程序设计基础大作业

实验报告

唐建宇 2017012221
软件 71

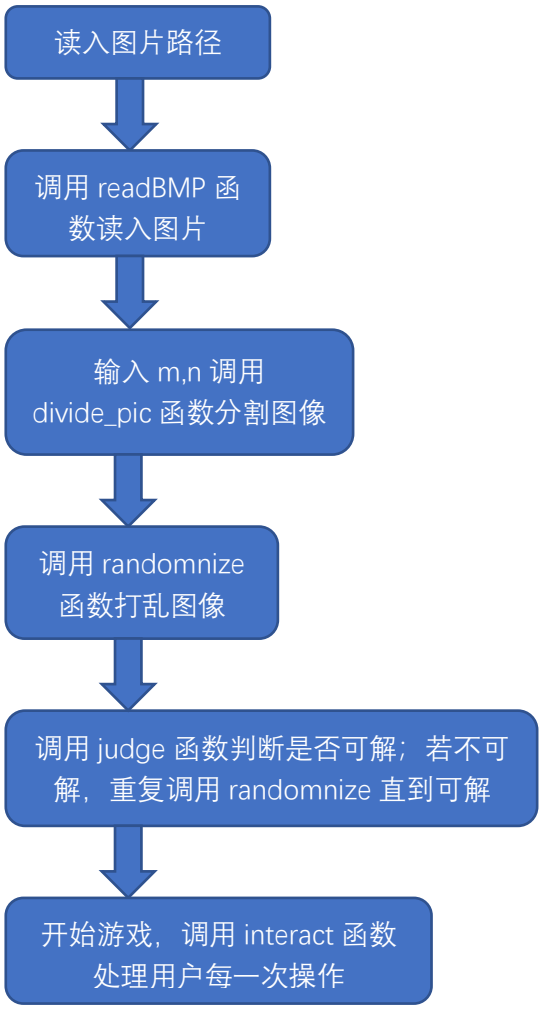
一、 环境说明

IDE: Visual Studio 2012
操作系统: Windows 10

二、 注意事项

1. 请确保读入的图片是 24 位 BMP 图片
2. 与可执行文件相同目录下存放了几个符合要求的实例图片

三、 程序整体运行流程



四、 核心思路

根据 BMP 图形格式，用两个结构体 BMP_head 和 BMP_info 存储图片开头 54 个字节的信息，图像像素区的信息逐像素（每 3 个字节）提取，按顺序存储到像素结构体 pixel 数组中。根据用户输入的分割方式，将图像分为 m*n 个 block，每个 block 存有相对应块中的像素信息。游戏操作过程是对一个数组进行操作，即每一个块对应一个数字，数组中的顺序反映了当前图像的状态，每次需要保存图片时，根据数组中的顺序将对应的 block 写入文件即完成了保存；游戏中保存进度等操作同样基于数组进行。

五、 关键函数与算法说明

1. BMP 图片处理部分

（1） 读图函数 bool readBMP(char *filename)

读取图片，若读取失败返回 0 并提示重新输入图片名与路径，将图像信息存入像素点 pixel 结构体的数组 picture_data 中。

算法：

存储图片开头 54 个字节信息的结构体：

```
struct header
{
    DWORD doc_size;
    WORD reserved_1;
    WORD reserved_2;
    DWORD data_start;
};

struct info_header
{
    DWORD size;
    long width;
    long height;
    WORD planes;
    WORD color_bit;
    DWORD compression;
    DWORD data_size;
    long xpels;
    long ypels;
    DWORD colors;
    DWORD important_colors;
};
```

以二进制读取模式打开图片，读取图片时，先分别读取以上两个

结构体。根据读到的长宽信息开始读取每一个像素的信息，遇到行位判断是否存在补零，如存在则跳过对应数量的零，关键代码如下：

```
for(int i=0;i<size;i++)
{
    fp.read((char*)&picture_data[i],sizeof(pixel));
    counter++;
    if(counter==BMP_info.width)
    {
        fp.seekg(add,std::ios::cur);
        counter=0;
    }
}
```

（2） 分割图形函数 void divide_pic(int n,int m,char *filename)

根据读图函数读入的图像信息与用户输入的分割要求，将图像分为 m 行 n 列，并在行列之间加入分割线，同时将右下角的块改为空白（灰色）。在进行分割的同时，将每一块的信息保存到一个 block 结构体中，每一个 block 对应的就是一块中的全部像素信息。将分割完后的整个图像信息读入一个字符型数组 data 中保存，之后的所有操作将是对这个字符型数组的操作，需要保存时直接写入这个数组即可。

算法：

为了写入图片需要，首先根据 m 和 n 值计算修改后的图片长和宽，以及大小，更改图片头（那两个结构体）的信息；接着开始增加分割线，具体做法是，按行写入，与读取时不同，每隔(width/n)加入一个黑色像素（0x000000）,每隔(height/m)行写入整行黑色像素。最终效果(已进行了打乱)：

（3） **写入函数 void writeblock(int n,int m,int block_num,int pos)**
将一个块（block）写入图片中一个指定的坐标，用于打乱和游戏中移动操作时更新图像信息。

算法：
定位到 data 中相关位置后写入即可。

（4） **打乱函数 void randomize(int m,int n)**
通过随机数生成一个随机的序列以达到打乱的目的。
根据打乱顺序调用 writeblock 函数更新 data 中的信息。

2. 判断可解性与自动求解部分

（1） **判断可解性函数 bool judge(int m,int n)**
如果可解，返回 1；否则返回 0。

算法：
通过数列的逆序数进行判断，空白块初始在右下角最终还要回到右下角，因此变换的次数一定是偶数次，而每一次移动操作都是一次变换，其对逆序数的变化位正负 1，所以能够还原的充分必要条件是当前状态的逆序数与理想状态的逆序数（在这里为 0）奇偶性相同，即当前状态的逆序数为偶数即可，因此，只要求出逆序数并判断奇偶性即可。

（2） **自动输出还原步骤函数 void autosolve()**
根据当前状态，程序会自动求解，并将步骤序列输出至屏幕。

能够快速实现较大 m 和 n 的求解(例如 30*30 大约用时 5 秒)，寻找路径本身是极快的，主要受限于屏幕输出的速度。

具体算法见亮点说明。

3. 游戏操作部分

操作函数 void interact(char ctr)

移动操作

以 W 为例

```
case 'W':
    if((blank-1)/n==m-1)
    {
        std::cout<<"非法移动，当前已在最顶部"<<std::endl;
        return;
    }
    swap(*(current+blank),*(current+blank+n));
    writeblock(n,m,current[blank],blank);
    writeblock(n,m,current[blank+n],blank+n);
    blank+=n;
    break;
```

首先判断是否达到边界，如果是合法移动操作，则在存有当前序列信息的 current 数组中进行交换，同时调用 writeblock 函数更新图像信息。

保存/读取进度操作 I/O

保存时，生成一个用户自定义名称的文本文件，向文件中写入当前的序列信息，即 current 数组。读取时，将文件的信息读入 current 数组并调用 writeblock 函数更新图像信息。

保存图片操作 G

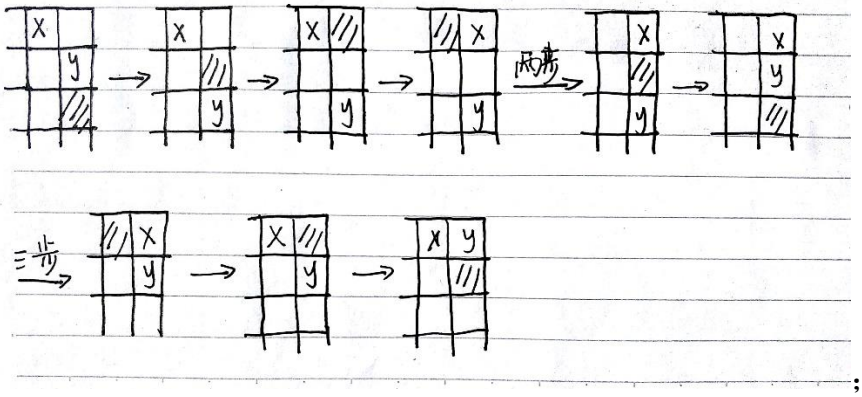
将当前的每一个块的信息写入一个给定的文件，即可保存。

```
case 'G':
    {
        char output_filename[256];
        printf("请输入您要保存图片的完整路径与名称:\n");
        std::cin>>output_filename;
        std::ofstream fout1(output_filename,std::fstream::binary);
        fout1.write((char*)data,len);
        fout1.close();
        printf("保存成功！\n");
        break;
    }
```

六、 附加功能与亮点

1. 较大 m 和 n 规模的自动求解

为了实现自动求解，我首先研究了人玩拼图的方法，经过学习和尝试，总结了一个方法：
按行从上到下从左到右复原每个位置，只要不是最下面两行和每行最后一个格子都是可以在不影响已复原的格子的情况下实现复原，这是显然的；
对于每一行最后一个位置，将应该在这个位置的格子移动到这个位置下方，并且将空白格移到他的下方，这总是可以在不影响已复原的格子的情况下容易地做到的。然后，按照一个固定的顺序即可将这个格子复原（如图所示）：



；

对于最下面两行，可以考虑将图像旋转 90 度，即变成了与上图相同的情况，这时候不再按行（旋转前的行）而是改为按列（旋转前的列）进行依次复原，根据上图所示的过程也可以看出，这是可以在不影响其他已复原格子的情况下完成的。

由此可以得出，拼图的复原是一个较机械的过程，其实不需要任何思考，只要按部就班处理即可。而我的自动还原算法也是依据上述过程，将这一过程用代码描述出来，通过 normalmove 函数（移动格子）和 blankmove 函数（移动空白格）实现简单的移动操作，对于上述特殊位置和过程单独处理即可。

我将这种算法与 A*进行了对比，通过 stl 中 map 里的哈希函数对序列进行哈希，对判断状态的操作进行了优化，但是也最多只能在合理的时间内解决约 7*7 的。而这种方法无疑高效、准确了许多。
复杂度分析：
每复原一个格子，移动次数约$5 \times (m + n)/2$，这是因为每移动一步，都需要将空白格重新移动到格子的移动方向，一般需要 4 步，再加上移动的一步，一共是 5 步，$\frac{m+n}{2}$为对曼哈顿距离的估计，平均情况约为此。因此复杂度为$O(mn(m + n))$。

2. 附加功能——永久保存进度

通过将当前状态写入文件，可以将进度永久保存下来，即在任何一次游戏中都可以读取以前已保存的进度然后继续游戏（只要存有记录的文件没有被人为删除）。

3. 附加功能——计时器与排行榜

游戏开始后，操作’B’为查看当前排行榜，会输出当前前十名的选手名及用时。

如果用户成功复原拼图，会输出用户的用时，根据用时，如果位于史上前十名，会提示用户输入自己的名字（英文字符），并更新排行榜。

排行榜通过 txt 文件的形式存储，在程序运行时，会将相关数据先读入一个链表中，并采用链表进行排序、更新等操作。