给定 n 个字符串，请对 n 个字符串按照字典序排列。

```
//因为set去重，所以不能用set;
#include<iostream>
#include<map>
using namespace std;
int main(){
    int num;
    map<string,int>arr;
    while(cin>>num){
        string str;
        while(num--){
            cin>>str;
            ++arr[str];
        }
        for(map<string,int>::iterator it=arr.begin();it!=arr.end();it++)
            for(int i=0;i<it->second;i++)
                cout<<it->first<<endl;
    }
    return 0;
}
```

写出一个程序，接受一个十六进制的数，输出该数值的十进制表示。（多组同时输入 ）

```
//输入输出流的都是字符串
#include<iostream>
using namespace std;
int main(){
    int a;
    while(cin>>hex>>a)
        cout<<a<<endl;
}
```

功能:输入一个正整数，按照从小到大的顺序输出它的所有质因子（重复的也要列举）（如180 的质因子为 2　2　3　3　5　）

最后一个数后面也要有空格

```
#include<iostream>
using namespace std;
int main(){
    long num;
    while(cin>>num){
        int i=2;
        while(num>1){
            while(num%i==0){
                num/=i;
                cout<<i<<" ";
            }
            ++i;
        }
        cout<<endl;
    }
    return 0;
}
```

数据表记录包含表索引和数值（int 范围的整数），请对表索引相同的记录进行合并，即将相同索引的数值进行求和运算，输出按照 key 值升序进行输出

```
#include<iostream>
#include<map>
using namespace std;
int main(){
    int n;
    int num1,num2;
    while(cin>>n){
        map<int,int>arr;
        while(n--){
            cin>>num1>>num2;
            arr[num1]+=num2;
        }
        for(map<int,int>::iterator it=arr.begin();it!=arr.end();it++){
            cout<<it->first<<" "<<it->second<<endl;
        }
    }
```

输入一个 int 型整数，按照从右向左的阅读顺序，返回一个不含重复数字的新的整数

```cpp
#include<iostream>
#include<vector>
using namespace std;
int main(){
    int num;
    int arr[9]={0};
    vector<int>ans;
    while(cin>>num){
        while(num>0){
            int mod=num%10;
            num/=10;
            if(arr[mod]!=1){
                ans.push_back(mod);
                arr[mod]=1;
            }
        }
        for(auto mod:ans){
            cout<<mod;
        }
        cout<<endl;

    }
}
```

Lily 上课时使用字母数字图片教小朋友们学习英语单词，每次都需要把这些图片按照大小（ASCII 码值从小到大）排列收好。请大家给 Lily 帮忙，通过 C 语言解决。

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
bool compare(const char&ch1,const char&ch2){
    return (ch1-'0')<(ch2-'0');
}
int main(){
    string str;
    while(cin>>str){
        sort(str.begin(),str.end(),compare);
        cout<<str<<endl;
    }
    return 0;
}
```

编写一个截取字符串的函数，输入为一个字符串和字节数，输出为按字节截取的字符串。但是要保证汉字不被截半个，如"我 ABC"4，应该截为"我 AB"，输入"我 ABC 汉 DEF"6，应该输出为"我 ABC"而不是"我 ABC+汉的半个"。

```cpp
//汉字占两个字节，每个字节的ASCII码最高位均为1，由于char默认为带符号类型，
#include<string>   所以汉字的ASCII码小于0，而英文数字等其他字符占一个字节
#include<iostream>
using namespace std;
int main(){
    string str;
    int n;
    while(cin>>str){
        cin>>n;
        int num=0;
        int all=n;
        while(all>=0&&str[all--]<0)
            ++num;
        if(num%2==0&&str[n-1]<0)
            --n;
        cout<<str.substr(0,n)<<endl;

    }
    return 0;
}
```

一个 DNA 序列由 A/C/G/T 四个字母的排列组合组成。G 和 C 的比例（定义为 GC-Ratio）是序列中 G 和 C 两个字母的总的出现次数除以总的字母数目（也就是序列长度）。在基因工程中，这个比例非常重要。因为高的 GC-Ratio 可能是基因的起始点。
给定一个很长的 DNA 序列，以及要求的最小子序列长度，研究人员经常会需要在其中找出 GC-Ratio 最高的子序列。

```cpp
#include<iostream>
using namespace std;
int main(){
    string str;
    int len1;
    while(cin>>str){
        cin>>len1;
        int len2=str.size();
        int count,maxlen=0;
        int left=0;
        for(int i=0;i<len2-len1;i++){
            count=0;
            for(int j=i;j<i+len1;j++){
                if(str[j]=='G'||str[j]=='C')
                    count++;
            }
            if(count>maxlen){
                left=i;
                maxlen=count;
            }
        }
        for(int i=0;i<len1;i++){
            cout<<str[left+i];
        }
        cout<<endl;
    }
}
```

•计算一个数字的立方根，不使用库函数输入参数的立方根，保留一位小数

```cpp
#include<iostream>

using namespace std;
inline double abs(double x){return (x>0?x:-x);}
int main(){
    double y;
    while(cin>>y){
        double x;
        for(x=1.0;abs(x*x*x-y)>1e-7;x=(2*x+y/x/x)/3);
        cout.precision(2);
        cout<<x<<endl;
    }
    return 0;
}
```

将一个字符中所有出现的数字前后加上符号“*”，其他字符保持不变

public static String MarkNum(String pInStr)

{

return null;

}

注意：输入数据可能有多行

```
#include<string>
#include<iostream>
using namespace std;
int main(){
    string str1;
    while(cin>>str1){
        int len=str1.size();
        bool first=true;
        for(int i=0;i<len;i++){
            char ch=str1[i];
            if(isdigit(ch)&&first){
                first=false;
                cout<<"*"<<ch;}
            else if(isalpha(ch)&&!first){
                first=true;
                cout<<"*"<<ch;
            }
            else
                cout<<ch;
        }
        if(isdigit(str1[len-1]))
           cout<<"*";
        cout<<endl;
    }
    return 0;
}
```

将一个英文语句以单词为单位逆序排放。例如“I am a boy”，逆序排放后为“boy a am I”

所有单词之间用一个空格隔开，语句中除了英文字母外，不再包含其他字符

```cpp
#include<string>
#include<string>
#include<stack>
#include<iostream>
using namespace std;
int main(){
    string str1;
    stack<string>ans;
    while(cin>>str1){

        ans.push(str1);
    }
    while(!ans.empty()){
        string top=ans.top();
        ans.pop();
        cout<<top;
        if(ans.size()>0)
            cout<<" ";
    }
    return 0;
}
```

•连续输入字符串，请按长度为 8 拆分每个字符串后输出到新的字符串数组；

•长度不是 8 整数倍的字符串请在后面补数字 0，空字符串不处理。

```cpp
#include<iostream>
using namespace std;
void handlestr(string str){
    int len=str.size();
    if(len<=8){
        while(8-len){
            str=str+'0';
            ++len;
        }
        cout<<str<<endl;
    }
    else{
        cout<<str.substr(0,8)<<endl;
        handlestr(str.substr(8,len-8));
    }
}
int main(){
    string str1,str2;
    while(cin>>str1>>str2){
        handlestr(str1);
        handlestr(str2);

    }
    return 0;
}
```

明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了N 个 1 到 1000 之间的随机整数（N≤1000），对于其中重复的数字，只保留一个，把其余相同的数去掉，不同的数对应着不同的学生的学号。然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作(同一个测试用例里可能会有多组数据，希望大家能正确处理)。

```cpp
#include<iostream>
using namespace std;
int main(){
    int N,n;
    while(cin>>N){
        int a[1001]={0};
        while(N--){
            cin>>n;
            a[n]=1;
        }
        for(int i=0;i<1001;i++){
            if(a[i])
                cout<<i<<endl;
        }

    }
    return 0;
}
```

写出一个程序，接受一个由字母和数字组成的字符串，和一个字符，然后输出输入字符串中含有该字符的个数。不区分大小写

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string str1;
    char ch1;
    while(cin>>str1>>ch1){
        int len1=str1.size();
        int count=0;
        int num1=ch1-'0';
        for(int i=0;i<len1;i++){
            int num2=str1[i]-'0';
            int num3=0;
            if(str1[i]<='Z'&&str1[i]>='A')
                num3=num2+32;
            else if(str1[i]<='z'&&str1[i]>='a')
                num3=num2-32;
            if(num1==num2||num1==num3)
                ++count;
        }
        cout<<count<<endl;
    }
    return 0;
}
```

计算字符串最后一个单词的长度，单词以空格隔开。

```cpp
#include<string>
#include<vector>
#include<iostream>
using namespace std;
int main(){
    string s;
    vector<string>ans;
    while(cin>>s);
    cout<<s.size()<<endl;
    return 0;
}
```

将给定的单链表　　：

重新排序为：

要求使用原地算法，不能改变节点内部的值，需要对实际的节点进行交换。

例如：

对于给定的单链表{1,2,3,4}，将其重新排序为{1,4,2,3}.

```cpp
class Solution {
public:
        void reorderList(ListNode *head) {
                if(head==nullptr||head->next==nullptr)
                        return;
                ListNode* fast=head;
                ListNode*slow=head;
                while(fast->next!=nullptr&&fast->next->next!=nullptr)
{
                        fast=fast->next->next;
                        slow=slow->next;
                }
                ListNode* after=slow->next;
                slow->next=nullptr;
                ListNode* pre=nullptr;
                while(after!=nullptr){
                        ListNode* tmp=after->next;
                        after->next=pre;
                        pre=after;
                        after=tmp;
                }
```

```
                ListNode *first=head;
                after=pre;
                while(first!=nullptr&&after!=nullptr){
                        ListNode*ftmp=first->next;
                        ListNode*atmp=after->next;
                        first->next=after;
                        first=ftmp;
                        after->next=first;
                        after=atmp;

                }
                return;
        }
};
```

有 N 个小朋友站在一排，每个小朋友都有一个评分

你现在要按以下的规则给孩子们分糖果：

- 每个小朋友至少要分得一颗糖果
- 分数高的小朋友要他比旁边得分低的小朋友分得的糖果多

你最少要分发多少颗糖果?

```
class Solution {
public:
    /**
     *
     * @param ratings int整型vector
     * @return int整型
     */
    int candy(vector<int>& ratings) {
        // write code here
        int len=ratings.size();
        vector<int>dp(len);
        int sum=0;
        for(int i=0;i<len;i++)
            dp[i]=1;
        for(int i=1;i<len;i++)
            if(ratings[i]>ratings[i-1])
                dp[i]=dp[i-1]+1;
        for(int i=len-2;i>=0;i--)
            if(ratings[i]>ratings[i+1]&&dp[i]<=dp[i+1])
                dp[i]=dp[i+1]+1;
        for(int i=0;i<len;i++)
            sum+=dp[i];
        return sum;
    }
};
```

使用插入排序对链表进行排序。

```
class Solution {
public:
    /**
     *
     * @param head ListNode类
     * @return ListNode类
     */
    ListNode* insertionSortList(ListNode* head) {
        // write code here
        if(head==nullptr||head->next==nullptr)
            return head;
        ListNode* dummy=new ListNode(0);
        ListNode* cur=head;
        while(cur!=nullptr){
            ListNode* pre=dummy;
            ListNode* next=cur->next;
            while(pre->next!=nullptr&&pre->next->val<cur->val){
                pre=pre->next;
            }
            cur->next=pre->next;
            pre->next=cur;
            cur=next;
        }
        return dummy->next;
    }
};
```

给定一组不重叠的时间区间，在时间区间中插入一个新的时间区间(如果有重叠的话就合并区间)。

这些时间区间初始是根据它们的开始时间排序的。

示例 1:

给定时间区间[1,3]，[6,9]，在这两个时间区间中插入时间区间[2,5]，并将它与原有的时间区间合并，变成[1,5],[6,9].

示例 2:

给定时间区间[1,2],[3,5],[6,7],[8,10],[12,16],在这些时间区间中插入时间区间[4,9]，并将它与原有的时间区间合并，变成[1,2],[3,10],[12,16].

这是因为时间区间[4,9]覆盖了时间区间[3,5],[6,7],[8,10].

```
class Solution {
public:
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval) {
        int len=intervals.size();
        vector<Interval>ans;
        int i=0;
        for(;i<len;i++){
            if(intervals[i].end<newInterval.start)
                ans.push_back(intervals[i]);
            else if(intervals[i].start>newInterval.end)
                break;
            else {
                newInterval.start=min(newInterval.start,intervals[i].start);
                newInterval.end=max(newInterval.end,intervals[i].end);
            }
        }
        ans.push_back(newInterval);
        for(;i<len;i++)
            ans.push_back(intervals[i]);
        return ans;
    }
};
```

一条仅包含字母‘A’ - ‘Z’的消息用下列的方式加密成数字

```
'A' -> 1↵'B' -> 2↵...↵'Z' -> 26
```

现在给出加密成数字的密文，请判断有多少种解密的方法
例如:
给出的密文为“12”，可以解密为"AB"(1 2) 或者"L"(12).
所以密文"12"的解密方法是 2 种.

```
    int numDecodings(string s) {
        // write code here
        int len=s.size();
        vector<int>dp(len+1,0);
        if(len==0||s[0]=='0')
            return 0;
        dp[0]=1;
        dp[1]=1;
        for(int i=2;i<=len;i++){
            if(s[i-1]>='1'&&s[i-1]<='9')
                dp[i]+=dp[i-1];
            if(s[i-2]=='1'||(s[i-2]=='2'&&s[i-1]>='0'&&s[i-1]<='6'))
                dp[i]+=dp[i-2];
        }
        return dp[len];
    }
};
```

将一个链表 m 位置到 n 位置之间的区间反转，要求时间复杂度 $O(1)$ ，空间复杂度 $O(n)$ 。
例如：
给出的链表为 $1 \to 2 \to 3 \to 4 \to 5 \to NULL, m = 2$ ， $n = 4$ ，
返回 $1 \to 4 \to 3 \to 2 \to 5 \to NULL$.
注意：
给出的 m, n 满足以下条件：
$1 \le m \le n \le$ 链表长度

```
ListNode* reverseBetween(ListNode* head, int m, int n) {
    // write code here
    ListNode* dummy=new ListNode(0);
    dummy->next=head;
    ListNode*prestart=dummy;
    ListNode*start=head;
    for(int i=1;i<m;i++){
        prestart=start;
        start=start->next;
    }
    for(int i=m;i<n;i++){
        ListNode*tmp=start->next;
        start->next=tmp->next;
        tmp->next=prestart->next;
        prestart->next=tmp;
    }
    return dummy->next;
}
```

给出两个用字符串表示的数字，将两个数字的乘积作为字符串返回。

备注：数字可以无限大，且是非负数。

```
    string multiply(string num1, string num2) {
        // write code here
        int len1=num1.size();
        int len2=num2.size();
        string s="";
        vector<int>tmp(len1+len2);
        for(int i=0;i<len1;i++)
            for(int j=0;j<len2;j++){
                tmp[i+j+1]+=(num1[i]-'0')*(num2[j]-'0');

            }
        int carry=0;
        for(int i=tmp.size()-1;i>=0;i--){
            tmp[i]+=carry;
            carry=tmp[i]/10;
            tmp[i]=tmp[i]%10;
        }
        int left=0;
        while(left<(tmp.size()-1)&&tmp[left]==0)
            left++;
        for(;left<tmp.size();left++){
            char tmpstr=tmp[left]+'0';
            s=s+tmpstr;
        }
        return s;
    }
```

给定一个字符串 s 和一组单词 dict，判断 s 是否可以用空格分割成一个单词序列，使得单词序列中所有的单词都是 dict 中的单词（序列可以包含一个或多个单词）。

例如:

给定 s= “leetcode” ；

dict=["leet", "code"].

返回 true，因为"leetcode"可以被分割成"leet code".

```
class Solution {
public:
    bool wordBreak(string s, unordered_set<string> &dict) {
        int len=s.size();
        vector<bool>dp(len+1,false);
        dp[0]=true;
        for(int pos=0;pos<len;pos++){
            for(int i=pos;i<len&&dp[pos];i++){
                string str=s.substr(pos,i-pos+1);
                if(dict.find(str)!=dict.end())
                    dp[i+1]=true;
            }
        }
        return dp[len];
    }
};
```

求给定二叉树的最小深度。最小深度是指树的根结点到最近叶子结点的最短路径上结点的数量。

```
int run(TreeNode* root) {
    // write code here
    if(root==nullptr)
        return 0;
    if(root->left==nullptr&&root->right==nullptr)
        return 1;
    int depth=0;
    queue<TreeNode*>que;
    que.push(root);
    while(!que.empty()){
        int len=que.size();
        depth++;
        for(int i=0;i<len;i++){
            TreeNode*cur=que.front();
            que.pop();
            if(cur->left==nullptr&&cur->right==nullptr)
                return depth;
            if(cur->left!=nullptr)
                que.push(cur->left);
            if(cur->right!=nullptr)
                que.push(cur->right);

        }
    }
```

现在有一个整数类型的数组，数组中只有一个元素只出现一次，其余元素都出现三次。你需要找出只出现一次的元素

注意：

你需要给出一个线性时间复杂度的算法，你能在不使用额外内存空间的情况下解决这个问题么?

```
class Solution {
public:
    /**
     *
     * @param A int整型一维数组
     * @param n int A数组长度
     * @return int整型
     */
    int singleNumber(int* A, int n) {
        // write code here
        int ones=0,twos=0,threes;
        for(int i=0;i<n;i++){
            int t=A[i];
            twos|=ones&t;
            ones^=t;
            threes=ones&twos;
            ones&=~threes;
            twos&=~threes;
        }
        return ones;

    }
};
```

现在有一个整数类型的数组，数组中素只有一个元素只出现一次，其余的元素都出现两次。

注意：

你需要给出一个线性时间复杂度的算法，你能在不使用额外内存空间的情况下解决这个问题么?

```cpp
class Solution {
public:
    /**
     *
     * @param A int整型一维数组
     * @param n int A数组长度
     * @return int整型
     */
    int singleNumber(int* A, int n) {
        // write code here
        int num=0;
        for(int i=0;i<n;i++)
            num^=A[i];
        return num;
    }
};
```

给出一个非负整数数组，你最初在数组第一个元素的位置

数组中的元素代表你在这个位置可以跳跃的最大长度

判断你是否能到达数组最后一个元素的位置

例如

A =[2,3,1,1,4], 返回 true.

A =[3,2,1,0,4], 返回 false.

```
class Solution {
public:
    /**
     *
     * @param A int整型一维数组
     * @param n int A数组长度
     * @return bool布尔型
     */
    bool canJump(int* A, int n) {
        // write code here
        int maxreach=0;
        for(int i=0;i<n&&i<=maxreach;i++){
            maxreach=max(maxreach,i+A[i]);
        }
        if(maxreach<n-1)
            return false;
        return true;
    }
};
```

给定一个 m x n 大小的矩阵（m 行，n 列），按螺旋的顺序返回矩阵中的所有元素。

例如，

给出以下矩阵：

```
[
   [ 1, 2, 3 ],
   [ 4, 5, 6 ],
   [ 7, 8, 9 ]
]
```

你应该返回[1,2,3,6,9,8,7,4,5]。

```
class Solution {
public:
        vector<int> spiralOrder(vector<vector<int> > &matrix) {
                vector<int>res;
                if(matrix.size()==0)
                        return res;
                int rowbegin=0;
                int rowend=matrix.size()-1;
```

```
                int colbegin=0;
                int colend=matrix[0].size()-1;
                while(colbegin<=colend&&rowbegin<=rowend){
                        for(int i=colbegin;i<=colend;i++)
                                res.push_back(matrix[rowbegin][i]);
                        ++rowbegin;
                        for(int i=rowbegin;i<=rowend;i++)
                                res.push_back(matrix[i][colend]);
                        --colend;
                        if(rowbegin<=rowend){
                                for(int i=colend;i>=colbegin;i--)
                                        res.push_back(matrix[rowend][
i]);
                        }
                        --rowend;
                        if(colbegin<=colend){
                                for(int i=rowend;i>=rowbegin;i--)
                                        res.push_back(matrix[i][colbe
gin]);}
                        ++colbegin;
                }
                return res;


        }
};
```

给出一个仅包含字符'('和')'的字符串，计算最长的格式正确的括号子串的长度。

对于字符串"(()"来说，最长的格式正确的子串是"()"，长度为 2.

再举一个例子：对于字符串")()())",来说，最长的格式正确的子串是"()()"，长度为 4.

```
int longestValidParentheses(string s) {
    // write code here
    stack<int>st;
    int len=s.size();
    int last=-1;
    int ans=0;
    for(int i=0;i<len;i++){
        if(s[i]=='(')
            st.push(i);
        else{
            if(!st.size())last=i;
            else{
                st.pop();
                if(st.size())
                    ans=max(ans,i-st.top());
                else
                    ans=max(ans,i-last);
            }
        }
    }
        return ans;
}
```

实现函数 next permutation（下一个排列）：将排列中的数字重新排列成字典序中的下一个更大的排列。将排列中的数字重新排列成字典序中的下一个更大的排列。

如果不存在这样的排列，则将其排列为字典序最小的排列（升序排列）

需要使用原地算法来解决这个问题，不能申请额外的内存空间

下面有机组样例，左边是输入的数据，右边是输出的答案

1,2,3→1,3,2

3,2,1→1,2,3

1,1,5→1,5,1

```
class Solution {
public:
    void nextPermutation(vector<int> &num) {
        int len=num.size();
        bool reverse=true;
        for(int i=len-1;i>0;i--){
            if(num[i-1]<num[i]){
                int tmp=num[i];
                int pos=i;
                for(int j=i;j<len;j++){
                    if(num[j]>num[i-1]&&num[j]<tmp){
                        tmp=num[j];
                        pos=j;
                    }
                }
                num[pos]=num[i-1];
                num[i-1]=tmp;
                sort(num.begin()+i,num.end());
                reverse=false;
                break;
                }
        }
        if(reverse)
            for(int i=0;i<len/2;i++)
                swap(num[i],num[len-i-1]);
        return ;

```

在不使用乘法运算符，除法运算符和取余运算符的情况下对两个数进行相除操作

```
    typedef long long ll;
    int divide(int dividend, int divisor) {
        // write code here
        ll a=abs((ll)dividend);
        ll b=abs(ll(divisor));
        int finalcount=0;
        while(a>=b){
            int count=1;
            long sum=b;
            while(sum+sum<=a){
                sum+=sum;
                count+=count;
            }
            a-=sum;
            finalcount+=count;
        }
        return(divisor>>31^dividend>>31)?-finalcount:finalcount;
    }
};
```

给出含有 n 个整数的数组 s，找出 s 中和加起来的和最接近给定的目标值的三个整数。返回这三个整数的和。你可以假设每个输入都只有唯一解。

例如，给定的整数 S = {-1 2 1 -4}, 目标值 = 1.↵ 最接近目标值的和为 2. (-1 + 2 + 1 = 2).

```
    int threeSumClosest(vector<int>& num, int target) {
        int len=num.size();
        sort(num.begin(),num.end());
        int res=num[0]+num[1]+num[len-1];
        for(int i=0;i<len-2;i++){
            int l=i+1;
            int r=len-1;
            while(l<r){
                int sum=num[i]+num[l]+num[r];
                if(sum<target)
                    l++;
                else
                    r--;
                if(abs(sum-target)<abs(res-target))
                    res=sum;
            }

        }
        return res;
    }
```

给出一个有 n 个元素的数组 S，S 中是否有元素 a,b,c 满足 a+b+c=0？找出数组 S 中所有满足条件的三元组。

注意：

1. 三元组（a、b、c）中的元素必须按非降序排列。（即 a≤b≤c）
2. 解集中不能包含重复的三元组。

> 例如，给定的数组 S = {-1 0 1 2 -1 -4},解集为(-1, 0, 1) (-1, -1, 2)

```
class Solution {
public:
    vector<vector<int> > threeSum(vector<int> &num) {
        int len=num.size();
        sort(num.begin(), num.end());
        vector<vector<int>>res;
        for(int k=0;k<len;k++){
            if(num[k]>0)
                break;
            if(k>0&&num[k]==num[k-1])continue;
            int l=k+1;
            int r=len-1;
            int tmp=-num[k];
            while(l<r){
                if(num[l]+num[r]==tmp){
                    res.push_back({num[k],num[l],num[r]});
                    while(l<r&&num[l+1]==num[l])++l;
                    while(l<r&&num[r-1]==num[r])--r;
                    ++l;--r;}
                else if(num[l]+num[r]<tmp)++l;
                else --r;
            }

        }
        return res;
```

在不使用额外的内存空间的条件下判断一个整数是否是回文数字

提示：

负整数可以是回文吗？（比如-1）

如果你在考虑将数字转化为字符串的话，请注意一下不能使用额外空间的限制

你可以将整数翻转。但是，如果你做过题目“反转数字”，你会知道将整数翻转可能会出现溢出的情况，你怎么处理这个问题?

```
class Solution {
public:
    /**
     *
     * @param x int整型
     * @return bool布尔型
     */
    bool isPalindrome(int x) {
        // write code here
        if(x<0)
            return false;
        int hi=1;
        while(x/hi>=10){
            hi*=10;
        }


        while(x!=0){
            int left=x/hi;
            int right=x%10;
            if(left!=right)return false;
            x=(x%hi)/10;
            hi=hi/100;

        }
        return true;
```

实现函数 atoi 。函数的功能为将字符串转化为整数

提示：仔细思考所有可能的输入情况。这个问题故意描述的很模糊（没有给出输入的限制），你需要自己考虑所有可能的情况。

```cpp
class Solution {
public:
    int atoi(const char *str) {
        if(*str=='\0')return 0;
        while(*str==' ')++str;
        int sign=1;
        if(*str=='+'||*str=='-')sign=(*str++=='+')?1:-1;
        long res=0;
        while(isdigit(*str)){
            res=res*10+*str++-'0';
            if(res*sign>=INT_MAX)return INT_MAX;
            if(res*sign<=INT_MIN)return INT_MIN;
        }
        return res*sign;

    }
};
```

给定两个代表非负数的链表，数字在链表中是反向存储的（链表头结点处的数字是个位数，第二个结点上的数字是十位数...），求这个两个数的和，结果也用链表表示。

输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)

输出： 7 -> 0 -> 8

```cpp
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    // write code here
    int carry=0,val=0;
    ListNode*root =new ListNode(0),*n=root;

    while(l1!=NULL||l2!=NULL||carry){
        int v1=0,v2=0;
        if(l1!=NULL){
            v1=l1->val;
            l1=l1->next;
        }
        if(l2!=NULL){
            v2=l2->val;
            l2=l2->next;
        }
        int sum=(v1+v2+carry);
        val=sum%10;
        ListNode*cur=new ListNode(val);
        n->next=cur;
        n=n->next;
        carry=sum/10;
    }
    return root->next;

}
```

假设渊子原来一个 BBS 上的密码为 zvbo9441987,为了方便记忆，他通过一种算法把这个密码变换成 YUANzhi1987，这个密码是他的名字和出生年份，怎么忘都忘不了，而且可以明目张胆地放在显眼的地方而不被别人知道真正的密码。

他是这么变换的，大家都知道手机上的字母： 1--1， abc--2， def--3， ghi--4， jkl--5， mno--6， pqrs--7， tuv--8 wxyz--9， 0--0,就这么简单，渊子把密码中出现的小写字母都变成对应的数字，数字和其他的符号都不做变换，

声明：密码中没有空格，而密码中出现的大写字母则变成小写之后往后移一位，如：X，先变成小写，再往后移一位，不就是 y 了嘛，简单吧。记住，z 往后移是 a 哦。

```cpp
#include<iostream>
#include<string>
using namespace std;
const string dict1="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
const string dict2="bcdefghijklmnopqrstuvwxyza22233344455566677778889999";

char Char_Change(char a){
    for(int i=0;i<dict1.size();i++)
        if(dict1[i]==a) return dict2[i];
    return a;
}

int main(){
    //string data="YUANzhi1987";
    string data;
    while(getline(cin,data)){
        for(int i=0;i<data.size();i++)
            data[i] = Char_Change(data[i]);
        cout<<data<<endl;
    }
    return 0;
}
```

```
int main(){
    string str1;
    while(cin>>str1){
        int len1=str1.size();
        for(int i=0;i<len1;i++){
            if(str1[i]>='A'&&str1[i]<'Z')
                str1[i]=str1[i]+32+1;
            else if(str1[i]=='Z')
                str1[i]='a';
            else if(isdigit(str1[i]))
                continue;
            else{
                int l=str1[i]-'a';
                if(l<15)
                    l=l/3+2;
                else if(l<19&&l>=15)
                    l=7;
                else if(l>=19&&l<22)
                    l=8;
                else
                    l=9;
                str1[i]='0'+l;
            }
        }
        cout<<str1<<endl;
    }
    return 0;
```

有这样一道智力题：“某商店规定：三个空汽水瓶可以换一瓶汽水。小张手上有十个空汽水瓶，她最多可以换多少瓶汽水喝？” 答案是 5 瓶，方法如下：先用 9 个空瓶子换 3 瓶汽水，喝掉 3 瓶满的，喝完以后 4 个空瓶子，用 3 个再换一瓶，喝掉这瓶满的，这时候剩 2 个空瓶子。然后你让老板先借给你一瓶汽水，喝掉这瓶满的，喝完以后用 3 个空瓶子换一瓶满的还给老板。如果小张手上有 n 个空汽水瓶，最多可以换多少瓶汽水喝

```
#include<iostream>
using namespace std;
int main(){
    int num;
    while(cin>>num){
        int sum=0;
        while(num>2){
            int mod;
            sum+=num/3;
            mod=num%3;
            num=num/3+mod;
        }
        if(num==2)
            sum++;
        cout<<sum<<endl;
    }
    return 0;
}
```

实现删除字符串中出现次数最少的字符，若多个字符出现次数一样，则都删除。输出删除这些单词后的字符串，字符串中其它字符保持原来的顺序。

注意每个输入文件有多组输入，即多个字符串用回车隔开

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    string str1;
    int a[26];
    while(cin>>str1){
        int len=str1.size();
        int min_num=len;

        for(int i=0;i<26;i++)
            a[i]=0;
        for(int i=0;i<len;i++){
            int s=str1[i]-'a';
            a[s]++;
            min_num=min(min_num,a[s]);
        }
        for(int i=0;i<len;i++){
            int s=str1[i]-'a';
            if(a[s]>min_num)
                cout<<str1[i];}
        cout<<endl;
    }
    return 0;

```

1、对输入的字符串进行加解密，并输出。

2 加密方法为：

当内容是英文字母时则用该英文字母的后一个字母替换，同时字母变换大小写,如字母 a 时则替换为 B；字母 Z 时则替换为 a；

当内容是数字时则把该数字加 1，如 0 替换 1，1 替换 2，9 替换 0；

其他字符不做变化。

3、解密方法为加密的逆过程。

```
#include<iostream>
#include<string>
using namespace std;
int main(){
	string s,s2;
	while(cin>>s){
		int len1=s.size();
		for(int i=0;i<len1;i++){
			if(s[i]>='a'&&s[i]<='y')
				s[i]=s[i]-32+1;
			else if(s[i]=='z')
				s[i]='A';
			else if(s[i]>='A'&&s[i]<='Y')
				s[i]=s[i]+32+1;
			else if(s[i]=='Z')
				s[i]='a';
			else if(s[i]>='0'&&s[i]<='8')
				s[i]=s[i]+1;
			else
				s[i]='0';
		}
		cout<<s<<endl;
		cin>>s2;
		int len2=s2.size();
		for(int i=0;i<len2;i++){
			if(s2[i]>'a'&&s2[i]<='z')
				s2[i]=s2[i]-32-1;
			else if(s2[i]=='a')
				s2[i]='Z';
			else if(s2[i]>'A'&&s2[i]<='Z')
				s2[i]=s2[i]+32-1;
			else if(s2[i]=='A')
				s2[i]='z';
			else if(s2[i]>'0'&&s2[i]<='9')
				s2[i]=s2[i]-1;
			else
				s2[i]='9';
		}
		cout<<s2<<endl;

	}
	return 0;
}
```

```
const string helper1 = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
const string helper2 = "BCDEFGHIJKLMNOPQRSTUVWXYZAbcdefghijklmnopqrstuvwxyza1234567890";
void Encrypt(string str){
    string res;
    for (int i = 0; i < str.size(); i++) {
        res += helper2[helper1.find(str[i])];
    }
    cout << res << endl;
}
void unEncrypt(string str){
    string res;
    for (int i = 0; i < str.size(); i++) {
        res += helper1[helper2.find(str[i])];
    }
    cout << res << endl;
}
int main(){
    string str1, str2;
    while(getline(cin, str1)){
        getline(cin, str2);
        Encrypt(str1);
        unEncrypt(str2);
    }
    return 0;
}
```

按照指定规则对输入的字符串进行处理。

详细描述：

将输入的两个字符串合并。

对合并后的字符串进行排序，要求为：下标为奇数的字符和下标为偶数的字符分别从小到大排序。这里的下标意思是字符在字符串中的位置。

对排序后的字符串进行操作，如果字符为 ‘0’ —— ‘9’ 或者 ‘A’ —— ‘F’ 或者 ‘a’ —— ‘f’ ，则对他们所代表的 16 进制的数进行 BIT 倒序的操作，并转换为相应的大写字符。如字符为 ‘4’ ，为 0100b，则翻转后为 0010b，也就是 2。转换后的字符为 ‘2’ ； 如字符为 ‘7’ ，为 0111b，则翻转后为 1110b，也就是 e。转换后的字符为大写 ‘E’ 。

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
int main(){
```

```
const string helper1="0123456789abcdefABCDEF";
const string helper2="084C2A6E195D3B7F5D3B7F";
string str1,str2;
while(cin>>str1>>str2){
        string s,s1,s2;
        s=str1+str2;
        int len=s.size();
        for(int i=0;i<len;i++){
                if(i%2==0)
                        s1+=s[i];
                else
                        s2+=s[i];
        }
        sort(s1.begin(),s1.end());
        sort(s2.begin(),s2.end());
        s.clear();
        for(int k=0,j=0,i=0;i<len;i++){
                if(i%2==0)
                        s+=s1[k++];
                else
                        s+=s2[j++];
        }
        for(int i=0;i<len;i++){
                int n=helper1.find(s[i]);
                if(n!=-1)
                        s[i]=helper2[n];
        }
        cout<<s<<endl;}
return 0;
```

原理：ip 地址的每段可以看成是一个 0-255 的整数，把每段拆分成一个二进制形式组合起来，然后把这个二进制数转变成
一个长整数。
举例：一个 ip 地址为 10.0.3.193

每段数字	相对应的二进制数
10	00001010
0	00000000
3	00000011
193	11000001

组合起来即为：00001010　00000000　00000011　11000001,转换为 10 进制数就是：167773121，即该 IP 地址转换后的数字就是它了。

的每段可以看成是一个 0-255 的整数，需要对 IP 地址进行校验

```
#include<stack>
#include<iostream>
using namespace std;
int main(){
        unsigned  int a,b,c,d;
        char ch;
        while(cin>>a>>ch>>b>>ch>>c>>ch>>d){
                cout<<((a<<24)|(b<<16)|(c<<8)|d)<<endl;
                cin>>a;
                cout<<((a&0xff000000)>>24)<<"."<<((a&0x00ff0000)>>16)
<<"."<<((a&0x0000ff00)>>8)<<"."<<((a&0x000000ff))<<endl;
        }
        return 0;
}
```

查找和排序

题目：输入任意（用户，成绩）序列，可以获得成绩从高到低或从低到高的排列,相同成绩都按先录入排列在前的规则处理。

例示:

jack 70

peter 96

Tom 70

smith 67

从高到低 成绩

peter 96

jack 70

Tom 70

smith 67

从低到高

smith 67

jack 70

Tom 70

peter 96

注：0 代表从高到低，1 代表从低到高

```cpp
using namespace std;
struct Mask{
    string name;
    int score;

};
bool cmp1(const Mask&a,const Mask&b){
        return a.score>b.score;
}
bool cmp2(const Mask&a,const Mask&b){
    return a.score<b.score;
}
int main(){
    int N,flag;
    while(cin>>N>>flag){
        vector<Mask>stud(N);
        for(int i=0;i<N;i++){
            cin>>stud[i].name>>stud[i].score;
        }
        if(flag==0)
            stable_sort(stud.begin(),stud.end(),cmp1);
        else
            stable_sort(stud.begin(),stud.end(),cmp2);
        for(int i=0;i<N;i++){
            cout<<stud[i].name<<" "<<stud[i].score<<endl;
        }
```

请设计一个算法完成两个超长正整数的加法。

输入参数：

String addend：加数

String augend：被加数

返回值：加法结果

*/

public String AddLongInteger(String addend, String augend)

{

/*在这里实现功能*/

return null;

}

本题有多组输入数据，请使用 while(cin>>)等方式读取

```
#include<iostream>
#include<string>
#include<stack>
using namespace std;
using namespace std;
void addzero(string&str,int targetlen,int len){
                if(len<targetlen)
                        for(int i=0;i<targetlen-len;i++)
                                str='0'+str;
        }
int main(){
        string str1,str2,cur1;
        int len1,len2;
        while(cin>>str1>>str2){
                len1=str1.length();
                len2=str2.length();
                int len=max(len1,len2);
                addzero(str1,len,len1);
                addzero(str2,len,len2);
                int num1,num2,sum,cur,c=0;
                for(int i=len-1;i>=0;i--){
                        num1=str1[i]-'0';
                        num2=str2[i]-'0';
                        sum=num1+num2+c;
                        cur=sum%10;
                        c=sum/10;
                        str1[i]='0'+cur;
                }
                if(c>0){
                        cur1='0'+cur;
                        str1=cur1+str1;
                }
                cout<<str1<<endl;
```

对于不同的字符串，我们希望能有办法判断相似程度，我们定义了一套操作方法来把两个

不相同的字符串变得相同，具体的操作方法如下：

1　修改一个字符，如把“a”替换为“b”。

2　增加一个字符，如把“abdd”变为“aebdd”。

3　删除一个字符，如把“travelling”变为“traveling”。

比如，对于“abcdefg”和“abcdef”两个字符串来说，我们认为可以通过增加和减少一个“g”的方式来达到目的。上面的两种方案，都只需要一次操作。把这个操作所需要的次数定义为两个字符串的距离，而相似度等于“距离＋1”的倒数。也就是说，“abcdefg”和“abcdef”的距离为 1，相似度为 1/2=0.5.

给定任意两个字符串，你是否能写出一个算法来计算出它们的相似度呢?

```
using namespace std;
int main(){
    string str1,str2;

    while(cin>>str1>>str2){
        int len1=str1.length();
        int len2=str2.length();
        vector<vector<int>> dp(len1+1,vector<int>(len2+1,0));
        int dist;

        for(int i=0;i<=len1;i++)
            dp[i][0]=i;
        for(int j=0;j<=len2;j++)
            dp[0][j]=j;

        for(int i=1;i<=len1;i++)
            for(int j=1;j<=len2;j++){
                if(str1[i-1]==str2[j-1])
                    dp[i][j]=dp[i-1][j-1];
                else{
                    int tmp=min(dp[i-1][j],dp[i][j-1]);
                    dp[i][j]=min(tmp,dp[i-1][j-1])+1;}
            }
        dist=dp[len1][len2];

        cout<<"1/"<<(dist+1)<<endl;
```

题目标题:

将两个整型数组按照升序合并，并且过滤掉重复数组元素[注: 题目更新了。输出之后有换行]

详细描述:

接口说明

原型:

```
#include<iostream>
#include<set>
using namespace std;
int main(){
    int n1,n2;
    set<int>S;
    while(cin>>n1){
        int num;
        S.clear();
        for(int i=0;i<n1;i++){
            cin>>num;
            S.insert(num);
        }
        cin>>n2;
        for(int i=0;i<n2;i++){
            cin>>num;
            S.insert(num);
        }
        for(auto it=S.begin();it!=S.end();it++)
            cout<<*it;
        cout<<endl;
    }
}
```

题目标题:

判断短字符串中的所有字符是否在长字符串中全部出现

详细描述:

接口说明

原型:

boolIsAllCharExist(char* pShortString,char* pLongString);

输入参数：

char*　pShortString：短字符串

char*　pLongString：长字符串

```
#include<iostream>
#include<string>
#include<set>
using namespace std;
int main(){
    set<char>set1;
    string str1,str2;
    while(cin>>str1>>str2){
        bool flag=true;
        int len1=str1.length(),len2=str2.length();
        set1.clear();
        for(int i=0;i<len2;i++){
            set1.insert(str2[i]);
        }
        for(int i=0;i<len1;i++)
            if(set1.find(str1[i])==set1.end()){
                flag=false;
                break;
            }
        if(flag)
            cout<<"true"<<endl;
        else
            cout<<"false"<<endl;}
    return 0;

}
```

首先输入要输入的整数个数 n，然后输入 n 个整数。输出为 n 个整数中负数的个数，和所有正整数的平均值，结果保留一位小数

```cpp
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n;
    while(cin>>n){
        int A[n],count1=0,count2=0,sum=0;
        for(int i=0;i<n;i++)
            cin>>A[i];
        for(int i=0;i<n;i++){
            if(A[i]>0){
                ++count1;
                sum+=A[i];
            }else if(A[i]<0)++count2;
        }
        double average=0.0;
        average=count1>0?sum*1.0/count1:0.0;
        printf("%d%s%0.1f\n",count2," ",average);
    }
```

输入整型数组和排序标识，对其元素按照升序或降序进行排序（一组测试用例可能会有多组数据）

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int n;
    while(cin>>n){
        int A[n],tag;
        for(int i=0;i<n;i++)
            cin>>A[i];
        cin>>tag;
        sort(A,A+n);
        if(tag==1)
            reverse(A,A+n);
        for(int i=0;i<n-1;i++)
            cout<<A[i]<<" ";
        cout<<A[n-1]<<endl;
    }
}
```

连续输入字符串(输出次数为N,字符串长度小于100)，请按长度为8拆分每个字符串后输出到新的字符串数组，长度不是8整数倍的字符串请在后面补数字0，空字符串不处理。
首先输入一个整数，为要输入的字符串个数。
例如：
输入：2
abc
12345789
输出：abc00000
12345678
90000000

```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
    for(int n;cin>>n;){
        vector<string>vec(n);
        for(auto&str:vec)cin>>str;
        for(auto&str:vec){
            if(str.length()%8!=0){
                str+=string(8-str.length()%8,'0');
            }
            for(int i=0;i<str.length()/8;i++){
                cout<<str.substr(8*i,8)<<endl;
            }
        }
    }

```

假设你有一个数组，其中第 i 个元素是某只股票在第 i 天的价格。

设计一个算法来求最大的利润。你最多可以进行两次交易。

注意:

你不能同时进行多个交易(即，你必须在再次购买之前出售之前买的股票)。

```
class Solution {
public:
    /**
     *
     * @param prices int整型vector
     * @return int整型
     */
    int maxProfit(vector<int>& prices) {
        // write code here
        int buy1=INT_MIN,sell1=0,buy2=INT_MIN,sell2=0;
        for(int i=0;i<prices.size();i++){
            buy1=max(buy1,-prices[i]);
            sell1=max(sell1,buy1+prices[i]);
            buy2=max(buy2,sell1-prices[i]);
            sell2=max(sell2,buy2+prices[i]);
        }
        return sell2;
    }
};
```

假设你有一个数组，其中第 i 个元素表示某只股票在第 i 天的价格。

设计一个算法来寻找最大的利润。你可以完成任意数量的交易(例如，多次购买和出售股票的一股)。但是，你不能同时进行多个交易(即，你必须在再次购买之前卖出之前买的股票)。

```
class Solution {
public:
    /**
     *
     * @param prices int整型vector
     * @return int整型
     */
    int maxProfit(vector<int>& prices) {
        // write code here
        if(prices.empty()||prices.size()<2)return 0;
        int res=0;
        for(int i=1;i<prices.size();i++){
            if(prices[i]>prices[i-1])
                res+=prices[i]-prices[i-1];
        }
        return res;

    }
};
```

假设你有一个数组，其中第　　个元素是股票在第　　天的价格。

你有一次买入和卖出的机会。(只有买入了股票以后才能卖出)。请你设计一个算法来计算可以获得的最大收益。

```
class Solution {
public:
    /**
     *
     * @param prices int整型vector
     * @return int整型
     */
    int maxProfit(vector<int>& prices) {
        // write code here
        int n=prices.size();
        int max=0;
        int lowest_price=prices[0];
        for(int i=1;i<n;i++){
            int tmp;
            tmp=prices[i]-lowest_price;
            if(tmp>max)max=tmp;
            if(prices[i]<lowest_price)lowest_price=prices[i];



        }
        return max;
    }
```

给出一个三角形，计算从三角形顶部到底部的最小路径和，每一步都可以移动到下面一行相邻的数字，

例如，给出的三角形如下：

```
[↵    [2],↵   [3,4],↵  [6,5,7],↵ [4,1,8,3]↵]
```

最小的从顶部到底部的路径和是 2　+　3　+　5　+　1　= 11。

注意：

```
class Solution {
public:
    int minimumTotal(vector<vector<int> > &triangle) {
        for(int i=triangle.size()-2;i>=0;i--)
            for(int j=0;j<triangle[i].size();j++){
                triangle[i][j]+=min(triangle[i+1][j],triangle[i+1][j+1]);

            }
        return triangle[0][0];

    }
};
```

给出一个索引 k，返回杨辉三角的第 k 行

例如，k=3，

返回[1,3,3,1].

备注：

你能将你的算法优化到只使用 O(k)的额外空间吗?

```
class Solution {
public:
    /**
     *
     * @param rowIndex int整型
     * @return int整型vector
     */
    vector<int> getRow(int rowIndex) {
        // write code here
        vector<int>dp(rowIndex+1,0);
        dp[0]=1;
        for(int i=1;i<=rowIndex;i++)
            for(int j=i;j>=1;j--)
                dp[j]=dp[j]+dp[j-1];
        return dp;
    }
};
```

给出一个值 numRows，生成杨辉三角的前 numRows 行

例如，给出　numRows　= 5,

返回

```
[↵    [1],↵    [1,1],↵   [1,2,1],↵  [1,3,3,1],↵ [1,4,6,4,1]↵]
```

```
class Solution {
public:
    /**
     *
     * @param numRows int整型
     * @return int整型vector<vector<>>
     */
    vector<vector<int> > generate(int numRows) {
        vector<vector<int>>res;
        for(int i=0;i<numRows;i++){
            vector<int>cur(i+1,1);
            for(int j=1;j<i;j++){
                cur[j]=res[i-1][j-1]+res[i-1][j];
            }
            res.push_back(cur);
        }
        return res;
```

给定一个二叉树

```
   struct TreeLinkNode {↲     TreeLinkNode *left;↲
TreeLinkNode *right;↲     TreeLinkNode *next;↲    }
```

填充所有节点的 next 指针，指向最接近它的同一层右边节点。如果没有同一层没有右边的节点，则应该将 next 指针设置为 NULL。

初始时，所有的 next 指针都为 NULL

注意：

- 你只能使用常量级的额外内存空间
- 可以假设给出的二叉树是一个完美的二叉树(即，所有叶子节点都位于同一层，而且每个父节点都有两个孩子节点)。

例如：给出如下的完美二叉树

```
        1↲       /  ↲      2    3↲     /   / ↲    4  5  6  7↲
```

调用完你给出的函数以后，这颗二叉树应该 变成：

```
        1 -> NULL↵       /  ↵      2 -> 3 -> NULL↵     /    / ↵
4->5->6->7 -> NULL
```

```
class Solution {
public:
    void connect(TreeLinkNode *root) {
        queue<TreeLinkNode*>store;
        TreeLinkNode*node=NULL;
        TreeLinkNode*pre;
        if(root==NULL)
            return;
        store.push(root);
        while(!store.empty()){
            pre=NULL;
            int n=store.size();
            for(int i=0;i<n;i++){
                node=store.front();
                store.pop();
                if(node->left)
                    store.push(node->left);
                if(node->right)
                    store.push(node->right);
                if(pre!=NULL)
                    pre->next=node;
                pre=node;
            }
            node->next=NULL;
```

给定两个字符串 S 和 T，返回 S 子序列等于 T 的不同子序列个数有多少个?

字符串的子序列是由原来的字符串删除一些字符（也可以不删除）在不改变相对位置的情

况下的剩余字符（例如，"ACE"is a subsequence of"ABCDE"但是"AEC"不是）

例如：

S ="rabbbit", T ="rabbit"

返回 3

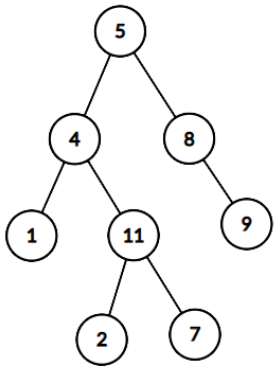
```cpp
class Solution {
public:
    /**
     *
     * @param S string字符串
     * @param T string字符串
     * @return int整型
     */
    int numDistinct(string S, string T) {
        // write code here
        int len=T.size();
        vector<int>array(len+1);
        array[0]=1;
        for(int i=1;i<S.size()+1;i++)
            for(int j=min(i,len);j>0;j--){
                if(S[i-1]==T[j-1])
                    array[j]=array[j]+array[j-1];
            }
        return array[len];
    }

};
```

给定一个二叉树

和一个值　　，请找出所有的根节点到叶子节点的节点值之和等于　　的路径，

例如：

给出如下的二叉树，　　，



返回

[

[5,4,11,2],

[5,8,9]

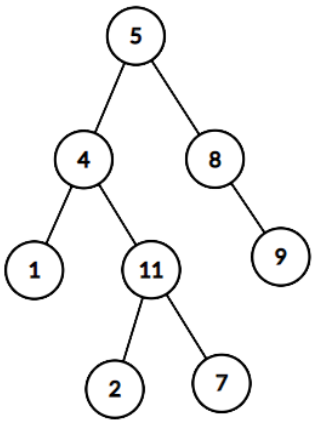
]

```
class Solution {
public:
    /**
     *
     * @param root TreeNode类
     * @param sum int整型
     * @return int整型vector<vector<>>
     */
    vector<vector<int> > pathSum(TreeNode* root, int sum) {
        // write code
        vector<vector<int>>res;
        vector<int>cur;
        pathsum2(root, sum, res,cur);
        return res;



    }
    void pathsum2(TreeNode*root,int sum,vector<vector<int>>&res,vector<int>cur){
        if(root==nullptr)
            return;
        sum-=root->val;
        cur.push_back(root->val);
        if(root->left==nullptr&&root->right==nullptr&&sum==0)
            res.push_back(cur);
        pathsum2(root->left,sum,res,cur);
        pathsum2(root->right,sum,res,cur);
    }
};
```

给定一个二叉树和一个值　　，判断是否有从根节点到叶子节点的节点值之和等于　　的路径，

例如：

给出如下的二叉树，　　，



返回 true，因为存在一条路径　　的节点值之和为 22

```
class Solution {
public:
    /**
     *
     * @param root TreeNode类
     * @param sum int整型
     * @return bool布尔型
     */
    bool hasPathSum(TreeNode* root, int sum) {
        if(root==nullptr)return false;
        sum-=root->val;
        if(sum==0&&root->left==nullptr&&root->right==nullptr)
            return true;
        return hasPathSum(root->left, sum)||hasPathSum(root->right,sum);
        // write code here
    }
};
```

本题要求判断给定的二叉树是否是平衡二叉树

平衡二叉树的性质为：　要么是一棵空树，要么任何一个节点的左右子树高度差的绝对值不超过 1。

一颗树的高度指的是树的根节点到所有节点的距离中的最大值。

```
class Solution {
public:
    /**
     *
     * @param root TreeNode类
     * @return bool布尔型
     */
    bool isBalanced(TreeNode* root) {
        if(root==nullptr)return true;
        if(abs(maxDepth(root->left)-maxDepth(root->right))>1)return false;
        return isBalanced(root->left)&&isBalanced(root->right);
    }
    int maxDepth(TreeNode*root){
        if(root==nullptr)return 0;
        return max(maxDepth(root->left),maxDepth(root->right))+1;
    }
};
```

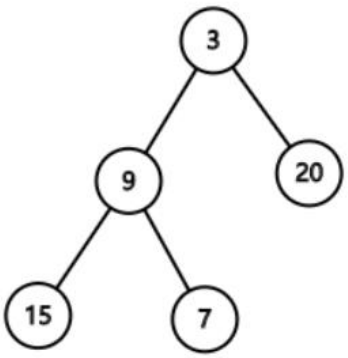
给定一个单链表，其中的元素按升序排序，请将它转化成平衡二叉搜索树（BST）

```
class Solution {
public:
    /**
     *
     * @param head ListNode类
     * @return TreeNode类
     */
    TreeNode* sortedListToBST(ListNode* head) {
        return toBST(head,nullptr);
    }
    TreeNode* toBST(ListNode* head,ListNode* tail){
        if(head==tail)
            return nullptr;
        ListNode* fast=head;
        ListNode*slow=head;
        while(fast!=tail&&fast->next!=tail){
            fast=fast->next->next;
            slow=slow->next;
        }
        TreeNode* root=new TreeNode(slow->val);
        root->left=toBST(head,slow);
        root->right=toBST(slow->next, tail);
        return root;
    }
};
```

给定一个二叉树，返回该二叉树层序遍历的结果，（从左到右，一层一层地遍历）

例如：

给定的二叉树是{3,9,20,#,#,15,7},



该二叉树层序遍历的结果是

[

[3],

[9,20],

[15,7]

```
vector<vector<int> > levelOrder(TreeNode* root) {
    // write code here
    vector<vector<int>>res;
    vector<int>cur;
    queue<TreeNode*>store;
    TreeNode*node=nullptr;
    if(root==nullptr)
        return res;
    store.push(root);
    while(!store.empty()){
        cur.clear();
        int n= store.size();
        for(int i=0;i<n;i++){
            node=store.front();
            store.pop();
            cur.push_back(node->val);
            if(node->left!=nullptr)
                store.push(node->left);
            if(node->right!=nullptr)
                store.push(node->right);

        }
        res.push_back(cur);
```

判断给出的二叉树是否是一个二叉搜索树（BST）

二叉搜索树的定义如下

- 一个节点的左子树上节点的值都小于自身的节点值
- 一个节点的右子树上节点的值都大于自身的节点值
- 所有节点的左右子树都必须是二叉搜索树

```
class Solution {
public:
    /**
     *
     * @param root TreeNode类
     * @return bool布尔型
     */
    bool isValidBST(TreeNode* root) {
        // write code here
         if(root==NULL)
             return true;
        stack<TreeNode*>bsttree;
        TreeNode*cur=root;
        TreeNode*pre=NULL;
        while(!bsttree.empty()||cur!=NULL){
            if(cur==NULL){
                cur=bsttree.top();
                bsttree.pop();
                if(pre!=NULL&&cur->val<=pre->val)
                    return false;
                pre=cur;
                cur=cur->right;
            }
            else{
                bsttree.push(cur);
                cur=cur->left;
            }
        }
        return true;
```

给定一棵二叉树，判断琪是否是自身的镜像（即：是否对称）

例如：下面这棵二叉树是对称的

```
    1
   / \
  2   2
 /\   /\
```

3 4 4 3

下面这棵二叉树不对称。

  1

  / \

 2 2

  \ \

  3 3

```
class Solution {
public:
    /**
     *
     * @param root TreeNode类
     * @return bool布尔型
     */
    bool isSymmetric(TreeNode* root) {
        if(root==NULL)return true;
        else
            return isequal(root->left,root->right);
        // write code here
    }
    bool isequal(TreeNode*leftroot,TreeNode*rightroot){
        if(leftroot==NULL&&rightroot==NULL)return true;
        else if(leftroot==NULL||rightroot==NULL)return false;
        else if(leftroot->val!=rightroot->val)return false;
        return isequal(leftroot->left, rightroot->right)&&isequal(rightroot->left,leftroot->right)
    }


};
```

给定一个由非负整数填充的 m x n 的二维数组，现在要从二维数组的左上角走到右下角，

请找出路径上的所有数字之和最小的路径。

```
int minPathSum(vector<vector<int> >& grid) {
    // write code here
    if(grid.empty())
        return 0;
    int row=grid.size();
    int col=grid[0].size();
    int res[col+1];
    fill(res,res+col+1,INT_MAX);
    res[1]=0;
    for(int i=1;i<=row;i++)
        for(int j=1;j<=col;j++){
            res[j]=min(res[j],res[j-1])+grid[i-1][j-1]
        }
    return res[col];
}
```

继续思考题目"Unique Paths":

如果在图中加入了一些障碍，有多少不同的路径?

分别用 0 和 1 代表空区域和障碍

例如

下图表示有一个障碍在 3*3 的图中央。

```
[
    [0,0,0],
    [0,1,0],
    [0,0,0]
]
```

```
    int uniquePathsWithObstacles(vector<vector<int> >& obstacleGrid) {
        if(obstacleGrid.empty()){
            return 0;
        }
        int m=obstacleGrid.size();
        int n=obstacleGrid[0].size();
        vector<vector<int>>dp(m,vector<int>(n,1));
        for(int i=0;i<m;i++){
            if(obstacleGrid[i][0]==1){
                dp[i][0]=0;
            }else
                 dp[i][0]=i==0?1:dp[i-1][0];
        }
        for(int j=0;j<n;j++){
            if(obstacleGrid[0][j]==1){
                dp[0][j]=0;
            }else
                dp[0][j]=j==0?1:dp[0][j-1];
        }
        for(int i=1;i<m;i++){
            for(int j=1;j<n;j++){
                if(obstacleGrid[i][j]==1)
                    dp[i][j]=0;
                else
                    dp[i][j]=dp[i-1][j]+dp[i][j-1];
            }
        }
        return dp[m-1][n-1];
```

一个机器人在 m×n 大小的地图的左上角（起点，下图中的标记“start"的位置）。

机器人每次向下或向右移动。机器人要到达地图的右下角。（终点，下图中的标记

“Finish"的位置）。

可以有多少种不同的路径从起点走到终点?

```
class Solution {
public:
    /**
     *
     * @param m int整型
     * @param n int整型
     * @return int整型
     */
    int uniquePaths(int m, int n) {
        // write code here
        vector<vector<int>>dp(m,vector<int>(n,1));
        for(int i=1;i<m;i++)
            for(int j=1;j<n;j++){
                dp[i][j]=dp[i][j-1]+dp[i-1][j];
            }
        return dp[m-1][n-1];
    }
};
```

将给定的链表向右转动 k 个位置，k 是非负数。

例如：

给定 1->2->3->4->5->null ， k=2，

返回 4->5->1->2->3->null。

```
    ListNode* rotateRight(ListNode* head, int k) {
        if(head==nullptr||k==0)
            return head;
        int len=1;
        ListNode *p=head;
        while(p->next){
            len++;
            p=p->next;
        }
        k=len-k%len;
        p->next=head;
        for(int i=0;i<k;i++){
            p=p->next;
        }
        head=p->next;
        p->next=nullptr;
        return head;
        // write code here
    }
```

给出一个无序的整数型数组，求不在给定数组里的最小的正整数

例如：

给出的数组为[1,2,0] 返回 3,

给出的数组为[3,4,-1,1] 返回 2.

你需要给出时间复杂度在 O(n)之内并且空间复杂度为常数级的算法

```
public:
    /**
     *
     * @param A int整型一维数组
     * @param n int A数组长度
     * @return int整型
     */
    int firstMissingPositive(int* A, int n) {
        // write code here
       for(int i=0;i<n;i++){
           while(A[i]>0&&A[i]<=n&&A[A[i]-1]!=A[i])
               swap(A[i],A[A[i]-1]);
       }
        for(int i=0;i<n;i++){
            if(A[i]!=i+1)
                return i+1;
        }
        return n+1;


    }
};
```

给出 n 对括号，请编写一个函数来生成所有的由 n 对括号组成的合法组合。

例如，给出 n=3，解集为：

"((()))", "(()())", "(())()", "()(())", "()()()"

```
vector<string> generateParenthesis(int n) {
    vector<string>list;
    dfs(n,0,0,"",list);
    return list;
}
void dfs(int n,int x,int y,string s,vector<string>&list){
    if(y==n){
        list.push_back(s);
        return;
    }
    if(x<n){
       dfs(n,x+1,y,s+"(",list);
    }
    if(x>y){
        dfs(n,x,y+1,s+")",list);
    }
}
```

给出两个二叉树，请写出一个判断两个二叉树是否相等的函数。

判断两个二叉树相等的条件是：两个二叉树的结构相同，并且相同的节点上具有相同的值。

```
class Solution {
public:
    /**
     *
     * @param p TreeNode类
     * @param q TreeNode类
     * @return bool布尔型
     */
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==NULL&&q==NULL)return true;
        else if(p==NULL||q==NULL)return false;
        else if(p->val!=q->val)return false;

        return isSameTree(p->left, q->left)&&isSameTree(q->right, p->right);

        // write code here
    }
};
```

给出一个仅包含字符'(',')','{','}','['和']',的字符串，判断给出的字符串是否是合法的括号序列

```
bool isValid(string s) {
    // write code here
    int n=s.size();
    stack<char>ch;
    for(int i=0;i<n;i++){
        if(s[i]=='('||s[i]=='['||s[i]=='{')
            ch.push(s[i]);
        else if(!ch.empty()){
            char top=ch.top();
            ch.pop();
            if(s[i]==')'&&top=='('||s[i]==']'&&top=='['||s[i]=='}'&&top=='{')
                continue;
            else
                return false;
        }
        else
            return false;
    }
    if(!ch.empty())return false;
    return true;
}
```

给出含有 n 个整数的数组 s，找出 s 中和加起来的和最接近给定的目标值的三个整数。返回这三个整数的和。你可以假设每个输入都只有唯一解。

```
例如，给定的整数 S = {-1 2 1 -4}, 目标值 = 1.↵↵ 最接近目标值的和为 2. (-1 + 2 + 1 = 2).
```

```
int threeSumClosest(vector<int>& num, int target) {
    int n=num.size();
    int result=num[0]+num[1]+num[2];
    sort(num.begin(),num.end());
    for(int i=0;i<n-2;i++){
        int start=i+1;
        int end=n-1;
        while(start<end){
        int sum=num[i]+num[start]+num[end];
            if(sum<target)
                start++;
            else
                end--;
            if(abs(sum-target)<abs(result-target))
                result=sum;

        }
```

给出一个索引 k，返回杨辉三角的第 k 行

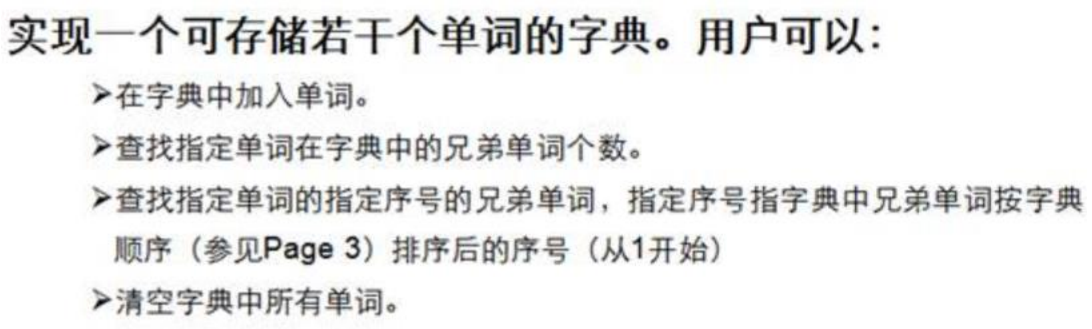
例如，k=3，

返回[1,3,3,1].

备注：

你能将你的算法优化到只使用 O(k)的额外空间吗?

```
class Solution {
public:
    /**
     *
     * @param rowIndex int整型
     * @return int整型vector
     */
    vector<int> getRow(int rowIndex) {
        // write code here
        vector<int>dp(rowIndex+1,0);
        dp[0]=1;
        for(int i=1;i<=rowIndex;i++)
            for(int j=i;j>=1;j--)
                dp[j]=dp[j]+dp[j-1];
        return dp;
    }
};
```




```
#include<string>
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
bool isbrother(string str1,string str2){
        if(str1.size()==str2.size()){
                if(str1==str2)
                        return false;
                sort(str1.begin(),str1.end());
                sort(str2.begin(),str2.end());
```

```
                if(str1==str2)
                        return true;
                return false;
        }
        return false;

}
int main(){
        int N,k;
        string str1;
        vector<string>input;
        while(cin>>N){
                input.clear();
                while(N--){
                        string str2;
                        cin>>str2;
                        input.push_back(str2);
                }
                sort(input.begin(),input.end());
                cin>>str1>>k;
                int len=input.size();
                int count=0;
                string tmp;
                for(int i=0;i<len;i++){
                        if(isbrother(input[i],str1)){
                                count++;
                                if(count==k)
                                        tmp=input[i];
                        }
                }

                cout<<count<<endl;
                if(count>0&&count>=k)
                        cout<<tmp<<endl;


        }
        return 0;

}
```

题目描述

现有一组砝码，重量互不相等，分别为 m1,m2,m3…mn;

每种砝码对应的数量为 x1,x2,x3...xn。现在要用这些砝码去称物体的重量(放在同一侧)，问能称出多少种不同的重量。

第一行：n --- 砝码数(范围[1,10])

第二行：m1 m2 m3 ... mn --- 每个砝码的重量(范围[1,2000])

第三行：x1 x2 x3 .... xn --- 每个砝码的数量(范围[1,6])

```
#include<iostream>
#include<set>
#include<vector>

using namespace std;
int main(){
    int n;
    while(cin>>n){
        vector<int>m(n,0);
        vector<int>x(n,0);
        set<int>w={0},tmp={0};
        for(auto&i:m)
            cin>>i;
        for(auto&i:x)
            cin>>i;
        for(int i=0;i<n;i++){
            for(int k=1;k<=x[i];k++){
```

```
                    for(int it:tmp){
                        w.insert(k*m[i]+it);
                    }
                }
                tmp=w;
                }
            cout<<w.size()<<endl;
        }
}
```

题目描述

若两个正整数的和为素数，则这两个正整数称之为“素数伴侣”，如 2 和 5、6 和 13，它们能应用于通信加密。现在密码学会请你设计一个程序，从已有的 N（N 为偶数）个正整数中挑选出若干对组成“素数伴侣”，挑选方案多种多样，例如有 4 个正整数：2，5，6，13，如果将 5 和 6 分为一组中只能得到一组“素数伴侣”，而将 2 和 5、6 和 13 编组将得到两组“素数伴侣”，能组成“素数伴侣”最多的方案称为“最佳方案”，当然密码学会希望你寻找出“最佳方案”。

输入:

有一个正偶数 N（N≤100），表示待挑选的自然数的个数。后面给出具体的数字，范围为[2,30000]。

输出:

输出一个整数 K，表示你求得的“最佳方案”组成“素数伴侣”的对数。

```cpp
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>
#include <algorithm>
#include <vector>
using namespace std;

vector<int> G[105];

bool flag[105];
int pre[105];

bool dfs(int k){
    int x;
    for(int i=0;i<G[k].size();i++){
        x=G[k][i];
        if (flag[x]) continue;
        flag[x]=true;
        if((pre[x]==0)||dfs(pre[x])){
            pre[x]=k;
            return true;
        }
    }
    return false;
}

bool isprime[80000];
int nums[105];

```

```cpp
int main(){
    memset(isprime,1,sizeof(isprime));
    isprime[0]=isprime[1]=false;
    for(int i=2;i*i<80000;i++){
        if(isprime[i])
            for(int j=i*i;j<80000;j+=i)
                isprime[j]=false;
    }

    int n;
    while(~scanf("%d",&n)){
        for(int i=1;i<=n;i++){
            scanf("%d",&nums[i]);
        }
        for(int i=1;i<=n;++i){
            for(int j=i+1;j<=n;++j){
                if(isprime[nums[i]+nums[j]]){
                    (nums[i]&1)?G[i].push_back(j):G[j].push_back(i);
                }
            }
        }

        memset(pre,0,sizeof(pre));
        int ans=0;
        for(int i=1;i<=n;i++){
            memset(flag,false,sizeof(flag));
            if (dfs(i)) ans++;
        }
        printf("%d\n",ans);

        for(int i=1;i<=n;++i){
            G[i].clear();
        }
```

输出 7 有关数字的个数，包括 7 的倍数，还有包含 7 的数字（如 17，27，37...70，71，72，73...）的个数（一组测试用例里可能有多组数据，请注意处理）

```cpp
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main()
{
    long int n;
    while (cin>>n)
    {

        int num=0;
        int i;
        string str;
 //     stringstream a;  这句放在这里就不好使了
        for (i=7;i<=n;i++)
        {
            //string str;
            stringstream a;
            a << i;
            a >> str;
            if ((str.find('7')!=str.npos)||(i%7==0))
            {
                num++;
            }
        }
        cout<<num<<endl;
    }
    return 0;
}
```

题目描述

开发一个简单错误记录功能小模块，能够记录出错的代码所在的文件名称和行号。

处理：

1、 记录最多8条错误记录，循环记录（或者说最后只输出最后出现的八条错误记录），对相同的错误记录（净文件名（保留最后16位）称和行号完全匹配）只记录一条，**错误计数增加；**

2、 超过16个字符的文件名称，只记录文件的最后有效16个字符；

3、 输入的文件可能带路径，记录文件名称不能带路径。

输入描述:

> 一行或多行字符串。每行包括带路径文件名称，行号，以空格隔开。

输出描述:

> 将所有的记录统计并将结果输出，格式：文件名 代码行数 数目，一个空格隔开，如：

示例1

输入 复制

```
E:\V1R2\product\fpgadrive.c  1325
```

```cpp
using namespace std;
string getfilename(string& path) {
    int pos = path.rfind('\\');
    string filename = path.substr(pos + 1);
    if (filename.size() > 16)
        return filename.substr(filename.size() - 16);
    return filename;
}

struct ErrRecord {
    string filename;
    int LineNo;
    int count;
    ErrRecord(string file, int no) :filename(file), LineNo(no), count(1) {};
    bool operator==(const ErrRecord& a) {
        return a.filename == filename && a.LineNo == LineNo;
    }
};

int main() {
    string str;
    int Line;
    vector<ErrRecord>myrec;
    myrec.clear();
    while (cin >> str >> Line) {
        string str1 = getfilename(str);
        ErrRecord record(str1, Line);
        auto res = find(myrec.begin(), myrec.end(), record);
        if (res == myrec.end()) {
            myrec.push_back(record);
        }
        else
            res->count++;
    }
    int count = 0;
    for (auto item : myrec) {
        if (count + 8 >= myrec.size()) {
            cout << item.filename << " " << item.LineNo << " " << item.count << endl;
        }
        count++;
```

马拉车

```cpp
using namespace std;
int Manacher(string oriStr) {
    string newStr;
    int len = oriStr.size();
    for (int i = 0; i < len; i++) {//插入间隔符
        newStr += '#';
        newStr += oriStr[i];
    }
    newStr += '#';
    len = 2 * len + 1;              //新串长度，必为奇数
    int maxRight = 0;               //当前访问到的所有回文子串中，所能触及的最右一个字符的位置
    int pos = 0;                    //maxRight对应的回文子串对称轴的位置
    int*RL = new int[len];          //RL[i]记录以i为对称轴的最长回文子串半径长度（对称轴到最左或最右的
    int maxLength = 0;              //记录最长回文子串长度
    for (int i = 0; i < len; i++) {
        if (i < maxRight) {         //分两种情况，i在maxRight左边和右边
            RL[i] = min(RL[2 * pos - i], maxRight - i);
        }
        else RL[i] = 1;
        while (i - RL[i]>=0 && RL[i] + i < len && newStr[i - RL[i]] == newStr[i + RL[i]])
            RL[i]++;                //以i为中心，在上步的基础上扩展，直至到达边界或左右字符不相等
        if (maxRight < RL[i] + i - 1) {//更新maxRight和pos
            maxRight = RL[i] + i - 1;
            pos = i;
        }
        maxLength = max(maxLength, RL[i] - 1);//对以i为中心的回文子串在原串总的长度即为RL[i] - 1
                                              //证明：新串中回文子串长度为2*RL[i]-1，其中RL[i]个
                                              //插入字符，则剩下的RL[i]-1个为原字符
    }
    return maxLength;
}
```

定义一个二维数组N*M（其中2<=N<=10;2<=M<=10），如5 × 5数组下所示:

int maze[5][5] = {

0, 1, 0, 0, 0,

0, 1, 0, 1, 0,

0, 0, 0, 0, 0,

0, 1, 1, 1, 0,

0, 0, 0, 1, 0,

};

它表示一个迷宫，其中的1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左上角到右下角的最短路线。入口点为[0,0],既第一空格是可以走的路。

Input

一个N × M的二维数组，表示一个迷宫。数据保证有唯一解,不考虑有多解的情况，即迷宫只有一条通道。

Output

```
    if (pre != -1) {
        Print(arr[pre].pre);
    }
    else
        return;
    cout << "(" << arr[pre].x << "," << arr[pre].y << ")" << endl;
}
void BFS(int n, int m,vector<vector<int>>&vis) {
    Point start(0, 0, -1);
    queue<Point>que;
    arr.clear();
    que.push(start);
    while (!que.empty()) {
        Point cur = que.front();
        que.pop();
        arr.push_back(cur);
        int x = cur.x, y = cur.y, nx, ny;
        for (int i = 0;i < 4;i++) {
            nx = x + dx[i];
            ny = y + dy[i];
            if (!isinrange(nx,n) || !isinrange(ny, m)|| vis[nx][ny])
                continue;
            vis[nx][ny] = 1;
            int pre = arr.size() - 1;
            Point newpoint(nx, ny, pre);
            que.push(newpoint);
            if (nx == n - 1 && ny == m - 1) {
                Print(arr.size() - 1);
                cout << "(" << n - 1 << "," << m - 1 << ")" << endl;
                return;
            }

        }


    }
}
int main() {
    while (cin >> n >> m) {
        vis = vector<vector<int>>(n, vector<int>(m, 0));
        for (auto& i : vis)
            for (auto& j : i) {
                cin >> j;
            }
        BFS(n, m,vis);
    }
    return 0;
}
```

输出倒数的数

```cpp
#include<iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) :val(x), next(NULL) {}
};
int main() {
    int N;

    int k;
    while (cin >> N) {
        ListNode* list = new ListNode(-1);
        ListNode* head = list;
        for (int i = 0;i < N;i++) {
            int data;
            cin >> data;
            ListNode* q = new ListNode(data);
            list->next = q;
            list = list->next;
        }
        cin >> k;
        if (k == 0)
            cout << "0" << endl;
        else if (N - k >= 0) {
            for (int i = 0;i <= N - k;i++)
                head = head->next;
            cout << head->val << endl;

        }
        return 0;
    }
}
```