

大作业—拼图游戏

实验报告

目录

一、简介与界面展示	2
二、代码与关键函数总览	3
三、功能介绍	6
四、BMP 图档的结构与代码实现	7
五、拼图自动还原的实现	
5.1 M x N 拼图的通解	8
5.2 优化路径-BFS 解 3 x 3 拼图	10
六、(附加)其他讨论	
6.1 大作业中出现的问题与调试过程	11
6.2 算法复杂度估算	13

班级：计 86

姓名：周恩贤

学号：2018011438

日期：2019/01/06

备注

编译与运行 IDE：devC++ Version5.11

电脑系统版本：Windows 10 64 bit

一、简介与界面展示

本次大作业，利用数组(或 vector)储存拼图状态。对于解好的 $M \times N$ 拼图，数组编号从 $0-(M*N-2)$ ，对应的数字为 $\underline{1-(M*N-1)}$ （当时认为数字从 1 开始编号似乎比较整齐），数组编号第 $M*N-1$ 的数为右下角空白格记为 -1。

打开 **Puzzle Game.exe**，游戏一开始印出了标题与读图的提示信息。正确读图后，进入选单模式。使用者可选择玩 M, N （目前最大解到 **300x300**，需几个小时）拼图或者玩 3×3 拼图，两者的差别是在于自动解的方法不同（前者找出了通解，后者用宽搜找优化解）

假定我们选择了 M, N 拼图，程序会提示使用者输入拼图的大小（假定为 10×10 ），确认大小后随机打乱、判断是否可解（若不可解进一步询问是否更改为可解拼图）、并询问使用者是否继续游玩。

```
=====Puzzle Game=====
Arthor : Zhou EnXian
Date : 2019.01.06
=====
Please enter the path of the picture:
Please make sure it is a 24 bit-depth bmp file
l.bmp
Read Picture Success!

Please select a mode(not case sensitive) :
1: M*N Mode [General Solution]
2: 3*3 Mode [BFS]
P: Quit the game

Please enter the size of the puzzle M,N (M,N > 1)
10 10

Puzzle after shuffling:
73 26 72 67 88 37 11 12 27 79
53 83 51 98 58 48 55 33 22 87
17 9 92 24 15 52 29 80 63 70
91 45 18 40 57 93 10 5 59 49
99 81 97 56 50 20 8 7 2 68
62 30 34 64 23 85 28 46 3 44
13 35 21 90 36 71 75 84 43 96
19 31 69 39 25 61 74 -1 1 76
42 66 47 95 6 14 54 32 41 94
86 38 89 78 60 4 82 16 65 77

Solvable!
Keep Playing? [y/n]
```

进入游戏页面，印出可输入的指令与当前状态的拼图。输入对应指令并执行操作，直到解完拼图后，输出已成功解完的提示信息。

```
Instructions(not case sensitive) :
W: Switch the blank block with its upper block
A: Switch the blank block with its left block
S: Switch the blank block with its lower block
D: Switch the blank block with its right block
R: Restart the game with the origin puzzle
G: Save the current graph. Next line enter its path
I: Save the current procedure for the game. Next line enter the file name (case sensitive)
O: Read the previous-saved procedure for the game. Next line enter the file name (case sensitive)
L: Print out all the names of the saved files
P: Automatically solve the game and print out the solution
Q: Quit the game (Return to menu)
Note : You can move many steps in one line of enter

73 26 72 67 88 37 11 12 27 79
53 83 51 98 58 48 55 33 22 87
17 9 92 24 15 52 29 80 63 70
91 45 18 40 57 93 10 5 59 49
99 81 97 56 50 20 8 7 2 68
62 30 34 64 23 85 28 46 3 44
13 35 21 90 36 71 75 84 43 96
19 31 69 39 25 61 74 -1 1 76
42 66 47 95 6 14 54 32 41 94
86 38 89 78 60 4 82 16 65 77

Instructions(not case sensitive) :
W: Switch the blank block with its upper block
A: Switch the blank block with its left block
S: Switch the blank block with its lower block
D: Switch the blank block with its right block
R: Restart the game with the origin puzzle
G: Save the current graph. Next line enter its path
I: Save the current procedure for the game. Next line enter the file name (case sensitive)
O: Read the previous-saved procedure for the game. Next line enter the file name (case sensitive)
L: Print out all the names of the saved files
P: Automatically solve the game and print out the solution
Q: Quit the game (Return to menu)
Note : You can move many steps in one line of enter

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 -1

Congratulations! You've solved the puzzle!
```

同时，依照不同状况输出提示信息（如指令错误请重新输入、存档时档案重名、读图时档案不存在、自动解解法太复杂是否仍要输出等等……）

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 -1 89
91 92 93 94 95 96 97 98 99 90

t
You've entered a wrong instruction!
Please re-enter:

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 -1 89
91 92 93 94 95 96 97 98 99 90

o
Enter file name: abc
File Not Found!
```

二、 代码与关键函数总览

本次大作业所编写之所有档案、结构体、函数及其介绍罗列如下：

myfunction.h	<p>含所有函数宣告的标头档（包含 <code><bits/stdc++.h></code> 以及 <code>"windows.h"</code>），被所有其他档案给引用</p>
main.cpp	<p>主函数所在的档案</p> <p><code>int main();</code> 最开始的游戏阶段。印出标题、读入图片并进入下一阶段（选单）</p>
print.cpp	<p>印出标题、选单、提示语句等等的函数，同时适当的加入清屏、暂停功能让游玩过程更顺畅</p> <p><code>void print_error(string s);</code> 印出错误提示 s，注意需要同时清空 cin 的缓存区（详见 p.）</p> <p><code>void print_title();</code> 印出标题，包含作者以及完成日期</p> <p><code>void print_menu();</code> 印出标题，供使用者选择模式（M x N 或 3 x 3）</p> <p><code>void print_instruction();</code> 在游戏过程中，印出使用者可输入的指令与其相关说明</p> <p><code>void print_puzzle(int M,int N,int puzzle[]);</code> 印出当前状态的拼图</p>
read.cpp	<p>游戏进行时，需要使用者输入指令并进行判断的函数。读入时，我们利用 <code>cin.fail();</code> 判断是否读入正确数据类型，进而利用 <code>switch</code> 语句进行指令判别</p> <p><code>void read_1();</code> 读入使用者选择的模式</p> <p><code>bool readsize(int &M,int &N);</code> 读入 M x N 拼图大小，回传是否正确读入两个大于 1 的整数</p> <p><code>void read_2(int M,int N,int puzzle[]);</code> 打乱拼图后并显示后，询问使用者是否继续游玩，否则退回选单</p> <p><code>struct storage{vector < int > p};</code> 用于拼图存档的结构体，其中的 vector 对应到一个拼图</p> <p><code>void read_MxN(int M,int N,int puzzle[]);</code> M x N 拼图读入指令的函数，用 while(True) 无穷回圈持续进行</p> <p><code>void read_3x3(int puzzle[],int M = 3,int N = 3);</code> 3 x 3 拼图读入指令的函数，用 while(True) 无穷回圈持续进行</p>

run.cpp	<p>选择模式后到游戏开始前所用到的函数</p> <pre>void run_MxN(); void run_3x3();</pre> <p>打乱拼图、判断拼图是否可解以及询问是否继续游玩</p> <pre>bool sum_of_reverse_number(int M,int N,int puzzle[]);</pre> <p>判断拼图逆序数和的奇偶性，复杂度为$O(M^2N^2)$</p> <pre>bool can_solve(int M,int N,int puzzle[);</pre> <p>利用逆序数和判断是否可解</p> <pre>void reshuffle(int M,int N,int puzzle[],bool mark);</pre> <p>使用者同意后，重洗拼图直到可解。这里利用了交换相邻两非空白格拼图使得排列的逆序数奇偶性改变的小技巧。</p>
playgame.cpp	<p>游戏过程所会用到的函数。较复杂的函数（自动解、存图）另外写</p> <pre>void playgame(int M,int N,int puzzle[);</pre> <p>使用者选择模式后，输出提示信息并开始游玩</p> <pre>int find(int M,int N,int puzzle[],int num);</pre> <p>找到数字为 num 的拼图之 index，线性查找复杂度$O(MN)$</p> <pre>void change(int M,int N,int puzzle[],char c)</pre> <p>若为合法移动，依照指令 c 交换空白格与其指定相邻的拼图</p> <pre>bool check_solved(int M,int N,int puzzle[);</pre> <p>移动后，判断拼图是否已成功解完并输出提示信息</p>
picture.cpp	<p>BMP 图像处理的相关函数，方法详见第四章</p> <pre>bool read_picture(string path,unsigned char **pic, unsigned int *W,unsigned int *H);</pre> <p>利用 fread 从 path (利用 c_str()函数转为 char*) 读取图片，注意 BMP 的档案格式，并回传是否正确读入给 read_pic()函数</p> <pre>void read_pic();</pre> <p>呼叫 read_picture()函数并依照是否成功读图输出提示信息</p> <pre>void save_pic(int M,int N,int puzzle[);</pre> <p>呼叫 puz_to_pic()、save_picture()并依照是否成功存图而输出信息</p> <pre>bool puz_to_pic(int M,int N,int puzzle[], unsigned char **pic,unsigned int W,unsigned int H);</pre> <p>将图片切小块，并依照拼图状态与之对应。回传是否成功分割对应</p> <pre>void save_picture(string path,unsigned char *pic, unsigned int W,unsigned int H);</pre> <p>利用 fwrite 输出图片至 path (利用 c_str()函数转为 char*)</p>
auto3x3.cpp	<p>利用 BFS 找寻 3x3 拼图的可行解，方法详见第五章</p> <pre>struct Node{vector < int > puzzle,int p_b,Node * parent,char c};</pre> <p>节点记录着拼图状态、空白位置、父节点以及对应的指令</p>

	<p>Node * Up, Down, Left, Right(Node * &a); 建构式，用来接起父节点同时改变拼图状态，注意移动是否合法</p> <p>int auto_solve3x3(int puzzle[],vector<char>&sol); 利用 STL 中的 queue 进行 BFS 求解，并回传过程所用步数</p>
autoMxN.cpp	<p>利用一定规律寻求任意大小 M x N 拼图的通解，方法详见第五章</p> <p>void auto_change(int M,int N,int puzzle[],char c,vector<char>&sol); 和先前写的 change 函数差不多,利用sol.push_back(c) 记录结果</p> <p>vector<char> operator + (vector<char> v1,vector<char> v2); 利用运算符重载的方式定义两 vector 相加为相接： $v1 + v2 = v1.insert(v1.end(), v2.begin(), v2.end())$</p> <p>void append(int M,int N,int puzzle[], vector<char>&sol, vector<char> s) 定义 append 函数即为 sol+s；同时依照指令串 s 移动拼图</p> <p>void move_h(int M, int N, int puzzle[], int c, vector<char>& sol) 将空白格水平移动至第 c 列</p> <p>void move_v(int M, int N, int puzzle[], int r, vector<char>& sol) 将空白格铅直移动至第 r 行</p> <p>void move(int r, int c, int i, int j, int M, int N, int puzzle[], vector<char>& sol) 把第 r 行 c 列的拼图移动到第 i 行 j 列</p> <p>void solve_1x2(int M, int N, int puzzle[], vector<char>& sol, int i) 求解第 i 行的最右边 1x2 小行块，方法详见第五章</p> <p>void solve_2x1(int M, int N, int puzzle[], vector<char>& sol) 求解第每一列最下方的 2x1 小列块，方法详见第五章</p> <p>long long autosolve(int M, int N, int puzzle[], vector<char>& sol) 自动解 MxN 拼图，同时回传过程累积所用步数</p>

三、 功能实现

本次大作业所要求之基本功能与其他功能条列于下：

- (1) 读图：其机制与代码比较复杂，详见第四章
- (2) 设置难度：利用 `bool readsize(int &M, int &N)` 函数实现，用 cin 读入并回传是否读入两个大于 1 的整数，如非则输出提示信息并再次读入。

注意 如果读入错误，需清除 cin 的缓存区与以确保能够再次读入。

- (3) 随机变换：利用 `std::random_shuffle(puzzle, puzzle + M * N)`；将数组中 $M*N$ 个元素重新打乱

注意 使用之前要 `srand(unsigned(time(NULL)))` 改变随机数的种子

- (4) 可行性分析：扣除空白格，我们将 $M*N$ 拼图视为 $1-(M*N-1)$ 的其中一种全排列，有可能为奇排列或偶排列。解完后的拼图即为 $\{1, 2, \dots, M*N-1\}$ ，逆序数和为 0 为偶排列。而当我们左右移动拼图时，相当于没有做任何变换（因为空白格-1 不算在排列中）；而上下移动拼图时，相当于连续对换了相邻 $N-1$ 个数。而我们又知道排列中任意对换两数会改变其排列，因此要考虑 N 的奇偶性与空白格所在的行数 C 。因此，拼图可解的条件为：

$$\text{拼图可解} \begin{cases} N \text{ 为奇数} \rightarrow \text{拼图为偶排列} \\ N \text{ 为偶数} \begin{cases} (M-1)-C \text{ 为偶数} \rightarrow \text{拼图为偶排列} \\ (M-1)-C \text{ 为奇数} \rightarrow \text{拼图为奇排列} \end{cases} \end{cases}$$

实践方法 利用两层回图计算逆序数和，再依照 N 、 M 、 C 进行判断

思考 我写了复杂度为 $O(M^2N^2)$ 的分析方法， M 、 N 过大时会运行很久（可利用 **归并排序** 的方法再进行优化）

- (5) 开始游玩：写于 run.cpp
- (5.1) 移动：利用已写好的 `find` 函数找到空白格位置，并依照指令判断移动是否合法，再利用 `std::swap` 交换指定拼图
- (5.2) 判断是否复原：从 $0 \sim M*N-2$ 判断 `puzzle[i]` 是否等于 $i+1$
- (5.3) 重新游戏：游戏一开始即利用临时变量 `int origin[M*N]` 记录着打乱后的拼图，并在收到指令后将 `puzzle` 重新赋值为 `origin`
- (5.4) 保存图像：其机制与代码比较复杂，详见第四章
- (5.5) 存档：因为游戏进度只要求在 程序运行期间有效 → 不需要使用文件存储。我们可以建立一个由 string 映射到 storage (storage 结构体含有一个代表拼图的 `vector<int>`) 的哈希表，利用 STL 中的 `map` 即可实现

附加 利用 `map.count()` 判断是否重名并印出是否复盖的提示

附加 暂定存档数量限制为 100 个，超过输出会覆盖旧档的提示

- (5.6) 读档：利用 `map.count()` 判断档案是否存在，接着读档 / 输出错误提示
- (5.7) 自动还原：有两种还原方法，详见第五章
- (5.8) **附加** 利用 `Sleep()`, `system("cls")` 适时实现暂停、清屏的功能
- (5.9) **附加** (指令 `L`) 输出所有档案名：利用 `map::iterator` 遍历所有档案

四、BMP 图档的结构与实现

(注:本章仅适用于 Windows3.0 以上之操作系统,读入 24 位 BMP 图)
参照维基百科,图档的结构为 **BMP 文件头**、**DIB 头** 及 **填充对齐后的像素数组**

1. BMP 文件头:其功用为文件识别

占 14 个 byte(unsigned char 数组实现),第 0 位和第 1 位 B 和 M。第十位为像素数组位置偏 = BMP 文件头大小 + DIB 头大小 = 54

2. DIB 文件头:图像的详细讯息

占 40 个 byte(unsigned char 数组实现),0~3 位为图片大小、4~7 位为图索宽度、8~11 位维图索高度。而本次大作业只预设读入 24 位 BMP 图故其他可以先不考虑。

3. 像素数组

对于每一个像素,用 3 个 byte 储存分别代表 R、G、B 的取值

注意 当宽度不为 4 的倍数再每一行会进行填充(padding)

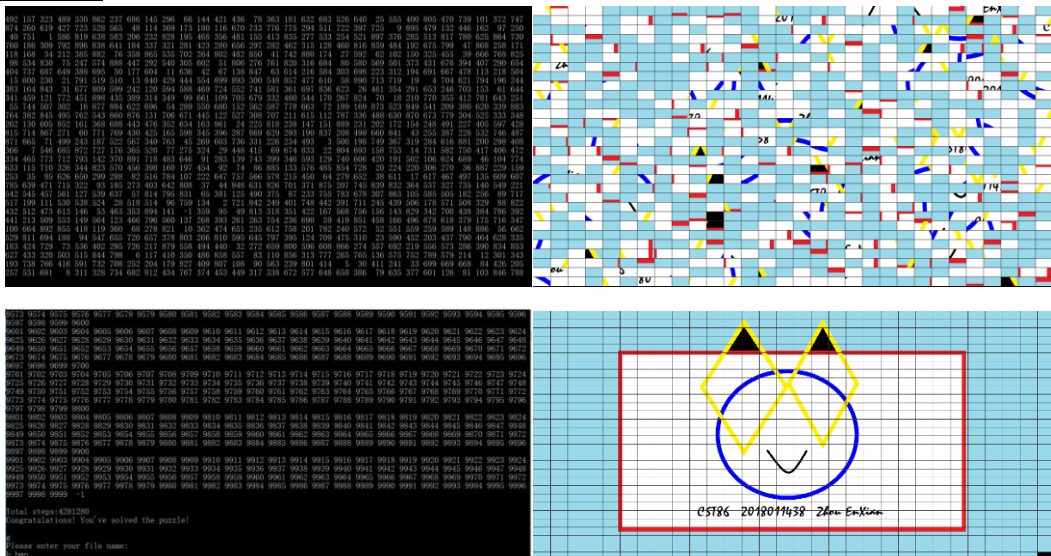
注意 储存时为逆序存入

了解了图档结构即可依照相应拼图写出读入、移动即印出功能了。利用 **fread** 从指定路径读入图片,依照当时拼图打乱图片并加入黑色横竖直线、最后利用 **fout** 在指定路径输出图片。

注意 M、N 过大会出现像素不够分割的情况,此时输出提示讯息

注意 当宽高无法整除时在右下方留黑

成果展示 以下印出随机打乱以及解完后的 30*30 的拼图



(本次大作业使用自制图片 1.bmp 进行测试)

五、 拼图自动还原的实现

5-1. $M \times N$ 拼图的通解

根据 M 、 N 不同的取值来讨论其解法。

(1) $M = N = 2$: 不难看出只要任意顺时针/逆时针绕圈即可解出 \rightarrow **2x2 必可解**

(2) $M = N = 3$: 我们可将拼图拆解成四个部份:

(A) 左上角元素 (B) 行块 (C) 列块 (D) 2x2 方块。依序把四部份排好如下图

1x1 左上角元素 (A)	1x2 行块 (B)
2x1 列块 (C)	2x2 方块 (D)

至于我们要如何排出这四个部份有多种办法, 这里列举我比较常用的方法。

Step 1:

\rightarrow 把 1 移到左上, 2、3 移到右下, 空白格移到右上

1		空白格
		2
		3

Step 2:

\rightarrow S S A W W D S

1	2	3
		空白格

Step 3:

\rightarrow 把 4, 7 移到左下, 空白格移到 1, 4 之间

1	2	3
空白格		
4	7	

Step 4:

\rightarrow S D

1	2	3
4		
7	空白格	

\rightarrow 右下角为 2x2 拼图, 必可解 \rightarrow **3x3 必可解**

(3) M 、 N 为任意数: 扩充 3 x 3 拼图的思路至 $M \times N$

1. 从第 0 行开始, 依次往下解每一行, 直到解出前 $M-2$ 行。

1-1 想成功解出第 i 行, 可先将第 i 行的前 $N-3$ 块拼图填满。


利用 3x3 拼图求解左上角元素的方法。(A 部份) function: move()

1	2		N
(前 i-1 行已经拼好)							
i*N+1	i*N+2	i*N+N-3	3 x 3 小拼图块		

1-2 解第 i 行剩下的 3 块 **function: void_solve_1x2()**

可发现，右边形成了一个 3 x 3 的小拼图块。

因此，利用 3 x 3 拼图解行块的解法即可成功解出第 i 行 (B 部份)

i*N+N-2		空白格		i*N+N-2	i*N+N-1	(i+1)*N
		i*N+N-1	S S A W W D S			空白格
		(i+1)*N				

* 注意特例，两者颠倒时，利用指定路径求解

	空白格	(i+1)*N
		i*N+N-1

通解: $D \rightarrow S \rightarrow A \rightarrow W \rightarrow D \rightarrow S \rightarrow S \rightarrow A \rightarrow W$
 $\rightarrow W \rightarrow D \rightarrow S \rightarrow S \rightarrow A \rightarrow W \rightarrow D \rightarrow S \rightarrow A$

2. 最后两行切割为 N 个 2*1 个编号为 k 的小块，对于每一编号 k 的小块，也利用 3 x 3 拼图列块的解法解出 **function: void_solve_2x1()**

空白格		S D	(M-2)*N+k	
(M-2)*N+k	(M-1)*N+k		(M-1)*N+k	空白格

* 注意特例,当两者颠倒时,透过指定的路径求解

(M-1)*N+k	空白格	或者是	空白格	(M-1)*N+k
(M-2)*N+k			(M-2)*N+k	

通解: $(A) \rightarrow S \rightarrow D \rightarrow D \rightarrow W \rightarrow A \rightarrow S \rightarrow A \rightarrow W \rightarrow D \rightarrow S$
 $\rightarrow D \rightarrow W \rightarrow A \rightarrow S \rightarrow D \rightarrow W \rightarrow A \rightarrow A \rightarrow S \rightarrow D$

3. 最后剩下 2x2 方块，2x2 必可解 \rightarrow MxN 必可解

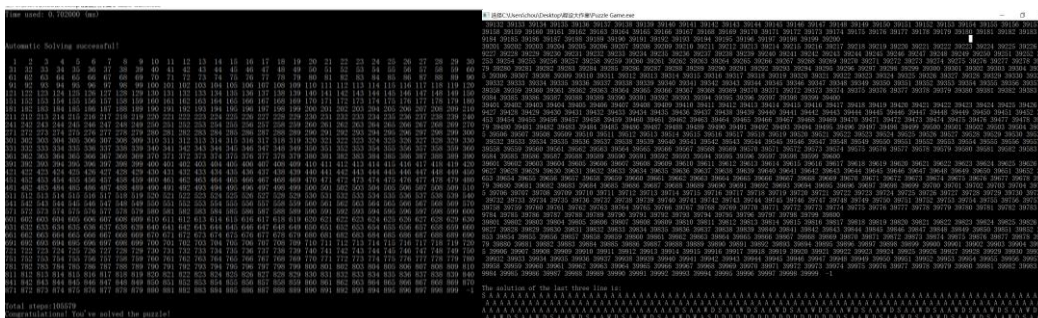
注意事项与讨论

1. 注意求解后面的小块时不可移动到前面已拼好的部份，本次大作业中，采取向 " 下 " 向 " 右 " 绕圈的方式（才不会移动到左上角已拼好的小块）。
2. 注意特殊情况。经由多次测试后，本次作业补充多个特判情形，盼能够将特殊情况都排除，确保一定能够完整解出拼图。
3. 经实测，目前最大能解出 300 x 300 的拼图（大概需要几小时的时间）
4. 此方法求解 MxN 拼图的概念是：从 1 开始，各个击破。不求最短路，只要能解出来，不论多少步都是好方法！
5. 对于 NxN 拼图，经试验后发现运行时间与步数大概和 N^4 成正比

成果展示 (按每行输出解法, 最后输出耗费时间、拼好的拼图与所用步数)

```
1 2 3 4 5
12 14 13 15 -1
21 24 8 11 20
19 7 17 22 9
18 6 16 23 10

The solution of row #1 is:
DDSAAWDSAAWDWASDSSSDWASDWWASWWSDWASSSDWASDWWASDSDWASDSASD
DWAWDSSAASSAWDDSAWDDSAWDSAAWWDSSAWWDSSA
```



30*30 拼图在 0.7 秒内
解完, 若完全清除输出
语句, 可达 0.2 秒!

成功解出 200*200 拼图
自动还原的过程耗费了
大概 20 分钟

5-2. 优化路径-BFS 解 3 × 3 拼图

在第 15 周王老师教完深搜、宽搜后, 突然想到拼图游戏或许也可以利用此方法。而因为拼图移动是一步一步, 在合法的情况下每一步能往上下左右走, 符合 BFS 一层一层的概念。如此就能大幅减少移动次数而自动解完拼图了。

注意事项与讨论

1. 搜索就是依照条件”剪枝后的枚举”。本次大作业中, 实作 Node* 的建构式判断移动是否合法进而剪枝。同时, 利用 STL 的 set<vector<int>> 储存已走过的拼图已去除重复而剪枝。
2. 要注意回溯的写法, 注意队列中指针的写法
3. 注意输出顺序 (Queue 的特点, First In First Out)

```
成果展示

6 8 3 Step 4: W Step 8: D Step 12: S Step 16: W Step 20: W Step 24: D
4 -1 5 6 8 -1 4 6 8 4 3 6 4 3 -1 -1 3 1 2 3
7 2 1 4 2 3 2 -1 3 2 -1 8 2 1 6 4 2 6 4 5 6
7 1 5 7 1 5 7 1 5 7 1 5 7 5 8 7 5 8 7 8 -1

Solving...
Step 1: S Step 5: A Step 9: D Step 13: S Step 17: A Step 21: D
6 8 3 4 2 3 4 6 8 4 3 6 4 3 1 -1 3 1 -1 3
4 2 5 7 1 5 7 1 5 7 1 5 7 5 8 7 5 8 7 5 8
7 -1 1 7 1 5 7 1 5 7 1 5 7 5 8 7 5 8 7 5 8

Step 2: D Step 6: A Step 10: W Step 14: D Step 18: S Step 22: S
6 8 3 -1 6 8 4 6 -1 4 3 6 4 1 3 1 2 3
4 2 5 4 2 3 2 3 8 2 1 8 2 -1 6 4 -1 6
7 1 -1 7 1 5 7 1 5 7 5 8 -1 7 5 8 7 5 8

Step 3: W Step 7: S Step 11: A Step 15: W Step 19: A Step 23: S
6 8 3 4 6 8 4 -1 6 4 3 6 4 1 3 1 2 3
4 2 -1 -1 2 3 2 3 8 2 1 -1 -1 2 6 4 5 6
7 1 5 7 1 5 7 1 5 7 5 8 7 5 8 7 5 8 7 -1 8

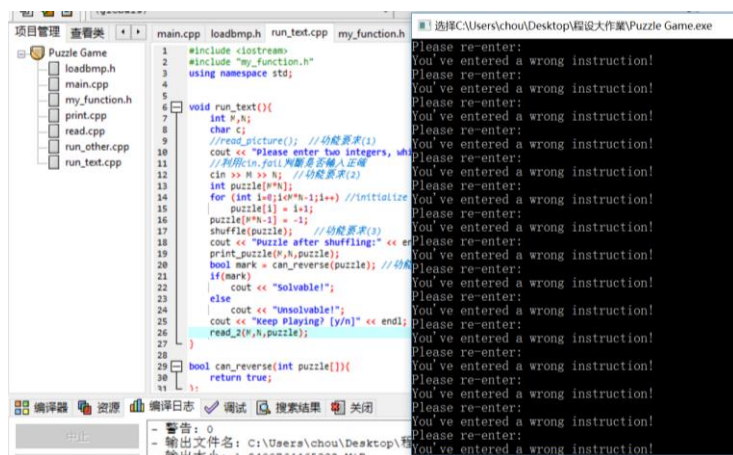
Time used: 1.465
The solution is: S D W W A A S D D W A S S D W W A S A W D S S D
Total steps: 24
Congratulations! You've solved the puzzle!
```

平均解法大概都在 15~20 步上下, 耗费时间 1 秒左右

六、(附加)其他讨论

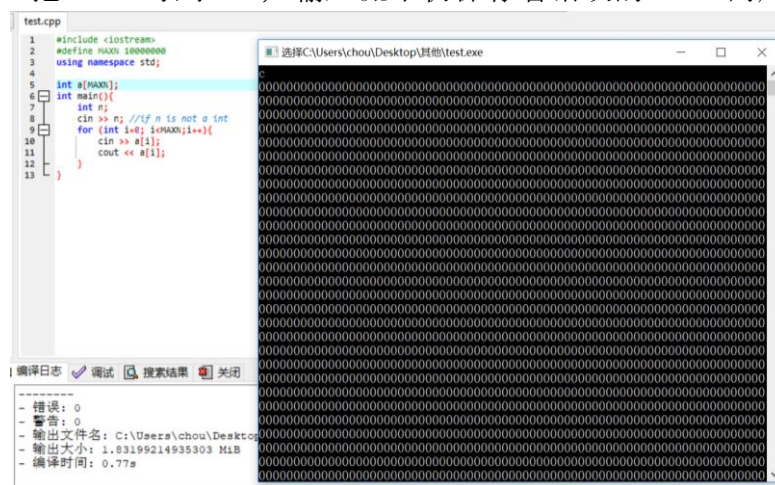
6-1.大作业中出现的问题与调试过程

(1) 未清空输入流的缓存区：当输入数据类型不符合，程序陷入死循环



推测可能原因：没有把 cin 的缓存区以及错误状态 flag 清空，因此下次再重新呼叫函数遇到 cin 时就无法进行输入操作。

实验测试：把 char 写到 int，输入流中仍保存着错误的 char 而产生错误



解决方法：当没有正确读入时，除了输出提示信息外，更重要的是要加入 `cin.clear();` `cin.clear();` 两语句，其中 `cin.clear();` 能够清空错误信息，而 `cin.sync();` 清空缓冲区，如此可解决因未清空输入流缓冲区而使程序无法运行的 bug。

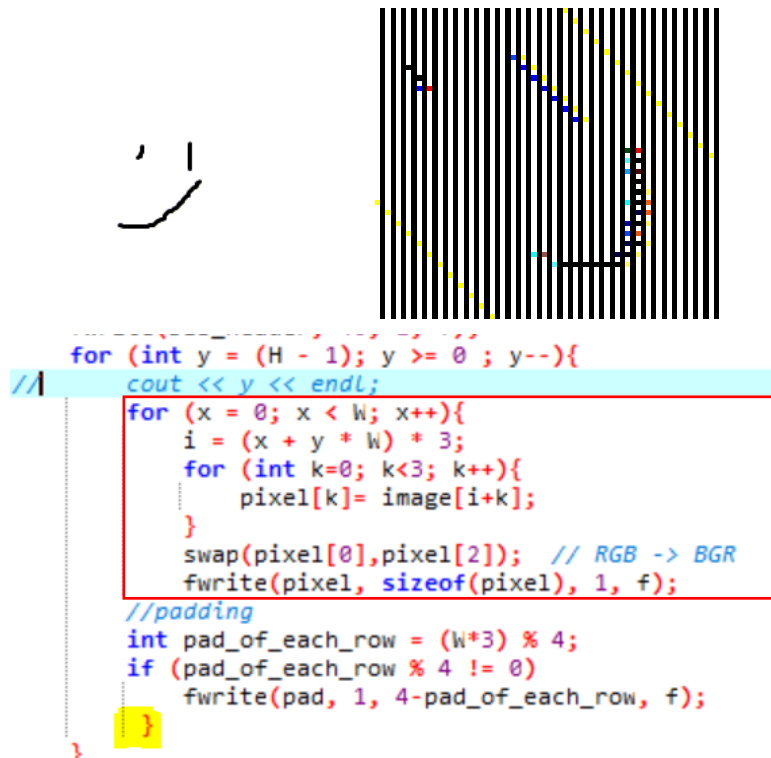
(2) 关于 unsigned 的问题

读图程序中的一段代码：`unsigned int y; for(y = H - 1; y ≥ 0; y --)`

因为 y 是 unsigned，恒正 → 死循环

解决方法：少用 unsigned (即使读图中的长、宽、高像素值都是正的)，注意终止条件

(3) 括号匹配问题：读图程序出现的问题以及错误代码如下所示。



标记的大括号应该写于红色框之后，应该是每读一行才读 padding（并非每读一格都读 padding）

解决方法：思路一定要理清楚。注意每模块代码所代表的实际意义

(4) 位深度不符：图片存成了 256 色位图（8 位图），但写的是 24 位图程序。因为我写的读图程序只判断档案是否存在而能正确 fread，但没有确定使用者提供的 BMP 图档是否为 24 位图就直接执行了

图像	
分辨率	215 x 222
宽度	215 像素
高度	222 像素
位深度	8

解决方法：输入提示，提醒使用者提供的图片一定要是 **24 位的 BMP 图** 或者是可以利用读图函数的返回值（把 bool 改成 int）定义不同的错误类型进行特判。

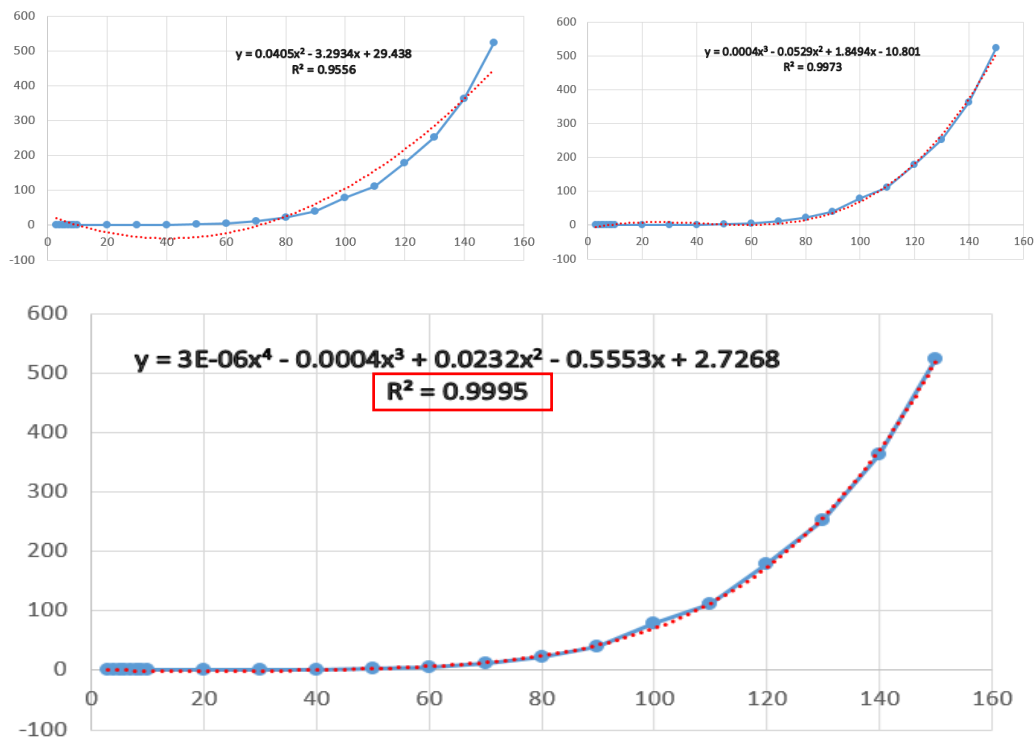
思考 可先判断档案路径的“后缀名”，如果后缀名不符合输出提示信息；同时，如果无后缀名，则先再路径最后加上.bmp 再进行 fread

思考 也可以利用 try-catch 语句进行错误特判

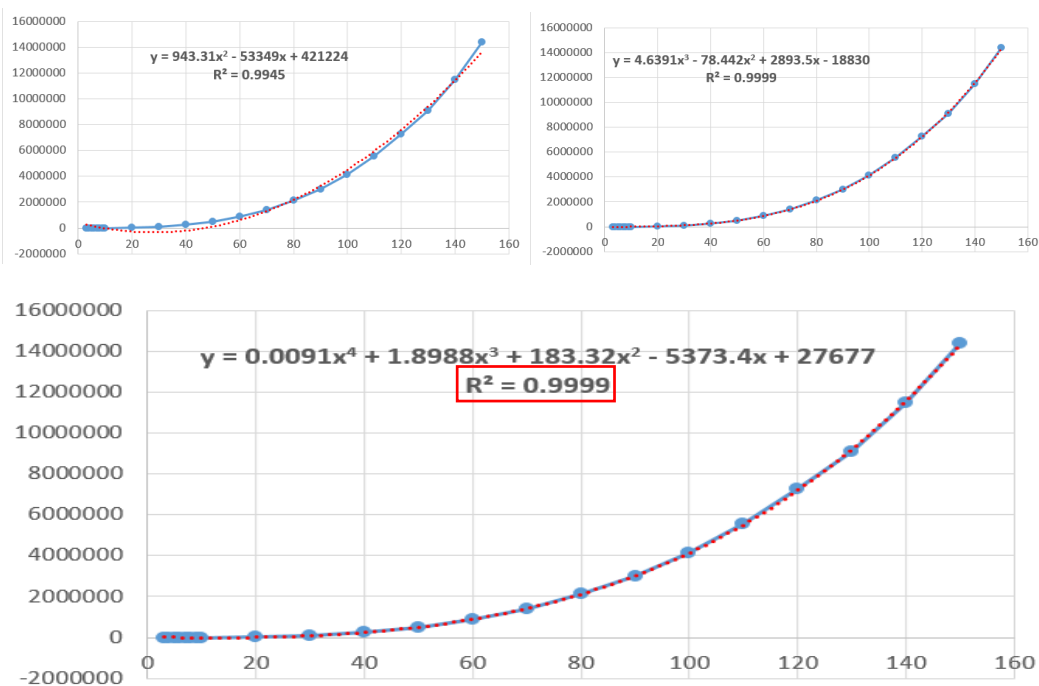
6-2. 自动还原算法复杂度估算

为了探讨估算自动解的算法复杂度，我们将自动解程序中的输出语句全部删除，并对不同大小的 $N \times N$ 拼图统计其运行时间和求解步数，再利用 Excel 进行趋势线拟和判断（数据存于 N-Time-Step 关系表.xlsx）

※时间-N 关系表



※求解步数-N 关系表



最优拟和为四次多项式，算法复杂度约为 $O(N^4) = O(N^{2^2}) = O(\text{size}^2)$

复杂度分析

自动还原的思路可写成以下的伪代码

For $i := 0$ to $\text{size} - 2$ do.....总 $M*N-1$ 个拼图

把 $\text{puzzle}[i]$ 移动至 $i + 1$ 的位置.....最多交换 $M+N$ 格

而交换的思路又如下

For $i := 0$ to $\text{size} - 2$ do

移动空白格至目标拼图旁边并交换.....最多查找 $M*N$ 格

因此复杂度约为 $O((MN - 1) * (M + N) * (MN)) \approx O(\text{size}^2 * (M + N))$

然而，因为交换的步数会随着拼图一块一块的完成而减少，故实际复杂度会再降低一些。