RM2018 高中生夏令营 19 组技术报告

组员及分工：

嵌入式: 邴小斐　钟庭杰

算法 && UI: 邴小斐

机械: 陈子讯　杨丰源　孙嘉陶　陈启

Vlog: 余之乐

一、 需求分析（abstract）

机械:

第一代横向抓取，每层抓取 5 个块，装三层，共 15 个块
第二代分为两部分，第一部分为爪子负责收集乐高块及搭建城堡，每次只能夹一个乐高块；第二部分为履带及摩擦轮同时工作，将其吸取并抬升。

算法:

1.写算法计算第一阶段比赛连接 16 个点并铺满地图的连接方式.
2. 设计并编写 UI, 使输入输出更方便, 提升用户体验.

嵌入式:

1. 遥控器右侧 sw 拨到中间移动模式并用遥控器 1/2/3/4 通道和鼠标控制底盘移动, 转向.
2. 前后两套方案共:
键盘按键控制: 推杆的 MG996 舵机, 传送带 M2006 电机 + C610 电调, 压杆两个 MG6221, 两个摩擦轮 M2006 电机 + C610 电调, 摩擦轮的大扭矩直流电机 + L298N 电机驱动板.
遥控器右侧 sw 拨到下面机械臂模式并用 1/2/3/4 通道控制: 机械臂抬升 M3508 + C620, 推杆整体上升 M3508 + C620.
遥控器左侧 sw: 机械臂夹取放置 MG995.
遥控器右侧 sw 拨到上面云台模式并用 1/2 通道控制图传的 2 轴云台 MG996/MG6221.
由于第一版方案和第二版方案的前些版本使用的电机过多, can1 不够, 所以写了 can2, can1 和 can2 同时使用(对第一版方案来说也不够, 所以很多结构改用用舵机).
3. 使用超声波传感器, 颜色传感器等辅助操作, 颜色传感器检测到颜色时图传前方相应的 RGB 灯亮, 摄像头前会亮一片于是操作手通过第一人称视角就能知道摄像头看不

到的车上的积木块哪个位置哪里亮了.

4. 为了用遥控器或键盘控制时，使电机稳定的到达某个位置或速度，使用了速度单环 pid，位置单环 pid，速度位置双环 pid，不同功能的电机分别调参.

5. 逆向写代码…随着机械结构的减少代码越来越少最后比赛只剩俩 2006 转摩擦轮了.

二 所需技术点，关键词

机械:

第一代：连杆结构（将周向运动转化为轴向运动）、绕线轮抬升（利用定滑轮）

第二代：平行四边形结构

算法:

方法共有 3 种: 1. 启发式.　　2. 基于连通性的状态压缩动态规划.　　3. 深度优先搜索

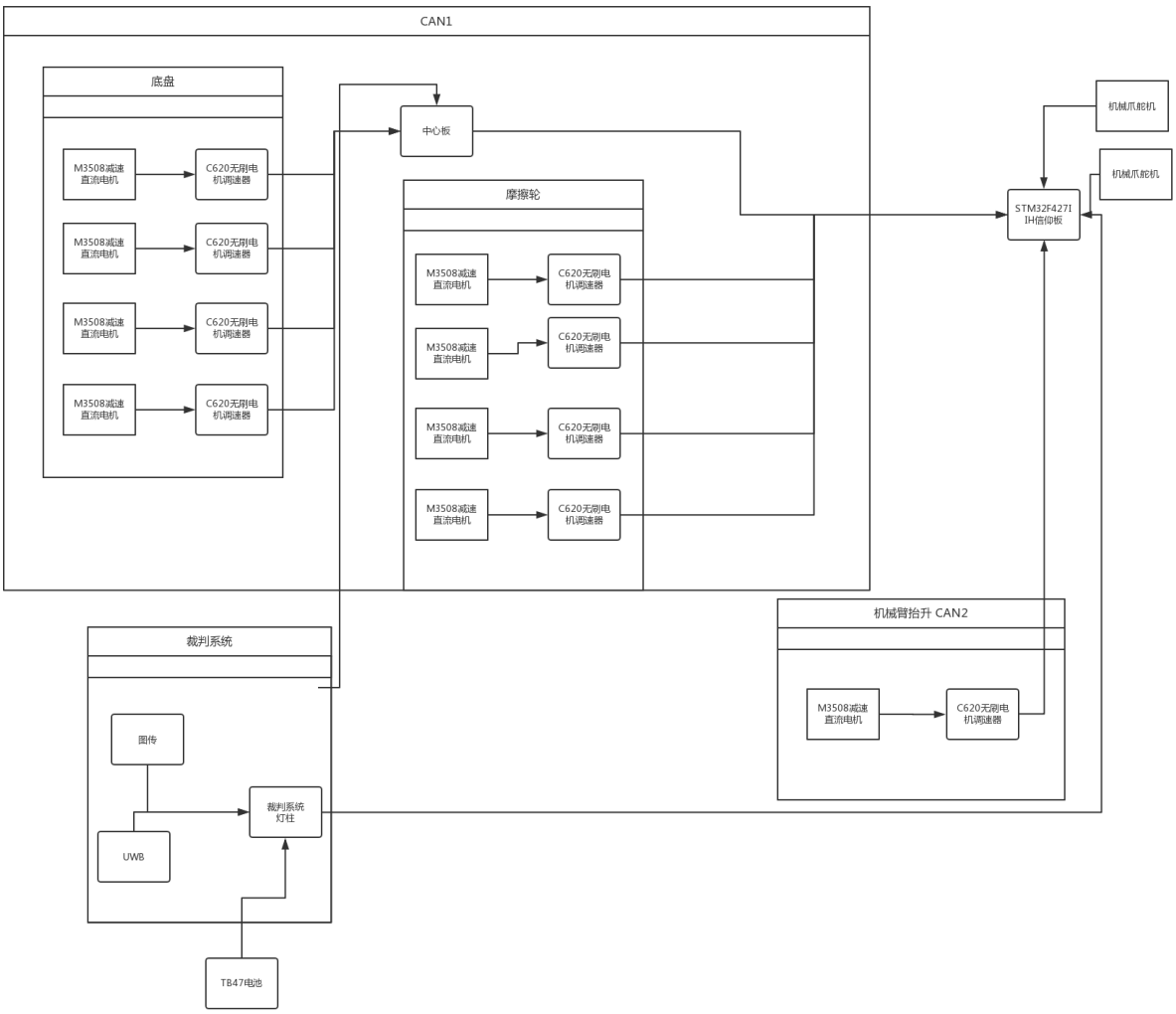
先使用深度优先搜索写完以后，发现稳定在 0.02-0.3s 之间，于是就没写另外两种.

UI: 查找资料了解 c，cpp 和 python 常见的用于写 UI 的框架，最后选择学习了 wxpython 和 QT，使用 QT 写了第一阶段算法的 UI.

嵌入式:

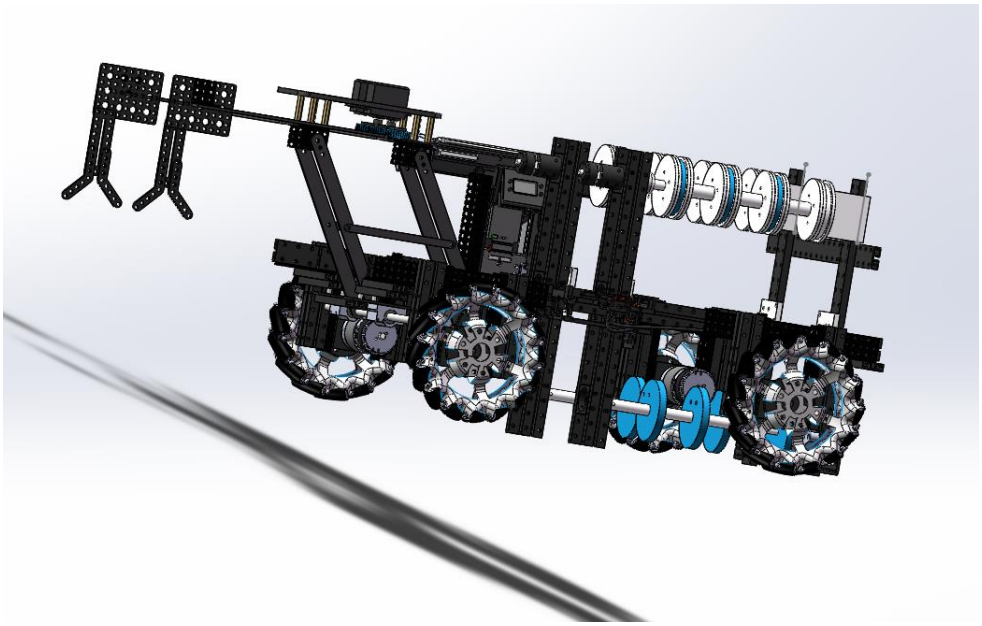
嵌入式部分: 如需求分析部分所示, M3508 和 M2006 使用 can 通信协议，使用了 can1 和 can2，大扭矩直流电机 + L298N 驱动板，舵机, RGB 灯也使用 PWM 控制.

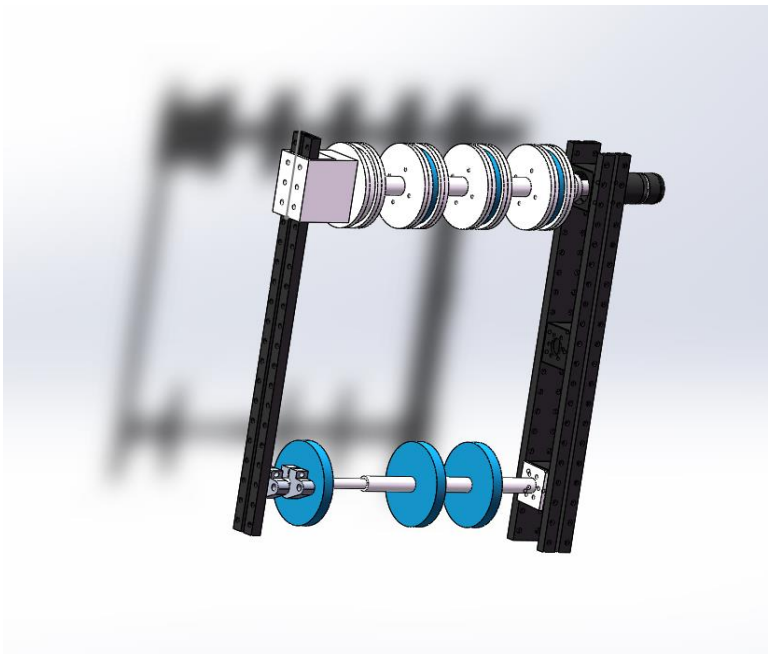
三 总体方案(第二版方案)


CAN1
底盘
M3508减速直流电机
C620无刷电机调速器
M3508减速直流电机
C620无刷电机调速器
M3508减速直流电机
C620无刷电机调速器
M3508减速直流电机
C620无刷电机调速器
中心板
摩擦轮
M3508减速直流电机
C620无刷电机调速器
M3508减速直流电机
C620无刷电机调速器
M3508减速直流电机
C620无刷电机调速器
M3508减速直流电机
C620无刷电机调速器
机械爪舵机
机械爪舵机
STM32F427I IH信仰板
裁判系统
图传
裁判系统灯柱
UWB
TB47电池
机械臂抬升 CAN2
M3508减速直流电机
C620无刷电机调速器

四 各模块方案

机械:





分工：

第一版和第二版机械臂, 第一版整车装配	陈子迅
第二版后部摩擦轮收集部分	杨丰源
第二版底盘装配及渲染图	孙嘉陶
图传 2 轴云台及部分其他结构的部分零件	陈启

后部轮子部分为第一届阶段的高速收集
底盘夹子部分负责第二阶段收集及加固城堡

嵌入式写出逻辑框图，然后分模块介绍工作，比如电磁阀通信，can 通信，调 pwm，调 pid 最好按照你们的逻辑顺序来做，从小模块到整体。硬件的写下硬件框图，搞清楚哪跟线连哪根线，从底盘到云台怎么走线。以及搞清楚板子的接口情况

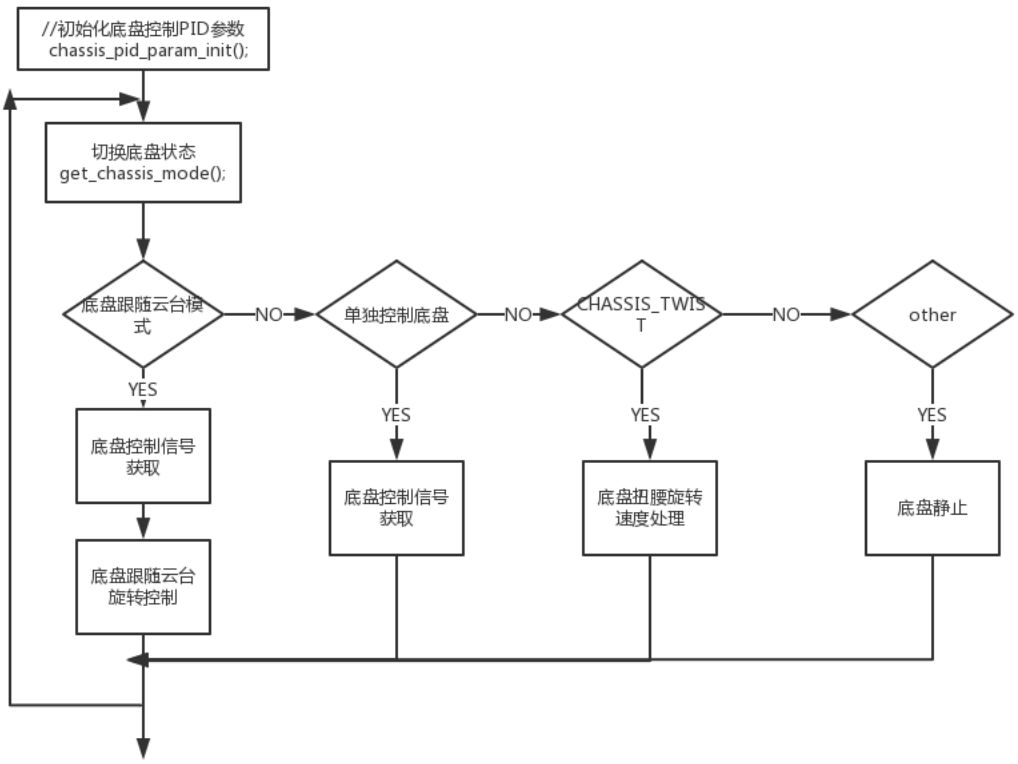
1、 算法部分写出流程图，思路和 UI 设计图

嵌入式:

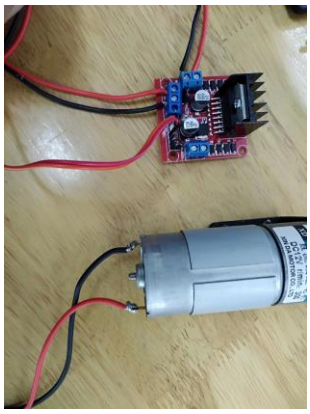
```
43    */
44  #define USER_TASK1 chassis_task
45  //#define USER_TASK2 gimbal_task
46  //#define USER_TASK3 detect_task
47  #define USER_TASK4 execute_task
48  //#define USER_TASK5
49
50  /**
```

两个线程

分别是控制底盘:



和机械臂等.

 L298N 电机驱动板

Can2 以及电机测试程序:

```
void send_2006_moto_current(int16_t current[])
{
  static uint8_t data[8];
  //int16_t trigger_current = trigger_moto_current;

  data[0] = current[0] >> 8;
  data[1] = current[0];
  data[2] = current[1] >> 8;
  data[3] = current[1];
  data[4] = current[2] >> 8;
  data[5] = current[2];
  data[6] = current[3] >> 8;
  data[7] = current[3];

  write_can(GIMBAL_CAN, CAN_GIMBAL_ID, data);
}

void send_w_3508_moto_current(int16_t current)
{
  static uint8_t data[8];
  //int16_t trigger_current = trigger_moto_current;

  data[0] = current >> 8;
  data[1] = current;
  data[2] = 0;
  data[3] = 0;
  data[4] = 0;
  data[5] = 0;
  data[6] = 0;
  data[7] = 0;

  write_can(2, CAN_CHASSIS_ID, data);
}
```

```
while(1) {
  //电机的速度给定
  w_2006_moto_speed[0] = rc.ch1 / RC_MAX_VALUE * MAX_WHEEL_RPM;
  w_2006_moto_speed[1] = rc.ch2 / RC_MAX_VALUE * MAX_WHEEL_RPM;
  w_2006_moto_speed[2] = rc.ch3 / RC_MAX_VALUE * MAX_WHEEL_RPM;
  w_2006_moto_speed[3] = rc.ch4 / RC_MAX_VALUE * MAX_WHEEL_RPM;
  w_3508_moto_speed = rc.ch4 / RC_MAX_VALUE * MAX_WHEEL_RPM;


  //闭环计算电机电流
  //test_moto_current[0] = pid_calc(&pid_test_moto, moto_test.speed_rpm, w_2006_moto_speed);

  for (int i = 0; i <= 3; i++)
    w_2006_moto_current[i] = pid_calc(&pid_2006_moto[i], w_2006_moto_ID_5_plus[i].speed_rpm, w_2006_moto_speed[i]);
  w_3508_moto_current = pid_calc(&pid_w_3508_moto, moto_3508_ID_1.speed_rpm, w_3508_moto_speed);
  send_w_3508_moto_current(w_3508_moto_current);
  osDelay(5);
  send_2006_moto_current(w_2006_moto_current);
  osDelay(5);
}
```

有一天突然 3 个一直没问题的版本的程序都突然不好用了(没动过), debug 发现如图的 can2 的 3508 发送电流函数运行了 101ms, 并不知道为什么突然不好用了. 参考 M2006+c610 电调信号时序的问题需要在 while(1)后加 osdelay(5), 变脑补在上电以后的 execute_task 线程进入 while(1)前加了 osdelay(3), 它就好了, 嵌入式可真神奇! (一直记得去年夏令营东流哥曾经说过, “玄学都是能力不足造成的”, 所以不相信玄学, 一定还是哪里出了问题. 会继续努力, 希望将来遇到玄学问题能找到原因.).

算法:

基于连通性的状态压缩动态规划是 08 年国集 dalao 写的一篇论文讲述的, 比较经典, 碰巧之前集训也遇到过相关的题. 所以看到这题时候的反应就是插头 dp.

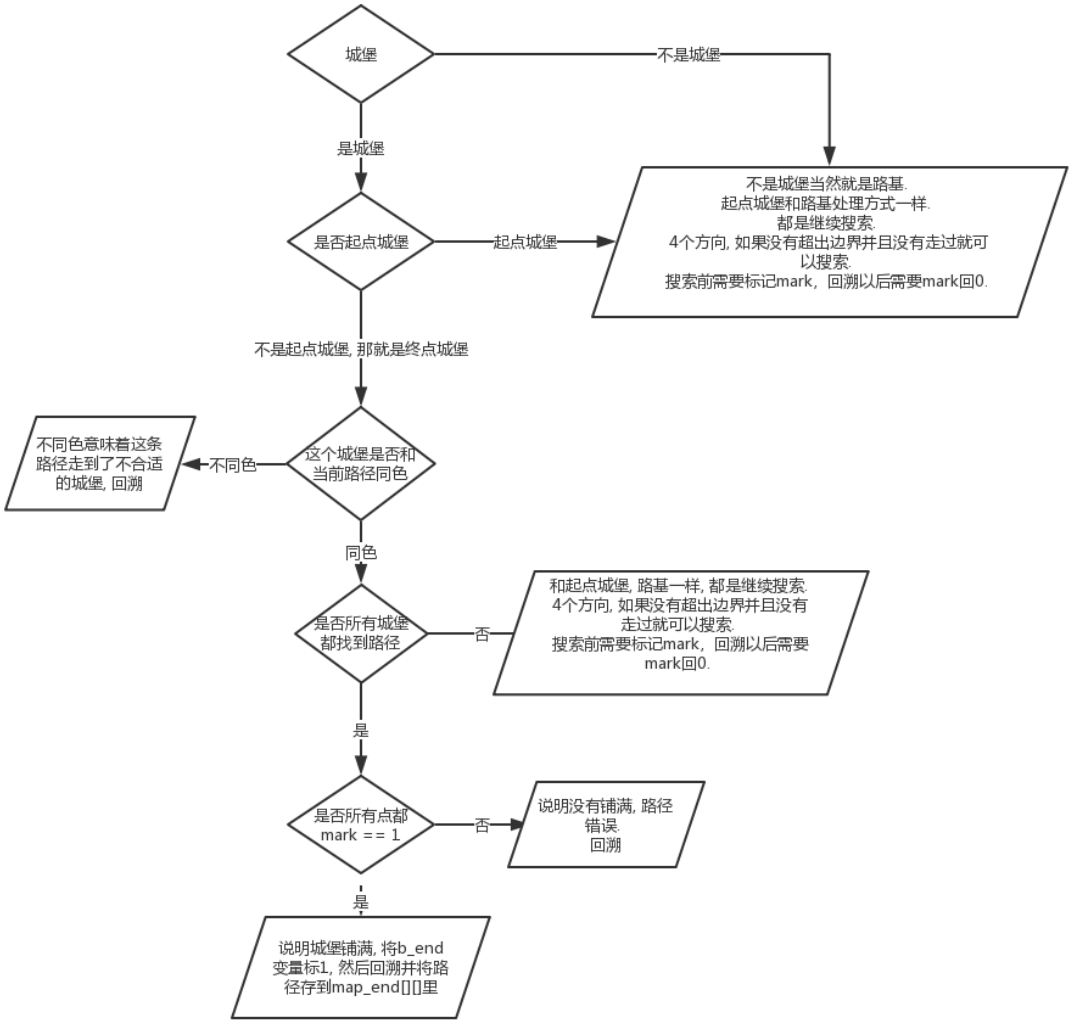
启发式是偶然从 GitHub 看到的, 还没仔细阅读, 等有时间学习一下.

接下来介绍写的深搜.

https://github.com/Webb-Bing/RoboMaster/blob/master/RoboMaster2018SummerCamp/Algorithm/EastFlow_Webb/EastFlow_Webb/EastFlow_Webb.cpp

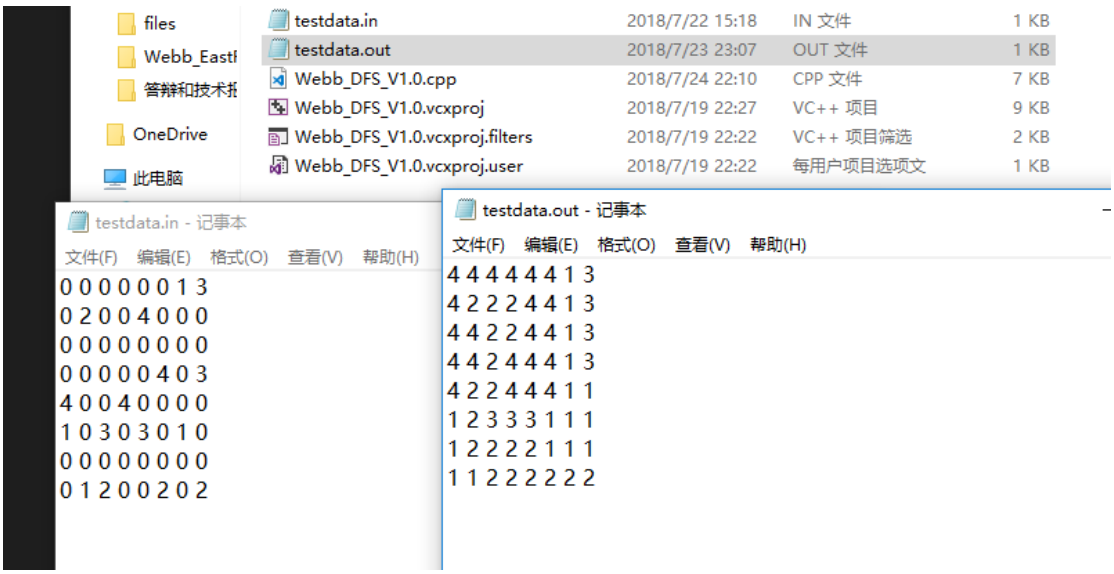
(优雅的 70 行)

```
void DFS(int i, int j, int color, int b_begin) {
    int ti, tj;
    if (map[i][j] && b_begin == 0) {
        if (map[i][j] == color) {
            for (int i = 1; i <= 16; i++)
                if (mark[castle_node[i].i][castle_node[i].j] == 0) {
                    mark[castle_node[i].i][castle_node[i].j] = 1;
                    DFS(castle_node[i].i, castle_node[i].j,
                        map[castle_node[i].i][castle_node[i].j], 1);
                    mark[castle_node[i].i][castle_node[i].j] = 0;
                    return;
                }
            for (int i = 1; i <= 8; i++)
                for (int j = 1; j <= 8; j++) {
                    if (mark[i][j] == 0) return;
                }
            b_end = 1;
        } else {
            return;
        }
    } else {
        for (int k = 0; k <= 3; k++) {
            ti = i + next_direction[k][0];
            tj = j + next_direction[k][1];
            if (ti >= 1 && ti <= 8 && tj >= 1 && tj <= 8 && mark[ti][tj] == 0) {
                mark[ti][tj] = 1;
                DFS(ti, tj, color, 0);
                mark[ti][tj] = 0;
                if (b_end == 1) {
                    map_end[i][j] = color;
                    return;
                }
            }
        }
    }
    return;
```

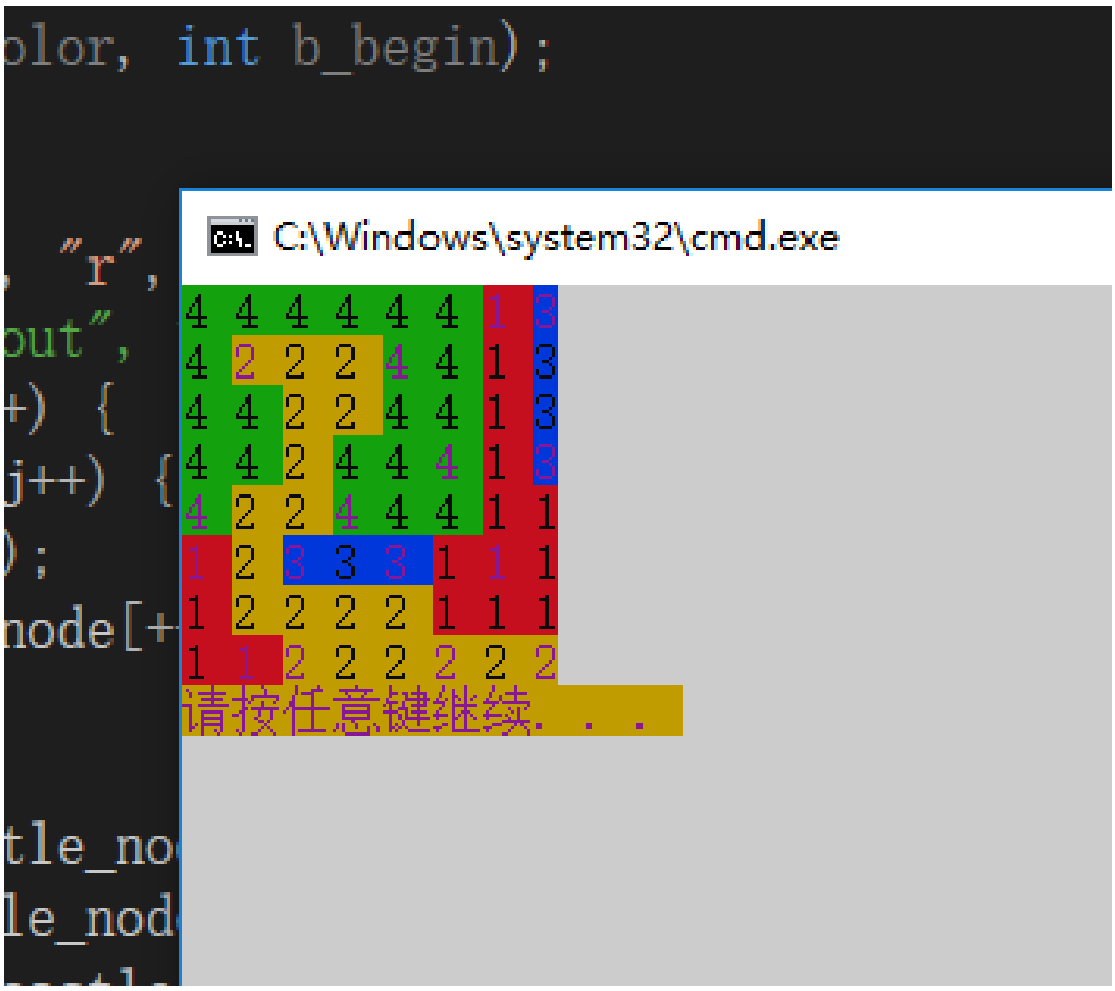

城堡
不是城堡
是城堡
是否起点城堡
起点城堡
不是城堡当然就是路基.
起点城堡和路基处理方式一样.
都是继续搜索.
4个方向, 如果没有超出边界并且没有走过就可以搜索.
搜索前需要标记mark, 回溯以后需要mark回0.
不是起点城堡, 那就是终点城堡
这个城堡是否和当前路径同色
不同色
不同色意味着这条路径走到了不合适的城堡, 回溯
同色
是否所有城堡都找到路径
否
和起点城堡, 路基一样, 都是继续搜索.
4个方向, 如果没有超出边界并且没有走过就可以搜索.
搜索前需要标记mark, 回溯以后需要mark回0.
是
是否所有点都mark == 1
否
说明没有铺满, 路径错误.
回溯
是
说明城堡铺满, 将b_end变量标1, 然后回溯并将路径存到map_end[][]里

输出:

1. 读写文件


files
Webb_EastI
答辩和技术报
OneDrive
此电脑
testdata.in 2018/7/22 15:18 IN 文件 1 KB
testdata.out 2018/7/23 23:07 OUT 文件 1 KB
Webb_DFS_V1.0.cpp 2018/7/24 22:10 CPP 文件 7 KB
Webb_DFS_V1.0.vcxproj 2018/7/19 22:27 VC++ 项目 9 KB
Webb_DFS_V1.0.vcxproj.filters 2018/7/19 22:22 VC++ 项目筛选 2 KB
Webb_DFS_V1.0.vcxproj.user 2018/7/19 22:22 每用户项目选项文 1 KB
testdata.in - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0 0 0 0 0 0 1 3
0 2 0 0 4 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 4 0 3
4 0 0 4 0 0 0 0
1 0 3 0 3 0 1 0
0 0 0 0 0 0 0 0
0 1 2 0 0 2 0 2
testdata.out - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
4 4 4 4 4 4 1 3
4 2 2 2 4 4 1 3
4 4 2 2 4 4 1 3
4 4 2 4 4 4 1 3
4 2 2 4 4 4 1 1
1 2 3 3 3 1 1 1
1 2 2 2 2 1 1 1
1 1 2 2 2 2 2 2

2. 控制台彩色输出



3. UI

就是批量创建多个 pushbutton, 然后响应同一个槽函数.

每次按下按钮 Map[i][j]++;　　(map[i][j] + 1) % 8 就是这个按钮的颜色.

然后一个 RUN 将鼠标点击的结果写入 testdata.in, 运行算法.exe, 读 testdata.in 并将结果输出到 testdata.out, 然后 UI 读 testdata.out 并显示颜色.

由于 4 小时 QT 从入门到写 UI, 读写文件的地方遇到了问题, 在输出 49 位以内(0-48)时正常显示, 49-64 时显示乱码, 65 位及以上时就输出前 64 位看起来正常, 65 位及以后是乱码. 之所以说看起来正常, 因为算法的程序读了以后算出来全是 0(首先算法是肯定没问题的并且一直 freopen 读取文件), 也就是说算法的程序读 testdata.in 有问题. 在手动新建 txt 并另存为 utf_8 的 testdata.in 时也是同样的问题. 所以应该是文件编码的问题, 15 号以后会重写读写文件部分并上交 UI 的代码.

(或者是算法读文件用流试试…? 因为平时刷算法题不开 o2 也不写优化习惯 freopen + scanf)

五 制作与测试流程

机械:

作简单的动力选型，抬升及夹子部分使用平行四边形结构，可以在旋转过程中使抬升部

分始终保持与固定部分平行。3508 扭矩为 3 牛米，抬升结构距离为 200mm 左右，重心距离为 180mm 左右，所以 3508 能提供的力为 13.7N 约为 1.7kg，显然所需结构重量是低于额定重量的。第二代设计计划将底盘尽可能的做到最小，轴距：452mm，轮距：360mm，小的底盘设计使我们的车辆可以在资源区的空隙中穿梭自如。

六 结果与评价

机械:

由于第一代设计过于庞大，在被某位胖胖的何姓 boss 怼过以后，意识到了我们组的设计失误。在热身赛前一天拆掉了自己的机器。第二代设计过程还算流畅，但其中需要大量的 3D 打印件，在后期的制作过程中，因 3D 打印件的问题后部的收集机构未能制作完成，只好阉割掉，只得在比赛前几小时临时制作其他的收集装置以确认自己能够上场。
平行四边形的抬升结构是抬升机构中最稳定效率最高的抬升结构之一，我们的摩擦轮收集也是在测试过程中也体现出了意料之外的高效。可惜的是夹子在测试稳定后未能上场展示。

嵌入式和算法: 完成任务.

七 附录（Appendix）请将代码及机械制图等粘贴在附录中

八 感想与感悟

太多.