­问题三的模型建立与求解

假设：

1. 所有乘客从等待队列出列时间相同
2. 假设在乘客准备乘车前并行车道上的第一批出租车已经抵达各列等候

**1.单个上车点位置的求解**

问题三要求设置一个上车点，并给出乘客和出租车的批量安排方案，使得乘车的效率最高。

乘车的效率我们以所有候车乘客完成搭载离开的总总时间作为乘车效率的度量。因此，我们先确定了乘客的平均移动速度为1.5 m/s，根据搜集到的机场资料，结合题目要求出租车批量从停车场行驶入缓冲区后再进入上车点路段，出租车进入后供乘客上车的列数不会超过20列，我们利用贪婪法确定乘客出队后的上车方案，用搜索与模拟的方法确定出租车总用时最少的列数以及上车点的位置。

乘客上车的顺序服从先到先服务的原则，即先来的乘客会先从上车点的排队队列中出队寻找上车车辆,上车点的排队队列中的人必须等待前面的人出队后，才能出队寻找自己的车辆，每一次管理人员都批量的从缓冲区放固定数量的出租车进入机场乘客搭乘的道路区域上，

在此人员密集区域，车与车之间至少需要保证1~2 m的距离以保证安全和方便乘客放置行李等物品，所以综合来看设置两车之间的间距为1.5m

（1）单侧上车设置单个上车点

单侧单上车点中点最优证明：

人车距离采用哈密顿格点距离，第一排和第二排的不同之处仅在竖直方向上所要经过的距离，所以只考虑横向所要经过的距离的不同即可，每个人分配的车不固定，考虑移动所需要的平均距离，即这个上车点到所有车的距离和假设上车点位于x,车与车之间的距离相同，取1，x的左侧有L辆车，右侧有R辆车。

下面讨论各个情况单一乘客的期望移动距离

* 当Left < Right时，上车点向右移动一个单位，距离之和会变小Right - Left个单位



* 当Left > Right 时，上车点向左移动一个单位，距离之和会变小Left - Right个单位



* 故上车点设在最中心处期望的距离最小，得证。



**2. 乘客上车方式的求解**

出租车是批量进入上车点地段的，即当前批次的出租车全部载客离开后才会进入下一批出租车，那么位于乘客队列头部的乘客能够尽快的离开是符合逻辑最优方案。在上车点路段，只有头部的车可以不依赖于其他车驶离上车点路段，并且，先出队的乘客在队列中的等待时间少，相对其他乘客抵达头部的时间少，所以当其抵达位于当前头部的汽车并上车准备出发后可以立即出发，并不需要等待前面的出发后才能够离开，在其离开后，后面的车作为新的头部也就能够以相对较短的时间离开当前路段，这样的顺序保证了两个并行车道车辆驶离的秩序。

据此，为了保证安全性，我们对乘客按下图所示的顺序安排乘客依次乘坐停放在上车点地段的出租车，

算法的建立：

变量的解空间：

列的数量：Column={2,3,4,……,Max} 表示所有小于可能的Max中所有的可能解

上车点的位置：Set={1,2,3,Max-1} 表示任意两车的中间位置

双目标优化的过程：

**STEP 1：基本参数的确定**

结合能够查询到的资料，我们确定了以下基本参数：

双并行车道上车点路段所能容纳最大车辆列数：Max\_colum = 20

汽车长度：Car\_length = 4.7 m

汽车之间的安全距离：safe\_distance = 2 m

航班飞机的上座率：client\_rate = 80 %

双车道汽车与路边的距离：d2 =

乘客移动速度：client\_v = 1.5 m

飞机上的最大乘客数量：Max\_plane\_count = 200

**STEP 2: 枚举所有可能的汽车列数**

上车点周围可能位于汽车列数为2 ~ Max\_colum之间，我们依次在不同的列数下进行子问题的最优化求解

**STEP 3：枚举所有可能的上车点**

为了方便乘客通行距离短，上车点总是设置在车与车的间隙位置，故设置单个上车点时，N排汽车共有N – 1 个可能的上车点，在固定了当前的列数情况下，对模拟乘客队列出队上车的情况再对当前固定的上车点进行总时间花费的计算

**STEP 4：上车点到所有位置总距离的计算**

对于任意一个按照上述排列的汽车C<i,j>，计算距离上车点的距离，将其视作哈曼顿距离，那么水平和竖直方向的距离就能分离成两个独立的距离，分开判断计算，最后计算总和即可。

画图表示距离

水平方向距离计算

Car<i,j>

i：表示水平标号

j：表示竖直标号

s：表示当前枚举到的上车点

C :汽车的长度

Sl：汽车之间的安全距离

\left\{\begin{array}{l}\frac{(c+sl)}{2} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad i =s \\\\ |i-s|·(c+sl) + \frac{(c+sl)}{2} \quad\quad i<s \\\\ \frac{c+sl}{2} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad i>s,i-s=1 \\\\ (|i-s|-1)\times (c+sl)+ \frac{(c+sl)}{2} \quad i>s,i-s\neq1 \end{array}\right.

竖直方向距离计算

\left\{\begin{array}{l} d\times3+cw\quad\quad j =1 \\\\ d \quad\quad\quad\quad\quad\quad j=2 \end{array}\right.

STEP 5 ：所有人员时间花费计算

通过确定的初始参数，计算当前列数和上车点所花费的总时间，并与全局最优值做比较，如果更优则更新全局最优值，判断是否达到搜索的边界，如果没有达到，返回STEP 2，继续对空间进行搜索，直到遍历所有的搜索空间，取得最优值

开始优化搜索过程

枚举所有的列数

枚举当前列数下所有可能的上车点

判断当前上车点和列的水平位置关系

确定与时间相关的各个初始参数

初始化全局最优解

枚举所有的列，求上车点到这些列的距离之和

上车点在当前列的邻近右侧

上车点在当前列的非邻近左侧

上车点在当前列的非邻近右侧

上车点在当前列的邻近左侧

判断当前上车点和列的竖直位置关系

当前位于第距离上车点远 的一侧

当前位于第距离上车点近 的一侧

计算水平竖直距离和

是否优于全局最优值

结束得到效率最高时候的列数和上车点

更新全局最优值

True

False

是否优于全局最优值

是否达到迭代边界

True

**算法流程图：**

代码：

|  |
| --- |
| 求解效率最高的列数和上车点位置 |
| 1. #include<bits/stdc++.h> 2. **using** **namespace** std; 3. #define rep(i,a,n) for(int i=a;i<n;i++) 4. #define per(i,a,n) for(int i=n-1;i>=a;i--) 5. #define fi first 6. #define se second 7. #define ll long long 8. #define pb push\_back 9. #define db double 10. #define close ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0); 11. #define eps 999 13. **typedef** pair<**int**,**int**>pii; 14. **typedef** vector<**int**> vi; 16. **const** ll mod=1e9+7; 17. // const db len=60; // 18. **const** db car\_len=2.4; 19. // const int tot\_car=180; 20. **const** db MAX\_COL=20; // 一行并行车道所能容纳的最大出租车数 21. **const** **int** client=200; // 一班客机下来的总乘客数 22. **const** db inf=1e10; 23. **const** db v=1.5; 25. **struct** record{ 26. **int** col,set; 27. db val; 28. }ans[100000]; 30. // int car\_number1 = tot\_car; // 单侧进入可以容纳的汽车数量 31. // int car\_number2 = tot\_car; // 双侧进入可以容纳的汽车数量 32. **int** nowclient=client; 33. db tot\_time; 34. record min\_time={inf,inf,inf};  37. **int** client\_cost()// 得到顾客来到车前上车花费的时间 38. { 39. srand(time(0)); 40. **int** x =rand(); 41. x%=10; 42. **if**(x>=0 && x<=8) **return** 5; 43. **else** **return** 10; 44. } 46. **void** init() 47. { 48. rep(i,2,MAX\_COL+1) 49. { 50. ans[i].val=1e10; 51. } 52. } 53. **void** one\_side() 54. { 55. init(); 56. tot\_time=0; 57. db max\_distance=-1e9; 58. db move\_time; 60. rep(col,2,MAX\_COL+1) // 枚举出租车的列数 61. { 62. move\_time=(4.7\*col+(col-1)\*2); 63. move\_time/=v; 64. rep(set,1,col) // 枚举所有上车点 65. { 66. nowclient=client; 67. record res; 68. // cout<<nowclient<<endl; 69. db time\_cost=0; 70. **int** round=1; 71. **while**(1){ 72. db time\_to\_out\_of\_que=1; 74. db max\_time=-1e9; 76. rep(c,1,col+1) // 当前所有列的出租车 77. { 78. rep(f,0,2) // 位于上下哪一层 79. { 80. db this\_time=0; 82. **if**(c == set) //即上车点在当前这一列的相邻右侧 83. { 84. this\_time += 3.35; //水平方向距离 86. **if**(f==0) this\_time += 0.65\*3+1.7; 87. **else** this\_time+=0.65; 88. max\_distance=max(max\_distance,this\_time); 89. --nowclient; 90. this\_time/=v; 92. } 93. **else** **if**(c < set) //这一列在上车点的左侧 94. { 95. this\_time += fabs(c-set)\*6.7+3.35;//水平方向距离 97. **if**(f==0) this\_time += 0.65\*3+1.7; 98. **else** this\_time+=0.65; 99. max\_distance=max(max\_distance,this\_time); 100. --nowclient; 101. this\_time /= v; 102. } 103. **else** //在右侧 104. { 105. **if**(c - set == 1) 106. { 107. this\_time += 3.35; //水平方向距离 109. **if**(f==0) this\_time += 0.65\*3+1.7;//竖直方向距离 110. **else** this\_time+=0.65; 111. max\_distance=max(max\_distance,this\_time); 112. --nowclient; 113. this\_time /= v; 114. } 115. **else** 116. { 117. this\_time += (fabs(c-set)-1)\*6.7+3.35; //水平 118. **if**(f==0) this\_time += 0.65\*3+1.7; 119. **else** this\_time += 0.65; 120. max\_distance=max(max\_distance,this\_time); 121. --nowclient; 122. this\_time /= v; 123. } 124. } 125. this\_time += time\_to\_out\_of\_que; 126. this\_time += (db)client\_cost(); // 加上乘客上车可能消耗的时间 127. **if**(round != 1) this\_time += move\_time; 128. // cout<<nowclient<<endl; 129. **if**(this\_time > max\_time) 130. { 131. max\_time=this\_time; 132. } 133. time\_to\_out\_of\_que++; 134. **if**(nowclient==0) **break**; 135. } 136. **if**(nowclient==0) **break**; 137. } 138. time\_cost+=max\_time; 139. **if**(nowclient==0) **break**; 140. round++; 141. } 142. **if**(time\_cost < ans[col].val) 143. { 144. ans[col]={col,set,time\_cost}; 145. } 146. **if**(time\_cost < min\_time.val) 147. min\_time={col,set,time\_cost}; 148. } 149. } 150. // cout<<"最大距离为： "<<max\_distance<<endl; 151. // rep(i,2,MAX\_COL+1) 152. // cout<<i<<"列"<<ans[i].val<<endl; 153. cout<<"最少是 ： "<<min\_time.col<<"列，"<<"上车点是 ："<<min\_time.set<<"花费时间是： "<<min\_time.val<<endl; 154. } 156. **int** main(){ 157. close 158. one\_side(); 159. } |

|  |
| --- |
| 求解效率最高的列数和上车点位置 |
| 1. #include<bits/stdc++.h> 2. **using** **namespace** std; 3. #define rep(i,a,n) for(int i=a;i<n;i++) 4. #define per(i,a,n) for(int i=n-1;i>=a;i--) 5. #define fi first 6. #define se second 7. #define ll long long 8. #define pb push\_back 9. #define db double 10. #define close ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0); 11. #define eps 999 13. **typedef** pair<**int**,**int**>pii; 14. **typedef** vector<**int**> vi; 16. **const** ll mod=1e9+7; 17. // const db len=60; // 18. **const** db car\_len=2.4; 19. // const int tot\_car=180; 20. **const** db MAX\_COL=20; // 一行并行车道所能容纳的最大出租车数 21. **const** **int** client=200; // 一班客机下来的总乘客数 22. **const** db inf=1e10; 23. **const** db v=1.5; 25. **struct** record{ 26. **int** col,set; 27. db val; 28. }ans[100000]; 30. // int car\_number1 = tot\_car; // 单侧进入可以容纳的汽车数量 31. // int car\_number2 = tot\_car; // 双侧进入可以容纳的汽车数量 32. **int** nowclient=client; 33. db tot\_time; 34. record min\_time={inf,inf,inf};  37. **int** client\_cost()// 得到顾客来到车前上车花费的时间 38. { 39. srand(time(0)); 40. **int** x =rand(); 41. x%=10; 42. **if**(x>=0 && x<=8) **return** 5; 43. **else** **return** 10; 44. } 46. **void** init() 47. { 48. rep(i,2,MAX\_COL+1) 49. { 50. ans[i].val=1e10; 51. } 52. } 53. **void** one\_side() 54. { 55. init(); 56. tot\_time=0; 57. db max\_distance=-1e9; 58. db move\_time; 60. rep(col,2,MAX\_COL+1) // 枚举出租车的列数 61. { 62. move\_time=(4.7\*col+(col-1)\*2); 63. move\_time/=v; 64. rep(set,1,col) // 枚举所有上车点 65. { 66. nowclient=client; 67. record res; 68. // cout<<nowclient<<endl; 69. db time\_cost=0; 70. **int** round=1; 71. **while**(1){ 72. db time\_to\_out\_of\_que=1; 74. db max\_time=-1e9; 76. rep(c,1,col+1) // 当前所有列的出租车 77. { 78. rep(f,0,2) // 位于上下哪一层 79. { 80. db this\_time=0; 82. **if**(c == set) //即上车点在当前这一列的相邻右侧 83. { 84. this\_time += 3.35; //水平方向距离 86. **if**(f==0) this\_time += 0.65\*3+1.7; 87. **else** this\_time+=0.65; 88. max\_distance=max(max\_distance,this\_time); 89. --nowclient; 90. this\_time/=v; 92. } 93. **else** **if**(c < set) //这一列在上车点的左侧 94. { 95. this\_time += fabs(c-set)\*6.7+3.35;//水平方向距离 97. **if**(f==0) this\_time += 0.65\*3+1.7; 98. **else** this\_time+=0.65; 99. max\_distance=max(max\_distance,this\_time); 100. --nowclient; 101. this\_time /= v; 102. } 103. **else** //在右侧 104. { 105. **if**(c - set == 1) 106. { 107. this\_time += 3.35; //水平方向距离 109. **if**(f==0) this\_time += 0.65\*3+1.7;//竖直方向距离 110. **else** this\_time+=0.65; 111. max\_distance=max(max\_distance,this\_time); 112. --nowclient; 113. this\_time /= v; 114. } 115. **else** 116. { 117. this\_time += (fabs(c-set)-1)\*6.7+3.35; //水平 118. **if**(f==0) this\_time += 0.65\*3+1.7; 119. **else** this\_time += 0.65; 120. max\_distance=max(max\_distance,this\_time); 121. --nowclient; 122. this\_time /= v; 123. } 124. } 125. this\_time += time\_to\_out\_of\_que; 126. this\_time += (db)client\_cost(); // 加上乘客上车可能消耗的时间 127. **if**(round != 1) this\_time += move\_time; 128. // cout<<nowclient<<endl; 129. **if**(this\_time > max\_time) 130. { 131. max\_time=this\_time; 132. } 133. time\_to\_out\_of\_que++; 134. **if**(nowclient==0) **break**; 135. } 136. **if**(nowclient==0) **break**; 137. } 138. time\_cost+=max\_time; 139. **if**(nowclient==0) **break**; 140. round++; 141. } 142. **if**(time\_cost < ans[col].val) 143. { 144. ans[col]={col,set,time\_cost}; 145. } 146. **if**(time\_cost < min\_time.val) 147. min\_time={col,set,time\_cost}; 148. } 149. } 150. // cout<<"最大距离为： "<<max\_distance<<endl; 151. // rep(i,2,MAX\_COL+1) 152. // cout<<i<<"列"<<ans[i].val<<endl; 153. cout<<"最少是 ： "<<min\_time.col<<"列，"<<"上车点是 ："<<min\_time.set<<"花费时间是： "<<min\_time.val<<endl; 154. } 156. **int** main(){ 157. close 158. one\_side(); 159. } |