R2-保温赛题解

祝大家期末顺利～

A dch解方程

```c
#include <stdio.h>

int main()
{
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    printf("%0.2f", ((double)(c - b)) / a);
}
```

B 三角形面积·改

```c
#include<stdio.h>

int main()
{
    int n;
    double a, b, c, p,ans;
    scanf("%d", &n);
    for (int i = 0; i < n;i++)
    {
        scanf("%lf%lf%lf", &a, &b, &c);
        p = (a + b + c) / 2;
        ans = p * (p - a) * (p - b) * (p - c);
        printf("%.1f\n", ans);
    }
}
```

C cbj统计成绩

```c
#include <stdio.h>
int main(){
    int n = 0;
    double cur,sum = 0;
    while(~scanf("%lf",&cur)){
        sum += cur;
        n += 1;
    }
    printf("%.2f",sum / n);
}
```

D 大化实验好难啊

本题中的砝码实际上就是二进制中的对应的位，该位为1即选择这一砝码，为0则不选择，每种砝码只有一个且二进制数与十进制数一一对应，所以是一个判断对应数二进制补码为1的位的问题。

```c
#include <stdio.h>
```

```
int main()
{
    unsigned int Mass = 0;
    scanf("%u", &Mass);
    for (int i = 0; i < 32; i++)
    {
        unsigned int ToolNum = 1u << i;
        if (ToolNum & Mass)
            printf("%u\n", ToolNum);
    }
    return 0;
}
```

E Dec to 2's Complement

由于我们需要输出整数的 64 位二进制补码表示，且整数不会超过 `long long` 的范围，因此需要我们使用 `long long` 类型来保存输入的整数。接下来我们将这个数据中的每一个 `bit` 提取出来即可，因为 `long long` 本身在计算机中就是以补码形式存储的。

```
#include<stdio.h>

void print_bin(long long n, int bits)
{
    if (bits)
    {
        print_bin(n >> 1, bits - 1);
        printf("%lld", n & 1);
    }
}

int main()
{
    long long n;
    while (~scanf("%lld", &n))
    {
        print_bin(n, 64);
        printf("\n");
    }
    return 0;
}
```

F 位 互 换

本题是一个**位运算**的基础题目，并没有涉及数据范围等坑点。大致需要掌握的基础技能有：取出特定位、特定位置

零、特定位置一个指定数。

当然，涉及位运算符的时候，运算符优先级容易出错，一劳永逸的方法是大家可以全部加上括号。

```
#include <stdio.h>

int main()
{
    int a, m, n, m_bit, n_bit, result;

    while (scanf("%d%d%d", &a, &m, &n) != EOF)
    {
        //分别提取第m n位
        m_bit = (a & (1 << m)) >> m;
        n_bit = (a & (1 << n)) >> n;
        //指定位置零
        a &= ~(1 << m);
        a &= ~(1 << n);
        //指定位置指定数
        a |= m_bit << n;
        a |= n_bit << m;
        //输出
        printf("%d\n", a);
    }
    return 0;
}
```

G LJF不会算分数2.0

```
#include<stdio.h>
int gcd(int a,int b)
{
    return a % b ? gcd(b, a % b) : b;
}
int main()
{
    int a, b, c, d, op;
    while(~scanf("%d/%d %d/%d %d",&a,&b,&c,&d,&op))
    {
        int fenzi,fenmu;
        if (op == 1)
            fenzi = a * d + b * c, fenmu = b * d;
        else if (op == 2)
            fenzi = a * d - b * c, fenmu = b * d;
        else if (op == 3)
            fenzi = a * c, fenmu = b * d;
        else
```

```
            fenzi = a * d, fenmu = b * c;
        int flag = 0;
        if(fenzi<0)
            flag = 1, fenzi = -fenzi;
        if(flag)
            printf("-");
        if(fenzi%fenmu==0)
            printf("%d\n", fenzi / fenmu);
        else
            printf("%d/%d\n", fenzi / gcd(fenzi, fenmu), fenmu / gcd(fenzi, fenmu));
    }
}
```

H 时刻3.0

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>

int main()
{
    int h1,m1,h2,m2,z1,z2;
    scanf("%d:%d %d",&h1,&m1,&z1);
    scanf("%d:%d %d",&h2,&m2,&z2);

    h1-=z1/100;
    h2-=z2/100;
    m1-=z1%100;
    m2-=z2%100;

    if(m2>=60)
    {
        m2-=60;
        h2++;
    }
    if(m2<0)
    {
        m2+=60;
        h2--;
    }
    if(m1>=60)
    {
        m1-=60;
        h1++;
    }
```

```
    if(m1<0)
    {
        m1+=60;
        h1--;
    }

    if(h1>=24) h1-=24;
    if(h1<0) h1+=24;
    if(h2>=24) h2-=24;
    if(h2<0) h2+=24;

    if(h1<h2) printf("2");
    else if(h1>h2) printf("1");
    else
    {
        if(m1<m2) printf("2");
        else if(m1>m2) printf("1");
        else printf("Same Time");
    }
    return 0;
}
```

I 荷家军进攻汉诺塔

本题是PPT中的原题，解题核心是一个递归函数，将整个搬动过程分为三个部分，以将柱从`from`经过`via`移动到`to`为例：

1. 将上面n-1个盘子从`from`经由`to`搬动到`via`。
2. 将最下面的n号盘从`from`搬动到`to`。
3. 将第一步搬动到`via`的塔经由`from`搬到`to`。

本题只要注意输入语句的写法即可,由于柱的标号是一个大写字母，所以使用`%c`输入，使用`char`型变量存储即可，无需使用字符串。

```
#include<stdio.h>

void hanoi(int n, char a, char b, char c);
// 把n个盘子从柱子 a 通过 b 挪到 c 上
void move(int n, char a, char c); // 把第n号盘子从柱子a挪到c上

int main()
{
    int num;
    char a, b, c;

    // freopen("out2.txt","w",stdout);
```

```
    scanf("%d %c %c %c",&num,&a,&b,&c);
    hanoi(num, a, b, c);
    return 0;
}
void move(int n, char from, char to)
{
    printf("move %d from %c to %c\n", n, from, to);
}
void hanoi(int n, char from, char via, char to)
{
    if (n == 1)
    {
        move(n, from, to);
        return;
    }
    hanoi(n - 1, from, to, via); // 把n-1个盘子从from通过
    move(n, from, to);
    hanoi(n - 1, via, from, to);
}
```

J 小迷弟的反向最大公约数

题目的三组样例以及hint已经暗示得很明显了，本题的正解就是斐波那契数列，本题的目的是为了让大家更深刻地了解递归。也可以用数学归纳法证明此结论（x）。

感兴趣的同学还可以查一查拉梅定理（Lamé's Theorem），对于辗转相除法步数的一些估计。以斐波那契数列相邻两项作为输入会使欧几里德算法达到最多调用次数。

```
#include <stdio.h>
long long fib[100]={1,1};
int main(){
    int i;
    for(i=2;i<=50;i++){
        fib[i]=fib[i-1]+fib[i-2];
    }
    int n;
    scanf("%d",&n);
    printf("%lld %lld",fib[n],fib[n-1]);
    return 0;
}
```

K 合成大西瓜2022

本题标准答案是采用数组模拟二进制加法，但由于数据范围较小，对算法并没有太多要求，只要能够按题意进行建模模拟即可

```
#include<stdio.h>
int main()
{
    int sum[1020]={0};
    int a,i;
    while(scanf("%d",&a)!=EOF)
    {
        if(sum[a]==0)
            sum[a]=1;
        else
        {
            sum[a]=0;
            for(i=a+1;i<=1019;i++)//进位
            {
                if(sum[i]==1)
                    sum[i]=0;
                else
                {
                    sum[i]=1;
                    break;
                }
            }
        }
        for(i=1019;;i--)
            if(sum[i]==1)
            {
                printf("%d\n",i);//输出最大的水果
                break;
            }
    }
    for(i=1019;i>=0;i--)
        if(sum[i]==1)
            printf("%d ",i);//输出全部水果
    return 0;
}
```

L 超级泡泡排序2021

本题是冒泡排序的改编（如果你还不懂冒泡排序的原理和算法，请仔细看课件，本分析基于你已经了解冒泡排序的基础），冒泡排序的排序关键字数**两个数之间的大小**，本题的解法大体仍然是冒泡排序框架，判断两个数是否交换时，第一关键字是数到m的距离，第二关键字是两个数的大小，伪代码如下：

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

```
    {
        if ((| a[j] - m | > | a[j + 1] - m |) 或者距离相等并且(a[j] > a[j + 1]))
        {
            flag = 1;
            两数交换
        }
    }
    if (!flag)
        break;
}
```

代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int n,m;
int a[1005];
int main(void)
{
    scanf("%d%d",&n,&m);
    for(int i=0;i<n;i++)
    scanf("%d",a+i);
    for(int i=0;i<n;i++)
    {
        int flag=1;
        for(int j=0;j<n-i-1;j++)
        {
            if(abs(a[j]-m)>abs(a[j+1]-m)||(abs(a[j]-m)==abs(a[j+1]-m)&&a[j]>a[j+1]))
            {
                int temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
                flag=0;
            }
        }
        if(flag)
        break;
    }
    for(int i=0;i<n;i++)
    printf("%d\n",a[i]);

    return 0;
}
```

M Ning的汉明距离

本题是字符串的简单处理加上循环。两个编码的汉明距离其实就是相对应的编码位上不同的个数，我们可以用二维字符数组，实现字符串的读入，然后二重循环遍历每一对编码，计算其汉明距离，更新最小值，注意最小值的初始值不能比n小。

```
#include <stdio.h>
#include <stdlib.h>

char code[102][1005];//储存字符串
int n, m;
int ans = 1005, distance;//ans为最后的最小值，初始值为1005
int main(void) {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++)
        scanf("%s", code[i]);

    for (int i = 0; i < m; i++) {
        for (int j = i + 1; j < m; j++) {
            distance = 0;//注意变量的重新赋值
            for (int k = 0; k < n; k++)
                distance += (code[i][k] != code[j][k]);

            if (distance < ans)ans = distance;//更新最小值
        }
    }
    printf("%d\n", ans);

    return 0;
}
```

N dch做不出毕业设计

这是一个很规范的二分法求根模版。由于题目中函数是单调递减的，所以当`f(mid)>y`，令`L=mid`；反之亦然。

题目的输出要求6位小数的精度，所以程序结束时的区间长度应该小于`1e-6`，选用`1e-8`即可。

由于本题中的函数很特殊，牛顿法、弦截法均不能AC，而二分法则是工程上最简单，最稳定的一种算法，其缺点是速度较慢。

```
#include<stdio.h>
#include<math.h>

double pi;
double eps = 1e-8;

double fun(double x) {
    return (pi - 2 * x) / 2 / sin(x) / sin(x) - 1 / tan(x);
}
```

```
int main() {
    double y, L, R, mid;
    pi = acos(-1);
    scanf("%lf", &y);
    L = 0, R = pi / 2;
    while ((R - L) > eps) {
        mid = (R + L) / 2;
        if (fun(mid) > y) {
            L = mid;
        } else {
            R = mid;
        }
    }
    mid = (R + L) / 2;
    printf("%0.6f", mid);
    printf("\n%0.10f", mid);
    return 0;
}
```

O 轰炸

本题可定义两个空间至少为n*m的数组进行记录每一个坐标被轰炸的次数以及最后一次轰炸的轮数，首先对数组进行初始化，接下来对于每一个被轰炸过的坐标对应的数组+1，并且记录论述，最后输入关键点的坐标，若关键点对应的值非0，则输出`Y`以及对应的两个整数，如果关键点对应的值为0，则输出`N`。

```
#include<stdio.h>

int main()
{
    int n, m, x, y,a[310][310]={},x1,y1,x2,y2,xi,yi,b[310][310];
    scanf("%d%d%d%d", &n, &m, &x, &y);
    for (int k = 1; k <= x;k++)
    {
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        for (int i = x1; i <= x2;i++)
            for (int j = y1; j <= y2;j++)
            {
                a[i][j]++;
                b[i][j] = k;
            }
    }
    while(y--)
    {
        scanf("%d%d", &xi, &yi);
        if(a[xi][yi]>0)
            printf("Y %d %d\n", a[xi][yi], b[xi][yi]);
```

```
        else
            printf("N\n");
    }
}
```

P 实矩阵乘法

```
#include <stdio.h>
long long ans[15][15],mul1[15][15],mul2[15][15];
int main()
{
    int n;
    int a[15],b[15];
    int i,j,k,l;
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        scanf("%d%d",&a[i],&b[i]);
    }
    for(i=1;i<=a[1];i++)
    {
        for(j=1;j<=b[1];j++)
        {
            scanf("%lld",&mul1[i][j]);
        }
    }
    int hang1=a[1];
    int lie1,lie2;
    for(i=2;i<=n;i++)
    {
        for(j=1;j<=14;j++)
        {
            for(k=1;k<=14;k++)
            {
                ans[j][k]=0;//重新初始化ans
            }
        }
        for(j=1;j<=a[i];j++)
        {
            for(k=1;k<=b[i];k++)
            {
                scanf("%lld",&mul2[j][k]);
            }
        }
        lie1=a[i];//b[i-1]亦可
        lie2=b[i];
        for(l=1;l<=hang1;l++)
```

```
        {
            for(j=1;j<=lie2;j++)
            {
                for(k=1;k<=lie1;k++)
                {
                    ans[l][j]+=mul1[l][k]*mul2[k][j];
                }
            }
        }
        for(j=1;j<=hang1;j++)
        {
            for(k=1;k<=lie2;k++)
            {
                mul1[j][k]=ans[j][k];
            }
        }
    }
    for(i=1;i<=hang1;i++)
    {
        for(j=1;j<=lie2;j++)
        {
            printf("%lld ",ans[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Q 二荷学姐买基金

- 考查:循环 数据范围,即使所有输入数据都在`int`范围内，但加和可能超过`int`范围.
- 思路:脑筋急转弯.本题由于不限制交易次数,因此可以想象每天都买入并卖出(实际并不是这样).只求所有上坡数据的两端之差的和即可(只要今天比昨天大就卖出).
- 实际上,上述就是一种非常简单的贪心算法.由于算法已经超出本课程的内容,有兴趣的同学可以自行百度.也可以用动态规划(DP)的思路求解.预告一下,此题还有后续哦~
- 可以不用数组

```
#include<stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    long long a,b,i,ans=0;
    scanf("%lld",&a);
    for(i=2;i<=n;i++)
    {
        scanf("%lld",&b);
```

```
        if(b-a>0)
        ans+=(b-a);
        a=b;
    }
    printf("%lld",ans);
    return 0;
}
```

R 二荷学姐买基金（续）

本题可以直接枚举所有方案，下面是枚举的做法，时间复杂度 $O(n^2)$

```
#include<stdio.h>

long long value[10005];

int main()
{
    int n, i, j, L, R;
    long long cur, ans;
    while (scanf("%d", &n) != EOF)
    {
        ans = -1;
        for (i = 0; i < n; i++)
        {
            scanf("%lld", &value[i]);
        }
        for (i = 1; i < n; i++)
        {
            for (j = i; j < n; j++)
            {
                cur = value[j] - value[i - 1];
                if (cur < 0)
                {
                    continue;
                }
                if (cur > ans)
                {
                    ans = cur;
                    L = i - 1;
                    R = j;
                }
            }
        }
        if (ans <= 0)
        {
            printf("No profit!\n");
```

```
        }
        else
        {
            printf("%lld %d %d\n", ans, L + 1, R + 1);
        }
    }
}
```

提供一个$O(n)$的做法，可供参考。

```
#include<stdio.h>
#include<string.h>

typedef long long ll;
ll stock[10010];
int skip[10010];
ll res, cur, start, finish;
int n;

int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; ++i) scanf("%lld", stock + i);
        memset(skip, 0, sizeof(skip));
        res = cur = start = finish = 0;

        for (int i = 1; i <= n; ++i) {
            if (skip[i]) continue;
            for (int j = i + 1; j <= n; ++j) {
                if (stock[j] <= stock[i]) {
                    i = j;
                    continue;
                }
                skip[j] = 1;
                if (stock[i] > stock[j]) continue;
                cur = stock[j] - stock[i];
                if (cur > res)
                    res = cur, start = i, finish = j;
            }
        }
        if (res > 0) printf("%lld %lld %lld\n", res, start, finish);
        else puts("No profit!");
    }
}
```

S 另类合成

本题是字符串+指针的考察；将字符串new覆盖字符串old可以使用strcpy函数，该函数会将一个字符串复制到，也可以理解为覆盖到一个字符数组，并在最后添加空字符，用法如下

```
strcpy(old,new);
```

如果想要在old的指定位置开始进行覆盖的话，改变strcpy函数中的第一个参数的地址即可，例

```
strcpy(old+1,new);//从old[1]开始，用字符串new进行覆盖，并在末尾添加空字符
```

因此，对于本题我们的写法就可以写成如下

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#define MAX_strLEN 10000
#define MAX_allLEN 100000
char old[1000000],new[100000];
int point;
int main(void)
{

    scanf("%s",old);

    while(scanf("%s",new)!=EOF)
    {
        scanf("%d",&point);
        strcpy(old+point,new);
    }
    puts(old);

    return 0;
}
```

T 逆序输出字符串

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>

char buffer[100][1020] = {0};

void reverse(char *str)
{
```

```
    int n = strlen(str);
    char tmp;
    int lb = 0,ub = n - 1;
    while(lb < ub)
    {
        tmp = str[lb];
        str[lb] = str[ub];
        str[ub] = tmp;
        lb++;
        ub--;
    }
}

int main()
{
    int cur = 0;
    while(gets(buffer[cur]))
    {
        reverse(buffer[cur]);
        cur++;
    }
    while(cur--)
    {
        puts(buffer[cur]);
    }
    return 0;
}
```

U NBSI码

按照题意，将每一位数字分别乘上1、2、3.....9，求和后将结果%11，然后判断识别码和它是否相同即可。

本题没有特别的坑点，只需要注意将char型的数字转换成int型的数字。

```
#include<stdio.h>
#include<string.h>

int res, len;
char s[15];

int main(){
 int ord=1, i;

 scanf("%s", s);

 len = strlen(s);
 for(i=0; i<len-1; ++i)
```

```
     if(s[i] != '-'){
      res += (s[i]-'0')*ord;
      ord ++;
     }
 res %= 11;
 if((s[len-1]!='X'&&res==s[len-1]-'0') || (s[len-1]=='X'&&res==10))
     printf("Right");
 else{
  for(i=0; i<len-1; ++i)
      printf("%c", s[i]);
  if(res == 10)
   printf("X");
  else    printf("%d", res);
 }
    return 0;
}
```

从这题往后题目难度有所增加

V 小迷弟的破镜重圆

本题需要我们快速计算一个区间 [L , R] 内数的和，如果我们每次都从a[L] 加到a[R] 。这样计算自然是很慢的，但如果我们提前计算出a[1]（这里我们假设数组下标从1开始）到a[t] 的和计算出来记为sum[t] 。 当我们需要a[L] 加到a[R] 的值时，我们只需要在之前预处理出的答案数组中找到sum[R]和sum[L-1]，二者做差就可以避免大量重复计算了（因为往往我们对于不同区间的和，都要重复计算很多次这些相加）。 于是我们预处理完毕，每次只需要看sum[i]是否和sum[j]模m同余就好了。

```
#include<stdio.h>
int a[50010],fir[50010],las[50010],sum[50010],n,m;
int main() {
    scanf("%d %d",&n,&m);
    int i,j,ans=0;
    for(i=1; i<=n; i++) {
        scanf("%d",&a[i]);
        sum[i]=(sum[i-1]+a[i])%m;//这里提前取模可以让程序效率++
        if(!sum[i]) ans=i;
    }
    for(i=1; i<=n; i++) {
        for(j=i; j<=n; j++) {
            if(sum[i]==sum[j]) {
                if(j-i>ans) {
                    ans=j-i;
                }
            }
        }
    }
    printf("%d",ans);
```

```
    return 0;
}
```

W 解谜小游戏

课件原题

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#define eps 1e-8
struct data {
    double value;
    int order;
    int rank;
    char str[100];
} list[500005];

int cmp1(const void *p1, const void *p2)
{
    if(fabs(((struct data *)p1)->value-((struct data *)p2)->value)<eps) return 0;
    else if(((struct data *)p1)->value>((struct data *)p2)->value) return 1;
    else return -1;
}

int cmp2(const void *p1, const void *p2)
{
    if(((struct data *)p1)->order==((struct data *)p2)->order) return 0;
    else if(((struct data *)p1)->order>((struct data *)p2)->order) return 1;
    else return -1;
}

int main()
{
    int n=0,i;
    while(scanf("%s",list[n].str)!=EOF)
    {
        list[n].order=n+1;
        list[n].value=atof(list[n].str);
        n++;
    }
    printf("%d\n",n);
    qsort(list,n,sizeof(struct data),cmp1);
    list[0].rank=1;
    for(i=1;i<n;i++)
```

```
    {
        if(fabs(list[i].value-list[i-1].value)<eps) list[i].rank=list[i-1].rank;
        else list[i].rank=i+1;
    }
    qsort(list,n,sizeof(struct data),cmp2);
    for(i=0;i<n;i++)
    {
        printf("%d: %s\n",list[i].rank,list[i].str);
    }
    return 0;
}
```

X ZJD学组合数

这个题目如果使用阶乘相除的方法会有溢出的问题，题目中已经暗示过要使用杨辉三角，或公式 $C_{n+1}^{m}=C_{n}^{m-1}+C_{n}^{m}$，来解决问题。

使用一维滚动数组循环

```
#include <stdio.h>
int main(void) {
    int a,n,m,i,j;
    unsigned long long ar1[201]= {0};
    unsigned long long ar2[201]= {0};
    ar2[0]=1;
    scanf("%d",&a);
    while(~scanf("%d%d",&n,&m)) {
        if(m<n) {
            printf("You're kidding me!\n");
            continue;
        }
        for(i=1; i<=n; i++) {
            ar1[0]=1,ar1[i]=1;
            for(j=1; j<=i; j++)
                ar1[j]=ar2[j]+ar2[j-1];
            for(j=0; j<=i; j++)
                ar2[j]=ar1[j];
        }
        printf("%llu\n",ar1[m]);
        for(i=0; i<201; i++)    ar1[i]=0,ar2[i]=0;
        ar2[0]=1;
    }

    return 0;
}
```

使用递归

```c
#include <stdio.h>
typedef unsigned long long ull;
ull arr[101][101]= {0};
ull Initial(int n,int m)
{
    if(arr[n][m]==0)
    {
        if(m==0||m==n) return arr[n][m]=1;
        else return arr[n][m]=Initial(n-1,m-1)+Initial(n-1,m);
    }
    else return arr[n][m];
}
int main(void) {
    int a,n,m,i,j;
    scanf("%d",&a);
    arr[0][0]=1;
    while(~scanf("%d%d",&n,&m)) {
        if(m>n) {
            printf("You're kidding me!\n");
            continue;
        }
        printf("%llu\n",Initial(n,m));
    }
    return 0;
}
```

质因数分解

```c
#include <stdio.h>

// 答案这个数中，每个因数有几个如ins[]={0,0,2,1,0,3}表示它有2个2，1个3，3个5等于1500
int ans[200] = {0};

// 将一个(num)数字分解质因数，放到ans中。如flag为1，表示加上这个质因数，为-1表示减去
void inNum(int num, int flag)
{
    int in = 2;
    while (num >= 2)
    {
        while (num % in == 0)
        {
            ans[in] += flag;
            num /= in;
        }
        in++;
    }
}
```

```
// 求组合数的程序
unsigned long long fun(int n, int m)
{
    int i, j;
    unsigned long long res = 1;
    // 按照组合数计算公式，将n!的所有质因数放到ans中
    for (i = 2; i <= n; ++i)
    {
        inNum(i, 1);
    }
    // 除去m!的因子
    for (i = 2; i <= m; ++i)
    {
        inNum(i, -1);
    }
    // 除去(n-m)!的因子
    for (i = n - m; i >= 2; --i)
    {
        inNum(i, -1);
    }

    // 将所有因子乘起来，得到答案
    for (i = 2; i <= n; ++i)
    {
        for (j = 0; j < ans[i]; ++j)
        {
            res *= i;
        }
    }
    return res;
}

int main()
{
    int a, i, j;
    int n, m;
    scanf("%d", &a);
    for (i = 0; i < a; ++i)
    {
        scanf("%d %d", &n, &m);
        // 初始化，也可以用memset函数（在string.h中）
        for (j = 0; j <= n; ++j)
        {
            ans[j] = 0;
        }
        if (n < m)
        {
            printf("You're kidding me!\n");
        }
```

```
        else
        {
            printf("%llu\n", fun(n, m));
        }
    }
}
```

Y 小雷枚举排列数

本题主要考查**递归**的基础知识，是一个很好的模板题，在以后的进阶题型中很常见。

基本思路是先预留待填充的数组，利用递归逐位向数组里填充数，只要之前没有用过的数都可以填充进待填充的位置。填充满n位后输出结果。

这里的输出结果在今后的进阶题目中可能很有用。

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

int a[1000];//存放生成的数
int book[1000]; //表示递归生成到哪一位置的数

void amn_out(int pos, int m, int n)//向a[pos]填数 m中取n
{

    if (pos == n + 1)//递归边界
    {
        //此时a[1]-a[n]填满数 顺次输出为待求全排列
        for (int j = 1; j <= n - 1; j++) {
            printf("%d ", a[j]);
        }
        printf("%d\n", a[n]);

        return;
    }
    for (int i = 1; i <= m; i++) {
        if (book[i] == 0)//如果i没有在数组里面
        {
            a[pos] = i;
            book[i] = 1;
            amn_out(pos + 1, m, n);//向a[pos+1]填数
            book[i] = 0;//收回，试下一个数

        }
    }
}
```

```c
int main() {
    int m, n;
    scanf("%d%d", &m, &n);
    amn_out(1, m, n);

    return 0;
}
```

Z 日期指示器

日子要一天一天过才妥当。当然，要计算也不是不可以，麻烦就是。

```c
#include <stdio.h>
#include <stdlib.h>
int isleap(int);
int main()
{
    // freopen("C:\\Users\\jingx\\Projects\\C_Practice\\in.txt", "r", stdin);
    int year1, month1, day1, year2, month2, day2;
    int MonthOfYear[2][13] = {{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}, {0,
31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};
    int n;

    scanf("%d", &n);
    while (n--)
    {
        int sum = 0;
        scanf("%d-%d-%d %d-%d-%d", &year1, &month1, &day1, &year2, &month2, &day2);
        if (year1 == year2)
        {
            if (month1 == month2)
            {
                if (day1 == day2)
                {
                    printf("Today\n");
                }
                else if (day1 == day2 - 1)
                {
                    printf("Yesterday\n");
                }
                else
                {
                    printf("%d days ago\n", day2 - day1);
                }
            }
            else
            {
```

```
                sum -= day1;
                int t = month1, tt = isleap(year1);
                while (t != month2)
                {
                    sum += MonthOfYear[tt][t];
                    t++;
                }
                sum += day2;
                if (sum == 1)
                    printf("Yesterday\n");
                else
                    printf("%02d-%02d %d days ago\n", month1, day1, sum);
            }
        }
        else
        {
            int y = year1, yy;
            int t = 1;
            yy = isleap(year1);
            while (t != month1)
            {
                sum -= MonthOfYear[yy][t];
                t++;
            }
            sum -= day1;
            t = 1;
            while (y != year2)
            {
                if (isleap(y))
                    sum += 366;
                else
                    sum += 365;
                y++;
            }
            yy = isleap(year2);
            while (t != month2)
            {
                sum += MonthOfYear[yy][t];
                t++;
            }
            sum += day2;
            if (sum == 1)
                printf("Yesterday\n");
            else
                printf("%d-%02d-%02d %d days ago\n", year1, month1, day1, sum);
        }
    }
    return 0;
}
```

```c
int isleap(int year)
{
    if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0))
        return 1;
    return 0;
}
```

BA 递归括号匹配

这是一道不难的填空题。只要你掌握了递归的原理，很容易完成填空：

```c
#include<stdio.h>

int Parenthesis()
{
    char c=getchar();
    if(c==')'||c==']'||c=='}'||c==EOF)
    {
        ungetc(c,stdin);//与getchar相反，向读入中退回一个字符
        return 1;
    }
    else if(c=='(')
    {
        int temp=Parenthesis();
        if(temp==0)//调用匹配失败
        {
            return 0;
        }
        c=getchar();
        if(c!=')')
        {
            return 0;
        }
        temp=Parenthesis();
        if(temp==0)
        {
            return 0;
        }
        return 1;
    }
    else if(c=='[')
    {
        int temp=Parenthesis();
        if(temp==0)
        {
            return 0;
        }
```

```
        c=getchar();
        if(c!=']')
        {
            return 0;
        }
        temp=Parenthesis();
        if(temp==0)
        {
            return 0;
        }
        return 1;
    }
    else if(c=='{')
    {
        int temp=Parenthesis();
        if(temp==0)
        {
            return 0;
        }
        c=getchar();
        if(c!='}')
        {
            return 0;
        }
        temp=Parenthesis();
        if(temp==0)
        {
            return 0;
        }
        return 1;
    }
}

int Matching()
{
    int ans=Parenthesis();//括号匹配
    if(ans==0)
    {
        return 0;
    }
    char c=getchar();//匹配完应该读完
    if(c!=EOF)
    {
        return 0;
    }
    return 1;
}

int main()
```

```
{
    int ans=Matching();
    if(ans==1)
    {
        printf("AC\n");
    }
    else
    {
        printf("CE\n");
    }
}
```

BB 开除记者

这题属于递归的应用；对于边长为 n 的方阵，在第 i 行，第 j 列的记者；我们定义函数 fire(int i, int j, int n) 来表示此记者的情况，如果函数值为 0 表示被开除，函数值为 1 表示不被开除，通过观察我们可以发现：

- 如果 n=1 ，应该返回 1 ；
- 如果 i≤n/2 并且 j≤n/2 ，那么应该返回 0 ；
- 如果 i≤n/2 并且 j>n/2，那么就进行到下一步分解，是否开除的情况等同于 方阵边长为 n/2 、记者位于第 i 行，第 j−n/2 列，所以返回 fire(i, j-n/2, n/2)；
- 同样的，如果 i>n/2 并且 j≤n/2 ，那么应该返回 fire(i-n/2, j, n/2)；
- 如果 i>n/2 并且 j>n/2 ，那么应该返回 fire(i-n/2, j-n/2, n/2) 。

```
#include<stdio.h>

int n;

int fire(int a,int b,int N)//表示方阵边长为N，记者位于第i行，第j列的情况
{
    int half=N/2;
    if(N==1)return 1;
    if(a<=half && b<=half)return 0;
    if(a<=half && b>half)return fire(a,b-half,half);
    if(a>half && b<=half)return fire(a-half,b,half);
    return fire(a-half,b-half,half);
}

int main(void)
{
    while(scanf("%d",&n)!=EOF)
    {
        int all=1<<n;
        for(int i=1; i<=all; i++)
        {
            for(int j=1; j<=all; j++)
                printf("%d ",fire(i,j,all));
```

```
            putchar('\n');
        }
        putchar('\n');
    }
    return 0;
}
```

示例代码2

```
#include<stdio.h>

char C[1100][1100];

int t;
int n,m;

int main()
{
    C[0][0]=1;
    int i,j;
    for (i = 1; i < 1100; ++i)
    {
        for (j = 0; j <= i; ++j)
        {
            C[i][j] =C[i - 1][j]^C[i - 1][j - 1];
        }
    }
    while(scanf("%d",&t)!=EOF)
    {
        int size=1<<t;
        for(i=0;i<size;i++)
        {
            for(j=size-1;j>=0;j--)
            {
                printf("%d ",C[i][j]);
            }
            printf("\n");
        }
        printf("\n");
    }
}
```

BC 两面包夹芝士

本题主要考察函数递归以及记忆化的方法。

假设荷沸水的总量为 x 个单位，那么净化 x 个单位的荷沸水所需要的最小费用可以用函数 $f(x)$ 来表示：

$$f(x) = min \begin{pmatrix} f(x-1)+c1 & \text{whenever} \\ f(\frac{x}{2})+c2 & \text{only when } x \ mod \ 2 = 0 \\ f(\frac{x}{3})+c3 & \text{only when } x \ mod \ 3 = 0 \end{pmatrix}$$

其中 min 代表最小值函数，mod 代表求余运算。同学们看到这里应该已经意识到可以使用递归来解决这个问题了，但是直接递归可能会导致 TLE，因为直接递归可能会对同一情况进行多次运算，导致不必要的时间开销。比如，计算 $f(12)$ 的时候会计算 $f(6)$ 和 $f(3)$，计算其中的 $f(6)$ 的时候又会计算 $f(3)$，可以发现 $f(3)$ 在这个情况中至少运算了两次，其中第二次计算实际上是不必要的，因为之前计算过一次了。因此我们可以开一个数组存储计算过的 $f(x)$，在递归函数即检查荷沸水的总量为 x 个单位时的最小花费是否被计算过，若计算过直接返回结果即可。

当然这道题目也可以采用循环来计算（动态规划），状态转移方程和题解开头给出的公式一样。

示例代码——递归

```
#include <stdio.h>
int s[100007], c1, c2, c3;

int dfs(int x) {
    if (s[x] != 0) {
        return s[x];
    }

    int min = dfs(x - 1) + c1;
    if (x % 2 == 0) {
        int r = dfs(x / 2) + c2;
        min = r < min ? r : min;
    }
    if (x % 3 == 0) {
        int r = dfs(x / 3) + c3;
        min = r < min ? r : min;
    }

    return s[x] = min;
}

int main()
{
    int n;
    scanf("%d %d %d", &c1, &c2, &c3);
    s[1] = c1;
    while (~scanf("%d", &n)) {
        printf("%d\n", dfs(n));
    }
}
```

示例代码——循环

```
#include <stdio.h>
```

```
int s[100007], c1, c2, c3;

int main()
{
    int n;
    scanf("%d %d %d", &c1, &c2, &c3);
    s[1] = c1;
    for (int i = 2; i <= 100000; i++) {
        s[i] = s[i - 1] + c1;
        if (i % 2 == 0) {
            int r = s[i / 2] + c2;
            s[i] = r < s[i] ? r : s[i];
        }
        if (i % 3 == 0) {
            int r = s[i / 3] + c3;
            s[i] = r < s[i] ? r : s[i];
        }
    }
    while (~scanf("%d", &n)) {
        printf("%d\n", s[n]);
    }
}
```

BD PlayFair

本题按照题目要求，首先需要定义一个 5×5 的矩阵作为密码表，并将密钥填充进去。同时需要一个数组统计哪些字母未出现过，将其按顺序编入密码表。对于多次重复使用的功能 `在密码表中找到X坐标` 、`在密码表中找到Y坐标` 和 `加密明文` 可以通过自定义函数的方式实现，以减少重复代码。

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>

char c1, c2;
char mima[5][5];

int findX(char p)//在密码表中找到该字母的x坐标
{
    int i, j;
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (mima[i][j] == p) return i;
        }
    }
}
```

```
int findY(char p)//在密码表中找到该字母的Y坐标
{
    int i, j;
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (mima[i][j] == p) return j;
        }
    }
}

void jiami(char p1, char p2) {
    if (findX(p1) == findX(p2))//p1 p2在同一行的情况
    {
        if (findY(p1) == 4) printf("%c", mima[findX(p1)][0]);//针对最后一行单独讨论
        else printf("%c", mima[findX(p1)][findY(p1) + 1]);
        if (findY(p2) == 4) printf("%c", mima[findX(p2)][0]);//针对最后一行单独讨论
        else printf("%c", mima[findX(p2)][findY(p2) + 1]);
    } else if (findY(p1) == findY(p2))//p1 p2在同一列的情况
    {
        if (findX(p1) == 4) printf("%c", mima[0][findY(p1)]);//针对最右一列单独讨论
        else printf("%c", mima[findX(p1) + 1][findY(p1)]);
        if (findX(p2) == 4) printf("%c", mima[0][findY(p2)]);//针对最右一列单独讨论
        else printf("%c", mima[findX(p2) + 1][findY(p2)]);
    } else printf("%c%c", mima[findX(p2)][findY(p1)], mima[findX(p1)][findY(p2)]);
}

int main() {
#ifdef DEBUG
    freopen("in2.txt","r",stdin);
    freopen("out2.txt","w",stdout);
#endif
    char s[20], in[300];
    int zimu[25] = {0};
    int i, j = 0, k = 0;
    scanf("%s", s);
    scanf("%s", in);
    for (i = 0; i < strlen(s); i++)//将密钥编入密码表
    {
        if (!zimu[tolower(s[i]) - 'a'])//确认密钥中的字母没有出现过
        {
            zimu[tolower(s[i]) - 'a']++;
            mima[j][k] = tolower(s[i]);
            j++;
            if (j == 5) {
                j = 0;
                k++;
            }
        }
```

```
    }
    for (i = 0; i < 25; i++)//将密钥中未出现的字母编入密码表
    {
        if (!zimu[i]) {
            mima[j][k] = i + 'a';
            j++;
            if (j == 5) {
                j = 0;
                k++;
            }
        }
    }
    for (i = 0; i < strlen(in);)//对明文进行加密
    {
        if (i == strlen(in) - 1)//明文仅剩一个字母，需要补字母x
        {
            jiami(tolower(in[i]), 'x');
            break;
        } else if (tolower(in[i]) == tolower(in[i + 1]))//明文字母相同，需要插入字母x
        {
            jiami(tolower(in[i]), 'x');
            i++;
        } else //一般情况
        {
            jiami(tolower(in[i]), tolower(in[i + 1]));
            i += 2;
        }
    }
    return 0;
}
```

BE 晶钻与明珠

这个题用了不太容易的想法。算法的前半段（贪心算法）已经写在题中了，但是后半段需要自己思考。

题中给出的的算法是，只要有晶钻就买晶钻，没有晶钻就买明珠，什么都没有就把明珠换成晶钻。同时也指出了这个原始算法的问题：按照这个算法，最后有可能会剩下明珠。

既然如此，不妨定量分析：假设剩下了x颗明珠。根据之前的算法，这意味着：之前存在没有晶钻的x天，用来买明珠了。

那么考虑天数x的奇偶性之后，**修改这x天的策略**，最多还能再多得到x/2颗晶钻。前一半天数买，后一半天数换，如果是奇数的话多余一天什么也做不了。

于是假设原始算法执行完，得到y颗晶钻与x颗明珠，那么最终答案就是y+x/2。

```
#include<stdio.h>
```

```
int t,n;
char s[2000005];

int main()
{
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d",&n);
        scanf("%s",s);
        int cnt1=0,ans=0;
        int i;
        for(i=0;i<n;i++)
        {
            switch(s[i])
            {
            case'0':
                if(cnt1>0)
                {
                    ans++;
                    cnt1--;
                }
            break;
            case'1':
                cnt1++;
            break;
            default:
                ans++;
            }
        }
        printf("%d\n",ans+cnt1/2);
    }
    return 0;
}
```

BF 教科书般的亵渎（位运算)

```
#include<stdio.h>
int main(){
    long long a[1000]={0};
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%lld",&a[i]);
    }
    int judge1=1,judge2=1,times=0;
```

```
    while(judge1&&judge2)
    {
        judge1=0;
        judge2=0;
        for(int i=0;i<n;i++)
        {
            if(a[i]!=0)
            {
                a[i]>>=1;
                if(a[i])judge1=1;
                if(!a[i])judge2=1;
            }
        }
        times++;
    }
    for(int i=0;i<n;i++)
    {
        if(a[i])
        {
            judge1=2;
            break;
        }

     }
    if(judge1==2)
    {
        printf("%d\n",times);
        printf("Some number left...");
    }
    else
    {
        printf("%d\n",times);
        printf("Clear!");
     }
    return 0;
}
```

BG 狗的寻友计划

```
#include<stdio.h>
#include<string.h>
char maze[105][105];
int book[10005][2];
int D[2],L[2];
int get(int x,int y,int n)
{
    if(x<(n-1)&&maze[x+1][y]=='L')return 1;
```

```
    else if(y<(n-1)&&maze[x][y+1]=='L')return 1;
    else if(x>0&&maze[x-1][y]=='L')return 1;
    else if(y>0&&maze[x][y-1]=='L')return 1;
    return 0;
}
void Find_way(int x,int y,int n,int step,int last_x,int last_y)
{
    book[step][0]=x;
    book[step][1]=y;
    //printf("%d%d\n",x,y);
    if(get(x,y,n))
    {
        for(int i=1;i<=step;i++)
        {
            if(maze[book[i][0]][book[i][1]]=='0')maze[book[i][0]][book[i][1]]='S';
        }
        return;
    }
    book[step][0]=x;
    book[step][1]=y;
    step++;
    if(x<(n-1)&&maze[x+1][y]=='0'&&x+1!=last_x)Find_way(x+1,y,n,step+1,x,y);
    if(y<(n-1)&&maze[x][y+1]=='0'&&y+1!=last_y)Find_way(x,y+1,n,step+1,x,y);
    if(x>0&&maze[x-1][y]=='0'&&x-1!=last_x)Find_way(x-1,y,n,step+1,x,y);
    if(y>0&&maze[x][y-1]=='0'&&y-1!=last_y)Find_way(x,y-1,n,step+1,x,y);
    book[step][0]=0;
    book[step][1]=0;
    return;
}
int main(){
    int n,i,j;
    char laji[5];
    scanf("%d",&n);
    scanf("%d%d%d%d",&D[0],&D[1],&L[0],&L[1]);
    gets(laji);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        maze[i][j]=getchar();
        gets(laji);
    }
    Find_way(D[0],D[1],n,0,D[0],D[1]);
    for(i=0;i<n;i++)
    {
        puts(maze[i]);
     }
}
```