# 

[需要注意的要点](#_uhk26cmzmkow)

[题目](#_yvjx0za9yi06)

[memcache 与 redis 的区别](#_24fti2ijvre1)

[旋转序列查找最小元素](#_s7fnnxhfyy6t)

[服务端与客户端同步倒计时](#_dnojgp4f8o04)

[签到间隔、签到上限问题](#_7ixxee8ia6bx)

[查找满足条件的数对数量](#_vayk1qj8knfd)

[冒泡排序算法](#_vymq8d7egqsp)

[两个数组求交集](#_av70hkkazvk6)

[矩形平移](#_otl2i2b1zimg)

[数组随机打乱](#_we5aezoxlp0w)

[从数组里找出出现一次的数字](#_iczxec374a03)

[区域颜色填充算法](#_k84mqqw1a6ka)

[模拟图形界面图标点选过程(C语言版)](#_az0f0ua1s1u7)

[在一个先增后减的序列中寻找第k小](#_6cjo5vkm9in3)

[思路一：排序，复杂度O(nlogn)](#_weql5nj5qo97)

[思路二：遍历，两个指针指向序列头尾，复杂度O(n)](#_d10hvh5jbig7)

[思路三：二分，复杂度O(logn)](#_jwosd5tefr9u)

[redis中key迁移](#_spxuq5wgzf6g)

[问题描述一](#_qmq25y52n2sp)

[问题描述二](#_xr2az9rnju8v)

[注意点](#_gxit5v2h7dvc)

[方案一](#_fg07i2fjafas)

[方案二](#_s51phwakhh1i)

[前端：实现简单轮播图](#_tl7if4f9a3fz)

[命令行扫雷](#_b43t71nmo7qp)

[传送塔 Part I (Transportation Tower I)](#_f76hfrdus8uc)

[微博系统的存储应该如何设计](#_qpp8cturbhr)

[How to wait for a response in a request to stream system](#_c9gk2kgq26u4)

[生成拼音索引问题](#_cuswkz62v5bo)

[背景](#_y5s4u53roerh)

[需求](#_1kwoiry905b4)

[分析](#_jn848w9tyc4u)

# 需要注意的要点

* 面试描述问题的过程，就是在考核和对方沟通的过程有没有障碍，是否一个可以沟通的人。
* 面试结束之前必须要问对方，有没有什么问题需要面试官解答。
* 尽可能提前阅读候选人简历，在项目经历里面发掘可以详细问的问题；避免在候选人面前翻简历来找问题。
* 候选人必须要动手写代码。

# 题目

## memcache 与 redis 的区别

* memcache 只提供 key value 数据的存取 API，redis 提供丰富的数据结构访问 API
  + 常用的数据结构有哪些
* 一份列表数据，假如只能用 key value 数据结构来存取，那要怎么修改
  + 并发修改会产生什么问题
    - 怎么解决并发修改产生的问题

## 旋转序列查找最小元素

给出一个旋转后的严格单调递增数组，找出最小的元素是什么，比如旋转后的有序数组是 6 8 9 1 3 4 5，则最小元素是1。

* 使用二分查找算法，时间复杂度可优化到O(logN)
* 建议作为 coding 题目
  + 若代码写出，可以问如果求最大元素要如何修改判断条件

## 服务端与客户端同步倒计时

游戏需要展示一个世界事件发生的倒计时，服务端要如何同步这个倒计时给所有客户端

* 要考虑客户端本地时间与服务器不一致、客户端网络延迟的问题

## 签到间隔、签到上限问题

游戏世界里面有多个签到点，玩家可以在任意签到点进行签到，每个玩家之间的签到互不影响，但是同一个玩家对同一个点进行签到一天最多只能签到五次，每次签到间隔必须大于十分钟。设计一个实时签到服务，接口接收参数 playerId, placeId，返回这次签到是否成功。

* 可用字典存储 playerId, placeId 对应的最后一次签到时间与今天内的签到次数
  + 进一步优化，可先用 playerId 作为一级 key 获取 placeId 到 value 的字典，再用二级字典加锁进行签到。这样一级字典可以使用读写锁，并发进行不同 player 的签到操作。考察设计数据结构、锁的能力。

## 查找满足条件的数对数量

给出两列数字，第一列数字从上到下是 1, 2, 3, … , n，第二列数字是 1, 2, 3, … , m。从左边一列和右边一列各选一个数字组成数对，求出有多少对数字满足这两个数字之和能被5整除。

* 最简单暴力的方法是两个 for 循环暴力枚举满足条件的数对，复杂度为O(n^2)
* 左边选出数字是a，右边选出数字是b，a+b能被5整除相当于b被5整除的余数是5-a。所以枚举左边一列的数字a，就可确定满足条件的b被5整除的余数是多少，然后可O(1)算出有多少个这样的数字b。Cal(b, m) = (m+4)/b + (b%5 <= m%5? 1: 0)。复杂度为O(n)
* 在上一步的基础上可以再次优化，a的本质是求能与 a%5 配对的b数量是多少，所以1、6、11、... 配对的数量都是相同的，求数量可以优化为 for(i := 1; i<=5; i++) ans += Cal(i, n) \* Cal(5-i, m)，复杂度O(1)

## 冒泡排序算法

给出一列1-10的乱序数字，请用冒泡排序的算法输出从小到大的数字序列

<?php

$a = array(1,3,7,2,8,10,9,6,5,4);

for ($i = 0; $i < 9; $i++) {

for($j = 0; $j < 9-$i; $j++) {

if ($a[$j+1] < $a[$j]) {

$tmp = $a[$j];

$a[$j] = $a[$j+1];

$a[$j+1] = $tmp;

}

}

}

print\_r($a);

## 两个数组求交集

描述一： （直截了当）

两个无序数组，同一数组中无重复元素，求两数组的交集。比如A={6，2，4，1}，B={2，9，4，3}，那么A&B={2，4}。输入为两个无序数组, 输出为两数组的交集。可以给出多个方案，并分别给出每个方案的时间复杂度。

描述二：（婉转）

给两个数组（不强调说是无序，而是给出一个数组例子让面试者观察），求这个数组的交集。比如A={6，2，4，1}，B={2，9，4，3}，那么A&B={2，4}。可以给出多个方案，并分别给出每个方案的时间复杂度。

PS：同一个数组内无重复元素。一开始面试官不提出，等待面试者发现然后提问。

思路：

1.采用遍历或者枚举的办法。用A数组里面的所有数去匹配B数组里面的数。假设两个数组的大小都是n，那么这种遍历的时间复杂度为O(n^2)。

同

2.预处理——先对这两个数组排序。一般情况下，数组排序的算法时间复杂度最低是O(nlogn)。当整数范围k较小时，用计数排序时间复杂度为O(n)。

PS：排序代码可以引导面试者写个O(nlogn)的排序（快速排序也可以刷掉大量水平一般的）。

然后，对两个从小到大排好序的数组A和B分别设置下标i和j，i和j从0开始，比较A[i]和B[j]。相等则找到交集的一个元素，i和j都自增；A[i]>B[j]则j自增；A[i]<B[j]则i自增。遍历下去，直到i和j中任一个到达数组尾。这一步的时间复杂度为O(n)。

3.将数组A hash到hash table中，然后继续将数组B hash到hash table中，如果发生碰撞则统计加1，最后可以得出数组的交集。时间复杂度也就是hash所有元素的复杂度O(n)。

4.方案2 的变形：先排其中一个数组，得到一个有序数组，O(nlogn)。然后遍历另一个无序数组的每一项，遍历的每一趟用二分的方法看有序数组是否存在相同项，O(nlogn)。总体也是O(nlogn)。

## 矩形平移

描述一：

一个M＊N的大矩形，里面有一个m＊n的小矩形（M>m，N>n，m和n都为奇数），小矩形中心坐标为（x1，y1）。小矩形内每个位置为1，而大矩形的其他位置为 0。 现在给出新的中心点（x2，y2），把小矩形 “剪切并粘贴” 到新的位置上。如果越界，就在另一头补全。例子如下，大矩形M＝5，N＝6；小矩形m＝3，n＝3；原来中心坐标（2，2），新中心坐标（4，5）。坐标从0开始，即左上角的位置坐标是（0，0）。

0 0 0 0 0 0 1 0 0 0 1 1

0 1 1 1 0 0 0 0 0 0 0 0

0 1 1 1 0 0 ——> 0 0 0 0 0 0

0 1 1 1 0 0 1 0 0 0 1 1

0 0 0 0 0 0 1 0 0 0 1 1

请写代码，输入为M，N，m，n，x1，y1，x2，y2，输出为把小矩形“剪切并粘贴”后的大矩形（输出格式可参照上图）。

描述二（难度比描述一稍大）：

一个M＊N的大矩形，里面有一个m＊n的小矩形（M>m，N>n）。小矩形内每个位置为1，而大矩形的其他位置为 0。现在需要平移小矩阵，水平方向平移 x 个单位（正数向右负数向左），竖直方向平移 y 个单位（正数向下负数向上）。如果越界，就在另一头补全。

例子如下，左边大矩形中的小矩形水平方向平移 +3，竖直方向平移 +2：

0 0 0 0 0 0 1 0 0 0 1 1

0 1 1 1 0 0 0 0 0 0 0 0

0 1 1 1 0 0 ——> 0 0 0 0 0 0

0 1 1 1 0 0 1 0 0 0 1 1

0 0 0 0 0 0 1 0 0 0 1 1

请写代码，输入为 matrix（二维数组），x，y，无输出，函数执行完后 matrix 应为把小矩形平移后的大矩形，要求在 matrix 上原地修改，不能开辟新的数组空间。矩阵中的值只能为0或1。

思路：关键是越界的处理，可用求模解决。

参考代码1：

#include<iostream>

using namespace std;

int p[1000][1000];

int main() {

int M, N, m, n, x, y, x2, y2;

cin>>M>>N>>m>>n>>x>>y>>x2>>y2;

for (int i = 0; i != M; i++) {

for (int j = 0; j != N; j++) {

p[i][j] = 0;

}

}

int xmin = x - (m-1)/2;

int xmax = x + (m-1)/2;

int ymin = y - (n-1)/2;

int ymax = y + (n-1)/2;

int x\_delta = x2 - x;

int y\_delta = y2 - y;

for (int i = xmin; i != xmax+1; i++) {

for (int j = ymin; j != ymax+1; j++) {

int i\_next = (i + x\_delta)%M;

int j\_next = (j + y\_delta)%N;

p[i\_next][j\_next] = 1;

}

}

for (int i = 0; i != M; i++) {

for (int j = 0; j != N; j++) {

cout<<p[i][j]<<" ";

}

cout<<endl;

}

}

参考代码2：

#include <stdio.h>

int M, N, m, n, cx, cy;

int map[10000];

int adjust(int a, int b) {

if(a<0) a+=b;

if(a>=b) a-=b;

return a;

}

void set(int x, int y) {

int a = (x \* N + y) / 8;

int b = (x \* N + y) % 8;

map[a] |= 1<<b;

}

int get(int x, int y) {

int a = (x \* N + y) / 8;

int b = (x \* N + y) % 8;

return (map[a] & (1<<b))? 1: 0;

}

int main() {

scanf("%d%d%d%d%d%d", &M, &N, &m, &n, &cx, &cy);

int minx = adjust(cx - m/2, M);

int maxx = adjust(cx + m/2 + 1, M);

int miny = adjust(cy - n/2, N);

int maxy = adjust(cy + n/2 + 1, N);

for(int x=minx; x != maxx; x= (x+1>=M? 0: x+1))

for(int y=miny; y!= maxy; y= (y+1>=N? 0: y+1))

set(x, y);

for(int x=0; x<M; x++) {

for(int y=0; y<N; y++) {

printf("%d ", get(x, y));

}

printf("\n");

}

}

扩展加深1：假如输入的是位图，并且小矩形每个位置不再单纯是1（比如是0-255中某个值），还是类似的把小矩形 “剪切并粘贴” 到新的位置上。

思路：开辟适当大小的新空间，把原来小矩形的信息缓存下来，同时把原位图的小矩形信息清空。然后再把缓存好的信息“复制”到原位图。

扩展加深2：在扩展加深1的基础上，要求在原位图上面操作。

PS：补充条件：M>>m，N>>n。防止小矩形太大出现的重叠问题，比如，假如中心向右下移动，小矩形太大，可能会出现新小矩形的右下角越界后从左上角位置进入原来小矩形内部。

思路：要求在原位图上面操作，则需要保证先”平移”的值不能覆盖了原图，参考C语言memmove()函数通过从字符串最后一个字符开始，从后往前移动，解决内存重叠问题的思路。因此，跟据原来中心和新中心的相对位置，判断小矩形的移动方向，假如往右下移动，则小矩形从右下角作为起始点开始逐个元素移动。越界的处理，同样可用求模解决。还有这考察了对位图的处理。

## 数组随机打乱

（适合当面面试，不适合网络面试）

假设有一个一维数组，现在需要把数组中的元素随机打乱。数组内元素类型自己定。

考察点：

1、是不是考虑到数组元素移动的代价

2、打乱后的元素，是不是真的随机分布

void random(DataType a[], int n)

{

int index, tmp, i;

srand(time(NULL));

for(i=n-1; i>0; i--)

{

index=rand()%i;

tmp=a[i];

a[i]=a[index];

a[index]=tmp;

}

}

进阶：限定条件，数组元素移动代价很高。引导其从访问数组层面来做（数组元素本身不动，迭代器随机取元素），写一个随机的迭代器（最好用面向对象的语言写）

考察点：

1、迭代器模式

2、为了随机访问，内部实现的数据结构设计

## 从数组里找出出现一次的数字

一、将1到100的数字放入长度为99的数组里，一个数字只出现一次，找出1到100之间没有出现在数组的数字。

思路：1到100的数字总和减去数组里所有数字的之和即为所求，即5050 - sum(array)

关键：不开辟额外空间

二、将1到101的数字放入长度为199的数组里，有一个数字出现一次，有一个数字没有出现，其余的数字都出现两次，找出出现一次的那个数字。

思路：用0和数组里的数字异或一遍即为所求

原理：相等的两个数异或结果为0

## 区域颜色填充算法

思想：类似的win画图工具的油漆桶，给出多边形边界和一个着色点，将与着色点连通的区域染成指定颜色。（面试官可以画图说明）

题目：给出一个二维数组（0代表空白，1代表边界）和一个起始点坐标（a，b），把与起始点连通的区域变成“2”。PS: 左上角坐标是(0,0)。

输入样例：

4 4

0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0

0 1 0 1 1 1 0 0

0 1 0 0 0 0 1 0

0 0 1 0 0 0 1 0

0 1 0 0 0 1 0 0

0 0 1 1 1 0 0 0

0 0 0 0 0 0 0 0

输出样例：

0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0

0 1 2 1 1 1 0 0

0 1 2 2 2 2 1 0

0 0 1 2 2 2 1 0

0 1 2 2 2 1 0 0

0 0 1 1 1 0 0 0

0 0 0 0 0 0 0 0

思路：基于深度优先或者广度优先从着色点开始搜索，判断4邻域（上下左右）是否是边界，不是边界继续搜索，直至遇到边界。基于深度优先来实现时，尽量不要用递归，因为递归多次嵌套，输入图像比较大的时候容易溢出。

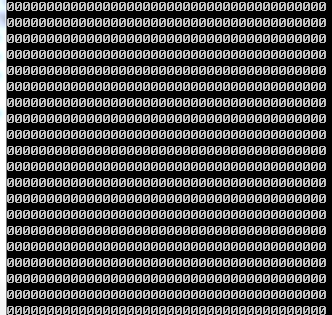
1、基于栈的实现

2、基于队列的实现

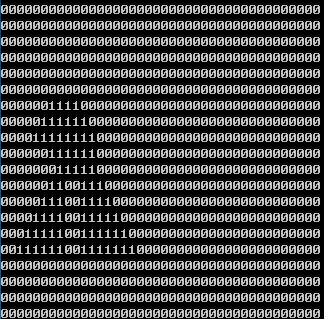
## 模拟图形界面图标点选过程(C语言版)

思想:模拟图形界面操作系统中，鼠标点击某图标后，图标被显示成选中状态的过程.

1 在这里用一个已给出的M\*N的值为0的矩阵模拟窗口



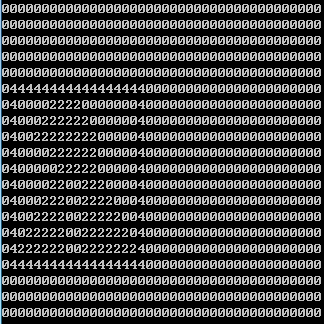
2 用一个值为1的值形成的图形表示某个位于窗口内图标(不贴边)



3 需要用户输入一个二维坐标，表示鼠标点击位置



4 如果点击位置在图形之上(即值为1)，则图形被选中(值由1变为2)，并且加以框选的形式(用4表示)



题目: 给出一个40\*20的矩阵，其值为0,表示一个空窗口，其中有一块值为1的区域，表示位于窗口中的某图标,图标不贴边.

接受用户输入一个位置(x,y)，给出一个算法，计算出如果(x,y)位于图标之上，则将图标值标为2,并计算出选中框(其值为4表示)

#define win\_width 40

#define win\_height 20

static int window[win\_width][win\_height];

//以下函数不需实现

extern makeWindow(); //将矩阵值赋全0,表示窗口

extern makeIcon(); //将图标值赋1,表示图标

extern showWindow(); //显示窗口,即把40\*20矩阵输出

//以上函数不需实现

main()

{

makeWindow();

makeIcon();

mouseClick(x,y);//需要实现

showWindow();

}

参考代码:

static icon\_start\_coor\_x = win\_width;

static icon\_start\_coor\_y = win\_height;

static icon\_end\_coor\_x = 0;

static icon\_end\_coor\_y = 0;

void coorCalculate(int x, int y)

{

if (x >= win\_width || x < 0) {

return;

}

if (y >= win\_height || y < 0) {

return;

}

if (window[x][y] == 1) {

window[x][y] = 2;

icon\_start\_coor\_x = x < icon\_start\_coor\_x ? x : icon\_start\_coor\_x;

icon\_start\_coor\_y = y < icon\_start\_coor\_y ? y : icon\_start\_coor\_y;

icon\_end\_coor\_x = x > icon\_end\_coor\_x ? x : icon\_end\_coor\_x;

icon\_end\_coor\_y = y > icon\_end\_coor\_y ? y : icon\_end\_coor\_y;

coorCalculate(x + 1, y);

coorCalculate(x, y + 1);

coorCalculate(x - 1, y);

coorCalculate(x, y - 1);

}

return;

}

void makeIconFrame()

{

int frame\_start\_coor\_x = icon\_start\_coor\_x - 1;

int frame\_start\_coor\_y = icon\_start\_coor\_y - 1;

int frame\_end\_coor\_x = icon\_end\_coor\_x + 1;

int frame\_end\_coor\_y = icon\_end\_coor\_y + 1;

int frame\_width = frame\_end\_coor\_x - frame\_start\_coor\_x;

int frame\_height = frame\_end\_coor\_y - frame\_start\_coor\_y;

for(int i = 0; i < frame\_width; i++)

{

window[frame\_start\_coor\_x+i][frame\_start\_coor\_y] = 4;

window[frame\_start\_coor\_x+i][frame\_end\_coor\_y] = 4;

}

for(int j = 0; j < frame\_height ; j++)

{

window[frame\_start\_coor\_x][frame\_start\_coor\_y+j] = 4;

window[frame\_end\_coor\_x][frame\_start\_coor\_y+j] = 4;

}

}

void mouseClick(int x, int y)

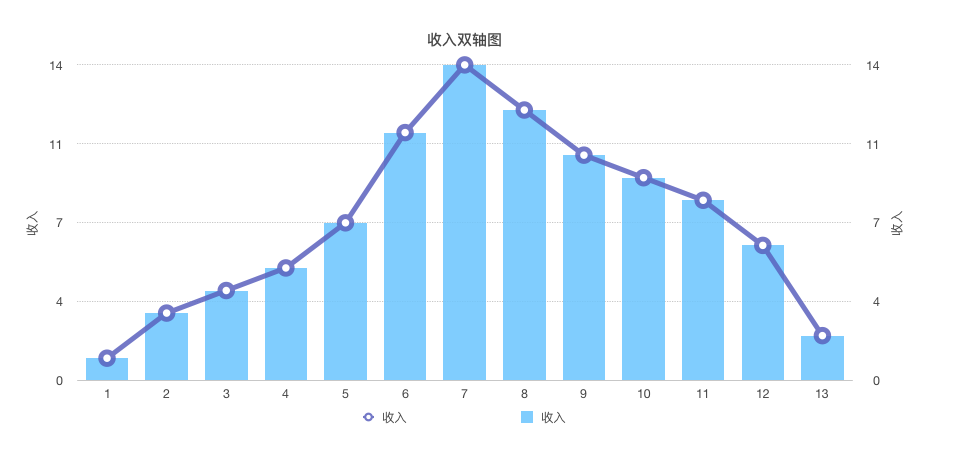
{

coorCalculate(x,y);

makeIconFrame();

}

## 在一个先增后减的序列中寻找第k小



一个长度为n的序列，先递增后递减，最大值在n/2(向下取整)处，现在要求找出第 k小的值。

条件:

n > 0, k > 1

可选择条件：

k <= n : 如果不给出该条件，看面试者能否考虑到这种情况

**输入**

n k

[此处输入序列...]

**输出**

第k小

输入样例

11 7

1 8 12 12 21 31 26 22 20 12 4

输出

20

输入样例

1 1

2

输出

2

问题抽象：在一个递增和一个递减的序列中寻找第k小 --> 两个有序序列中找第k小

### 思路一：排序，复杂度O(nlogn)

### 思路二：遍历，两个指针指向序列头尾，复杂度O(n)

|  |
| --- |
| **//arr为题中的序列**  **int** **findByTwoPoint**(**int** k, **int** n, **int** arr[]) {  **int** head = 0, tail = n - 1, mid = n / 2;  **for** ( ; head <= mid && tail >= mid && k > 1; k --) {  arr[head] < arr[tail] ? head++ : tail--;  }    **if** (k == 1) **return** min(arr[head], arr[tail]);  **if** (head <= mid) **return** arr[head + k];  **return** arr[tail - k]; } |

### 思路三：二分，复杂度O(logn)

将序列拆分成两个序列，前半部分为序列a[al...ar])，后半部分为序列d[dl...dr]，中间的数字归为前半部分。

分别取两序列的中间位置am = (al + ar) / 2, dm = (dl + dr) / 2

计序列a的前半段和d的后半段数量和为half = (am - al + 1) + (dr - dm + 1)

1. 当harf > k:

* 若a[am] < d[dm]， 则d的前半部分序列的数值一定不会小于第k小的数值，故可舍去d的前部分序列d[dl...dm], 此后只需从a[al...ar]和d[dm+1...dr]中搜索第k小。
* 若a[am] >= d[dm]， 则a的后半部分序列的数值一定不会小于第k小的数值，故可舍去a的后部分序列a[am...ar], 此后只需从a[al...am-1]和d[dl...dr]中搜索第k小。

2.当harf <= k:

* 若a[am] < d[dm], 则a的前半部分序列的数值一定不会大于第k小的数值，故可舍去a的前半部分，此后只需从a[am+1...ar]和d[dl...dr]中搜索第k - (am - al + 1) 小。
* 若a[am] >= d[dm], 则d的后半部分序列的数值一定不会大于第k小的数值，故可舍去d的后半部分，此后只需从a[al...ar]和d[dl...dm-1]中搜索第k - (dr - dm + 1) 小。

递归实现：

|  |
| --- |
| **int** **findByDeepBinarySearch**(**int** al, **int** ar, **int** dl, **int** dr, **int** k, **int** arr[]) {  **if**(al > ar) **return** arr[dr - k + 1];  **if**(dl > dr) **return** arr[al + k - 1];  **int** am = (al + ar) / 2, dm = (dl + dr) / 2;  **if** ((am - al + 1) + (dr - dm + 1) > k) {  **if** (arr[am] < arr[dm]) **return** dfs(al, ar, dm + 1, dr, k, arr);  **return** dfs(al, am - 1, dl, dr, k, arr);  } **else** {  **if** (arr[am] < arr[dm]) **return** dfs(am + 1, ar, dl, dr, k - (am - al + 1), arr);  **return** dfs(al, ar, dl, dm - 1, k - (dr - dm + 1), arr);   } } |

循环实现:

|  |
| --- |
| **int** **findByLoopBinarySearch**(**int** k, **int** n, **int** arr[]) {  **int** mid = n / 2;  **int** al = 0, ar = mid, dl = mid + 1, dr = n - 1;  **while**(al <= ar && dl <= dr) {  **int** am = (al + ar) / 2, dm = (dl + dr) / 2;  **if** ((am - al + 1) + (dr - dm + 1) > k) {  **if** (arr[am] < arr[dm]) dl = dm + 1;  **else** ar = am - 1;  } **else** {  **if** (arr[am] < arr[dm]) {  k -= (am - al + 1);  al = am + 1;  } **else** {  k -= (dr - dm + 1);  dr = dm - 1;  }  }  }  **if** (al > ar) **return** arr[dr - k + 1];  **return** arr[al + k - 1]; } |

## redis中key迁移

### 问题描述一

视频的播放次数缓存在 redis 中，key 的格式为 video:{id}（每个视频的id不一样）, 值为整数。有个线上服务AServer, 客户端向AServer发送获取视频播放次数的请求时，AServer会在redis执行Get video:{id}来获取视频的播放次数，当客户端观看视频时,Aserver会在redis 中执行Incr video:{id} 使播放次数加1。还有一个线下服务BServer, Bserver 周期把video:{id}的值写回数据库表t\_media中，现在希望将上述的video:{id}改为media:{id}, 如何实现使得用户感知不到这次修改，也不会丢失数据。



### 问题描述二

线上部署了一个视频播放次数的服务，实现两个功能：获取某个视频的播放次数、增加1次某个视频的播放次数。

此服务由以下组成：

1. MySQL。MySQL 持久存储视频播放次数。
2. Redis。Redis 保存着视频的播放次数，Redis 是内存数据库，用于快速缓存。
3. HttpApiServer。HttpApiServer 是部署在多台服务器上的分布式无状态 HTTP 服务器，提供两个 API
   1. 获取视频播放次数，通过调用 Redis 的 GET video:{id} 实现。
   2. 增加一次视频播放次数，通过调用 Redis 的 INCR video:{id} 实现。
4. OfflineServer。OfflineServer 是只在一台服务器部署的线下进程，每隔1秒会遍历一遍 Redis 的 video:{id}，获取播放次数，更新 MySQL。

现在希望将 Redis 的 video:{id} 重命名为 media:{id}，请设计一系列的 HttpApiServer 与 OfflineServer 兼容更新操作，使得 HttpApiServer 的 API 调用者感知不到这次修改，也不会丢失数据。



### 注意点

不能用锁，这个条件可以先不说，看面试者怎么回答。

加不能用锁的条件是为了增加难度，还有一点是因为播放次数增加和获取是个高频的操作，如果加锁，影响性能。

### 方案一

步骤1: 更新Aserver 使AServer以video:{id} 和media:{id}中较大的值作为播放次数，播放视频的时候同时incr video:{id} 和 incr media:{id}

步骤2: 更新Bserver使BServer以video:{id} 和media:{id}中较大的值作为播放次数

步骤3: 重命名video:id 为media:id (redis中 RENAME video:{id} media:{id})

步骤4: 更新Aserver使Aserver只读写key media:{id}

步骤5: 删除所有的video:{id}

### 方案二

步骤1：更新Aserver 使播放视频时同时incr video:{id} 和 incr media:{id}

步骤2：copy video:{id} media{id}

步骤3: 更新BServer只读media:{id}

步骤4：更新AServer只读写media:{id}

步骤5：删除video:{id}

## 前端：实现简单轮播图

要点：

1.现场coding，面试者自己确定演示方式，可选plunker（熟练调试方式）

2.要求适应图片资源可变（src和count）

3.扩展控制操作（点选，悬停）

问题：怎么避免控制操作和轮播动画冲突

autoTimer：轮播时间间隔

waitingTimer：控制操作结束

private refreshWaitingTimeout() {

this.stopWaitingTimeout();

this.startWaitingTimeout();

}

private startWaitingTimeout() {

if (!this.timeoutId) {

this.timeoutId = setTimeout(() => {

this.startAutoTimer();

}, this.timeForWaitingMs);

}

}

private stopWaitingTimeout() {

if (this.timeoutId) {

clearTimeout(this.timeoutId);

this.timeoutId = null;

}

}

private startAutoTimer() {

this.intervalId = setInterval(() => {

this.selected = (this.selected + 1) % this.items.length;

}, this.interval);

}

private stopAutoTimer() {

if (this.intervalId) {

clearInterval(this.intervalId);

this.intervalId = null;

}

}

## 命令行扫雷

题目：

实现扫雷游戏。

该游戏在一个10\*10的棋盘中，随机分布着15个雷。

棋盘由以下符号表示：

‘#’ 表示该格子未揭开

‘ ‘ 表示该格子附近8个格子没有雷

‘1-8’ 数字表示该格子附近有多少个雷

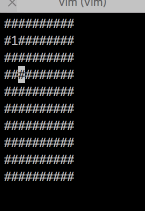
‘\*’ 表示该格子是地雷

玩家可以按 WASD 进行上下左右移动光标，按空格揭开格子。当揭开的格子是地雷时，打印’you lose’；当揭开格子附近有地雷时，显示附近有多少个雷；当揭开格子附近无地雷时，显示为空格并上下左右四周扩散；当揭开格子后剩余格子全是地雷时，打印 ‘you win’。

实现扫雷程序完成 生成棋盘、响应用户操作、更新界面 的一个完整程序。

例子：https://media.giphy.com/media/xUA7b5A0LyBXULNdtu/giphy.gif

PS：现场面的话我一般给电脑敲代码，最后允许 debug。限定30分钟看候选人完成程度多少。



提供 API 接口与程序框架

#include <stdio.h>

#include <ncurses.h>

#include <unistd.h>

#include <cstring>

#include <time.h>

#include <stdlib.h>

using namespace std;

// 以下四个接口为初始化应用，可跳过

// initscr();

// noecho();

// curs\_set(1);

// endwin();

// move(x, y); // 把光标移动到 (x, y)

// mvprintw(x, y, "%s", str); // 与 printf() 类似，不同之处在于是从 (x,y) 坐标开始打印

// refresh(); // move 函数和 mvprintw 函数调用后不会立即生效，需调用此 refresh 生效

// ch = getch(); // 获取用户输入字符

// compile command

// g++ -lcurses main.cpp

int main (void) {

// init setting

initscr();

noecho();

curs\_set(1);

// do the stuff

mvprintw(10, 10, "hello");

move(5, 5);

refresh();

sleep(3);

endwin();

return 0;

}

参考代码：

#include <stdio.h>

#include <ncurses.h>

#include <unistd.h>

#include <cstring>

#include <time.h>

#include <stdlib.h>

#include <algorithm>

using namespace std;

#define ff(i, n) for (int i = 0, END = (n); i < END; i ++)

#define fff(i, n, m) for (int i = (n), END = (m); i <= END; i ++)

#define clr(a, b) memset(a, b, sizeof(a))

const int N = 10;

const int M = 15;

int mat[N + 3][N + 3];

bool vis[N + 3][N + 3];

int px, py;

int remain;

int status;

void init() {

status = 0;

remain = N \* N - M;

px = 1, py = 1;

clr(mat, 0);

clr(vis, 0);

srand(time(0));

ff (m, M) {

int x = rand() % 10 + 1;

int y = rand() % 10 + 1;

if (mat[x][y] == 0) {

mat[x][y] = -1;

fff(i, -1, 1) fff(j, -1, 1) if (i || j) {

if (mat[x + i][y + j] != -1) {

mat[x + i][y +j] ++;

}

}

} else {

m --;

}

}

}

void draw() {

fff (i, 1, 10) fff (j, 1, 10) {

if (!vis[i][j]) {

mvprintw(i - 1, j - 1, "#");

} else {

if (mat[i][j] == -1) {

mvprintw(i - 1, j - 1, "\*");

} else if (mat[i][j] == 0) {

mvprintw(i - 1, j - 1, " ");

} else {

mvprintw(i - 1, j - 1, "%d", mat[i][j]);

}

}

}

move(px - 1, py - 1);

if (status == 1) {

mvprintw(N, 0, "you win");

} else if (status == -1) {

mvprintw(N, 0, "you lose");

}

refresh();

}

int que[N \* N + 2];

int dir[][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

void bfs() {

int l = 0, r = 0;

que[r ++] = (px << 8) + py;

while (l < r) {

int now = que[l ++];

int x = now >> 8, y = now & 0xFF;

vis[x][y] = true;

remain --;

ff (i, 4) {

int vx = x + dir[i][0], vy = y + dir[i][1];

if (1 <= vx && vx <= N && 1 <= vy && vy <= N) {

if (vis[vx][vy] == false) {

vis[vx][vy] = true;

remain --;

if (mat[vx][vy] == 0) {

que[r ++] = (vx << 8) + vy;

}

}

}

}

}

}

void clip() {

if (mat[px][py] == -1) {

status = -1;

} else if (mat[px][py] > 0) {

remain --;

} else {

bfs();

if (remain <= 0) status = 1;

}

vis[px][py] = true;

}

int main (void) {

// init setting

initscr ();

noecho();

curs\_set(1);

init();

do {

draw();

if (status != 0) break;

char ch = getchar();

switch (ch) {

case 'w': px = max(1, px - 1); break;

case 's': px = min(N, px + 1); break;

case 'a': py = max(1, py - 1); break;

case 'd': py = min(N, py + 1); break;

case ' ': clip();

}

} while (true);

sleep(3);

endwin();

return 0;

}

候选人版本：

using namespace std;

void printMp(){

for(int i =0;i<10;++i){

cout<<"##########"<<endl;

}

}

int checkPos(int x,int y,vector<vector<int> >mp){

int count = 0;

if(x>0)

count+=mp[x-1][y];

if(y>0)

count+=mp[x][y-1];

if(x<9)

count+=mp[x+1][y];

if(y<9)

count+=mp[x][y+1];

if(x>0&&y>0)

count+=mp[x-1][y-1];

if(x>0&&y<9)

count+=mp[x-1][y+1];

if(x<9&&y<9)

count+=mp[x+1][y+1];

if(x<9&&y>0)

count+=mp[x+1][y-1];

return count;

}

void depSearch(int x,int y vector<vector<int> >&mp,vector<vector<int> >&screen){

if(screen[x][y]!='#')

return;

int c = checkPos(x,y);

if(c>0){

mvprintw(y, x, "%d", c);

screen[x][y] = '0'+c;

return;

}

mvprintw(y, x, " ");

screen[x][y] = ' ';

if(x>0)

depSearch(x-1,y,mp,screen);

if(y>0)

depSearch(x,y-1,mp,screen);

if(x<9)

depSearch(x+1,y,mp,screen);

if(y<9)

depSearch(x,y+1,mp,screen);

if(x>0&&y>0)

depSearch(x-1,y-1,mp,screen);

if(x>0&&y<9)

depSearch(x-1,y+1,mp,screen);

if(x<9&&y<9)

depSearch(x+1,y+1,mp,screen);

if(x<9&&y>0)

depSearch(x+1,y-1,mp,screen);

}

int main(){

vector<vector<int> >mp(10,vector<int>(10,0));

vector<vector<char> >screen(10,vector<char>(10,'#'));

for(int i=0;i<15;++i){

int x = rand()%10;

int y = rand()%10;

while(mp[x][y]){

x = rand()%10;

y = rand()%10;

}

mp[x][y]=1;

}

printMp();

int posx = 0;

int posy = 0;

move(0,0);

while(true){

ch = getch();

switch(ch){

case 'w':

if(posy>0) --posy;

move(posx,posy);

break;

case 's':

if(posy<9) ++posy;

move(posx,posy);

break;

case 'a':

if(posx>0) --posx;

move(posx,posy);

break;

case 'd':

if(posx<9) ++posx;

move(posx,posy);

break;

case ' ':

if(mp[posy][posx]){

cout<<endl<<"you lose!"<<endl;

}else if(screen[posy][posx]=='#'){

depSearch(posy,posx,mp,screen);

}

}

refresh();

}

}

## 传送塔 Part I (Transportation Tower I)

输入

1. 第一行输入一个整数 N (1 <= N <= 100)，表示传送塔层数
2. 接下来 N 次以下描述的循环输入
   1. 输入一行 Ex Ey Tx Ty，(1 <= Tx <= Ex <= 10000, 1 <= Ty <= Ey <= 10000)，分别代表每一层的矩形地图的长和宽, 从别的层传送过来时的 坐标 x, y
   2. 接下来 Ex 行循环输入, (取值见下说明)
      1. U0 U1 U2 … U(Ey-1)
3. 输入一行 X Y，表示玩家起始在 第一层 的方格坐标，可以保证该方格一定是普通方格（见下说明)，X = 0, Y =0 表示左上角坐标

输出

1. 如果玩家从起点走到终点方格，则用一行输出只由 ‘w’, ‘s’, ‘a’, ‘d’ 组成的字符串，含义见下说明。
2. 如果玩家不能从起点走到终点方格，则输出 “DEAD”

说明:

1.

U = 0 普通方格（即可通行）

U = -1 阻碍物方格，玩家不能进入阻碍物方格

U = k, (1 <= k <= N) 传送方格，走到传送方格就会传送到塔的第 k 层

U = 127，终点方格

2.

对于每一层 (Tx, Ty) 保证不会是阻碍物方格

3.

‘w’ : (x, y) -> (x, y-1)

‘s’ : (x, y) -> (x, y+1)

‘a’ : (x, y) -> (x-1, y)

‘d’ : (x, y) -> (x+1, y)

题解链接：<https://docs.google.com/document/d/1G6df3JB37-dUau5B6K7ulufxtVbRjx92TU1iY2U8pRU/edit>

## 微博系统的存储应该如何设计

1. 每日活跃（有使用行为）用户 100W
2. 总用户数 5000W
3. 平每小时发微博 3000 条以上（有高峰）
4. 平均每个人 follow 20 个以上的人（有的人特别多）
5. 平均每个人被 10 个人 follow（有的人特别多）
6. 功能，刷新关注的人发的最新微博
7. 功能，发微博

## How to wait for a response in a request to stream system

Sending http request to server whose backend is designed as stream system, how to wait for something finished and then response to client ?

1. Client polling request
2. Send notification through persistent connection

## 生成拼音索引问题

### 背景

有一个搜索系统，可以根据拼音搜索出对应名称的用户。搜索支持从名称中间开始匹配，例如，对于名为“周杰伦”的用户，以下拼音的全部或者任意前半部分都可以搜索出来：

zhoujielun

jielun

lun

为了支持这个功能，需要对用户名称的拼音建立索引。索引方式已经确定，以“周杰伦”为例，这个名称会建立以下索引记录：

zhoujielun

jielun

lun

这三条记录都对应同一个用户ID。在搜索的时候，使用字符串的前序匹配，即可找出所有符合条件的用户ID。

### 需求

现在你的工作是，实现一个函数，把用户的名称转换成索引中的拼音列表。

参数输入：用户名称，如“周杰伦”

返回值输出：拼音列表，如“zhoujielun”，“jielun”，“lun”

在系统中已经提供了一个工具函数，该函数根据字符返回对应的拼音：

|  |
| --- |
| **const** wchar\_t\* **GetPinyin**(**wchar\_t** ch); |

返回的是一个字符串，字符串中包含该字符的所有拼音，每个拼音之间以逗号（,）分隔。例如“长”字会返回“chang,zhang”；“江”字会返回“jiang”。如果传入的字符不存在对应拼音，该函数返回nullptr。

注意：

* 如果名称中存在多音字，那么所有拼音组合都要返回。例如“长江”需要返回：

changjiang

zhangjiang

jiang

* 用户可以自由填写任意名称，请考虑名称中可能出现的各种情况。
* 如果字符没有对应的拼音，可以忽略该字符。例如“在2018年”应返回：

zainian

nian

* 函数的名称、参数和返回值自定。可以使用你最熟悉的语言来写这个函数，但同时要用这个语言把GetPinyin的声明重新写一遍。

### 分析

这道题的考察点主要是字符串处理以及数组乘法。一般思路：

1. 获取每个字符的拼音列表，形成二维数组。
2. 从一元数组的第一个元素开始遍历数组，对每个元素的拼音列表执行乘法，生成结果。
3. 继续从一元数组的下一个元素开始遍历数组，重复执行以上乘法。
4. 把所有结果串接起来。

最终结果需要的空间复杂度（Ln为第n个字符的拼音数量）：

|  |
| --- |
| L1 \* L2 \* L3 \* ... \* Ln +  L2 \* L3 \* ... \* Ln +  L3 \* ... \* Ln +  ... +  Ln |

附加的开放性问题：

1. 如果字符串很长而且有很多多音字，最终结果很可能会耗尽内存。有什么方法可以避免？