前言

对于基础知识和大多数的易错点在期中串讲的时候讲得已经很详细了，因此这一次不再重复强调了，建议同学们翻看一下期中的ppt~

这次的串讲重点会放在解题思路和代码实现的技巧方面~

以及下文会提到很多前面几次上机中的题目，或许同学们许多都忘记了，但建议大家可以回过头去看一看，或许曾经觉得很难的题目现在看来会简单许多~

输入输出

这个板块可以看看期中串讲的ppt，那里应该讲得很详细了（就是期中ppt的最开始部分）。

包括例题在期中ppt里也介绍了很多~

重点复习内容：

- 各种类型的输入输出

  注意无符号整数的输入输出：`%llu`、`%u`，使用`d`本地可能正确但提交到OJ会出问题。

  `printf`的用法：

  `%[flags][width][.precision][length]specifier`

specifier	含义
d	十进制输出带符号整数
o	八进制输出无符号整数
X,x	十六进制输出无符号整数
u	十进制输出无符号整数
f	小数形式输出实数
e，E	指数形式输出实数
g，G	在f和e中选取较短的那一个形式输出实数
c	字符
s	字符串
p	指针地址

  建议自己试一试就记住了~

- 格式化输入输出

  `%m.nlf`、`%nd`、`%0nd`

  上次没有讲到的一个扩展：

  ```
  int a=3,x=12;
  printf("%0*d",a,x);//输出结果012
  a=4;
  printf("%0*d",a,x);//输出结果0012
  //其它格式化输出类型%m.nlf、%nd等以此类推
  ```

- 字符和数字混合的输入输出

  输入格式为：`(x,y)`、`theta=...`等（考过好多次了，例题见期中ppt）

- 字符串的输入输出（scanf和gets）
- 浮点数的输出（向上取整、向下取整、四舍五入分别怎么实现？）

字符画类题目

比较简单，一般会作为签到题，考前可以做几个作为练习：

循环类字符画：

国庆 - Problem ?. 菱形（OJ的课程题目里还能找到这道题）

```
//输出一个占n个字符高度、占n个字符宽度的菱形图案
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
 *************
  ***********
   *********
    *******
     *****
      ***
       *
```

固定类字符画：

C1 - Problem B. AC

```
          ****            ******************
         ******           ******************
        **    **          **
       **      **         **
      ************        **
     **************       **
    **             **     **
   **               **    **
  **                 **   **
 **                   **  ******************
**                     ** ******************
```

C7 - Problem A. 佛像

```
                    _ooOoo_
                   o8888888o
                   88" . "88
                   (| -_- |)
                    O\ = /O
                ____/`---'\____
              .'  \\|     |//  `.
             /  \\|||  :  |||//  \
            /  _||||| -:- |||||-  \
            |   | \\\  -  /// |   |
            | \_|  ''\---/''  |_/ |
            \  .-\__  `-`  ___/-. /
          ___`. .'  /--.--\  `. . __
       ."" '<  `.___\_<|>_/___.'  >'"".
      | | :  `- \`.;`\ _ /`;.`/ - ` : | |
      \  \ `-.   \_ __\ /__ _/   .-` /  /
 ======`-.____`-.___\_____/___.-`____.-'======
                    `=---='
```

这道题同时涉及转义字符的应用，再次总结一遍：

```
int main(){
    printf("\\"),printf("\n");
    printf("\""),printf("\n");
    printf("\'"),printf("\n");
    printf("%%"),printf("\n");//划重点.jpg
}
```

排序

冒泡排序

```
//取翻翻课件模板吧
```

qsort

一个来自隔壁助教`XIAO7`写的博客：https://blog.csdn.net/yeshengxiaohuli/article/details/121413622

```
qsort(a,n,sizeof(xx),cmp);
//起始地址 长度 每个元素的大小 比较函数
//如若数组是从1开始储存的就是qsort(a+1,n,sizeof(xx),cmp);
```

cmp函数的原则：返回一个int型，第一个参数应该放在第二个参数前面返回负数，否则返回正数。

一维数组、单关键字排序

- 对`int`类型的一维数组排序：

```
int cmp(const void *p,const void *q){
    int *a=(int*)p,*b=(int*q);
    return (*a)-(*b);//从小到大
    //从大到小：return -(*a)+(*b);
}
qsort(a,n,sizeof(int),cmp);
```

- 对`double`类型的一维数组排序：（其它浮点类型如`float`等以此类推）

```
int cmp(const void *p,const void *q){
    double *a=(double*)p,*b=(double*)q;
    return ((*a)<(*b))?-1:1;//从小到大
    //从大到小：return ((*a)<(*b))?1:-1;
}
qsort(a,n,sizeof(int),cmp);
```

- 对`long long`类型的一维数组排序：

```
int cmp(const void *p,const void *q){
    long long *a=(long long*)p,*b=(long long*)q;
    return ((*a)<(*b))?-1:1;//从小到大
    //从大到小：return ((*a)<(*b))?1:-1;
}
qsort(a,n,sizeof(int),cmp);
```

二维数组、多关键字排序

二维数组、多关键字排序

把多个关键字储存在二维数组的“一行”里，（如第一关键字是`a[i][0]`，第二关键字是`a[i][1]`,···）

然后把**二维数组的“一行”看作一个整体**进行排序：

- 双关键字：

```
int A[N][2];//A[i][0]为第一关键字、A[i][1]为第二关键字
int cmp(const void *p,const void *q){
    int *a=(int*)p,*b=(int*)q;
    //第一关键字A[0]不相等时按A[0]排序，相等时按第二关键字A[1]排序（从小到大）
    return a[0]==b[0]?a[1]-b[1]:a[0]-b[0];
}
qsort(A,n,sizeof(A[0]),cmp);
```

- 三关键字：

```
int A[N][3];//A[i][0]为第一关键字、A[i][1]为第二关键字、A[i][2]为第三关键字
int cmp(const void *p,const void *q){
    int *a=(int*)p,*b=(int*)q;
    //从小到大
    if(a[0]==b[0])
        return a[1]==b[1]?a[2]-b[2]:a[1]-b[1];
    return a[0]-b[0];
}
qsort(A,n,sizeof(A[0]),cmp);
```

- E8 - Problem G. 多关键字排序部分参考代码：

```
int n,k,a[N][15];
int cmp(const void *p,const void *q){
    int *a=(int*)p,*b=(int*)q;
    for(int i=1;i<=k;i++){
        if(a[i]<b[i]) return (i&1)?-1:1;
        else if(a[i]>b[i]) return (i&1)?1:-1;
    }
    return a[0]-b[0];
}
int main(){
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;i++){
        a[i][0]=i;
        for(int j=1;j<=k;j++)
            scanf("%d",&a[i][j]);
    }
    qsort(a+1,n,sizeof(a[0]),cmp);
    for(int i=1;i<=n;i++)
        printf("%d ",a[i][0]);
    return 0;
}
```

字符串排序

字符串排序类似于二维数组的排序，毕竟多个字符串就是一个二维数组，每行表示一个字符串，只不过字符串比较时可以使用`string.h`中的库函数。

```
int cmp(const void *p,const void *q){
    char *a=(char*)p,*b=(char*)q;
    return strcmp(a,b);//按字典序从小到大排序
    //按字典序从大到小排序return -strcmp(a,b);
}
qsort(str,n,sizeof(str[0]),cmp);
```

编号排序

当对于二维数组/字符串排序时，并不需要真的对整个二维数组进行排序，因为这样在进行的排序的交换操作时会交换一整行的内容，我们可以定义一个数组`id[i]`表示当前排在第`i`位的元素的编号，然后对`id`数组进行排序，这样每次交换仅仅交换了一个`int`变量，效率更高。

例：对字符串的长度进行排序：

```
int n,len[N],id[N];
char str[N][N];
int cmp(const void *p,const void *q){
    int *i=(int*)p,*j=(int*)q;
    return len[*i]-len[*j];//按字符串长度从小到大排序
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%s",str[i]);
        len[i]=strlen(str[i]),id[i]=i;
    }
    qsort(id+1,n,sizeof(int),cmp);
    for(int i=1;i<=n;i++)
        printf("%s\n",str[id[i]]);
    return 0;
}
```

桶排序

当排序的数的值域比较小时，可以使用一个数组`cnt[x]`记录`x`出现了多少次，然后从小到大遍历`x`可能的取值，根据`cnt[x]`的值进行输出就能达到排序的效果。

```
int n,cnt[N+5];//N为x可能取值的最大值
int main(){
    scanf("%d",&n);
    for(int i=1,x;i<=n;i++)
        scanf("%d",&x),cnt[x]++;
    for(int i=0;i<=N;i++)
        for(;cnt[i];cnt[i]--)
            printf("%d ",i);
    return 0;
}
```

字符串的处理

字符串的读入

```
//遇见空格或换行终止
scanf("%s",str);
//遇见换行终止
gets(str);
```

常用的字符串函数

```
//头文件：#include<string.h>
strcpy(a,b);//将字符串b复制给字符串a
strcmp(a,b);//比较a、b的字典序大小，a<b返回负数，a=b返回0，a>b返回正数
int len=strlen(a);//求字符串的长度

char *p=strstr(a,b);
//strstr函数用于判断字符串b是否是a的子串。
//如果是，则该函数返回b在a中首次出现的地址；
//否则，返回NULL。int pos=a-p;pos为b在a中第一次出现的位置(p!=NULL)
//上一行是为什么呢，自行查看地址的加减计算原理QAQ

//string.h中还有不少函数，但使用较多的就是这些，其它函数如果要使用请务必查询清楚其使用方法。
```

特别提醒：strlen的使用：

错误示范：

```
#define N 100005
char str[N];
int main(){
    scanf("%s",str);
    for(int i=0;i<(int)strlen(str);i++)
        printf("%c ",str[i]);
}
/*
设字符串的长度为n，每次调用strlen(str)都会花费O(n)的时间
上文的写法调用了n次花费了O(n^2)的时间，当n较大时可能导致TLE */
```

正确示范：

```
#define N 100005
char str[N];
int main(){
    scanf("%s",str);
    int n=strlen(str);
    for(int i=0;i<n;i++)
        printf("%c ",str[i]);
}
```

字符串的匹配与替换：

建议做几个题练习练习：

- C6 - Problem D. 字符串匹配
- C7 - Problem D. 子串逆置_PRO

- E7 - Problem G. good->perfect
- C8 - Problem B. EDGnb
- Mid - Problem F. 又是可爱的字符串

字符串的匹配问题需要特别注意边界的处理！

两个错误示例：看看以下的字符串匹配代码有什么问题

```
//C6 - Problem D. 字符串匹配
for(c=0;c<i;c++)//设置大循环，用于逐位向左挪动，移动
{
    d=0;//每次比较后必要的清零（卡了好久 。。。）
    for(n=0;n<j;n++)//设置小循环，用于逐个比较是否有相同的字符串
    {
        if(a[c+n]==b[n])//如果一个字符相同，d加1
            d++;
    }
    if(d==j) //如果都相同，证明字符串匹配
    {
        m=strstr(a,b)-a; //算出b字符在a字符第几位首次出现
        printf("YES %d\n",m); //输出要求
        break;//结束循环
    }
}
/*
思路和注释都很清晰，但a[c+n]==b[n]这个地方还需要判断一下c+n是否小于字符串a的长度i
否则可能会导致数组越界（能够通过是因为数据比较弱QAQ）
*/
```

```
//Mid - Problem F. 又是可爱的字符串
#define ll long long
#define max(a,b) (((a) > (b)) ? (a) : (b))
#define min(a,b) (((a) < (b)) ? (a) : (b))
char target[2000], ori[2000];
int main() {
    scanf("%s %s", target, ori);
    int count = 0;
    for (int i = 0;ori[i] != 0;i++) {
        int dif = 0;
        for (int j = 0;target[j] != 0;j++) {
            if (target[j] != ori[i + j]) dif++;
        }
        if (dif <= 3)count++;
    }
    /*
    正确的代码：
    int lent = strlen(target), lenori = strlen(ori);
    for (int i = 0;i < lenori - lent;i++) {
        int dif = 0;
        for (int j = 0;j < lent;j++) {
            if (target[j] != ori[i + j]) dif++;
        }
        if (dif <= 3)count++;
    }
    */
    printf("%d", count);
}
/*
因为没有考虑越界，当S为ABC是会把AB认为是可爱的。
*/
```

sscanf 和 sprintf处理字符串

具体的用法可以百度一下，这里补赘述了，介绍一个主要的应用，从字符串中提取数字：

C7 - Problem J. MIPS

```c
int main() {
 char x[100],y[100];
 scanf("%s",x);
 scanf("%s",y);
 int a,b,c,d;
 sscanf(x,"%d+%di",&a,&b);
 sscanf(y,"%d+%di",&c,&d);
 printf("%d+%di\n",a+c,b+d);
 return 0;
}
```

输入

```
1+2i 3+4i
```

输出

```
4+6i
```

位运算

基本运算

`&`、`|`、`^`、`>>`、`<<`、`~`（01取反）

需要注意的基本问题：

- 位运算的优先级：建议不要去记，全部打括号吧。
- 注意位移运算的溢出问题：如 `1<<50` 会导致溢出，应该 `1ll<<50`（同理 `1ll<<63` 会溢出，应该 `1ull<<63`）。

常用数值（善用16进制）

```c
//16进制的一位可以直接对应为2进制的四位，使得人脑进制转换非常方便
unsigned int a = 0xff000000; //11111111 00000000 00000000 00000000
unsigned int b = 0x00ff0000; //00000000 11111111 00000000 00000000
unsigned int c = 0x0000ff00; //00000000 00000000 11111111 00000000
unsigned int d = 0x000000ff; //00000000 00000000 00000000 11111111

//低n位为1
int n=13;
unsigned int e = (1 << n) - 1; //00000000 00000000 00011111 11111111

//低[l,r]位为1，其余为0
int l=5,r=15;
unsigned int f = ((1<<(l-1))-1) ^ ((1<<r)-1);
//00000000 00000000 01111111 11110000
```

一些简单常见的位运算操作

我的写法不一定时最优的，建议同学们总结出自己习惯的便于理解和记忆的写法~

```c
unsigned int x;
scanf("%u",&x);
unsigned int a = (x>>i)&1;//x的从低到高第i位(0或1)

x = x|(1<<(i-1)) //将从低到高第i位(表示2^(i-1)位)的值赋值为1
x = x&(~(1<<(i-1))) //将从低到高第i位(表示2^(i-1)位)的值赋值为0
x = x^(1<<(i-1)) //将从低到高第i位(表示2^(i-1)位)的值取反

unsigned int b = x&((1<<n)-1);
/*提取x的末尾低n位(其它位为0)
x = 00000000 00000000 01111111 11110000
n = 7
b = 00000000 00000000 00000000 01110000*/

unsigned int c = x&(((1<<(l-1))-1) ^ ((1<<r)-1));
/*提取x从低到高[l,r]位(其它位为0)
x = 00000000 00000000 01111111 01110000
l = 6, r = 10
c = 00000000 00000000 00000011 01100000*/

unsigned int d = c>>(l-1);
/*提取x从低到高[l,r]位并移动到最低位
x = 00000000 00000000 01111111 01110000
```

```
l = 6, r = 10
c = 00000000 00000000 00000011 01100000
d = 0000...000 11011*/

/*另一种写法：
int len=r-l+1;
unsigned int d = (x>>(l-1))&((1<<len)-1);*/
```

对异或运算的理解

- 异或`1`相当于`01`取反。
- 具有交换律。
- 多个`01`异或其结果为`0`还是`1`取决于`1`的个数的奇偶性。

异或运算的应用：

E8 - problem J. 寻找单身狗（看看题解，掌握如何找到一直单身狗的情况就差不多了）

合理利用和设置函数

合理利用函数不仅可以减轻代码量，也能减轻不少思维量，也能更方便第套用模板（下面举了几个应用上文位运算模板的例子）。

- C5 - Problem C. 输出距离

  从减少代码量的角度来看，可以把`printf`放进函数。

```
void getDis(int i,int j){
    double d=sqrt(sqr(x[i]-x[j])+sqr(y[i]-y[j])+sqr(z[i]-z[j]));
    printf("%.2f\n",d);
}
int main(){
    for(int i=1;i<=4;i++)
        scanf("%lf%lf%lf",&x[i],&y[i],&z[i]);
    getDis(2,4),getDis(1,3),getDis(2,3);
    getDis(3,4),getDis(1,2),getDis(1,4);
    return 0;
}
```

- C3 - Problem F. 模拟汇编

```
int f(int x,int L,int R){
    int l=33-R,r=33-L;
    /*从左往右的第x位，就是从低到高的第33-x位
    这个怎么快速推呢就是考虑从左往右的第1位，是从低到高的第32位；
    从左往右的第2位，是从低到高的第31位；
    ...*/
    int len=r-l+1;
    return (x>>(l-1))&((1<<len)-1);//这两行不就是上面位运算部分一模一样的的模板吗
}
int main(){
    int n;
    scanf("%x",&n);
    printf("add $%d,$%d,$%d",f(n,17,21),f(n,7,11),f(n,12,16));
    return 0;
}
/*
当然这个写法并不是最短的最优的写法，但在有模板的前提下，它是一种思维量和代码量都较小的方式。
*/
```

- E8 - Problem D. 位运算

```
unsigned long long f(unsigned long long x,int l,int r){
    l++,r++;//题目中的第几位是从0开始的，模板中的第几位是从1开始的
    int len=r-l+1;
    return (x>>(l-1))&((1ull<<len)-1);//这两行又是一模一样的
}
int main(){
    unsigned long long n;int m;
    while(~scanf("%llu%d",&n,&m)){
```

```
        unsigned long long x=f(n,0,m);//取出0-m位
        unsigned long long y=f(n,m+1,2*m+1);//取出m+1-2m+1位
        unsigned long long z=f(n,2*m+2,63);//取数2m+2-最高位
        printf("%llu\n",(z<<(2*m+2))|(x<<(m+1))|y);//合体
    }
    return 0;
}
/*
这里使用函数的意义就是：把特殊问题一般化，我们先考虑实现“取出[l,r]位”这一功能封装成函数
然后接下来就是考虑“我应该取出哪些位”（显然是0-m和m+1 - 2m+1和2m+2-最高位）这些部分的思维量和代码量就都变小了很多QAQ。

至少比先思考如何取出0-m位
再如何取出m+1-2m+1位
再如何取出高位
要简单一点点（特别是在有模板的情况下）

上面的模拟汇编这个题就更明显了
当写好了一般化取出[l,r]位二进制的函数后（就是套用模板节省思维时间），接下来的部分就非常简单。
*/
```

函数的传参

- `int`、`long long`、`double`、`char`..等类型的变量应该都没什么问题。
- 如何传参一维数组：

```
int f(int *a){
    //body
}
int main(){
    int a[10];
    int ans=f(a);
}
```

  其它类型包括字符串类型以此类推。

- 如何传参二维数组：

```
#define N 25
int f(int a[][N]){
    //body
}
int main(){
    int a[N][N];
    int ans=f(a);
}
```

- 如何返回多个数值：

  C7 - Problem E. Gcd-Expression

```
void Example(int *a,int *b,int *c){
    *a=1,*b=2,*c=3;
}
int main(){
    int a=0,b=0,c=0;
    printf("%d %d %d\n",a,b,c);
    Example(&a,&b,&c);
    printf("%d %d %d\n",a,b,c);
}
```

数组的应用

数组与映射

大多数时候我们把数组看作一个数列，`a[0]`表示数列的第一个数，`a[1]`表示数组的第二个数···

但有些时候可以把数组看作一个关于整数的函数或者映射，如可以把`sin(x)`的值存入`f[x]`中，此时数组`f`就可以看作`sin(x)`函数。（这个例子看起来数组没有什么必要，因为其对应的函数是显性的比较明显的，但当函数是隐性的时候，就可以利用数组进行求解。）

例如 `hash表`、`桶排序` 都是数组在以上意义下的应用。

- 期中 - Problem H. 我的朋友很少...吗?

  用数组 `f[x][y]` 表示 `xy` 是否是朋友。

- C7 - Problem G. 划分测试集 以及 上文提到的桶排序

  E8 - Problem F. xf当助教（一）

  用数组 `cnt[x]` 表示数字 `x` 出现的次数。（**这似乎出现了很多次**）

- E2 - Problem E. 水水的跳格子

  用数组 `pos[x]` 表示 `ascii` 码为 `x` 的字母最近一次出现的位置。

- C8 - Problem G. 跳格子

  用数组 `g[x]` 表示跳到第 `x` 个格子，最后一次跳了三格的情况数，`f[x]` 表示跳到第 `x` 个格子，最后一次跳了一格或两格的情况数。

以上情况数组实现的都是定义域为**非负整数**的函数/映射，那么对于定义域不为正整数的隐函数，如何用利用数组实现映射。方法就是很多次题解或者Hint里所提到过的**hash表。**

思路为先将定义域 I 映射到非负整数域，然后将在将函数值映射到数组。

如 C7 - Problem G. 划分测试集 中数字的范围其实是 `[-10000,10000]` 先构建一个 `x->x+10000` 的映射，然后使用 `cnt[x+10000]` 表示 `x` 出现的次数。

以上方法应用最多的地方为有关字符串处理的问题，通常会使用以下函数将字符串映射到一个整数：

```
#define key 3711
#define mod 1000000
//key 和 mod可以设置为其它你喜欢的值
int Hash(char*str){
    int h=0;
    for(char*p=str;*p!='\0';p++)
        h=(h*key%mod+(*p))%mod;
    return h;
}
```

关于数组越界的理解与处理：

E5 - Problem K. 多项式的加法

二维数组的应用

- 注意不要越界
- 访问一个格子的四周（四个方向或八个方向？）
  - C6 - Problem H. 康威生命游戏
  - E8 - Problem B. 小秋月打码

```
//方向数组的应用，以下是我的小秋月打码的代码实现
int d[8][2]={{-1,0},{1,0},{0,-1},{0,1},{-1,-1},{-1,1},{1,-1},{1,1}};
/*以上是8个方向，4个方向是什么样呢：
d[4][2]={{-1,0},{1,0},{0,-1},{0,1}}*/
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++){
        int sum=a[i][j],cnt=1;
        for(int k=0;k<8;k++){
            int x=i+d[k][0],y=j+d[k][1];
            if(x>=1 && x<=n && y>=1 && y<=m)
                cnt++,sum+=a[x][y];
        }
        printf("%3d%c",(int)(1.0*sum/cnt+0.5),j==m?10:32);
    }
```

- 矩阵相乘（准备模板.jpg）

递归

如何分析并实现递归类题目。

递归三要素：

- 函数所要实现的功能
- 终止的条件
- 找出与函数等价的关系式

例题：

- C5 - Problem B. 错误的斐波那契数列
  - 函数要实现的功能：`f(n)` 表示 错误的斐波那契数列 的第n项；
  - 终止的条件：`n=1,2,3` 时终止递归 `f(n)=1`；
  - 找出与函数等价的关系式：题目已经给出：`f(n)=f(n-1)+f(n-3)`。

```
int f(int x){
    return x<=3?1:(f(x-3)+f(x-1));
}
```

- C5 - Problem G. xf学组合数学
  - 函数要实现的功能：`C(m,n)` 表示从m个数中取n个无序的数字一共有多少种取法;
  - 终止的条件：

$$
\begin{aligned}
&\text{若}m<n\text{,显然无法从}m\text{个数中选出}n\text{个数！因此}C(m,n)=0(\text{当}m<n\text{时})\\
&\text{从}m\text{个数中选取}0\text{个数，显然只有一种选法就是什么都不选！因此}C(m,0)=1\\
&\text{从}m\text{个数中选取}1\text{个数，显然有}m\text{种选法！因此}C(m,1)=m
\end{aligned}
$$

  - 找出与函数等价的关系式：`C(m,n)=C(m-1,n-1)+C(m-1,n)`
- C5 - Problem H. 可爱的数列
  - 函数要实现的功能：`F(l,r)` 表示题目中的 $f([a_l,a_l+1,\ldots,a_r])$。
  - 终止的条件：
    - 题目要求：`f([0])=0`、`f([1])=1`；
    - 因此终止条件：`l=r` 时终止条件 `F(l,r)` =$f([a_l])$=a_l。
  - 找出与函数等价的关系式：
    - 题目要求：
      $f([a_1,a_2,\cdots,a_k,a_{k+1},\cdots,a_{2k}])=(f([a_1,a_2,\cdots,a_k])+f([a_{k+1},\cdots,a_{2k}]))\ \&\ (f([a_1,a_2,\cdots,a_k])-f([a_{k+1},\cdots,a_{2k}])$
    - `F(l,r)=(F(l,mid)+F(mid+1,r))&(F(l,mid)-F(mid+1,r))`
- Mid - Problem G. 数组填充
  - 函数要实现的功能：`f(n)` 表示长度为 `n` 的数组的填充方案？（然后就感觉做不下去了.jpg）

    这是因为 `f(n)` 所能给出的信息太少了，我们可以增加一些参数：

    `f(n,x)` 表示长度为 `n` 的数组填充出总和mod 3为 `x` 的方案数。
  - 终止的条件：`n=1` 时：

```
for(i=l;i<=r;i++)
    ans+=(i%3==mod);
```

  - 找出与函数等价的关系式：考虑最后一个元素即第 `n` 个元素的填充：

```
if(mod==0)
    return cnt(1,0)*cnt(n-1,0) + cnt(1,1)*cnt(n-1,2) + cnt(1,2)*cnt(n-1,1);
if(mod==1)
    return cnt(1,0)*cnt(n-1,1) + cnt(1,1)*cnt(n-1,0) + cnt(1,2)*cnt(n-1,2);
if(mod==2)
    return cnt(1,0)*cnt(n-1,2) + cnt(1,1)*cnt(n-1,1) + cnt(1,2)*cnt(n-1,0);
```

- C8 - Problem G. 跳格子

  当这道题的数据范围比较小时也可以用递归实现，因此选在这里再巩固一下递归的思维。
  - 函数要实现的功能：`f(n)` 表示按要求跳到第 `n` 格的方案数？（然后就感觉做不下去了.jpg）

    这是因为 `f(n)` 所能给出的信息太少了，我们不知道如何限制不能 连续跳两次三格 这个条件，因此我们可以增加一些参数，比如计算当前这一步怎么跳时，我们只要知道上一步是不是跳了三格，就能推导出这一步能不能跳三格，那么就把最后一步是不是三格加入状态中：
    - `f(n,0)` 表示按要求跳到第 `n` 格且最后一步不是三格。
    - `f(n,1)` 表示按要求跳到第 `n` 格且最后一步是三格。
  - 终止的条件：

    `f(0,0)=1`、`f(0,1)=0`。
  - 找出与函数等价的关系式：

`f(n,0)=....`

枚举

如何实现高效且代码量短的枚举（枚举的对象和顺序是怎么样的）。（暴力出奇迹.jpg）

- 枚举所有情况求最优解或者总和的问题：（大胆尝试，估算总情况数）

  能够枚举就没必要花时间分析最优解的性质（有时候很可能没有性质QAQ）

  即使能够分析出一些最优解的性质，既然能够枚举就省省自己的脑子和代码量吧QAQ

  - C2 - Problem H. 一元线性回归
    - 题目要求：**删除某一组数据并重新计算剩余**$(n-1)$**组数据的**a,b,s。请帮帮小H，编程求解何时可以使"偏差值"s最小,并输出此时s的值。
    - 一共有几种删除方案：`n`种，直接枚举.jpg
  - C4 - Problem F. 愿此行，终抵群星
    - 题目要求：判断一堆数中有没有相等的两个数。
    - 看看数据范围发现 n 最大为 1000，暴力枚举判断 a_i 和 a_j 是否相等也只有 n^2 种情况是完全可以接受的。（当然我们再进一步分析这道题，题目给出的数列是有序的，相等的元素一定相邻，这可以减少枚举的次数，但既然能够暴力就不入暴力吧（暴论.jpg））
  - E4 - Problem E. 超可爱的XIAO7和RGB字符串
    - 修改字母不会改变字符串的长度 n，长度为 n 的可爱的字符串即原字符串修改后的最终结果最多有不超过 n^2 种可能。根据数范围这个数量是可以接受的因此直接枚举即可。
  - C7 - Problem H. 错误的幻方
    - 交换的两个数字在不同行不同列：一定会导致两行和两列的和产生错误。找到这两行两列，枚举其四个交点的交换方案即可。（为什么能枚举呢因为4个点最多$C_4^2=6$种，非常少，以及仔细分析一下其实只有两种。）
    - 交换的两个数字在同一行：会导致两列的和产生错误，找到这两列，枚举这两个数可能在的行（最多也就 n 种情况，也不多是可以枚举的.jpg），尝试交换并验证即可。
    - 同一列与同一行同理。

- 枚举对象的选择：
  - E6 - Problem F. 寻找五子相连
    - 枚举最五子相连中最**左侧**（竖着的相连就是最上侧）的那一枚棋子。
    - 然后就只需要判断四个方向（往右、往下、往右上、往右下）是否有连续的五个棋子。
  - C3 - Problem I. xf买彩票
    - 来自题目的重要提示：三位或位数更多的数若能被8整除，则其最后三位数组成的数字可以被8整除，如1919808的最后三位是808，而808能被8整除，所以1919808可以被8整除。
    - 因此只要字符串中存在任意`1-3`个位置上面的数组合起来是`8`的倍数。
    - 实现方式：枚举所有情况 n^3。
  - E2 - Problem H. 回文日期
  - C5 - Problem G. 素数日期

    这两个题目期中串讲讲枚举的时候提到了（让数字+1判断一下是否是合法日期，不比考虑一堆情况让日期+1更简单吗。）
  - E2 - Problem I. Long Long Factorial 阶乘末尾的`0`。

    这道题关于因数的枚举方式可以去查看一下当时的题解~

其它模板

- 数位拆分

  准备模板，如何把一个数字的数位拆开：
  - E2 - Problem D. 数位平均
  - C3 - Problem G. 士谔2173&&传源2171
  - E4 - Problem G. 士谔2173&&传源2171_PLUS
  - E6 - Problem D. 数位拆解
- 进制转换

  准备模板.jpg
- 素因数分解

    - C2 - Problem J. 分解质因数
- 日期相关

  还是准备模板.jpg

  比如如何计算某一天是一年开始的第多少天、距离年末多少天、任意两天之间间隔多少天...

  （期中串讲的时候我应该给过两个）

    - C2 - Problem I. 节日快乐吗
    - E2 - Problem H. 回文日期
    - C5 - Problem G. 素数日期
- 回文的判断

  （这个挺简单应该不需要模板吧）

    - E6 - Problem B. 回文魔咒
    - E2 - Problem H. 回文日期
- 最大公约数 and 最小公倍数

  同样准备模板.jpg

    - C5 - Problem F. 分数加法
    - E4 - Problem A. 最小公倍数

模拟类题目

根据题意进行一些操作

- C2 - Problem E. Monica的加密
- C4 - Problem I. Monica的羊
- C4 - Problem J. HDT
- E4 - Problem D. 水水的浮点数二进制
- E6 - Problem H. 循环填数
- E8 - Problem E. 点到谁就选谁

感觉没什么好说的，读清题目按要求模拟就可以了。

对于比较复杂的模拟的建议：

- 在动手写代码之前，在草纸上尽可能地写好要实现的流程。
- 在代码中，尽量把每个部分模块化，写成函数或结构体等。
- 对于一些可能重复用到的概念，可以统一转化，方便处理：如，某题给你 "YY-MM-DD 时：分" 把它抽取到一个函数，处理成秒，会减少概念混淆。
- 调试时分块调试。模块化的好处就是可以方便的单独调某一部分。
- 写代码的时候一定要思路清晰，不要想到什么写什么，要按照落在纸上的步骤写。

  （think twice and code once）

根据给定的复杂公式进行计算

- C2 - Problem F. 淑芬来啦
- E2 - Problem F. 由果推因
- E2 - Problem G. 补给广播站的建设
- C5 - Problem E. GPA的计算
- E4 - Problem I. 古墓丽影——Euler's Legacy

总之就是不要被题目吓到，按题目给出的公式一步一步算就行，特别是对于淑芬来啦和补给广播站的建设这一类的公式特别长特别复杂的题目，一定要**将公式拆分**，观察是否有重复出现的部分，一步一步地计算（这样也方便debug）。

C2 - Problem F. 淑芬来啦

$$x = \begin{cases} \frac{1}{\sqrt{2\pi}c} e^{-\frac{(x_1-p)^2+(y_1-p)^2}{2c^2}} (x_1 + \frac{|z_1|\tan 2\theta}{\sqrt{1+b^2/a^2}}) & , x_1 < 0 \\ \frac{1}{\sqrt{2\pi}c} e^{-\frac{(x_1-p)^2+(y_1-p)^2}{2c^2}} (x_1 - \frac{|z_1|\tan 2\theta}{\sqrt{1+b^2/a^2}}) & , x_1 > 0 \end{cases}$$

$$y = \frac{1}{\sqrt{2\pi}c} e^{-\frac{(x_1-p)^2+(y_1-p)^2}{2c^2}} (y_1 + \frac{b}{a}(x - x_1))$$

$$z = \frac{\arctan{(p\cos x)}^{\ln(1+|\sin x|)}}{2 + |\sinh y|}$$

观察后发现会有一些重复出现的部分：

$\frac{1}{\sqrt{2\pi}c}e^{-\frac{(x_1-p)^2+(y_1-p)^2}{2c^2}}$以及$\frac{|z_1|\tan 2\theta}{\sqrt{1+b^2/a^2}}$

先将这两部分计算出，再进行后续计算。

关于指针的一些问题

不要调用野指针，使用指针请务必初始化QAQ。

```
//C6 - Problem D. 字符串匹配
#include<stdio.h>
#include<string.h>
//int lena, lenb;
char a[105], b[105];
int main() {
    char *p;
    gets(a), gets(b);
    if (strstr(a, b) != NULL) {
        printf("YES");
        *p =(strstr(a, b) - a);
        printf(" %d", *p);
    } else printf("NO");
    return 0;
}
/*
以上代码能够AC纯属侥幸.jpg
char *p;没有初始值是一个野指针，后面直接对*p进行了赋值，大多数情况都会导致运行错误。
且地址和地址进行运算后的返回值是整数类型而不是char类型。
定义指针后一定要对其进行初始化。
  + 可以使其指向已知变量的地址，如：
    int b=0;
    int *p=&b;
  + 或使其等于已经定义的地址/指针，如：
    int a[10];
    int *p=a;
    int *q=p;
  + 使用malloc函数：(需要头文件#include<stdlib.h>)
    int *p=(int*)malloc(sizeof(int));
    char *q=(char*)malloc(sizeof(char));
    double *r=(double*)malloc(sizeof(double));
*/
```

一些较高级算法的简单介绍

前缀和

定义

那么对于一个数组的前缀，例如数组`a = [1,2,3,4,5]`，我们维护一个由前缀的和组成的数组`sum`，`sum[i]`表示数组中`a[0]~ a[i]`的和。

```
sum[0] = a[0]
sum[1] = a[0] + a[1]
sum[2] = a[0] + a[1] + a[2]
sum[3] = a[0] + a[1] + a[2] + a[3]
sum[4] = a[0] + a[1] + a[2] + a[3] + a[4]
```

`sum`数组就被称为前缀和数组。

作用

前缀和的最主要目的就是求子数组的和的大小，如求`a[l]-a[r]`的和：

```
a[l]+a[l+1]+a[l+2]+...+a[r]=sum[r]-sum[l-1]
```

应用

Mid - Problem I. 规矩

矩形的两条对角线为直径：寻找矩形 -> 寻找直径

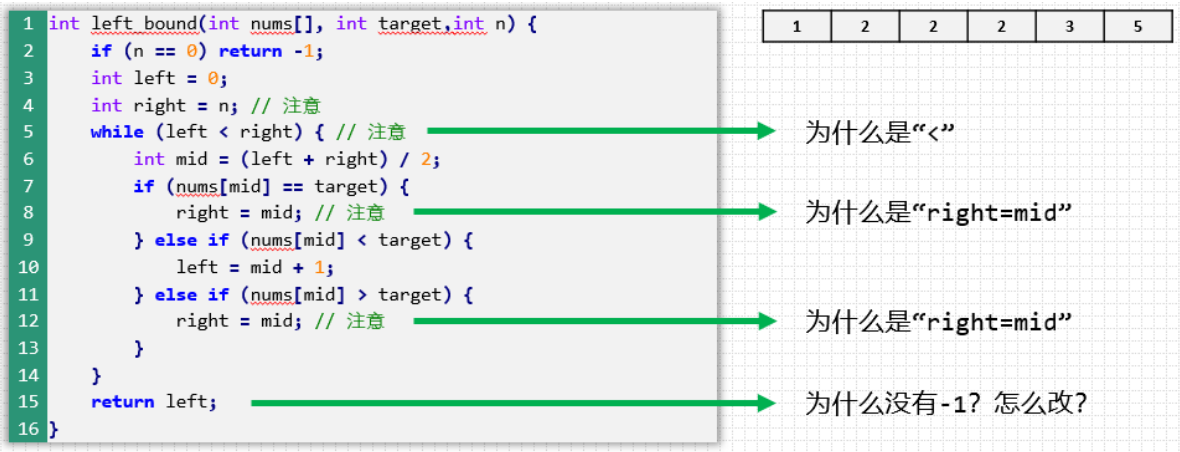
连结第`i`个点和第`j`个点形成的弦为直径当且仅当第`i`个点到第`j`的弧长为周长的一半，即`a[i+1]+a[i+2]+...+a[j]`（使用前缀和处理）。

二分查找

C6 - Problem G. 二分查找·改

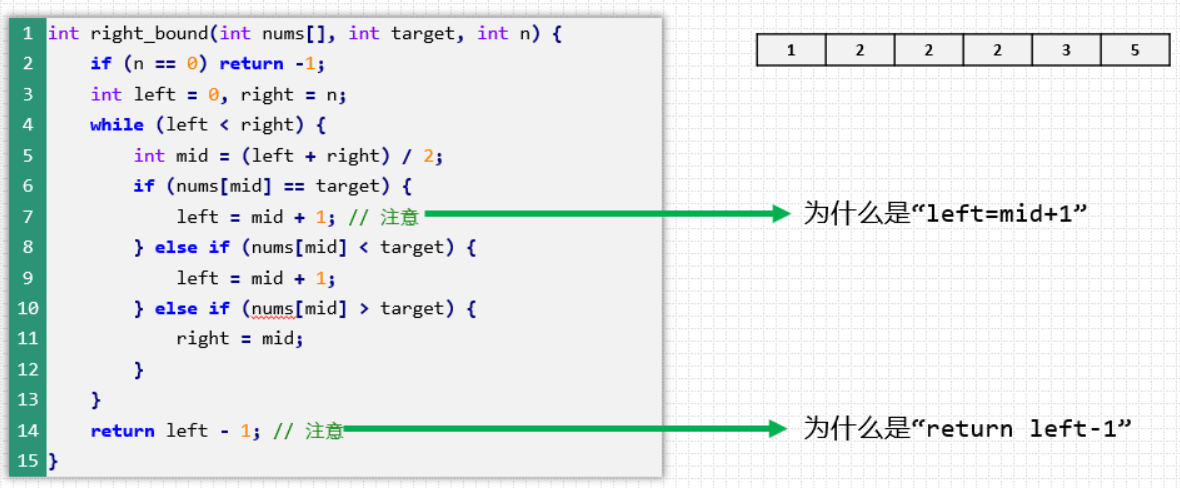
C8 - Problem J. 排列之差

- 查找左边界




```
int left_bound(int nums[], int target,int n) {
    if (n == 0) return -1;
    int left = 0;
    int right = n; // 注意
    while (left < right) { // 注意
        int mid = (left + right) / 2;
        if (nums[mid] == target) {
            right = mid; // 注意
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else if (nums[mid] > target) {
            right = mid; // 注意
        }
    }
    return left;
}
```

- 查找右边界




```
int right_bound(int nums[], int target, int n) {
    if (n == 0) return -1;
    int left = 0, right = n;
    while (left < right) {
        int mid = (left + right) / 2;
        if (nums[mid] == target) {
            left = mid + 1; // 注意
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else if (nums[mid] > target) {
            right = mid;
        }
    }
    return left - 1; // 注意
}
```

高精度

- E2 - Problem J. Long Long Double Factorial
- C8 - Problem H. 简单易懂的题

高精度本质上来说就是用数组（或者字符串）来按位存储大数。

习惯上，下标最小的位置存放的是数字的最低位，这样可以保证权值对齐；同时，加、减、乘的运算一般都从个位开始进行。

四则运算，就是**列竖式。**

主要掌握以下两种大多数情况就够用了：

高精度加法：

整型变量成高精度：

最后：祝大家期末考试取得满意的成绩~

Author YUKILSY