实验报告：有趣的实验

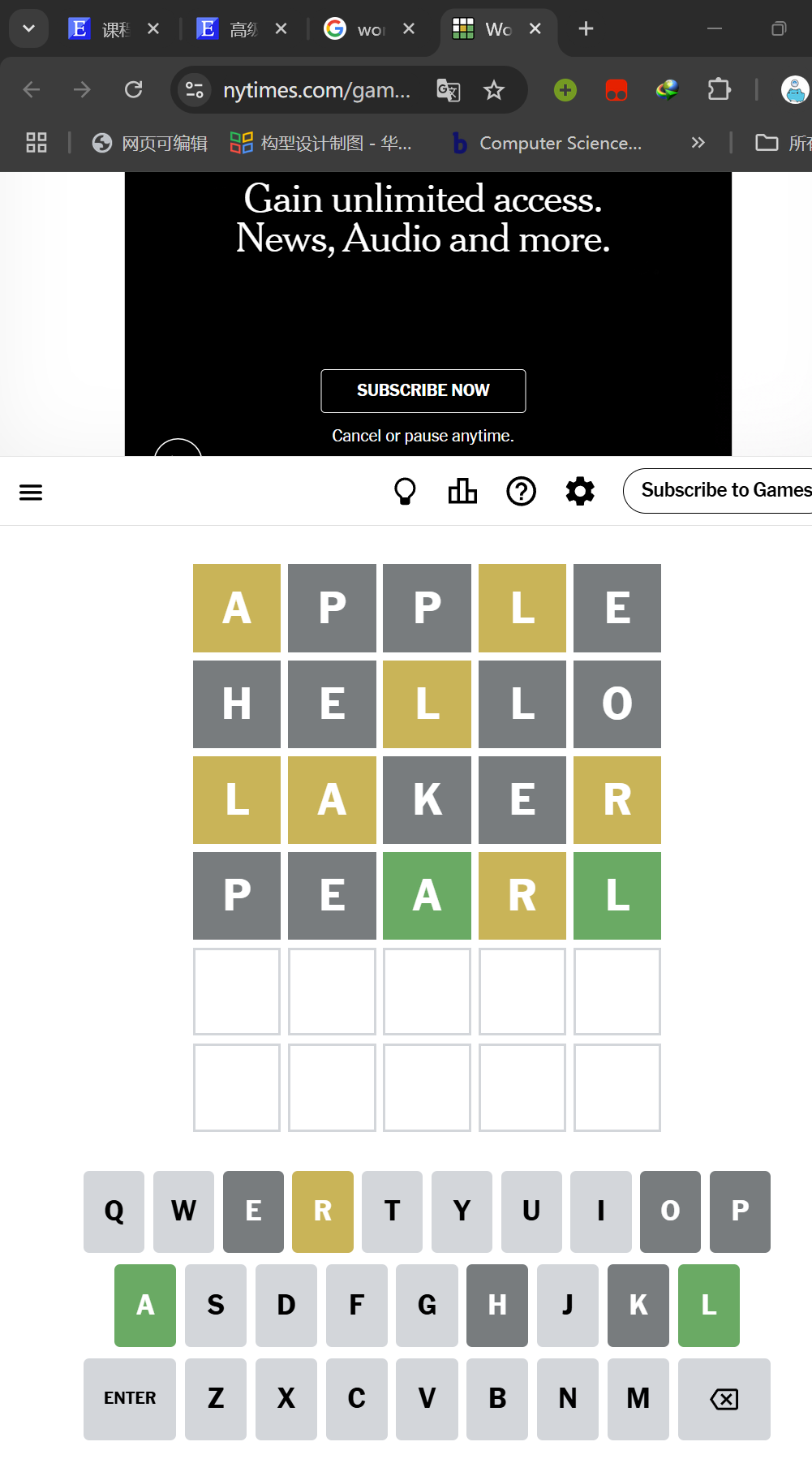
樊子谦

软件学院 软件工程 2024级1班

（因为确实很有趣，所以都做了一遍）

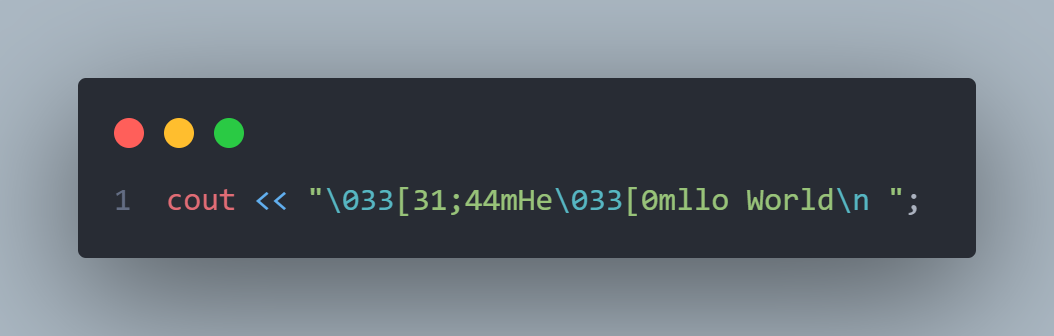
【1】                      猜数游戏。由计算机“想”一个四位数，请人猜这个四位数是多少。人输入四位数字后，计算机首先判断这四位数字中有几位是猜对了，并且在对的数字中又有几位位置也是对的，将结果显示出来，给人以提示，请人再猜，直到人猜出计算机所想的四位数是多少为止。

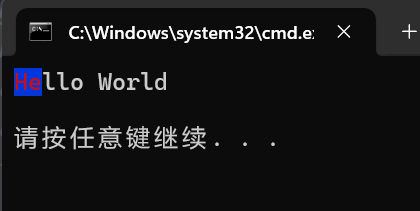
开始的时候并没有看懂题目，也不太明白“有几位是猜对了，并且在对的数字中又有几位位置也是对的”这个怎么表示。忽然灵光一闪：



这不就是我们小时候经常玩的wordle嘛。好了，“将结果显示出来”这个也能够明白怎么解决了。如果位置对了就用绿色底色的数字，如果位置不对但是数字对了就用黄色底色的数字，如果数字不对就用红色底色的数字。问题是，我们现在还没有学什么图形化的知识，有没有什么简单的方式可以做出这个视觉效果呢？

百度一下，发现可以用这样的输出控制控制台的颜色。\033是控制符，形似[42;30m这样的指令是颜色代码。[42](vscode-file://vscode-app/c:/Users/tomfa/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)表示绿色背景，30表示黑色前景。测试如下：





然后就没有什么困难了。依照题意实现代码就行。

#include <bits/stdc++.h>

using namespace std;

int get\_rand\_num(int lower, int upper) {

mt19937 rd(random\_device{}());

uniform\_int\_distribution<int> dist(lower, upper);

return dist(rd);

}

void output(string guess, string answer) {

// 这里采用wordle的视觉效果，如果位置对了就用绿色底色的数字，如果位置不对但是数字对了就用黄色底色的数字，如果数字不对就用红色底色的数字

for (int i = 0; i < 4; i++) {

if (guess[i] == answer[i]) {

cout << "\033[42;30m" << guess[i] << "\033[0m";

} else if (answer.find(guess[i]) != string::npos) {

cout << "\033[43;30m" << guess[i] << "\033[0m";

} else {

cout << "\033[41;30m" << guess[i] << "\033[0m";

}

}

cout << endl;

}

void game() {

int answer = get\_rand\_num(1000, 9999);

cout << "Rule:\n";

cout << "Please guess a 4-digit number!" << endl;

cout << "If the position is right, output the digit with the \033[42;30mgreen\033[0m background;\n";

cout << "If the position is wrong, but the number is right, use the number with the \033[43;30myellow\033[0m background;\n";

cout << "If the number is wrong, use the number with the \033[41;30mred\033[0m background.\n";

cout << "Let's start!\n\n";

int guess;

while (guess != answer) {

cin >> guess;

if (guess < 1000 || guess > 9999) {

cout << "Invalid input. Please enter a 4-digit number." << endl;

continue;

}

output(to\_string(guess), to\_string(answer));

}

cout << "You win! ^\_^" << endl;

}

int main() {

game();

bool new\_game;

cout << "New game? (1/0): ";

cin >> new\_game;

if (new\_game) {

system("clear");

main();

} else {

cout << "Goodbye!" << endl;

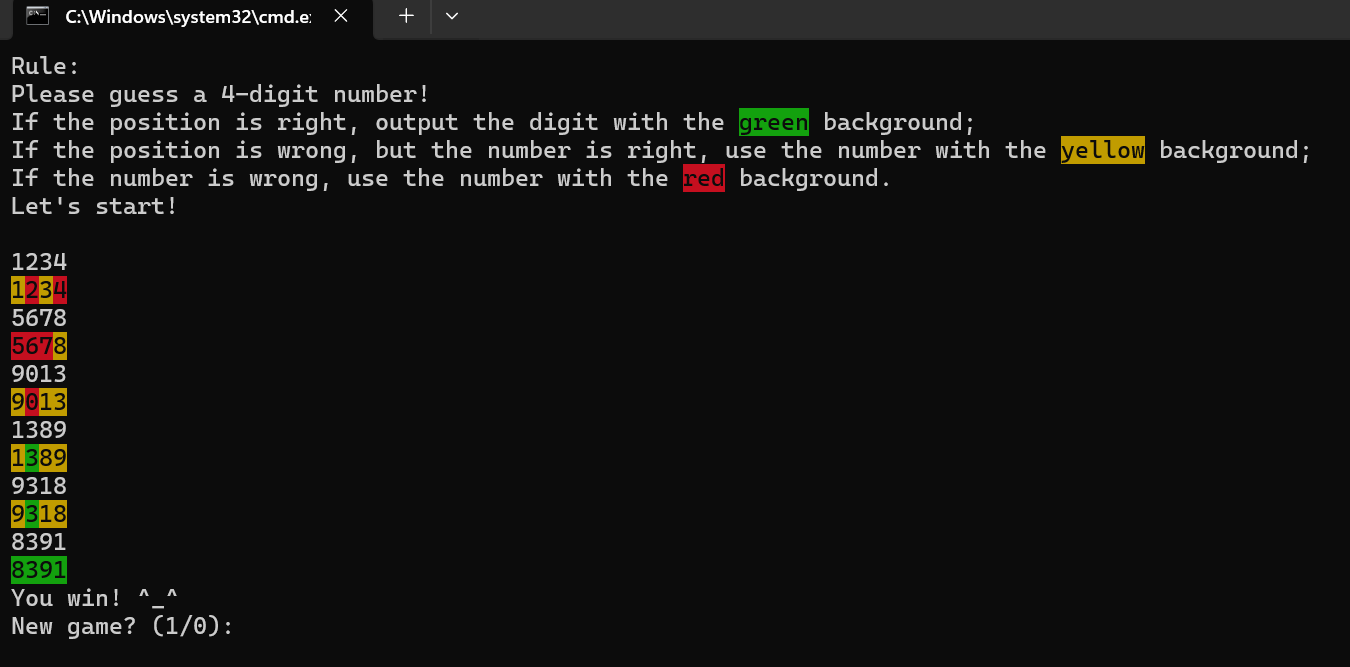
return 0;

}

return 0;

}

结果如下，视觉效果上很好。



【2】                      自动发牌。一副扑克有52张牌，打桥牌时应将牌分给四个人。请设计一个程序完成自动发牌的工作。要求：黑桃用S(Spaces)表示；红桃用H(Hearts)表示；方块用D(Diamonds)表示；梅花用C(Clubs)表示。按照打桥牌的规定，每人应当有13张牌。在人工发牌时，先进行洗牌……

一个朴素且糟糕的想法：先生成一个牌堆里面有所有的牌，用vis标记是否已经抽出，用count记录已经抽出的牌的张数，当count小于52的时候反复执行：

#伪代码

if 抽到了已经抽过的牌

continue

else

把这张牌发给现在正在被发牌的玩家

count++

vis[这张牌]=1

但很遗憾，这个算法的复杂度高达n^2，假想现在只剩下一张牌，那么抽到它的概率只有1/52，期望次数是52次能抽到。虽然数据量小，这样的复杂度我们心理上肯定是不接受。

然后我们想到了一个更简单粗暴的算法：把52张牌生成出来，然后随机打乱。（这里要引用一个新的知识点-shuffle，用来把一个序列随机打乱），然后平均分成4份，这样就可以啦！具体实现见代码。

（P.S. 这个算法的称呼好像就叫做“洗牌算法”）

#include <bits/stdc++.h>

using namespace std;

const char type[] = {'S', 'H', 'D', 'C'};

const string number[] = {"A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"};

vector<string> get\_card\_sequence() {

vector<string> card\_sequence;

for (const auto &t : type) {

for (const auto &n : number) {

string card;

card += t;

card += n;

card\_sequence.push\_back(card);

}

}

mt19937 rd(random\_device{}());

shuffle(card\_sequence.begin(), card\_sequence.end(), rd);

return card\_sequence;

}

void deal\_cards() {

vector<string> card\_sequence = get\_card\_sequence();

for (int i = 0; i < 4; i++) {

cout << "Player " << i + 1 << ": ";

for (int j = 0; j < 13; j++) {

cout << card\_sequence[i \* 13 + j] << " ";

}

cout << endl;

}

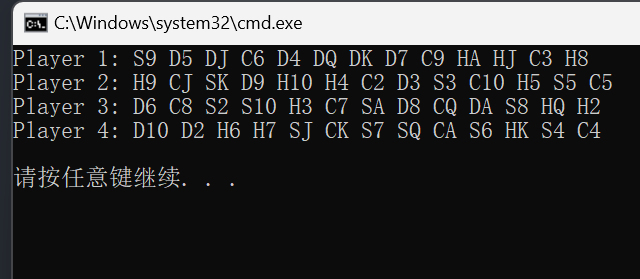
}

int main() {

deal\_cards();

return 0;

}



【3】                      在图中的九个点上,空出中间的点,其余的点上任意填入数字1到8;1的位置固定不动,然后移动其余的数字,使1到8顺时针从小到大排列.移动的规律是:只能将数字沿线移向空白的点。请编程显示数字移动过程。



粗略算一下，假如无章法地进行记忆化搜索，搜索树最大是多大？

1-8和空格可以处在任何位置。因此利用高中数学排列组合的知识，搜索树大小最大为9!=362880。再加上最终状态我们是确定的（因为1不能移动，状态唯一），所以这个规模明显很适合搜索。

然后开始写搜索！很辛苦地写了一百多行代码：

#include <bits/stdc++.h>

using namespace std;

const vector<vector<int>> final\_result[8] = {

{{1, 2, 3},

{8, 0, 4},

{7, 6, 5}},

{{8, 1, 2},

{7, 0, 3},

{6, 5, 4}},

{{7, 8, 1},

{6, 0, 2},

{5, 4, 3}},

{{2, 3, 4},

{1, 0, 5},

{8, 7, 6}},

{{6, 7, 8},

{5, 0, 1},

{4, 3, 2}},

{{3, 4, 5},

{2, 0, 6},

{1, 8, 7}},

{{4, 5, 6},

{3, 0, 7},

{2, 1, 8}},

{{5, 6, 7},

{4, 0, 8},

{3, 2, 1}},

};

void generate(auto &a, auto &ans) {

int one\_place = 0;

a[1][1] = 0;

vector<int> v;

for (int i = 1; i <= 8; i++) v.push\_back(i);

mt19937 g(random\_device{}());

shuffle(v.begin(), v.end(), g);

int flag = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (i == 1 && j == 1) {

flag = 1;

continue;

}

a[i][j] = v[i \* 3 + j - flag];

if (a[i][j] == 1) one\_place = i \* 3 + j - flag;

}

}

ans = final\_result[one\_place];

}

void output(const auto &a) {

/\*

按照一个米字格的格式字符画输出

1-2-3

|\|/|

4-0-5

|/|\|

6-7-8

\*/

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (a[i][j] == 0)

cout << "0";

else

cout << a[i][j];

if (j != 2) {

cout << "-";

}

}

cout << '\n';

if (i != 2)

cout << "|\\|/|" << '\n';

}

cout << '\n';

}

#define pii pair<int, int>

const int dx[] = {0, 0, 1, -1, 1, 1, -1, -1};

const int dy[] = {1, -1, 0, 0, -1, 1, -1, 1};

void bfs(vector<vector<int>> &a, const vector<vector<int>> &ans) {

map<vector<vector<int>>, bool> mp;

queue<pair<vector<vector<int>>, pii>> q;

queue<stack<vector<vector<int>>>> q2;

q.push({a, {1, 1}});

mp[a] = 1;

while (!q.empty()) {

auto [now, zero] = q.front();

q.pop();

auto st = q2.front();

q2.pop();

if (now == ans) {

while (!st.empty()) {

output(st.top());

st.pop();

}

return;

}

auto [zx, zy] = zero;

for (int i = 0; i < 8; i++) {

int x = zx + dx[i];

int y = zy + dy[i];

if (x >= 0 && x < 3 && y >= 0 && y < 3) {

swap(now[x][y], now[zx][zy]);

if (!mp[now]) {

q.push({now, {x, y}});

mp[now] = 1;

st.push(now);

q2.push(st);

}

swap(now[x][y], now[zx][zy]);

}

}

}

}

void play() {

vector<vector<int>> a(3, vector<int>(3, 0)), ans = a;

// 输出a

generate(a, ans);

output(a);

// output(ans);

bfs(a, ans);

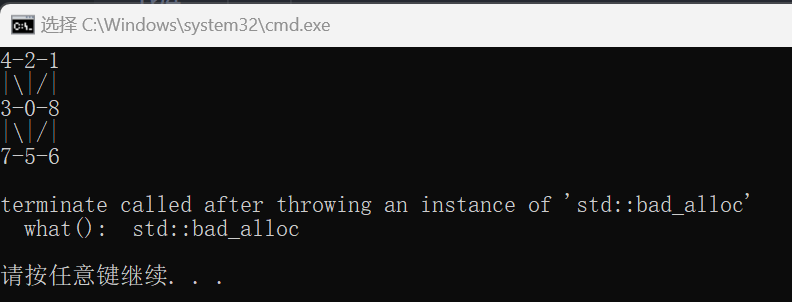
}

int main() {

play();

return 0;

}



看到结果有点崩溃了……

分析一下原因：题目要求显示数字移动过程，于是我们就用了stack来记录搜索过程的移动过程。到树根的时候，空间的消耗是depth^2规模的（等差数列，1+2+3+…）。想到这里我们就可以否决掉搜索了。

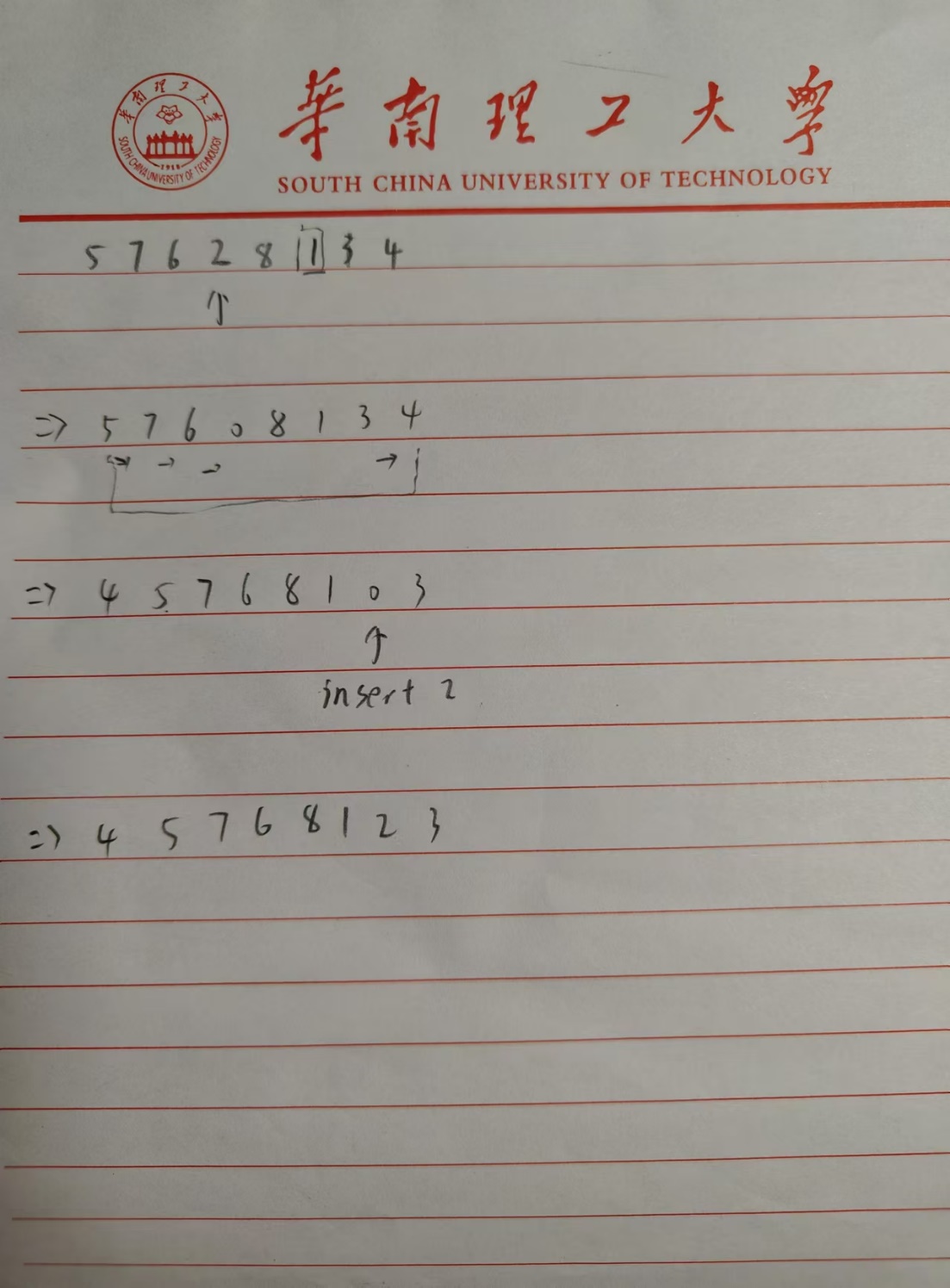
这个时候同学过来看到满脸愁容的我，提示：这个游戏是有策略的！

要求最终状态为四周顺时针形成一个有序的序列，而中间的那个点四通八达，可以到达任一点。这就类似于一个乱序序列，只给我一个临时变量，要求我排序为正序序列。那很容易联想到，可以运用插入排序的思想解决这个问题。

即，从1开始，找到2在哪里，然后让2先到中间，其它的都逆时针退一个（在序列里面相当于轮换右移）直到补全0.以此类推直到有序。

既然这样想了，那就可以用一个序列而不需要数组来表示棋盘了，可喜可贺。不过要多写一步操作来把这个序列变成棋盘形状。

实现过程中，枚举当前需要被归位的数字（2~8），枚举完了就结束了。



（具体过程如上图所示）

#include <bits/stdc++.h>

using namespace std;

auto transfrom(const auto &a, int mid\_place) {

vector<vector<int>> b(3, vector<int>(3));

b[0][0] = a[0];

b[0][1] = a[1];

b[0][2] = a[2];

b[1][2] = a[3];

b[2][2] = a[4];

b[2][1] = a[5];

b[2][0] = a[6];

b[1][0] = a[7];

b[1][1] = mid\_place;

return b;

}

void print(const auto &v) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (v[i][j] == 0)

cout << "0";

else

cout << v[i][j];

if (j != 2) {

cout << "-";

}

}

cout << '\n';

if (i == 0)

cout << "|\\|/|" << '\n';

if (i == 1)

cout << "|/|\\|" << '\n';

}

cout << '\n';

system("pause");

}

void change(auto &a, int t, int ind) {

// 第一个参数为一维序列，第二个为要被交换到中间的数字

int swap\_ind = 0; // 在a中索引要被交换的数字的位置

for (int i = 0; i < 8; i++) {

if (a[i] == t) {

swap\_ind = i;

break;

}

}

a[swap\_ind] = 0; // 先把这个位置挖空

print(transfrom(a, t));

for (int i = swap\_ind - 1;; i--) {

if (i == -1) i = 7;

swap(a[(i + 1) % 8], a[i]);

print(transfrom(a, t));

if (i == ind) break;

}

a[ind] = t;

print(transfrom(a, 0));

}

int generate(auto &a) { // 返回顺时针的情况下，左上角为第一个位置，1在第几个位置

for (int i = 1; i <= 8; i++) a.push\_back(i);

mt19937 g(random\_device{}());

shuffle(a.begin(), a.end(), g);

for (int i = 0; i < 8; i++) {

if (a[i] == 1) return i;

}

return -1;

}

void play() {

vector<int> a;

int place\_of\_1 = generate(a);

cout << "The initial state:\n";

print(transfrom(a, 0));

for (int move\_num = 2; move\_num <= 8; move\_num++) {

int ind = (place\_of\_1 + move\_num - 1) % 8; // 当前枚举到的数字原本应该处于的下标

if (a[ind] != move\_num) change(a, move\_num, ind);

}

cout << "Done!";

}

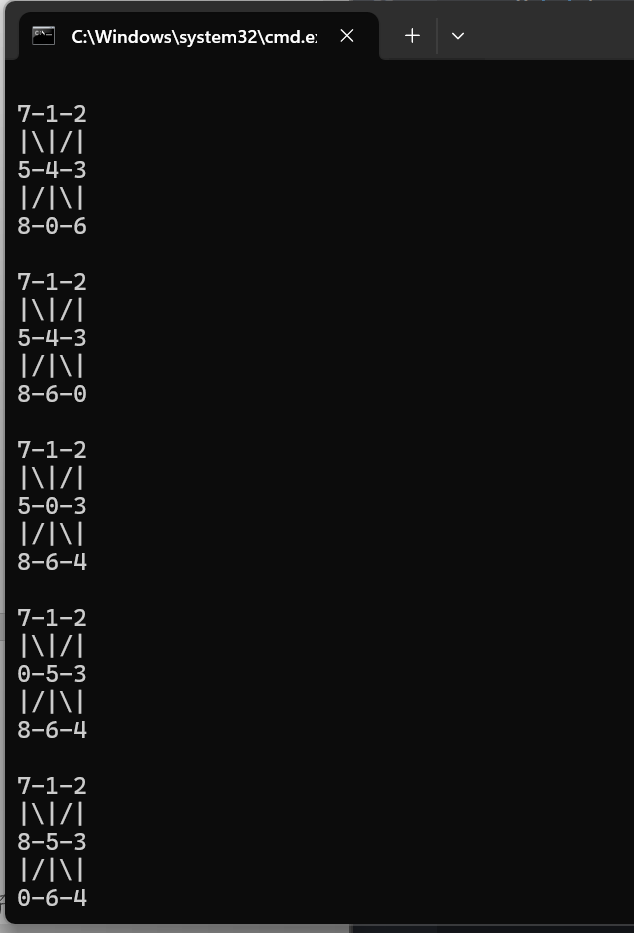
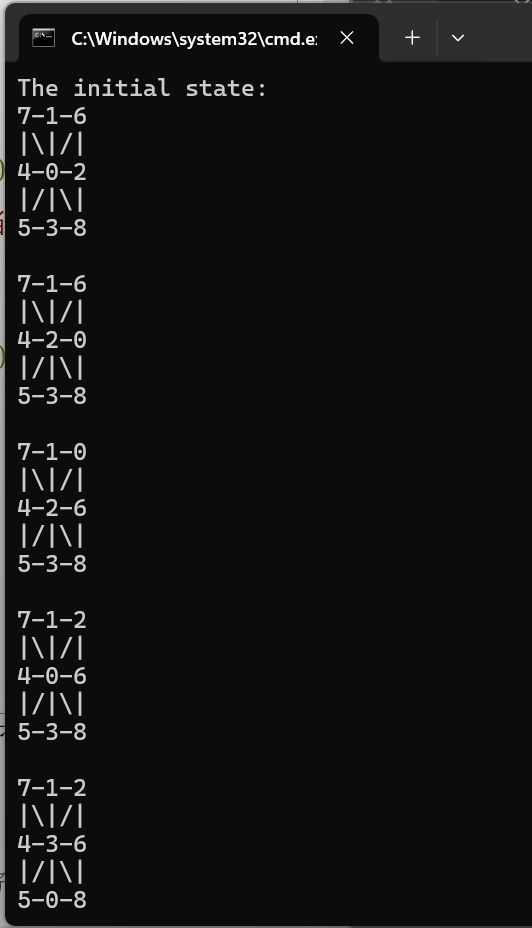
int main() {

play();

return 0;

}

输出太长不全部展示，这里只展现部分，可以看出来实现上是没有问题的：



【4】                      游戏的目的是用最少的步数将上图中白子和黑子的位置进行交换：游戏的规则是：(1)一次只能移动一个棋子； (2)棋子可以向空格中移动，也可以跳过一个对方的棋子进入空格，但不能向后跳，也不能跳过两个子。请用计算机实现上述游戏。例如：

有三个白子和三个黑子如下图布置：

○ ○ ○ \_ ● ● ●

游戏的目的是用最少的步数将上图中白子和黑子的位置进行交换：

● ● ● \_ ○ ○ ○

黑白子可用不同的符号来表示。

感觉还是可以深搜，但是经过【3】的教训，是不是不太敢了？

先谨慎思考一下：

根据排列组合，棋盘的状态只有7!/(3!\*3!)=140种，还是很安全的，我们可以大胆用搜索。因为这里要求【最少步数】，显然用广搜是比深搜好的。

至于记忆化，因为规模小，所以直接用七位的vector和map实现就好，转换成”1110222”这样的整数甚至二进制数，虽然复杂度优秀，但是实现很麻烦。

#include <bits/stdc++.h>

using namespace std;

struct Node {

vector<int> state;

int step;

queue<vector<int>> s; // answer’s step

Node(vector<int> state, int step, queue<vector<int>> s) : state(state), step(step), s(s) {}

};

void solve() {

vector<int> a = {1, 1, 1, 0, 2, 2, 2};

map<vector<int>, bool> m;

m[a] = true;

queue<Node> q;

queue<vector<int>> init;

init.push(a);

q.push(Node(a, 0, init));

while (!q.empty()) {

auto [state, step, s] = q.front();

q.pop();

auto pos = find(state.begin(), state.end(), 0) - state.begin();

// cout << pos;

if (state == vector<int>{2, 2, 2, 0, 1, 1, 1}) {

cout << "The min step is:" << step << endl;

while (!s.empty()) {

auto state = s.front();

s.pop();

for (const auto &i : state) cout << i << " ";

cout << '\n';

}

return;

}

for (int i = -2; i <= 2; i++) {

if (pos + i < 0 || pos + i >= 7 || i == 0) continue;

swap(state[pos], state[pos + i]);

if (!m[state]) {

m[state] = true;

s.push(state);

q.push({state, step + 1, s});

}

swap(state[pos], state[pos + i]);

}

}

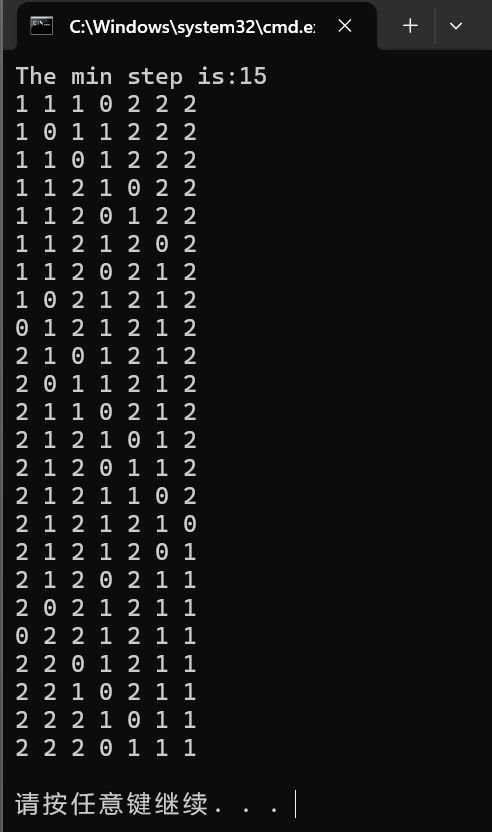
}

int main() {

solve();

return 0;

}



输出应该没有问题。

【5】                      写一个程序从输入文件中去读取四行大写字母（全都是大写的，每行不超过 100100 个字符），然后用柱状图输出每个字符在输入文件中出现的次数。严格地按照输出样例来安排你的输出格式。（洛谷P1598）

例如：输入

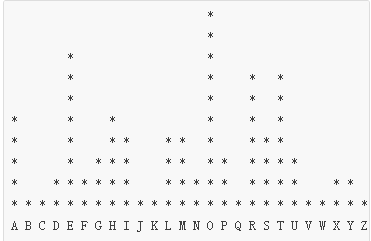
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.

THIS IS AN EXAMPLE TO TEST FOR YOUR

HISTOGRAM PROGRAM.

HELLO!

则输出为：



看到题面眉头一皱，从输入文件中去读取？要用freopen？但是又没有给文件名。去洛谷搜了一下原题发现似乎并没有这个要求。那就忽略掉。

然后就是考虑读入。读取四行大写字母，要么用getline，要么直接while(cin>>s)，这里为了更加应景（指配合题目要求的“四行”），用前者。然后开26个桶，遍历每一个字符，如果是大写字母就把对应的桶里面的个数+1.

输出怎么办？直接考虑输出的那个形式比较复杂，然后我们灵机一动——只要我们把整个图上下颠倒，赋值给一个bool数组，然后再倒过来输出，是0的地方输出空格，是1的地方输出星星，这样就很方便了。

#include <bits/stdc++.h>

using namespace std;

void solve() {

int count[26];

fill(count, count + 26, 0);

for (int i = 1; i <= 4; i++) {

string s;

getline(cin, s);

// cout << s << '\n';

for (const auto &c : s) {

if (c >= 'A' && c <= 'Z') count[c - 'A']++;

}

}

int max\_count = \*max\_element(count, count + 26);

vector<vector<bool>> ans(max\_count, vector<bool>(26, false));

for (int i = 0; i < 26; i++) {

for (int j = 0; j < count[i]; j++) {

ans[j][i] = true;

}

}

for (int j = max\_count; j >= 1; j--) {

for (int i = 0; i < 26; i++) {

if (ans[j - 1][i]) {

cout << "\*";

} else {

cout << " ";

}

if (i != 25) {

cout << " ";

}

}

cout << endl;

}

for (char c = 'A'; c <= 'Z'; c++) cout << c << ' ';

}

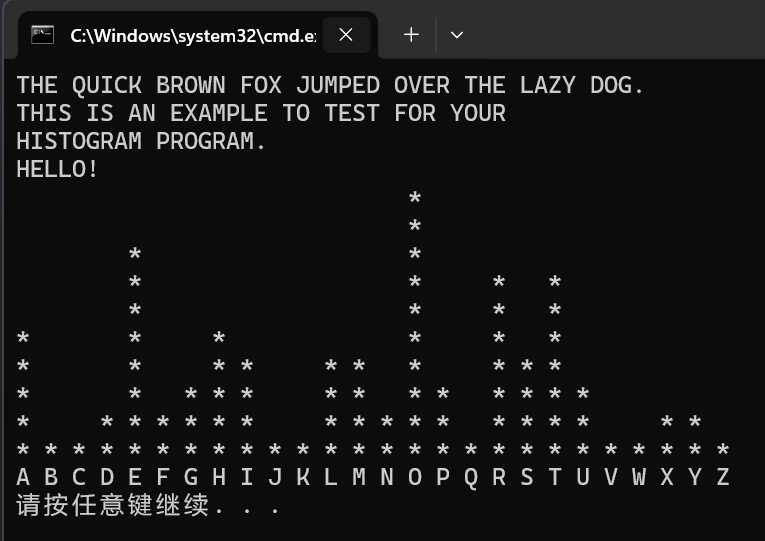
int main() {

solve();

return 0;

}

处理算是这几题里面最简单的了，输出如下：



看起来没有错，交一下试试看：



实验心得：

在这次实验中，我体验了多个有趣的编程挑战，每个都让我对算法和编程有了更深的理解。

猜数游戏让我重温了童年的wordle游戏，同时也学习了如何通过控制台颜色代码来增强用户交互体验。最初对题目的理解有些困难，但一旦我将问题与wordle联系起来，一切都变得明朗起来。通过这次实验，我不仅提高了我的逻辑思维能力，还学会了如何将复杂问题简化，并通过编程实现直观的用户体验。

自动发牌实验教会了我如何通过随机化和算法优化来简化问题。最初的想法虽然简单，但效率低下。通过学习洗牌算法，我意识到了算法优化的重要性，并学会了如何用更高效的方法解决问题。

数字移动实验是一个关于搜索策略的挑战。最初我尝试了暴力搜索，但很快意识到这种方法的局限性。在同学的提示下，我学会了如何运用插入排序的思想来解决问题，这让我对算法的应用有了更深的认识。

黑白子交换游戏让我体验了广度优先搜索（BFS）的魅力。通过这次实验，我学会了如何使用BFS来寻找最优解，并理解了记忆化搜索在解决小规模问题中的应用。

字符计数实验则是对输入处理和数据可视化的一次实践。我学会了如何从输入中提取信息，并使用柱状图来直观地展示数据。这次实验提高了我的数据处理能力和编程技巧。

总的来说，这次实验不仅增强了我的编程能力，还让我学会了如何将理论知识应用到实际问题中。每个实验都让我对编程有了更深的理解和欣赏，我期待将这些新学到的技能应用到未来的项目中。