目录

[一、 C++中的stl库： 1](#_Toc177072382)

[1. Vector容器： 1](#_Toc177072383)

[头文件： 1](#_Toc177072384)

[初始化： 1](#_Toc177072385)

[Vector中的使用方法： 2](#_Toc177072386)

[排序： 3](#_Toc177072387)

[访问： 5](#_Toc177072388)

[2. Stack栈 6](#_Toc177072389)

[头文件 6](#_Toc177072390)

[函数声明和使用方法： 6](#_Toc177072391)

[遍历： 7](#_Toc177072392)

[3. Set容器： 7](#_Toc177072393)

[头文件和声明 7](#_Toc177072394)

[使用方法： 8](#_Toc177072395)

[访问： 8](#_Toc177072396)

[重载运算符 9](#_Toc177072397)

[其他set： 10](#_Toc177072398)

[4. String容器 10](#_Toc177072399)

[初始化，头文件 10](#_Toc177072400)

[String访问： 10](#_Toc177072401)

[特性 11](#_Toc177072402)

[读入规则 12](#_Toc177072403)

[方法： 12](#_Toc177072404)

[Eg： 14](#_Toc177072405)

[5. Queue 14](#_Toc177072406)

[初始化 14](#_Toc177072407)

[方法 15](#_Toc177072408)

[队列模拟： 15](#_Toc177072409)

[二、 常见的算法，方法 16](#_Toc177072410)

[1. 阶乘的n非常大求结果有多少位0 16](#_Toc177072411)

[2. 循环日程赛算法 17](#_Toc177072412)

[3. 字符串筛选 元音 18](#_Toc177072413)

[4. 旋转图像 20](#_Toc177072414)

[5. 岛屿数量：（图论） 21](#_Toc177072415)

[6. 腐烂的橘子 24](#_Toc177072416)

[7. 二叉树的层序遍历： 26](#_Toc177072417)

[三、 STL函数： 27](#_Toc177072418)

# C++中的stl库：

## Vector容器：

### 头文件：

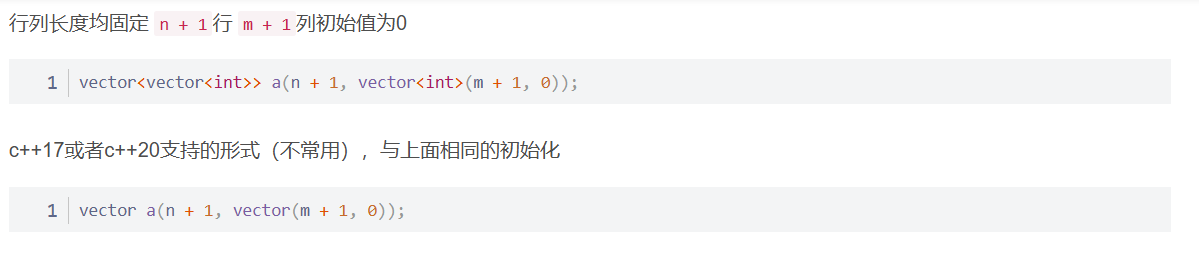
#include <vector>

数学相关的头文件：#include <algorithm>

### 初始化：

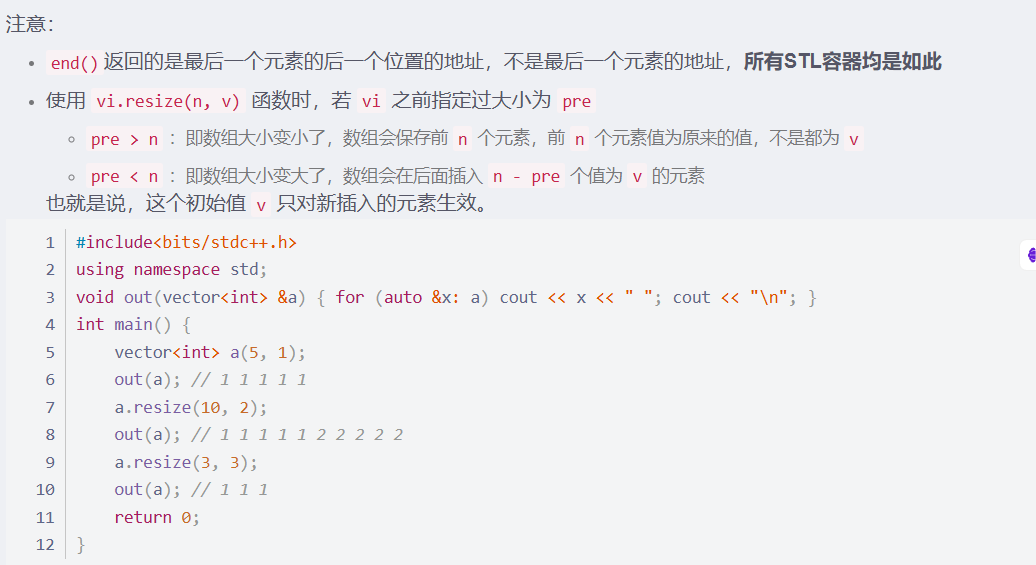




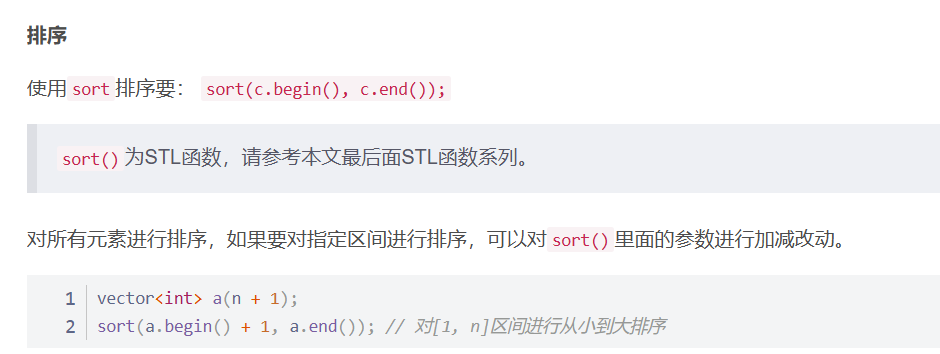


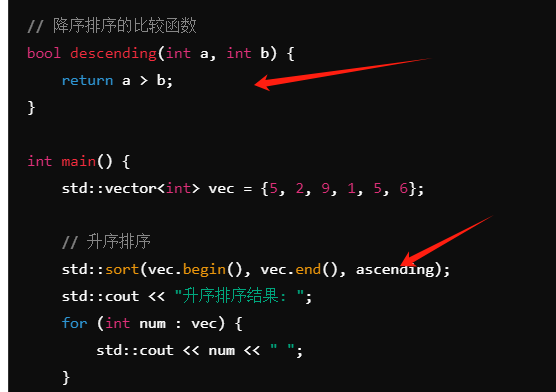
### Vector中的使用方法：





### 排序：







### 访问：









## Stack栈

### 头文件

#include<stack>

### 函数声明和使用方法：



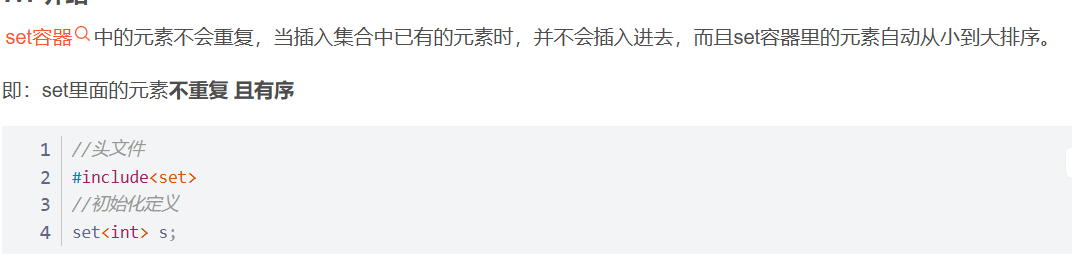
### 遍历：





## Set容器：

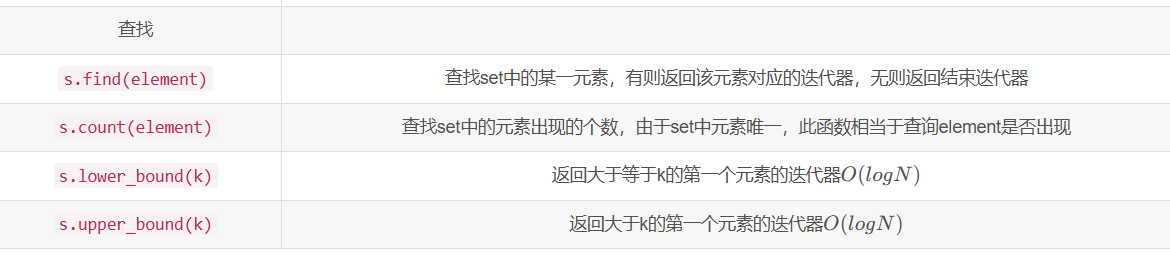
### 头文件和声明



#include<unordered\_set>

### 使用方法：





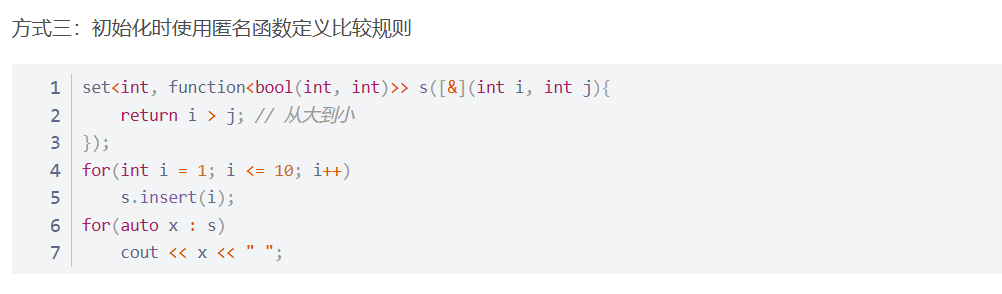
### 访问：



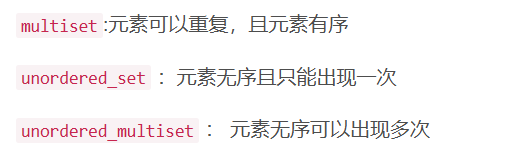
### 重载运算符







### 其他set：

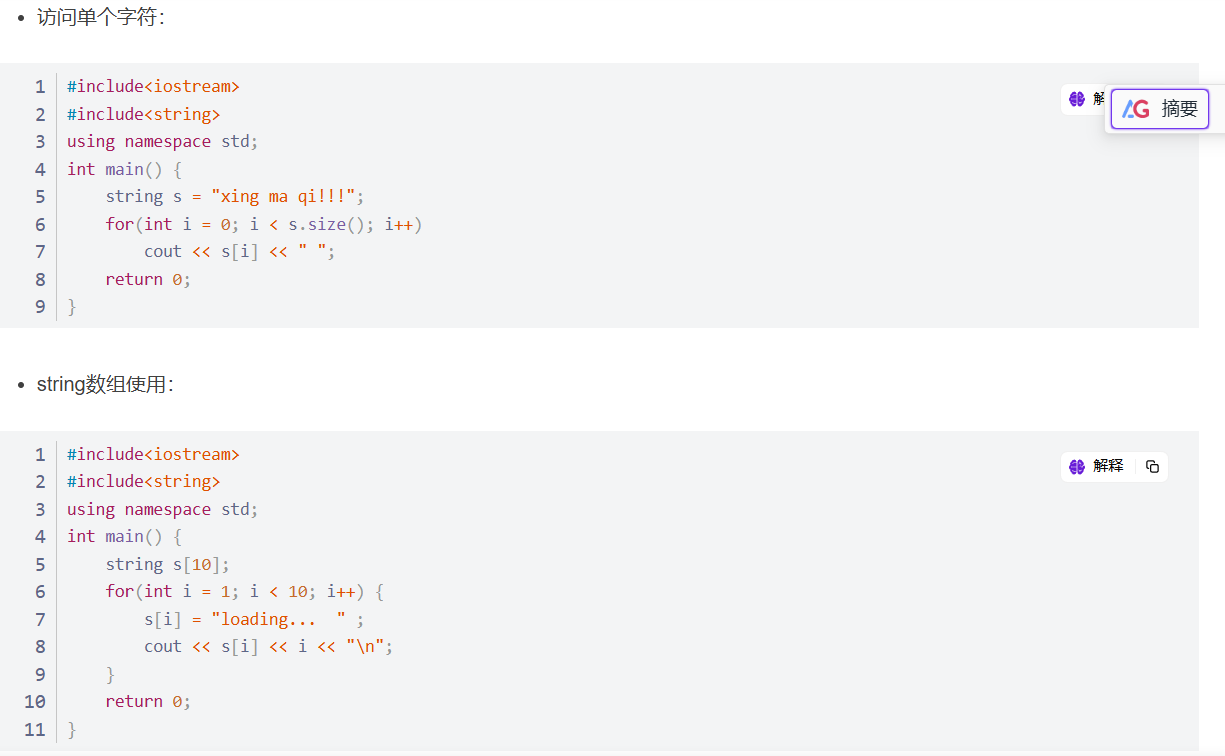


## String容器

### 初始化，头文件



### String访问：



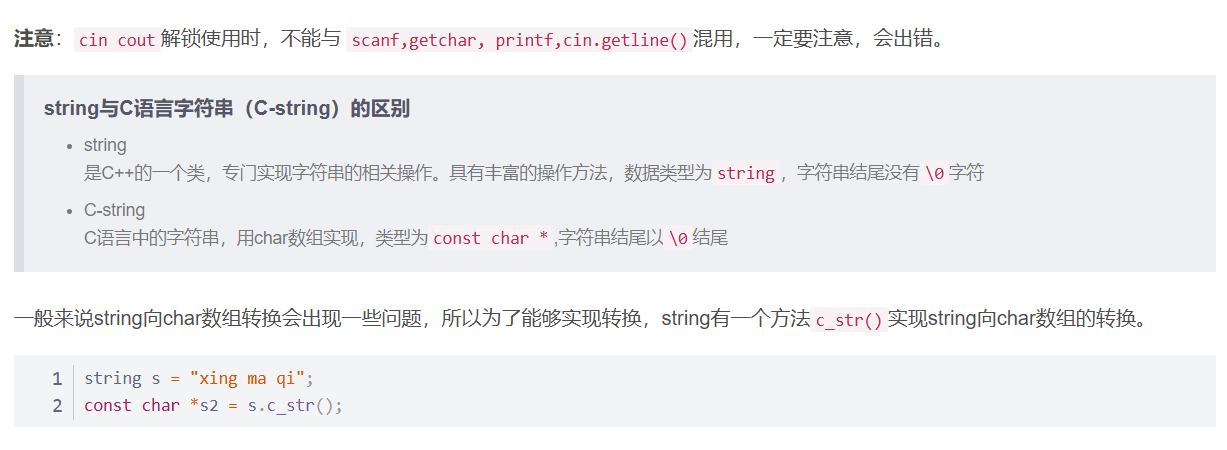
### 特性



### 读入规则

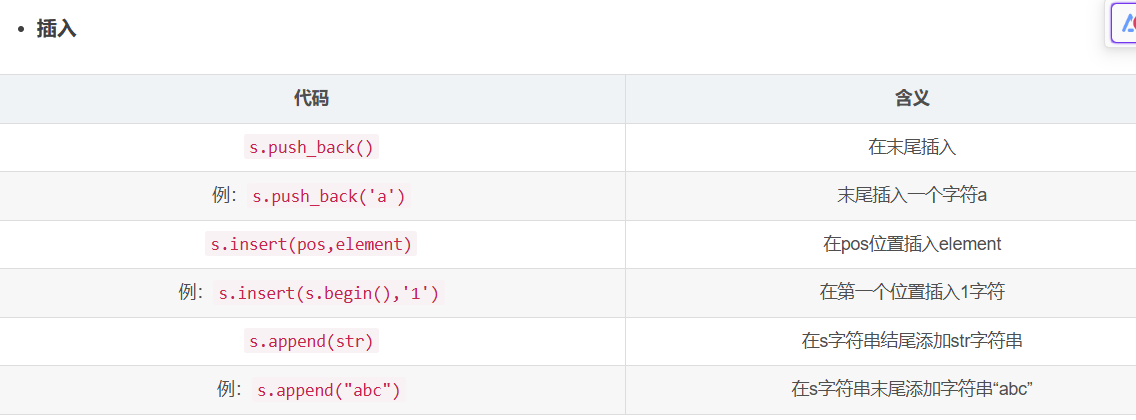






### 方法：

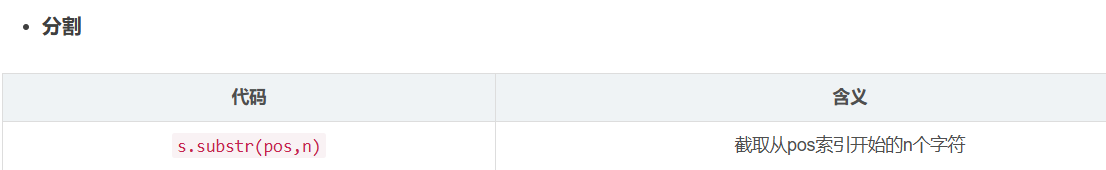




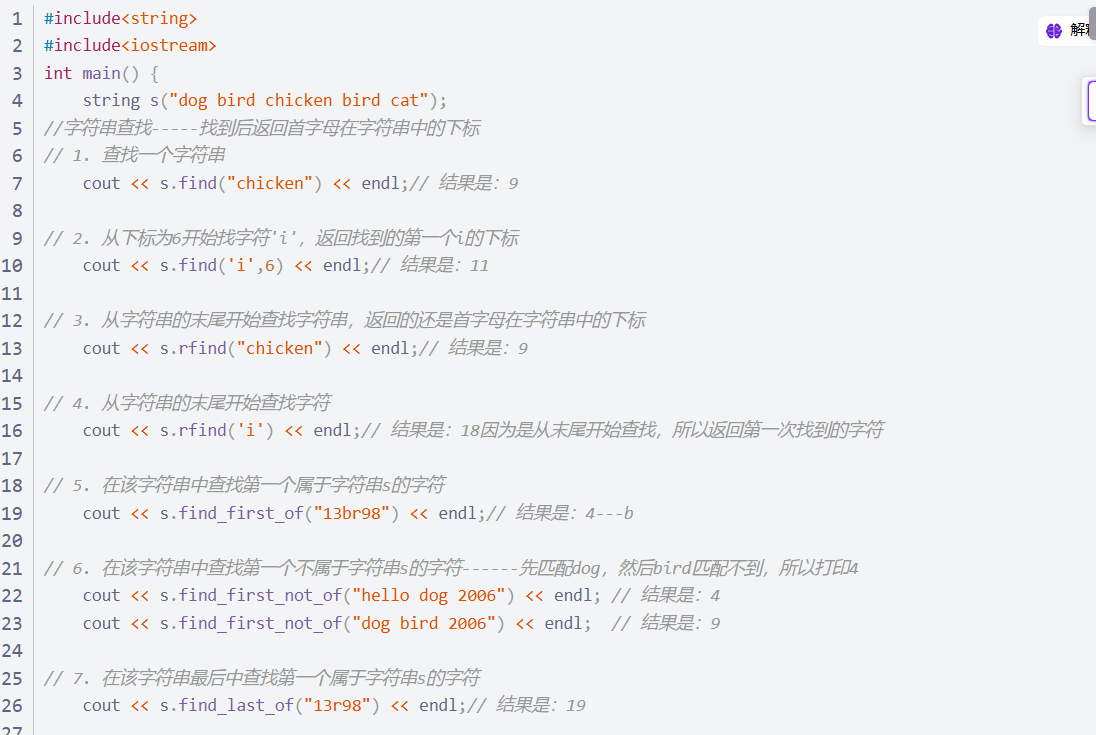








### Eg：





## Queue

### 初始化



### 方法

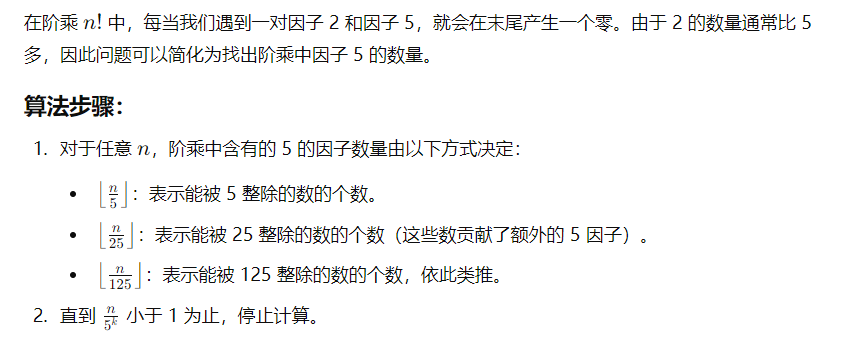


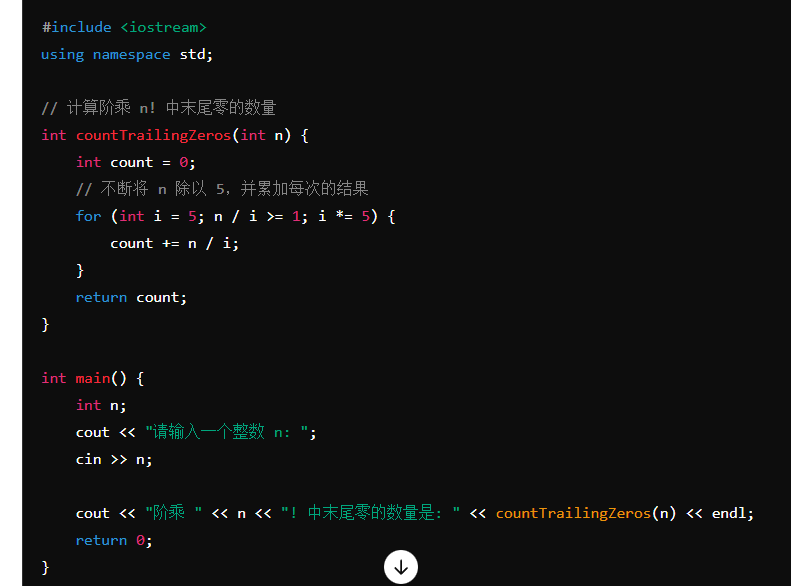
### 队列模拟：



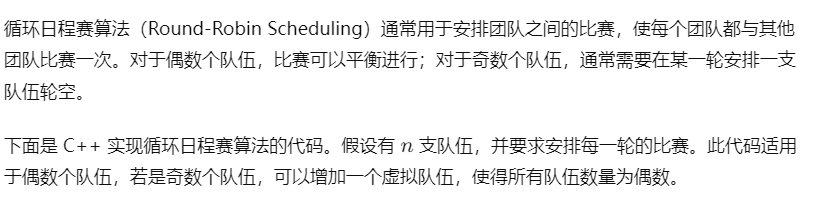
# 常见的算法，方法

## 阶乘的n非常大求结果有多少位0





## 循环日程赛算法



#include <iostream>

#include <vector>

using namespace std;

// 打印一轮比赛的安排

void printRound(int round, const vector<pair<int, int>>& matches) {

cout << "Round " << round + 1 << ":" << endl;

for (auto match : matches) {

cout << "Team " << match.first << " vs Team " << match.second << endl;

}

cout << endl;

}

// 生成并打印循环日程赛安排

void generateRoundRobinSchedule(int n) {

// 如果队伍数为奇数，增加一个虚拟队伍（编号为 0）

bool isOdd = (n % 2 != 0);

if (isOdd) n += 1;

vector<int> teams(n);

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

teams[i] = i + 1; // 初始化队伍编号，从1开始

}

for (int round = 0; round < n - 1; ++round) {

vector<pair<int, int>> matches; // 存储每轮的比赛对

for (int i = 0; i < n / 2; ++i) {

int team1 = teams[i];

int team2 = teams[n - 1 - i];

if (team1 != n && team2 != n) { // 排除虚拟队伍

matches.push\_back({team1, team2});

}

}

printRound(round, matches);

// 旋转队伍：固定第一个队伍，其他队伍顺时针轮换

int lastTeam = teams[n - 1];

for (int i = n - 1; i > 1; --i) {

teams[i] = teams[i - 1];

}

teams[1] = lastTeam;

}

}

int main() {

int n;

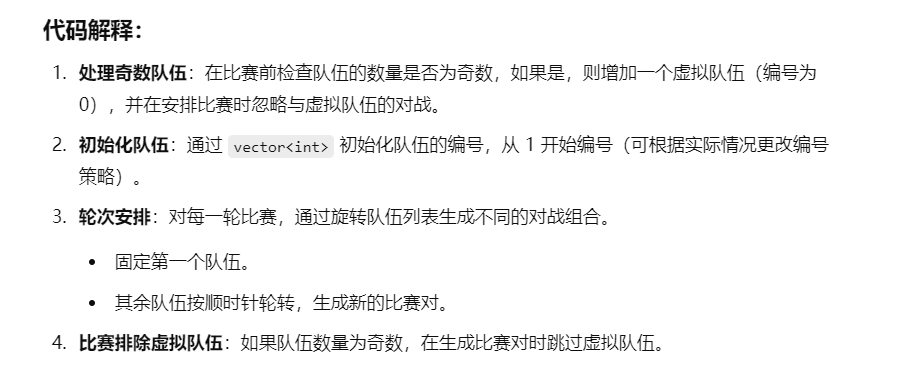
cout << "请输入队伍数量: ";

cin >> n;

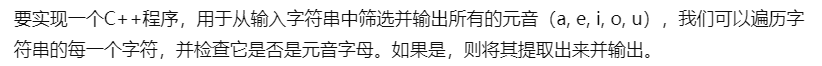
generateRoundRobinSchedule(n);

return 0;

}



## 字符串筛选 元音



#include <iostream>

#include <string>

using namespace std;

// 检查字符是否为元音

bool isVowel(char c) {

c = tolower(c); // 将字符转换为小写，方便判断

return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');

}

int main() {

string input;

// 输入字符串

cout << "请输入一个字符串: ";

getline(cin, input);

string vowels = ""; // 用于存储筛选出的元音字符

// 遍历输入字符串，筛选出元音

for (char c : input) {

if (isVowel(c)) {

vowels += c;

}

}

// 输出结果

if (!vowels.empty()) {

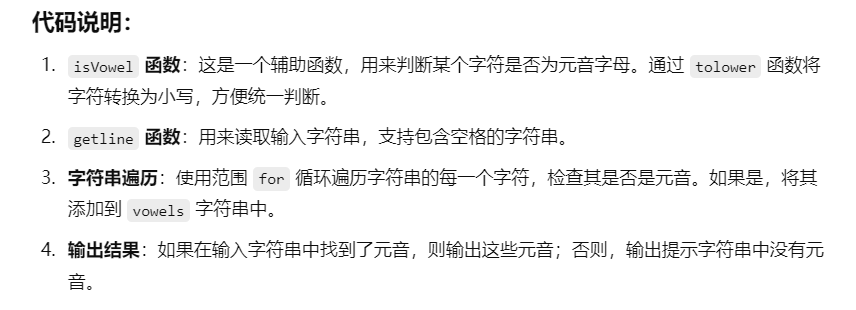
cout << "字符串中的元音是: " << vowels << endl;

} else {

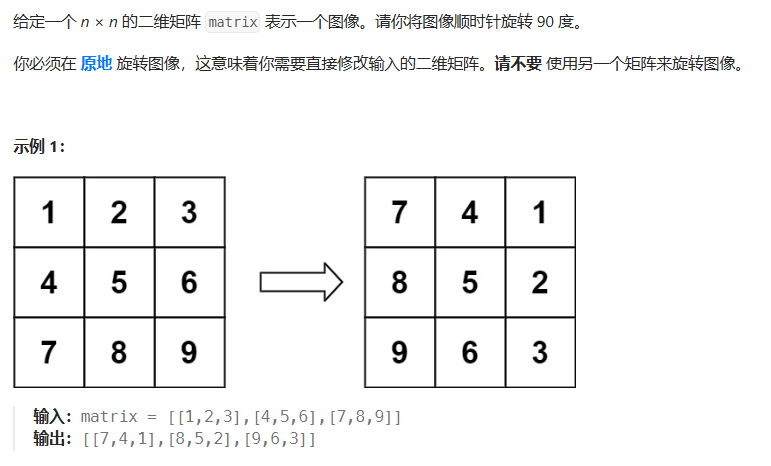
cout << "字符串中没有元音。" << endl;

}

return 0;

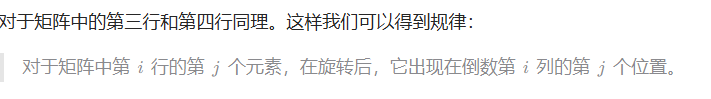
} 

## 旋转图像





观察规律，得到了



## 岛屿数量：（图论）



深度优先：

class Solution {

private:

void dfs(vector<vector<char>>& grid, int r, int c) {

int nr = grid.size();

int nc = grid[0].size();

grid[r][c] = '0';

if (r - 1 >= 0 && grid[r-1][c] == '1') dfs(grid, r - 1, c);

if (r + 1 < nr && grid[r+1][c] == '1') dfs(grid, r + 1, c);

if (c - 1 >= 0 && grid[r][c-1] == '1') dfs(grid, r, c - 1);

if (c + 1 < nc && grid[r][c+1] == '1') dfs(grid, r, c + 1);

}

public:

int numIslands(vector<vector<char>>& grid) {

int nr = grid.size();

if (!nr) return 0;

int nc = grid[0].size();

int num\_islands = 0;

for (int r = 0; r < nr; ++r) {

for (int c = 0; c < nc; ++c) {

if (grid[r][c] == '1') {

++num\_islands;

dfs(grid, r, c);

}

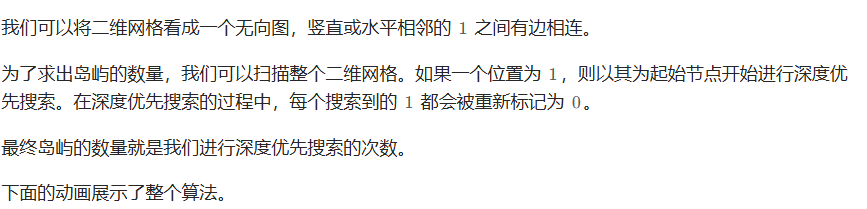
}

}

return num\_islands;

}

};



**广度优先：**

**class Solution {**

**public:**

**int numIslands(vector<vector<char>>& grid) {**

**int nr = grid.size();**

**if (!nr) return 0;**

**int nc = grid[0].size();**

**int num\_islands = 0;**

**for (int r = 0; r < nr; ++r) {**

**for (int c = 0; c < nc; ++c) {**

**if (grid[r][c] == '1') {**

**++num\_islands;**

**grid[r][c] = '0';**

**queue<pair<int, int>> neighbors;**

**neighbors.push({r, c});**

**while (!neighbors.empty()) {**

**auto rc = neighbors.front();**

**neighbors.pop();**

**int row = rc.first, col = rc.second;**

**if (row - 1 >= 0 && grid[row-1][col] == '1') {**

**neighbors.push({row-1, col});**

**grid[row-1][col] = '0';**

**}**

**if (row + 1 < nr && grid[row+1][col] == '1') {**

**neighbors.push({row+1, col});**

**grid[row+1][col] = '0';**

**}**

**if (col - 1 >= 0 && grid[row][col-1] == '1') {**

**neighbors.push({row, col-1});**

**grid[row][col-1] = '0';**

**}**

**if (col + 1 < nc && grid[row][col+1] == '1') {**

**neighbors.push({row, col+1});**

**grid[row][col+1] = '0';**

**}**

**}**

**}**

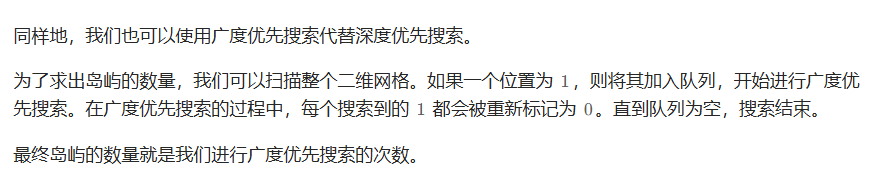
**}**

**}**

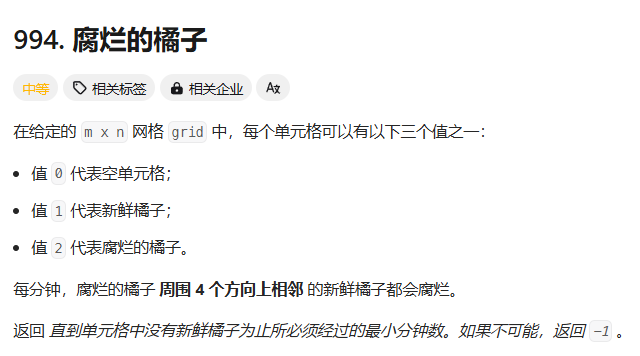
**return num\_islands;**

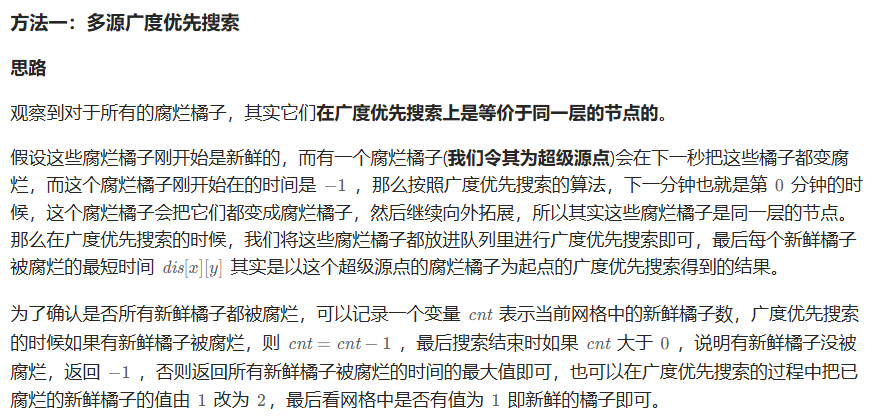
**}**

**};**



## 腐烂的橘子





class Solution {

int cnt;

int dis[10][10];

int dir\_x[4] = {0, 1, 0, -1};

int dir\_y[4] = {1, 0, -1, 0};

public:

int orangesRotting(vector<vector<int>>& grid) {

queue<pair<int, int>>Q;

memset(dis, -1, sizeof(dis));

cnt = 0;

int n = (int)grid.size(), m = (int)grid[0].size(), ans = 0;

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

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

if (grid[i][j] == 2) {

Q.emplace(i, j);

dis[i][j] = 0;

}

else if (grid[i][j] == 1) {

cnt += 1;

}

}

}

while (!Q.empty()){

auto [r, c] = Q.front();

Q.pop();

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

int tx = r + dir\_x[i];

int ty = c + dir\_y[i];

if (tx < 0|| tx >= n || ty < 0|| ty >= m || ~dis[tx][ty] || !grid[tx][ty]) {

continue;

}

dis[tx][ty] = dis[r][c] + 1;

Q.emplace(tx, ty);

if (grid[tx][ty] == 1) {

cnt -= 1;

ans = dis[tx][ty];

if (!cnt) {

break;

}

}

}

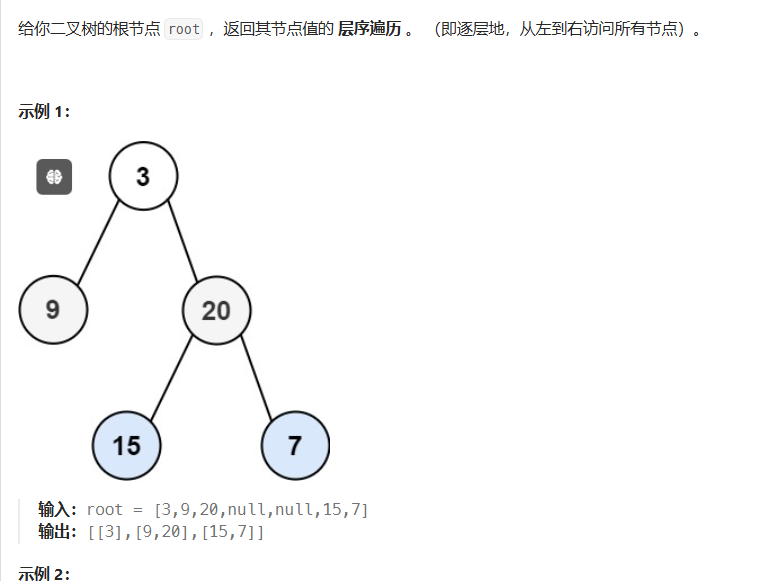
}

return cnt ? -1 : ans;

}

};

## 二叉树的层序遍历：



class Solution {

public:

vector<vector<int>> levelOrder(TreeNode\* root) {

vector <vector <int>> ret;

if (!root) {

return ret;

}

queue <TreeNode\*> q;

q.push(root);

while (!q.empty()) {

int currentLevelSize = q.size();

ret.push\_back(vector <int> ());

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

auto node = q.front(); q.pop();

ret.back().push\_back(node->val);

if (node->left) q.push(node->left);

if (node->right) q.push(node->right);

}

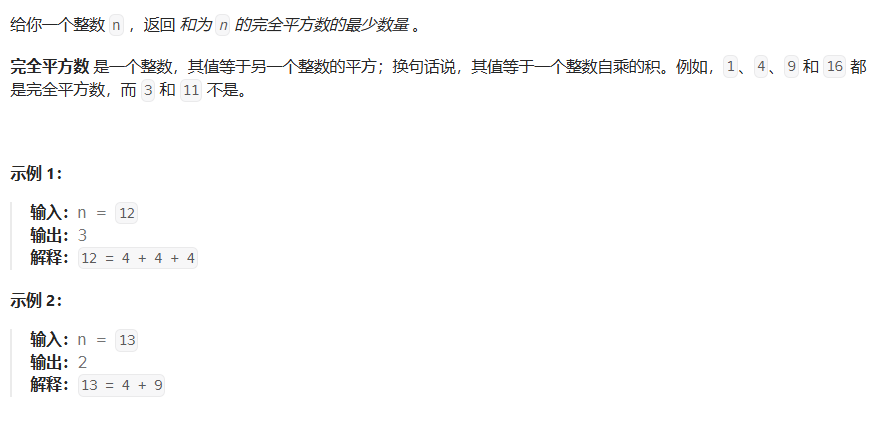
}

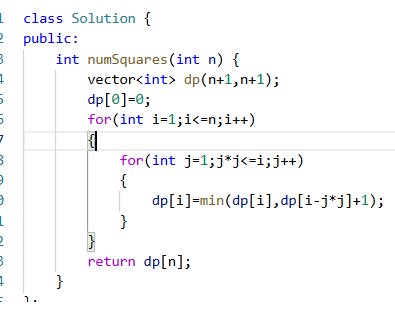
return ret;

}

};

### 完全平方数：





### 单词拆分：





# STL函数：