数据结构

【城市交通路径规划系统】

项目报告

姓名：马颢宸

学号：23307110426

一、项目思路

1、项目要求整理

（1）规划给定的多个车辆从起点到终点的时间之和最短。

（2）考虑道路的最大车流量限制。

（3）车辆同时出发，且车辆前行途中不可停车、车辆行驶在某条路上时不可改变方向。

2、项目实现思路

根据项目要求，首先考虑使用 **Dijkstra 算法**求出车辆的最短路径。其次在 Dijkstra 算法的基础上，通过**遍历**每条道路的实时流量信息查找拥堵路段，并对其上行驶车辆**更换路径**，再次使用 **Dijkstra 算法寻找次优路径**，直至所有路段均满足车流量限制为止。

对于基础功能的实现，由于此时不考虑车流量的限制，故只要实现每辆车用时最短即可实现所有车辆的时间之和最短。由于车辆单位时间行驶道路长度为 1，故车辆行驶的最短路径数学上等于最短用时。使用 Dijkstra 算法实现求解。

对于高级功能的实现，首先使用 Dijkstra 算法在不考虑车流量限制的情况下算出所有车辆的最短路径，同时根据 Dijkstra 算法所得 path、dist 数组计算该车经过每条路的时间，并将该车记录在对应路段的对应时间上。此时所得方案为不考虑车流量时的最优解，并得到当前所有道路的实时流量信息。

遍历每条道路的每个时间点道路信息，并将其实时流量与最大流量进行比较。记录该方案下的拥堵路段。对每条拥堵路段，取出当前时间点在该路段上的所有车辆，并对其进行二次 Dijkstra 规划。此次规划，将原有拥堵路段从 Dijkstra 算法的考虑地图中删去，从而实现寻找将车辆从该路段上移去的方案。对每辆汽车而言，移去该路段后 Dijkstra 算法的方案为汽车能到达目的地的次优方案。比较每辆汽车更改方案后的时间，并选取用时最少的 n 辆汽车实施方案更换，剩余汽车仍使用原方案。此处 n 值为拥堵路段实时流量与最大流量的差值，即满足道路车流量限制最少应移去的车辆。更换方案后的总路径规划相较于第一次路径规划为次优方案，也是在满足该道路车流量限制下的最优方案。

更新道路信息后，再次循环遍历，比较道路实时流量与最大流量，并重复上述操作，直至没有拥堵路段为止。

特别地，考虑存在为某辆车更改方案后存在原饱和或非饱和路段变至过饱和情况，且为满足解的正确性与最优性，考虑**对每辆车记录其已更换过的方案**。例如，若道路 1 在 t==2 时过饱和，需移去一辆车。在上述操作后，车辆 A 从道路 1 中移出，并更换方案行驶道路 2。

同时将（1，2）计入车辆 A，表示车辆 A 已从此方案更换过。后续发现，道路 2 过饱和并仍需更换车辆 A，车辆 A 再次换回原方案，移入道路 1 中。此时道路 1 再次过饱和，若直接比较，将再次移动车辆 A 进入道路 2，程序将陷入无限的循环。故检查 A 是否含有（1，2），即表示在 t==2 时刻在道路 1 的方案。若含有，则在本次移动时选取除去车辆 A 以外的最优车辆 B 进行移动。从而避免程序陷入无限循环，同时也最大化的贴近时间最优解。

由于保证测试点解的存在性，故使用该方案，在不存在必须停车等待的情况下，经过有限次循环替换操作，一定能求出满足车流量限制的路径方案，并最大化的靠近所用时间最短的最优方案。

3、项目实现思想

项目整体使用了 **Dijkstra 算法**实现最短路径求取；从**最优方案出发，逐步求解次优方案**并替换，直至满足所有题设条件为止；路径规划的优先级为车辆的行驶时间。

二、项目功能

1、数据结构实现

```
int N; //地点数量
int M; //车辆数量
```

根据题设输入格式，设置 N、M 存储地点、车辆数量信息。

```
struct Nomove{
    int i;
    int j;
    int t;
}; //记录已发生过替换的道路信息

struct Car{
    int start; //出发地
    int end; //目的地
    int time; //行驶时间
    stack<int> road; //存放车辆行驶路径的栈 ---> 由于在path中车辆行驶路径是逆序遍历的，所以使用栈存放，便于后续输出
    vector<Nomove> no_move;//记录车辆是否在该路径该时间点已经发生过替换
}; //单辆车信息
Car car_num[20]; //存放所有车辆的数组 编号从0开始
```

设立**结构体 Car**，存储车辆相关信息。包括：出发地 start、目的地 end、行驶时间 time。用**栈**存放车辆的形式路径，用**向量 vector** 记录车辆已替换的方案信息。设立**结构体 Nomove**，存储替换方案的道路信息，包括道路编号 i、j，拥堵时间点 t。

用**容量为 20 的数组** car_num 存储每辆车，下标从 0 开始。（与题设下标从 1 开始有冲突，在最后输出时加 1 即可。后续道路信息相同处理）

```
struct road { //道路信息
    int f_t[400] = {0}; //每个时间点的流量，预测最多有400个时间点，初始化为0
    int f; //道路流量
    int l; //道路长度
    stack<int> car_t[400]; //每个时间点道路上的车辆
};
road roads[20][20]; //存放所有道路的数组 编号从0开始

struct r_overf { //道路拥堵信息
    int i; //道路编号
    int j;
    int t; //时间点
    int number; //超载车辆数目
};
stack<r_overf> overf; //存放道路拥堵信息的栈
```

设立**结构体 road**，存储道路的相关信息。包括：最大车流量 f，道路长度 l，**容量为 400 数组 f_t** 存储每个时间点的道路实时流量。（估计单辆车行驶的最长时间不超过 400），用**栈数组**存放每个时间点道路上的车辆信息。并利用 **20x20 的邻接矩阵 roads** 存储每条道路，下标从 0 开始。

设立**结构体 r_overf**，存储拥堵道路的相关信息。包括：道路编号 i、j，拥堵时间 t，道路超载车辆数目 number。用**栈**存储拥堵道路的信息。

```
#define MAX 0x7fffffff //定义无穷大
```

宏定义 MAX 为无穷大，用于后续对道路信息的初始化与删除操作。

2、主函数介绍

主函数首先调用 **init 函数**对交通系统的各信息进行初始化。

其次，利用 **cin 和三个并行的循环**依次读入各交通信息，并存储在相应数据结构内。

先调用 **Dijkstra_recordf 函数**计算每辆车不考虑车流量时的最短路径，并同时记录各道路的实时交通信息（该函数同时可满足基础功能的实现）。

设置变量 T，并调用 **time_end 函数**将最大车辆行驶时间赋值给 T，便于简化后续遍历道路时间数组寻找拥堵路段的操作。调用 **check_f 函数**寻找拥堵路段，并将拥堵路段存入栈 overf 中。

设置 **while 循环条件**为栈非空，即当且仅当 overf 中不存放任何信息，当前方案下不存在拥堵路段时，跳出循环，得出最终方案。

循环内，调用 **re_plan 函数**，对栈顶拥堵路径进行重规划。重规划结束后，由于方案被更换，故再次对 T 赋值最大车辆行驶时间，并再次调用 **check_f 函数**查找拥堵路段。重复操作，直至跳出 while 循环。

最后，根据每辆车的行驶时间计算方案总行驶时间。利用**栈顶 top 操作**输出各车的行驶路径，并按照要求格式设置空格与换行。

主函数执行结束，返回 0。

3、功能函数介绍

init

init 函数通过 for 循环实现对道路最大流量、道路长度、车辆起始地、目的地和行驶时

间的初始化。防止因为内存中脏数据影响后续判断。

在 main 函数中调用。

Dijkstra_recordf

Dijkstra_recordf 函数利用 Dijkstra 算法实现求解最短路径的基础功能。设立 dist 数组存放起点到各点的最短距离，path 数组存放当前最短路径，S 数组标记该点是否已找到最短路径。在算法执行结束后，因为 path 数组内实则为逆向的行驶路径，故使用栈存储刚好可以使车辆行驶路径栈顶为起始地点，并且在循环 push 的过程中，从最后的时间点开始，逆时间记录路段各时间点的实时流量与车辆信息。特别地，由于使用邻接矩阵存储，故该处也应对称记录两次信息。最后用 dist[end]更新车辆的行驶时间。

在 main 函数中调用。

Dijkstra

Dijkstra 函数仍旧利用 Dijkstra 算法计算最短路径。在 Dijkstra_recordf 函数的基础上，删去了关于道路信息的记录，仅将车辆路径与行驶时间存放在临时数组 car_temp 中。car_temp 在函数 re_plan 中定义。

在 re_plan 函数中调用。

check_f

check_f 利用三重 for 循环，遍历查找当前方案下的拥堵路段。特别的，函数 check_f 函数参数为 T，可在第三重循环时使用。即只在当前方案的最长用时内探查每条路径的时间数组，减少无效探查次数。通过比较实时流量与最大流量判断路径是否拥堵。对应拥堵路径，设置遍历 temp 存储路径相关信息，并将其计入栈 overf 内。特别的，由于每更改一条路径的相关方案，都会导致其他路径的流量变化。即，改变一条路径方案后，其后续原先探查的拥堵路径流量已被改变，数据失效。故此处经探查到第一条拥堵路径便返回，即栈内只存放一条路径，从而可减少无效的探查操作。

在 main 函数中调用。

time_end

time_end 函数利用 for 循环，遍历查找当前方案下最长的车辆行驶时间，并返回该时间。

在 main 函数中调用。

sort_car

sort_car 函数利用 sort 方法对数组 re_car 中的车辆进行由小到大排序。排序依据为 re_car 中车辆对应在 car_temp 中记录的车辆行驶时间。排序后再从后往前依次将车辆计入栈 car 中，使栈顶为最少耗时的车辆。

在 compare 函数中调用。

compare

compare 函数调用 sort_car 函数计算栈 temp_car，并根据 cur.number 的大小逐次从取出栈顶元素并存入数组 fresh_car 内。fresh_car 内存储需要实际更换方案的车辆，在 re-plan 函数中定义。

在 re_plan 函数中调用。

remove_stack

remove_stack 函数用于从栈中移除特定值的元素。在更新道路信息时，由于车辆更换道路，故需在原方案道路存放车辆信息的栈中将该车辆移除。同时由于使用邻接矩阵存放道路，故移除时也应对应对称进行两次。

在 re_plan 函数中调用。

re_plan

re_plan 函数用于重新规划超载路段。函数在一个 while 循环中进行，循环条件为 overf 栈非空。首先取 overf 栈顶的拥堵路段并记录，设立并初始化存放需要重规划的车辆信息的数组 car_temp。将需要重规划的车辆加入数组 re_car 中。注意此处使用 top、pop 操作结合取栈中元素，操作结束后，需再次将车辆信息 pop 回栈中，以保证道路信息在更换方案前不被改变。

将该拥堵路段长度设为 0，以在后续 Dijkstra 函数调用中能正确排除该路径。Dijkstra 操作结束后，恢复路段长度。遍历 re_car 数组中的车辆，依次进行 Dijkstra 操作。特别的，需要先判断该车辆对应的已替换方案中是否有当前路径。若有，则直接将该车辆的更换后行驶时间设置为 MAX，并直接进入下一循环。该操作是为了保证已经替换过该路径的车辆重

返路径后不会进行二次替换，防止程序陷入无限循环。调用 compare 函数求解真正需要更换路径的车辆，并存入 fresh_car 数组中。

遍历 fresh_car 中的车辆。首先将该拥堵路径信息加入该车辆的已替换方案中。其次遍历该车辆原方案的所有路径，逐个删去相关路径中有关该车辆的信息。即相关时间点流量-1，调用 remove_stack 函数从栈中移除车辆。最后根据 car_temp 中存储的车辆新方案更新车辆和道路信息。即更新车辆的行驶时间、路径存放栈，相关路径道路流量+1，将车辆计入道路车辆栈中。

在 main 函数中调用。

三、测试结果

包括 PJ 文档中提供的示例样例在内，共自行设置了 8 个测试点，每个测试点运行结果如下：

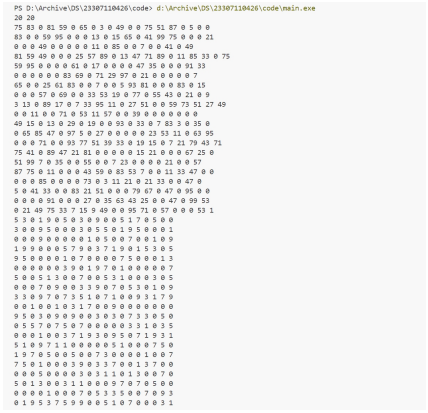
（1）

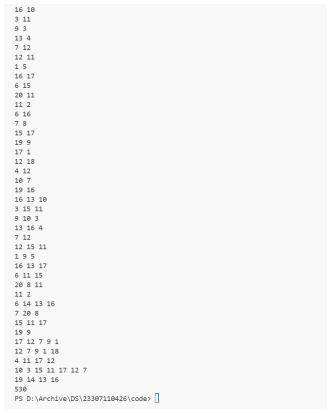
（2）

（3）

（4）

（5）





（6）

（7）

（8）

测试数据输入与输出均按照标准格式

四、收获感想

作为一名转专业的学生，这是我就读计算机专业的第一学期，也是学习 Cpp 语言的第一学期。第一次独立完成一个小型项目，还是比较有成就感的。虽然最后的实现可能不是最优解，但也在我最大的努力下实现了方案的求解以及时间最短的最大化求解。

相较于上机有测试点进行检验，本次项目 pj 隐藏测试点，大大增大了实验难度。第一遍代码写完后，在只有示例样例的情况下，只能检验基础功能的正确性。并不知道自己的高级功能是否能成功实现。在构造了几个测试点后，才发现第一遍的代码内仍有许多 bug。逐个修正后，才得出最终提交的代码版本。

本学期的数据结构学习也告一段落了。非常感谢认真负责的张老师，也非常感谢一直在给予我帮助、细心耐心的各位助教。最后，祝大家期末顺利，寒假愉快。

五、附件

1、获得帮助

感谢孔恩燊同学提供的测试点 text5（20x20 数据点）。

2、测试点数据

8 个测试数据在提交的第二个压缩包中，命名为“测试点数据”。