城市交通路径规划系统

陈纳新
23307140093

一、整体实现思路

明确目标优先级：解的**存在性**高于解的好坏。即在**所有情况**下都能为每辆车找到较短路径并保证每条道路每一时刻不超过最大车流量限制**优于**在大部分情况下为所有车辆找到的路径的时间总和更短，但少部分情况下由于**算法问题**即便路径存在也无法找到路径。

初始想法：不考虑流量限制，为每辆车规划最短路径；如果每辆车都走最短路径，找出超出车流量的路径，记录经过此路段的车，为各辆车重新规划不经过此路段的路径并挑出使得总时间增加最少的车辆们让其改道。但是，此想法存在**问题**：如果拥挤路段是各车通往终点的唯一路径，则各车找不到删除此道路后的路径。但各车实际上可以选择先绕道等待拥挤路段空出后继续驶入该道路，因此部分情况下，此算法得不出路径，故**否决**。

最终思路：每一时刻，考虑这一时刻仍需行驶在当前道路上即当前道路未行驶完的车辆，更新实际车流量，将达到最大车流量限制的道路排除出当前时刻的路径规划，利用 Dijkstra 算法重新依次为位于地点需要选择下一道路的车辆规划从当前地点到目的地的最短路径，如果车辆找到最短路径，就让该车辆驶入当前时刻规划出的最短路径中的第一段道路，如果车辆找不到最短路径，说明车辆到目的地必须经过的某段道路此时已达到最大车流量限制，则让车辆驶入邻近的最短的未达到车流量限制的道路以等待必经道路空闲时刻找路径，更新实际车流量，判断驶入道路是否达到最大车流量，达到则排除出当前时刻后续车辆的路径规划中，为下一辆位于地点需要选择下一道路的车辆规划路径直至所有位于地点需要选择下一道路的车辆在当前时刻的路径都规划完成。考虑下一时刻，直至所有车辆都已到达目的地。

二、已实现功能的思路

见代码注释

(一）数据结构

车辆：

```
struct car {
    int start;//起点
    int end;//目的地
    bool state;//是否到达目的地
    int present_position[2];//当前所在道路的端点
    int stay_time;//在当前所在道路上还需停留的时间
    vector<int> path;//行驶过的路径：用地点记录
};
```

交通网络：

```
vector<vector<int>> map_(location_num, vector<int>(location_num, 0));
```

```
for (int i = 0; i < location_num; i++) {//地图
    for (int j = 0; j < location_num; j++) {
        cin >> map_[i][j];
    }
}
```

交通流量限制：

```
vector<vector<int>> limit(location_num, vector<int>(location_num, 0))
```

```
for (int i = 0; i < location_num; i++) {//最大车流量限制
    for (int j = 0; j < location_num; j++) {
        cin >> limit[i][j];
    }
}
```

（二）函数

某时刻路径规划：

```
void improveddijkstra(vector<vector<int>> map_, vector<vector<int>> limit, vector<car>& cars) {//为车辆规划路径
    int n = map_.size();
    vector<vector<int>> reality(n, vector<int>(n, 0));//初始化实际车流量
    for (auto& car : cars) {
        if (car.stay_time != 0) {//将上一单位时间从车辆还需在当前道路上停留的时间中减去
            car.stay_time--;
        }
        if (car.stay_time != 0) {//若车辆还需在当前道路上行驶，则记录入实际车流量中
            reality[car.present_position[0]][car.present_position[1]]++;
            reality[car.present_position[1]][car.present_position[0]]++;
            if (reality[car.present_position[0]][car.present_position[1]] == limit[car.present_position[0]][car.present_position[1]])
            //若实际车流量达到最大车流量限制，则在当前时间删去达到最大车流量限制的路径
            {
                map_[car.present_position[0]][car.present_position[1]] = 0;
                map_[car.present_position[1]][car.present_position[0]] = 0;
            }
        }
    }
```

```
    for (auto& car : cars) {
        if (car.state == 1) continue;//如果车已到达目的地，不考虑
        if (car.stay_time != 0) {//跳过继续停留在当前道路的车，为已到达当前道路终点的车重新规划路径
            continue;
        }
        if (car.present_position[1] == car.end - 1) {//如果车到达的道路终点是目的地，则将车的状态更新为已到达，考虑其他车
            car.state = 1;
            continue;
        }
        int start = car.start - 1;//将起点序号映射至数组下标
        int end = car.end - 1;//将目的地序号映射至数组下标
```

```
vector<bool> s;//应用dijkstra算法找起点到目的地的最短路径
vector<int> dist;
vector<int> min_path;
for (int i = 0;i < n;i++) {
    if (map_[start][i] != 0) {
        dist.push_back(map_[start][i]);
    }
    else dist.push_back(INT_MAX);
    s.push_back(0);
    if (i != start && dist[i] < INT_MAX) {
        min_path.push_back(start);
    }
    else min_path.push_back(-1);
}
s[start] = 1;
dist[start] = 0;
int min;
int w;
```

```
for (int i = 0;i < n - 1;i++) {
    min = INT_MAX;
    int u = start;
    for (int j = 0;j < n;j++) {
        if (s[j] == 0 && dist[j] < min) {
            u = j;
            min = dist[j];
        }
    }
    s[u] = 1;
    for (int k = 0;k < n;k++) {
        if (map_[u][k] != 0) {
            w = map_[u][k];
        }
        else w = INT_MAX;
        if (s[k] == 0 && w < INT_MAX && dist[u] + w < dist[k]) {
            dist[k] = dist[u] + w;
            min_path[k] = u;
        }
    }
}
```

```
if (min_path[end] != -1) {//如果找到起点到目的地的最短路径
    for (int j = end;j != start;j = min_path[j]) car.present_position[1] = j;
    car.path.push_back(car.present_position[1]);//将起点到目的地的路径中的第一段道路记录进行驶过的路径中
    car.present_position[0] = start;//将当前所在道路的起始端点更新为车辆起点
    car.start = car.present_position[1] + 1;//将车辆起点即规划下一道路的初始点更新为当前所在道路的终点
    car.stay_time = map_[car.present_position[0]][car.present_position[1]];//将车辆需在当前道路停留的时间更新为道路长度
    reality[car.present_position[0]][car.present_position[1]]++;//更新实际车流量
    reality[car.present_position[1]][car.present_position[0]]++;
    if (reality[car.present_position[0]][car.present_position[1]] == limit[car.present_position[0]][car.present_position[1]])
        //若实际车流量达到最大车流量限制，则在当前时间删去达到最大车流量限制的路径
    {
        map_[car.present_position[0]][car.present_position[1]] = 0;
        map_[car.present_position[1]][car.present_position[0]] = 0;
    }
}
```

```
        else {//如果起点到目的地的最短路径未找到，说明路径必经道路此时达到最大车流量，
            //那么暂时让车辆驶入未达到最大车流量的最短道路等待必经道路空缺后重新规划路径
            //如找到最短路径的情况更新车辆信息及实际车流量并判断驶入道路是否达到最大车流量限制，达到则在当前时刻删去
            int j = -1;
            for (int k = 0;k < n;k++) {
                if (dist[k] < INT_MAX && j == -1 && k != start) j = k;
                if (dist[k] < dist[j] && k != start) j = k;
            }
            car.present_position[1] = j;
            car.path.push_back(car.present_position[1]);
            car.present_position[0] = start;
            car.start = car.present_position[1] + 1;
            car.stay_time = map_[car.present_position[0]][car.present_position[1]];
            reality[car.present_position[0]][car.present_position[1]]++;
            reality[car.present_position[1]][car.present_position[0]]++;
            if (reality[car.present_position[0]][car.present_position[1]] == limit[car.present_position[0]][car.present_position[1]])
            {
                map_[car.present_position[0]][car.present_position[1]] = 0;
                map_[car.present_position[1]][car.present_position[0]] = 0;
            }
        }
    }
}
```

输出：

```
void show(vector<car> x, vector<vector<int>> map_) {//输出所有车辆的路径及总时间
    int time = 0;//初始化
    int pre, p;//记录车辆所在道路的两个端点
    for (auto& i : x) {//遍历车辆
        p = -1;//初始化地点
        for (auto& j : i.path) {//遍历路径中的地点
            cout << j + 1 << " ";//输出车辆路径地点
            pre = p;
            p = j;//更新车辆所在道路端点
            if (pre != -1) time += map_[pre][p];//将道路长度即所需行驶时间加入总时间
        }
        cout << endl;
    }
    cout << time;//输出总时间
}
```

主函数：

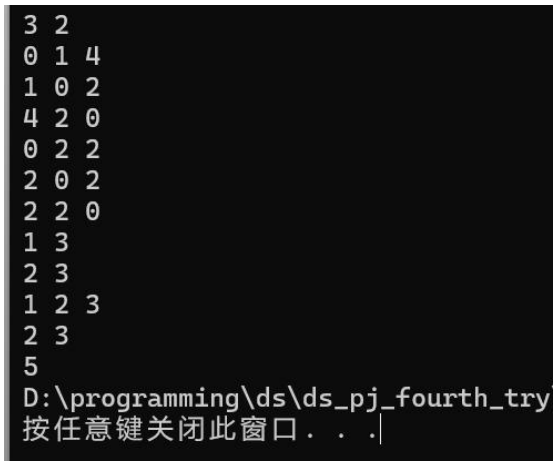
```
int main() {
    int location_num, car_num;//初始化地图，车流量限制
    cin >> location_num >> car_num;
    vector<vector<int>> map_(location_num, vector<int>(location_num, 0));
    vector<vector<int>> limit(location_num, vector<int>(location_num, 0));

    for (int i = 0; i < location_num; i++) {//地图
        for (int j = 0; j < location_num; j++) {
            cin >> map_[i][j];
        }
    }

    for (int i = 0; i < location_num; i++) {//最大车流量限制
        for (int j = 0; j < location_num; j++) {
            cin >> limit[i][j];
        }
    }
```

```
    vector<car> cars;
    for (int i = 0; i < car_num; i++) {
        car car1;
        cars.push_back(car1);
    }
    for (int i = 0; i < car_num; i++) {//初始化各车起点，目的地
        cin >> cars[i].start >> cars[i].end;
    }

    for (int i = 0; i < car_num; i++) {//初始化各车其他参量
        cars[i].state = 0; cars[i].stay_time = 0;
        cars[i].path.push_back(cars[i].start - 1);
        cars[i].present_position[0] = cars[i].start;
        cars[i].present_position[1] = -1;
    }

    while (1) {//循环直至所有车辆到达目的地，1次循环代表1单位时间
        improveddijkstra(map_, limit, cars);//每时刻重新为各车规划路径
        int i;
        for (i = 0;i < car_num;i++) {
            if (cars[i].state == 0) break;//如果有车未到达目的地，退出
        }
        if (i == car_num) break;//判断是否所有车辆到达目的地
    }
    show(cars, map_);//输出
    return 0;
}
```

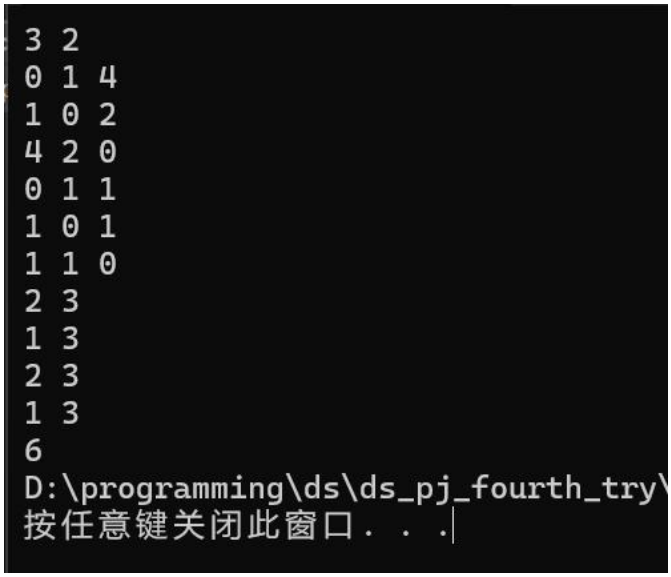
三、测试用例及结果

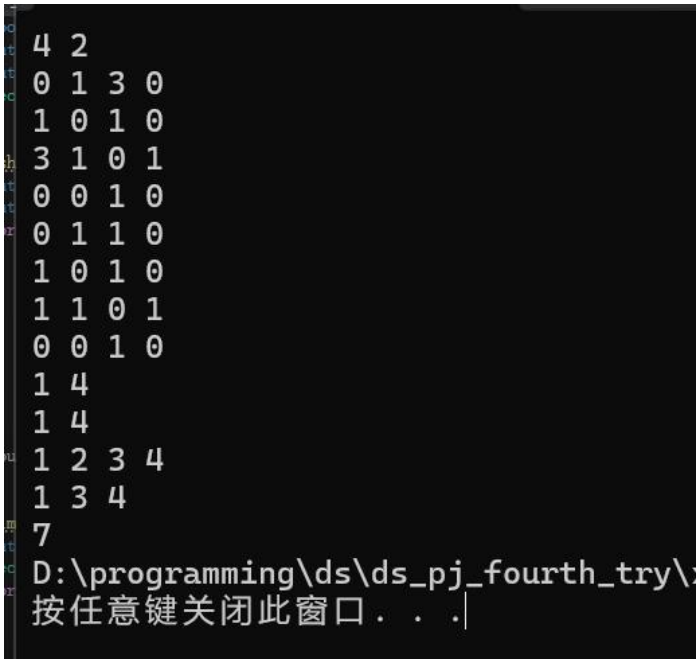
交通流量限制足够大：

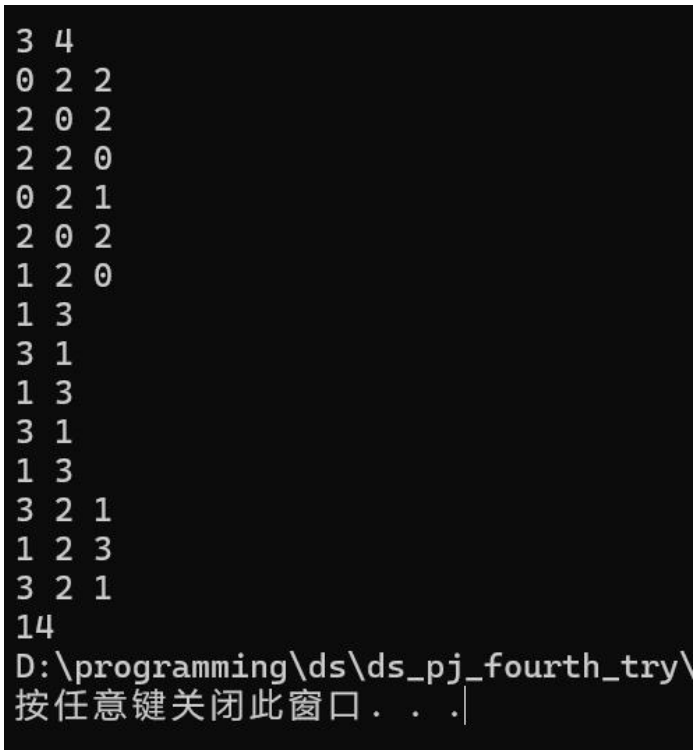
```
5 3
0 0 2 0 0
0 0 3 0 0
2 3 0 1 0
0 0 1 0 4
0 0 0 4 0
0 0 6 0 0
0 0 3 0 0
6 3 0 5 0
0 0 5 0 4
0 0 0 4 0
1 4
2 5
3 1
1 3 4
2 3 4 5
3 1
13
D:\programming\ds\ds_pj_fourth_try\
按任意键关闭此窗口. . .
```


3 2
0 1 4
1 0 2
4 2 0
0 2 2
2 0 2
2 2 0
1 3
2 3
1 2 3
2 3
5
D:\programming\ds\ds_pj_fourth_try
按任意键关闭此窗口. . .

需要考虑交通流量限制：

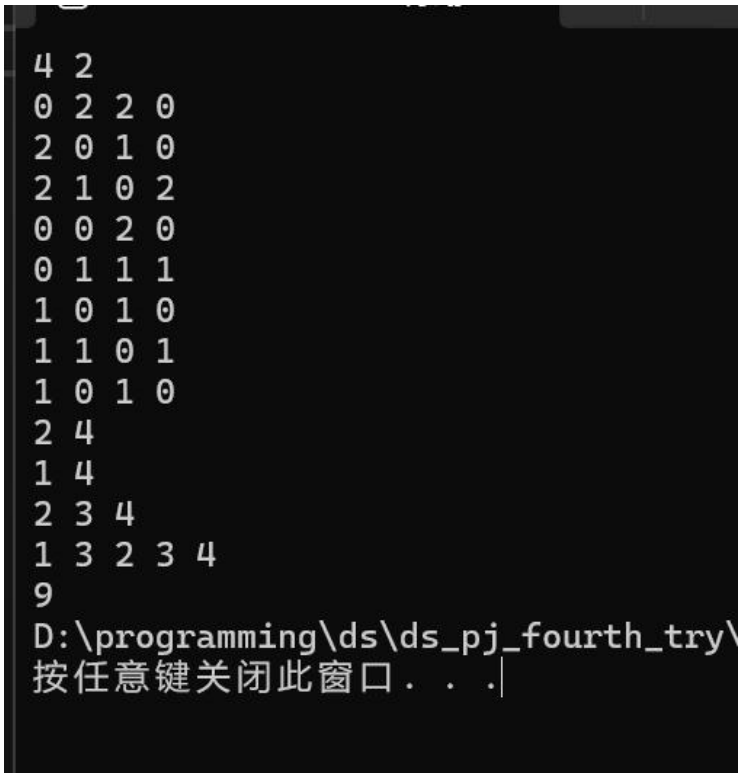

3 2
0 1 4
1 0 2
4 2 0
0 1 1
1 0 1
1 1 0
2 3
1 3
2 3
1 3
6
D:\programming\ds\ds_pj_fourth_try\
按任意键关闭此窗口. . .


4 2
0 1 3 0
1 0 1 0
3 1 0 1
0 0 1 0
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
1 4
1 4
1 2 3 4
1 3 4
7
D:\programming\ds\ds_pj_fourth_try\
按任意键关闭此窗口. . .



*此算法仅能保证解的存在性，不保证解是最优的

例：找到，但并非最优



四、注明

与王佳琦 23307140095 讨论