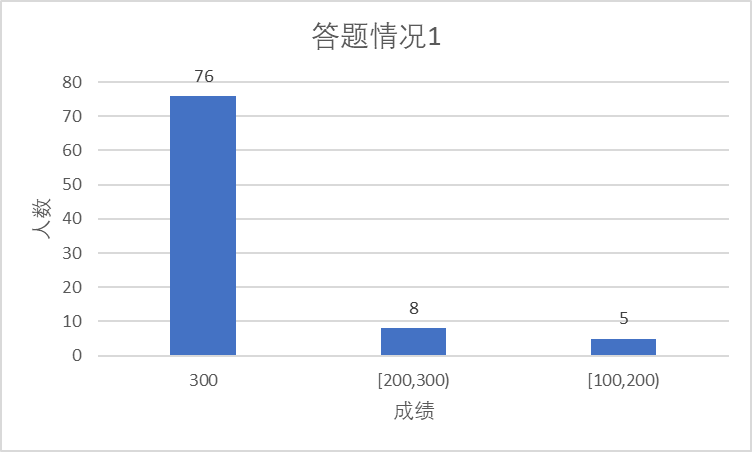
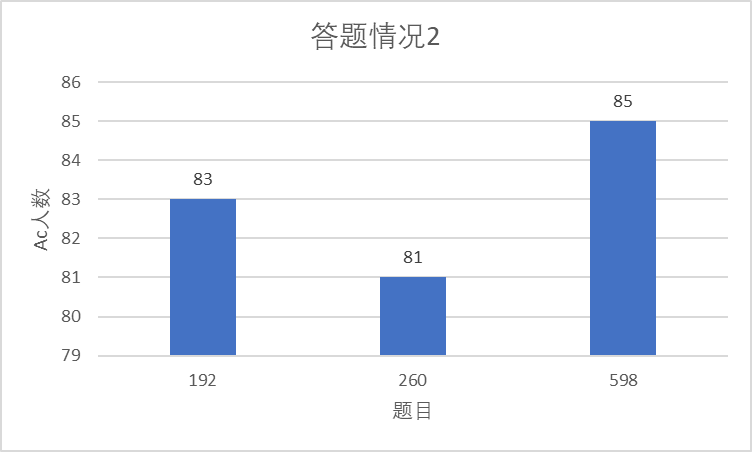
**2023.11.28\_课后作业**

**完成情况**





**题目详解**

**#192** [哥德巴赫猜想](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/192.html)

【解题思路】

1）通过函数判断一个数是否为素数

2）检查是否存在素数对

示例代码

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int prime(int n)

{ //判断素数的函数

if (n == 1) //1不是素数

return 0;

int temp = (int)sqrt(n);

for (int i = 2; i <= temp; i++)

if (n % i == 0)

return 0;

return 1; //n是素数

}

int main(){

int n;

scanf("%d",&n);

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

if(prime(i) && prime(n-i)){

printf("%d=%d+%d",n,i,n-i);

break;

}

}//for

system("pause");

return 0;

}

**#260** [**二分查找**](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/260/eno/1204.html)

【解题思路】

1）对数组元素做升序排列

2）使用二分查找算法在有序数组中查找数字 m。在每次迭代中，计算中间索引 mid，如果中间值等于 m，则记录比较次数 num 并输出结果。

3）如果中间值小于 m，则在右半部分继续查找，将 low 更新为 mid+1。

4）如果中间值大于 m，则在左半部分继续查找，将 high 更新为 mid-1。

示例代码

#include <stdio.h>

#include <stdlib.h>

int main(){

int n,m;

scanf("%d\n%d",&n,&m);

int a[n];

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

scanf("%d",&a[i]);

for(int i = 0;i<n-1;i++) //冒泡

for(int j = 0;j<n-i-1;j++)

if(a[j] > a[j+1]){

int tmp = a[j];

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

a[j+1]=tmp;

}

int num=0,low=0,high=n-1,mid; //num存储比较次数

int tag = 0;

while(low<=high){ //循环找数字

mid = (low+high)/2;

if(a[mid] == m){

num++;

tag = 1;

break;

}

else if(a[mid] < m){

num++;

low = mid+1;

}

else{

num++;

high = mid-1;

}

}//while

if(tag == 1)

printf("%d\n%d",mid,num);

else

printf("-1\n%d",num);

system("pause");

return 0;

}

**#598** [**string(直接提交版）**](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/598/eno/1204.html)

**【解题思路】**

注：排列不会改变字符串中每个字符的个数，只有当两个字符串每个字符的**个数均相等**时，一个字符串才是另一个字符串的排列。

1）记 s1的长度为 n，我们可以遍历 s2中的每个长度为 n的子串，判断 子串 和 s1中每个字符的个数是否相等，若相等则说明该子串是 s1的一个排列。

2）使用 数组1 统计 s1 中各个字符的个数，用 数组2 统计当前遍历的子串中各个字符的个数。

3）滑动窗口每向右滑动一次，就多统计一次进入窗口的字符，少统计一次离开窗口的字符。然后判断两个数组是否相等~

示例代码

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int equals(int\* cnt1, int\* cnt2) {

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

if (cnt1[i] != cnt2[i]) {

return 0;

}

}

return 1;

}

int check(char\* s1, char\* s2) { //排列不会改变字符串中每个字符的个数，所以只有当两个字符串每个字符的个数均相等时，一个字符串才是另一个字符串的排列。

int n = strlen(s1), m = strlen(s2);

if (n > m) {

printf("false");

return 0;

}

int cnt1[26], cnt2[26];

memset(cnt1, 0, sizeof(cnt1));

memset(cnt2, 0, sizeof(cnt2));

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

cnt1[s1[i] - 'a']++;

cnt2[s2[i] - 'a']++;

}

if (equals(cnt1, cnt2)) {

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

printf("%c",s2[i]);

return 0;

}

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

cnt2[s2[i] - 'a']++;

cnt2[s2[i - n] - 'a']--; //滑动窗口每向右滑动一次，就多统计一次进入窗口的字符，少统计一次离开窗口的字符

if (equals(cnt1, cnt2)) {

for (int j = i-n+1; j <= i; j++)

printf("%c",s2[j]);

return 0;

}

}

printf("false");

return 0;

}

int main(){

char s1[200],s2[200];

gets(s1);

gets(s2);

// scanf("%s\n%s",s1,s2);

check(s1,s2);

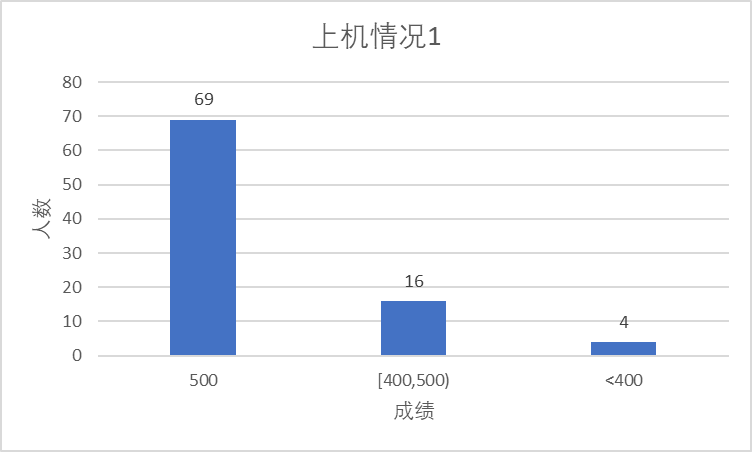
system("pause");

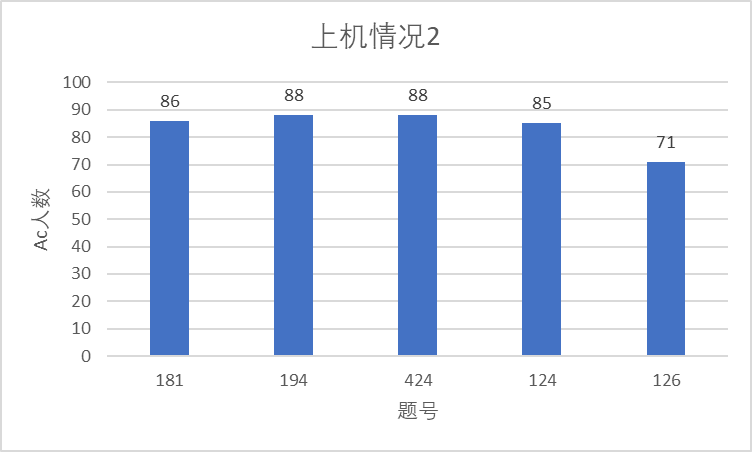
return 0;

}

**2023.12.05\_上机题**

**完成情况**





**题目详解**

**#181.** [奇怪的数列](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/181/eno/1207.html)

【解题思路】

1）递推关系为Fn = (A \* F(n-1) + B \* F(n-2)) % 7

2）n 等于 1 或 2，函数返回1

3）使用迭代方法计算数列的第n项，通过**循环**逐步计算

示例代码

#include<stdio.h>

#include <stdlib.h>

int f\_n(int a,int b,int n)

{

if(n==1||n==2)

return 1;

else

{

int f1=1,f2=1,fn;

for(int i=3;i<=n;i++){ //每次循环算

fn=((a\*f2)+(b\*f1))%7;

f1=f2;

f2=fn;

}

return fn;

}

}

int main()

{

int A,B,n;

scanf("%d %d %d",&A,&B,&n);

printf("%d",f\_n(A,B,n));

system("pause");

return 0;

}

**#194.** [进制转换](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/194/eno/1207.html)

【解题思路】

1）声明数组存储转换后的结果

2）**余数** 为对应进制的每一位，大于等于10，将其转换为相应的字符形式

3）将存储的字符**逆序输出**

示例代码

#include <stdio.h>

#include <stdlib.h>

int main(){

int d,h;

scanf("%d%d",&d,&h);

char a[100];

int k,i=0;

while(d){

k = d%h;

if(k>=10)

a[i] = k+55;

else

a[i] = k+48;

d /= h;

i++;

}

for(int j=i-1;j>=0;j--)

printf("%c",a[j]);

system("pause");

return 0;

}

**#424.** [算数比赛](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/424/eno/1207.html)

**【解题思路】**

1）题目要求：两个数的和，只能出现一次。如5， 5=1+4； 5=2+3 只算一个

2）用一个数组存放两数之和，可以计算出现次数 或 标记为1

3）用一个数组存放出现的数，直接用作下标

示例代码

#include <stdio.h>

#include <stdlib.h>

int t[20001]={0},g[10001]={0};

// t[i]表示值为i的数在集合中两两相加出现了几次，g[i]表示值为i的数是否在集合中，1为在，0为不在

int main(){

int n,a[100],ans=0;

scanf("%d",&n);

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

scanf("%d",&a[i]);//读入

g[a[i]]=1;//在集合中赋值为1

}

for (int i=1;i<n;i++)//枚举

for (int j=i+1;j<=n;j++)

t[a[i]+a[j]]++;//被加出来

for (int i=1;i<=10000;i++)

if ( t[i]>0 && g[i])

ans++;//判断是否满足，满足ans++

printf("%d\n",ans);

system("pause");

return 0;

}

**#124.** [滑雪](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/124/eno/1207.html)

【解题思路】

1）获取二维数组的行数和列数，以及数组的具体元素值。

2）通过嵌套的两个循环遍历每个数组元素，对每个元素调用递归函数。

3）该函数从当前坐标开始，递归地探索周围四个方向的元素，如果找到比当前元素值小的相邻元素，则继续向那个方向进行搜索，最终找到最长的递增路径长度。

4）通过**递归**来寻找递增路径，同时通过**全局变量max**来记录最大路径长度。

如果使用循环并不困难的话，最好使用循环哦

|  |  |  |
| --- | --- | --- |
|  | 循环 | 递归 |
| 优点 | 性能：循环通常比递归更高效，因为它们不会引入额外的函数调用开销和堆栈空间。  可控制：更精确地控制迭代次数和流程 | 简洁性：递归可以使代码更加简洁和易读  问题分解：递归允许将问题分解为更小的子问题，理解和解决复杂问题 |
| 缺点 | 代码复杂性  问题分解：循环可能不如递归清晰 | 性能：如果调用层数比较深，每次都要创建新的变量，需要增加额外的堆栈处理，会对执行效率有一定影响，占用过多的内存资源。  堆栈溢出：如果递归深度太大，可能导致堆栈溢出 |

示例代码

#include <stdio.h>

#include <stdlib.h>

int max=0, ROW, COL, a[51][51];

void search( int x, int y, int length )

{ if(x<1||y<1||x>ROW||y>COL)

return;

length++;

if(length>max)

max=length;

if( a[x][y]>a[x-1][y] )

search(x-1, y, length );

if( a[x][y]>a[x+1][y] )

search(x+1, y, length );

if( a[x][y]>a[x][y-1] )

search(x ,y-1, length );

if( a[x][y]>a[x][y+1] )

search(x, y+1, length );

}

int main() {

scanf("%d %d",&ROW,&COL);

for(int i=1; i<=ROW; i++)

for(int j=1; j<=COL; j++)

scanf("%d",&a[i][j]);

for(int i=1; i<=ROW; i++) //逐点

for(int j=1; j<=COL; j++)

search(i, j, 0);

printf("%d",max);

system("pause");

return 0;

}

**#126.** [travel](http://yoj.ruc.edu.cn/index.php/index/problem/detail/pno/126/eno/1207.html)

【解题思路】

问题：在二维数组中寻找**从左上角到右下角**的路径，使路径上的元素和最大

1）获取二维数组的行数和列数，以及数组的具体元素值。

2）从左上角的起点开始递归地探索下、左、右三个方向的元素，同时记录路径上的元素和 。

3）递归深度遍历完一条路径后，通过**回溯**方式**撤销当前节点的标记**，继续探索其他可能的路径。

4）整个算法的核心是通过递归和回溯方式穷举所有可能的路径，找到其中元素和最大的路径。

示例代码

#include <stdio.h>

#include <stdlib.h>

int map[9][9][2];

short m, n;

int sum=-2000;

int mine=0;

void explore(int x, int y){

if( x<1 || x>n || y<1 || y>m )

return; //临界

if( map[x][y][1]==1 )

return; //该点已标记

mine+=map[x][y ][0];

map[x][y][1]=1; //已探索

if( x==n&&y==m ) // 当前探索完毕

if(sum<mine)

sum=mine;

explore(x+1, y); // 下

explore(x, y-1); // 左

explore(x, y+1); // 右

// 回溯

mine-=map[x][y][0];

map[x][y][1]=0;

}

int main() {

scanf("%hd %hd", &n, &m);

int i, j;

for(i=1;i<=n;i++)

for(j=1;j<=m;j++){

scanf("%d", &map[i][j][0]);

map[i][j][1]=0;

}

explore(1, 1);

printf("%d", sum);

system("pause");

return 0;

}

**写在最后?**?

递归是指一个函数调用自身的过程。递归通常用于解决可以分解为**相似子问题**的问题。递归的实现包括两个主要部分：

基本情况（Base Case）：定义递归终止的条件，通常是最简单的情况，不再需要递归调用的情况。

递归步骤：在每次递归调用中，问题被分解为更小的子问题，并通过递归调用解决这些子问题。