一. 实验概述

(1) 实验目的

熟悉和掌握启发式搜索的定义、估价函数和算法过程，并利用 A*算法求解 8 数码难题，理解求解流程和搜索顺序。

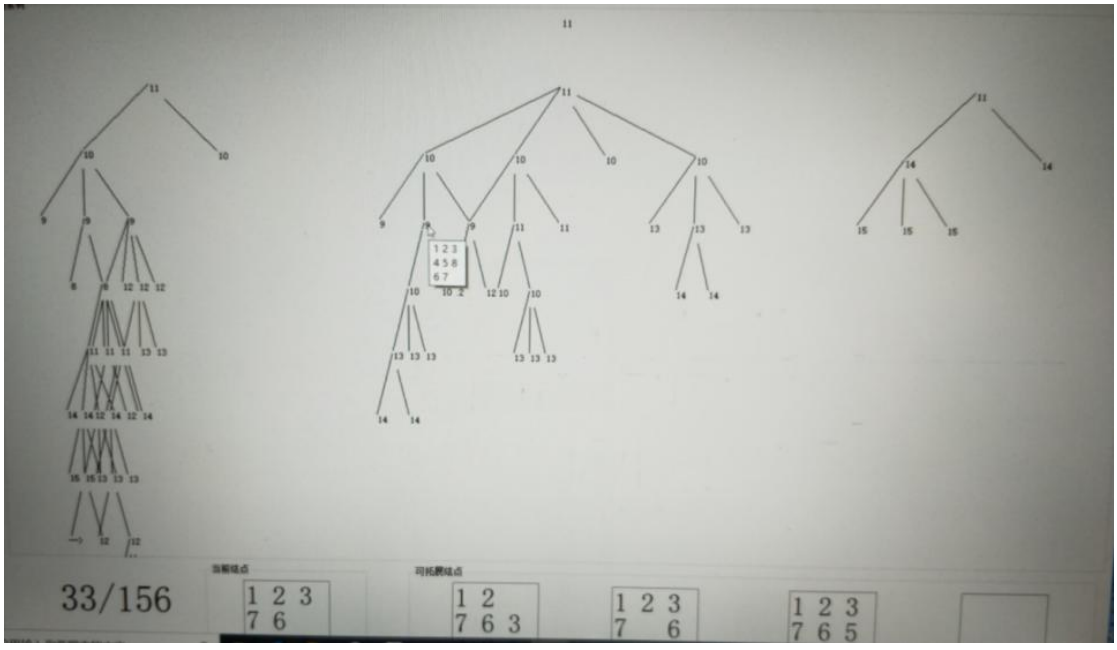
(2) 实验内容

1.以 8 数码问题为例实现 A*算法的求解程序（编程语言不限），要求设计两种不同的估价函数。

2.设置相同初始状态和目标状态，针对不同的估价函数，求得问题的解，并比较它们对搜索算法性能的影响，包括扩展节点数、生成节点数和运行时间。画出不同启发函数 h(n)求解 8 数码问题的结果比较表，进行性能分析。

3.要求界面显示初始状态，目标状态和中间搜索步骤。

4.画出图示所示的搜索生成的树，在每个节点显示对应节点的 f*(n)值，以显示搜索过程，以红色标注出最终结果所选用的路线。



二. 实验方案设计

(1) 总体设计思路与总体架构

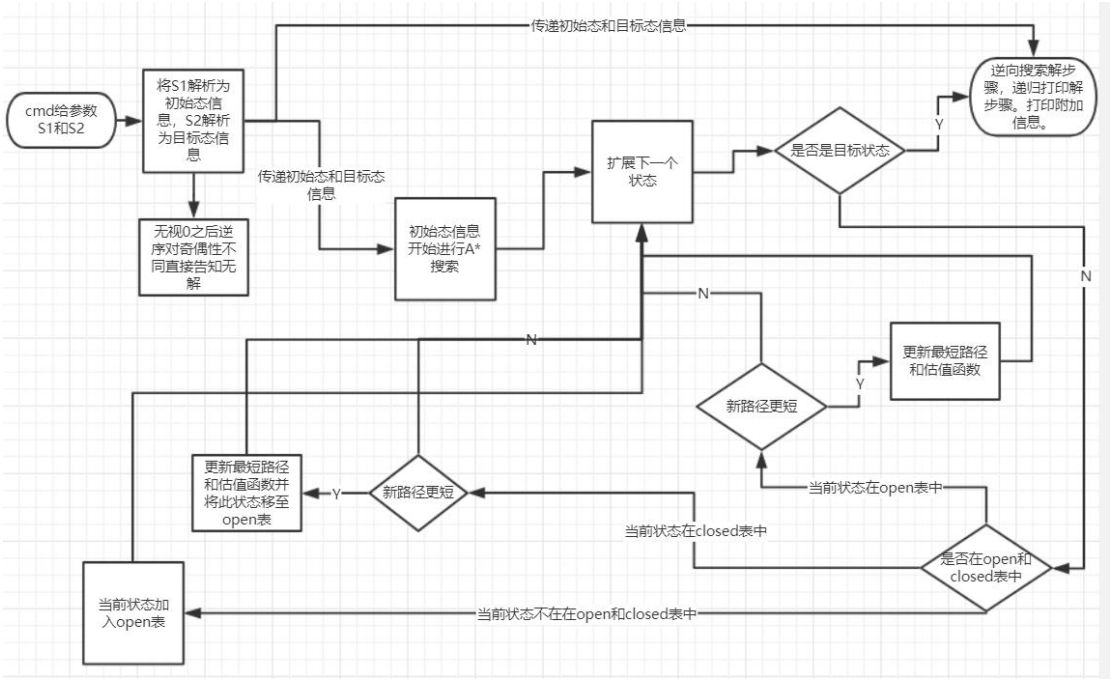
总体设计框架分为三大部分：

①输入及初始化部分，通过输入字符串化的初始态和目标态参数运行 8shuma.exe，对接收到的参数做一些预处理和变量初始化。

②搜索部分，接收经处理之后的初始态和目标态信息，从初始态开始按照 A*搜索的方法搜索到达目标态的最少步数，并且对每个中间状态做记录。

③打印解部分，从目标态开始，利用搜索部分为每个中间态做的记录信息，逆向搜索最短路径，递归打印出移动步骤，并且打印生成节点数目、扩展节点数目、用时、每一步的 f(n)等附加信息。

流程图如下：



(2) 核心算法及基本原理：

A*搜索算法：

设计评估函数 f(n)=g(n)+h(n)，g(n)=到达结点 n 时已经花费的代价，h(n)=从结点 n 到目标结点的最小代价的估计值，f(n)=经过结点 n 的最小代价的估计值。

每次优先扩展 f(n)最小的结点，并对由此结点 n 生成的结点 x,判断 x 是否在 open 表和 closed 表中(其中 open 表是尚未扩展的结点，closed 表是已完成扩展的结点)，若既不在 open 表也不再 closed 表中，则将该结点加入到 open 表中；若在 open 表中则若是沿着一条更短的路径到达该结点，则更新该结点的 g(n)和 f(n)；若在 closed 表中，则若是沿着一条更短的路径到达该结点，则更新该结点的 g(n)和 f(n)，并且将该结点移入

open 表中。当扩展结点为目标结点时，搜索结束。

当 h(n)<=h*(n)时(h*(n)为结点 n 到目标结点的真实最小代价)，称 h(n)为可采纳的，此时 A*搜索树搜索版本是最优的，即一定能求得起始结点到目标结点的最小代价。

两种 h(n)的设计：

①$h_1(n)$=不在位的棋子数，这种启发式函数通过将原问题松弛为任意两个棋子可以交换位置而得到。

②$h_2(n)$=所有棋子到其目标位置的曼哈顿距离，这种启发式函数通过将原问题松弛为相邻棋子可以交换位置而得到。

(3) 模块设计：本实验的具体模块设计

①预处理模块：

将所输入的起始状态和目标状态的字符串解析为棋盘局面。事先记录下目标状态的各棋子的横纵位置，便于在需要的时候快速计算出 h(n)。将初始状态放入 open 表中为开始 A*搜索过程做准备。

②搜索模块：

每次取出 open 表中的 f(n)最小的结点进行扩展，按照上述 A*搜索算法生成新结点、更新估值函数，调整 open 表和 closed 表。统计生成结点和扩展结点数量等信息。并且每次生成或者更新一个结点时，需要记录这个结点的最短路径上的前驱结点，便于最终逆向递归打印解。

③open 和 closed 表模块：

以各个状态的 f(n)为关键值建立小根堆，小根堆记录了 open 表和 closed 表，关键值 f(n)不等于 MAX_INT 的元素处于 open 表，关键值 f(n)等于 MAX_INT 的元素处于 closed 表。每次取 open 表中 f(n)最小的结点，相当于从小根堆中弹出堆顶元素，之后调整维护小根堆。扩展完一个结点 n 后，维持 g(n)不变，将 f(n)强置为 MAX_INT，标记为 closed 表元素，同时也使得其不会从小根堆中被弹出。将 closed 表中元素移入 open 表的操作，在更新 f(n)时就会自然地将 f(n)有 MAX_INT 更新为一个非 MAX_INT 值，从而相当于从 closed 表移入了 open 表。

④hash 模块：

在生成节点时，得到的是棋盘局面信息，而并非直接得到某个结点。如果扫描一遍整个堆来确定棋盘局面和结点的对应关系则会大大增加时间复杂度，因此需要通过 hash 的方式将棋盘局面信息映射到 open 表或者 closed 表中的结点，或者映射到既不在 open 表中也不在 closed 表中的结点。

⑤后处理模块：

在扩展完目标结点后，从目标结点开始，利用搜索模块中记录的前驱结点的信息不断地递归打印出最短路径，并且打印出各个结点的棋盘局面和对应的 f(n)值、步数等信息。统计程序运行时间。

(4) 其他创新内容或优化算法

状态压缩 hash：

要将棋盘局面信息映射到 open 表和 closed 表，就需要用数组元素下标和数组元素值来构建映射关系实现随机读取。如果直接将棋盘局面的字符串转化为数值来作为下标，则最大需要 10^9 长度的数组来存储 9 位数字。实际上每次可能出现的局面总数不会超过 9!/2=181440 个，用 10^9 量级长度的数组去存储造成了大量的空间浪费。注意到，只需要确定 8 个位置的数字值就可以确定唯一棋盘局面，只需要 7 个位置的数字值就可以确定唯二棋盘局面。存储 7 个位置的数字只需要 10^7 量级长度的数组，大大节省了空间，并且在可承受范围内。考虑到 7 个位置数字确定的是唯二的局面，因此存在 hash 冲突，会将两个棋盘局面映射到数组的同一位置。采用后移 10^7 的方法来解决 hash 冲突。当一个局面被映射到数组中位置 x 时若该位置已被映射，则重新映射到 $x+10^7$ 的位置上，此时一定不会再 hash 冲突。仅仅是加倍了数组长度，达到 $2*10^7$，依然在可承受范围内。如此一来就实现了 O(1)时间的随机读取，将棋盘局面映射到结点。

三. 实验过程

(1) 环境说明：

本实验使用 Windows 10 操作系统，使用 C++语言在 Visual Studio 2019 (v142)平台上开发，核心使用库是 iostream、iomanip、stdlib、cstring、ctime、cmath 等。

⑵ 源代码文件清单，主要函数清单

main.cpp：程序源文件，实现了上述所有模块和算法。

8.bat：批处理程序，用于为 eightshuma.exe 提供参数并将运行结果重定向到不同文件比较影响程序运行速度的因素(如启发式函数等)

int find(const char st[]),用于获取已有状态在 hash 表中的位置

int hn1(const char st[]),用于计算第一种 h(n)

int hn2(const char st[]),用于计算第二种 h(n)

void push(int x),新元素入堆后，用以维护堆的上移更新操作，同时更新 hash 表

void maintain(int x),堆顶元素弹出后，用以维护堆的下移更新操作，同时更新 hash 表

int cal(const char st[]),统计状态对应字符串去除 0 以后的逆序对个数，返回奇偶性，用来方便检查是否有解

void check(const char st[]),检查初始状态能否到达目标状态，不能则输出"无解"

void print1(const node t),打印搜索成功的最短路径

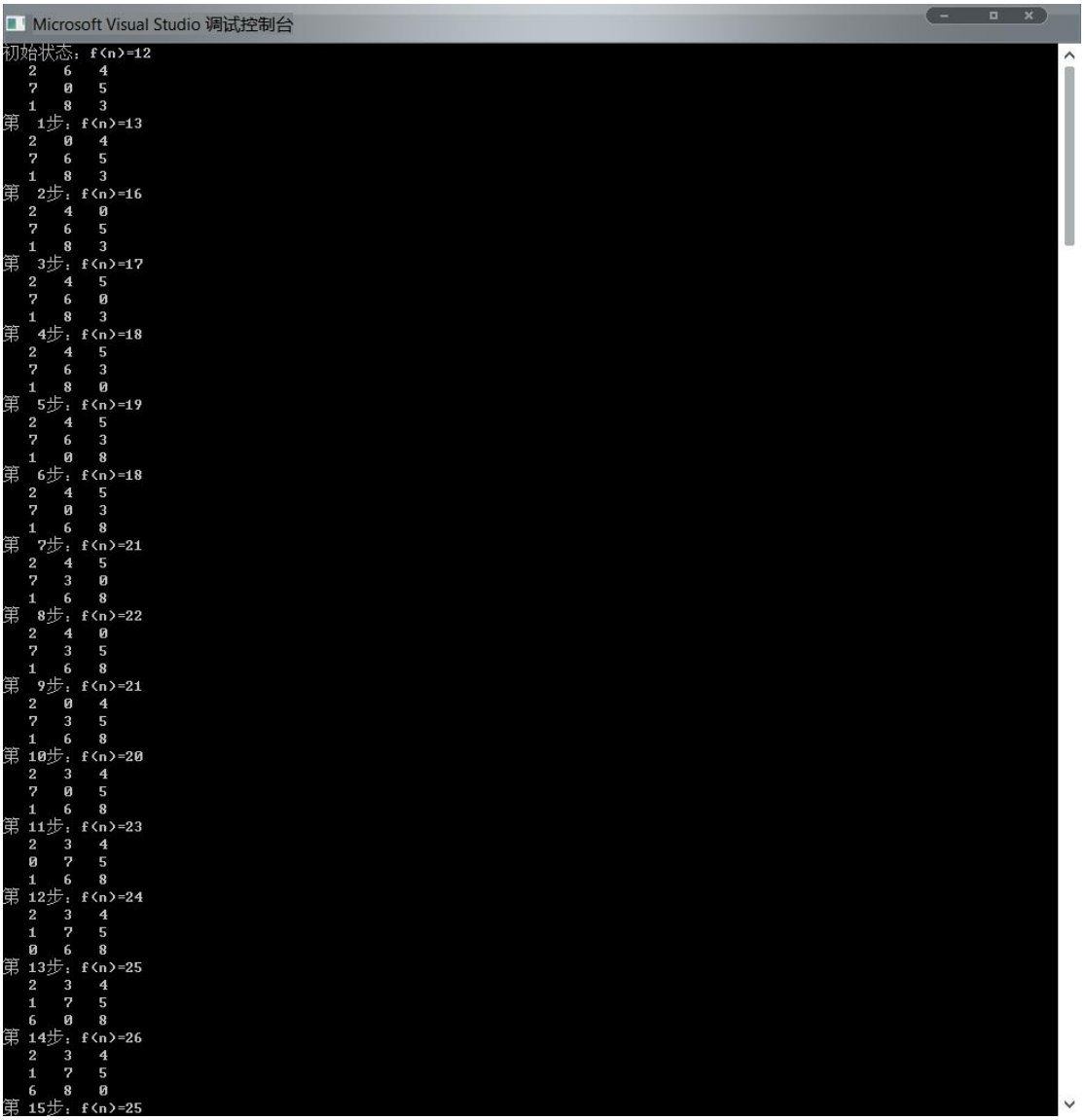
void print2(const int tim),打印扩展节点数量、生成节点数量、用时等信息

int main(int ac, char** ag),通过带参数的 main 函数，在 cmd 命令行下运行。主要实现了输入的初始化以及 A*搜索的主体过程

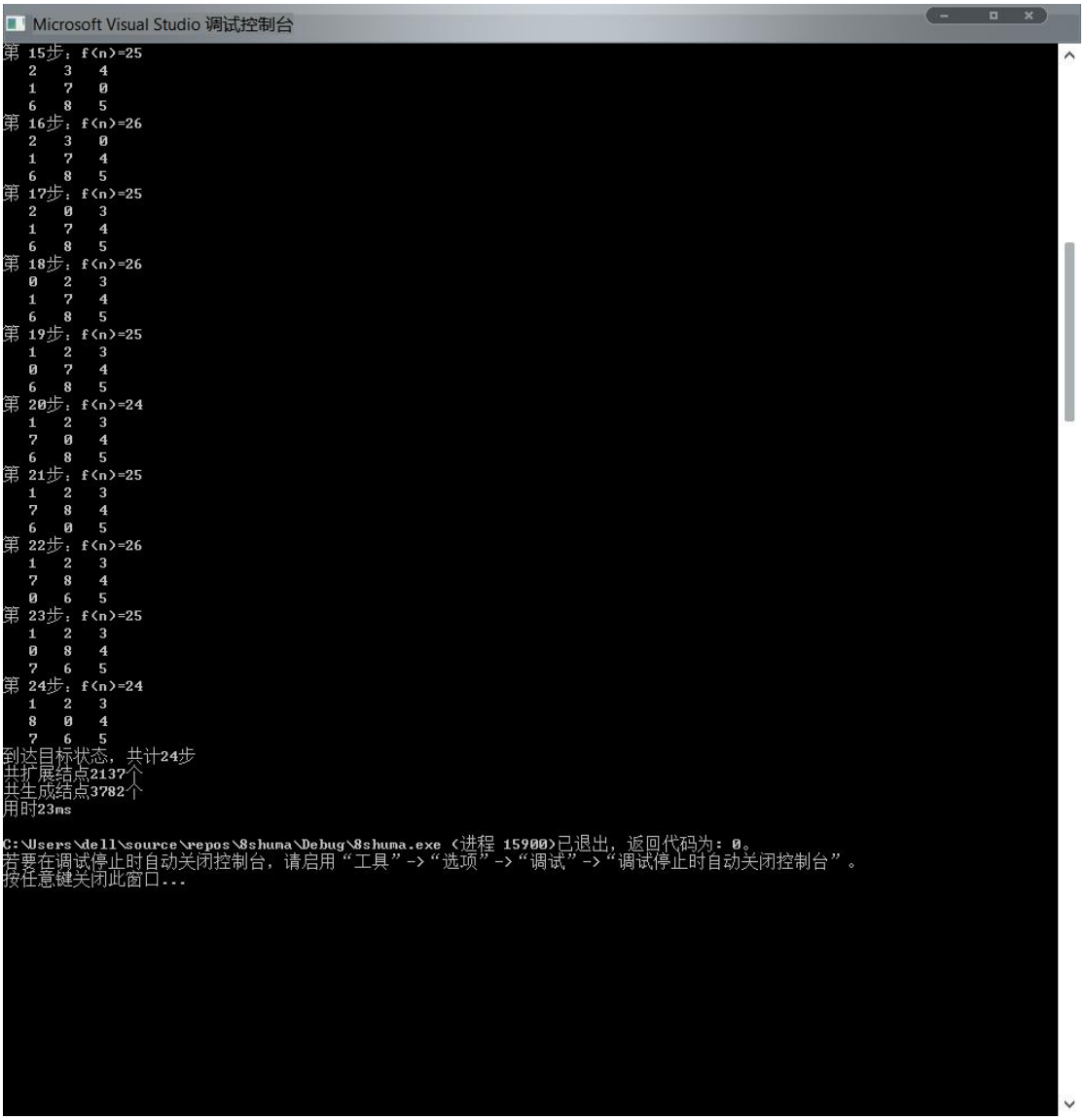
⑶ 实验结果展示

Visual Studio 2019 (v142)集成环境用控制台运行的结果

初始态 264705183，目标态 123804765：



初始态：264705183 f(n)=12 子节点 204765183 264075183 264750183 264785103
第 1步：204765183 f(n)=13 子节点 024765183 240765183 264705183
第 2步：240765183 f(n)=16 子节点 204765183 245760183
第 3步：245760183 f(n)=17 子节点 240765183 245706183 245763180
第 4步：245763180 f(n)=18 子节点 245760183 245763108
第 5步：245763108 f(n)=19 子节点 245703168 245763018 245763180
第 6步：245703168 f(n)=18 子节点 205743168 245073168 245730168 245763108
第 7步：245730168 f(n)=21 子节点 240735168 245703168 245738160
第 8步：240735168 f(n)=22 子节点 204735168 245730168
第 9步：204735168 f(n)=21 子节点 024735168 240735168 234705168
第 10步：234705168 f(n)=20 子节点 204735168 234075168 234750168 234765108
第 11步：234075168 f(n)=23 子节点 034275168 234705168 234175068
第 12步：234175068 f(n)=24 子节点 234075168 234175608
第 13步：234175608 f(n)=25 子节点 234105678 234175068 234175680
第 14步：234175680 f(n)=26 子节点 234170685 234175608
第 15步：234170685 f(n)=25 子节点 230174685 234107685 234175680
第 16步：230174685 f(n)=26 子节点 203174685 234170685
第 17步：203174685 f(n)=25 子节点 023174685 230174685 273104685
第 18步：023174685 f(n)=26 子节点 203174685 123074685
第 19步：123074685 f(n)=25 子节点 023174685 123704685 123674085
第 20步：123704685 f(n)=24 子节点 103724685 123074685 123740685 123784605
第 21步：123784605 f(n)=25 子节点 123704685 123784065 123784650
第 22步：123784065 f(n)=26 子节点 123084765 123784605
第 23步：123084765 f(n)=25 子节点 023184765 123804765 123784065
第 24步：123804765 f(n)=24



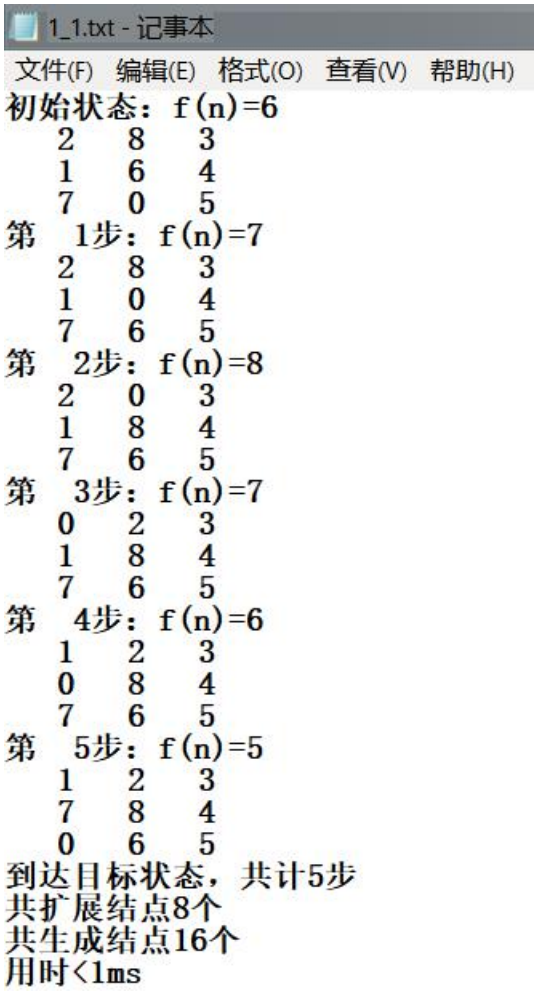
ans.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

初始态：264705183 f(n)=12 子节点	204765183 f(n)=13	264075183 f(n)=15	264750183 f(n)=15	264785103 f(n)=13
第 1步：204765183 f(n)=13 子节点	024765183 f(n)=14	240765183 f(n)=16	264705183 f(n)=14	
第 2步：240765183 f(n)=16 子节点	204765183 f(n)=15	245760183 f(n)=17		
第 3步：245760183 f(n)=17 子节点	240765183 f(n)=18	245706183 f(n)=18	245763180 f(n)=18	
第 4步：245763180 f(n)=18 子节点	245760183 f(n)=19	245763108 f(n)=19		
第 5步：245763108 f(n)=19 子节点	245703168 f(n)=18	245763018 f(n)=22	245763180 f(n)=20	
第 6步：245703168 f(n)=18 子节点	205743168 f(n)=19	245073168 f(n)=21	245730168 f(n)=21	245763108 f(n)=21
第 7步：245730168 f(n)=21 子节点	240735168 f(n)=22	245703168 f(n)=20	245738160 f(n)=22	
第 8步：240735168 f(n)=22 子节点	204735168 f(n)=21	245730168 f(n)=23		
第 9步：204735168 f(n)=21 子节点	024735168 f(n)=22	240735168 f(n)=24	234705168 f(n)=20	
第 10步：234705168 f(n)=20 子节点	204735168 f(n)=23	234075168 f(n)=23	234750168 f(n)=23	234765108 f(n)=23
第 11步：234075168 f(n)=23 子节点	034275168 f(n)=26	234705168 f(n)=22	234175068 f(n)=24	
第 12步：234175068 f(n)=24 子节点	234075168 f(n)=25	234175608 f(n)=25		
第 13步：234175608 f(n)=25 子节点	234105678 f(n)=24	234175068 f(n)=26	234175680 f(n)=26	
第 14步：234175680 f(n)=26 子节点	234170685 f(n)=25	234175608 f(n)=27		
第 15步：234170685 f(n)=25 子节点	230174685 f(n)=26	234107685 f(n)=26	234175680 f(n)=28	
第 16步：230174685 f(n)=26 子节点	203174685 f(n)=25	234170685 f(n)=27		
第 17步：203174685 f(n)=25 子节点	023174685 f(n)=26	230174685 f(n)=28	273104685 f(n)=26	
第 18步：023174685 f(n)=26 子节点	203174685 f(n)=27	123074685 f(n)=25		
第 19步：123074685 f(n)=25 子节点	023174685 f(n)=28	123704685 f(n)=24	123674085 f(n)=28	
第 20步：123704685 f(n)=24 子节点	103724685 f(n)=27	123074685 f(n)=27	123740685 f(n)=27	123784605 f(n)=25
第 21步：123784605 f(n)=25 子节点	123704685 f(n)=26	123784065 f(n)=26	123784650 f(n)=28	
第 22步：123784065 f(n)=26 子节点	123084765 f(n)=25	123784605 f(n)=27		
第 23步：123084765 f(n)=25 子节点	023184765 f(n)=28	123804765 f(n)=24	123784065 f(n)=28	
第 24步：123804765 f(n)=24				

cmd 下运行重定向到文本文件的部分结果

起始态 283164705 目标态 123784065:



初始态：283164705 f(n)=6	子节点	283104765	283164075	283164750	
第 1步：283104765 f(n)=7	子节点	203184765	283014765	283140765	283164705
第 2步：203184765 f(n)=8	子节点	023184765	230184765	283104765	
第 3步：023184765 f(n)=7	子节点	203184765	123084765		
第 4步：123084765 f(n)=6	子节点	023184765	123804765	123784065	
第 5步：123784065 f(n)=5					

起始态 **603712458** 目标态 **123784065**:

1_2.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
初始状态：f(n)=16
    6    0    3
    7    1    2
    4    5    8
第  1步：f(n)=15
    6    1    3
    7    0    2
    4    5    8
第  2步：f(n)=16
    6    1    3
    7    5    2
    4    0    8
第  3步：f(n)=15
    6    1    3
    7    5    2
    0    4    8
第  4步：f(n)=18
    6    1    3
    0    5    2
    7    4    8
第  5步：f(n)=21
    6    1    3
    5    0    2
    7    4    8
第  6步：f(n)=22
    6    1    3
    5    2    0
    7    4    8
第  7步：f(n)=21
    6    1    3
    5    2    8
    7    4    0
第  8步：f(n)=20
    6    1    3
    5    2    8
    7    0    4
第  9步：f(n)=21
    6    1    3
    5    2    8
    0    7    4
第 10步：f(n)=22
    6    1    3
    0    2    8
    5    7    4
第 11步：f(n)=23
    0    1    3
    6    2    8
    5    7    4
第 12步：f(n)=24
    1    0    3
    6    2    8
    5    7    4
第 13步：f(n)=23
    1    2    3
    6    0    8
    5    7    4
第 14步：f(n)=22
    1    2    3
    6    7    8
    5    0    4
第 15步：f(n)=21
    1    2    3
    6    7    8
    0    5    4
第 16步：f(n)=22
    1    2    3
    0    7    8
    6    5    4
第 17步：f(n)=23
    1    2    3
    7    0    8
    6    5    4
第 18步：f(n)=24
    1    2    3
    7    8    0
    6    5    4
第 19步：f(n)=23
    1    2    3
    7    8    4
    6    5    0
第 20步：f(n)=22
    1    2    3
    7    8    4
    6    0    5
第 21步：f(n)=21
    1    2    3
    7    8    4
    0    6    5
到达目标状态，共计21步
共扩展结点927个
共生成结点1499个
用时9ms
```

初始态：603712458 f(n)=16	子节点	063712458	630712458	613702458	
第 1步：613702458 f(n)=15	子节点	603712458	613072458	613720458	613752408
第 2步：613752408 f(n)=16	子节点	613702458	613752048	613752480	
第 3步：613752048 f(n)=15	子节点	613052748	613752408		
第 4步：613052748 f(n)=18	子节点	013652748	613502748	613752048	
第 5步：613502748 f(n)=21	子节点	603512748	613052748	613520748	613542708
第 6步：613520748 f(n)=22	子节点	610523748	613502748	613528740	
第 7步：613528740 f(n)=21	子节点	613520748	613528704		
第 8步：613528704 f(n)=20	子节点	613508724	613528074	613528740	
第 9步：613528074 f(n)=21	子节点	613028574	613528704		
第 10步：613028574 f(n)=22	子节点	013628574	613208574	613528074	
第 11步：013628574 f(n)=23	子节点	103628574	613028574		
第 12步：103628574 f(n)=24	子节点	013628574	130628574	123608574	
第 13步：123608574 f(n)=23	子节点	103628574	123068574	123680574	123678504
第 14步：123678504 f(n)=22	子节点	123608574	123678054	123678540	
第 15步：123678054 f(n)=21	子节点	123078654	123678504		
第 16步：123078654 f(n)=22	子节点	023178654	123708654	123678054	
第 17步：123708654 f(n)=23	子节点	103728654	123078654	123780654	123758604
第 18步：123780654 f(n)=24	子节点	120783654	123708654	123784650	
第 19步：123784650 f(n)=23	子节点	123780654	123784605		
第 20步：123784605 f(n)=22	子节点	123704685	123784065	123784650	
第 21步：123784065 f(n)=21					

起始态 273645801 目标态 603712458:

1_10.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
初始状态：f(n)=16
   2   7   3
   6   4   5
   8   0   1
第  1步：f(n)=15
   2   7   3
   6   0   5
   8   4   1
第  2步：f(n)=18
   2   7   3
   0   6   5
   8   4   1
第  3步：f(n)=17
   0   7   3
   2   6   5
   8   4   1
第  4步：f(n)=16
   7   0   3
   2   6   5
   8   4   1
第  5步：f(n)=17
   7   6   3
   2   0   5
   8   4   1
第  6步：f(n)=18
   7   6   3
   0   2   5
   8   4   1
第  7步：f(n)=21
   7   6   3
   8   2   5
   0   4   1
第  8步：f(n)=20
   7   6   3
   8   2   5
   4   0   1
第  9步：f(n)=21
   7   6   3
   8   2   5
   4   1   0
第 10步：f(n)=20
   7   6   3
   8   2   0
   4   1   5
第 11步：f(n)=19
   7   6   3
   8   0   2
   4   1   5
第 12步：f(n)=20
   7   6   3
   0   8   2
   4   1   5
第 13步：f(n)=19
   0   6   3
   7   8   2
   4   1   5
第 14步：f(n)=18
   6   0   3
   7   8   2
   4   1   5
第 15步：f(n)=21
   6   3   0
   7   8   2
   4   1   5
第 16步：f(n)=24
   6   3   2
   7   8   0
   4   1   5
第 17步：f(n)=23
   6   3   2
   7   0   8
   4   1   5
第 18步：f(n)=24
   6   3   2
   7   1   8
   4   0   5
第 19步：f(n)=25
   6   3   2
   7   1   8
   4   5   0
第 20步：f(n)=24
   6   3   2
   7   1   0
   4   5   8
第 21步：f(n)=23
   6   3   0
   7   1   2
   4   5   8
第 22步：f(n)=22
   6   0   3
   7   1   2
   4   5   8
到达目标状态，共计22步
共扩展结点1342个
共生成结点2243个
用时14ms
```

初始态：273645801 f(n)=16 子节点 273605841 273645081 273645810
第 1步：273605841 f(n)=15 子节点 203675841 273065841 273650841 273645801
第 2步：273065841 f(n)=18 子节点 073265841 273605841 273865041
第 3步：073265841 f(n)=17 子节点 703265841 273065841
第 4步：703265841 f(n)=16 子节点 073265841 730265841 763205841
第 5步：763205841 f(n)=17 子节点 703265841 763025841 763250841 763245801
第 6步：763025841 f(n)=18 子节点 063725841 763205841 763825041
第 7步：763825041 f(n)=21 子节点 763025841 763825401
第 8步：763825401 f(n)=20 子节点 763805421 763825041 763825410
第 9步：763825410 f(n)=21 子节点 763820415 763825401
第 10步：763820415 f(n)=20 子节点 760823415 763802415 763825410
第 11步：763802415 f(n)=19 子节点 703862415 763082415 763820415 763812405
第 12步：763082415 f(n)=20 子节点 063782415 763802415 763482015
第 13步：063782415 f(n)=19 子节点 603782415 763082415
第 14步：603782415 f(n)=18 子节点 063782415 630782415 683702415
第 15步：630782415 f(n)=21 子节点 603782415 632780415
第 16步：632780415 f(n)=24 子节点 630782415 632708415 632785410
第 17步：632708415 f(n)=23 子节点 602738415 632078415 632780415 632718405
第 18步：632718405 f(n)=24 子节点 632708415 632718045 632718450
第 19步：632718450 f(n)=25 子节点 632710458 632718405
第 20步：632710458 f(n)=24 子节点 630712458 632701458 632718450
第 21步：630712458 f(n)=23 子节点 603712458 632710458
第 22步：603712458 f(n)=22

实验结论:

通过比较在相同起始态和目标态时分别使用 h1(n)和 h2(n)作为启发式函数得出解所需要的时间(下图所示)，可以发现当 h(n)越接近 h*(n)时，A*算法的效率越高。

起始态	目标态	h1(n)用时(ms)	h2(n)用时(ms)	h1(n)生成结点数	h2(n)生成结点数	h1(n)扩展结点数	h2(n)扩展结点数	最短步数
283164705	123784065	<1	<1	13	16	6	8	5
603712458	123784065	42	9	6679	1499	4276	927	21
264705183	123784065	109	24	15163	3619	10069	2336	22
804265173	123784065	153	28	20704	3828	13761	2449	23
123804765	603712458	155	20	21475	3262	14031	1980	23
273645801	123804765	3	<1	499	92	284	50	15
283164705	123804765	<1	<1	14	12	6	5	5
804265173	283164705	11	4	1921	774	1176	463	18
264705183	603712458	43	5	6886	831	4303	487	21
273645801	603712458	83	20	12407	2243	7818	1342	22

四. 总结

(1) 实验中存在的问题及解决方案

问题 1. 原始程序效率低下
解决方案：开始运用 STL 实现程序，用 vector 来实现 closed 表，用 priority_queue 来实现 open 表，这样程序的编写比较简单直观，每次从 open 表中取最小的元素方便。但是程序的性能有很大不足。因为每次查询都要遍历 open 表以及 closed 表一遍，时间开销非常大，以至于对于简单的样例需要数十毫秒，甚至对于复杂样例需要数秒的时间，为了解决这个问题，采用哈希表+堆的方式存储 open、closed 表，通过哈希表记录索引的位置，再用堆来排序，通过将 fn 不同赋值实现 open 表与 closed 表中元素的转换。

问题 2：无法快速判断一个状态对应的节点是否属于 open/closed 表
解决方案：使用 hash 建立状态到结点的映射关系

问题 3：集成环境比较大量数据的运行时间不方便
解决方案：使用 cmd 运行，批处理程序一次运行许多样例将结果分别重定向到不同文件，更容易比较效率

(2) 后续改进方向

①希望能找到一种比 h2(n)更接近 h*(n)的启发式函数来进一步提高 A*搜索的效率。
②在堆操作上做一些优化，或者直接寻找到表示和维护 open/closed 表更好的方法。
③尝试寻找 8 数码问题的更优搜索算法，比如双向广度优先搜索。