第 1 章 基础知识

练习 1：月份天数

【解题思路】

本题只需要对 2 月的特殊情况进行单独处理，然后将 30 天和 31 的月份分开合并考虑，根据题意编写即可。

```
#include<iostream>
using namespace std;
bool leap(int year)
{
   return (year%4==0 && year%100!=0 || year%400==0);
}
int main()
{
   int feb;
   int year;
   int mon;
   int day;
   cin>>year>>mon;
   if(leap(year))
     feb =29;
   else
     feb =28;
    switch(mon){
      case 1:case 3:case 5:case 7:case 8:case 10:case 12: day=31;break;
      case 4:case 6:case 9:case 11:day=30;break;
      case 2:
          day=feb;break;
   }
   cout<<day<<endl;
   return 0;
}
```

练习 2：ISBN 号码

【解题思路】

本题根据题意编写即可，注意 int 和 char 之间的转换规则。

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
```

```
int main()
{
    int i, j, n, sum, id;
    int a[15];
    char str[15];
    cin>>str;   //输入 ISBN 号

    j = 0;
    for(i=0; i<11; i++)
    {
        if(i!=1 && i!=5)
        {                          //将存储 ISBN 号的 str[]中数字字符转化为对应的
            a[j] = str[i]-48;   //十进制数赋值给 a[]（不包括最后一位识别码）
            j++;
        }
    }

    sum = 0;
    n = 1;
    for(i=0; i<9; i++)
    {
        sum += a[i]*n;  //对 ISBN 中 9 个数字做算术计算并把结果赋值给 sum
        n++;
    }

    id = sum % 11;  //由 ISBN 中九个数字计算出识别号 IDcode

    /*判断计算出的 id 是否等于 ISBN 的识别号*/
    if(id == str[12]-48 || id == str[12]-78)    //如果计算出的 id 是 1~9，则是 IBSN 字符 str[12]对应的十进制数字
    {                          //如果 id 是 10 则是 ISBN 字符 str 中的字符 X 对应的十进制数
        cout<<"right";
    }
    else    //如果识别号不正确
    {
        j = 0;
        /*先输出除最后一位 IDcode 外的前 12 位字符*/
```

```
        for(i=0; i<12; i++)

        {
            cout<<str[i];
        }
        /*输出最后一位识别码*/
            if(id == 10)    //如果计算出的识别码是 10 则转化为 X
            {
                cout<<'X';
            }
            else    //如果不是 10，则输出计算出的识别码 id
            {
                cout<<id;
            }
    }
    return 0;
}
```

练习 3：十进制转十六进制

【解题思路】

本题优先考虑是否质数，质数的分解只有自己，然后再对其他数字的情况进行拆解，用一个全局变量进行累加，求出结果。

```
#include <iostream>
#include <vector>
using namespace std;
int sum=0;
// 判断是否是质数
bool IsPrime(int n)
{
   int i;
   for (i = 2; i <= n/2; ++i)
      if (n%i == 0)
         return false;

   if (i > n/2)
      return true;
   else
      return false;
}

int main()
```

```
{
    vector<int> v; // 存储 2-b 的所有质数
    int a,b;
    cin >> a >> b;
    for (int i = 2; i<=b; ++i) // 存储质数过程
    {
        if (IsPrime(i))
            v.push_back(i);
    }
    for ( int i = a; i <= b; ++i) // 从 a 开始处理直到 b
    {
        if (IsPrime(i))
            sum++;
        else  // 不是质数分别处理
        {
            int temp = i;// 暂存 i
            int index = 0; //存储质数的数组下标 索引
            while (temp != 1)  // 当前数字没有被除尽时继续
            {
                if (temp%v[index] == 0)  // 从第一个质数开始除
                {
                    sum++;
                    temp /= v[index];
                    index = 0; // 还原    即 继续从第一个质数 2 开始尝试

                }
                else  // 不能整除的话尝试下一个质数
                    index++;
            }
        }
    }
    cout <<sum<< endl;
    return 0;
}
```

练习 4：十进制转十六进制

【解题思路】要考虑转化之后最高位是 0 的情况 具体思路代码体现

【参考代码】

```
#include <iostream>
using namespace std;
```

```
int main()
{
	int a;
	cin >> a;
	char b[100];
	int i = 0;
	if (a == 0)
		cout << '0';
	else
	{
		while (a != 0)
		{
			if (a % 16 >= 10)
				b[i++] = a % 16 + 'A' - 10;
			else
			{
				b[i++] = a % 16 + '0';
			}
			a = a / 16;
		}
	}
	for (int k = i - 1; k >= 0; k--)
		cout << b[k];
	return 0;
}
```

练习 5：回文数

【解题思路】

将输入的 n 进制数转换为一个数组，里面存放 n 进制数的 10 进制表示的每一位，然后第 i 位与第 len-i+1 位相加，得到一个结果后，再转换为 n 进制，实现 n 进制数正序与逆序相加， 得到一个新的 n 进制数；再判断是否是回文数，如果是回文数则输出 step，如果不是则 step++，继续判断。

【参考代码】

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
int n, step;
string m;
int num[1005];
//判断是否为回文数
```

```
bool judge(int x) {
    for (int i = 1; i <= x/2; ++i) {
        if (num[i] != num[x - i + 1])
            return false;
    }
    return true;
}
//n进制数相加
int add(int x) {
    int ans[1005] = {0};
    for (int i = 1; i <= x; ++i) {
        ans[i] = num[i] + num[x - i + 1] + ans[i];
        ans[i+1] += ans[i]/n;
        ans[i] %= n;
    }
    if (ans[x + 1]) {
        x++;
    }
    for (int i = x; i >= 1; --i) {
        num[i] = ans[i];
    }
    return x;
}
int main() {
    cin >> n >> m;
    int len = m.size();
    //转换为十进制
    for (int i = 1; i <= len; ++i) {
        if (m[i - 1] < 65) {
            num[i] = m[i - 1] - '0';
        } else {
            num[i] = m[i - 1] - 'A' + 10;
        }
    }
    while(step <= 30) {
        if (judge(len)) {
            cout << "STEP=" << step << endl;
            return 0;
        }
        step++;
        len = add(len);
```

```
    }
    cout << "Impossible!" << endl;
    return 0;
}
```

练习 6：统计单词个数

【解题思路】

先用 getline()函数接受 s1、s2，然后用 tolower()函数将 s1,s2 全部转为小写，通过灵活运用 find()函数匹配 s1 的位置和个数

重点：1.熟练使用 getline()函数接受输入文本(包括空格)；
2.熟练使用 tolower()函数转换成小写；
3.find()函数的使用：参数类型以及返回数值；

【参考代码】

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string s1;
    string s2;
    getline(cin,s1);
    getline(cin,s2);

    for(int i=0;i<s1.length();i++)
    {
        s1[i]=tolower(s1[i]);
    }
    for(int i=0;i<s2.length();i++)
    {
        s2[i]=tolower(s2[i]);
    }

    s1= " "+s1+" ";
    s2= " "+s2+" ";

    if(s2.find(s1)==string::npos)     //在 s2 找到 s1 返回第一个字符的索引；如果没找到返回 string::npos
    {
        cout<<"-1"<<endl;
```

```
    }
    else
    {
        int fristp=s2.find(s1);
        int s=0;
        int beta=s2.find(s1);
        while(beta!=string::npos)
        {
            ++s;
            beta=s2.find(s1,beta+1);
        }
        cout<<s<<" "<<fristp<<endl;
    }
    return 0;
}
```

第 2 章 模拟法

练习 1：报数游戏

【解析】一道简单的模拟题：要模拟出现的情况，主要要推出一个规律在边界的位置时，下一个报数的位置就是 2*n-2；这样就可以推算出 每个编号下一个报数的位置；直接进行判断。

【参考程序】

```
#include <cstdio>
#include <cstring>
bool judge(int n)//判断函数，判断是否含 7，或者是 7 的倍数
{
     if( !(n % 7) ) return true;
     while(n)
    {
        if( n % 10 == 7 )
            return true;
        n /= 10;
    }
    return false;
}
int main()
{
    int n,m,k,i;
    while(scanf("%d%d%d",&n,&m,&k)&&n&&m&&k)
    {
        int count=0;
        for(i=7;;i++)
        {
            if(judge(i))
            {
                int x=i%(2*n-2);//得出的是初始编号
                if(x==0)
                 x=2*n-2;
                if(x==m || (x>=n && 2*n-x==m ))//符合题意的情况
                    count++;
                if(count==k)
                    break;
            }
        }
        printf("%d\n",i);
```

```
    }
return 0;
}
```

练习 2：猴子选大王

【解析】

用一个数组，数组中用 1 表示猴子在圈中，用 0 表示猴子已经出圈，数组下标对应与猴子编号对应（例如数组元素 p[0]值为 1，表示第 1 只猴子尚在圈中，即 p[i]代表编号为 i+1 的猴子是否在圈中）。

一只猴子出圈，则将对应的数组值置为 0；在报数过程中，要跨过值为 0 的猴子。 数到最后一只猴子时需要折回到下标为 0 的位置，可以用一个变量%m。猴子出圈后，还将对应元素的值置为 0。

【参考程序】

```
#include <stdio.h>
#define MaxSize 300
void king(int m,int n)
{
    int p[MaxSize];
    int i,j,t;
    for (i=0; i<m; i++)          //记录 m 只猴子在圈中
        p[i]=1;
    t=-1;                        //首次报数将从起始位置为 0，即第 1 只猴子开始，因为在使用 p[t]前 t 要加 1
    for (i=1; i<=m; i++)         //循环要执行 m 次，有 m 个猴子要出圈
    {
        j=1;                    // j 用于报数
        while(j<=n)  //
        {
            t=(t+1)%m;          //看下一只猴子,到达最后时要折回去,所以用%m
            if (p[t]==1) j++;     //仅当 q 猴子在圈中，这个位置才报数
        }
       p[t]=0;                 //猴子出圈

    }
    printf("%d",t);
    printf("\n");
}

int main()
{
```

```
    int m,n;
    while(scanf("%d %d", &m, &n))
    {
       if(m==0&&n==0)
       break;
       king(m,n);
    }

    return 0;
}
```

第 3 章 枚举法

练习 1：生日蜡烛

【解题思路】
题意可以看出，这是一道从某一个数开始连续累加到另一个数结束，其结果为 236，求第一个数的问题，我们可以利用求和公式：首项+末项乘以项数除以 2，再利用双重循环模拟时间来解决问题。
答案：26
【参考代码】

```
#include<stdio.h>
int main()
{
   int n,i,s=0;
   for(i=1;i<100;i++)
   {
       n=i;
       for(int j=i;j<100;j++)
       {
           s=(n+j)*(j-n+1);
       if(s==472)
       {
           printf("%d",n);
           break;
       }
       }
   }
   return 0;
 }
```

练习 2：奖券数目

【解题思路】
在 10000 到 99999 总共 90000 个数字当中，去除带有 4 的数字。我们就循环 90000 次，然后对每个数进行判断，如果有就计数，最后用 90000 减去计数，就是总共的奖券数。
答案：52488
【参考代码】

```
#include<stdio.h>
int n=0;
void f(int k)
{
```

```
        if(k%10==4 || k/10%10==4 || k/100%10==4 || k/1000%10==4 ||
k/10000==4)
        {
            n++;
        }
    }
    int main()
    {
        int i;
        for(i=10000;i<=99999;i++)
        {
            f(i);
        }
        printf("%d",90000-n);
        return 0;
     }
```

练习 3：不定方程求解

【解题思路】

给定 abc，找出关于 ax+by=c 的所有整数解，先利用循环，遍历其中一个数的所有可能性，再用公式表达出另一个数，并带入原公式进行判断，成立则符合条件。

【参考代码】

```
#include<stdio.h>
int  main()
{
        int  a,b,c;
        int  x,y;
        int  count=0;
        scanf("%d%d%d",&a,&b,&c);
        for(x=0;x<=c/a;x++)     //因为 x=(c-by)/a,所以 x<=c/a
        {
                y=(c-a*x)/b;          //y=(c-ax)/b
                if(a*x+b*y==c)
                        count++;
        }
        printf("%d\n",count);
        return  0;
}
```

练习 4：选数

【解题思路】
见代码注释。
【参考代码】
```
#include<iostream>
#include<math.h>
using  namespace  std;
int  x[20],n,k;//依照题目所设
bool  isprime(int  n){//判断是否质数
        int  s=sqrt(double(n));
        for(int  i=2;i<=s;i++){
                if(n%i==0)return  false;
        }
        return  true;
}
int  rule(int  choose_left_num,int  already_sum,int  start,int  end){
//choose_left_num 为剩余的 k，already_sum 为前面累加的和，start 和 end
为全组合剩下数字的选取范围；调用递归生成全组合，在过程中逐渐把 K 个数相
加，当选取的数个数为 0 时，直接返回前面的累加和是否为质数即可
        if(choose_left_num==0)return  isprime(already_sum);
        int  sum=0;
        for(int  i=start;i<=end;i++){
                sum+=rule(choose_left_num-1,already_sum+x[i],i+1,end);
        }
        return  sum;
}
int  main(){
        cin>>n>>k;
        for(int  i  =0;i<n;i++)cin>>x[i];
        cout<<rule(k,0,0,n-1);//调用递归解决问题
}
```

练习 5：火柴棍等式

【解题思路】
对于本题，我们把数字作为数组的参数，数组里放对应数所需的火柴数，就可以把每个数字所对应的火柴数表示出来，再利用循环暴力枚举所有可能性，把符合要求的三个数字相加，与最开始输入的总数相比，若相等答案+1，反之则继续进入循环，直到所有可能性都遍历结束，答案就出来了。

【参考代码】
```
#include  <stdio.h>
#include  <stdlib.h>
int  fun(int  x)
```

```
{
        int  num  =  0;
        int  f[]  =  {  6,  2,  5,  5,  4,  5,  6,  3,  7,  6  };
        while  (x  /  10  !=  0)
        {
                num  +=  f[x  %  10];
                x  =  x  /  10;
        }

        num  +=  f[x];
        return  num;


}

int  main()
{
        int  a  =  0;
        int  b  =  0;
        int  c  =  0;
        int  sum  =  0;
        int  m  =  0;
        scanf("%d",  &m);
        for  (a  =  0;  a  <=  1111;  a++)
        {
                for  (b  =  0;  b  <=  1111;  b++)
                {
                        c  =  a  +  b;
                        if  (fun(a)  +  fun(b)  +  fun(c)  ==  m-4)
                        {
                                sum++;
                        }
                }
        }

        printf("%d",  sum);
        return  0;
}
```

练习 6：比例简化

【解题思路】
这是一道数学题。这道题运用了一个“逼近”的思想，需要在可能的情况下将所有合法解无限地向差值尽可能小的情况逼近。来看一下数学上的推导过程。

A’/B’ >= A/B，

根据十字相乘法可以把除法运算变成整数乘法，在 int 中解决问题，即：

A’ * B >= B’*A

由于数据比较小，我们想到暴力枚举。最后把 ans1(a’)ans2(b’)的初值设置一下。开始跑两层循环，判断我们手写 gcd 函数，下一步便是难以考虑的：逼近。我们要维护最小差值，也就是一直更新 ans1,ans2，那么我们怎么判断什么时候更新 ans1,ans2 呢？

判断条件：

A’1/B’1 > A’2/B’2

即：

A’1*B’2 > A’2*B’1

【参考代码】

```
#include<cstdio>
using  namespace  std;
int  a,b,ans1,ans2,l;
int  gcd(int  x,int  y)
{
        if(y==0)
                return  x;
        return  gcd(y,x%y);
}
int  main()
{
        scanf("%d%d%d",&a,&b,&l);
        ans1=l;ans2=l;
        for(int  i=1;i<=l;i++)
                for(int  j=1;j<=l;j++)
                      if(gcd(i,j)==1  &&  i*b>=j*a  &&  i*ans2<j*ans1)
                                ans1=i,ans2=j;
        printf("%d  %d",ans1,ans2);
        return  0;
}
```

第四章 递推和递归

练习 1：组合数

【解析】递推公式 C(n,m)=C(n-1,m)+C(n-1,m-1)

【参考程序】

```
int f(int n, int m)
{
   if(m>n) return 0;
      if(m==0) return 1;
   return f(n-1,m-1) + f(n-1,m);
}
```

练习 2：李白打酒

【解析】

题目中有十次花，五次店，因为最后一次之后酒为 0，故最后一次应该遇到花。运用递归的思想，每次都具备两种可能性，遇花或者遇店。终止条件：遇到店后，最后一次遇到花时酒还剩一斗。

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
int sum=0;
void f(int a,int b,int c)
{
   if(a==0&&b==1&&c==1)
      sum++;
   if(a>0)
      f(a-1,b,c*2);
   if(b>0)
      f(a,b-1,c-1);
}
int main()
{
   int dian=5;
   int flo=10;
   int liquor=2;
   f(dian,flo,liquor);
   cout<<sum<<endl;
   return 0;
}
```

练习 3：组合数

【解析】

循环初始可以从 123 开始，是最小的且数字不重复的三位数。3 个三位数中最小的那一个，最大可以循环到 333，333*3=999。数组中记录的是数字出现的次数，也就是每出现一次就+1，如 s[i/100]+=1; 然后后面就要判断数组中 1~9 每个元素的数字是否都是 1。在最后判断的地方，只需要把数组 1~9 元素的值相加，如果相加结果是 9 则说明每个数字都只出现一次，毕竟如果有一个数字出现两次的话相加不可能为 9 的。

【参考程序】

```
#include<iostream>
using namespace std;
int main(){
    int s[10];//定义一个数组记录 0~9 出现的次数
    for(int i=123;i<333;i++){
        for(int m=0;m<10;m++)
            s[m]=0;
        s[i/100]=1;
        s[i/10%10]=1;
        s[i%10]=1;
        int j=i*2;
        s[j/100]=1;
        s[j/10%10]=1;
        s[j%10]=1;
        int k=i*3;
        s[k/100]=1;
        s[k/10%10]=1;
        s[k%10]=1;
        int sum=0;
        for(int m=1;m<10;m++)
            sum+=s[m];
        //将 1~9 每个数字出现次数加起来，如果等于 9 则每个数字都出现了一次
        if(sum==9) cout<<i<<" "<<j<<" "<<k<<endl;
    }
    return 0;
}
```

练习 4：最大公约数

【解析】

使用辗转相除法

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
int GCD(int a,int b)
{
    if(a%b==0)
       return b;
    GCD(b,a%b);
}
int main()
{
    int m,n;
    cin>>m>>n;
    cout<<GCD(m,n);
}
```

练习 5：带分数

【解析】

递推枚举，先从 a 开始枚举 ，然后 c 进行枚举，根据式子 n = a + b / c，求出 b = n * c - a * c （爆 int ， 开 long long） ，然后判断 a ， b ， c 中数字是否包含 1 — 9 ，且只出现一次。

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10;
int n;
int result ;
bool used[N];
bool sta[N];
bool check(int a, int c){
    long long b = n *(long long) c - a * c;
    if (!a || !b || !c) return false; // 当 a,b,c 中存在 0 时，返回 false
    memcpy(sta,used,sizeof(used)); // 讲 used 的值赋给 sta  防止影响后面的进行
    while(b){
        int x = b % 10;
        b /= 10;
        if (!x || sta[x]) return false;
        sta[x] = true;
    }
    for (int i = 1; i < 10; i++ ){
        if (!sta[i]) return false;
    }
    return true;
```

```
}
void dfs_c(int a, int c){
    if (check(a,c)) result ++;  //如果符合条件 那么 ++
    for (int  i = 1; i <= 9; i++){
        if (!used[i]){
            used[i] = true;
            dfs_c(a,c*10+i);  // c 进行增加
            used[i] = false;  // 恢复现场
        }

    }
}
void dfs_a(int a){
    if (a >= n) return;
    if (a) dfs_c (a,0);
    for (int i = 1; i <= 9;i++){
        if(!used[i]){
            used[i] = true;
            dfs_a(a*10+i); // a 进行增加
            used[i] = false; // 恢复现场
        }

    }
}
int main()
{
    cin>>n;
    dfs_a(0);
    cout<< result <<endl;
    return 0;
}
```

练习 6：八皇后问题

【解析】

只用一个有 8 个元素的数组记录已经摆放的棋子摆在什么位置，当要放置一个新的棋子时，只需要判断它与已经放置的棋子之间是否冲突就行了。

【参考程序】

```
#include <stdio.h>
int ans[92][8], n, b, i, j, num, hang[8];
void queen(int i){
   int j, k;
   if(i == 8){ //一组新的解产生了
      for(j = 0; j < 8; j++)  ans[num][j] = hang[j] + 1;
      num++;
```

```
            return;
        }
        for (j=0; j<8; j++){ //将当前皇后 i 逐一尝试放置在不同的列
             for(k=0; k<i; k++) //逐一判定 i 与前面的皇后是否冲突
                 if( hang[k] == j || (k - i) == (hang[k] - j) || (i - k)
== (hang[k] - j )) break;
            if (k == i) {  //放置 i，尝试第 i+1 个皇后
                hang[i] = j;
                queen(i + 1);
            }
        }
    }
    int main()
    {
        num=0;
        queen(0);
        scanf("%d", &n);
        for(i = 0; i < n; i++){
            scanf("%d", &b);
            for(j = 0; j < 8; j++)  printf("%d", ans[b - 1][j]);
            printf("\n");
        }
        return 0;
    }
```

第 5 章 贪心

练习 1：删数

【解题思路】
从前往后进行遍历，每当找到一个数比后面的数大时就删除该数，再从前一位开始继续遍历，（前一位的目的是防止出现如 582 这样的情况，8>2，8 删除后变为 52，5>2，如果不从前一位进行遍历则会漏掉这种情况），因为想要保证数字尽可能的小，优先要让高位的数字变小，经过如上的遍历后，高位数字能够在题目条件下保证数字尽可能的小，如若所有数字都从小到大排列，才会删除末尾最大的个位数。详细代码及注释如下：

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<algorithm>
#define N 20
using namespace std;
int main()
{
    int t;
    char str[260];
    int i,j;
    int k=0;

    cin>>str;
    cin>>t;
    int len=strlen(str);
    while(t--)
    {

        for(i=k;i<len;i++)
            if(str[i]>str[i+1]) //每次将一个递减的序列中第一个数字删除
            {
              k=i-1;  //每次定位到所删数的前面一位开始
                for(j=i;j<len;j++)
                    str[j]=str[j+1];
                break;   //删除 1 个数后，跳出内循环，即继续寻找下一个数
            }
        len--;
      //当有递减序列，删除了一个数，长度减 1，从该位置继续进行下一
```

次循环，若为全递增序列，则刚好去掉尾部最大的个位数

```
        }
        i=0;
        while(i<=len-1&&str[i]=='0')
            i++;
        if(i==len)
            cout<<"0"<<endl;  //特殊情况，全删，为0
        else
            for(j=i;j<=len-1;j++)
                cout<<str[j];
        return 0;
    }
```

练习二：翻硬币

【解题思路】

N个硬币一共有N-1种不同的翻转情况，而第一个和最后一个硬币想要改变状态只有一种情况，其他有两种，第一个硬币翻转后，其状态已经固定，可以无视，这时第二个硬币则可以当作第一个硬币来看待（递推的思想）所以本题则只需要遍历模拟，从前往后对初始状态和目标状态进行对比，遇到不同时计数器+1，再改变后面一个硬币的状态即可（后续遍历本硬币的状态对求出结果没有影响）。

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
int main()
{
    string c1,c2;
    cin >> c1;
    cin >> c2;
    int cnt=0;
    for(int i = 0;i < c1.size() - 1 ;i++)
    {
       if(c1[i] != c2[i])
       {
          cnt++;
           if(c1[i+1] == 'o') //改变后一个硬币的状态
               c1[i+1] ='*';
           else c1[i+1] ='o';
       }

   }
```

```
    cout <<cnt<<endl;
    return 0;

}
```

练习三：分糖果

【解题思路】

最终每个小朋友的糖果数量可以计算出来，等于糖果总数除以 n。

求出数组的前缀和,根据下面的公式可以求出最小值。

数组 s[i]表示第 i 个小朋友传递给第 i+1 个小朋友的糖果数量

那么第 i 个小朋友的糖果数经过传递操作后最终的数量应该是 a[i] - s[i] + s[i - 1]

题目要求最终所有小朋友手中糖果数相同，并且保证有解。

综上所述，我们获得如下方程组:

$\bar{a}=(a1+a2+a3+\cdots+an)/n$

$\bar{a}=a1+sn-s1$

$\bar{a}=a2+s1-s2$

$\bar{a}=a3+s2-s3$

.....

$\bar{a}=an+sn-1-sn$

我们对 22 中的方程组进行变形，即可获得如下形式：

$S1=Sn-(\bar{a}-a1)$

$S2=S1-(\bar{a}-a2)=Sn-(\bar{a}-a1)-(\bar{a}-a2)=Sn-(2\bar{a}-(a1+a2))$

$S3=S2-(\bar{a}-a3)=Sn-(3\bar{a}-(a1+a2+a3))$

$S4=S3-(\bar{a}-a4)=Sn-(4\bar{a}-(a1+a2+a3+a4))$

.....

$Sn-1=Sn-1-((n-1)\ \bar{a}-(a1+a2+\cdots+an-1))$

$Sn=Sn$

而题目让我们求解的的等式是 $|s1|+|s2|+|s3|+\cdots+|sn|$

经过上述的方程组，我们可以把该式子转换成绝对值不等式

$res=|Sn-(a^{\frown\frown}-a1)|+|Sn-(2a^{\frown\frown}-(a1+a2))|+|Sn-(4a^{\frown}(a1+a2+a3+a4))|+\cdots+|Sn|$

求得以上式子，最后只要寻找中位数就可以了，主要是利用了数学+贪心的性质进行求解

【参考代码】

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 1e6 + 10;

int n;
LL s[N], c[N];

int main() {
    scanf("%d", &n);
    //求出前缀和，方便计算后面 c[i] = i * A - (a[1] + a[2] + ... + a[i])
中的区间部分
    for (int i = 1; i <= n; ++i) scanf("%lld", &s[i]), s[i] += s[i - 1];
    LL A = s[n] / n; //A 就是平均数 a
    for (int i = 1, k = 0; i < n; ++i, ++k) c[k] = i * A - s[i];
    c[n - 1] = 0; //对应的是|s_n - 0|
    nth_element(c, c + n / 2, c + n);   //找中位数
    LL res = 0;
    for (int i = 0; i < n; ++i) res += abs(c[i] - c[n / 2]);
    printf("%lld\n", res);
    return 0;
}
```

练习四：推销员

【解题思路】

【输入输出样例 1 说明】

X=1:向住户 5 推销，往返走路的疲劳值为 5+5，推销的疲劳值为 5，总疲劳值为 15。

X=2:向住户 4、5 推销，往返走路的疲劳值为 5+5，推销的疲劳值为 4+5，总疲劳值为 5+5+4+5=19。

X=3:向住户 3、4、5 推销，往返走路的疲劳值为 5+5，推销的疲劳值 3+4+5，总疲劳值为 5+5+3+4+5=22。

X=4:向住户 2、3、4、5 推销，往返走路的疲劳值为 5+5，推销的疲劳值 2+3+4+5，总疲劳值 5+5+2+3+4+5=24。

X=5:向住户 1、2、3、4、5 推销，往返走路的疲劳值为 5+5，推销的疲劳值 1+2+3+4+5，总疲劳值 5+5+1+2+3+4+5=25。

【输入输出样例 2 说明】

X=1：向住户 4 推销，往返走路的疲劳值为 4+4，推销的疲劳值为 4，总疲劳值 4+4+4=12。
X=2：向住户 1、4 推销，往返走路的疲劳值为 4+4，推销的疲劳值为 5+4，总疲劳值 4+4+5+4=17。
X=3：向住户 1、2、4 推销，往返走路的疲劳值为 4+4，推销的疲劳值为 5+4+4，总疲劳值 4+4+5+4+4=21。
X=4：向住户 1、2、3、4 推销，往返走路的疲劳值为 4+4，推销的疲劳值为 5+4+3+4，总疲劳值 4+4+5+4+3+4=24。或者向住户 1、2、4、5 推销，往返走路的疲劳值为 5+5，推销的疲劳值为 5+4+4+1，总疲劳值 5+5+5+4+4+1=24。
X=5：向住户 1、2、3、4、5 推销，往返走路的疲劳值为 5+5，推销的疲劳值为 5+4+3+4+1，
总疲劳值 5+5+5+4+3+4+1=27。

本题主要用到了前缀和以及贪心的思想，利用前缀和进行预处理存储到各家的距离和，主要是发现如下的性质：向 x 家住户推销产品的最大花费只有两种情况：推销给 Ai 最大的 x 家，或者推销给 Ai 最大的 x-1 家，然后最后一家尽可能的远。

【参考代码】

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

typedef pair<int, int> PII;

const int N = 100010;

int n;
PII w[N];
int f[N];   // 表示前 i 个 a 的和
int g[N];   // 表示前 i 个 s 的最大值
int h[N];   // 表示 i~n 中 2s + a 的最大值

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i ++ ) scanf("%d", &w[i].second);
    for (int i = 1; i <= n; i ++ ) scanf("%d", &w[i].first);
    sort(w + 1, w + n + 1);
    reverse(w + 1, w + n + 1);
    for (int i = 1; i <= n; i ++ ) f[i] = f[i - 1] + w[i].first;
    for (int i = 1; i <= n; i ++ ) g[i] = max(g[i - 1], w[i].second);
```

```
        for (int i = n; i; i -- ) h[i] = max(h[i + 1], 2 * w[i].second
+ w[i].first);
        for (int i = 1; i <= n; i ++ ) printf("%d\n", max(f[i] + 2 * g[i],
f[i - 1] + h[i]));
        return 0;
    }
```

练习五：排座位

【解题思路】

本题用到的是贪心和排序的算法，不同行列独立进行计算，不同行、列之间是完全独立的。即不管将哪行、哪列切开，对其余的行列都是没有任何影响的。因此可以分别考虑行和列。

对于行来说，问题变成：

去掉哪 K 行，可以使得最后剩下的行间的边数最少。这里去掉边数最多的 K 行一定是最优的。否则可以将选出的行替换成边数最多的 K 行，且结果不会变差。

【参考代码】

```
    #include<cstdio>
    #include<algorithm>
    using namespace std;
    struct zj{
        int sum,num;
    }x[1000],y[1000];
    bool cmp1(const zj &x,const zj &y){return x.sum>y.sum;}
    bool cmp2(const zj &x,const zj &y){return x.num<y.num;}
    int n,m,k,l,d;
    int x1,y1,x2,y2;
    int main(){
        scanf("%d%d%d%d%d",&m,&n,&k,&l,&d);
        for(int i=1;i<=d;i++){
            scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

if(x1==x2)y[min(y1,y2)].num=min(y1,y2),y[min(y1,y2)].sum++;

if(y1==y2)x[min(x1,x2)].num=min(x1,x2),x[min(x1,x2)].sum++;
        }
        sort(y+1,y+n+1,cmp1);
```

```
    sort(x+1,x+m+1,cmp1);
    sort(y+1,y+l+1,cmp2);
    sort(x+1,x+k+1,cmp2);
    for(int i=1;i<=k;i++)printf("%d ",x[i].num);
    printf("\n");
    for(int i=1;i<=l;i++)printf("%d ",y[i].num);
    return 0;
}
```

练习六：分纸牌

【解题思路】

本题限制了条件，纸牌总数必为纸牌堆数 N 的倍数，所以每一堆的纸牌最终数量可以直接求出，向后遍历时，每遍历一个就会将该堆的纸牌数量补齐，然后再对补齐所需的卡牌数（可负）从后一堆中取得即可。

【参考代码】

```
#include <iostream>
using namespace std;
const int N = 110;
int a[N];
int n , total;
int ans;
int main()
{
    cin >> n;
    for(int i = 1 ; i <= n ; i++) cin >> a[i] , total += a[i];
    int avg = total / n;
    for(int i = 1 ; i <= n ; i++)
        if(a[i] != avg) a[i + 1] += a[i] - avg , ans++;
    cout << ans << endl;
    return 0;
}
```

第 6 章 搜索

练习 1：凑算式

【解析】

通过暴力生成 9 的全排列，然后用数组存放字母对应生成的数字，判断等式是否成立。

【参考程序】

```
#include<iostream>
using namespace std;
int ans=0;
bool vis[10];
int num[9];
void solve(){

if((num[0]+(double)num[1]/num[2]+(double)(num[3]*100+num[4]*10+num[5])
/(num[6]*100+num[7]*10+num[8]))==10)
        ans++;
    return;
}
void dfs(int d){
    if(d==9){
        solve();
        return;
    }
    for(int i=1;i<10;i++){
        if(!vis[i]){
            vis[i]=true;
            num[d]=i;
            dfs(d+1);
            vis[i]=false;
        }
    }
}
int main(){
    dfs(0);
    cout<<ans<<endl;
return 0;
}
```

练习 2：玩具蛇

【解析】这一题属于典型的 DFS 算法题。

1. 选出玩具蛇第一节放置的位置，很明显 4x4 的格子都可以。就是说 16 个格子任意一个都可以放玩具蛇的第一节。
2.从玩具蛇第一节出发，利用 dfs 算法，进行深搜。
3.条件为相邻两节成直线为下一节放置的方向，只能为相邻的上下左右四个格子，其中已放过的格子，以及越界时不放置。

【参考程序】

```
#include<bits/stdc++.h>
using namespace std;
bool vis[10][10];
int ans;
int dx[]={-1,0,1,0},dy[]={0,1,0,-1};
void dfs(int x,int y,int cnt){
    if(cnt==16){
        ans++;
        return;
    }
    for(int i=0;i<4;i++){
        int a=x+dx[i],b=y+dy[i];
        if(a>=1&&a<=4&&b>=1&&b<=4&&!vis[a][b]){
            vis[a][b]=true;
            dfs(a,b,cnt+1);
            vis[a][b]=false;
        }
    }
}
int main()
{
    for (int i = 1; i <= 4; i++){
     for (int j = 1; j <= 4; j++){
        vis[i][j]=true;//注意初始化
         dfs(i,j,1);
         vis[i][j]=false;
     }
 }
    cout << ans;
    return 0;
}
```

练习 3：递归入门

【解析】

这一题用深搜遍历。

利用深度优先搜索递归的思路来遍历，在 n 个数里选 k 个，n 个数存放在一个数组里，递归所有可能的情况，然后判断是否为素数，这就是本题核心。

因此递归函数 DFS 有三个参数，index 是第 Index 个数，nowK 是当前选择的个数，sum 是当前已选数字之和。

【参考程序】

```
#include <cstdio>
int n,k,p[22];
int count ;
bool isprime(int c)
{
    if(c<2)
        return false;
    for(int i=2;i*i<=c;++i)
    {
        if(c%i==0)
            return false;
    }
    return true;
 }
void Dfs(int index,int nowk,int sum)
{
    if(nowk==k)
    {
        if(isprime(sum)) //如果为素数，统计量+1
            count++;
        return;
    }
    if(index==n||nowk>k)//如果处理完 n 个数，或者超过 k 个数，返回
        return;
    Dfs(index+1,nowk+1,sum+p[index]);//选择当前数
    Dfs(index+1,nowk,sum); //不选当前数
}
int main()
{
    count=0;
    scanf("%d%d",&n,&k);
    for(int i=0;i<n;++i)
    {
        scanf("%d",&p[i]);
    }
    Dfs(0,0,0);
    printf("%d\n",count);
    return 0;
}
```

练习 4：找朋友

【解析】

1、使用广度优先搜索

2、使用结构体数组来保存路径、消耗的时间

【参考程序】

```
#include<bits/stdc++.h>
using namespace std;
char tu[16][16];
int book[16][16];
int n,m;
int jx[] = {0,-1,0,1};
int jy[] ={1,0,-1,0};
struct node
{
    int x;
    int y;
    int ans;
}q[300],t,f;
void bfs(int x,int y)
{
    int i;
    queue<node>g;
    t.x = x;
    t.y =y;
    t.ans = 0;
    g.push(t);
    book[x][y] = 1;
    while(!g.empty())
    {
        f = g.front();
        g.pop();
        if(tu[f.x][f.y]=='Y')
        {
        printf("%d\n",f.ans);
        return ;
        }
        for(i =0;i<4;++i)
        {
           t.x = f.x+jx[i];
           t.y = f.y+jy[i];

if(t.x>=0&&t.x<n&&t.y>=0&&t.y<m&&tu[t.x][t.y]!='#'&&book[t.x][t.y]==0
)
           {
```

```
            t.ans = f.ans+1;
            g.push(t);
            book[t.x][t.y] = 1;
            }
        }
    }
    printf("-1\n");
    return ;

}
int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        memset(tu,0,sizeof(tu));
        memset(book,0,sizeof(book));
        for(int i=0;i<n;i++)
        {
            scanf("%*c%s",tu[i]);
        }
        int ff=0;
        int i,j;
        for( i=0;i<n;i++)
        {
        for( j=0;j<m;j++)
        {
            if(tu[i][j]=='X')
            {
            ff=1;
            break;
            }
        }
        if(ff) break;
        }
        bfs(i,j);
    }
    return 0;
}
```

练习 5：马的遍历

【解析】

1、使用广度优先搜索。
2、注意：马走“日”，走的方向有八个
【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
queue<pair<int,int> > q;
int step[500][500];
bool visited[500][500];
const int x_gain[] ={-1,-2,-2,-1,1,2,2,1};
const int y_gain[] ={2,1,-1,-2,-2,-1,1,2};
int main(){
   memset(step,-1,sizeof(step));
   memset(visited,false,sizeof(visited));
   int n,m,x,y;
   cin>>n>>m>>x>>y;
   q.push(make_pair(x,y));
   step[x][y]=0;
   visited[x][y]=true;
   while(!q.empty()){
       int x_now=q.front().first;
       int y_now=q.front().second;
       q.pop();
       for(int i=0;i<8;i++){
           int xx=x_now+x_gain[i];
           int yy=y_now+y_gain[i];
           if(xx<1||xx>n||yy<1||yy>m||visited[xx][yy]) continue;
           visited[xx][yy]=true;
           step[xx][yy]=step[x_now][y_now]+1;
           q.push(make_pair(xx,yy));
       }
   }
   for(int i=1;i<=n;i++){
       for(int j=1;j<=m;j++){
           printf("%d ",step[i][j]);
       }
       cout<<endl;
   }
   return 0;
}
```

练习 6：大臣的旅费

【解析】

1. 任选一个起始节点开始 dfs，直到到达距离它最远的节点 maxu
2.从 maxu 再次开始 dfs 或 bfs，直到到达距离 maxu 最远的距离 p
3.maxu 到 p 的路径就是树的直径
【参考程序】

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6 + 10;
int n ;
int w[N] , e[N] , ne[N] , h[N] , idx ;
int maxu , maxd ;
void add(int a , int b , int c )
{
    e[idx] = b , w[idx] = c , ne[idx] = h[a] , h[a] = idx ++ ;
}
void dfs(int u , int fa , int d)
{
    for(int i = h[u] ; i != -1 ; i = ne[i])
    {
        int j = e[i] ;
        int k = w[i] ;
        if(j == fa) continue ;
        if(maxd < d + k)
        {
            maxd = d + k ;
            maxu = j ;
        }
        dfs(j,u,d + k);
    }
}
int main()
{
    cin >> n ;
    memset(h,-1,sizeof h);
    int k = n - 1 ;
    while(k--)
    {
        int a , b , c ;
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
        add(b,a,c);
    }
    dfs(1,-1,0);
    dfs(maxu,-1,0);
    cout << maxd * 10 + (maxd + 111) * maxd / 2 << endl ;
```

```
    return 0;
}
```

第 7 章 动态规划

练习 1：爬楼梯

【解析】

因为每次只能走 1，2，3 阶，所以，当一共 5 级台阶时，可以看成在走了 4 阶基础上再跨 1 阶 + 走了 3 阶基础上再跨 2 阶 +走了 2 阶基础上再跨 3 阶。

```
#include<iostream>
using namespace std;
typedef long long ll;
ll f[61];
int main()
{
   f[1]=1;
   f[2]=2;
   f[3]=4;
   for(int i=4;i<=60;i++)
      f[i]=f[i-1]+f[i-2]+f[i-3];
   cout<<f[60];
}
```

练习 2：数字三角形

【解析】

从上层往底下 dp 需要考虑边界的问题，从底层网上 dp 会容易很多。

```
#include<iostream>
using namespace std;

const int N=110;
int f[N][N];
int n;

int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=i;j++)
            cin>>f[i][j];

    for(int i=n;i>=1;i--)
        for(int j=i;j>=1;j--)
            f[i][j]=max(f[i+1][j],f[i+1][j+1])+f[i][j];
```

```
    cout<<f[1][1]<<endl;
}
```

练习 3：最长公共子序列

【解析】

对于最长公共子序列，我们可分成两种情况。如果当 s1[i]和 s2[j]相等，及匹配上时，我们只需要在 f[i-1][j-1]的情况下(去除 s1[i]，s2[j]本身)+1 即可。如果不相等，则需要到 f[i - 1][j](去除 s1[i])或者 f[i][j - 1]（去除 s2[j]）中找一个最大值。

```
#include <iostream>
#include <string.h>
using namespace std;

string s1,s2;
int f[1001][1001];

int main()
{
   int T;
   cin>>T;
   while(T--)
   {
      cin>>s1>>s2;
      for (int i = 1; i <= s1.length(); i++)
         for (int j = 1; j <= s2.length(); j++)
         {
            if (s1[i] == s2[j])
               f[i][j] = f[i - 1][j - 1] + 1;
            else
               f[i][j] = max(f[i - 1][j], f[i][j - 1]);
         }
      cout << f[s1.length()][s2.length()] <<endl;
      memset(f,0,sizeof(f));
   }
   return 0;
}
```

练习四：方格取数

【解析】

若只走一次，那么要么从上往下，要么从左往右，

sum[i][j]=max(sum[i-1][j],sum[i][j-1)+a[i][j];

若走两次，我们可以理解为两个人同时往下走，那么可以理解为一个人从上往下，另一个人从上往下，等等，分别对应的是上上，上左，左上，左左。

sum[i][j][h][k]=max{sum[i-1][j][h][k-1],sum[i][j-1][h-1][k],sum[i

-1][j][h-1][k],sum[i][j-1][h][k-1]}+a[i][j]+a[h][k]

走过的地方数就被取走了,所以注意不能重复加数,也就是要特殊考虑一下,i == h&& j == k 的情况，只能加一个 a[i][j];

```
#include<iostream>
#include<algorithm>
using namespace std;
int n;
int a[11][11];
int sum[11][11][11][11];
int main() {
   int n;
   cin >> n;
   int x, y, z;
   while (cin >> x >> y >> z) {
      if (x == 0 && y == 0 && z == 0) break;
      a[x][y] = z;
   }
   for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++)
         for (int h = 1; h <= n; h++)
            for (int k = 1; k <= n; k++)
            {
               if (i == h && j == k)
                  sum[i][j][h][k] = max(max(sum[i - 1][j][h][k - 1], sum[i][j - 1][h - 1][k]), max(sum[i - 1][j][h - 1][k], sum[i][j - 1][h][k - 1])) + a[i][j];
               else sum[i][j][h][k] = max(max(sum[i - 1][j][h][k - 1], sum[i][j - 1][h - 1][k]), max(sum[i - 1][j][h - 1][k], sum[i][j - 1][h][k - 1])) + a[i][j] + a[h][k];

            }
   }
   cout << sum[n][n][n][n] << endl;
   return 0;
}
```

练习五：排课

【分析】

在每个步骤我们都要选一门课程。假装我们已经拿到了最优决策方案，我现在就想知道最优解中最后一个决策到底是什么？也就是 An 你是选了还是不选？

这里插一句非常关键话：我们一开始把课程按结束时间排好序。

现在我们就可以写出递归表达式了：

$$OPT(n)=max\{\ \ OPT(pre(i))+Wi\ \ \ ,\ OPT(i-1)\}$$

其中 pre(i)是表示第 i 门课程开始时已经结束的课程。前者代表选了，后者代表没选

```
#include <iostream>
#include<algorithm>
using namespace std;
int N;
class course
{
public:
   float start;
   float end;
   int w;
}Course[5010];
int result;
int pre(int n) {//求课程 n 开始时结束的所有课程
   float current_start;
   int result = 0, i;
   current_start = Course[n].start;
   for (i = 0; i < N; i++) {
      if (Course[i].end > current_start) {
         result = i - 1;
         break;
      }
   }
   return result;
}

int  Scheduling_DP(int n) {
   int m = 0;
   if (n < 0)
      return 0;
   else if (n == 0) {
      return Course[0].w;
   }
   else {
      m = max((Scheduling_DP(pre(n)) + Course[n].w),
```

```
Scheduling_DP(n - 1));
            return m;
        }
    }

    int main()
    {
        scanf_s("%d", &N);
        for (int i = 0; i < N; i++)
        {
            int sta, stb, ena, enb, num;
            scanf_s("%d:%d%d:%d%d", &sta, &stb, &ena, &enb, &num);
            Course[i].start = sta + double(stb) / 60;
            Course[i].end = ena + double(enb) / 60;
            Course[i].w = num;
        }

        result = Scheduling_DP(N - 1);
        cout << result << endl;
}
```

练习六：滑雪

【解题报告】

典型的动态规划题目，采用记忆化搜索，利用一个数组保存每个点的最大值，(动态规划的优点，避免重复计算子问题) 对每个点 进行上下左右 的求解，该点 的最大值 肯定是 从 四个方向 中最大 的 +1。 按照这个思想，不然求解。

```
#include <stdio.h>
#include <math.h>
#define MAX_NUM 101

int map_arr[MAX_NUM][MAX_NUM]; //保存各个点高度
int dp_arr[MAX_NUM][MAX_NUM];  //保存各个点的最大值

int dp_x[] = {-1, 0, 1, 0}; //四个方向
int dp_y[] = {0, -1, 0, 1};
int row, column;
//返回 2 个数中的最大值
int find_max( int num1, int num2 )
{
    if (num1 > num2)
        return num1;
    else
        return num2;
```

```
    }
    int search_max( int x, int y )
    {
        int i;
        int vx_, vy_;
        if (dp_arr[x][y])  //如果不是0，即已经找到该点的最大值，直接返回，避免重复计算
            return dp_arr[x][y];
        for (i=0; i<4; i++)   //找出四个方向中满足条件的最大值。
        {
            vx_ = x + dp_x[i];    //四周坐标x就为j为0的值，纵坐标y就为j为1的值，所以y不用j变量
            vy_ = y + dp_y[i];
            if (map_arr[vx_][vy_] != -1 && vx_>=0 && vx_<row && vy_>=0 && vx_<column && map_arr[vx_][vy_] < map_arr[x][y])
            {
                dp_arr[x][y] = find_max(dp_arr[x][y], search_max(vx_, vy_)+1);
            }
        }

        return dp_arr[x][y];

    }

    int main()
    {

        int i, j;
        int max_;

        while (scanf("%d %d", &row, &column) == 2)  //取得行列数
        {
            for (i=0; i<MAX_NUM; i++ ) //初始化各个点高度,默认-1，表示不存在。
            {
                for (j=0; j<MAX_NUM; j++)
                {
                    map_arr[i][j] = -1;

                }
            }

            for (i=1; i<=row; i++)
```

```
            {
                for (j=1; j<=column; j++)
                {
                    scanf("%d", &map_arr[i][j]); //输入各个点的高度
                    dp_arr[i][j] = 0;
                }
            }

            max_ = 0;

            for (i=1; i<=row; i++)
            {
                for (j=1; j<=column; j++)
                {
                    max_ = find_max(max_, search_max(i, j)); //找出最
大值
                }
            }

            printf("%d\n", max_ + 1 ); //＋1.  因为最后一个点没被算进
去。
        }

        return 0;
    }
```

第 8 章 其他算法

练习 1：二叉树中节点的深度

【解析】

本题考查对数据结构二叉排序树的掌握，通过数据构建二叉排序树，并遍历寻找目标节点返回所在深度。

【代码】

```
#include<iostream>
using namespace std;
#include<stdlib.h>
int depth;

typedef struct node
{
    int key;
    struct node *lchild, *rchild;

}BSTNode, *BSTree;

void InsertBST(BSTree *bst, int key)
{
    BSTree s;
    if (*bst == NULL)
    {
        s = (BSTree)malloc(sizeof(BSTNode));
        s->key = key;
        s->lchild = NULL;
        s->rchild = NULL;
        *bst = s;
    }
    else if (key<(*bst)->key)  InsertBST(&((*bst)->lchild), key);
    else if (key > (*bst)->key)          InsertBST(&((*bst)->rchild), key);
}

void CreateBST(BSTree *bst)
{
    int n;
    *bst = NULL;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        int key;
        cin>>key;
```

```
            InsertBST(bst,key);
        }

}

void inorder(BSTree bt)
{
    if (bt != NULL)
    {
        inorder(bt->lchild);
        printf("%d ",bt->key);
        inorder(bt->rchild);
    }
}

int FindTreeDeep(BSTree BT)
{
      int deep=0;
      if(BT)
      {
          int lchilddeep=FindTreeDeep(BT->lchild);
          int rchilddeep=FindTreeDeep(BT->rchild);
          deep=lchilddeep>=rchilddeep?lchilddeep+1:rchilddeep+1;
      }
      return deep;
}
int GetDepth(BSTree BT)
{
        int rHeight,lHeight;
        if(BT)
    {

        lHeight=GetDepth(BT->lchild);
        rHeight=GetDepth(BT->rchild);
        return (lHeight>rHeight? lHeight:rHeight)+1;
    }
    else
    {
        return 0;
    }

}

int GetSubDepth(BSTree BT,int x)
```

```
{

    if(BT)
    {

        if(BT->key==x)
        {
            cout<<depth-GetDepth(BT)+1;
            exit(0);
        }
        else
        {
            if(BT->lchild)    GetSubDepth(BT->lchild,x);
            if(BT->rchild)    GetSubDepth(BT->rchild,x);
        }
    }
    else
    {
        return 0;
    }
}


int main()
{
    BSTree bt;
    CreateBST(&bt);
    inorder(bt);
    printf("\n");
    depth=FindTreeDeep(bt);
    int n;
    cin>>n;
    printf("%d\n",GetSubDepth(bt,n));
    return 0;
}
```

练习 2：合根植物

【解析】

该题是典型的并查集问题，只需要将输入的数据先进行并操做，最后再进行查操作。

【代码】

```
#include<iostream>
#include<cstring>
using namespace std;
```

```
const int inf = 0x3f3f3f3f;
const int maxn = 10051005;
int m,n,k,pre[maxn],ans;
bool vis[maxn];

void init()
{
	memset(vis,0,sizeof(vis));
	for(int i=1;i<=n*m;i++)
		pre[i] = i;
}

int find(int x)
{
	if(x!=pre[x])
		return pre[x] = find(pre[x]);
	return x;
}

void join(int x,int y)
{
	int xx = find(x);
	int yy = find(y);
	if(xx!=yy)
	{
		pre[yy] = xx;
	}
}

int main()
{
	scanf("%d%d%d",&n,&m,&k);
	init();
	for(int i=0;i<k;i++)
	{
		int x,y;
		scanf("%d%d",&x,&y);
		join(x,y);
	}
	for(int i=1;i<=n*m;i++)
	{
		vis[find(i)] = 1;
	}
```

```
    for(int i=1;i<=n*m;i++)
        ans += vis[i];
    cout<<ans<<endl;
    return 0;
}
```

练习 3：最短路

【解析】

由题干可知本题让我们有向图路上的最短路径，这里我们用典型的 Floyd 算法求得最短路径。

【代码】

```
#include<iostream>
using namespace std;
#include<algorithm>
int a[20000][20000];
int main(){
    int n,m;
    cin>>n>>m;
    int u.v,l;
    for(int i=1;i<=m;i++)
    {
        cin>>u>>v>>l;
        a[u][v]=l;
    }
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            for(int k=1;k<=n;k++)
            {
                if(a[i][j]&&a[j][k]) a[i][k]=min(a[i][k],a[i][j]+a[j][k]);
            }
    for(int i=2;i<=n;i++)
        cout<<a[1][i]<<endl;
    return 0;
}
```

练习 4：生日礼物

【解析】

其实就是维护序列的左右端点。先将所有的珠子按照位置排序，然后考虑每次让左端点后移，右端点也后移直到恰好有 k 种颜色为止，更新答案。

【代码】

```
#include<iostream>
#include<algorithm>
#define N 200000
```

```
using namespace std;

int n,m,vis[63],size,cnt;
struct data {
	int pos,x;
}a[N];

int cmp(data a,data b)
{
	return a.pos<b.pos;
}

void change(int x,int v)
{
	if (v==1&&!vis[a[x].x]) size++;
	if (v==-1&&vis[a[x].x]==1) size--;
	vis[a[x].x]+=v;
}
int main()
{
	scanf("%d%d",&n,&m);
	for (int i=1;i<=m;i++){
		int c; scanf("%d",&c);
		for (int j=1;j<=c;j++){
			int x; scanf("%d",&x);
			a[++cnt].pos=x; a[cnt].x=i;
		}
	}
	sort(a+1,a+cnt+1,cmp);
	int l=1; int r=1; change(1,1);
	int ans=a[cnt].pos;
	while (l<=r){
		while (size<m&&r<cnt) change(++r,1);
		if (size==m)	ans=min(ans,a[r].pos-a[l].pos);
		change(l++,-1);
	}
	printf("%d\n",ans);
}
```