 1. 获取数据表结构

describe table 'table\_name';

此操作将返回该数据表单页最大字节数, 字段名称和字段类型

* 1. 列出所有数据表

select tables;

此操作将返回加载到内存中的所有表名称

* 1. 增加（插入）新数据

insert into 'table\_name' ('parm\_name\_1', ...) values ('value\_1', ...);

其中要求插入的数据必须指定主键，在前一参数表中需包含主键字段

同时也可以略去对参数的指定

insert into 'table\_name' values ('value\_1', ...);

此时参数个数应与表格字段个数相同, NEDB 会按照表格字段顺序进行填充.

指定字段有误时，数据库会返回对应的错误信息

* 1. 删除数据

delete from 'table\_name' where 'condition\_1' and ...;

在判断条件时用的where语句若省略则会清空全表

And前后需要加上空格

* 1. 查询数据

select 'parm\_name\_1', ... from 'table\_name' where 'condition\_1' and ...;

要取出数据行的所有字段, 请使用 \* 代替指定字段名称.

当 where 语句省略时, NEDB 会遍历全表进行搜索.

注意在 , 和 and 前后需要加上空格.

* 1. 修改数据

update table\_name set 'parm\_name\_1' = 'value\_1', ... where 'condition\_1' and ...;

主键值不能被修改.

当 where 语句省略时, NEDB 会将全表数据进行相应替换.

注意在 and 前后需要加上空格.

1. 命令执行器

对传入语句解析结果进行判定，并执行对应的数据库操作

1. B+Tree

详见文档数据结构说明报告部分

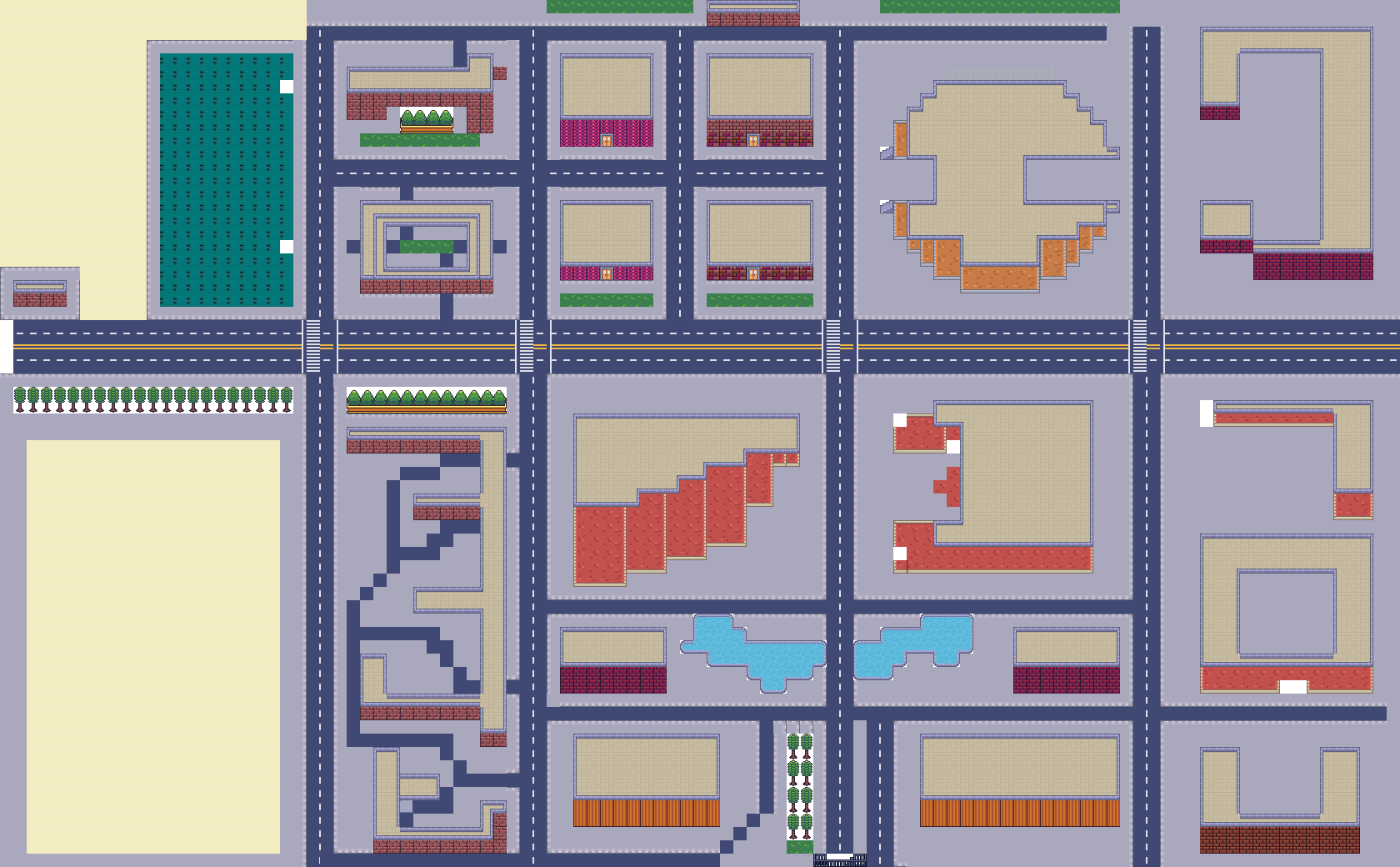
1. 文件存取
2. 异常
3. 外部接口
4. 任务锁

#### 地图寻路模块

**概述**

结合需求中“使用真实的校园地图信息”、“和常规不一样的导航算法”等要求，基于总体架构的良好GUI风格，小组成员开始探索比课本引入的“节点——边”模式更好的地图可视化方式。以游戏软件中经常出现的2D地图设计和寻路模式为灵感，小组成员查阅了相关资料，发现了在帝国时代等游戏中使用的寻路算法A\*，自行实现后以此为基础设计了全新的地图模块。

同时通过用于编辑游戏地图的Tiled软件和相关的素材，本小组设计了真实感与还原度更好的地图，以沙河校区为例：



**地图设计**

新的地图模块的基本单位（数据结构）是“节点”，最基础的节点会被分为“可走”(walkable，即路)和“不可走”(unwalkable，即障碍)两种类型，它们作为单个像素点形成了整个地图。尽管在路径表的基础上进行了升级，但最底层依然是一个较大的二维数组，这就方便了上层的呈现方式。

进一步地，对需求中不同寻路策略的支持也可以非常方便：只需在原本的节点类型上进行细分，对于不同的路径节点给予不同的判定条件（详见寻路算法部分）即可。

同时，这一机制的引入也有利于上层的可视化：最底层的“地图”只是一个有不同数字的二维数组，理论上，只需要在前端的呈现中令数组中不同类型的数字分别对应不同的贴图，就可以呈现出一个较为精细的可视化效果。

在此基础上，利用其他软件，小组成员较好地复现了沙河校区与西土城校区的地图，其精细度相比常规寻路算法的底层有较明显的提升。

其中，沙河校区为65\*105个节点组成的数组，西土城校区为75\*110个节点的数组。

细分的地图节点类型包括：

入口

主干道

人行道

绿化带

教学楼

学生宿舍

辅助设施

体育设施

水

不可进入的工地与围墙

**寻路算法**

在地图数据结构的基础上实现了A\*算法的设计。A\*算法的思想和一些性能分析已经在数据结构说明报告中给出。这里要补充的是本小组的地图寻路模块实现更高需求的方式。

比较底层的需求是要支持几种不同的导航策略。在细化了路径类型的基础上可以较容易地实现。我们的路径细分为支持自行车的主道和人行道。

一般来说：各种策略与拥挤度的区别与衡量，都只需要体现在节点移动时的惩罚值大小上。

最短距离策略：因为只是单纯距离最短即可，故此策略下各路径节点类型的惩罚值是统一的，不细分，只有可走/不可走的限制。基本即距离最短=经过的总路径节点数最少。

最短时间策略：通过调整各路的惩罚值实现对不同道路类型、不同道路拥挤度的模拟。经过参数的调整后，基本上最短时间=经过路径的总惩罚值最小。

交通工具的最短时间策略：在此策略下对路径细分为主干道和人行道，自行车只能走主干道，同时调整走自行车道的速度比人行道要更高，结合最短时间策略中的拥挤度模拟后便实现了交通工具的最短时间策略，仍是经过路径的总惩罚值最小，只不过该模式下惩罚值有了更细微的调整。

本小组对惩罚值的实现最后体现在不同的地图上，不同寻路策略下使用的地图会有区别。可以考虑在此基础上进行改进，以更巧妙的设计将较重复的地图文件整合为同一张。

**可视化输出**

在成功实现寻路的基础上，本小组进一步地实现了路径和地图的可视化输出，地图的可视化前文已经提及过，搜索免费素材后可以生成更精细化的地图，随后最简单的方式就是作为一张图片直接表现出来，日后可以考虑通过节点类型的一一对应给出更精细的地图节点呈现和选择方式。

进一步地，在此基础上也实现了寻找路径的可视化输出。//

#### 文件查重/压缩模块

**概述**

分析需求并查阅现有资料后，本小组选择使用MD5算法作为信息摘要的核心。在自行复现了MD5算法后，依赖MD5作为文件查重的核心。同时整合Linux系统自带的gzip命令实现了文件压缩功能。

故现在“对课程资料/作业进行压缩、去重的需求”可以被实现为：将用户上传的文件传输到后端服务器后，经过模块自动地进行压缩，得到的压缩文件会在内存中，经过C++的文件流操作被读入进缓冲区，扫描了全部文件内容后生成唯一的MD5码，该编码将会随着文件一同被保存。在去重时，只需要将生成的MD5编码同现有的各编码进行比较，即可判定是否唯一。由以上这些，基本实现了文件维护相关的所有内容。

**md5模块设计**

#### 时钟/闹钟模块

#### 教务系统/日程模块

#### 排序模块

**测试报告**

**评价和改进意见报告**

#### 评价

#### 改进意见

* 1. 在业务逻辑上进行优化，因为时间较为仓促，本小组的逻辑结构上存在较多可以优化的地方
  2. 可以通过调整前端的互动方式支持更精细的地图操作和地图界面。
  3. 可以自主实现更多底层的诸如vector、string等基本要素代替STL等现成提供的数据结构和算法，在实现软件的基础上更契合数据结构课程设计的目标。
  4. 重构一些底层的数据结构（如B+树）以优化复杂度的常数项
  5. 调整前后端通信的机制优化信息的交互
  6. 通过压力测试等方式检测后端服务器的可靠性
  7. 解决可能存在的BUG

**用户使用说明报告**