AI 第一次大作业——火柴变换

姜卓 自 74 2017010550

一、搜索算法思路

本次大作业的程序语言为 c 语言，创建火柴数位结构体 MATCH_NUM，包含单个火柴数位变化标志 matchflag 和火柴数位表示的数字 matchdata，搜索思路是定义好火柴数位的状态空间，包含每个火柴数位的 matchflag 和相应的 matchdata，则对于可以经过一步变换的式子，该算法可以搜索出变换后的等式。

现对代码中主要函数和重要技巧加以说明：

1、 通过定义火柴数位变化标志规定了火柴棍的合法移动

其中 0/10/100/1000 的设置可以使得六个火柴数位的合法移动状态唯一确定，详情见代码注释。

```
10
11  //定义单个火柴数位变化标志
12  #define FLAG_KEEP     0       //不移动
13  #define FLAG_MINUS    10      //减一根火柴
14  #define FLAG_ADD      100     //加一根火柴
15  #define FLAG_SELF     1000    //拿一根放到自身其他地方
16
17  //定义合法移动（六个火柴数位）
18  #define MOVE_TO_ANOTHER 110   // 10 + 100 + 0*4，从一个火柴数位拿一根，放到另一个火柴数位上
19  #define MOVE_TO_SELF    1000  // 1000 + 0*5，自己拿一根放自己身上其他位置
20  #define MOVE_FROM_PLUS  100   // 100 + 0*5，从加号上拿一根放一个数身上
21  #define MOVE_TO_MINUS   10    // 10+0*5,从一个火柴数位拿一根，放到减号上
22
```

2、 状态空间的定义

函数 int　GetnumAfterMove 输入当下火柴数位 number 可以对其状态空间遍历，返回该 number 的状态数 StateCount.下图为 number=0 的示例。

```
44  //定义状态空间，并返回状态个数
45  int GetnumAfterMove(int number, MATCH_NUM* afterMatch)
46  {
47      int StateCount = 0;
48      switch (number)
49      {
50      case 0:
51      {
52          afterMatch[0].matchflag = FLAG_KEEP; afterMatch[0].matchdata = 0;
53          afterMatch[1].matchflag = FLAG_ADD; afterMatch[1].matchdata = 8;
54          afterMatch[2].matchflag = FLAG_SELF; afterMatch[2].matchdata = 6;
55          afterMatch[3].matchflag = FLAG_SELF; afterMatch[3].matchdata = 9;
56          StateCount = 4;
57          break;
58      }
```

3、 确定移动后等式的运算符号

函数 char　CheckMatchMove 输入各火柴数位变化标志位，结合用户/系统初始给定的符号 SCANF_SIGNAL，通过前文提到的合法移动定义，得出移动后的符号位 csymbol,并返回给函数，下图为一示例。

```
//给出合法移动后的等式符号类型
char CheckMatchMove(int flag1, int flag2, int flag3, int flag4, int flag5, int flag6)
{
    char csymbol = SYMBOL_ERROR;
    int matchflagSum = flag1 + flag2 + flag3 + flag4 + flag5 + flag6;
    if (SCANF_SIGNAL == '+')
    {
        if (matchflagSum == MOVE_TO_ANOTHER)
            csymbol = SYMBOL_ADD;
        else if (matchflagSum == MOVE_TO_SELF)
            csymbol = SYMBOL_ADD;
        else if (matchflagSum == MOVE_FROM_PLUS)
            csymbol = SYMBOL_MINUS;
    }

```

4、输出满足移动规则且使得等式成立的式子

函数 int printResult 的功能大抵是将式中各火柴数位的数字和运算符在满足移动规则的条件下转化成移动后的 matchdata 和运算符，并用全局变量 output 和 signal_output[0]存储，供后续访问。data[0]是一个标记值，作为函数返回值供后续使用。下图为输出等式为相加类型时的示例。

```
//输出满足移动规则且使得等式成立的式子
int printResult(M match1, M match2, M match3, M match4, M match5, M match6)
{

    int* data = new int[7];
    data[0]=0;
    int flag1 = match1->matchflag;    data[1] = match1->matchdata;
    int flag2 = match2->matchflag;    data[2]= match2->matchdata;
    int flag3 = match3->matchflag;    data[3] = match3->matchdata;
    int flag4 = match4->matchflag;    data[4] = match4->matchdata;
    int flag5 = match5->matchflag;    data[5] = match5->matchdata;
    int flag6 = match6->matchflag;    data[6] = match6->matchdata;

    char cSymbol = CheckMatchMove(flag1, flag2, flag3, flag4, flag5, flag6);
    if ((cSymbol == SYMBOL_ADD) && (data[1] * 10 + data[2] + data[3] * 10 + data[4] == data[5] * 10 + data[6]))
    {
        if (mode == 1)
        {
            output[0]=data[1]; output[1]=data[2]; output[2]=data[3]; output[3]=data[4]; output[4]=data[5]; output[5]=data[6];
            signal_output[0]='+';

        }
        checkSignal = '+';
        data[0]++;
    }
```

5、开始游戏，遍历搜索

bool play()函数执行搜索算法，对于六个火柴数位的所有状态空间遍历，执行方式为 printResult 函数。返回类型为 bool，为了判断给定的六个数字是否能符合游戏规则，即如果遍历结束后，计数变量 COUNT 刚好为遍历次数的相反数时，（printResult 在不符合规则时返回-1），bool 值为假。这个设计可以在后面随机生成式子时作为一个判断条件，控制随机式在符合规则时输出。

```
//开始游戏，函数内部不断遍历搜索
bool play(int data1, int data2, int data3, int data4, int data5, int data6)
{
    int COUNT=0;
    //六重循环 （本题搜索空间较小，最多10^6次运算,在statecount限制下会更小,故可行）
    int loop1, loop2, loop3, loop4, loop5, loop6;

    MATCH_NUM matchStick1[10];
    MATCH_NUM matchStick2[10];
    MATCH_NUM matchStick3[10];
    MATCH_NUM matchStick4[10];
    MATCH_NUM matchStick5[10];
    MATCH_NUM matchStick6[10];

    int StateCount1 = GetnumAfterMove(data1, matchStick1);
    int StateCount2 = GetnumAfterMove(data2, matchStick2);
    int StateCount3 = GetnumAfterMove(data3, matchStick3);
    int StateCount4 = GetnumAfterMove(data4, matchStick4);
    int StateCou int F1 GetnumAfterMove(data5, matchStick5);
    int StateCount6 = GetnumAfterMove(data6, matchStick6);

    for (loop1 = 0; loop1 < StateCount1; loop1++)
    {
        for (loop2 = 0; loop2 < StateCount2; loop2++)
        {
            for (loop3 = 0; loop3 < StateCount3; loop3++)
            {
                for (loop4 = 0; loop4 < StateCount4; loop4++)
                {
                    for (loop5 = 0; loop5 < StateCount5; loop5++)
                    {
                        for (loop6 = 0; loop6 < StateCount6; loop6++)
                        {
                            COUNT+= printResult(&matchStick1[loop1], &matchStick2[loop2], &matchStick3[loop3],
                                &matchStick4[loop4], &matchStick5[loop5], &matchStick6[loop6]);
                        }
                    }
                }
            }
        }
    }

    if ((COUNT== -StateCount1 * StateCount2 * StateCount3 * StateCount4 * StateCount5 * StateCount6))    △ equa
        return false;
    else
        return true;
```

二、UI 设计框架

界面相对简洁，能够满足游戏需求，基本控件为按钮，qlabel(在两个输入框下方以及运算符和等号)，qline 和数码管，先将使用方法加以说明。

游戏有两个模式：自定义模式和题库模式。

自定义模式：玩家首先需点击【自定义模式】，输入框内由玩家自己输入式子(两位数的加减乘运算)，点击【输入等式】后上方数码管显示输入，同时输入框下方显示相应提示：

情况 1“HAVE A SOLUTION!”表示有解，可在输入答案旁边的输入框写下你的答案，输入完成点击【输入答案】，会在下方提示“YOU WIN”or“YOU LOSE”，点击【显示答案】，数码管显示正确答案。

情况 2“NO ANSWER!”表示无解，退出即可。

题库模式：该模式系统随机出题，玩家需首先点击【题库模式】，然后点击【出题】，等待系统产生题目（有时候会很慢，时间达到 10 秒），然后下面用户可以在输入框写下你的答案，输入完成点击【输入答案】，会在下方提示“YOU WIN”or“YOU LOSE”，点击【查看答案】，数码管显示正确答案。

现将两种模式分别加以演示：

1．自定义模式

(a)点击自定义模式按钮，点击后其显示为“开始游戏”

(b)在框中自定义等式，eg：23+34=37

(c)点击输入等式后，数码管显示，提示信息：HAVE A SOLUTION !

(d) 在输入答案旁边的框中写下你的答案,然后点击“输入答案”，提示“YOU WIN!”

(e)点击“显示答案”，数码管显示答案。

2．题库模式

(a)点击自定义模式按钮，点击后其显示为“开始游戏”，再点击“出题”，等待一会系统便出题，数码管显示。

(b)输入答案后，点击“输入答案”，提示“YOU WIN!”

(c)点击查看答案，数码管加以显示。

三、调试体会和创作心得

1、 选择一种合适的编程语言很重要，本次作业我用 c 语言编写，在实现一些功能时发现，想要访问某些重要函数中的变量十分不方便，因此我也设了一些全局变量，也更改了原本函数的类型，比如 play 函数原本时 void 型，后为了实现后续功能，将其改成了 bool 型。以及 printResult 函数处理后的结果，我也另设数组加以储存。事后我想如果用 c++, 或者 python 的话，这样不必要的工作可以减少很多。

2、 要学会优化算法，且有一定的前瞻意识。我在遍历状态空间时发现对于一个 play()函数，里面有 6 个火柴数位，每一个数位的每种状态都要考虑，所以用了 6 个 for 循环，复杂度很高，当时我想每个数位最多 6 种情况，6^6 也不过是常数级，后来在实现随机给式子的时候发现导致运算速度很慢，这时才想到了一些效率更高的算法，可是时间已不允许我修改了。

3、 debug 要十分有耐心。对于一个比较大的工程，一次就像做对是很难的，都需要反复修改。对于某些之前不是很熟悉的软件和方法，出错的几率很高，想搞明白也很难，这时就需要高度的耐心和细心，反复推敲，认真思考。

4、 最后，本次大作业虽然自己实现的功能并不多，只有两个，但是也花费了我很多的时间和精力，也让我收获良多，study harder ！！！