程序设计实践课程报告

姓名：赵伯远
学号：211440128
班级：人工智能 2101
序号：75

2023 年 6 月 8 日

目录

1 课题一：运动会分数统计

1.1 任务描述

参加运动会有 n 个学校，学校编号为 1……n。比赛分成 m 个男子项目和 w 个女子项目。项目编号为男子:1～m，女子:m+1～m+w。不同的项目取前五名或前三名积分；取前五名的积分分别为：7、5、3、2、1，前三名的积分分别为：5、3、2；哪些项目取前五名或前三名由学生自己设定。（m<=20, n<=20）

1.2 功能要求

1. 可以输入各个项目的前三名或前五名的成绩；
2. 能统计各学校总分；
3. 可以按学校编号、学校总分、男女团体总分排序输出；
4. 可以按学校编号查询学校某个项目的情况；
5. 可以按项目编号查询取得前三或前五名的学校。
6. 允许用户指定某项目采取其他名次的取法。

1.3 需求分析

此程序主要实现的功能有：

1. 收集每个学校的男女团队在各项目中的得分。
2. 按照学校编号、总分、男女团体得分进行排序。
3. 提供查询功能：查询指定学校在指定项目中的得分，查询指定项目取得前几名的学校。

首先需要构建顺序表储存相关信息，如参赛学校的名称，获得的分数，比赛项目，及其相关的赋分规则，以及参赛选手信息。然后需要输入比赛相关信息，如参数学校，比赛项目的相关信息，至此基础信息完备。在项目结束后输入相关获奖人员信息即可按照积分给出各种排名。

程序流程图如下：

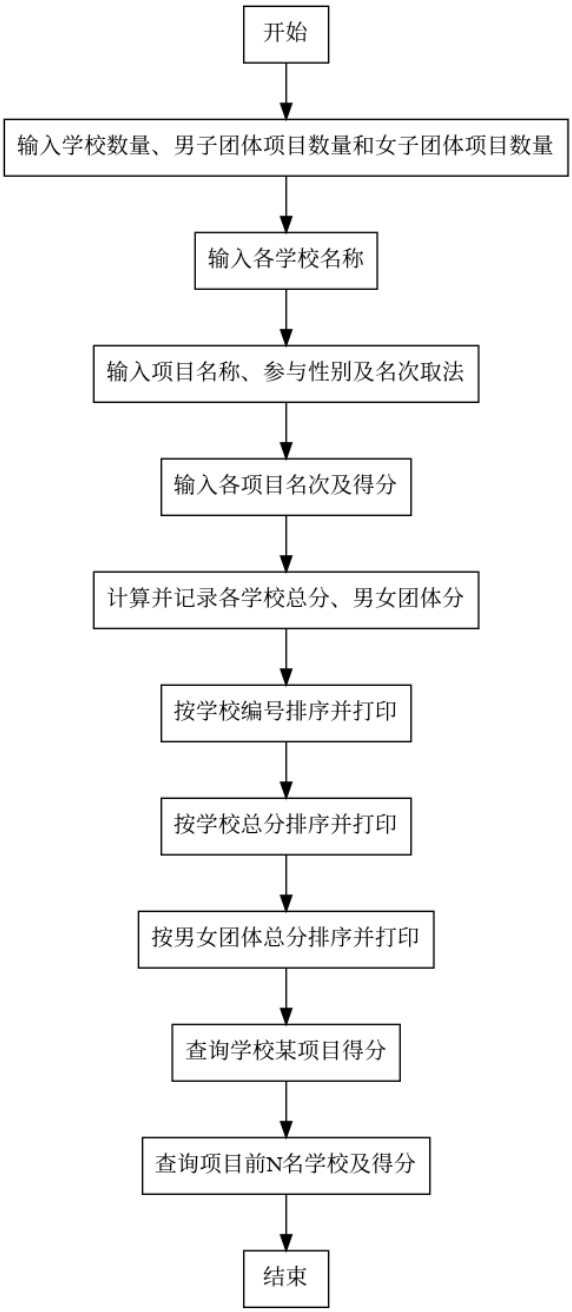
开始
输入学校数量、男子团体项目数量和女子团体项目数量
输入各学校名称
输入项目名称、参与性别及名次取法
输入各项目名次及得分
计算并记录各学校总分、男女团体分
按学校编号排序并打印
按学校总分排序并打印
按男女团体总分排序并打印
查询学校某项目得分
查询项目前N名学校及得分
结束

图 1.1: 程序流程图

1.4 概要分析

```cpp
//定义学校结构体，包含学校的信息和比赛得分
struct School {
    int id;  // 学校ID
    string name;  // 学校名字
    int total_score = 0;  // 学校总得分
    int male_score = 0;  // 男子团体总得分
    int female_score = 0;  // 女子团体总得分
    map<int, int> event_scores;  // 按项目编号存储每个项目的得分
};

//定义比赛项目结构体，包含项目的信息和每个学校在该项目中的得分
struct Event {
    int id;  // 项目ID
    string name;  // 项目名字
    string gender;  // 项目性别分类，m表示男子项目，w表示女子项目
    int num_rankings;  // 用户指定的名次取法，例如前三名或前五名
    vector<pair<int, int>> school_scores;  // 每个学校在该项目中的得分，pair中第一个元素为学校ID，第二个元素为该学校在该项目中的得分
};

//比较函数，用于按照学校ID排序
bool compare_by_id(const School& a, const School& b) {
    return a.id < b.id;
}

//比较函数，用于按照学校总得分排序
bool compare_by_total_score(const School& a, const School& b) {
    return a.total_score > b.total_score;
}

//比较函数，用于按照男女团体总得分排序
bool compare_by_male_female_score(const School& a, const School& b) {
    if (a.male_score != b.male_score) {
        return a.male_score > b.male_score;
    }
    return a.female_score > b.female_score;
}
```

图 1.2: 程序基本结构体 & 基本函数

1.5 详细分析

完整代码如下：

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <string>

using namespace std;

//定义学校结构体，包含学校的信息和比赛得分
struct School {
    int id;  // 学校 ID
    string name;  // 学校名字
    int total_score = 0;  // 学校总得分
```

```
    int male_score = 0;  // 男子团体总得分
    int female_score = 0;  // 女子团体总得分
    map<int, int> event_scores;  // 按项目编号存储每个项目的得分
};

//定义比赛项目结构体，包含项目的信息和每个学校在该项目中的得分
struct Event {
    int id;  // 项目 ID
    string name;  // 项目名字
    string gender;  // 项目性别分类，m 表示男子项目，w 表示女子项目
    int num_rankings;  // 用户指定的名次取法，例如前三名或前五名
    vector<pair<int, int>> school_scores;  // 每个学校在该项目中的得分，
                                           //pair 中第一个元素为学校 ID，
                                           //第二个元素为该学校在该项目中的得分
};

//比较函数，用于按照学校 ID 排序
bool compare_by_id(const School& a, const School& b) {
    return a.id < b.id;
}

//比较函数，用于按照学校总得分排序
bool compare_by_total_score(const School& a, const School& b) {
    return a.total_score > b.total_score;
}

//比较函数，用于按照男女团体总得分排序
bool compare_by_male_female_score(const School& a, const School& b) {
    if (a.male_score != b.male_score) {
        return a.male_score > b.male_score;
    }
    return a.female_score > b.female_score;
}


int main() {
    int n, m, w;
    cout << " 请输入学校数量、男子团体项目数量、女子团体项目数量：" << endl;
    cin >> n >> m >> w;
```

```
    cout << " 请输入学校名称： " << endl;
    vector<School> schools(n);
    for (int i = 0; i < n; ++i) {
        schools[i].id = i + 1;
        cin >> schools[i].name;
    }
    cout << " 请输入项目名称，参与性别及名次取法： " << endl; // 让用户指定每个项目的名次取法
    vector<Event> events(m + w);
    for (int i = 0; i < m + w; ++i) {
        events[i].id = i + 1;
        // 获取用户指定的名次取法
        cin >> events[i].name >> events[i].gender
            >> events[i].num_rankings;
    }

    for (auto& event : events) {
        for (int i = 0; i < event.num_rankings; ++i) { // 根据用户指定的名次取法计算学校得分
            int school_id, score;
            cout<< " 请输入项目 " << event.name << " 的第 "
                << i + 1 << " 名学校编号和得分： " << endl;
            cin >> school_id >> score;
            event.school_scores.push_back({school_id, score});

            School& school = schools[school_id - 1];
            school.total_score += score;
            if (event.gender=="m") {
                school.male_score += score;
            } else {
                school.female_score += score;
                cout<< school.female_score<<endl;
            }
            school.event_scores[event.id] = score;
        }
    }
    // 输出学校编号排序
    sort(schools.begin(), schools.end(), compare_by_id);
    cout << " 按学校编号排序： " << endl;
    for (const auto& school : schools) {
        cout << school.name << " (编号： " << school.id << ")" << endl;
```

```
92      }
93      cout << endl;
94
95      // 输出学校总分排序
96      sort(schools.begin(), schools.end(), compare_by_total_score);
97      cout << " 按学校总分排序: " << endl;
98      for (const auto& school : schools) {
99          cout << school.name << " (总分: " << school.total_score << ")" << endl;
100     }
101     cout << endl;
102
103     // 输出男女团体总分排序
104     sort(schools.begin(), schools.end(), compare_by_male_female_score);
105     cout << " 按男女团体总分排序: " << endl;
106     for (const auto& school : schools) {
107         cout << school.name << " (男子团队分数: " << school.male_score
108              << ", 女子团队分数: " << school.female_score << ")" << endl;
109     }
110     cout << endl;
111
112     // 查询学校某个项目的情况
113     int query_school_id, query_event_id;
114     cout << " 请输入查询的学校编号和项目编号: " << endl;
115     cin >> query_school_id >> query_event_id;
116     const School& query_school = schools[query_school_id - 1];
117     auto it = query_school.event_scores.find(query_event_id);
118     if (it != query_school.event_scores.end()) {
119         cout << query_school.name << " 在项目 " << events[query_event_id - 1].name
120              << " 中的得分为: " << it->second << endl;
121     } else {
122         cout << query_school.name << " 在项目 " << events[query_event_id - 1].name
123              << " 中没有得分" << endl;
124     }
125     cout << endl;
126
127     // 按项目编号查询取得前三或前五名的学校
128     cout << " 请输入要查询的项目编号: " << endl;
129     int query_event_id2;
130     cin >> query_event_id2;
```

```
131     const Event& query_event = events[query_event_id2 - 1];
132 
133     cout << " 在项目 " << query_event.name << " 中取得前"
134          << query_event.num_rankings << " 名的学校有：" << endl;
135     for (const auto& school_score : query_event.school_scores) {
136         cout << schools[school_score.first - 1].name << " (得分："
137              << school_score.second << ")" << endl;
138     }
139 
140     return 0;
141 }
```

1.6 调试分析

1. 问题：在计算学校得分时，学校编号的索引与实际编号存在偏差。

   解决方案：在读取学校编号后，需要将其减 1，以符合实际索引。

2. 问题：按学校编号排序输出时，学校编号不是按照升序排列。

   解决方案：可以在排序函数 sort(schools.begin(), schools.end(), compare_by_id) 之前添加调用 compare_by_id 函数的输出语句，检查是否正确比较学校编号。

3. 问题：程序无法正确查询取得前三或前五名的学校。

   解决方案：根据用户输入的项目编号，获取该项目的参与学校列表，并根据学校的得分进行排序。然后输出前三或前五名学校的名称和得分。确保在处理边界情况时进行适当的错误检查。

1.7 用户手册

1. 演示程序的运行环境为 Ubuntu 20.04 系统，GCC 9.4.0 x86_64-linux-gnu.。执行指令为

   cd ”/media/zby/SSD 数据盘/Program-Practice/Sport/” && g++ main.cpp -o main && ”/media/zby/SSD 数据盘/Program-Practice/Sport/”main

图 1.3: 初始界面

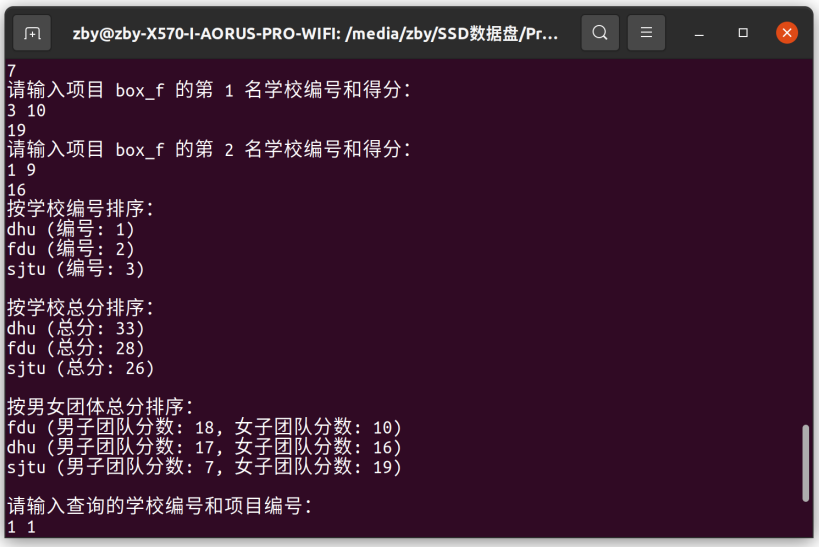

图 1.4: 输入数据

图 1.5: 查询操作

1.8 总结

本次实验我不拘泥于顺序结构的实现，同时去了解了 STL 中的 map 和 set 容器的使用，以及对于迭代器的使用。在实验过程中，我对于 STL 中的 map 和 set 容器的使用有了更深的理解。同时，我也对于 STL 中的 sort 函数有了更深的理解。

2 课题二：停车场管理系统

2.1 任务描述

设计一个停车场管理系统，模拟停车场的运作，此程序具备以下功能：

(1) 若车辆到达，则显示汽车在停车场内或者便道上的停车位置；

(2) 若车辆离去，则显示汽车在停车场内停留的时间和应缴纳的费用（在便道上停留的时间不收费）

2.2 基本要求

1. 要求以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入和输入数据序列进行模拟管理。
2. 要求处理的数据元素包括三个数据项：汽车“到达”或“离去”信息，汽车牌照号码及到达或离去的时刻。
3. 要求栈以顺序结构实现，队列以链表实现。

2.3 需求分析

主要流程为由用户输入进入停车场的车辆的信息，例如车牌号和进入时间以及离开时间，然后程序会自动生成所需要缴纳的费用。

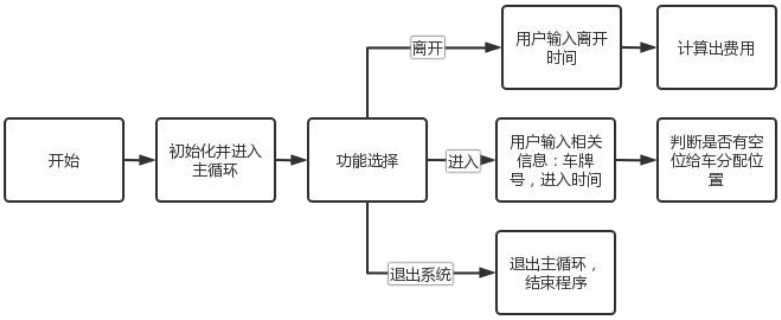

图 2.1: 程序流程图

由于没有外部的辅助设备，帮忙输入相关信息，这里让用户输入的方式来进行完成。

2.4 概要设计

```
1    typedef struct {
2        int hour;
```

```
    int min;
  } time;
  typedef struct {
    int num;
    int position;
    time t;
    float money;
  } Car;
  typedef struct Node {
    Car data;
    struct Node *next;
  } CQueueNode;
  typedef struct {
    Car elem[NUM + 1];
    int top;
  } Stack;
  typedef struct {
    CQueueNode *front;
    CQueueNode *rear;
  } LinkQueue;
  void InitQueue(LinkQueue *Q);
  // 初始化队列
  int EnterQueue(LinkQueue *Q, Car *t); // 进队
  void InitStack(Stack *S);
  // 初始化栈
  void Push(Stack *S, Car *r);
  // 压栈
  int IsEmpty(Stack *S);
  // 判断栈空
  int IsFull(Stack *S);
  // 判断栈满
  int GetTop(Stack *S, Car *n);
  int DeleteQueue(LinkQueue *Q, Car *x);
  void CarIn(Stack *S, LinkQueue *Q, Car *r);
  void CostCalculate(Car *r, int h, int m);
  void CarOut(Stack *S, Stack *S0, Car *r, LinkQueue *Q);
```

2.5 详细设计

```c
#include <stdio.h>
#include <stdlib.h>
#define NUM 20
typedef struct {
  int hour;
  int min;
} time;
typedef struct {
  int num;
  int position;
  time t;
  float money;
} Car;
typedef struct Node {
  Car data;
  struct Node *next;
} CQueueNode;
typedef struct {
  Car elem[NUM + 1];
  int top;
} Stack;
typedef struct {
  CQueueNode *front;
  CQueueNode *rear;
} LinkQueue;
void InitQueue(LinkQueue *Q);          // 初始化队列
int EnterQueue(LinkQueue *Q, Car *t); // 进队
void InitStack(Stack *S);              // 初始化栈
void Push(Stack *S, Car *r);           // 压栈
int IsEmpty(Stack *S);                 // 判断栈空
int IsFull(Stack *S);                  // 判断栈满
int GetTop(Stack *S, Car *n);
int DeleteQueue(LinkQueue *Q, Car *x);
void CarIn(Stack *S, LinkQueue *Q, Car *r);
void CostCalculation(Car *r, int h, int m);
void CarOut(Stack *S, Stack *S0, Car *r, LinkQueue *Q);
int main(void) {
  int n, m, i = 1, j, flag = 0;
```

```
39    Car c[10];
40    Stack S, S0;
41    LinkQueue Q;    // 便道
42    InitStack(&S);  // 堆栈 S
43    InitStack(&S0); // 临时堆栈 S0
44    InitQueue(&Q);
45    while (1) {
46      printf("\t\t\t\t 欢迎停车");
47      printf("\n\t\t 请选择:\n");
48      printf("\n\t\t 1 : 进入停车场");
49      printf("\n\t\t 2 : 离开停车场");
50      printf("\n\t\t 3 : 退出系统\n");
51      printf("\n");
52      scanf("%d", &m);
53      switch (m) {
54      case 1:
55        printf("\n\t\t 请输入车牌号:");
56        scanf("%d", &c[i].num);
57        printf("\n\t\t 请输入到达/离开时间 (形如 2:00):");
58        scanf("%d:%d", &c[i].t.hour, &c[i].t.min);
59        CarIn(&S, &Q, &c[i]);
60        i++; // 车辆的情况
61        break;
62      case 2:
63        printf("\n\t\t 请输入车牌号:");
64        scanf("%d", &n);
65        for (j = 0; j < 10; j++)
66          if (n == c[j].num)
67            break;
68        printf("\n\t\t 请输入到达/离开时间 (形如 2:00):");
69        scanf("%d:%d", &c[j].t.hour, &c[j].t.min);
70        CarOut(&S, &S0, &c[j], &Q); // 车辆的情况
71        break;
72      case 3:
73        flag = 1;
74        break;
75      default:
76        printf("\n\t\t 请输入 1 或 2 或 3\n");
77      }
```

```
78      if (flag)
79        break; // 结束程序
80    }
81    return 0;
82  }
83  void InitQueue(LinkQueue *Q) {
84    Q->front = (CQueueNode *)malloc(sizeof(CQueueNode));
85    if (Q->front != NULL) {
86      Q->rear = Q->front;
87      Q->front->next = NULL;
88    }
89  }
90  int EnterQueue(LinkQueue *Q, Car *t) {
91    CQueueNode *NewNode;
92    NewNode = (CQueueNode *)malloc(sizeof(CQueueNode));
93    if (NewNode != NULL) {
94      NewNode->data.num = t->num;
95      NewNode->data.t.hour = t->t.hour;
96      NewNode->data.t.min = t->t.min;
97      NewNode->next = NULL;
98      Q->rear->next = NewNode;
99      Q->rear = NewNode;
100     return 1;
101   } else
102     return 0;
103 }
104 void InitStack(Stack *S) { S->top = 0; }
105
106 void Push(Stack *S, Car *r) {
107   S->top++;
108   S->elem[S->top].num = r->num;
109   r->position = S->elem[S->top].position = S->top;
110   S->elem[S->top].t.hour = r->t.hour;
111   S->elem[S->top].t.min = r->t.min;
112 }
113 int IsEmpty(Stack *S) // 判断车库是否为空
114 {
115   return (S->top == 0 ? 1 : 0);
116 }
```

```
117 int IsFull(Stack *S) // 判断车库是否为满
118 {
119   return (S->top == NUM ? 1 : 0);
120 }
121 int GetTop(Stack *S, Car *n) // 车离开车库
122 {
123   n->num = S->elem[S->top].num;
124   n->position = S->elem[S->top].position;
125   n->t.hour = S->elem[S->top].t.hour;
126   n->t.min = S->elem[S->top].t.min;
127   return 1;
128 }
129 int DeleteQueue(LinkQueue *Q, Car *x) {
130   CQueueNode *p;
131   if (Q->front == Q->rear)
132     return 0;          // 判断便道为空
133   p = Q->front->next; // 将便道中的车放入车库 Q->front->next = p->next;
134   if (Q->rear == p)
135     Q->rear = Q->front;
136   x->num = p->data.num;
137   x->t.hour = p->data.t.hour;
138   x->t.min = p->data.t.min;
139   free(p); // 释放临时指针
140   return 1;
141 }
142 void CarIn(Stack *S, LinkQueue *Q, Car *r) {
143   if (IsFull(S)) {
144     printf(" 停车场已满，请在便道中等待");
145     EnterQueue(Q, r); // 车进入便道
146   } else {
147     Push(S, r);
148     printf("\n\t\t 所在位置 %d", r->position); // 打印车的位置}
149   }
150 }
151 void CostCalculation(Car *r, int h, int m) {
152   if (m > r->t.min) {
153     r->t.min += 60;
154     r->t.hour -= 1;
155   }
```

```
156   h = r->t.hour - h;
157   m = r->t.min - m;
158   printf("\n\t\t 停车 %d 小时 %d 分钟\n", h, m);
159   printf(" 每小时收费 10 元\n");
160   h = h * 20;
161   m = h + m;
162   r->money = 0.5 * m;
163   printf(" 请支付金额%.2f 元\n", r->money); // 输出车主应付金额
164 }
165 void CarOut(Stack *S, Stack *S0, Car *r, LinkQueue *Q) {
166   int tag = S->top;
167   Car x;
168   if (IsEmpty(S))
169     printf(" 不存在该车辆");
170   else {
171     for (; r->num != S->elem[tag].num && tag > 0; tag--) {
172       Push(S0, &S->elem[tag]);
173       S->top--;
174     }
175     if (r->num == S->elem[tag].num) {
176       CostCalculation(r, S->elem[tag].t.hour, S->elem[tag].t.min);
177       S->top--;
178       for (; S0->top > 0; S0->top--)
179         Push(S, &S0->elem[S0->top]);
180       if (S->top < NUM &&
181           Q->front != Q->rear) // 判断车库是否有此车，有就找到此车，然后退出
182       {
183         DeleteQueue(Q, &x);
184         Push(S, &x);
185       }
186     } else if (tag == 0) // 过道中的车无需收车费
187     {
188       printf(" 没有进入停车场支付金额 0 元");
189       for (; S0->top > 0; S0->top--)
190         Push(S, &S0->elem[S0->top]);
191     }
192   }
193 }
```

2.6 调试分析

在开发过程中我曾遇到一些问题，如下：

1. 由于对栈和队列的理解不够深刻，导致在编写代码时出现了一些错误，如何正确的使用栈和队列，如何正确的初始化栈和队列，如何正确的判断栈和队列是否为空，如何正确的判断栈和队列是否为满，如何正确的入栈和出栈，如何正确的入队和出队等等。
2. 之前未考虑到车辆进入停车场时，停车场已满，车辆进入便道等待的情况，以及停车场没有车的情况导致的下标越界，后加上错误处理即可。

2.7 用户手册

1. 演示程序的运行环境为 Ubuntu 20.04，编译器为 gcc 9.4.0，编译选项为-O，执行指令为：gcc -O -o main main.cpp
2. 进入演示程序后即进入控制台：

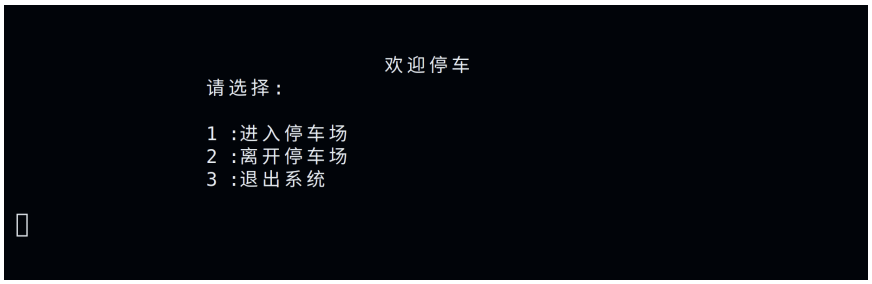

图 2.2: 停车场管理系统演示程序控制台

3. 输入 1 输入相应信息后，即可完成停车操作
4. 输入 2 离开即可计算出费用
5. 输入 3 即可退出系统

2.8 测试结果

1. 输入 1，车牌号 123456，以及到达时间 2:00
2. 输入 1，车牌号 654321，以及到达时间 3:00

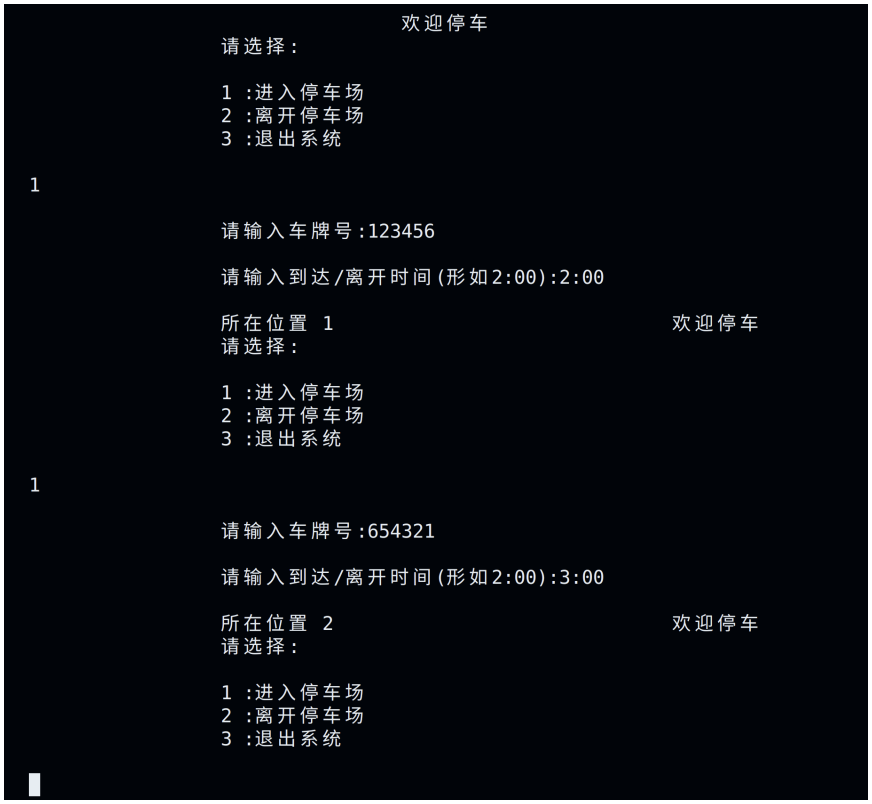

图 2.3: 输入测试

3. 输入 2, 输入车牌号 123465，输入离开时间 3:00
4. 输入 2, 输入车牌号 654321，输入离开时间 4:00

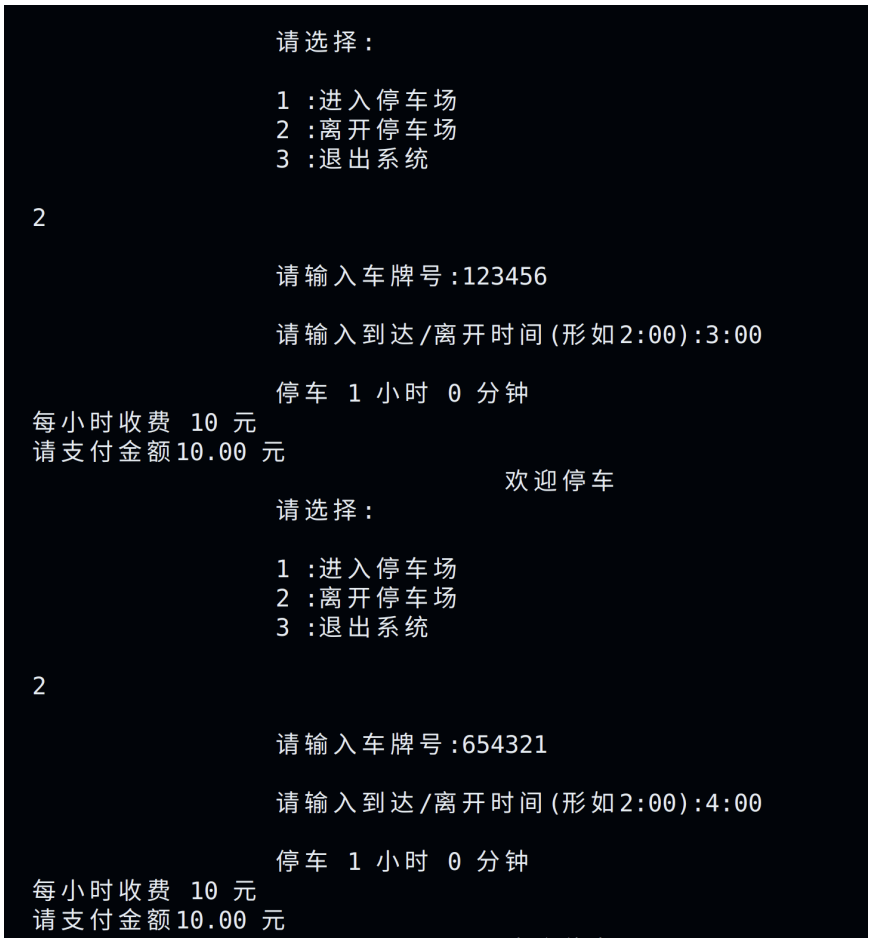

图 2.4: 输出测试

5. 测试便道停车功能

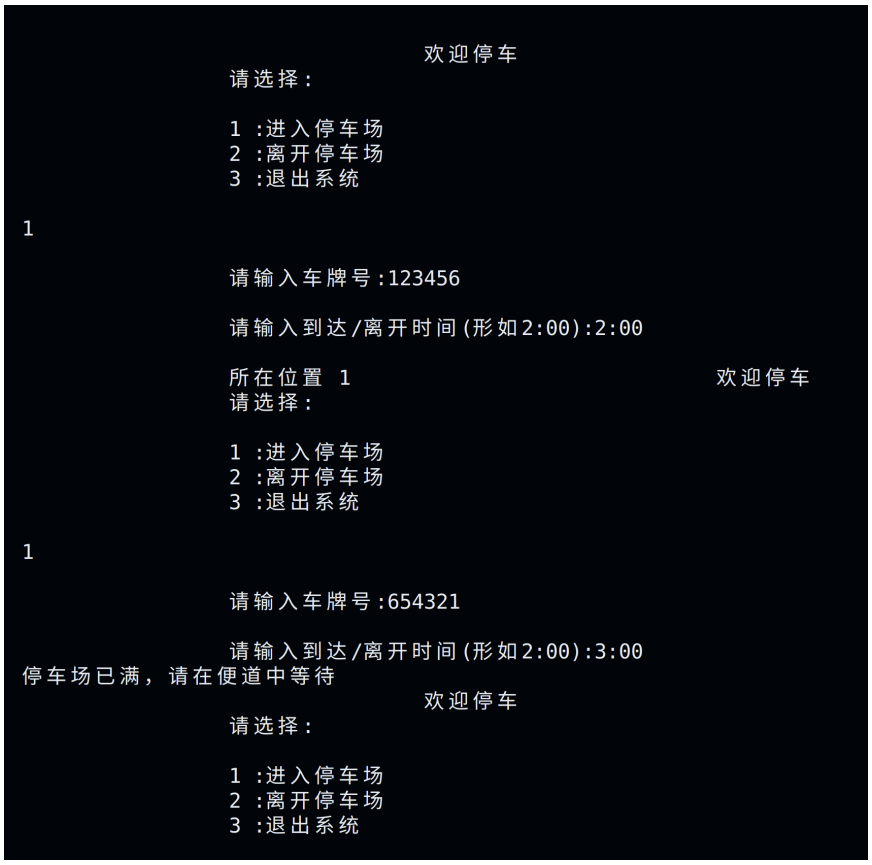

图 2.5: 测试便道停车功能

2.9 总结

本次实验中，我学习了栈和队列的相关知识，掌握了栈和队列的基本操作，以及如何使用栈和队列解决实际问题。在实验过程中，我遇到了一些问题，如何正确的使用栈和队列，如何正确的初始化栈和队列，如何正确的判断栈和队列是否为空，如何正确的判断栈和队列是否为满，如何正确的入栈和出栈，如何正确的入队和出队等等。通过本次实验，我对栈和队列有了更深刻的理解，对栈和队列的操作也更加熟练。

3 课题三：迷宫问题

3.1 任务描述

迷宫问题是取自心理学的一个古典实验。实验中，把一只老鼠从一个没有顶的大盒子的门放入，在盒中设置了许多墙，对行进的方向形成了多处阻挡。盒子仅仅有一个出口，在出口处放置了一块奶酪，吸引老鼠在迷宫中寻找道路以到达出口。重复对老鼠进行上述实验，看老鼠能在多久找到出口。请设计一个算法实现迷宫问题求解。

3.2 功能要求

找到迷宫中的一条通路，从入口到出口，或者确定没有通路。

3.3 需求分析

在对迷宫问题进行建模时，传统的深度优先搜索（DFS）和广度优先搜索（BFS）常常被用作求解策略。然而，这两种方法虽然可以解决问题，但它们并不完全符合现实世界中老鼠寻找奶酪的行为特征。DFS 和 BFS 都属于盲目搜索，需要提前知道整个迷宫的信息，但现实中的老鼠并不可能掌握这样的全局信息。老鼠并不知道迷宫的大小或边界，它所知道的只是在哪个方向上能闻到奶酪的气味，以及前面是否有墙。因此，我考虑使用 Q-learning 算法最大程度上贴合题意同时解决问题。

Q-learning 是一种模型无关的强化学习算法，这意味着该算法并不需要对环境有先验知识，例如它不需要知道地图的边界，这与老鼠的实际行为相匹配。老鼠在寻找奶酪的过程中，并不会事先知道整个迷宫的布局，也就是说，它不知道地图的边界在哪里，或者不知道奶酪在迷宫的哪个位置。

老鼠通过尝试不同的行动和感知环境的反馈，来学习如何在迷宫中找到奶酪。这与 Q-learning 算法中的探索（Exploration）和利用（Exploitation）的概念非常相似。老鼠会在探索和利用之间做出平衡，尝试新的行动来探索迷宫，同时利用已知的信息来找到奶酪。

至于老鼠能够感知到奶酪的气味，我们可以将其理解为环境给予的奖励（Reward）。在 Q-learning 算法中，当老鼠在迷宫中移动并最终找到奶酪时，它会得到一定的奖励，这就相当于感知到奶酪的气味。这样，老鼠会学习到一种策略（Policy），指导它如何从起点找到奶酪。

3.4 概要设计

程序基本结构设计如下：

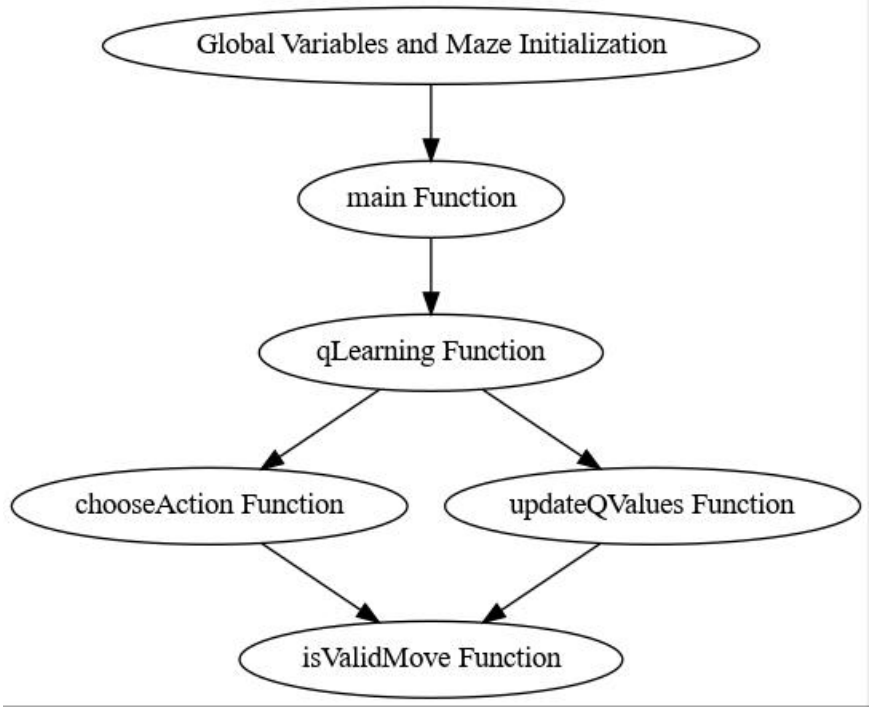

图 3.1: 程序结构

各部分叙述如下：

1. **全局变量定义**：在代码的开头定义了一些全局变量，如学习率、折扣因子、迭代次数、epsilon 以及每次迭代的最大步数等。这些全局变量在整个程序中被多次使用。

2. **迷宫和动作定义**：定义了迷宫的大小、起点和终点，以及可执行的动作。动作被定义为一个二维数组，表示各个方向的移动。

3. **函数定义**

   - **isValidMove**：检查给定的行和列是否是迷宫中的有效位置。
   - **chooseAction**：基于当前的状态和 Q 值，选择一个动作。这个动作可以是随机的（探索），也可以是当前 Q 值最高的（利用）。
   - **updateQValues**: 更新 Q 值表。新的 Q 值是基于旧的 Q 值、当前的奖励和最大可能的未来奖励计算的。
   - **qLearning**: 实现 Q-learning 算法的主体部分。首先，初始化 Q 值表，然后进行多次迭代。在每次迭代中，从起点开始，选择并执行动作，然后更新 Q 值，直到到达终点或达到最大步数。迭代结束后，选择 Q 值最高的动作，找到从起点到终点的路径。

4. **主函数**: 主函数首先从文件读入迷宫的数据，然后调用 **qLearning** 函数进行学习和找路径。最后，返回 0 表示程序正常结束。

3.5 详细设计

Q-learning 代码

```cpp
#include <bits/stdc++.h> // 引入常用库

// 定义强化学习的一些参数
const double learningRate = 0.9;     // 学习率
const double discountFactor = 0.9;   // 折扣因子
const int numEpisodes = 5000;        // 总的训练轮数
double epsilon = 0.9;                // -贪婪策略中的参数
const double epsilonDecay = 0.99;    //  的衰减系数
const int maxStepsPerEpisode = 3000; // 每轮训练的最大步数

// 定义迷宫的相关参数
int numRows = 6;                     // 迷宫的行数
int numCols = 5;                     // 迷宫的列数
std::vector<std::vector<int>> maze; // 二维向量表示的迷宫，0 表示通道，1 表示墙

// 定义起点和终点
int startRow = 0;         // 起点的行坐标
int startCol = 0;         // 起点的列坐标
int endRow = numRows - 1; // 终点的行坐标
int endCol = numCols - 1; // 终点的列坐标

// 定义 8 种可能的行动：上、下、左、右、左上、右上、左下、右下
int actions[8][2] = {{-1, 0},  {1, 0},  {0, -1}, {0, 1},
                     {-1, -1}, {-1, 1}, {1, -1}, {1, 1}};
int action_cnt = 8; // 行动的种类数

// 判断是否是合法的移动
bool isValidMove(int row, int col) {
  if (row >= 0 && row < numRows && col >= 0 && col < numCols &&
      maze[row][col] == 0) {
    return true;
  }
  return false;
}

// 根据当前状态和 Q 值表选择行动
int chooseAction(int row, int col,
```

```
38                  std::vector<std::vector<std::vector<double>>> &qValues,
39                  int previousAction) {
40      // epsilon-greedy 策略选择动作
41
42      double random =
43          static_cast<double>(rand()) / RAND_MAX; // 生成一个 [0, 1) 之间的随机数
44      if (random < epsilon) {                   // 以  的概率随机选择动作
45        std::vector<int> validActions;          // 存储所有有效的动作
46        for (int i = 0; i < action_cnt; i++) { // 遍历所有可能的动作
47          int newRow = row + actions[i][0];     // 计算执行动作后的新位置
48          int newCol = col + actions[i][1];
49          if (isValidMove(newRow, newCol)) { // 如果该动作是有效的
50            validActions.push_back(i); // 将该动作加入到有效动作列表中
51          }
52        }
53
54        if (validActions.empty()) { // 如果没有有效的动作
55          return -1;                // 返回-1 表示无有效动作
56        }
57
58        return validActions[rand() %
59                            validActions.size()]; // 在所有有效动作中随机选择一个
60      }
61
62      // 以 1- 的概率选择最优的动作
63      int bestAction = -1;                      // 存储最优动作
64      double maxValue = -1e9;                   // 存储最大的 Q 值
65      for (int i = 0; i < action_cnt; i++) { // 遍历所有可能的动作
66        int newRow = row + actions[i][0];     // 计算执行动作后的新位置
67        int newCol = col + actions[i][1];
68        // 如果该动作是有效的并且 Q 值大于当前最大的 Q 值，那么更新最优动作和最大 Q 值
69        if (isValidMove(newRow, newCol) && qValues[row][col][i] > maxValue &&
70            (previousAction == -1 || // 对于第一步，任何动作都可以是最优动作
71             (actions[i][0] !=
72                  -actions[previousAction][0] || // 不选取与上一步相反的动作
73              actions[i][1] != -actions[previousAction][1]))) {
74          maxValue = qValues[row][col][i];
75          bestAction = i;
76        }
```

```
77        }
78        return bestAction; // 返回最优动作
79      }
80
81      // 根据公式更新 Q 值表
82      void updateQValues(int currentRow, int currentCol, int newRow, int newCol,
83                         int action,
84                         std::vector<std::vector<std::vector<double>>> &qValues) {
85        double maxQValue = -1e9;                // 存储最大的 Q 值
86        for (int i = 0; i < action_cnt; i++) { // 遍历所有可能的动作
87          // 如果 Q 值大于当前最大的 Q 值，那么更新最大 Q 值
88          if (qValues[newRow][newCol][i] > maxQValue) {
89            maxQValue = qValues[newRow][newCol][i];
90          }
91        }
92
93        // 定义奖励值，如果到达目标位置则奖励为 100，否则按离终点的距离给予惩罚
94        double distance=sqrt(pow(newRow - endRow, 2) + pow(newCol - endCol, 2));
95        double reward = (distance == 0) ? 100 : -distance;
96        // 根据 Q 学习算法的公式更新 Q 值
97        qValues[currentRow][currentCol][action] +=
98            learningRate * (reward + discountFactor * maxQValue -
99                            qValues[currentRow][currentCol][action]);
100     }
101
102     // Q 学习的主要函数
103     void qLearning() {
104       srand(time(0)); // 设置随机种子
105       // 初始化 Q 值表，所有的 Q 值都设置为 0
106       std::vector<std::vector<std::vector<double>>> qValues(
107           numRows, std::vector<std::vector<double>>(
108                        numCols, std::vector<double>(action_cnt, 0)));
109
110       // 进行 numEpisodes 轮训练
111       for (int episode = 0; episode < numEpisodes; episode++) {
112         int currentRow = startRow; // 当前位置的行坐标
113         int currentCol = startCol; // 当前位置的列坐标
114
115         int step = 0;              // 计步器
```

```
116         int previousAction = -1; // 上一步的动作
117         // 当未达到目标且未超过每轮最大步数时，继续训练
118         while ((currentRow != endRow || currentCol != endCol) &&
119                step < maxStepsPerEpisode) {
120           // 选择一个动作
121           int action = chooseAction(currentRow, currentCol, qValues, -1);
122
123           if (action == -1) { // 如果没有有效的动作
124             break;            // 结束本轮训练
125           }
126
127           // 执行选定的动作，更新当前位置
128           int newRow = currentRow + actions[action][0];
129           int newCol = currentCol + actions[action][1];
130
131           if (isValidMove(newRow, newCol)) { // 如果移动后的位置有效
132             // 更新 Q 值表
133             updateQValues(currentRow, currentCol, newRow, newCol, action, qValues);
134             currentRow = newRow; // 更新当前位置
135             currentCol = newCol;
136             previousAction = action; // 更新上一步的动作
137           }
138           step++; // 计步器增加
139         }
140         //   按衰减系数衰减
141         epsilon *= epsilonDecay;
142       }
143
144       double originalEpsilon = epsilon; // 存储原始的   值
145       epsilon = 0; //   设为 0，即在路径规划时只采用贪婪策略
146
147       // 显示找到的路径
148       std::vector<std::pair<int, int>> path; // 存储路径
149       int currentRow = startRow;             // 当前位置的行坐标
150       int currentCol = startCol;             // 当前位置的列坐标
151       path.push_back(std::make_pair(currentRow, currentCol)); // 将起点加入到路径中
152       int previousAction1 = -1;                                // 上一步的动作
153
154       while (currentRow != endRow ||
```

```
155             currentCol != endCol) { // 当未到达终点时，继续行走
156         // 选择最优动作
157         int bestAction =
158             chooseAction(currentRow, currentCol, qValues, previousAction1);
159 
160         if (bestAction == -1) { // 如果没有有效的动作
161           std::cout << "There is no valid path!" << std::endl; // 输出提示信息
162           return;                                              // 结束函数执行
163         }
164 
165         // 执行选定的动作，更新当前位置
166         int newRow = currentRow + actions[bestAction][0];
167         int newCol = currentCol + actions[bestAction][1];
168 
169         if (isValidMove(newRow, newCol)) { // 如果移动后的位置有效
170           currentRow = newRow;             // 更新当前位置
171           currentCol = newCol;
172           path.push_back(
173               std::make_pair(currentRow, currentCol)); // 将新位置加入到路径中
174           previousAction1 = bestAction;                // 更新上一步的动作
175         }
176       }
177 
178       // 输出找到的路径
179       for (const auto &p : path) {
180         std::cout << "(" << p.first << ", " << p.second << ") ";
181       }
182       std::cout << std::endl;
183 
184       // 恢复原始的   值
185       epsilon = originalEpsilon;
186     }
187 
188     // 主函数
189     int main() {
190       // 从文件读取迷宫数据
191       freopen("/media/zby/SSD 数据盘/Program-Practice/Maze/build/maze.txt", "r",
192               stdin);
193       std::cin >> numRows >> numCols; // 输入迷宫的行数和列数
```

```
194       maze.resize(numRows, std::vector<int>(numCols)); // 调整迷宫大小
195       for (int i = 0; i < numRows; i++) {              // 输入迷宫数据
196         for (int j = 0; j < numCols; j++) {
197           std::cin >> maze[i][j];
198         }
199       }
200
201       qLearning(); // 执行 Q 学习算法
202
203       return 0; // 程序结束
204     }
```

附上一开始写的 DFS 版本

```
1     #include <iostream>
2     #include <vector>
3     #include <stack>
4
5     // 定义迷宫的相关参数
6     int numRows = 6;                    // 迷宫的行数
7     int numCols = 5;                    // 迷宫的列数
8     std::vector<std::vector<int>> maze; // 二维向量表示的迷宫，0 表示通道，1 表示墙
9
10    // 定义 8 种可能的行动：上、下、左、右、左上、右上、左下、右下
11    int actions[8][2] = {{-1, 0},  {1, 0},  {0, -1}, {0, 1},
12                         {-1, -1}, {-1, 1}, {1, -1}, {1, 1}};
13    int action_cnt = 8; // 行动的种类数
14
15    // 判断是否是合法的移动
16    bool isValidMove(int row, int col) {
17      if (row >= 0 && row < numRows && col >= 0 && col < numCols &&
18          maze[row][col] == 0) {
19        return true;
20      }
21      return false;
22    }
23
24    // 使用深度优先搜索解决迷宫问题
25    bool dfs(int currentRow, int currentCol, std::vector<std::vector<bool>>& visited,
26             std::vector<std::pair<int, int>>& path) {
27      // 到达终点，返回 true
```

```
28        if (currentRow == numRows - 1 && currentCol == numCols - 1) {
29          return true;
30        }
31
32        visited[currentRow][currentCol] = true; // 将当前位置标记为已访问
33
34        // 遍历所有可能的动作
35        for (int i = 0; i < action_cnt; i++) {
36          int newRow = currentRow + actions[i][0]; // 计算执行动作后的新位置
37          int newCol = currentCol + actions[i][1];
38
39          // 如果新位置是合法的且未被访问过
40          if (isValidMove(newRow, newCol) && !visited[newRow][newCol]) {
41            // 将新位置加入到路径中
42            path.push_back(std::make_pair(newRow, newCol));
43
44            // 递归调用 DFS 搜索
45            if (dfs(newRow, newCol, visited, path)) {
46              return true; // 如果找到一条路径，返回 true
47            }
48
49            // 未找到路径，回溯
50            path.pop_back();
51          }
52        }
53
54        return false; // 未找到路径，返回 false
55      }
56
57      // 解决迷宫问题的主函数
58      void solveMaze() {
59        // 创建一个二维布尔数组来记录访问状态
60        std::vector<std::vector<bool>> visited(numRows, std::vector<bool>(numCols, false));
61
62        std::vector<std::pair<int, int>> path; // 存储路径
63        path.push_back(std::make_pair(0, 0));  // 将起点加入到路径中
64
65        if (dfs(0, 0, visited, path)) {
66          // 输出找到的路径
```

```
67          for (const auto& p : path) {
68            std::cout << "(" << p.first << ", " << p.second << ") ";
69          }
70          std::cout << std::endl;
71        } else {
72          std::cout << "There is no valid path!" << std::endl;
73        }
74      }
75
76      int main() {
77        // 从文件读取迷宫数据
78        freopen("/media/zby/SSD 数据盘/Program-Practice/Maze/build/maze.txt", "r", stdin);
79        std::cin >> numRows >> numCols; // 输入迷宫的行数和列数
80        maze.resize(numRows, std::vector<int>(numCols)); // 调整迷宫大小
81        for (int i = 0; i < numRows; i++) {              // 输入迷宫数据
82          for (int j = 0; j < numCols; j++) {
83            std::cin >> maze[i][j];
84          }
85        }
86
87        solveMaze(); // 解决迷宫问题
88
89        return 0; // 程序结束
90      }
```

3.6 调试分析

1. 一开始写了 dfs 版本，但是因为太简单了所以改用 qlearning 了
2. 开始的时候因为以为题意是四方向移动，发现当无解的时候程序会死循环，因此加入了最大步数的限制，因此后来改成了八方向移动，这样就不会出现死循环的情况了。
3. 一开始考虑奖励函数设为到终点为 100 其余为-1，后考虑老鼠离终点越近奖励越大，因此改成了距离终点的距离，这样老鼠就会尽可能的走向终点。
4. 一开始地图直接在程序中定义，但是调整迷宫不方便，因此改成了从文件中读取迷宫数据。

3.7 用户手册

演示程序的运行环境为 Ubuntu 20.04，编译器为 g++ 9.4.0，编译选项为-O，Q-learning 的执行指令为

```
cd "/media/zby/SSD 数据盘/Program-Practice/Maze/"
&& g++ main.cpp -o main
&& "/media/zby/SSD 数据盘/Program-Practice/Maze/"main"
```

DFS 的执行指令为

```
cd "/media/zby/SSD 数据盘/Program-Practice/Maze/"
&& g++ dfs.cpp -o dfs
&& "/media/zby/SSD 数据盘/Program-Practice/Maze/"dfs"
```

⋆ 迷宫文件的格式为：第一行为迷宫的行数和列数，接下来的行数为迷宫的数据，*0* 表示通道，*1* 表示墙，如下所示。

```
6 5
0 1 0 0 0
1 0 1 1 0
1 0 1 1 0
1 1 0 1 1
0 0 1 0 1
1 1 1 0 0
```

图 3.2: 迷宫样例数据

3.8 测试结果

```
[Running] cd "/media/zby/SSD数据盘/Program-Practice/Maze/" && g++ main.cpp -o main && "/media/zby/SSD数据盘/
Program-Practice/Maze/"main
(0, 0) (1, 1) (2, 1) (3, 2) (4, 3) (5, 4)

[Done] exited with code=0 in 0.976 seconds
```

图 3.3: 迷宫有解情况 (Q-learning)

```
[Running] cd "/media/zby/SSD数据盘/Program-Practice/Maze/" && g++ dfs.cpp -o dfs && "/media/zby/SSD数据盘/
Program-Practice/Maze/"dfs
(0, 0) (1, 1) (2, 1) (3, 2) (4, 3) (5, 3) (5, 4)

[Done] exited with code=0 in 0.456 seconds
```

图 3.4: 迷宫有解情况 (DFS)

```
[Running] cd "/media/zby/SSD数据盘/Program-Practice/Maze/" && g++ main.cpp -o main && "/media/zby/SSD数据盘/
Program-Practice/Maze/"main
There is no valid path!

[Done] exited with code=0 in 0.977 seconds
```

图 3.5: 迷宫无解情况

3.9 总结与分析

Q 学习算法和深度优先搜索算法在解决迷宫问题上有着不同的特点。Q 学习算法是一种基于强化学习的方法，通过训练和学习来得到最优策略。它能够找到最优路径，但需要预先训练并更新 Q 值表。深度优先搜索算法是一种盲目搜索方法，通过穷举所有可能的路径来找到一条路径。它能够找到路径，但不保证是最优路径，且没有训练过程。

选择哪种算法取决于具体的需求和问题情况。如果希望找到最优路径，并且有足够的训练数据和时间进行学习，那么 Q 学习算法是一个好的选择。如果只需要找到任意可行的路径，并且不需要训练过程，那么深度优先搜索算法是一个简单且有效的方法。在实际应用中，根据问题的特点和要求，可以选择适合的算法来解决迷宫问题，或者结合两种算法的优点进行改进和优化。