

1、【促销活动】

题目描述:系统需要提供人民币(CNY)、美元(USD)、英镑(GBP)、港币(HKD) 价值转换功能。为了简单处理,题目的说明和考生调试可以用下面的默认汇率:

1 CNY = 2 HKD

1 USD = 8 CNY

1 GBP = 2 USD

请按照汇率实现货币最大兑换和最小兑换功能。例如:最大兑换,即尽量往价值高的货币兑换: Currency(1200, CNY) = 75 GBP

Currency(127, HKD) = 3 GBP 1 USD 7 CNY 1 HKD

最小兑换,即尽量往价值低的货币兑换,直接转换为多少 HKD 即可。Currency(20, CNY) = 40 HKD

汇率是实时变化的,但是短期内大致符合当前价值规律,即相同数值的货币,价值满足 GBP>USD>CNY>HKD, 每组输入数据都会有不同的输入汇率,需要按设置的汇率进行兑换。

解答要求

时间限制: 1000ms, 内存限制: 64MB 输入

示例输入如下: 2 8 2 127 HKD MAX

- (1) 第 1 个数字表示 1CNY 可以换成多少 HKD。
- (2) 第 2 个数字表示 1USD 可以换成多少 CNY
- (3) 第 3 个数字表示 1GBP 可以换成多少 USD
- (4) 第 4 个和第 5 个 表示 数量和货币类型(大写)
- (5) 第 6 个表示最大兑换(MAX)还是最小兑换(MIN)
- (6) 需要校验前 4 个数字必须是大于 0 的整数。不合法直接返回 "ERROR"
- (7) 货币字母大写简称(HKD、CNY、USD、GBP)、兑换方式字母大写简称(MIN/MAX)需要判断合法性,只有全部字母大写并且内容相同才是合法。
- (8) 任何其它形式(空缺、大小写不一致、出现负数、小数点、非法字符等)均不合法,直接输出字符串 "ERROR"
- (9) 每个输入数据为一行,前后无空格,数据之间以 1 个空格间隔
- (10) 转换代码不需要考虑整数溢出输出

输出字符串 "3 GBP 1 USD 7 CNY 1 HKD"

- (1) 注意输出每组货币都是 数字+空格+货币种类简称
- (2) 多组不同货币也是以 1 个空格分开,例如 3 GBP 1 USD (3)输出字符串前后没有空格
- (4) 输出字符串前后没有双引号(")
- (5) 若输入不合法直接输出字符串 "ERROR"
- (6) 最大输出时遇到结果为 0 的单位不需要输出。例如输入:2 8 2 17 HKD MAX

错误输出:0 GBP 1 USD 0 CNY 1 HKD 正确输出:1 USD 1 HKD

样例

输入样例 1 复制

2 8 2 127 HKD MAX

1 6 1 127 HKD max

2 8 2 17 HKD MAX

输出样例 1

3 GBP 1 USD 7 CNY 1 HKD

ERROR

1 USD 1 HKD

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

int Split(string input, vector<string>& output, string pattern);
string MaxExchange(int count, int cnyToHkd, int usdTocny, int gbpTousd);
string MinExchange(int count);
bool ParamValid(vector<string> param, string& style, string& type, int& cnyToHkd, int& usdTocny, int& gbpTousd, int& count);

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        // 人民币转港币汇率 美元转人民币汇率 英镑转美元的汇率
        int cnyToHkd = 2, usdTocny = 8, gbpTousd = 2;
        int count = 0;
        string style, type;

        vector<string> inputArray; Split(input_string, inputArray, " ");
        // 输入参数转换为对应的值, 并做验证
        if (!ParamValid(inputArray, style, type, cnyToHkd, usdTocny, gbpTousd, count))
        {
```

```

        cout << "ERROR" << endl;
        continue;
    }
    // 将金额全部转换为汇率最小的港币HKD
    if (style == "HKD") // 港币
    {
        count = count;
    }
    else if (style == "CNY") // 人民币
    {
        count = count * cnyToHkd;
    }
    else if (style == "USD") // 美元
    {
        count = count * usdTocny * cnyToHkd;
    }
    else if (style == "GBP") // 英镑
    {
        count = count * gbpTousd * usdTocny * cnyToHkd;
    }
    if (type == "MAX")
    {
        cout << MaxExchange(count, cnyToHkd, usdTocny, gbpTousd) << endl; continue;
    }
    cout << MinExchange(count) << endl;
}
return 0;
}

int Split(string input, vector<string>& output, string pattern)
{
    string::size_type pos;
    input += pattern; // 扩展字符串以方便操作
    for (int i = 0; i < input.size(); i++)
    {
        pos = input.find(pattern, i); if (pos < input.size())
        {
            string s = input.substr(i, pos - i); if ((s != " ") && (!s.empty()))
            {
                output.push_back(s);
            }
            i = pos + pattern.size() - 1;
        }
    }
    return 0;
}

string MaxExchange(int count, int cnyToHkd, int usdTocny, int gbpTousd)
{
    int gbpCount = count / (cnyToHkd * usdTocny * gbpTousd); // 英镑整数
    int gbpnext = count % (cnyToHkd * usdTocny * gbpTousd); // 英镑剩余的部分
    int usdCount = gbpnext / (cnyToHkd * usdTocny); // 美元整数
    int usdnext = gbpnext % (cnyToHkd * usdTocny); // 美元剩余的部分
    int cnyCount = usdnext / (cnyToHkd); // 人民币整数
    int cnynext = usdnext % cnyToHkd; // 人民币剩余的部分
    string result;
    result.append(gbpCount == 0 ? "" : to_string(gbpCount) + " GBP ");
    result.append(usdCount == 0 ? "" : to_string(usdCount) + " USD ");
    result.append(cnyCount == 0 ? "" : to_string(cnyCount) + " CNY ");
    result.append(cnynext == 0 ? "" : to_string(cnynext) + " HKD");
    return result;
}

string MinExchange(int count)
{
    return to_string(count) + " HKD";
}

bool ParamValid(vector<string> param, string& style, string& type, int& cnyToHkd, int& usdTocny, int& gbpTousd, int& count)
{
    if (param.size() != 6)
    {
        return false;
    }
    vector<string> style_vector; style_vector.push_back("GBP"); style_vector.push_back("USD");
    style_vector.push_back("CNY"); style_vector.push_back("HKD");

    vector<string> type_vector; type_vector.push_back("MAX"); type_vector.push_back("MIN");

    cnyToHkd = stoi(param[0]);
    usdTocny = stoi(param[1]); gbpTousd = stoi(param[2]); count = stoi(param[3]);
    // 前四个是大于0的整数
    if ((cnyToHkd <= 0) || (usdTocny <= 0) || (gbpTousd <= 0) || (count <= 0))
    {
        return false;
    }
}

```

```

    }
    bool style_flag = false;
    for (int i = 0; i < style_vector.size(); i++)
    {
        if (param[4] == style_vector[i])
        {
            style_flag = true;
        }
    }

    if (!style_flag)
    {
        return false;
    }

    style = param[4];
    bool type_flag = false;
    for (int i = 0; i < type_vector.size(); i++)
    {
        if (param[5] == type_vector[i])
        {
            type_flag = true;
        }
    }
    if (!type_flag)
    {
        return false;
    }
    type = param[5];
    return true;
}

```

2、【拆分输出字符串】

题目描述：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

```

#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main(int argc, char** argv)
{
    int N;
    while (cin >> N)
    {
        vector<string>string_vector;
        for (int i = 0; i < N; i++)
        {
            string input = "";
            cin >> input;
            if (!input.empty())
            {
                string_vector.push_back(input);
            }
        }
        for (int i = 0; i < string_vector.size(); i++)
        {
            while (string_vector[i].size() > 8)//只要是字符串长度大于8
            {
                cout << string_vector[i].substr(0, 8) << endl;//从0开始，输出长度为8个字符
                string_vector[i] = string_vector[i].substr(8);//更新字符串从第8个位置后开始
            }
            cout << string_vector[i].append(8 - string_vector[i].size(), '0') << endl;//调用append函数补0
        }
    }
    return 0;
}

```

3、【一个正整数到 Excel 编号之间的转换】

题目描述：完成从一个正整数到 Excel 编号之间的转换。

用过 excel 的都知道 excel 的列编号是这样的：

abc.....zaaabac.....azbabbbc.....yzzazbzc.....zzaaaaabaac.....

分别代表以下编号：

123.....26272829.....52535455.....676677678679.....702703704705.....请写个函数，完成从一个正整数到这种字符串之间的转换。

原型：intTranslate(intN,string&ans)

功能:正整数到 Excel 编号字符串转换

输入:N

输出:ans

返回:0

分析:

我的第一感觉是 A,B,C...,Z,AA,AB,AC,...ZZ,AAA,AAB.....可以转换成“进制数”表示形式，由于一共 26 个英文字母，可以转换成 26 进制表示形式。转换成 26 进制形式后 26 进制数 1 对应 A、2 对应 B...25 对应 Y，这样 Z 只能对应 26 进制数 10。而十进制数 27 转换成 26 进制数为 11 即可对应成 AA，十进制数 52 转换成 26 进制数为 20 对应为 AZ。所有我们需要对能被 26 整除的正整数进行特殊处理。

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

int Translate(int N, string& ans)
{
    if ((N <= 0))
    {
        return -1;
    }
    ans = "";
    while (N > 0)
    {
        N -= 1;
        ans += ((N % 26) + 'A');
        N /= 26;
    }
    reverse(ans.begin(), ans.end());
    return 0;
}

int main(int argc, char** argv)
{
    int num;
    while (1)
    {
        cin >> num;
        string str;
        Translate(num, str);
        cout << str << endl;
    }
    return 0;
}
```

4、【字符串简单数据解压缩】

题目描述：将一段压缩后的字符串解压缩，并且排序输出

解压规则：

每个字符串后面跟随一个数字，表示这个字符串的重复次数。例如，“a5”解压缩的结果为“aaaaa”；“abc3”解压缩后的结果为“abcabcabc”。

排序规则：

1、根据每个字符串的重复次数升序排序，然后输出结果。例如，“a3b2”，输出的结果为“bbaaaa”。

2、如果字符重复次数一样，则根据 ASCII 编码顺序做升序排序，然后输出结果。例如，“b2a2”，输出的结果为“aabb”

输入描述：

输入的原始字符串仅包含字母和数字

输出描述：

输出的结果字符串仅包含字母

示例 1

输入：a11b2bac3bad3abcd2

输出：bbabcbdbcdbacbacbacbadbadbadadaaaaaaaaaa

```
#include<iostream>
#include<string>
#include<vector>
#include<map>
#include<algorithm>
using namespace std;

typedef struct TypeN
{
    vector<string> vector_string;
}TypeN;
int main(int argc, char** argv)
{
    string input_string;
    while (cin >> input_string)
    {
        string temp;
        int len = input_string.size();
        int i = 0;
        int num = 0;
        int flag = 0;
        map<int, TypeN> typen_map;
        while (i < len)
        {
            if (input_string[i] >= '0' && input_string[i] <= '9')
            {
                num = num * 10 + input_string[i] - '0';
                i++;
                while (input_string[i] >= '0' && input_string[i] <= '9' && i < len)
                {
                    num = num * 10 + input_string[i] - '0';//字符转为10进制数字
                    i++;
                }
                TypeN string_struct;
                if (typen_map.find(num) == typen_map.end())//键值对不存在
                {
                    typen_map[num] = string_struct;
                }
                else
                {
                    string_struct = typen_map[num];
                }
                string mm = "";
                for (int j = 0; j < num; j++)
                {
                    mm += temp;
                }
                string_struct.vector_string.push_back(mm);
                typen_map[num] = string_struct;
                temp = "";
                num = 0;
            }
            else
            {
                temp.push_back(input_string[i++]);
            }
        }
        for (int i = 0; i < 1024; i++)
        {
            TypeN sn = typen_map[i];
            sort(sn.vector_string.begin(), sn.vector_string.end());
            for (int i = sn.vector_string.size() - 1; i > -1; i--)
            {
                cout << sn.vector_string[i];
            }
        }
        cout << endl;
    }
    return 0;
}
```

5、【字符串压缩】

通过键盘输入一串小写字母(a~z)组成的字符串。请编写一个字符串压缩程序， 将字符串中连续出席的重复字母进行压缩，并输出压缩后的字符串。

压缩规则：

1. 仅压缩连续重复出现的字符。比如字符串"abcbc"由于无连续重复字符，压缩后的字符串还是"abcbc"。
2. 压缩字段的格式为"字符重复的次数+字符"。例如：字符串"xxxxxxxxyz"压缩后就成为"3x6yz"

```
#include <iostream>
#include <string>
using namespace std;
string myZIP(string str);

string myZIP(string str)
{
    string zip; // "aaaabbndaa" 压缩后 a4b2nda2
    int n = 1;
    for (int i = 0; i < str.Length(); i++)
    {
        if (str[i] == str[i + 1])
        {
            n++;
        }
        else
        {
            string tmp = to_string(n); // int 转 string
            zip += str[i];
            if (n != 1)
                zip += tmp;
            n = 1;
        }
    }
    return zip;
}

int main()
{
    string str = ""; // 压缩后 a4b2nda2
    getline(cin, str);
    cout << myZIP(str) << endl;
    return 0;
}
```

6、【搜索矩阵】

题目描述：实现一个程序 `search_matrix(matrix)`, 参数 `matrix` 是一个仅包含 0 或 1 两种数字的矩阵, 程序应返回输入矩阵中包含的最大正方形子矩阵(长和宽相等)的区域面积。
例如: 如果 `matrix` 是["1010111111","000000011","1010110111","0000110001", 那么它看起来像下面的阵:

```
1010111111
0000000111
1010110111
0000110001
```

对于上面的输入, 最大的子矩阵是一个 3x3 的矩阵, 程序只要返回最大子矩阵的面积即可, 如上面的矩阵即返回 9(3x3)。

输入描述:

第 1 行输入为一个数字 N, 代表下面有几行

第 2 行到第 N+1 行是代表矩阵的 0 和 1 组成的字符串, 每行的长度相同

输出描述:

返回一个数字, 代表输入矩阵的最大正方形子矩阵的面积。

示例 1

输入

```
3
110
111
110
```

输出

```
4
```

```
#include <string>
#include <vector>
#include <iostream>
using namespace std;
int MaxSquare(vector<vector<char> > matrix)
{
    int len = 0;
    if (matrix.size() == 0 || matrix[0].size() == 0)
    {
        return 0;
    }
    // 创建vector<vector<int>> 大小为matrix.size(), 里面的每个元素为vector<int>, 大小为matrix[0].size(), 初始值为0
    vector<vector<int>> dp(matrix.size(), vector<int>(matrix[0].size(), 0));
    for (int i = 0; i < matrix.size(); ++i)
    {
        for (int j = 0; j < matrix[0].size(); ++j)
        {
            if (matrix[i][j] == '1') // 右下角的只为了1
            {
                if (i == 0 || j == 0)
                {
                    dp[i][j] = 1; // 起始点
                }
                else
                {
                    dp[i][j] = min(min(dp[i][j - 1], dp[i - 1][j]), dp[i - 1][j - 1]) + 1;
                }
                len = max(len, dp[i][j]);
            }
        }
    }
    return len * len;
}

int main(int argc, char** argv)
{
    int N;
    while (cin >> N)
    {
        vector<vector<char> > matrix;
        for (int i = 0; i < N; i++)
        {
            string input;
            cin >> input;
            vector<char> char_vec;
            for (int j = 0; j < input.size(); j++)
            {
                char_vec.push_back(input[j]);
            }
            matrix.push_back(char_vec);
        }
        cout << MaxSquare(matrix) << endl;
    }
    return 0;
}
```

7、【猴子吃桃】

题目描述：孙悟空喜欢吃蟠桃，一天他乘守卫蟠桃园的天兵天将离开了而偷偷的来到王母娘娘的蟠桃园偷吃蟠桃。已知蟠桃园有 N 棵蟠桃树，第 i 棵蟠桃树上有 $N[i]$ （大于 0）个蟠桃，天兵天将将在 H （不小于蟠桃树棵数）小时后回来。孙悟空可以决定他吃蟠桃的速度 K （单位：个/小时），每个小时他会选则一颗蟠桃树，从中吃掉 K 个蟠桃，如果这棵树上的蟠桃数小于 K ，他将吃掉这棵树上所有蟠桃，然后这一小时内不再吃其余蟠桃树上的蟠桃。孙悟空喜欢慢慢吃，但仍想在天兵天将回来前将所有蟠桃吃完。求孙悟空可以在 H 小时内吃掉所有蟠桃的最小速度 K （ K 为整数）。

输入描述

从标准输入中读取一行数字，前面数字表示每棵树上蟠桃个数，最后的数字表示天兵天将离开的时间。

输出描述

吃掉所有蟠桃的最小速度 K （ K 为整数）或输入异常时输出 -1。

示例 1

输入：

3 11 6 7 8

输出：

4

思路

1. 输入如果不是正整数则报错，定义一个字符串类型 n ，先判断 n 是不是只含数字，如果不是，则输出 -1

2. 判断输入是否含 0，我们可以通过字符串变整数得到变量 a ，如果桃子树上为 0 颗果子，那么肯定异常嘛，输出 -1

3. 在输入正常的情况下，要判断有没有桃树。如果没有桃树，则数组 res 只有一个值，那么异常

4. 除猴子吃的限定小时外，对其他的树上果子进行排序，因为猴子最多吃桃树上果子最多的那颗树，多了就不符合题意最少，是不是，所以猴子 1 小时内吃的范围 k 在 1 到桃树上最多果子数

5. 下面就是遍历 k 的范围啦，判断吃每颗树要的时间加起来就是总时间小于等于规定时间，则符合题意了。如果在遍历范围内，没有符合题意的则输出 -1 啦

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
bool IsNormal(string str)
{
    for (int i = 0; str[i] != '\0'; i++)
    {
        if (str[i] < '0' || str[i] > '9')
        {
            return false;
        }
    }
    return true;
}

int Split(string input, vector<string>& output, string pattern)
{
    string::size_type pos;
    input += pattern; //扩展字符串以方便操作
    for (int i = 0; i < input.size(); i++)
    {
        pos = input.find(pattern, i);
        if (pos < input.size())
        {
            string s = input.substr(i, pos - i);
            if ((s != " ") && (!s.empty()))
            {
                output.push_back(s);
            }
            i = pos + pattern.size() - 1;
        }
    }
    return 0;
}

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        vector<int> result;
        vector<string> numbers;
        bool flag = true;
        numbers.clear();
        Split(input_string, numbers, " ");
        for (int i = 0; i < numbers.size(); i++)
        {
            if (!IsNormal(numbers[i]))
            {
                flag = false;
                cout << -1 << endl;
                break;
            }
            else
            {
                int temp = atoi(numbers[i].c_str());
                if (temp == 0)
                {
                    flag = false; cout << -1 << endl;
                    break;
                }
            }
        }
    }
}
```



```

        result.push_back(temp);
    }
}
if (flag)
{
    if (result.size() == 1)
    {
        flag = false; cout << -1 << endl;
    }
}
if (flag)
{
    int k = 1;
    sort(result.begin(), result.end() - 1);
    bool normal = false;
    while (k <= result[result.size() - 2])
    {
        int count = 0;
        for (int i = 0; i < result.size() - 1; i++)
        {
            if (result[i] % k == 0)
            {
                count += result[i] / k;
            }
            else
            {
                count += result[i] / k + 1;
            }
        }
        if (count <= result[result.size() - 1])
        {
            normal = true;
            cout << k << endl;
            break;
        }
        k++;
    }
    if (!normal)
    {
        cout << -1 << endl;
    }
}
return 0;
}

```

8、【水果搬运问题】

一组工人搬运一批水果，用一维数组存储工人编号和水果名称以及搬运重量，要求先按水果分组，然后按搬运重量排序输出。

输入描述:

第一行包括一个整数 $N(1 \leq N \leq 100)$, 代表工人的个数。接下来的 N 行每行包括两个整数 p 和 q , 分别代表每个工人的编号和搬运重量, 以及一个字符串 m , 代表水果的名称。

输出描述:

先按水果分组, 然后按工人的搬运重量从小到大进行排序, 并将排序后的信息打印出来。如果工人搬运的重量相同, 则按照编号的大小从小到大排序, 并且要求水果的输出次序同输入次序。

示例 1

输入

5

Apple 1 80

Apple 2 62

Apple 4 73

Orange 4 65

Orange 1 90

Apple 3 91

Orange 3 88

Orange 5 90

输出

Apple 2 62

Apple 4 73

Apple 1 80

Apple 3 91

Orange 4 65

Orange 3 88

Orange 1 90

Orange 5 90

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;
bool Compare(pair<int, int>& a, pair<int, int>& b)
{
    if (a.second != b.second)
    {
        return a.second < b.second;
    }
    else
    {
        return a.first < b.first;
    }
}
int main(int argc, char** argv)
{
    map<string, vector<pair<int, int>>> mapTest;
    int N;
    while (cin >> N)
    {
        mapTest.clear();
        string name;
        int id, weight;
        for (int i = 0; i < N; i++)
        {
            cin >> name >> id >> weight;
            mapTest[name].push_back(pair<int, int>(id, weight));
        }
        for (auto it = mapTest.begin(); it != mapTest.end(); it++)
        {
            sort(it->second.begin(), it->second.end(), Compare);
            for (int i = 0; i < it->second.size(); i++)
            {
                cout << it->first << " " << it->second[i].first << " " << it->second[i].second << endl;
            }
        }
    }
    return 0;
}
```

```
#include <iostream>
#include <string>
#include <vector>
```

```

#include <map>
#include <algorithm>
using namespace std;
typedef struct FruitTransport
{
    string name;
    int id;
    int weight;
}FruitTransport;
bool Compare(FruitTransport& a, FruitTransport& b)
{
    if (a.name != b.name)
    {
        return a.name < b.name;
    }
    else if (a.weight != b.weight)
    {
        return a.weight < b.weight;
    }
    else
    {
        return a.id <= b.id;
    }
}
int main(int argc, char** argv)
{
    vector<FruitTransport> vector_struct;
    int N;
    while (cin >> N)
    {
        vector_struct.clear();
        string name;
        int id, weight;
        for (int i = 0; i < N; i++)
        {
            FruitTransport ft;
            cin >> name >> id >> weight;
            ft.name = name;
            ft.id = id;
            ft.weight = weight;
            vector_struct.push_back(ft);
        }
        vector<FruitTransport>::iterator it = vector_struct.begin();
        sort(vector_struct.begin(), vector_struct.end(), Compare);
        for (auto it = vector_struct.begin(); it != vector_struct.end(); it++)
        {
            cout << it->name << " " << it->id << " " << it->weight << endl;
        }
    }
    return 0;
}

```

9、【水果摊小买卖】

题目描述：小王手里有点闲钱，想着做点卖水果的小买卖。给出两个数组 m 、 n ，用 $m[i]$ 代表第 i 个水果的成本价， $n[i]$ 代表第 i 水果能卖出的价钱，假如现在有本钱 k ，试问最后最多能赚多少钱？

说明：

1 每种水果只需买一次，只能卖一次

2 数组 m 、 n 大小不超过 50

3 数组元素为正整数，不超过 1000

输入描述：

1 数组 m 、 n

2 本钱 k

备注：

1 首行输入逗号分隔的数组 m 的元素值

2 第二行输入逗号分隔的数组 n 的元素值

3 第三行输入本钱

输出描述：

最多能赚取多少钱。

示例 1

输入

4,2,6,4

5,3,8,7

15

输出

22

解题思路：

最直接的想法就是用贪算法，每次选择当前成本能够购买的、利润最大的水果，然后卖出去，继续选择更新之后成本能够购买的、利润最大的水果，然后卖出去。

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <sstream>
using namespace std;
//输入是一行字符，比如"4,2,6,4"，输出是vector:{4,2,6,4}
vector<int> StringToVector(const string& str)
{
    stringstream sstr(str);
    string token;
    vector<string> strings;
    while (getline(sstr, token, ','))
    {
        strings.push_back(token);
    }
    vector<int> nums;
    for (int i = 0; i < strings.size(); ++i)
    {
        int num = 0;
        for (int j = 0; j < strings[i].size(); ++j)
        {
            num = num * 10 + strings[i][j] - '0';
        }
        nums.push_back(num);
    }
    return nums;
}
//定义商品信息：本钱cost,利润profit,是否被买过isbuy
typedef struct ProductInfo
{
    int cost;
    int profit;
    int isbuy = 0;
}ProductInfo;
//定义比较函数，利润大的排前面
bool Compare(const ProductInfo& left, const ProductInfo& right)
{
    return left.profit > right.profit;
}
//检查遍历一遍之后的商品信息，看看遍历一遍之后赚到的钱能不能买到这次遍历过程中因为钱不够买不起的水果
bool CheckInfo(const vector<ProductInfo>& product_infos, int k)
{
    bool flag = false;
    for (int i = 0; i < product_infos.size(); ++i)
    {
        if (product_infos[i].isbuy == 1)
        {
            continue;
        }
        if (product_infos[i].profit <= 0)
```

```

        {
            continue;
        }
        //如果该次遍历的时候中间存在有的水果买不起，但是遍历完了赚到更多的钱之后，能够买得起的情况，则还需要进行下一
次遍历
        if (k > product_infos[i].cost)
        {
            flag = true;
            break;
        }
    }
    return flag;
}
int main(int argc, char** argv)
{
    string m;
    while (getline(cin, m))
    {
        if (m.empty())
        {
            break;
        }
        string n;
        getline(cin, n);
        int k;
        cin >> k;
        vector<int> vec_m = StringToVector(m);
        vector<int> vec_n = StringToVector(n);
        //根据读取进来的本钱、售价，计算商品的信息
        vector<ProductInfo> product_infos;
        for (int i = 0; i < vec_m.size(); ++i)
        {
            ProductInfo product_info;
            product_info.cost = vec_m[i];
            product_info.profit = vec_n[i] - vec_m[i];
            if (product_info.profit <= 0)
            {
                continue;
            }
            product_info.isbuy = 0;
            product_infos.push_back(product_info);
        }
        sort(product_infos.begin(), product_infos.end(), Compare); //按照利润从大到小排序，优先买进利润高的水果
        int cost_k = k;
        while (CheckInfo(product_infos, cost_k))
        {
            //按照利润从高到低遍历所有水果，能买就买，买完就买收取利润，买不起就跳过，留待下次有更多钱了再返回来查看
            for (int i = 0; i < product_infos.size(); ++i)
            {
                if (cost_k < product_infos[i].cost)
                {
                    continue;
                }
                if (product_infos[i].profit <= 0)
                {
                    continue;
                }
                if (product_infos[i].isbuy == 1)
                {
                    continue;
                }
                cost_k = cost_k + product_infos[i].profit;
                product_infos[i].isbuy = 1;
            }
        }
        cout << cost_k << endl;
    }
    return 0;
}

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <numeric>
using namespace std;

int Str2Num(string s)
{
    int len = s.size();
    int n = 0;
    for (int i = 0; i < len; i++)
    {
        n = n * 10 + s[i] - '0';
    }
    return n;
}

```

```

void Apart(string str, vector<int>& num)
{
    int pos, n, len;
    string s;
    while (!str.empty())
    {
        len = str.size();
        pos = str.find(',');
        if (pos > 0) {
            s = str.substr(0, pos);
            n = Str2Num(s);
            num.push_back(n);
            str = str.substr(pos + 1, len - pos - 1);
        }
        else {
            n = Str2Num(str);
            num.push_back(n);
            break;
        }
    }
}

bool comp(pair<int, double>& a, pair<int, double>& b)
{
    return a.second > b.second;
}

int main(int argc, char** argv)
{
    string inputm, inputn;

    while (getline(cin, inputm))
    {
        //Input
        vector<int> m, n;
        int lenin = inputm.size();
        getline(cin, inputn);
        int k;
        cin >> k;

        Apart(inputm, m);
        Apart(inputn, n);

        //Calculate average profit and sort it
        int len = m.size();
        vector<pair<int, double>> avr;
        for (int i = 0; i < len; i++)
        {
            avr.push_back(make_pair(i, (n[i] - m[i]) / (1.0 * m[i])));
        }
        sort(avr.begin(), avr.end(), comp);

        //Buy and sell
        vector<int> visit(len, 0), win;
        int i, idx, klast = -1;
        bool brk = false;
        while (!avr.empty())
        {
            i = 0;
            while (i < len)
            {
                idx = avr[i].first;
                if (visit[i] == 0 && m[i] < k)
                {
                    k -= m[i];
                    win.push_back(n[i]);
                    visit[i] = 1;
                }
                i++;
            }
            k += accumulate(win.begin(), win.end(), 0);
            win.clear();

            if (k == klast)
            {
                break;
            }
            klast = k;
        }

        cout << k << endl;
    }

    return 0;
}

```

10、【免单统计】

题目描述：华为商城举办了一个促销活动，如果某顾客是某一秒内最早时刻下单的顾客（可能是多个人），则可以获取免单。请你编程计算有多少顾客可以获取免单。

解答要求

时间限制：3000ms,内存限制：64MB

输入

输入为 n 行数据，每一行表示一位顾客的下单时间
以（年-月-日时-分-秒.毫秒）yyyy-MM-ddHH:mm:ss.fff 形式给出。

0<n<50000

2000<yyyy<2020

0<MM<=12

0<dd<=28

0<HH<=23

0<mm<=59

0<ss<=59

0<fff<=999

所有输入保证合法。

输出

输出一个整数，表示有多少顾客可以获取免单。

样例

输入样例 1

2019-01-01 00:00:00.001

2019-01-01 00:00:00.002

2019-01-01 00:00:00.003

输出样例 1

1

输入样例 2

2019-01-01 08:59:00.123

2019-01-01 08:59:00.123

2018-12-28 10:08:00.999

输出样例 2

3

输入样例 3

2019-01-01 00:00:00.004

2019-01-01 00:00:00.004

2019-01-01 00:00:01.006

2019-01-01 00:00:01.006

2019-01-01 00:00:01.005

输出样例 3

3

提示

样例 1 中，三个订单都是同一秒内下单，只有第一个订单最早下单，可以免单。

样例 2 中，前两个订单是同一秒内同一时刻（也是最早）下单，都可免单，第三个订单是当前秒内唯一一个订单（也是最早），也可免单。

样例 3 中，前两个订单是同一秒内同一时刻（也是最早）下单，第三第四个订单不是当前秒内最早下单，不可免单，第五个订单可以免单。

解题思路：

把订单时间字符串（2019-01-0100:00:00.001）按小数点分为第一部分和第二部分，第一部分是整数部分的字符串，第二部分是小数部分的字符串。

建立一个 map，以整数部分为 key，小数部分和第一个下单的人的个数为 value，记录下每秒钟第一个下单的人。

统计 map 中所有的每秒第一个下单的人数。

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

struct FirstOrder
{
    int microSecond;
    int count;
};

string GetFirstPartTime(string timeStr)
{
    size_t n=timeStr.find_first_of('.');
    string firstPart(timeStr.c_str(),timeStr.c_str()+n);
    return firstPart;
}

int GetSecondPartTime(string timeStr)
{
    size_t n=timeStr.find_first_of('.');
    n++;
    return atoi(timeStr.c_str()+n);
}

int main(int argc,char** argv)
{
    string line;
    map<string,FirstOrder> staticsMap;
    //记录每秒钟第一个下单的人数
    while(getline(cin,line))
    {
        string s=GetFirstPartTime(line);
        int ms=GetSecondPartTime(line);
        map<string,FirstOrder>::iterator it=staticsMap.find(s);
        if(it!=staticsMap.end())//在map中已经存在，需要更新
        {
            if(ms<it->second.microSecond)
            {
                it->second.microSecond=ms;
                it->second.count=1;
            }
            else if(ms==it->second.microSecond)
            {
                it->second.count++;
            }
        }
        else//不存在，需要插入
        {
            FirstOrder order;
            order.microSecond=ms;
```

```
        order.count=1;
        staticsMap.insert(make_pair(s,order));
    }
}

//统计所有第一个下单的人数
size_t freeOrderNum=0;
map<string,FirstOrder>::iterator it=staticsMap.begin();
while(it!=staticsMap.end())
{
    freeOrderNum+=it->second.count;
    it++;
}
cout<<freeOrderNum<<endl;
}
```


11、【股票最大收益】

题目：假设知道某段连续时间内股票价格，计算通过买入卖出可获得的最大收益。

输入一个大小为 n 的数 $price(p_1, p_2, p_3, p_4, \dots, p_n)$, p_i 是第 i 天的股票价格。

p_i 的格式为股票价格(非负整型)加上货币单位 Y 或者 S , 其中 Y 代表人民币, S 代表美元, 这里规定 1 美元可以兑换 7 人民币。

p_i 样例 1: 123Y 代表 123 元人民币

p_i 样例 2: 123S 代表 123 元美元, 可兑换 861 人民币

假设你可以在任何一天买入或者卖出股票, 也可以选择放弃交易, 请计其在交易周期 n 天内你能获得的最大收益(以人民币计算)。

输入描述:

输入一个包含交易周期内各天股票价格的字符串, 以空格分隔。不考虑输入异常情况。

输出描述:

输出一个整型数代表在交易周期 n 天内你能获得的最大收益, n 不能超过 10000

备注: 股票价格只会用 Y 人民币或 S 美元进行输入, 不考虑其他情况。

示例 1

输入:

2Y 3S 4S 6Y 8S

输出:

76

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int MaxProfit(vector<int> prices);
int Split(string input, vector<string>& output, string pattern);
int main(int argc, char** argv)
{
    string inputstring;
    while (1)
    {
        getline(cin, inputstring);
        if (!inputstring.empty())
        {
            vector<string> prices_string;
            Split(inputstring, prices_string, " ");
            vector<int> prices_int;
            for (int i = 0; i < prices_string.size(); i++)
            {
                if (prices_string[i][prices_string[i].size() - 1] == 'Y')
                {
                    prices_int.push_back(atoi((prices_string[i].substr(0, prices_string[i].size() - 1)).c_str()));
                }
                else
                {
                    prices_int.push_back(7 * atoi((prices_string[i].substr(0, prices_string[i].size() - 1)).c_str()));
                }
            }
            cout << MaxProfit(prices_int) << endl;
        }
    }
    return 0;
}

int MaxProfit(vector<int> prices)
{
    int len = prices.size();
    vector<int> change(len, 0);
    int maxProfit = 0;
    for (int i = 1; i < len; i++)
    {
        change[i] = prices[i] - prices[i - 1];
        if (change[i] > 0)
        {
            maxProfit += change[i];
        }
    }
    return maxProfit;
}

int Split(string input, vector<string>& output, string pattern)
{
    string::size_type pos;
    input += pattern; //扩展字符串以方便操作
    for (int i = 0; i < input.size(); i++)
    {
        pos = input.find(pattern, i);
        if (pos < input.size())
        {
            string s = input.substr(i, pos - i);
            if ((s != " ") && (!s.empty()))
            {
                output.push_back(s);
            }
            i = pos + pattern.size() - 1;
        }
    }
    return 0;
}
```

12、【独木桥】

题目描述

在河上有一座独木桥，一只青蛙想沿着独木桥从河的一侧跳到另一侧。在桥上有一些石子，青蛙很讨厌踩在这些石子上。由于桥的长度和青蛙一次跳过的距离都是正整数，我们可以把独木桥上青蛙可能到达的点看成数轴上的一串整点：0，1，……，L（其中L是桥的长度）。坐标为0的点表示桥的起点，坐标为L的点表示桥的终点。青蛙从桥的起点开始，不停的向终点方向跳跃。一次跳跃的距离是S到T之间的任意正整数（包括S,T）。当青蛙跳到或跳过坐标为L的点时，就算青蛙已经跳出了独木桥。

题目给出独木桥的长度L，青蛙跳跃的距离范围S,T，桥上石子的位置。你的任务是确定青蛙要想过河，最少需要踩到的石子数

输入输出格式

输入格式：

输入文件 river.in 的第一行有一个正整数L（ $1 \leq L \leq 10^9$ ），表示独木桥的长度。第二行有三个正整数S，T，M，分别表示青蛙一次跳跃的最小距离，最大距离，及桥上石子的个数，其中 $1 \leq S \leq T \leq 10$ ， $1 \leq M \leq 100$ 。第三行有M个不同的正整数分别表示这M个石子在数轴上的位置（数据保证桥的起点和终点处没有石子）。所有相邻的整数之间用一个空格隔开。

输出格式：

输出文件 river.out 只包括一个整数，表示青蛙过河最少需要踩到的石子数。

示例 1

输入：

```
10
2 3 5
2 3 5 6 7
```

输出：

```
2
```

说明

对于 30% 的数据， $L \leq 10000$ ；

对于全部的数据， $L \leq 10^9$ 。

解题思路

和前几天写的吃药问题差不多，l 的范围太大，无法作为数组下标，所以先离散化，再 DP。两点间的距离 d 大于 t 时，一定可以由 d%t 跳过来，所以最多只需要 $t+d\%t$ 种距离的状态就可以表示这两个石子之间的任意距离关系。这样就把题目中的 10^9 压缩成了 $2*t*m$ 最多不超过 2000，然后就可以放心大胆地用 DP 了。不过要注意题目中的“当青蛙跳到或跳过坐标为L的点时，就算青蛙已经跳出了独木桥”，所以 DP 的终点是一个范围而非确切的一个点，最后还要在这个范围内取最小值。

```
#include <iostream>
#include <string.h>
#include <algorithm>
using namespace std;
int dp[1000010];

int main(int argc, char** argv)
{
    int i, l, n, s, t, d, j, m;
    while(1)
    {
        cin >> l >> s >> t >> n;
        memset(dp, 0, sizeof(dp));
        for(i = 0; i < n; i++)
        {
            cin >> d;
            dp[d] = 1;
        }
        for(i = 1; i >= 0; i--)
        {
            m = 110;
            for(j = s; j <= t; j++)
            {
                m = min(dp[i+j], m);
            }
            dp[i] += m;
        }
        cout << dp[0] << endl;
    }
    return 0;
}
```

13、【多道批处理调度】

题目描述：某多处理器多道批处理系统一次允许将所有作业调入内存，且能并行执行，其并行数等于处理机个数。该系统采用 SJF 的调度方式（最短作业优先，系统在调度时，总是优先调度执行处理时间最短的作业）。

现给定处理器个数 m ，作业数 n ，每个作业的处理时间分别为 t_1, t_2, \dots, t_n 。

当 $n > m$ 时，首先处理时间短的 m 个作业进入处理器处理，其他的进入等待，当某个作业处理完成时，依次从等待队列中取处理时间最短的作业进入处理。

求系统处理完所有作业的耗时为多少？

注：不考虑作业切换的消耗。

输入描述：

输入 2 行，第一行为 2 个整数（采用空格分隔），分别表示处理器个数 m 和作业数 n ；第二行输入 n 个整数（采用空格分隔），表示每个作业的处理时长 t_1, t_2, \dots, t_n 。

$0 < m, n \leq 100, 0 < t_1, t_2, \dots, t_n \leq 100$ 。

输出描述：

输出处理总时长

示例 1：

输入

3 5

8 4 3 1 10

输出

13

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

int main(int argc, char** argv)
{
    int m, n;

    while (cin >> m >> n)
    {
        int input;
        vector<int> time;
        for (int i = 0; i < n; i++)
        {
            cin >> input;
            time.push_back(input);
        }

        sort(time.begin(), time.end());

        int length = time.size();
        int index = length % m - 1;
        int result = 0;
        for (int i = index; i < length; i = i + m)
        {
            result += time[i];
        }

        cout << result << endl;
    }

    return 0;
}
```

14、【数组的转换】

题目描述:

给定一个数组（列表）

示例: [1,2,3,4]

如上一个数组，转换之后为:

[4,3,2,1]

即元素倒着排序生成一个新的数组（列表）

输入描述:

为了简化输入，数据按照字符串形式输入，每个元素中间空格分离

输出描述:

输出倒叙之后的数据

示例 1

输入:

1234

输出:

4321

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

int main(int argc, char** argv)
{
    string input;
    getline(cin, input);
    reverse(input.begin(), input.end());
    cout<<input<<endl;
    return 0;
}
```

15、【藏宝图】

题目描述:

牛牛拿到了一个藏宝图，顺着藏宝图的指示，牛牛发现了一个藏宝盒，藏宝盒上有一个机关，机关每次会显示两个字符串 s 和 t ，根据古老的传说，牛牛需要每次都回答 t 是否是 s 的子序列。注意，子序列不要求在原字符串中是连续的，例如串 abc ，它的子序列就有{空串, a,b,c,ab,bc,abc }8 种。

输入描述:

每个输入包含一个测试用例。每个测试用例包含两行长度不超过 10 的不包含空格的可见 ASCII 字符串。

输出描述:

输出一行 “Yes” 或者 “No” 表示结果。

示例 1

输入

x.nowcoder.com

ooo

输出

Yes

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

int main(int argc, char** argv)
{
    string input_string;
    while(cin>>input_string)
    {
        string input_substring;
        cin>>input_substring;
        int a=0,b=0;
        while(a<input_string.Length())
        {
            if(input_string[a++]==input_substring[b])
            {
                b++;
            }
        }
        if(b==input_substring.Length())
        {
            cout<<"Yes"<<endl;
        }
        else
        {
            cout<<"No"<<endl;
        }
    }
    return 0;
}
```

16、【数字字符串组合倒序】

题目描述：对数字，字符，数字串，字符串，以及数字与字符串组合进行倒序排列。

字符范围：由 a 到 z，A 到 Z，数字范围：由 0 到 9

符号”的定义

(1) 做为连接符使用时作为字符串的一部分，例如 “20-years” 作为一个整体字符串呈现；

(2) 连续出现 2 个’及以上时视为字符串间隔符，如 “out - standing”中的“-”视为间隔符，是 2 个独立整体字符串“out”和“standing”；

除了 1，2 里面定义的字符以外其他的所有字符，都是非法字符，作为字符串的间隔符处理，倒序后间隔符作为空格处理；

要求倒排后的单词间隔符以一个空格表示；如果有多个间隔符时，倒排转换后也只允许出现一个空格间隔符；

输入示例

laman20-yearsout-standing@*-stu-dent

输出示例

dentstustandingout20-yearsanaml

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int IsValid(char* p)
{
    if (*p >= 'a' && *p <= 'z' || *p >= 'A' && *p <= 'Z' || *p >= '0' && *p <= '9')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int main(int argc, char** argv)
{
    char str[250]; // char * str 会有问题：从标准输入中读取字符串，要声明为字符数组，不能是一个指向字符的指针
    char* p = str;
    char Char[100];
    char CharArray[100][20] = { '\0' };
    char(*pc)[20];
    pc = CharArray;
    gets_s(str); // scanf("%s", str); scanf 遇到空格字符会停止; gets 可以读取一行，遇到换行符停止
    // I am an 20-years out--standing @ *-stu- dent
    int len = strlen(str);
    for (int i = 0; i < len; i++)
    {
        if (IsValid(p))
        {
            Char[i] = *p;
        }
        else if (*p == '-' && IsValid(p - 1) && IsValid(p + 1))
        {
            Char[i] = *p;
        }
        else
        {
            Char[i] = ' ';
        }
        p++;
    }
    Char[len] = '\0';
    int i = 0;
    int j = 0; // 行的索引，有几个单词
    int k = 0; // 列的索引，每个单词有几个字符
    while (Char[i] != '\0')
    {
        if (IsValid(Char + i) || Char[i] == '-')
        {
            (*(pc + j) + k) = Char[i];
            k++;
        }
        else
        {
            {
                j++;
                k = 0;
            }
            i++;
        }
    }
    for (int i = j; i >= 0; i--)
    {
        if (*(pc + i) != '\0')
        {
            printf("%s ", *(pc + i));
        }
    }
    printf("\n");
    system("pause");
    return 0;
}
```

17、【在字符串中找出连续最长的数字串(含“+-”号)】

题目描述：请在—个字符串中找出连续最长的数字串，并返回这个数字串。

如果存在长度相同的连续数字串，返回最后—个。如果没有符合条件的字符串，返回空字符串“”。

注意：

数字串可以由数字“0-9”、小数点“.”、正负号“±”组成，长度包括组成数字串的所有符号。“.”、“±”仅能出现—次，“.”的两边必须是数字，“±”仅能出现在开头且其后必须要有数字。

长度不定，可能含有空格。

例子：1234567890abcd9.+12345.678.9ed

输出：+12345.678

```
#include <regex>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

string GetMaxLengthDigitalString(const string& inputString)
{
    regex numRegex("[+-]?[0-9]+(\\.[0-9]+)?");
    smatch result;
    string outputString;
    string::const_iterator iter=inputString.begin();
    while(regex_search(iter,inputString.end(),result,numRegex))
    {
        string tmpString=result[0];
        if(tmpString.size()>outputString.size())
        {
            tmpString.swap(outputString);
        }

        iter=result[0].first;
        while(++iter!=result[0].second&&(*iter)!='.')
        {}
    }
    return outputString;
}

int main (int argc,char** argv)
{
    string input_string;
    while(getline(cin,input_string))
    {
        cout<<GetMaxLengthDigitalString(input_string)<<endl;
    }
    return 0;
}
```

18、【在字符串中找出连续最长的数字串】

题目描述

请一个在字符串中找出连续最长的数字串，并返回这个字符串；如果存在长度相同的连续数字串，返回最后一个连续数字串；

注意：数字串是由数字和“.”组成的（长度包括“.”在内），“.”两边必须是数字，比如：

数字串“1234”的长度就小于数字串“00055”，数字串“1234.56789”的长度大于数字串“123456789”

输入描述：

字符串输入为 ASCII 编码，长度不定，可能含有空格，请读取完整一行数据作为输入

输出描述：

如果没有符合条件的数字串，返回空字符串“”

示例 1

输入：

abcd123.4567.890.123

abcd12345ed125ss123058789

abcd12345ss54761

输出

4567.890

123058789

54761

```
#include <regex>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

string GetMaxLengthDigitalString(const string& inputString)
{
    regex numRegex("[+-]?[0-9]+(\\.[0-9]+)?");
    smatch result;
    string outputString;
    string::const_iterator iter=inputString.begin();
    while(regex_search(iter,inputString.end(),result,numRegex))
    {
        string tmpString=result[0];
        if(tmpString.size()>outputString.size())
        {
            tmpString.swap(outputString);
        }

        iter=result[0].first;
        while(++iter!=result[0].second&&(*iter)!='.')
        {}
    }
    return outputString;
}

int main(int argc,char** argv)
{
    string input_string;
    while(getline(cin,input_string))
    {
        cout<<GetMaxLengthDigitalString(input_string)<<endl;
    }
    return 0;
}
```


19、【删除字符串中字符最少字符】

题目描述：删除字符串中出现次数最少的字符，如果有多个字符出现次数一样，则都删除。

输入描述：

输入 abcd

字符串中只包含小写英文字母。

输出描述：

dd

示例 1:

输入

abcd

输出

dd

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

int main(int argc, char** argv)
{
    string str;
    while(cin>>str)
    {
        int cnt=INT_MAX;
        map<char,int> mp;
        bool flag=true;
        for(auto i:str)
        {
            mp[i]++;
        }
        for(auto i:mp)
        {
            if(cnt>i.second)
            {
                cnt=i.second;
            }
        }
        for(auto i:str)
        {
            if(mp[i]==cnt)
            {
                continue;
            }
            cout<<i;
            flag=false;
        }
        if(flag)
        {
            cout<<"empty";
        }
        cout<<endl;
    }
    return 0;
}
```

20、【查找众数及中位数】

题目描述:

- 1.众数是指一组数据中出现次数最多的那个数，众数可以是多个
- 2.中位数是指把一组数据从小到大排列，最中间的那个数，如果这组数据的个数是奇数，那最中间那个就是中位数，如果这组数据的个数为偶数，那就把中间的两个数之和除以 2，所得的结果就是中位数
- 3.查找整型数组中元素的众数并组成一个新的数组，求新数组的中位数

输入描述:

输入一个一维整型数组，数组大小取值范围 $0 < N < 1000$ ，数组中每个元素取值范围 $0 < E < 1000$

输出描述:

输出众数组成的新数组的中位数

示例 1:

输入

10 11 21 19 21 17 21 16 21 18 15

输出

21

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    unordered_map<int, int> unmap;
    int input;
    vector<int> v;
    vector<int> new_v;
    int cnt=0;
    while(cin>>input)
    {
        v.push_back(input);
    }
    for(auto i:v)
    {
        unmap[i]++;
    }
    for(auto i:unmap)
    {
        if(cnt<i.second)
        {
            cnt=i.second;
        }
    }
    for(auto i:unmap)
    {
        if(cnt==i.second)
        {
            new_v.push_back(i.first);
        }
    }
    sort(new_v.begin(), new_v.end());
    if(new_v.size()%2==0&&new_v.size())
    {
        cout<<(new_v[new_v.size()/2]+new_v[new_v.size()/2-1])/2<<endl;
    }
    else if(new_v.size()%2!=0)
    {
        cout<<new_v[new_v.size()/2]<<endl;
    }
    else
    {
        cout<<endl;
    }
    return 0;
}
```

21、【高效的任务规划】

题目描述:

你有 n 台机器编号为 $1 \sim n$ ，每台都需要完成完成一项工作，机器经过配置后都能完成独立完成一项工作。假设第 i 台机器你需要花 B 分钟进行设置，然后开始运行， J 分钟后完成任务。现在，你需要选择布置工作的顺序，使得用最短的时间完成所有工作。注意，不能同时对两台进行配置，但配置完成的机器们可以同时执行他们各自的工作。

输入描述:

第一行输入代表总共有 M 组任务数据 ($1 \leq M \leq 10$)。

每组数第一行为一个整数指定机器的数量 N ($0 < N \leq 1000$)。随后的 N 行每行两个整数，第一个表示 B ($0 \leq B \leq 10000$)，第二个表示 J ($0 \leq J \leq 10000$)。

每组数据连续输入，不会用空行分隔。各组任务单独计时。

输出描述:

对于每组任务，输出最短完成时间，且每组的结果独占一行。例如，两组任务就应该有两行输出。

示例 1:

输入

1

1

22

输出

4

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

bool cmp(vector<int>& x,vector<int>& y)
{
    if(x[1]>y[1])
    {
        return true;
    }
    return false;
}

int main(int argc,char** argv)
{
    int M;
    cin>>M;
    while(M--)
    {
        int N;
        cin>>N;
        int ret=0;
        int time[1001]={};
        int last_time=0;
        vector<vector<int>>> doWork(N,vector<int>(2,0));
        for(int i=0;i<N;++i)
        {
            int B,J;
            cin>>B>>J;
            doWork[i][0]=B;
            doWork[i][1]=J;
        }
        sort(doWork.begin(),doWork.end(),cmp);
        for(int i=0;i<N;++i)
        {
            time[i]=last_time+doWork[i][0]+doWork[i][1];
            last_time+=doWork[i][0];
            if(ret<time[i])
            {
                ret=time[i];
            }
        }
        cout<<ret<<endl;
    }
    return 0;
}
```

22、【字符串筛选排序】

题目描述：输入一个由 n 个大小写字母组成的字符串，按照 Ascii 码值从小到大的排序规则，查找字符串中第 k 个最小 ascii 码值的字母（k>=1），输出该字母所在字符串的位置索引(字符串的第一个字符位置索引为 0)。

k 如果大于字符串长度，则输出最大 ascii 值的字母所在字符串的位置索引，如果有重复的字母，则输出字母的最小位置索引。

输入描述：

第一行输入一个由大小写字母组成的字符串

第二行输入 k，k 必须大于 0，k 可以大于输入字符串的长度

输出描述：

输出字符串中第 k 个最小 ascii 码值的字母所在字符串的位置索引。k 如果大于字符串长度，则输出最大 ascii 值的字母所在字符串的位置索引，如果第 k 个最小 ascii 码值的字母存在重复，则输出该字母的最小位置索引。

示例 1:

输入

AbCdEfG

3

输出

5

```
#include<iostream>
#include<unordered_map>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;

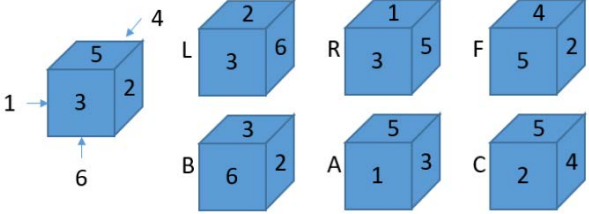
int main(int argc, char** argv)
{
    string str;
    int k;
    while(getline(cin, str))
    {
        cin >> k;
        if(k < str.size())
        {
            vector<char> ve1; // 存储字符
            for(int i=0; i<str.size(); i++)
            {
                ve1.push_back(str[i]);
            }
            sort(ve1.begin(), ve1.end());
            for(int i=0; i<str.size(); i++)
            {
                if(ve1[k-1] == str[i])
                {
                    cout << i;
                    break;
                }
            }
        }
        else if(k > str.size())
        {
            vector<char> ve2; // 存储字符
            for(int i=0; i<str.size(); i++)
            {
                ve2.push_back(str[i]);
            }
            sort(ve2.begin(), ve2.end());
            for(int i=0; i<str.size(); i++)
            {
                if(ve2[str.size()-1] == str[i])
                {
                    cout << i;
                    break;
                }
            }
        }
        cout << endl;
    }
    return 0;
}
```

23、【转骰子】

题目描述：骰子是一个立方体，每个面一个数字，初始为左 1，右 2，前 3（观察者方向），后 4，上 5，下 6，用 123456 表示这个状态，放置到平面上，可以向左翻转（用 L 表示

向左翻转 1 次），可以向右翻转（用 R 表示向右翻转 1 次），可以向前翻转（用 F 表示向前翻转 1 次），可以向后翻转（用 B 表示向后翻转 1 次），可以逆时针旋转（用 A 表示逆时针旋转 90 度），可以顺时针旋转（用 C 表示顺时针旋转 90 度），现从 123456 这个初始状态开始，根据输入的动作序列，计算得到最终的状态。

骰子的初始状态和初始状态转动后的状态如图所示



输入描述：

输入一行，为只包含 LRFBAC 的字母序列，最大长度 50，字母可重复

输出描述：

输出最终状态

示例 1:

输入

LR

输出

123456

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    string str;
    while(getline(cin, str))
    {
        vector<char> ve_num;
        string str_num="123456";
        for(int i=0;i<str_num.size();i++)
        {
            ve_num.push_back(str_num[i]);
        }
        for(int i=0;i<str.size();i++)
        {
            if(str[i]=='L')
            {
                swap(ve_num[0],ve_num[5]);
                swap(ve_num[0],ve_num[1]);
                swap(ve_num[0],ve_num[4]);
            }
            else if(str[i]=='R')
            {
                swap(ve_num[0],ve_num[5]);
                swap(ve_num[1],ve_num[4]);
                swap(ve_num[4],ve_num[5]);
            }
            else if(str[i]=='F')
            {
                swap(ve_num[4],ve_num[2]);
                swap(ve_num[3],ve_num[5]);
                swap(ve_num[4],ve_num[5]);
            }
            else if(str[i]=='B')
            {
                swap(ve_num[5],ve_num[2]);
                swap(ve_num[4],ve_num[3]);
                swap(ve_num[4],ve_num[5]);
            }
            else if(str[i]=='A')
            {
                swap(ve_num[0],ve_num[3]);
                swap(ve_num[1],ve_num[2]);
                swap(ve_num[2],ve_num[3]);
            }
        }
    }
}
```

```

        else if(str[i]=='C')
        {
            swap(ve_num[2],ve_num[0]);
            swap(ve_num[1],ve_num[3]);
            swap(ve_num[2],ve_num[3]);
        }
    }
    for(int i=0;i<ve_num.size();i++)
    {
        cout<<ve_num[i];
    }
    cout<<endl;
}
return 0;
}

```

24、【按身高和体重排队】

题目描述：某学校举行运动会，学生们按编号(1、2、3...n)进行标识，现需要按照身高由低到高排列，对身高相同的人，按体重由轻到重排列；对于身高体重都相同的人，维持原有的编号顺序关系。请输出排列后的学生编号。

输入描述：

两个序列，每个序列由 n 个正整数组成（ $0 < n \leq 100$ ）。第一个序列中的数值代表身高，第二个序列中的数值代表体重。

输出描述：

排列结果，每个数值都是原始序列中的学生编号，编号从 1 开始

示例 1

输入：

4

100100120130

40306050

输出：2134

```

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

typedef struct Student
{
    int number;
    int height;
    int weight;
}Student;

bool operator<(const Student& st1,const Student& st2)
{
    if(st1.height>st2.height)
    {
        return true;
    }
    else if((st1.height==st2.height)&&(st1.weight>st2.weight))
    {
        return true;
    }
    else
    {
        return false;
    }
}

void showStudent(const Student& st)
{
    cout<<st.number<<" ";
}

int main(int argc,char** argv)
{
    vector<Student> sts;
    unsigned n;
    Student st;

    cin>>n;
    for(unsigned i=0;i<n;++i)
    {
        st.number=i+1;
        cin>>st.height>>st.weight;
        sts.push_back(st);
    }
}

```

```

    }

    sort(sts.begin(), sts.end());
    for_each(sts.begin(), sts.end(), showStudent); // 注意这里修改成begin(), end()
    cout << endl;
    return 0;
}

```

25、【集五福】

题目描述:

集五福作为近年来大家喜闻乐见迎新春活动，集合爱国福、富强福、和谐福、友善福、敬业福即可分享超大红包以 0 和 1 组成的长度为 5 的字符串代表每个人所得到的福卡，每一位代表一种福卡，1 表示已经获得该福卡，单类型福卡不超过 1 张，随机抽取一个不超过 10 人团队，求该团队最多可以集齐多少套五福

输入描述:

输入类似 11010,00110,

由 0、1 组成的长度为 5 的字符串，

代表指定团队中每个人福卡获得情况

注意 1: 1 人也可以是一个团队注意 2: 多人之间的福卡以英文逗号隔开

输出描述:

输出该团队能凑齐多少套五福

示例 1

输入: 11001,11100

输出: 0

```

#include<iostream>
#include<string>
#include<vector>
using namespace std;

void CollectionWufu(string input_string, int& output_counts);
void Split(const string& src, const string& separator, vector<string>& dest);

int main(int argc, char** argv)
{
    string input_string;
    int wufu_counts = 0;
    while (cin >> input_string)
    {
        CollectionWufu(input_string, wufu_counts);
        cout << wufu_counts << endl;
    }

    return 0;
}

void CollectionWufu(string input_string, int& output_counts)
{
    output_counts = 0;
    vector<int> postions;
    vector<string> vector_string;

    Split(input_string, ",", vector_string);

    for (int i = 0; i < vector_string.size(); i++)
    {
        int pos = 0;
        bool flag = true;
        if (pos = vector_string[i].find("0", 0) != string::npos)
        {
            flag = false;
        }
        if (flag == true)
        {
            output_counts++;
        }
    }
}

void Split(const string& src, const string& separator, vector<string>& dest) // 字符串分割到string数组
{
    string str = src;
    string substring;
    string::size_type start = 0, index = 0;

```

```

dest.clear();
index = str.find_first_of(separator, start);
do
{
    if (index != string::npos)
    {
        substring = str.substr(start, index - start);
        dest.push_back(substring);
        start = index + separator.size();
        index = str.find(separator, start);

        if (start == string::npos)
        {
            break;
        }
    }
} while (index != string::npos);

//thelastpart
substring = str.substr(start);
dest.push_back(substring);
}

```

26、【求所需的快递员最少位数】

描述：有一快递配送线路 arr[], arr[i]=0 时表示无需配送，arr[i]=1 时表示需要配送。

条件：每个快递员只能配送连续的两个点，即 arr[i-1],arr[i],arr[i+1]

输出：输入 arr[]，输出最少需要快递员的数量

```

#include <iostream>
#include "vector"

using namespace std;

class Solution
{
public:
    int Postman(const vector<int>& arr)
    {
        vector<int> myVec = arr;
        int iCount = 0;
        if (arr.size() < 1 || arr.size() > 100000)
        {
            return 0;
        }
        if (arr.size() <= 3)
        {
            for (int i = 0; i < arr.size(); i++)
            {
                if (arr[i] == 1)
                {
                    iCount = 1;
                }
            }
            return iCount;
        }

        if (arr[0] == 0)
        {
            myVec.erase(myVec.begin());
        }
        else
        {
            bool bResult = arr[0] == 0 && arr[1] == 0 && arr[2] == 0;
            myVec.erase(myVec.begin(), myVec.begin() + 3);
            iCount++;
        }

        return iCount += Postman(myVec);
    }
};

int main(int argc, char** argv)
{
    Solution* so = new Solution();
    cout << so->Postman(vector<int> { 0, 0, 1, 0, 0, 1 }) << endl;           // 2
    cout << so->Postman(vector<int> { 1, 0, 1 }) << endl;                   // 1
    cout << so->Postman(vector<int> { 1, 0, 1, 0, 1, 0, 1 }) << endl;       // 2
    cout << so->Postman(vector<int> { 1, 1, 1, 1 }) << endl;                 // 2
    cout << so->Postman(vector<int> { 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0 }) << endl; // 3
    cout << so->Postman(vector<int> { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }) << endl; // 4
}

```



```

    cout << so->Postman(vector<int> { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }) << endl;    // 4
    cout << so->Postman(vector<int> { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }) << endl;    // 3
    cout << so->Postman(vector<int> { 0, 0, 0, 0, 1, 1, 1, 1, 0 }) << endl;        // 1
    free(so);
    return 0;
}

```

27、【德州扑克】

题目描述：五张牌，每张牌由牌大小和花色组成，牌大小 2~10、J、Q、K、A，牌花色为红桃、黑桃、梅花、方块四种花色之一。判断牌型：牌型 1，同花顺：同一花色的顺子，如红桃 2 红桃 3 红桃 4 红桃 5 红桃 6。牌型 2，四条：四张相同数字 + 单张，如红桃 A 黑桃 A 梅花 A 方块 A + 黑桃 K。牌型 3，葫芦：三张相同数字 + 一对，如红桃 5 黑桃 5 梅花 5 + 方块 9 梅花 9。牌型 4，同花：同一花色，如方块 3 方块 7 方块 10 方块 J 方块 Q。牌型 5，顺子：花色不一样的顺子，如红桃 2 黑桃 3 红桃 4 红桃 5 方块 6。牌型 6，三条：三张相同 + 两张单。牌型 7，其他。说明：1）五张牌里不会出现牌大小和花色完全相同的牌。2）编号小的牌型较大，如同花顺比四条大，依次类推。3）包含 A 的合法的顺子只有 10 J Q K A 和 A 2 3 4 5，类似 K A 2 3 4 的序列不认为是顺子。

输入描述：

输入由 5 行组成，每行为一张牌大小和花色，牌大小为 2~10、J、Q、K、A，花色分别用字符 H、S、C、D 表示红桃、黑桃、梅花、方块。

输出描述：

输出牌型序号，5 张牌符合多种牌型时，取最大的牌型序号输出。

示例 1

输入：

4 H

5 S

6 C

7 D

8 D

输出：5

```

#include <iostream>
#include <map>
#include <list>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
enum CardType
{
    CARD_TYPE_UNKNOWN,
    CARD_TYPE_H, //红桃
    CARD_TYPE_S, //黑桃
    CARD_TYPE_C, //梅花
    CARD_TYPE_D, //方块
};
enum CardResult
{
    CARD_RESULT_TONGHUASHUN = 1,
    CARD_RESULT_SITIAO,
    CARD_RESULT_HULU,
    CARD_RESULT_TONGHUA,
    CARD_RESULT_SHUNZI,
    CARD_RESULT_SANTIAO,
    CARD_RESULT_OTHER,
};
struct CardInfo
{
    CardType eType;
    char byNum;
};
int split(string& str, char& a, char& b)
{
    if (str.Length() < 3)
    {
        return -1;
    }
    a = str[0];
    b = str[2];
    return 0;
}
class CardRule
{
public:
    CardRule();
    ~CardRule();
    CardResult Judge(vector<CardInfo>& vec);
    void Sort(vector<CardInfo>& vec);

    bool IsShunzi(vector<CardInfo>& vec);
}

```

```

    bool IsTonghua(vector<CardInfo>& vec);
    bool IsSitiao(vector<CardInfo>& vec);
    bool IsHulu(vector<CardInfo>& vec);
    bool IsSantiao(vector<CardInfo>& vec);
private:
};

CardRule::CardRule()
{
}

CardRule::~CardRule()
{
}

CardResult CardRule::Judge(vector<CardInfo>& vec)
{
    bool bTongHua = IsTonghua(vec);
    bool bShunzi = IsShunzi(vec);
    if (bTongHua && bShunzi)
    {
        return CARD_RESULT_TONGHUASHUN;
    }
    else if (IsSitiao(vec))
    {
        return CARD_RESULT_SITIAO;
    }
    else if (IsHulu(vec))
    {
        return CARD_RESULT_HULU;
    }
    else if (bTongHua)
    {
        return CARD_RESULT_TONGHUA;
    }
    else if (bShunzi)
    {
        return CARD_RESULT_SHUNZI;
    }
    else if (IsSantiao(vec))
    {
        return CARD_RESULT_SANTIAO;
    }
    else
    {
        return CARD_RESULT_OTHER;
    }
}

bool Compare(CardInfo lhs, CardInfo rhs)
{
    return lhs.byNum < rhs.byNum ? true : false;
}

bool Unique(CardInfo lhs, CardInfo rhs)
{
    return lhs.byNum == rhs.byNum ? false : true;
}

void CardRule::Sort(vector<CardInfo>& vec)
{
    sort(vec.begin(), vec.end(), Compare);
}

bool CardRule::IsShunzi(vector<CardInfo>& vec)
{
    for (vector<CardInfo>::iterator iter = vec.begin() + 1; iter != vec.end(); iter++)
    {
        if ((*iter).byNum - (*(iter - 1)).byNum != 1)
        {
            return false;
        }
    }

    return true;
}

bool CardRule::IsTonghua(vector<CardInfo>& vec)
{
    for (vector<CardInfo>::iterator iter = vec.begin() + 1; iter != vec.end(); iter++)
    {
        if ((*iter).eType != (*(vec.begin())).eType)
        {
            return false;
        }
    }
    return true;
}

bool CardRule::IsSitiao(vector<CardInfo>& vec)
{
    map<char, int> mapCount; //<数字, 个数>

```

```

for (vector<CardInfo>::iterator iter = vec.begin(); iter != vec.end(); iter++)
{
    if (mapCount.end() == mapCount.find((*iter).byNum))
    {
        mapCount[(*iter).byNum] = 1;
    }
    else
    {
        mapCount[(*iter).byNum]++;
    }
}
if (mapCount.size() == 2)
{
    for (map<char, int>::iterator iter = mapCount.begin(); iter != mapCount.end(); iter++)
    {
        if ((*iter).second == 4) return true;
    }
}
return false;
}
bool CardRule::IsHulu(vector<CardInfo>& vec)
{
    map<char, int> mapCount; //<数字, 个数>
    for (vector<CardInfo>::iterator iter = vec.begin(); iter != vec.end(); iter++)
    {
        if (mapCount.end() == mapCount.find((*iter).byNum))
        {
            mapCount[(*iter).byNum] = 1;
        }
        else
        {
            mapCount[(*iter).byNum]++;
        }
    }
    if (mapCount.size() == 2)
    {
        for (map<char, int>::iterator iter = mapCount.begin(); iter != mapCount.end(); iter++)
        {
            if ((*iter).second == 3 || (*iter).second == 2) return true;
        }
    }
    return false;
}
bool CardRule::IsSantiao(vector<CardInfo>& vec)
{
    map<char, int> mapCount; //<数字, 个数>
    for (vector<CardInfo>::iterator iter = vec.begin(); iter != vec.end(); iter++)
    {
        if (mapCount.end() == mapCount.find((*iter).byNum))
        {
            mapCount[(*iter).byNum] = 1;
        }
        else
        {
            mapCount[(*iter).byNum]++;
        }
    }
    if (mapCount.size() == 3)
    {
        for (map<char, int>::iterator iter = mapCount.begin(); iter != mapCount.end(); iter++)
        {
            if ((*iter).second == 3) return true;
        }
    }
    return false;
}
CardType CardConvert(char c)
{
    switch (c)
    {
        case 'H': return CARD_TYPE_H;
        case 'C': return CARD_TYPE_C;
        case 'S': return CARD_TYPE_S;
        case 'D': return CARD_TYPE_D;
        default:
            break;
    }
    return CARD_TYPE_UNKNOWN;
}
int main(int argc, char** argv)
{
    do
    {
        vector<CardInfo> vec;
        for (size_t i = 0; i < 5; i++)
        {

```

```

        CardInfo st;
        memset(&st, 0, sizeof(CardInfo));
        std::string str;
        getline(cin, str);
        char cCardType;
        split(str, st.byNum, cCardType);
        st.eType = CardConvert(cCardType);
        if (st.eType == CARD_TYPE_UNKNOWN)
        {
            cout << "error input" << endl;
            return -1;
        }
        vec.push_back(st);
    }
    CardRule cRule;
    cout << cRule.Judge(vec) << endl;
} while (true);

return 0;
}

```

28、【卡片组成最大数】

题目描述：小组中每位都有一张卡片，卡片上是 6 位内的正整数，将卡片连起来可以组成多种数字，计算组成的最大数字。

输入描述：

“#”号分割的多个正整数字符串，不需要考虑非数字异常情况，小组最多 25 个人

输出描述：

最大的数字字符串

示例 1

输入：22,221

输出：22221

```

#include <iostream>
#include <cmath>
#include <vector>
#include <string>
using namespace std;

bool Compare(int num1, int num2)
{
    int count1 = 0, count2 = 0;
    int nm1 = num1, nm2 = num2;
    while (nm1)
    {
        count1++;
        nm1 /= 10;
    }
    while (nm2)
    {
        count2++;
        nm2 /= 10;
    }
    //pow(a,b):即求得a的b次方，这里是将两个数连接起来进行下一步的比较。
    int a = num1 * pow(10.0, count2) + num2;
    int b = num2 * pow(10.0, count1) + num1;

    return a > b ? true : false;
}

int Split(string input, vector<int>& output, string pattern)
{
    string::size_type pos;
    input += pattern; //扩展字符串以方便操作
    for (int i = 0; i < input.size(); i++)
    {
        pos = input.find(pattern, i); if (pos < input.size())
        {
            string temp = input.substr(i, pos - i); if ((temp != " ") && (!temp.empty()))
            {
                output.push_back(atoi(temp.c_str()));
            }
            i = pos + pattern.size() - 1;
        }
    }
    return 0;
}

int main(int argc, char** argv)
{

```

```

string input_string;
while (getline(cin, input_string))
{
    vector<int> input;
    Split(input_string, input, ",");

    int tmp;
    for (int i = 0; i <= input.size() - 1; ++i)
    {
        for (int j = 0; j < input.size() - i - 1; j++)
        {
            //如果a>b需要交换, 否则, 不交换
            if (Compare(input[j], input[j + 1]))
            {
                tmp = input[j];
                input[j] = input[j + 1];
                input[j + 1] = tmp;
            }
        }
    }
    for (int i = input.size() - 1; i >= 0; --i)
    {
        cout << input[i];
    }
    cout << endl;
}
return 0;
}

```

29、【字符串统计】

题目描述：给定两个字符集合，一个为全量字符集，一个为已占用字符集。已占用的字符集中的字符不能再使用，要求输出剩余可用字符集。

输入描述：

1、输入为一个字符串，一定包含@符号。@前的为全量字符集，@后的字为已占用字符集。 2、已占用字符集中的字符一定是全量字符集中的字符。字符集中的字符跟字符之间使用英文逗号分隔。 3、每个字符都表示为字符加数字的形式，用英文冒号分隔，比如 a:1，表示 1 个 a 字符。 4、字符只考虑英文字母，区分大小写，数字只考虑正整形，数量不超过 100。 5、如果一个字符都没被占用，@标识仍然存在，例如 a:3,b:5,c:2@

输出描述：

输出可用字符集，不同的输出字符集之间回车换行。 注意，输出的字符顺序要跟输入一致。不能输出 b:3,a:2,c:2 如果某个字符已被全被占用，不需要再输出。

示例 1

输入: a:3,b:5,c:2@a:1,b:2

输出: a:2,b:3,c:2

```

#include <iostream>
#include <string>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <assert.h>
using namespace std;
//分割字符串函数
vector<string> split(const string& str, const string& delim)
{
    vector<string> res;
    if (" " == str)
        return res;
    char* strs = new char[str.Length() + 1];
    strcpy(strs, str.c_str());
    char* d = new char[delim.Length() + 1];
    strcpy(d, delim.c_str());
    char* p = strtok(strs, d);
    while (p)
    {
        string s = p;
        res.push_back(s);
        p = strtok(NULL, d);
    }
    return res;
}
void StringCount(vector<string> str, vector<int>& counts)
{
    for (int i = 0; i < str.size(); i++)
    {
        string num = str[i].substr(2, str[i].length() - 2);
        char* nums = new char[num.Length() + 1];
        strcpy(nums, num.c_str());
        int number = atoi(nums);
        counts[str[i][0]] = number;
    }
}

```

```

string CalResString2(string all, string used)
{
    string res = "";
    vector<string> allStr = split(all, ",");
    vector<string> usedStr = split(used, ",");
    vector<int> allCounts(256, 0);
    vector<int> usedCounts(256, 0);
    vector<int> resCounts(256, 0);
    StringCount(allStr, allCounts);
    StringCount(usedStr, usedCounts);
    for (int i = 0; i < 256; i++)
    {
        resCounts[i] = allCounts[i] - usedCounts[i];
        assert(resCounts[i] >= 0);
        if (resCounts[i] > 0)
        {
            char ch = i;
            res = res + ch + ":" + to_string(resCounts[i]) + ",";
        }
    }
    res.erase(res.Length() - 1); //最后一个逗号去掉
    return res;
}

int main(int argc, char** argv)
{
    string str;
    while (cin >> str)
    {
        string strAll = "";
        string used = "";
        if (str.find("@") != string::npos)
        {
            int index = str.find("@");
            strAll = str.substr(0, index);
            used = str.substr(index + 1, str.Length() - index);
            string res = CalResString2(strAll, used);
            cout << res << endl;
        }
    }
    return 0;
}

```

30、【区间交集】

题目描述：给定一组闭区间，其中部分区间存在交集。任意两个给定区间的交集，称为公共区间（如：[1,2],[2,3]的公共区间为[2,2]，[3,5],[3,6]的公共区间为 [3,5]）。公共区间之间若存在交集，则需要合并（如：[1,3],[3,5]区间存在交集[3,3]，须合并为[1,5]）。按升序排列输出合并后的区间列表。

输入描述：

一组区间列表，

区间数为 N： 0 <= N <= 1000；

区间元素为 X： -10000 <= X <= 10000。

输出描述：升序排列的合并后区间列表

备注：

1、 区间元素均为数字，不考虑字母、符号等异常输入。 2、单个区间认定为无公共区间。

示例 1

输入：

0 3

1 3

3 5

3 6

输出：

15

```

#include <bits/stdc++.h>
using namespace std;
int main(int argc, char** argv)
{
    int n, ans = 0; // ans存储最终结果
    scanf("%d", &n);
    vector<pair<int, int>> v1(n), v2(n); // 分别存储两个区间
    for (int i = 0; i < n; ++i)
    {
        scanf("%d%d", &v1[i].first, &v1[i].second);
    }
    for (int i = 0; i < n; ++i)

```

```

{
    scanf("%d%d", &v2[i].first, &v2[i].second);
}
for (auto p1 : v1)
{
    for (auto p2 : v2)
    {
        if (p1.first <= p2.second && p1.second >= p2.first)
        {
            ans += min(p1.second, p2.second) - max(p1.first, p2.first); //加上重叠区间
        }
    }
}
printf("%d", ans);
return 0;
}

```

31、【仿 LISP 运算】

题目描述: LISP 语言唯一的语法就是括号要配对。形如 (OP P1 P2 ...)，括号内元素由单个空格分割。其中第一个元素 OP 为操作符，后续元素均为其参数，参数个数取决于操作符类型 注意：参数 P1,P2 也有可能是另外一个嵌套的 (OP P1 P2 ...) 当前 OP 类型为 add/sub/mul/div（全小写），分别代表整数的加减乘除法 简单起见，所有 OP 参数个数均为 2 举例:- 输入: (mul 3 -7) 输出: -21- 输入: (add 1 2) 输出: 3- 输入: (sub (mul 2 4) (div 9 3)) 输出: 5- 输入: (div 1 0) 输出: error 题目涉及数字均为整数，可能为负；不考虑 32 位溢出翻转，计算过程中也不会发生 32 位溢出翻转 除零错误时，输出 "error"，除法遇除不尽，向下取整，即 $3/2 = 1$

输入描述:

输入为长度不超过 512 的字符串，用例保证了无语法错误

输出描述:

输出计算结果或者 "error"

示例 1

输入: (div 12 (sub 45 45))

输出:

error

```

#include<iostream>
#include <stack>
#include<string>
#include<vector>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    vector<string> st;
    stack<int> res;
    string str;
    getline(cin, str);
    bool flag = true;
    int N = str.size();
    for (int i = N - 1; i >= 0;)
    {
        if (str[i] == ')') || str[i] == ' ' || str[i] == '(')
            i = i - 1;
        else if (str[i] >= 'a' && str[i] <= 'z')
        {
            string op = str.substr(i - 2, 3);
            st.push_back(op);
            i -= 3;
        }
        else
        {
            int spaceIndex = str.rfind(' ', i);
            if ((spaceIndex < 0) || (str[spaceIndex + 1] <= 'z' && str[spaceIndex + 1] >= 'a'))
            {
                spaceIndex = str.rfind('(', i);
            }
            string num = str.substr(spaceIndex + 1, i - spaceIndex);
            st.push_back(num);
            i = spaceIndex;
        }
    }
    int n = st.size();
    for (int j = 0; j < n; ++j)
    {
        string s = st[j];
        if (s == "add" || s == "sub" || s == "mul" || s == "div")
        {
            if (res.size() < 2)
            {

```

```

        return 0;
    }
    int num2 = res.top(); res.pop();
    int num1 = res.top(); res.pop();
    int result = 0;

    if (s == "add")
    {
        result = num1 + num2;
    }
    else if (s == "sub")
    {
        result = num2 - num1;
    }
    else if (s == "mul")
    {
        result = num1 * num2;
    }
    else if (s == "div")
    {
        if (num1 == 0)
        {
            flag = false;
        }
        else
        {
            result = num2 / num1;
        }
    }

    res.push(result);
}
else
{
    res.push(atoi(s.c_str()));
}
}
if (!flag)
{
    cout << "error" << endl;
}
else
{
    cout << res.top() << endl;
}
}
}

```

32、【Words 重量】

题目描述：每个句子由多个单词组成，句子中的每个单词的长度都可能不一样， 我们假设每个单词的长度 LW 为该单词的重量，请给出整个句子的平均重量 V 。

输入描述：

输入只有一行，包含一个字符串 S (长度不会超过 100)，代表整个句子，句子中只包含大小写的英文字母（不包含标点符号），每个单词之间有一个空格。

输出描述：输出 S 的平均重量 V （每个单词的长度之和/单词个数）(四舍五入保留两位小数)。

示例 1

输入：Who Love Mate

输出：3.67

```

#include <iostream>
#include <string>

using namespace std;

bool IsSeparator(char ch); //判断一个字符是不是分隔符（空格、逗号，句号）
int CharCount(string str); //计算句子中字母数的函数
int WordCount(string sentence); //计算句子中单词个数的函数

int main(int argc, char** argv)
{
    string sentence;
    getline(cin, sentence);
    int letterNum = CharCount(sentence);
    int wordNum = WordCount(sentence);
    cout << letterNum * 1.0 / wordNum;
    return 0;
}

int WordCount(string sentence)
{
    int wordNum = 0;
    int len = sentence.Length();
    int i = 0;
    while (true)

```



```

{
    //如果是分隔符，就跳过
    if (IsSeparator(sentence[i]))
    {
        while (i <= len - 1 && IsSeparator(sentence[i]))
        {
            i++;
        }
    }
    //如果不是分隔符，就说明遇到单词，wordNum++，然后逐步跳过该单词
    else if (!IsSeparator(sentence[i]))
    {
        wordNum++;
        while (i <= len - 1 && !IsSeparator(sentence[i]))
        {
            i++;
        }
    }
    if (i > len - 1)
    {
        break;
    }
}
return wordNum;
}

bool IsSeparator(char ch)
{
    if (ch == ' ' || ch == ',' || ch == '.')
        return true;
    return false;
}

int CharCount(string str)
{
    int len = str.Length();
    int cnt = 0;
    for (int i = 0; i < len; i++)
    {
        if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z'))
        {
            cnt++;
        }
    }
    return cnt;
}
}

```

33、【We Are A Team】

题目描述: 总共有 n 个人在机房，每个人有一个标号 ($1 \leq \text{标号} \leq n$)，他们分成了多个团队，需要你根据收到的 m 条消息判定指定的两个人是否在一个团队中，具体的：

- 1、消息构成为 $a\ b\ c$ ，整数 a 、 b 分别代表两个人的标号，整数 c 代表指令
- 2、 $c == 0$ 代表 a 和 b 在一个团队内
- 3、 $c == 1$ 代表需要判定 a 和 b 的关系，如果 a 和 b 是一个团队，输出一行 'we are a team', 如果不是，输出一行 'we are not a team'
- 4、 c 为其他值，或当前行 a 或 b 超出 $1 \sim n$ 的范围，输出 'da pian zi'

输入描述:

- 1、第一行包含两个整数 n ， m ($1 \leq n, m \leq 100000$)，分别表示有 n 个人和 m 条消息
- 2、随后的 m 行，每行一条消息，消息格式为： $a\ b\ c$ ($1 \leq a, b \leq n, 0 \leq c \leq 1$)

输出描述:

- 1、 $c == 1$ ，根据 a 和 b 是否在一个团队中输出一行字符串，在一个团队中输出 'we are a team', 不在一个团队中输出 'we are not a team'
- 2、 c 为其他值，或当前行 a 或 b 的标号小于 1 或者大于 n 时，输出字符串 'da pian zi'
- 3、如果第一行 n 和 m 的值超出约定的范围时，输出字符串 'Null'

输入：

5 7

1 2 0

4 5 0

2 3 0

1 2 1

2 3 1

4 5 1

1 5 1

输出：

We are a team

We are a team

We are a team

We are not a team

输入:

5 6

1 2 0

1 2 1

1 5 0

2 3 1

2 5 1

1 3 2

输出:

we are a team

we are not a team

we are a team

da pian zi

```
class Node :
    def __init__(self, n) :
        #列表存储节点，下标是子节点，内容是父节点，初始化各节点的父节点是自己
        self.parent = list(range(n + 1))
    def get_root(self, i) : #循环查找父节点，每经过节点，指向该节点的父节点直到节点的父节点是自己
        while i != self.parent[i]:
            i = self.parent[i]
        return i
    def one_team(self, i, j) : #判断i, j是否连通，父节点是否是同一个
        return self.get_root(i) == self.get_root(j)
    def union(self, i, j) : #i的父节点指向j，将i的森林加到j上
        i_root = self.get_root(i)
        j_root = self.get_root(j)
        self.parent[i_root] = j_root

while True:
    try :
        m, n = map(int, input().split())
        team = Node(m)
        if m < 1 or m > 100000 or n < 1 or n > 100000 :
            print("NULL")
            break
        for i in range(n) :
            x, y, c = map(int, input().split())
            # union a, b in one team
            if x < 1 or x > m or y < 1 or y > m:
                print("da pian zi")
                continue
            if c == 0 :
                team.union(x, y)
            elif c == 1 :
                ret = team.one_team(x, y)
                if ret == True :
                    print("We are a team")
                else :
                    print("We are not a team")
            else:
                print("da pian zi")
        except :
            break
```

34、【整数对最小和】

题目描述: 给定两个整数数组 array1、array2，数组元素按升序排列。假设从 array1、array2 中分别取出一个元素可构成一对元素，现在需要取出 k 对元素，并对取出的所有元素求和，计算和的最小值 注意：两对元素如果对应于 array1、array2 中的两个下标均相同，则视为同一对元素。

输入描述:

输入两行数组 array1、array2，

每行首个数字为数组大小 size(0 < size <= 100);

0 < array1[i] <= 1000 0 < array2[i] <= 1000

接下来一行为正整数 k

0 < k <= array1.size() * array2.size()

输出描述:

满足要求的最小和

示例 1

输入:

3 1 1 2

3 1 2 3

2

输出:4

```
#include <iostream>

using namespace std;

struct num
{
    int x;
    int y;
};

int main(int argc, char** argv)
{
    num min;
    int n;
    num data[1000];
    while (cin >> n) {
        cin >> data[0].x >> data[0].y;
        min.x = data[0].x;
        min.y = data[0].y;
        for (int i = 1; i < n; i++) {
            cin >> data[i].x >> data[i].y;
            if (data[i].x < min.x) {
                min.x = data[i].x;
                min.y = data[i].y;
            }
            if ((data[i].x == min.x) && data[i].y < min.y) {
                min.x = data[i].x;
                min.y = data[i].y;
            }
        }
        cout << min.x << " " << min.y << endl;
    }
    return 0;
}
```

35、【寻找身高相近的小朋友】

题目描述: 小明今年升学到小学一年级，来到新班级后发现其他小朋友们身高参差不齐，然后就想基于各小朋友和自己的身高差对他们进行排序，请帮他实现排序。

输入描述:

第一行为正整数 H 和 N，0<H<200，为小明的身高，0<N<Hi<200（1<=i<=N），且 N 个正整数各不相同。

输出描述:

输出排序结果，各正整数以空格分割。和小明身高差绝对值最小的小朋友排在前面，和小明身高差绝对值最大的小朋友排在最后，如果两个小朋友和小明身高差一样，则个子较小的小朋友排在前面。

示例 1

输入:

100 10

95 96 97 98 99 101 102 103 104 105

输出:

99 101 98 102 97 103 96 104 95 105

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    vector<int> temp;
    int inpiut;
    int closestnumber=0;
    int numbers=0;
    cin>>closestnumber>>numbers;
    for(int i=0;i<numbers;i++)
    {
        cin>>inpiut;
        temp.push_back(inpiut);
    }
    sort(temp.begin(),temp.end());
    for(int i=0;i<temp.size();i++)
```

```

{
    cout<<temp[i]<<" ";
}
cout<<endl;
vector<int> front_vec;
vector<int> after_vec;
int mid=0;

int temp_length=temp.size();
for(int i=0;i<temp_length;i++)
{
    if(closestnumber>temp[i])
    {
        front_vec.push_back(temp[i]);
    }
    else if(closestnumber<temp[i])
    {
        after_vec.push_back(temp[i]);
    }
    else
    {
        mid=temp[i];
    }
}
int front_length=front_vec.size();
int after_length=after_vec.size();
vector<int>::iterator Iter=after_vec.begin();
if(mid!=0)
{
    cout<<mid<<" ";
    if(front_length<after_length)
    {
        cout<<"front_length<after_length"<<endl;
        for(int i=0;i<front_length;i++)
        {
            cout<<front_vec.back()<<" "<<after_vec.front()<<" ";
            front_vec.pop_back();
            Iter=after_vec.begin();
            after_vec.erase(Iter);
        }
        for(int j=0;j<after_length-front_length;j++)
        {
            cout<<after_vec.back()<<" ";
            after_vec.pop_back();
        }
        cout<<endl;
    }
    else if(front_length>after_length)
    {
        cout<<"front_length>after_length"<<endl;
        for(int i=0;i<after_length;i++)
        {
            cout<<front_vec.back()<<" "<<after_vec.front()<<" ";
            front_vec.pop_back();
            Iter=after_vec.begin();
            after_vec.erase(Iter);
        }
        for(int j=0;j<front_length-after_length;j++)
        {
            cout<<front_vec.back()<<" ";
            front_vec.pop_back();
        }
        cout<<endl;
    }
    else
    {
        cout<<"front_length=after_length"<<endl;
        for(int i=0;i<front_length;i++)
        {
            cout<<front_vec.back()<<" "<<after_vec.front()<<" ";
            front_vec.pop_back();
            Iter=after_vec.begin();
            after_vec.erase(Iter);
        }
        cout<<endl;
    }
}
else
{
    if(front_length<after_length)
    {
        cout<<"front_length<after_length"<<endl;
        for(int i=0;i<front_length;i++)
        {
            cout<<front_vec.back()<<" "<<after_vec.front()<<" ";
            front_vec.pop_back();

```

```

        Iter=after_vec.begin();
        after_vec.erase(Iter);
    }
    for(int j=0;j<after_length-front_length;j++)
    {
        cout<<after_vec.back()<<" ";
        after_vec.pop_back();
    }
    cout<<endl;
}
else if(front_length>after_length)
{
    cout<<"front_length>after_length"<<endl;
    for(int i=0;i<after_length;i++)
    {
        cout<<front_vec.back()<<" "<<after_vec.front()<<" ";
        front_vec.pop_back();
        Iter=after_vec.begin();
        after_vec.erase(Iter);
    }
    for(int j=0;j<front_length-after_length;j++)
    {
        cout<<front_vec.back()<<" ";
        front_vec.pop_back();
    }
    cout<<endl;
}
else
{
    cout<<"front_length=after_length"<<endl;
    for(int i=0;i<front_length;i++)
    {
        cout<<front_vec.back()<<" "<<after_vec.front()<<" ";
        front_vec.pop_back();
        Iter=after_vec.begin();
        after_vec.erase(Iter);
    }
    cout<<endl;
}
}
return 0;
}

```

36、【按身高和体重排队】

题目描述:

某学校举行运动会，学生们按编号(1、2、3...n)进行标识，现需要按照身高由低到高排列，对身高相同的人，按体重由轻到重排列；对于身高体重都相同的人，维持原有的编号顺序关系。请输出排列后的学生编号。

输入描述:

两个序列，每个序列由 n 个正整数组成（ $0 < n \leq 100$ ）。第一个序列中的数值代表身高，第二个序列中的数值代表体重。

输出描述:

排列结果，每个数值都是原始序列中的学生编号，编号从 1 开始

示例 1

输入:

4

100 100 120 130

40 30 60 50

输出: 2 1 3 4

```

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

typedef struct Student
{
    int number;
    int height;
    int weight;
}Student;

bool operator<(const Student& st1,const Student& st2)
{

```

```

        if(st1.height>st2.height)
        {
            return true;
        }
        else if((st1.height==st2.height)&&(st1.weight>st2.weight))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

void showStudent(const Student& st)
{
    cout<<st.number<<" ";
}

int main(int argc,char** argv)
{
    vector<Student> sts;
    unsigned n;
    Student st;

    cin>>n;
    for(unsigned i=0;i<n;++i)
    {
        st.number=i+1;
        cin>>st.height>>st.weight;
        sts.push_back(st);
    }

    sort(sts.begin(),sts.end());
    for_each(sts.begin(),sts.end(),showStudent);//注意这里改成begin(),end()
    cout<<endl;
    return 0;
}

```

37、【磁盘容量排序】

题目描述:

磁盘的容量单位常用的有 M, G, T 这三个等级, 它们之间的换算关系是 1T=1000G, 1G=1000M, 现在给定 n 块磁盘的容量, 请对它们按从小到大的顺序进行排序, 例如给定 3 块盘的容量, 1T, 20M, 3G, 排序后的结果未 20M, 3G, 1T

输入描述:

- 1、每个测试用例第一行包含一个整数 n(2<=n<=1000), 表示磁盘的个数, 接下的 n 行, 每行一个字符串 (长度大于 2, 小于 10), 表示磁盘的容量, 格式为: mv, 其中 m 表示容量大小, v 表示容量单位, 例如 20M, 1T, 30G

- 2、磁盘容量 m 表示十进制数范围为 1 到 1000 的正整数, 容量单位 v 的范围只包含题目中提到的 M, G, T 三种, 换算关系如题目描述

输出描述:

输出 n 行, 表示 n 块磁盘容量排序后的结果

示例 1:

输入:

3

1T

20M

3G

输出:

20M

3G

1T

```

#include <iostream>
#include <unordered_map>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int StrToInt(string str)
{
    if(str[str.size()-1]=='M')
    {
        return stoi(str.substr(0,str.size()-1));
    }
    else if(str[str.size()-1]=='G')

```

```

    {
        return stoi(str.substr(0,str.size()-1))*1000;
    }
    else if(str[str.size()-1]=='T')
    {
        return stoi(str.substr(0,str.size()-1))*1000000;
    }

    return 0;
}

bool Compare(const string& strA,const string& strB)
{
    int a=StrToInt(strA);
    int b=StrToInt(strB);

    //升序排序
    return a<b;
}

int main(int argc,char** argv)
{
    int n;
    while(cin>>n)
    {
        string str;
        vector<string> vec;

        while(n--)
        {
            cin>>str;
            vec.push_back(str);
        }

        sort(vec.begin(),vec.end(),Compare);

        for(auto i:vec)
        {
            cout<<i<<endl;
        }

        return 0;
    }
}

```

38、【可以组成网络的服务器】

题目描述:

在一个机房中，服务器的位置标识在 $n*m$ 的整数矩阵网格中，1 表示单元格上有服务器，0 表示没有。如果两台服务器位于同一行或者同一列中紧邻的位置，则认为它们之间可以组成一个局域网。请你统计机房中最大的局域网包含的服务器个数。

输入描述:

第一行输入两个正整数， n 和 m ， $0 < n, m \leq 100$ 之后为 $n*m$ 的二维数组，代表服务器信息

输出描述:

最大局域网包含的服务器个数。

示例 1

输入:

22

10

11

输出:

3

```

#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
using namespace std;

int main(int argc, char** argv)
{
    //vector<vector<int>>
    v{{1,0,1,0,1,1},{0,1,0,1,0,0},{1,0,1,0,0,0},{0,1,0,1,0,0},{1,0,0,0,1,0},{1,0,0,0,0,1}};
    vector<vector<int>>
    v{ {1,0,1,0,1,1},{0,1,0,0,0,0},{1,0,1,0,0,0},{0,0,0,1,0,0},{1,0,0,0,1,0},{1,0,0,0,0,1} };
    unordered_map<int, set<int>> a;
    auto len = v.size();
    for (int i = 0; i < len; i++)
    {

```

```

        for (int j = 0; j < len; j++)
        {
            if (v[i][j] == 1)
            {
                a[i].insert(j);
            }
        }
    }
    for (auto iter = a.begin(); iter != a.end(); iter++)
    {
        int num = iter->first;
        set<int> tmp = iter->second;
        for (int j : tmp)
        {
            if (num != j)
            {
                a[j].insert(tmp.begin(), tmp.end());
            }
        }
    }
    set<set<int>> ret;
    for (auto iter = a.begin(); iter != a.end(); iter++)
    {
        set<int> tmp = iter->second;
        ret.insert(tmp);
    }
    cout << ret.size() << endl;

    return 0;
}

```

39、【字符串加密】

题目描述:

给你一串未加密的字符串 str，通过对字符串的每一个字母进行改变来实现加密，加密方式是在每一个字母 str[i]偏移特定数组元素 a[i]的量，数组 a 前三位已经赋值：a[0]=1,a[1]=2,a[2]=4。当 i>=3 时，数组元素 a[i]=a[i-1]+a[i-2]+a[i-3]，例如：原文 abcde 加密后 bdgkr，其中偏移量分别是 1,2,4,7,13。

输入描述:

第一行为一个整数 n (1<=n<=1000)，表示有 n 组测试数据，每组数据包含一行，原文 str（只含有小写字母，0<长度<=50）。

输出描述:

每组测试数据输出一行，表示字符串的密文

示例 1

输入:1xy

输出:ya

```

#include <iostream>
#include <vector>
#include <string>

using namespace std;

void encrypt(string& str)
{
    vector<int> array;
    array.push_back(1);
    array.push_back(2);
    array.push_back(4);
    int length=str.size();
    for(int i=0;i<3;i++)
    {
        str[i] = str[i]+array[i];
        if(str[i]>'z')
        {
            str[i]='a';
        }
    }
    for(int i=3;i<length;i++)
    {
        array[i]=array[i-1]+array[i-2]+array[i-3];
        str[i]+=array[i];
        if(str[i]>'z')
        {
            str[i]='a';
        }
    }
}

int main(int argc, char** argv)
{
    int numbers=0;
}

```



```

string str;
vector<string> input;
while(1)
{
    input.clear();
    cin>>numbers;
    for(int i=0;i<numbers;i++)
    {
        cin>>str;
        input.push_back(str);
    }
    for(int i=0;i<numbers;i++)
    {
        string temp=input[i];
        encrypt(temp);
        cout<<temp<<endl;
    }
}

return 0;
}

```

40、【全排列】

题目描述： 给定一个没有重复数字的序列，返回其所有可能的全排列。

输入描述：

输入:[1,2,3]

输出描述：

输出:[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

示例 1:

输入: 1,2,3

输出: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

```

#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
using namespace std;

class Solution
{
public:
    void backtrack(vector<vector<int>>& res, vector<int>& output, int first, int len)
    {
        // 所有数都填完了
        if (first == len)
        {
            res.emplace_back(output);
            return;
        }
        for (int i = first; i < len; ++i)
        {
            // 动态维护数组
            swap(output[i], output[first]);
            // 继续递归填下一个数
            backtrack(res, output, first + 1, len);
            // 撤销操作
            swap(output[i], output[first]);
        }
    }
    vector<vector<int>> permute(vector<int>& nums)
    {
        vector<vector<int>> res;
        backtrack(res, nums, 0, (int)nums.size());
        return res;
    }
};

```

41、【全排列 II】

给定一个可包含重复数字的序列 `nums`，按任意顺序 返回所有不重复的全排列。

示例 1:

输入: `nums = [1,1,2]`

输出:

[[1,1,2],

[1,2,1],

[2,1,1]]

示例 2:

输入: nums = [1,2,3]

输出: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <algorithm>
using namespace std;

class Solution
{
    vector<int> vis;

public:
    void backtrack(vector<int>& nums, vector<vector<int>>& ans, int idx, vector<int>& perm)
    {
        if (idx == nums.size())
        {
            ans.emplace_back(perm);
            return;
        }
        for (int i = 0; i < (int)nums.size(); ++i)
        {
            if (vis[i] || (i > 0 && nums[i] == nums[i - 1] && !vis[i - 1]))
            {
                continue;
            }
            perm.emplace_back(nums[i]);
            vis[i] = 1;
            backtrack(nums, ans, idx + 1, perm);
            vis[i] = 0;
            perm.pop_back();
        }
    }

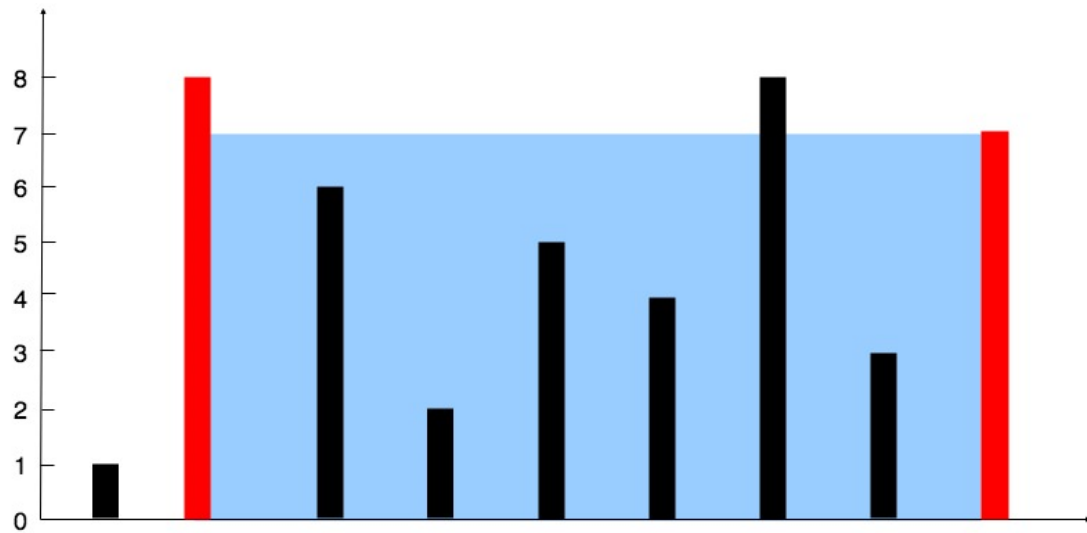
    vector<vector<int>> permuteUnique(vector<int>& nums)
    {
        vector<vector<int>> ans;
        vector<int> perm;
        vis.resize(nums.size());
        sort(nums.begin(), nums.end());
        backtrack(nums, ans, 0, perm);
        return ans;
    }
};
```

42、【盛最多水的容器】

题目描述:

给定 n 个非负整数 a_1, a_2, \dots, a_n ，每个数代表坐标中的一个点 (i, a_i) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, a_i) 和 $(i, 0)$ 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2。



图中垂直线代表输入数组 1,8,6,2,5,4,8,3,7。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

输入描述:

1,8,6,2,5,4,8,3,7

输出描述:

49

示例 1:

输入: 1,8,6,2,5,4,8,3,7

输出: 49

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
using namespace std;

int MaxArea(vector<int>& height)
{
    int l = 0, r = height.size() - 1;
    int ans = 0;
    while (l < r)
    {
        int area = min(height[l], height[r]) * (r - l);
        ans = max(ans, area);
        if (height[l] <= height[r])
        {
            ++l;
        }
        else
        {
            --r;
        }
    }
    return ans;
}

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}
```

```

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        input_string = input_string.substr(1, input_string.size() - 1);
        vector<int> vec;
        Split(input_string, vec, ' ');

        cout << MaxArea(vec) << endl;
    }
    return 0;
}

```

43、【跳跃比赛】

题目描述：给出一组正整数，你从第一个数向最后一个数方向跳跃，每次至少跳跃 1 格，每个数的值表示你从这个位置可以跳跃的最大长度。计算如何以最少的跳跃次数跳到最后一个数。

输入描述：

第一行表示有多少个数 n

第二行开始依次是 1 到 n 个数，一个数一行

输出描述：

输出一行，表示最少跳跃的次数。

示例 1:

输入: 72321215

输出: 3

```

#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    int n;
    int i, j;
    cin >> n;
    vector<int> v(n);
    for (i = 0; i < n; i++)
    {
        cin >> v[i];
    }
    vector<int> dp(n, n - 1);
    dp[0] = 0;
    for (i = 1; i < n; i++)
    {
        for (j = i - 1; j >= 0; j--)
        {
            if (v[j] >= (i - j))
            {
                dp[i] = min(dp[i], dp[j] + 1);
            }
        }
    }
    cout << dp[n - 1] << endl;
    return 0;
}

```

44、【字符串处理】

题目描述：让我们来玩个字符串消除游戏吧,给定一个只包含大写字母的字符串 s，消除过程是如下进行的：

1)如果 s 包含长度为 2 的由相同字母组成的子串，那么这些子串会被消除，余下的子串拼成新的字符串。

例如"ABCCBCCCCAA"中"CC","CC"和"AA"会被同时消除，余下"AB","C"和"B"拼成新的字符串"ABBC"。

2)上述消除会反复一轮一轮进行，直到新的字符串不包含相邻的相同字符为止。

例如"ABCCBCCCCAA"经过一轮消除得到"ABBC"，再经过一轮消除得到"AC"

输入描述：

输入由大写字母组成的字符串 s，长度不超过 100。

输出描述：

若最后可以把整个字符串全部消除,就输出 Yes,否则输出 No。

示例 1:

输入: ABCCBA

输出: Yes

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

int main(int argc, char** argv)
{
    int n = 0;
    int m = 0;
    int leng = 0;
    int i, j = 0;
    char str[100] = { 0 };
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%s", str);
        //printf("str is %s\r\n", str);
        leng = strlen(str);
        for (j = 0; j < leng; j++)
        {
            if (str[j] == str[j + 1])
            {
                //删除元素
                for (m = j; m < leng - 2; m++)
                {
                    str[m] = str[m + 2];
                }
                //printf("str 1 is %s\r\n", str);
                leng = leng - 2;
                j = -1;
            }
        }
        //printf("leng is %d\r\n", leng);
        if (leng == 0) {
            printf("Yes\r\n");
        }
        else {
            printf("No\r\n");
        }
    }
}
```

45、【航班预定座位数】

题目描述: 有 n 个航班, 它们分别从 1 到 n 进行编号。

我们这儿有一份航班预订表, 表中第 i 条预订记录 $\text{bookings}[i] = [i, j, k]$ 意味着我们在从 i 到 j 的每个航班上预订了 k 个座位。

请返回一个长度为 n 的数组 answer , 按航班编号顺序返回每个航班上预订的座位数。

输入描述:

前几行为航班预定信息, 最后一行为航班数量

输出描述:

长度为 n 的数组

示例 1:

输入: 1,2,10 2,3,20 2,5,25 5

输出: 10,55,45,25,25

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <algorithm>
using namespace std;

class Solution
{
public:
    vector<int> corpFlightBookings(vector<vector<int>>& bookings, int n)
    {
        vector<int> res(n + 1, 0);
        for (auto& b : bookings)
        {
            res[b[0] - 1] += b[2];
            res[b[1]] -= b[2];
        }
        for (int i = 1; i < n; ++i)
```

```

        {
            res[i] += res[i - 1];
        }
        res.pop_back();
        return res;
    }
};

```

46、【密钥格式化】

题目描述：给定一个非空字符串 S，其被 N 个'-'分隔成 N+1 的子串，给定正整数 K，要求除第一个子串外，其余的串每 K 个用'-'分隔，并将小写字母转换为大写。

输入描述：

正整数 K 和'-'分割的字符串，如：

2

25G3C-abc-d

输出描述：

转换后的字符串，

示例 1:

输入: S = "5F3Z-2e-9-w", K = 4

输出: "5F3Z-2E9W"

解释: 字符串 S 被分成了两个部分，每部分 4 个字符:

注意，两个额外的破折号需要删掉。

示例 2:

输入: S = "2-5g-3-j", K = 2

输出: "2-5G-3J"

解释: 字符串 S 被分成了 3 个部分，按照前面的规则描述，第一部分的字符可以少于给定的数量，其余部分皆为 2 个字符。

```

#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <algorithm>
using namespace std;

class Solution
{
public:
    string licenseKeyFormatting(string S, int K)
    {
        string str;
        for (const auto& ch : S)
        {
            if (ch == '-') {
                continue;
            }
            else if (isdigit(ch))
            {
                str += ch;
            }
            else // alpha
            {
                str += toupper(ch);
            }
        }

        if (str.size() < K)
        {
            return str;
        }

        string res;
        int i = 0;
        int N = str.size();
        res += str.substr(i, N % K);
        for (i = N % K; i <= N - K; i += K)
        {
            res += '-' + str.substr(i, K);
        }

        return N % K != 0 ? res : res.substr(1);
    }
};

```

47、【服务器广播】

题目描述：服务器连接方式包括直接相连，间接连接。A 和 B 直接连接，B 和 C 直接连接，则 A 和 C 间接连接。直接连接和间接连接都可以发送广播。

给出一个 $N \times N$ 数组，代表 N 个服务器。 $matrix[i][j]=1$ ，则代表 i 和 j 直接连接；不等于 1 时，代表 i 和 j 不直接连接。 $matrix[i][i]=1$ ，即自己和自己直接连接。 $matrix[i][j]=matrix[j][i]$ 。

计算初始需要给几台服务器广播，才可以使每个服务器都收到广播。

输入描述：

$n \times n$ 矩阵

输出描述：

整数

示例 1:

输入：110110001

输出：2

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
using namespace std;

int main(int argc, char** argv)
{
    //vector<vector<int>>
    v{{1,0,1,0,1,1},{0,1,0,1,0,0},{1,0,1,0,0,0},{0,1,0,1,0,0},{1,0,0,0,1,0},{1,0,0,0,0,1}};
    vector<vector<int>>
    v{ {1,0,1,0,1,1},{0,1,0,0,0,0},{1,0,1,0,0,0},{0,0,0,1,0,0},{1,0,0,0,1,0},{1,0,0,0,0,1} };
    unordered_map<int, set<int>> a;
    auto len = v.size();
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            if (v[i][j] == 1)
            {
                a[i].insert(j);
            }
        }
        for (auto iter = a.begin(); iter != a.end(); iter++)
        {
            int num = iter->first;
            set<int> tmp = iter->second;
            for (int j : tmp)
            {
                if (num != j)
                {
                    a[j].insert(tmp.begin(), tmp.end());
                }
            }
        }
        set<set<int>> ret;
        for (auto iter = a.begin(); iter != a.end(); iter++)
        {
            set<int> tmp = iter->second;
            ret.insert(tmp);
        }
        cout << ret.size() << endl;
    }
    return 0;
}
```

48、【分子弹】

题目描述：射击训练，需要给每个士兵发子弹，发子弹的个数规则是根据士兵的训练成绩给定的，传入士兵的训练成绩，要求相邻士兵中，成绩好的士兵的子弹数必须更多，

每个士兵至少分配一个子弹。

输入描述：

输入每个士兵的训练成绩，如 1,2,3，代表 3 位士兵的成绩分别为 1,2,3

输出描述：

最少分配的子弹数

示例 1:

输入：1,2,2

输出：4

```
#include <iostream>
```

```

#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <algorithm>
using namespace std;

class Solution {
public:
    int candy(vector<int>& ratings)
    {
        int n = ratings.size();
        vector<int> left(n);
        for (int i = 0; i < n; i++)
        {
            if (i > 0 && ratings[i] > ratings[i - 1])
            {
                left[i] = left[i - 1] + 1;
            }
            else
            {
                left[i] = 1;
            }
        }
        int right = 0, ret = 0;
        for (int i = n - 1; i >= 0; i--)
        {
            if (i < n - 1 && ratings[i] > ratings[i + 1])
            {
                right++;
            }
            else
            {
                right = 1;
            }
            ret += max(left[i], right);
        }
        return ret;
    }
};

```

49、【无重复字符的最长子串】

题目描述：给定一个字符串，请你找出其中不含有重复字符的最长子串的长度。

输入描述：

输入只有一行，包含一个字符串 S，S 中只包含小写字母

输出描述：

输出不含有重复字符的最长子串的长度

示例 1:

输入: abcabcbb

输出: 3

```

class Solution
{
public:

    int lengthOfLongestSubstring(string s) {
        map<char, int> hash;
        int ans = 0;
        int i = 0;
        int j = 0;
        int n = s.Length();
        while (i < n && j < n)
        {
            if (hash.find(s[j]) == hash.end())
            {
                hash[s[j++]] = j; // 等价于hash[s[j]]=j;j++; j自增必须要在下一条语句中的j-i之前
                ans = max(ans, j - i);
            }
            else
            {
                hash.erase(s[i++]); // 根据键值删除哈希表中元素。该条语句等价于hash.erase(s[i]);i++; 每回
                // 向右滑动一位（此处可以改善，见方法三）
            }
        }
        return ans;
    }
};

```



```

class Solution {
public:
    int lengthOfLongestSubstring(string s)
    {
        map<char, int> hash;
        int ans = 0;
        int n = s.size();
        for (int i = 0, j = 0; j < n; j++)
        {
            if (hash.find(s[j]) != hash.end())
            {
                i = max(hash.find(s[j])->second + 1, i); //当发现重复字符时，i直接跳到重复字符的下一位，改进了方法2中一个字符一个字符滑动窗口的方法
            }
            ans = max(ans, j - i + 1);
            hash[s[j]] = j;
        }
        return ans;
    }
};

```

50、【九宫格按键输入】

题目描述：九宫格按键输入，判断输出，有英文和数字两个模式，默认是数字模式，数字模式直接输出数字，英文模式连续按同一个按键会依次出现这个按键上的字母，如果输入"/"或者其他字符，则循环中断

1 (.,) 2 (abc) 3 (def)
 4 (ghi) 5 (jkl) 6 (mno)
 7 (pqrs) 8 (tuv) 9 (wxyz)
 #0 (空格) /

要求输入一串按键，输出屏幕显示

输入描述：

输入范围为数字 0~9 和字符'#'、'/'，输出屏幕显示，例如，

在数字模式下，输入 1234，显示 1234

在英文模式下，输入 1234，显示,adg

输出描述：

- 1、#用于切换模式，默认是数字模式，执行#后切换为英文模式；
- 2、/表示延迟，例如在英文模式下，输入 22/222，显示为 bc；
- 3、英文模式下，多次按同一键，例如输入 22222，显示为 b；

示例 1:

输入：123#222235/56

输出：123adajm

```

var nineInput = function(arr) {
    const button=["0 ", "1,.,?!","2ABC","3DEF","4GHI","5JKL","6MNO","7PQRS","8TUV","9WXYZ"];
    let str='';
    for(let i=0;i<arr.length;i++){
        let num=arr[i][0];
        let index=(arr[i].length>button[num].length?(arr[i].length%button[num].length):arr[i].length)-1
        str+=button[num][index]
    }
    return str
};

```

```

console.log(nineInput(['22', '5555', '22', '666', '00', '88', '888', '7777', '4444', '666', '44']))

```

```

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

char a[][7] = { " 0 ", "1,.,?!","C2ABC","F3DEF","I4GHI","L5JKL","O6MNO","S7PQRS","V8TUV","Z9WXYZ" };

int main(int argc, char** argv)
{
    char b[505];
    int i;
    int len;
    int sum, k = 0, sum1 = 0;
    while (gets_s(b) != NULL)
    {
        len = strlen(b);
        b[len] = ' ';
        for (i = 0; i <= len; i++)
        {
            if (b[i] != ' ')
            {

```

```

        if (k == 0)
        {
            //记录你安的是那个键
            sum = b[i] - '0';
            k = 1;
        }
        //按了几次。
        sum1++;
    }
    else
    {
        //去掉循环部分,去掉a[0]的那个多余的
        int score = strlen(a[sum]);
        score = score - 1;
        //判断是不是循环的
        if (sum1 >= score)
            sum1 = sum1 % score;
        //输出那个位置的字母
        printf("%c", a[sum][sum1]);
        //初始化
        sum1 = 0;
        k = 0;
    }
}
printf("\n");
}
return 0;
}

```

51、【最长回文子串】

题目描述: 给定一个字符串 s , 找到 s 中最长的回文子串。你可以假设 s 的最大长度为 1000。

示例 1:

输入: "babad"

输出: "bab"

注意: "aba" 也是一个有效答案。

```

class Solution
{
public:
    string longestPalindrome(string s)
    {
        if (s.size() < 2) return s;
        int maxlen = 0, start = 0;
        for (int i = 0; i < s.size() - 1; ++i)
        {
            find(s, i, i, start, maxlen);
            find(s, i, i + 1, start, maxlen);
        }
        return s.substr(start, maxlen);
    }
    void find(string s, int left, int right, int& start, int& maxlen)
    {
        while (left >= 0 && right < s.size() && s[left] == s[right]) //向两边扩散
        {
            --left;
            ++right;
        }
        if (maxlen < right - left - 1)
        {
            start = left + 1;
            maxlen = right - left - 1;
        }
    }
};

```

52、【两个字符串的最长公共子串】

题目描述: 查找两个字符串 a,b 中的最长公共子串。若有多个，输出在较短串中最先出现的那个。

输入描述:

输入两个字符串

输出描述:

返回重复出现的字符

输入

abcde fghijklmnop

abcsafjklmnopqrstuvw

输出

jklmnop

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;
string LCCS(const string& str1, const string& str2)
{
    string ret;
    int sz1 = str1.size(), sz2 = str2.size();
    vector<vector<int>> dp(sz1 + 1, vector<int>(sz2 + 1, 0));
    for (int i = 1; i <= sz1; ++i)
    {
        for (int j = 1; j <= sz2; ++j)
        {
            if (str1[i - 1] == str2[j - 1])
            {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                if (dp[i][j] > ret.size())
                    ret = str1.substr(i - dp[i][j], dp[i][j]);
            }
            else
                dp[i][j] = 0;
        }
    }
    return ret;
}

int main(int argc, char** argv)
{
    string str1, str2;
    while (cin >> str1 >> str2)
    {
        if (str1.size() > str2.size())
            swap(str1, str2);
        cout << LCCS(str1, str2) << endl;
    }
    return 0;
}
```

53、【括号生成】

题目描述：给定一个无重复元素的数组和一个目标数 **target**，找出数组中所有可以使数字和为 **target** 的组合。

数组中的数字可以无限制重复被选取。

说明：

所有数字（包括 **target**）都是正整数。

解集不能包含重复的组合。

输入描述：

输入:2,3,6,7 7

输出描述：

[[7],[2,2,3]]

示例 1:

输入: 5,3,6,79

输出: [[3,3,3],[3,6]]

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

class Solution
{
public:
    void dfs(vector<int> candidates, int start, int target, vector<int> item, vector<vector<int>> & res)
    {
        if (target < 0)
        {
            return;
        }
        if (target == 0)
        {
            res.push_back(item);
            return;
        }
        for (int i = start; i < candidates.size(); i++)
        {
            item.push_back(candidates[i]);
```

```

        dfs(candidates, i, target - candidates[i], item, res); //注意参数i,可以选择重复元素而又可以保证不会
        发生重复(i一直是递增的)
        item.pop_back();
    }
}
vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> > res;
    vector<int> item;
    // item保存当前dfs的解空间
    // res保存所有满足条件的解
    dfs(candidates, 0, target, item, res);
    return res;
}
};

```

54、【相加】

题目描述: 华老师让你们计算一个等式, $s = a + aa + aaa + aaaa + aa...a$ (总共有 n 个这样的数相加) 的值, 其中 a 是一个 $1 \sim 9$ 的数字。例如当 $a=2, n=5$ 时, $s = 2 + 22 + 222 + 2222 + 22222$ 。

输入描述:

输入一行包含两个整数 $a(1 \leq a \leq 9)$ 和 $n(1 \leq n \leq 9)$, 其中 n 为几个这样的数字相加 (如描述中的例子 n 是 5)。 a 和 n 用空格隔开

输出描述:

输出一个符合题目算法的结果 s 。

示例 1:

输入: 25

输出: 24690

```

#include<stdio.h>
void main()
{
    int a, n, s = 0, t;
    printf("输入a和n: ");
    scanf("%d%d", &a, &n);
    t = a; //t = a1
    printf("s=");
    while (n > 0)
    {
        printf("%d", t); //显示每项相加的数
        s += t;
        a = a * 10; //每次a2扩大10倍加上a1
        t = t + a; //t = an;没循环一次n+1;
        n--;
        if (n != 0) printf("+");
    }
    printf("%d", s);
}

```

55、【幸运数字】

题目描述: 我们这么定义幸运数: 若 k 能被 x 、 y 、 z 任意一个数整除就可认为 k 是幸运数。现在你的任务是计算区间 $[1, n]$ 之间一共有多少个幸运数。

输入描述:

输入只有一行, 包含四个整数 $n, x, y, z(1 \leq n \leq 10^9, 1 \leq x, y, z \leq 100)$, 分别用空格隔开

输出描述:

输出只有一行, 即幸运数的个数。

示例 1:

输入: 10 3 5 7

输出: 6

```

#include<stdio.h>
int main()
{
    int n, x, y, z, num, i;
    num = 0;
    scanf_s("%d%d%d%d", &n, &x, &y, &z);
    for (i = 1; i <= n; i++)
    {
        if (i % x == 0 && i % y == 0 && i % z == 0)
        {
            num++;
        }
    }
    printf("%d", num);
    return 0;
}

```

56、【奥运会排行榜】

题目描述：2012 伦敦奥运会即将到来，大家都非常关注奖牌榜的情况，现在我们假设奖牌榜的排名规则如下：1、首先 goldmedal 数量多的排在前面；2、其次 silvermedal 数量多的排在前面；3、然后 bronzemedal 数量多的排在前面；4、若以上三个条件仍无法区分名次，则以国家名称的字典序排定。我们假设国家名称不超过 20 个字符、各种奖牌数不超过 100，且大于等于 0。

输入描述：

第一行输入一个整数 N(0<N<21)，代表国家数量;然后接下来的 N 行，每行包含一个字符串 Namei 表示每个国家的名称，和三个整数 Gi、Si、Bi 表示每个获得的 goldmedal、silvermedal、bronzemedal 的数量，以空格隔开，如(China512021)，具体见样例输入。

输出描述：

输出奖牌榜的依次顺序，只输出国家名称，各占一行，具体见样例输出。

示例 1:

输入：3 China 512021 American 5011 Japan000

输出：China American Japan

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

typedef struct stat
{
    char name[32];
    int gold;
    int silver;
    int bronze;
}Stat;

void switch_stat(Stat* stat1, Stat* stat2)
{
    Stat tmp;

    if (!stat1 || !stat2)
    {
        return;
    }

    tmp = *stat1;
    *stat1 = *stat2;
    *stat2 = tmp;
}

/* Put your function here */

int main(int argc, char* argv[])
{
    int i, j;
    unsigned int len;
    Stat* vector = NULL;

    /* get the array lenth from input */
    scanf("%u", &len);

    vector = (Stat*)malloc(len * sizeof(Stat));
    if (NULL == vector) {
        return 1;
    }

    /* get the array value from input */
    for (i = 0; i < len; i++)
    {
        scanf("%s %d %d %d", vector[i].name, &vector[i].gold, &vector[i].silver, &vector[i].bronze);
    }

    /* sort the array */
    for (i = 0; i < len - 1; i++)
    {
        for (j = i + 1; j < len; j++)
        {
            if (vector[i].gold < vector[j].gold)
            {
                switch_stat(&vector[i], &vector[j]);
            }
            else if (vector[i].gold == vector[j].gold)
            {
                if (vector[i].silver < vector[j].silver)
                {
                    switch_stat(&vector[i], &vector[j]);
                }
                else if (vector[i].silver == vector[j].silver)
                {
                    if (vector[i].bronze < vector[j].bronze)
```

```

        {
            switch_stat(&vector[i], &vector[j]);
        }
        else if (vector[i].bronze == vector[j].bronze)
        {
            if (strcmp(vector[i].name, vector[j].name) > 0)
            {
                switch_stat(&vector[i], &vector[j]);
            }
        }
    }
}

for (i = 0; i < len; i++)
{
    printf("%s\n", vector[i].name);
}

free(vector);
return 0;
}

```

57、【连续字母】

题目描述：给你一个字符串，只包含大写字母，求同一字母连续出现的最大次数。例如“AAAABBCDHHH”，同一字母连续出现的最大次数为 4，因为一开始 A 连续出现了 4 次。

输入描述：

输入一个子串(1<长度<=100)。

输出描述：

输出对应每个子串同一字母连续出现的最大次数。

示例 1:

输入: AAAABBCDHHH

输出: 4

```

#include<iostream>
#include <string.h>
using namespace std;
int main()
{
    int cases, max[100];
    cout << "输入测试数据个数:" << endl;
    cin >> cases;
    for (int i = 0; i < cases; i++)
    {
        char a[100];
        cin >> a;
        int count = 1;
        int len = strlen(a);
        for (int j = 0; j < len; j++)
        {
            if (a[j] != a[j + 1])
            {
                if (count > max[i])
                {
                    max[i] = count;
                    count = 1;
                }
            }
            else
            {
                count++;
            }
        }
    }
    for (int k = 0; k < cases; k++)
    {
        cout << max[k] << endl;
    }
    return 0;
}

```

58、【单词压缩编码】

题目描述：给定一个单词列表，我们将这个列表编码成一个索引字符串 S 与一个索引列表 A。

例如，如果这个列表是["time","me","bell"]，我们就可以将其表示为 S="time#bell#" 和 indexes=[0,2,5]。

对于每一个索引，我们可以通过从字符串 S 中索引的位置开始读取字符串，直到"#"结束，来恢复我们之前的单词列表。

那么成功对给定单词列表进行编码的最小字符串长度是多少呢？

输入描述:

输入:words=["time","me","bell"]

输出:10

说明:S="time#bell#", indexes=[0,2,5]。

输出描述:

编码长度

示例 1:

输入: time,me,bell

输出: 10

```
static bool cmp(string a, string b)
{
    return a.Length() > b.Length();
}
int minimumLengthEncoding(vector<string>& words)
{
    sort(words.begin(), words.end(), cmp);
    string ans = words[0] + "#";
    for (int i = 1; i < words.size(); i++)
    {
        string::size_type idx;
        idx = ans.find(words[i]);
        while (idx != string::npos)
        {
            if (idx != string::npos && ans[idx + words[i].Length()] == '#') break;
            idx = ans.find(words[i], idx + 1);
        }
        if (idx == string::npos) ans += words[i] + "#";
    }
    return ans.Length();
}
```

59、【整数翻转】

题目描述: 给出一个 32 位的有符号整数，你需要将这个整数中每位上的数字进行反转。输出结果超出有效范围的输入，则返回 0

输入描述:

输入为符合要求的整数，无需校验

输出描述:

输出翻转后的整数，即一个 32 位的有符号整数，超出范围返回 0

示例 1:

输入: -123

输出: -321

```
class Solution {
public:
    int reverse(int x)
    {
        string s;
        if (x < 0) s += '-';
        x = abs(x);
        for (; x; x /= 10)
        {
            s += x % 10 + 0x30;
        }
        try
        {
            return stoi(s);
        }
        catch (exception& e)
        {
            return 0;
        }
    }
};
```

60、【元音字母请大写】

题目描述: solo 从小就对英文字母非常感兴趣，尤其是元音字母(a,e,i,o,u,A,E,I,O,U)，他在写日记的时候都会把元音字母写成大写的，辅音字母则都写成小写，虽然别人看起来很别扭，但是 solo 却非常熟练。你试试把一个句子翻译成 solo 写日记的习惯吧。

输入描述:

输入一个字符串 S(长度不超过 100，只包含大小写的英文字母和空格)。

输出描述:

按照 solo 写日记的习惯输出翻译后的字符串 S。

示例 1:

输入: Intheend,It'snottheyearsinyourlifethatcount.It'sthelifeinyoureyears.

输出: InthEEnd,It'snOthEyEArSinyOUrIfIfEthAtcOUnt.It'sthElIfElInyOUryEARs.

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

int num = 0;

int main()
{
    char str[101] = { 0 };
    int i = 0;
    int j = 0;

    while ((str[j] = getchar()) != '\n')
    {
        j++;
    }

    //gets(str);

    for (i = 0; i < 100; i++) {
        if (str[i] >= 'A' && str[i] <= 'Z') {
            if (str[i] != 'A' && str[i] != 'E' && str[i] != 'I' && str[i] != 'O' && str[i] != 'U') {
                str[i] = str[i] + 32;
            }
        }
        else if (str[i] >= 'a' && str[i] <= 'z') {
            if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u') {
                str[i] = str[i] - 32;
            }
        }
    }

    printf("%s\n", str);
    return 0;
}
```

61、【求两个长长整型的数据的和】

求两个长长整型的数据的和并输出

```
#include<stdio.h>
#include<string.h>
#include<malloc.h>
int main()
{
    char* num1, * num2; //两个长长整型数据
    char* sum;
    // inttemp;
    int len_num1, len_num2; // 两个长长整型数据的长度
    int len_max, len_min;
    num1 = (char*)malloc(sizeof(char));
    num2 = (char*)malloc(sizeof(char));
    printf("输入两个长长整型数据: ");
    scanf("%s", num1);
    printf("输入两个长长整型数据: ");
    scanf("%s", num2);
    len_num1 = strlen(num1);
    len_num2 = strlen(num2);
    len_max = (len_num1 >= len_num2) ? len_num1 : len_num2;
    len_min = (len_num1 <= len_num2) ? len_num1 : len_num2;
    int len_max1 = len_max;
    sum = (char*)malloc(sizeof(char) * len_max);
    memset(sum, 0x00, len_max + 1); //切忌初始化
    for (; len_num1 > 0 && len_num2 > 0; len_num1--, len_num2--)
    {
        sum[len_max--] = ((num1[len_num1 - 1] - '0') + (num2[len_num2 - 1] - '0'));
    }
    if (len_num1 > 0)
    {
        sum[len_max--] = num1[len_num1 - 1] - '0';
        len_num1--;
    }
    if (len_num2 > 0)
    {
        sum[len_max--] = num2[len_num2 - 1] - '0';
        len_num2--;
    }
    for (int j = len_max1; j >= 0; j--) //实现进位操作
```



```

{
    // temp=sum[j]-'0';
    if (sum[j] >= 10)
    {
        sum[j - 1] += sum[j] / 10;
        sum[j] %= 10;
    }
}
char* outsum = (char*)malloc(sizeof(char) * len_max1);
int j = 0;
while (sum[j] == 0) //跳出头部0 元素
    j++;
int m = 0;
for (m = 0; m < len_max1; j++, m++)
    outsum[m] = sum[j] + '0';
outsum[m] = '\0';
printf("输出两长长整型数据之和:%s\n", outsum);
return 0;
}

```

62、【求数组的最小数和最大数之和】

第一题的题目大概是输入整型数组求数组的最小数和最大数之和，例如输入 1,2,3,4 则输出为 5，当输入只有一个数的时候，则最小数和最大数都是该数，例如只输入 1，则输出为 2；另外数组的长度不超过 50。

```

#include<stdio.h>
int main(int argc, char** argv)
{
    int num[50] = { 0 };
    int i, n;

    printf("请输入整型数组的长度(1~50): ");
    scanf("%d", &n);

    printf("请输入整型数组的元素: ");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &num[i]);
    }

    int min_num = num[0];
    int max_num = num[0];
    for (int j = 0; j < n; j++)
    {
        if (max_num < num[j])
        {
            max_num = num[j];
        }
        else if (min_num > num[j])
        {
            min_num = num[j];
        }
    }
    int sum = min_num + max_num;
    printf("数组中最大与最小值之和: %d\n", sum);
    return 0;
}

```

63、【圆桌队列】

一组人 (n 个)，围成一圈，从某人开始数到第三个人出列，再接着从下一个人开始数，最终输出最终出列的人 (约瑟夫环是一个数学的应用问题：已知 n 个人 (以编号 1, 2, 3...n 分别表示) 围坐在一张圆桌周围。从编号为 k 的人开始报数，数到 m 的那个人出列；他的下一个人又从 1 开始报数，数到 m 的那个人又出列；依此规律重复下去，直到圆桌周围的人全部出列。)

```

#include <list>
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

int solve(int n, int k, int m) {
    list<int> data;
    for (int i = 1; i <= n; ++i) data.push_back(i);
    int cnt = 1;
    list<int>::iterator iter = data.begin();

```

```

while (cnt < k) { ++iter; ++cnt; }
while (data.size() > 1) {
    for (int i = 1; i < m; ++i) {
        ++iter;
        if (iter == data.end()) iter = data.begin();
    }
    iter = data.erase(iter);
    if (iter == data.end()) iter = data.begin();
}
return data.front();
}

int main()
{
    int n, k, m;
    cin >> n >> k >> m;
    cout << solve(n, k, m) << endl;
}

```

64、【找出字符串中最小和最大数字】

输入一串字符，只包含“0-10”和“”，找出其中最小的数字和最大的数字（可能不止一个），输出最后剩余数字个数。如输入“3,3,4,5,6,7,7”

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
    char str[100];
    printf("输入一组字符串: \n");
    scanf("%s", &str);
    int len = strlen(str);
    int array[100];
    int count = 0;
    for (int i = 0; i < len; i++)
    {
        if (str[i] >= '0' && str[i] <= '9')
            array[count++] = str[i] - '0';
    }
    array[count] = '\0';
    int result = count;
    int min = array[0];
    int max = array[0];
    for (int j = 0; j < count; j++)
    {
        if (max < array[j])
            max = array[j];
        else if (min > array[j])
            min = array[j];
    }
    for (int k = 0; k < count; k++)
    {
        if (array[k] == min)
            result--;
        if (array[k] == max)
            result--;
    }
    printf("%d\n", result);
}

```

65、【删除子串】

删除子串，只要是原串中有相同的子串就删掉，不管有多少个，返回子串个数。

```

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
int delete_sub_str(const char* str, const char* sub_str, char* result)
{
    assert(str != NULL && sub_str != NULL);
    const char* p, * q;
    char* t, * temp;
    p = str;
    q = sub_str;
    t = result;
    int n, count = 0;
    n = strlen(q);
    temp = (char*)malloc(n + 1);
    memset(temp, 0x00, n + 1);
    while (*p)

```

```

{
    memcpy(temp, p, n);
    if (strcmp(temp, q) == 0)
    {
        count++;
        memset(temp, 0x00, n + 1);
        p = p + n;
    }
    else
    {
        *t = *p;
        p++;
        t++;
        memset(temp, 0x00, n + 1);
    }
}
free(temp);
return count;
}

void main(int argc, char** argv)
{
    char s[100] = { '\0' };
    int num = delete_sub_str("123abc12de234fg1hi34j123k", "123", s);
    printf("Thenumber of sub_str is %d\r\n", num);
    printf("Theresult string is %s\r\n", s);
}

```

66、【高精度十进制加法】

要求编程实现上述高精度的十进制加法。要求实现函数：

void add (const char *num1,const char *num2, char *result)

【输入】num1: 字符串形式操作数 1, 如果操作数为负, 则 num1[0]为符号位 '-' num2: 字符串形式操作数 2, 如果操作数为负, 则 num2[0]为符号位 '-'

【输出】result: 保存加法计算结果字符串, 如果结果为负, 则 result[0]为符号位。

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void move(char* str, int length) //移除字母前的 "-" 符号
{
    if (str[0] != '-')
        return;
    int i;
    for (i = 0; i < length - 1; i++)
        str[i] = str[i + 1];
    str[i] = '\0';
}

int remove_zero(char* result, int length)
{
    int count = 0;
    for (int i = length - 1; i > 0; i--) //从最后开始移除0, 直到遇到非0 数字, 只对最 初位置上的0 不予判断
    {
        if (result[i] == '0')
        {
            result[i] = '\0';
            count++;
        }
        else
            return length - count;
    }
    return length - count;
}

void reverse(char* result, int length) //将字符串倒转
{
    char temp;
    for (int i = 0; i <= (length - 1) / 2; i++)
    {
        temp = result[i];
        result[i] = result[length - 1 - i];
        result[length - 1 - i] = temp;
    }
}

int real_add(char* str1, char* str2, char* result, const bool flag)
{
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int n1, n2, another = 0; //another 表示进位
    int cur_rs = 0; //表示result 的当前位数
    int i, j;
    int curSum;
    for (i = len1 - 1, j = len2 - 1; i >= 0 && j >= 0; i--, j--)
    {
        n1 = str1[i] - '0';

```

```

        n2 = str2[j] - '0';
        curSum = n1 + n2 + another;
        result[cur_rs++] = curSum % 10 + '0';
        another = curSum / 10;
    }
    if (j < 0)
    {
        while (i >= 0) //遍历str1 剩余各位
        {
            n1 = str1[i--] - '0';
            curSum = n1 + another;
            result[cur_rs++] = curSum % 10 + '0';
            another = curSum / 10;
        }
        if (another != 0) //如果还有进位未加上
            result[cur_rs++] = another + '0';
    }
    else
    {
        while (j >= 0)
        {
            n2 = str2[j--] - '0';
            curSum = n2 + another;
            result[cur_rs++] = curSum % 10 + '0';
            another = curSum / 10;
        }
        if (another != 0)
            result[cur_rs++] = another + '0';
    }
    result[cur_rs] = '\0';
    cur_rs = remove_zero(result, cur_rs);
    if (!flag)
    {
        result[cur_rs++] = '-';
        result[cur_rs] = '\0';
    }
    reverse(result, strlen(result));
    return cur_rs;
}

int real_minus(char* str1, char* str2, char* result) //使用str1 减去str2
{
    char big[100], small[100];
    int big_len, sml_len;
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    bool flag = false; //用于标记str2 是否比str1 大
    if (len1 < len2)
        flag = true;
    else if (len1 == len2)
    {
        if (strcmp(str1, str2) == 0)
        {
            result[0] = '0';
            result[1] = '\0';
            return 1;
        }
        else if (strcmp(str1, str2) < 0)
            flag = true;
    }
    if (flag) //将str1 和str2 交换, 确保str1 指向的值是其中较大者, 最后通过flag 确定要不要给前面加 - 号
    {
        char* temp = str1;
        str1 = str2;
        str2 = temp;
        len1 = strlen(str1);
        len2 = strlen(str2);
    }
    int n1, n2, another = 0; //another 表示是否有借位
    int i, j;
    int cur_rs = 0;
    int curMinus;
    for (i = len1 - 1, j = len2 - 1; i >= 0 && j >= 0; i--, j--)
    {
        n1 = str1[i] - '0';
        n2 = str2[j] - '0';
        if (n1 >= n2 + another)
        {
            result[cur_rs++] = (n1 - n2 - another) + '0';
            another = 0;
        }
        else
        {
            result[cur_rs++] = (n1 + 10 - n2 - another) + '0';
            another = 1;
        }
    }
    while (i >= 0)

```

```

{
    n1 = str1[i--] - '0';
    if (another != 0)
    {
        n1 -= another;
        another = 0;
    }
    result[cur_rs++] = n1 + '0';
}
result[cur_rs] = '\0';
cur_rs = remove_zero(result, cur_rs);
if (flag)
{
    result[cur_rs++] = '-';
    result[cur_rs] = '\0';
}
reverse(result, cur_rs);
return cur_rs;
}

void addi(const char* num1, const char* num2, char* result)
{
    int len1 = strlen(num1);
    int len2 = strlen(num2);
    int rs_len;
    if (!len1 || !len2)
        return;
    char str1[100], str2[100];
    strncpy(str1, num1, len1);
    str1[len1] = '\0';
    strncpy(str2, num2, len2);
    str2[len2] = '\0';
    if (str1[0] == '-' && str2[0] == '-')
    {
        move(str1, len1);
        move(str2, len2);
        rs_len = real_add(str1, str2, result, false);
    }
    else if (str1[0] == '-')
    {
        move(str1, len1);
        rs_len = real_minus(str2, str1, result);
    }
    else if (str2[0] == '-')
    {
        move(str2, len2);
        rs_len = real_minus(str1, str2, result);
    }
    else
        rs_len = real_add(str1, str2, result, true);
}

//int main(int argc, char *argv[])
int main(int argc, char** argv)
{
    char num1[100], num2[100];
    printf("请输入两个整型数据: \n");
    scanf_s("%s%s", num1, num2);
    char result[100];
    memset(result, 0, 100);
    addi(num1, num2, result);
    printf("%s\n", result);
    return 0;
}

```

67、【成绩的及格线】

描述: 10 个学生考完期末考试评卷完成后, A 老师需要划出及格线, 要求如下:

及格线是 10 的倍数;

保证至少有 60%的学生及格;

如果所有的学生都高于 60 分, 则及格线为 60 分输入: 输入 10 个整数, 取值 0~100

输出: 输出及格线, 10 的倍数

```

#include<stdio.h>
void bubblesort(int arr[])
{
    int i, j, temp;
    for (i = 0; i < 10; i++)
        for (j = 0; j < 9 - i && arr[j] > arr[j + 1]; j++)
        {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
}

```

```

int GetPassLine(int a[])
{
    bubblesort(a);
    if (a[0] >= 60)
    {
        return 60;
    }
    else
    {
        return(((int)a[4] / 10) * 10);
    }
}
main()
{
    int a[10] = { 0 };
    int result;
    printf("请随机输入10 个成绩 (0-100) : \n");
    scanf("%d%d%d%d%d%d%d%d%d%d", &a[0], &a[1], &a[2], &a[3], &a[4], &a[5], &a[6], &a[7], &a[8], &a[9]);
    printf("\n");
    result = GetPassLine(a);
    printf("及格线为:%d\n", result);
    return 1;
}

```

68、【长廊电灯数】

描述：一条长廊里依次装有 $n(1 \leq n \leq 65535)$ 盏电灯，从头到尾编号 1、2、3、...n-1、n。每盏电灯由一个拉线开关控制。开始，电灯全部关着。

有 n 个学生从长廊穿过。第一个学生把号码凡是 1 的倍数的电灯的开关拉一下；接着第二个学生把号码凡是 2 的倍数的电灯的开关拉一下；接着第三个学生把号码凡是 3 的倍数的电灯的开关拉一下；如此继续下去，最后第 n 个学生把号码凡是 n 的倍数的电灯的开关拉一下。 n 个学生按此规定走完后，长廊里电灯有几盏亮着。注：电灯数和学生数一致。

输入：电灯的数量

输出：亮着的电灯数量样例输入：3

样例输出：1

```

#include<stdio.h>
#define Max_Bubl_Num 65535
int GetLightLampNum(int n)
{
    int BubNum[Max_Bubl_Num] = { 0 }; //0 表示灯灭, 1 表示灯亮
    unsigned int i, j;
    unsigned int count = 0;
    for (i = 1; i <= n; i++)
    {
        for (j = i; j <= n && j % i == 0; j++)
        {
            BubNum[j - 1] += 1;
            BubNum[j - 1] = BubNum[j - 1] % 2;
        }
    }
    for (int k = 0; k < n; k++)
    {
        if (BubNum[k] == 1)
            count++;
    }
    return count;
}
int main(int argc, char** argv)
{
    int n, result;
    printf("请输入灯的数量 (1-65535) :\n");
    scanf_s("%d", &n);
    result = GetLightLampNum(n);
    printf("最后亮灯的数量为:%d\n", result);
    return 0;
}

```

69、【乘地铁站数】

题目描述：已知 2 条地铁线路，其中 A 为环线，B 为东西向线路，线路都是双向的。经过的站点名分别如下，两条线交叉的换乘点用 T1、T2 表示。编写程序，任意输入两个站点名称，输出乘坐地铁最少需要经过的车站数量（含输入的起点和终点，换乘站点只计算一次）。

地铁线 A（环线）经过车站：A1 A2 A3 A4 A5 A6A7 A8 A9 T1 A10 A11 A12 A13 T2

A14 A15 A16 A17 A18

地铁线 B（直线）经过车站：B1 B2 B3 B4 B5 T1B6 B7 B8 B9 B10 T2 B11 B12 B13 B14 B15

输入描述：输入两个不同的站名

输出描述: 输出最少经过的站数,含输入的起点和终点, 换乘站点只计算一次输入样例: A1 A3

输出样例: 3

```
#include<stdio.h>
#include<string>
#include<queue>
#include<vector>
using namespace std;
#define MAX 35
#define SUBWAY_A 20
#define SUBWAY_B 15
typedef struct node
{
    int adjvex;
    struct node* next;
}edgenode;

typedef struct
{
    char name[10];
    bool flag;
    edgenode* link;
}vexnode;
const char subway_name1[SUBWAY_A][10] =
{ "A1", "A2", "A3", "A4", "A5", "A6", "A7", "A8", "A9", "T1", "A10", "A11", "A12", "A13", "T2", "A14", "A15", "A16", "A17", "A18" };
const char subway_name2[SUBWAY_B][10] =
{ "B1", "B2", "B3", "B4", "B5", "B6", "B7", "B8", "B9", "B10", "B11", "B12", "B13", "B14", "B15" };
void creat(vexnode ga[])
{
    int i;
    edgenode* p;
    for (i = 0; i < MAX; i++)
    {
        ga[i].link = NULL;
        ga[i].flag = true;
        if (i < SUBWAY_A)strcpy(ga[i].name, subway_name1[i]);
        else
            strcpy(ga[i].name, subway_name2[i - 20]);
    }
    //A 地铁建邻接表
    for (i = 1; i < SUBWAY_A - 1; i++)
    {
        p = (edgenode*)malloc(sizeof(edgenode));
        p->adjvex = i - 1;
        p->next = NULL;
        ga[i].link = p;
        p = (edgenode*)malloc(sizeof(edgenode));
        p->adjvex = i + 1;
        p->next = NULL;
        ga[i].link->next = p;
        if (i == 9)
        {
            p = (edgenode*)malloc(sizeof(edgenode));
            p->adjvex = SUBWAY_A + 4;
            p->next = NULL;
            ga[i].link->next->next = p;
            p = (edgenode*)malloc(sizeof(edgenode));
            p->adjvex = SUBWAY_A + 5;
            p->next = NULL;
            ga[i].link->next->next->next = p;
        }
        else if (i == 14)
        {
            p = (edgenode*)malloc(sizeof(edgenode));
            p->adjvex = SUBWAY_A + 9;
            p->next = NULL;
            ga[i].link->next->next = p;
            p = (edgenode*)malloc(sizeof(edgenode));
            p->adjvex = SUBWAY_A + 10;
            p->next = NULL;
            ga[i].link->next->next->next = p;
        }
    }
    p = (edgenode*)malloc(sizeof(edgenode));
    p->adjvex = SUBWAY_A - 1;
    p->next = NULL;
    ga[0].link = p;
    p = (edgenode*)malloc(sizeof(edgenode));
    p->adjvex = 1;
    p->next = NULL;
    ga[0].link->next = p;
    p = (edgenode*)malloc(sizeof(edgenode));
    p->adjvex = SUBWAY_A - 2;
    p->next = NULL;
    ga[SUBWAY_A - 1].link = p;
    p = (edgenode*)malloc(sizeof(edgenode));
    p->adjvex = 0;
```

```

p->next = NULL;
ga[SUBWAY_A - 1].link->next = p;
//B 地铁建邻接表
for (i = 1; i < SUBWAY_B - 1; i++)
{
    if (i == 4 || i == 5 || i == 9 || i == 10)
        continue;
    p = (edgenode*)malloc(sizeof(edgenode));
    p->adjvex = SUBWAY_A + i - 1;
    p->next = NULL;
    ga[i + SUBWAY_A].link = p;
    p = (edgenode*)malloc(sizeof(edgenode));
    p->adjvex = SUBWAY_A + i + 1;
    p->next = NULL;
    ga[i + SUBWAY_A].link->next = p;
}
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = SUBWAY_A + 3;
p->next = NULL;
ga[SUBWAY_A + 4].link = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = 9;
p->next = NULL;
ga[SUBWAY_A + 4].link->next = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = 9;
p->next = NULL;
ga[SUBWAY_A + 5].link = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = SUBWAY_A + 6;
p->next = NULL;
ga[SUBWAY_A + 5].link->next = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = SUBWAY_A + 8;
p->next = NULL;
ga[SUBWAY_A + 9].link = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = 14;
p->next = NULL;
ga[SUBWAY_A + 9].link->next = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = 14;
p->next = NULL;
ga[SUBWAY_A + 10].link = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = SUBWAY_A + 11;
p->next = NULL;
ga[SUBWAY_A + 10].link->next = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = SUBWAY_A + 1;
p->next = NULL;
ga[SUBWAY_A].link = p;
p = (edgenode*)malloc(sizeof(edgenode));
p->adjvex = SUBWAY_A + SUBWAY_B - 2;
p->next = NULL;
ga[SUBWAY_A + SUBWAY_B - 1].link = p;
// 打印各邻接节点
for (i = 0; i < MAX; i++)
{
    printf("%s:", ga[i].name);
    edgenode* s;
    s = ga[i].link;
    while (s != NULL)
    {
        printf("->%s", ga[s->adjvex].name);
        s = s->next;
    }
    printf("\n");
}
}

int main(int argc, char** argv)
{
    vexnode ga[MAX];
    creat(ga);
    int i;
    char str[2][10];
    while (scanf("%s%s", str[0], str[1]) != EOF)
    {
        int temp = 0;
        for (i = 0; i < MAX; i++) {
            ga[i].flag = true;
            if (!strcmp(str[0], ga[i].name))temp = i;
        }
        queue<vexnode>q;
        q.push(ga[temp]);
        ga[temp].flag = false;
        int count = 0;
    }
}

```



```

int start = 0;
int end = 1;
bool find_flag = false;
while (!q.empty())
{
    if (find_flag)
        break;
    count++;
    printf("*****\n");
    printf("第%d 层搜索: ", count);
    int temp_end = end;
    while (start < temp_end)
    {
        printf("%s", q.front().name);
        if (!strcmp(q.front().name, str[1]))
        {
            find_flag = true;
            break;
        }
        edgenode* s;
        s = q.front().link;
        while (s != NULL)
        {
            if (ga[s->adjvex].flag)
            {
                q.push(ga[s->adjvex]);
                ga[s->adjvex].flag = false;
                end++;
                //printf("%s ", ga[s->adjvex].name);
            }
            s = s->next;
        }
        q.pop();
        start++;
    }
    printf("\n");
    printf("%d\n", count);
}
return 0;
}

```

70、【字串转换 a->b,b->c】

问题描述:

将输入的字符串(字符串仅包含小写字母'a'到'z'),按照如下规则,循环转换后输出: a->b,b->c,...,y->z,z->a; 若输入的字符串连续出现两个字母相同时,后一个字母需要连续转换 2 次。例如: aa 转换为 bc, zz 转换为 ab; 当连续相同字母超过两个时,第三个出现的字母按第一次出现算。

要求实现函数:

void convert(char *input,char* output)

【输入】 char *input, 输入的字符串

【输出】 char *output, 输出的字符串

【返回】 无#include<stdio.h> #include<string.h> #include<stdlib.h>

```

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void convert(char* input, char* output)
{
    if (input == NULL)
        return;
    char temp = '\0';
    int len_input = strlen(input);
    int i;
    int flag = 0;
    for (i = 0; i < len_input; i++)
    {
        if (input[i] != temp)
        {
            output[i] = (input[i] - 'a' + 1) % 26 + 'a';
            temp = input[i];
            flag = 1;
        }
        else
        {
            if (flag == 1)
            {
                output[i] = (input[i] - 'a' + 2) % 26 + 'a';
                temp = input[i];
                flag = 0;
            }
        }
    }
}

```

```

        else
        {
            output[i] = (input[i] - 'a' + 1) % 26 + 'a';
            temp = input[i];
            flag = 1;
        }
    }
    output[i] = '\0';
}
void main()
{
    char* input = "xyz";
    char output[256];
    // scanf("%s",input);
    convert(input, output);
    printf("%s\n", output);
}

```

71、【在字符串中找单词】

在给定字符串中找出单词（“单词”由大写字母和小写字母字符构成，其他非字母字符视为单词的间隔，如空格、问号、数字等等；另外单个字母不算单词）：找到单词后，按照长度进行降序排序，（排序时如果长度相同，则按出现的顺序进行排列），然后输出到一个新的字符串中；如果某个单词重复出现多次，则只输出一次；如果整个输入的字符串中没有找到单词，请输出空串。输出的单词之间使用一个“空格”隔开，最后一个单词后不加空格。

要求实现函数：

void my_word(char input[], char output[])

【输入】 char input[], 输入的字符串

【输出】 char output[], 输出的字符串

【返回】 无

```

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
void my_word(char input[], char output[])
{
    char* p;
    char* temp;
    char* word[10];
    int len_input = strlen(input);
    int i, j;
    char except[] = ",";
    char* blank = " ";
    i = 0;
    for (i = 0; i < len_input; i++)
    {
        if (input[i] < 'A' || (input[i] > 'Z' && input[i] < 'a') || input[i] > 'z')
        {
            input[i] = ',';
        }
    }
    j = 0;
    /*保存取出的单词*/
    p = strtok(input, except);
    while (NULL != p)
    {
        word[j++] = p;
        p = strtok(NULL, except);
    }
    for (i = 0; i < 5; i++)
        printf("%s", word[i]);
    /*对单词按照长度降序排序，冒泡法*/
    for (i = 0; i < 5; i++)
    {
        for (j = 1; j < 5 - i; j++)
        {
            if (strlen(word[j - 1]) < strlen(word[j]))
            {
                temp = word[j];
                word[j] = word[j - 1];
                word[j - 1] = temp;
            }
        }
    }
    /*删除相同单词*/
    for (i = 0; i < 5; i++)
    {
        for (j = i + 1; j < 5; j++)
        {
            if (strcmp(word[i], word[j]) == 0)
                word[j] = "\0";
        }
    }
}

```

```

    }
    /*将单词连接起来输出*/
    for (j = 0; j < 5; j++)
    {
        if (0 == j)
        {
            strncpy(output, word[j], strlen(word[j]) + 1);
        }
        else
        {
            strcat(output, blank);
            strcat(output, word[j]);
        }
    }
    return;
}
int main()
{
    char input[] = "some local buses, some1234123drivers";
    printf("筛选之前的字符串:%s\n", input);
    char output[30];
    my_word(input, output);
    printf("筛选之后的字符串:%s", output);
    printf("\n");
    return 0;
}

```

72、【数组中不同的一个数】

数组中数字都两两相同，只有一个不同，找出该数字：

```

int findUnique(int* a, int len)
{
    int i = 1;
    int temp = a[0];
    for (; i < len; i++)
    {
        temp = temp ^ a[i];
    }
    printf("%d", temp);
}

```

73、【找到数组中相同的两个数】

数组中数字两两相同，有两个不同，找出这两个：

```

#include <stdlib.h>
#include <stdio.h>
int a[] = { 1,1,2,4,3,3,2,5 };
int findXorSum(int* a, int len)
{
    int i = 0;
    int temp = 0;
    for (; i < len; i++)
    {
        temp = temp ^ a[i];
    }
    return temp;
}
int findFirstBit1(int n)
{
    int count = 1;
    while (!(n & 1))
    {
        n = n >> 1;
        count++;
    }
    return count;
}
int isBit1(int a, int count)
{
    a = a >> count - 1;
    return(a & 1);
}
void findTwoUnique(int* a, int len)
{
    int i = 0;
    int m = 0, n = 0;
    int temp = findXorSum(a, len);
    int count = findFirstBit1(temp);
    for (; i < len; i++)
    {
        if (isBit1(a[i], count))

```

```

        {
            m = m ^ a[i];
        }
        else
        {
            n = n ^ a[i];
        }
    }
    printf("%d,%d", m, n);
}
int main()
{
    findTwoUnique(a, 8);
}

```

74、【链表翻转】

给出一个链表和一个数 k，比如链表 1→2→3→4→5→6，k=2，则翻转后 2→1→4→3→6→5，若 k=3，翻转后 3→2→1→6→5→4，若 k=4，翻转后 4→3→2→1→5→6，用程序实现

思想：采用遍历链表，分成 length/k 组，对每组进行逆转，逆转的同时要将逆转后的尾和头连接起来

```

#include "stdio.h"
#include "stdlib.h"
#include <malloc.h>
typedef struct Node {
    int value;
    Node* next;
}LinkedList;
void Converse(LinkedList* pPre, LinkedList* pCur)
{ //链表逆转
    LinkedList* p = NULL;
    LinkedList* pNext = NULL;
    p = pPre->next;
    LinkedList* p1 = NULL;
    if (pCur != NULL)
        pNext = pCur->next;
    while (p != pNext)
    {
        p1 = p->next;
        p->next = pPre;
        pPre = p;
        p = p1;
    }
}

int main()
{
    int count = 0, k, i = 0, j = 0, flag = 1, length = 0, groups = 0;
    scanf("%d", &k);
    LinkedList* pPre = (LinkedList*)malloc(sizeof(LinkedList));
    LinkedList* pCur = (LinkedList*)malloc(sizeof(LinkedList));
    LinkedList* pNext = (LinkedList*)malloc(sizeof(LinkedList));
    LinkedList* head = NULL;
    LinkedList* pTempTail = NULL; //指向逆转之后的尾部
    LinkedList* pTempHead = NULL;
    pCur->value = 1;
    pPre = pCur; //创建初始链表
    for (i = 2; i <= 6; i++) {
        LinkedList* node = (LinkedList*)malloc(sizeof(LinkedList));
        node->value = i;
        pCur->next = node;
        pCur = node;
    }
    pCur->next = NULL; //最后一定要置NULL，c++中用new 则无须置NULL
    pCur = pPre;
    while (pCur != NULL)
    {
        length++;
        pCur = pCur->next;
    }
    i = 0;
    groups = length / k; //分成K 段
    pCur = pPre;
    while (i <= groups)
    {
        count = 0;
        while (count < k - 1 && i < groups)
        {
            pCur = pCur->next;
            count++;
        }
        if (i < groups)

```

```

    {
        pNext = pCur->next;
        pTempHead = pCur; /*没做翻转之前的头部,变成了翻转之后的尾
        部*/
        if (flag == 0)
        {
            pTempTail->next = pTempHead;
        }
        pTempTail = pPre;
        Converse(pPre, pCur);
        //pTempTail= pPre;
        if (flag == 1)
        {
            head = pCur;
            flag = 0;
        }
        pCur = pNext;
    }
    else
    {
        pTempTail->next = pNext;
    }
    pPre = pCur;
    i++;
}
pCur = head;
while (j < length)
{
    j++;
    printf("%d", pCur->value);
    pCur = pCur->next;
}
printf("\n");
// system("pause");
return 0;
}

```

75、【找到含有相同字符的子串】

将第一行中含有第二行中"23"的数输出并排序

输入一行数字: 123 423 5645 875 186523

在输入第二行: 23

将第一行中含有第二行中"23"的数输出并排序结果即: 123 423 186523

```

#include<stdio.h>
#define M 20
int main()
{
    int a[M];
    int i, j, s, temp;
    int sort[M], t = 0;
    char c = ' ';
    i = 0;
    while (c != '\n')
    {
        scanf("%d%c", &temp, &c);
        a[i++] = temp;
    }
    scanf("%d", &s);
    for (j = 0; j < i; j++)
    {
        temp = a[j];
        if (temp % 100 == s)
        {
            sort[t++] = a[j];
        }
        else
            temp /= 10;
    }
    for (i = 0; i < t - 1; i++)
        for (j = 0; j < t - i - 1; j++)
        {
            if (sort[j] > sort[j + 1])
            {
                temp = sort[j + 1];
                sort[j + 1] = sort[j];
                sort[j] = temp;
            }
        }
    for (i = 0; i < t; i++)
        printf("%d", sort[i]);
    printf("\n");
}

```

```
}    return 0;
```

76、【字符串以 N 为单位分段】

输入 m 个字符串和一个整数 n, 把字符串 M 化成以 N 为单位的段, 不足的位数用 0 补齐。

如 n=8 m=9 ,

123456789 划分为: 12345678

90000000

123 化为: 12300000

2 题

77、【电话号码翻译】

将电话号码 one two.....nine zero

翻译成 1 29 0

中间会有 double

例如输入: OneTwoThree 输出: 123

输入: OneTwoDoubleTwo 输出: 1222

输入: 1Two2 输出: ERROR

输入: DoubleDoubleTwo 输出: ERROR

有空格, 非法字符, 两个 Double 相连, Double 位于最后一个单词都错误

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    char a[11][11] = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "double" };
    char temp[11], c = ' ';
    int i, j, f, d = 0;
    while (c != '\n')
    {
        scanf("%s%c", &temp, &c);
        f = 0;
        for (j = 0; j < 11; j++)
        {
            if (!strcmp(temp, a[j]) && j < 10)
            {
                printf("%d", j);
                f = 1;
                if (d == 1)
                {
                    printf("%d", j);
                    d = 0;
                }
            }
            else if (!strcmp(temp, a[j]) && j == 10)
            {
                d = 1;
                f = 1;
            }
        }
        if (f == 0)
            break;
        if (d == 1 || f == 0)
            printf("error\n");
        printf("\n");
        return 0;
    }
}
```

78、【将整数倒序输出，剔除重复数据】

输入一个整数, 如 12336544, 或 1750. 然后从最后一位开始倒过来输出, 最后如果是 0, 则不输出, 输出的数字是不带重复数字的, 所以上面的输出是 456321 和 571。如

如果是负数, 比如输入-175, 输出-571。

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<malloc.h>
int main()
```

```

{
    char* input = (char*)malloc(sizeof(char));
    scanf("%s", input);
    int a[10] = { 0 }, i, flag = 0, flag1 = 0;
    int len = strlen(input);
    if (input[0] == '-')
    {
        flag = 1;
        for (i = 0; i < len; i++)
            input[i] = input[i + 1];
    }
    int len1 = strlen(input);
    int n[50], temp;
    int count = 0;
    for (i = 0; i < len1; i++)
    {
        temp = input[i] - '0';
        if (a[temp] == 0)
        {
            n[count++] = temp;
            a[temp] = 1;
        }
    }
    n[count] = '\0';
    if (flag == 1)
        printf("-");
    for (int ii = count - 1; ii >= 0; ii--)
    {
        if (n[ii] != 0 || flag1 != 0)
        {
            printf("%d", n[ii]);
            flag1 = 1;
        }
    }
    printf("\n");
    return 0;
}

```

79、【括号匹配】

编程的时候，if 条件里面的“(”、“)”括号经常出现不匹配的情况导致编译不过，请编写程序检测输入一行 if 语句中的圆括号是否匹配正确。同时输出语句中出现的左括号和右括号数量，如 if((a==1)&&(b==1))是正确的，而 if((a==1))&&(b==1)) 是错误的。注意 if 语句的最外面至少有一对括号。提示：用堆栈来做。

输入：if((a==1)&&(b==1)) 输出：RIGH 3 3

输入：if((a==1))&&(b==1)) 输出：WRONG 3 4

```

#include<stdio.h>
#include<string.h>
int main()
{
    char s[800] = { '\0' };
    scanf_s("%s", &s);
    // char s[]="if(());";
    int len = strlen(s);
    int i, left = 0, right = 0;
    int a[50], k = 0, flag = 1;
    for (i = 0; i < len; i++)
    {
        if (s[i] == '(')
        {
            left++;
            a[k] = 1;
            k++;
        }
        else if (s[i] == ')')
        {
            right++;
            if (a[k - 1] == 1 && k > 0)
            {
                a[k - 1] = 0;
                k--;
            }
            else
                flag = 0;
        }
        if ((i == 2 && s[i] != '(') || (i == len - 1 && s[i] != ')'))
            flag = 0;
    }
    if (a[0] == 0 && flag != 0)
        printf("RIGHT");
    else
        printf("WRONG");
    printf("%d%d\n", left, right);
    return 0;
}

```

```
}
```

80、【表达式括号匹配】

$(1+(2+3)*(3+(8+0))+1-2)$ 这是一个简单的数学表达式,今天不是计算它的值,而是比较它的括号匹配是否正确。前面这个式子可以简化为 $(0(0))$ 这样的括号我们认为它是匹配正确的,而 $((0))$ 这样的我们就说他是错误的。注意括号里面的表达式可能是错的,也可能有多个空格,对于这些我们是不用去管的,我们只关心括号是否使用正确。

输入:

给出一行表达式(长度不超过100)。

输出:

如果匹配正确输出括号的对数,否则输出-1

样例

输入样例 1

$(1+(2+3)*(3+(8+0))+1-2)$

输出样例 1

4

```
#include<iostream>
#include <string>
#include <stack>

using namespace std;

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        string temp = "";

        for (int i = 0; i < input_string.size(); i++)
        {
            if ((input_string[i] == '(') || (input_string[i] == '('))
            {
                temp += input_string[i];
            }
        }
        if (temp.empty())
        {
            cout << -1 << endl;
            continue;
        }
        stack<char> c_stack;
        int count = 0;
        bool count_flag = false;
        for (int i = 0; i < temp.size(); i++)
        {
            if (temp[i] == '(')
            {
                c_stack.push(temp[i]);
            }
            else
            {
                if (!c_stack.empty())
                {
                    c_stack.pop();
                    count++;
                }
                else
                {
                    count_flag = true;
                    break;
                }
            }
        }
        if (c_stack.empty() && !count_flag)
        {
            cout << count << endl;
        }
        else
        {
            cout << -1 << endl;
        }
    }

    return 0;
}
```


81、【括号匹配】

给定一个字符串，里边可能包含“()”、“[]”、“{}”三种括号，请编写程序检查该字符串中的括号是否成对出现，且嵌套关系正确。

输出: true:若括号成对出现且嵌套关系正确，或该字符串中无括号字符;

false:若未正确使用括号字符。

实现时，无需考虑非法输入。

输入描述:

输入为:

字符串

例子: (1+2)/(0.5+1)

输出描述:

输出为:

字符串

例子: true

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

int whatType(char ch)
{
    if (ch == '[' || ch == '{' || ch == '(')
    {
        return -1;
    }
    else if (ch == ']' || ch == '}' || ch == ')')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int ifFit(char a, char b)
{
    if (a == '{' && b == '}') return 1;
    if (a == '[' && b == ']') return 1;
    if (a == '(' && b == ')') return 1;
    return 0;
}

int main(int argc, char** argv)
{
    vector<char> characters;
    string a;
    cin >> a;
    for (int i = 0; i < a.Length(); ++i)
    {
        if (whatType(a[i]) == -1) {
            characters.push_back(a[i]);
        }
        else if (whatType(a[i]) == 1)
        {
            if (characters.empty())
            {
                cout << "false"; return 0;
            }
            else
            {
                if (ifFit(characters[characters.size() - 1], a[i]) == 0)
                {
                    cout << "false";
                    return 0;
                }
                else
                {
                    characters.pop_back();
                }
            }
        }
    }
    if (characters.empty())
    {
        return 0;
    }
}
```

```

        cout << "true";
    }
    else
    {
        cout << "false";
    }
    return 0;
}

```

82、【约瑟夫问题】

输入一个由随机数组成的数列（数列中每个数均是大于 0 的整数，长度已知），和初始计数值 m。从数列首位置开始计数，计数到 m 后，将数列该位置数值替换计数值 m，并将数列该位置数值出列，然后从下一位置重新开始计数，直到数列所有数值出列为止。如果计数到达数列尾段，则返回数列首位置继续计数。请编程实现上述计数过程，同时输出数值出列的顺序

比如：输入的随机数列为：3,1,2,4，初始计数值 m=7，从数列首位置开始计数（数值 3 所在位置）

第一轮计数出列数字为 2，计数值更新 m=2，出列后数列为 3,1,4，从数值 4 所在位置重新开始计数

第二轮计数出列数字为 3，计数值更新 m=3，出列后数列为 1,4，从数值 1 所在位置开始计数

第三轮计数出列数字为 1，计数值更新 m=1，出列后数列为 4，从数值 4 所在位置开始计数

最后一轮计数出列数字为 4，计数过程完成。输出数值出列顺序为：2,3,1,4。

要求实现函数：

void array_iterate(int len, int input_array[], int m, int output_array[])

【输入】 int len: 输入数列的长度； int input_array[]: 输入的初始数列 int m: 初始计数值

【输出】 int output_array[]: 输出的数值出列顺序

【返回】 无示例:

输入: int input_array[] = {3,1,2,4}, int len = 4, m=7

输出: output_array[] = {2,3,1,4}

解题思路:

每次出列一个数值，需要对 m、input_array、output_array、输出位置 outPos、起始位置 startPos 进行更新；

对于输出位置 outPos 的计算是关键！通过分析可知，outPos=(startPos+m-1)%num

代码实现:

```

#include<stdio.h>
void print_array(int len, int array[])
{
    for (int i = 0; i < len; i++)
        printf("%d ", array[i]);
    printf("\n");
}
//input_array:a[0]...a[len-1]
void array_iterate(int len, int input_array[], int m, int output_array[])
{
    int startPos = 0;
    int outPos;
    int nIter = len - 1;
    int num = len;
    for (; nIter >= 0; nIter--)
    {
        outPos = (m + startPos - 1) % num;
        m = input_array[outPos];
        startPos = outPos;
        printf("outPos is %d, new m is %d\n", outPos, m);
        output_array[len - nIter - 1] = input_array[outPos];
        for (int i = outPos; i < num - 1; i++)
            input_array[i] = input_array[i + 1];
        num--;
        print_array(num, input_array);
    }
}

int main(int argc, char** argv)
{
    int input_array[] = { 3,1,2,4 };
    int output_array[4] = { 0 };
    array_iterate(4, input_array, 7, output_array);
    print_array(4, output_array);
    return 0;
}

```

83、【字符串出现次数最多的字符及次数】

统计数字出现的次数，最大次数的统计出来举例：

输入：323324423343

输出：3,6

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(int argc, char** argv)
{
    char* num = "323324423343";
    int a[10] = { 0 };
    int len = strlen(num);
    int i, j, temp, count = 0, maxnum = 0;
    printf("%d\n", len);
    for (i = 0; i < len; i++)
    {
        temp = num[i] - '0';
        a[temp]++;
    }
    int temp1 = a[0];
    for (j = 0; j < 10; j++)
    {
        if (a[j] != 0)
        {
            count++;
            temp1 = (temp1 > a[j]) ? temp1 : a[j];
            printf("%d%d\n", a[j], j);
        }
    }
    printf("数字出现次数为:%d\n", count);
    printf("最大次数为:%d\n", temp1);
    return 0;
}
```

84、【字符串单词首字母转换成大写】

举例：

输入：this is a book

返回：This Is A Book

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(int argc, char** argv)
{
    char input[] = "this is a book";
    char output[256] = { '\0' };
    int i, len;
    len = strlen(input);
    printf("变换前的字符串为:%s\n", input);
    for (i = 0; i < len; i++)
    {
        if (input[i] != ' ')
            input[i] -= 32;
        if (input[i] == ' ')
            input[i + 1] -= 32;
        output[i] = input[i];
    }
    printf("变换后的字符串为:%s\n", output);
    return 0;
}
```

85、【子串分离】

题目描述：

通过键盘输入任意一个字符串序列，字符串可能包含多个子串，子串以空格分隔。请编写一

个程序，自动分离出各个子串，并使用' '将其分隔，并且在最后也补充一个' '并将子

串存储。

如果输入"abc def gh i d"，结果将是 abc,def,gh,i,d,

要求实现函数：

void DivideString(const char *pInputStr, long lInputLen, char *pOutputStr);

【输入】 pInputStr: 输入字符串

lInputLen: 输入字符串长度

【输出】 pOutputStr: 输出字符串，空间已经开辟好，与输入字符串等长；

```
#include <stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
void DivideString(const char* pInputStr, long lInputLen, char* pOutputStr)
{
    int cnt;
    const char* p = pInputStr;
    while (*p != NULL)
    {
        if (*p != ' ')
        {
            cnt = 0;
            *pOutputStr++ = *p++;
        }
        else
        {
            cnt++;
            p++;
            if (cnt == 1)
                *pOutputStr++ = ',';
        }
    }
    *pOutputStr++ = ',';
    *pOutputStr = '\\0';
}
void main()
{
    const char* str = "abcdef gh i d";
    long len = strlen(str);
    char* outstr = (char*)malloc(sizeof(str));
    //char outstr[100];
    DivideString(str, len, outstr);
    printf("%s", outstr);
    printf("\\n");
}
```

86、【最大时间】

题目描述:

给定一个数组,里面有 6 个整数,求这个数组能够表示的最大 24 进制的时间是多少,输出这个时间,无法表示输出 invalid。

输入描述:

输入为一个整数数组,数组内有六个整数。输入整数数组长度为 6,不需要考虑其它长度,元素值为 0 或者正整数,6 个数字每个数字只能使用一次。

输出描述:

输出为一个 24 进制格式的时间,或者字符串"invalid"

备注:输出时间格式为 xxxxx 式。

示例 1:

输入[0,2,3,0,5,6]

输出 23:56:00

```
#include <iostream>
#include <unordered_map>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void SplitInt(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}
```

```

    }
}

string LargestTimeFromDigits(vector<int>& vec)
{
    sort(vec.begin(), vec.end());
    string maxTime = "";
    bool hasNextPermutation = true;
    while (hasNextPermutation)
    {
        string hour = to_string(vec[0]) + to_string(vec[1]);
        string minutes = to_string(vec[2]) + to_string(vec[3]);
        string seconds = to_string(vec[4]) + to_string(vec[5]);
        hasNextPermutation = next_permutation(vec.begin(), vec.end());
        auto curTime = hour + ":" + minutes + ":" + seconds;
        if (stoi(hour) >= 24 || stoi(minutes) >= 60 || stoi(seconds) >= 60)
        {
            continue;
        }
        maxTime = max(maxTime, curTime);
    }

    return maxTime;
}

int main(int argc, char** argv)
{
    string input;
    while (getline(cin, input))
    {
        if (!input.empty())
        {
            input = input.substr(1, input.size() - 1);
            vector<int> vec;
            SplitInt(input, vec, ',');
            string time_s = LargestTimeFromDigits(vec);
            if (!time_s.empty())
            {
                cout << time_s << endl;
            }
            else
            {
                cout << "invalid" << endl;
            }
        }
    }
    return 0;
}

```

87、【数组拼接】

现在有多组整数数组,需要将他们合并成一个新的数组,合并规则,从每个数组里按顺序取出固定长度的内容合并到新的数组中,取完的内容会删除掉,如果该行不足固定长度或者已经为空,则直接取出剩余部分的内容放到新的数组中,继续下一行。

如样例 1, 获得长度 3,遍历第一行,获得 2, 5, 6;再遍历第二行,获得 1, 7, 4;再循环回到第一行,获得 7, 9, 5;再遍历第二行,获得 3, 4;再回到第一行,获得 7,按顺序拼接成最终结果。

输入描述:

第一行是每次读取的固定长度,长度>0

第 2-n 行是需要合并的数组,不同的数组用回车换行分隔,数组内部用逗号分隔

输出描述:

输出一个新的数组, 用逗号分隔。

示例 1:

输入

3

2,5,6,7,9,5,7

1,7,4,3,4

输出

2,5,6,1,7,4,7,9,5,3,4,7

```

#include <iostream>
#include <string>
#include <stdio.h>
#include <queue>

using namespace std;

```

```

int main(int argc, char** argv)
{
    queue<int> a[10000];
    int res[100005];
    int num;
    char ch;
    int cnt = 0;
    string str;

    int k;
    cin >> k;
    while (scanf_s("%d%c", &num, &ch) != EOF)
    {
        a[cnt].push(num);
        if (ch == '\n')
        {
            cnt++;
        }
    }
    num = 0;
    while (true)
    {
        int flag = 0;
        for (int i = 0; i < cnt; i++) //cnt代表输入的数组的个数
        {
            int l = 0;
            while (!a[i].empty() && l < k)
            {
                flag = 1;
                int ans = a[i].front();
                a[i].pop();
                l++;
                res[num++] = ans;
            }
        }
        if (flag == 0)
            break;
    }

    for (int i = 0; i < num; i++) //以 1, 2, 3, 4 的格式输出合并后的新数组
    {
        cout << res[i];
        if (i == num - 1)
            printf("\n");
        else
            printf(",");
    }

    return 0;
}

```

88、【拼接 URL】

题目描述:

给定一个 URL 前缀和 URL 后缀,需要将其连接为一个完整的 URL,如果前缀结尾和后缀开头都没有 “/”,则需自动补上 “/” 连接符, 如果前缀结尾和后缀开头都有 “/”, 需自动去重。

例如:

1)前缀:/a/;后缀:/b/;连接后 URL 为:/a/b,

2)前缀:/a/;后缀:/b; 连接后 URL 为:/a/b,

3)前缀:/a;后缀:b;连接后 URL 为:/a/b,

4)前缀为空;后缀为空;连接后 URL 为:/

约束:不用考虑前后缀 URL 不合法情况

输入描述:

URL 前缀 (一个小于 100 的字符串), URL 后缀 (一个小于 100 的字符串)

输出描述:

拼接后的 URL.

示例 1

输入

/a/b

输出

/a/b

```

#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

```

```

using namespace std;

/*函数声明*/
static int SpliceString(string prefix, string suffix, string& splice_url);
int SpliceURL();

static int SpliceString(string prefix, string suffix, string& splice_url)
{
    if ((prefix == "") && (suffix == ""))
    {
        splice_url = "/";
        return -1;
    }
    string prefix_temp, suffix_temp;
    int prefix_pos = prefix.find_first_of("/");
    int suffix_pos = suffix.find_first_of("/");

    if ((prefix_pos != -1))
    {
        prefix.erase(remove(prefix.begin(), prefix.end(), '/'), prefix.end());
    }
    if ((suffix_pos != -1))
    {
        suffix.erase(remove(suffix.begin(), suffix.end(), '/'), suffix.end());
    }

    if (prefix != "")
    {
        prefix_temp = "/" + prefix;
    }
    if (suffix != "")
    {
        suffix_temp = "/" + suffix;
    }

    splice_url = prefix_temp + suffix_temp;

    return 1;
}

int SpliceURL()
{
    string url_string, prefix_string, suffix_string, splice_url;
    while (1)
    {
        printf(">>");
        getline(cin, url_string);
        prefix_string = url_string.substr(0, url_string.find_first_of(","));
        suffix_string = url_string.erase(0, url_string.find_last_of(',') + 1);
        SpliceString(prefix_string, suffix_string, splice_url);
        printf("%s\n", splice_url.c_str());
    }
    return 1;
}

```

89、【科学计算器】

实现一个科学计算器,完成基本的计算功能.

输入描述:

加减乘除的输入数据

输出描述:

计算结果

示例 1:

输入

-1*6

输出

-6

```

#include<stdio.h>
int main(int argc, char** argv)
{
    int i, sum, flag = 0;
    char op = '0';
    scanf_s("%d", &sum);
    while (scanf_s("%c", &op) && op != '\n') //输入的是运算符
    {
        scanf_s("%d", &i);
    }
}

```

```

        if (op == '\\n')
        {
            break;
        }
        else if (op == '+')
        {
            sum = sum + i;
        }
        else if (op == '-')
        {
            sum = sum - i;
        }
        else if (op == '*')
        {
            sum = sum * i;
        }
        else if (op == '/')
        {
            if (i == 0)
            {
                flag = 1;
            }
            else
            {
                sum = sum / i;
            }
        }
        else
        {
            flag = 1;
        }
    }
    if (flag == 1)
    {
        printf("ERROR");
    }
    else
    {
        printf("%d", sum);
    }
    return 0;
}

```

90、【日期工具函数】

某公司每月要举行一次或者多次聚餐,由团队领导随机挑选当月的一周的某一天,公司档案记录的是某年某月第几个周周几这样的记录,有一天领导希望能够整理下,将这些记录转换为公历白期,由于历史记录众多,秘书 MM 觉得手动去查很辛苦,需要身为程序员的你帮忙解决下,将这些记录换为公历日期。

其中有一部分记录记错了,比如 2018 年 2 月第 8 周周一:对于这样不可能存在的也要进行异常处理;当输入不合法时输出 0:

每月的第一周和最后一周一般都不完整;例如第一周可能没有周一,周二,最后一周可能没有周六,周日;在这种异常情况下,要求输出第一周的周一,显然不存在,这个时候需要按异常处理,输出 0。

2000 年 1 月 1 日为周六;为了简化题目,测试用例的年份均大于 2000 年,年份的取值范围为 2000-9999

另:闰年 2 月份是 29 天; 闰年:被 4 整除且不能被 100 整除的年份,或者能被 400 整除的年份为闰年。

输入描述:

任意的年月周周几

输入格式为

YYYY 空格 mm 空格 W 空格 D

YYYY 表示年,空格表示空格符,mm 表示月份(1-12),W 表示第几周(1-9),D 表示周几(1-7,分别表示周一,周二,周三...周日)

例如:

2018 02 3 1

表示为 2018 年 2 月第三周的周一

2018 02 8 1 表示 2018 年 2 月第八周的周一

输出描述:

输出为正常日期格式为 YYYY-mm-DD

YYYY 表示年, mm 表示月, DD 表示日

```

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

//基姆拉尔森计算公式计算某年某月某日对应的星期
int CaculateWeekDay(int yy, int mm, int d)
{
    if (mm == 1 || mm == 2) {
        mm += 12;
    }
}

```



```

        yy--;
    }
    int iWeek = (d + 2 * mm + 3 * (mm + 1) / 5 + yy + yy / 4 - yy / 100 + yy / 400) % 7;
    return iWeek + 1;
}

//注意这里的星期: 周日-周六 (一周的开始是周日)

int a[][12] = { { 31,28,31,30,31,30,31,31,30,31,30,31 }, { 31,29,31,30,31,30,31,31,30,31,30,31 } };
int b[8] = { 0,1,2,3,4,5,6,0 };
//判断是否是闰年
int IsLeapYear(int y)
{
    if ((y % 400 == 0) || ((y % 4 == 0) && (y % 100 != 0)))
        return 1;
    else return 0;
}

int CaculateDay(int yy, int mm, int w, int n)
{
    //输入判断是否符合条件
    if (mm < 1 || mm > 12)return 0;
    if (w < 1 || w > 6)return 0;
    if (n < 1 || n > 7)return 0;
    //计算每月1号星期几-----》为下面判断周几是否有:
    //如2019年7月1号是星期一, 那么按照周日为一周的开始, 则n=7,return 0: CaculateDay函数 按照此逻辑
    //如果周一为一周开始, 参考CaculateDay2函数

    int days;
    int week_day = CaculateWeekDay(yy, mm, 1); //计算每月1号星期
    if (w == 1)
    {
        //第一周需要检查n是否超过
        if (b[n] < b[week_day])
            return 0;
        else
            days = b[n] - b[week_day] + 1;
    }
    else
    {
        days = b[6] - b[week_day] + 1;
        days += (w - 2) * 7;
        days += (b[n] + 1);
        if (days <= 0 || days > a[IsLeapYear(yy)][mm - 1])
        {
            return 0;
        }
    }
    return days;
}

int CaculateDay2(int yy, int mm, int w, int n)
{
    //输入判断是否符合条件
    if (mm < 1 || mm > 12)return 0;
    if (w < 1 || w > 6)return 0;
    if (n < 1 || n > 7)return 0;

    int days;
    int week_day = CaculateWeekDay(yy, mm, 1); //计算每月1号星期
    if (w == 1) {
        //第一周需要检查n是否超过
        if (n < week_day)
            return 0;
        else
            days = n - week_day + 1;
    }
    else {
        days = 7 - week_day + 1;
        days += (w - 2) * 7;
        days += n;
        if (days <= 0 || days > a[IsLeapYear(yy)][mm - 1])
            return 0;
    }
    return days;
}

int main(int argc, char** argv)
{
    int yy, mm, w, n;
    while (cin >> yy >> mm >> w >> n)
    {
        //int day = CaculateDay(yy, mm, w, n);
        int day = CaculateDay2(yy, mm, w, n);
        if (day == 0)
            cout << day << endl;
        else

```

```

        {
            cout << yy << "-" << setw(2) << setfill('0') << mm << "-" << setw(2) << setfill('0') << day <<
endl;
        }
    }
    return 0;
}

```

91、【带空白字符的字符串匹配】

给定两行空的字符串，第一行的字符串包含了部分空白（空格、tab），第二行字符串不包含任何空白，请从第一行字符串中匹配第二行字符串，匹配时忽略空白及 tab，输出第一行字符串中第二行字符串出现的次数；

输入描述：

第一行输入小于 1K 的字符串，包含了部分空白（空格、tab）

第二行输入小于 1K 的字符串，不包含任何空白

输出描述：

输出匹配到的次数（如果遇到连续字符如 bbb,查找 bb,记为匹配两次）

示例 1:

输入

Abb ba

bb

输出

2

```

#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <algorithm>
using namespace std;

string& Trim(string& str, string::size_type pos = 0)
{
    static const string delim = " \t"; //删除空格或者tab字符
    pos = str.find_first_of(delim, pos);
    if (pos == string::npos)
    {
        return str;
    }
    return Trim(str.erase(pos, 1));
}

void main(int argc, char** argv)
{
    while (1)
    {
        string str1 ;
        string str2 ;
        getline(cin, str1);
        getline(cin, str2);
        str1 = Trim(str1);
        str2 = Trim(str2);

        int count = 0;
        for (int i = 0; i < str1.length(); i++)
        {
            string temp = str1.substr(i, i + str2.length());
            string::size_type pos = temp.find(str2);
            if (pos != string::npos)
            {
                count++;
            }
        }
        cout << count << endl;
    }
}

```

92、【ABR 车路协同场景】

数轴×有两个点的序列 A={A1, A2,...Am}和 B={B1,B2,Bn}, A 和 Bj 均为正整数, A、B 已经从小到大排好序, A、B 均肯定不为空, 给定一个距离 R（正整数），列出同时满足如下条件的所有（Ai, Bj）数对：

条件

1) Ai <= Bj

2) A_i, B_j 距离小于等于 R ，但如果 A_i 找不到 R 范围内的 B_j ，则列出距它最近的 1 个 B_j ，当然此种情况仍然要满足 1)，但如果仍然找不到，就丢弃 A_i

原型

车路协同场景，一条路上发生了有很多事件（A），要通过很多路测设备（B）广播给路上的车，需要给每个事件找到一个合适的路测设备去发送广播消息。

输入描述

按照人易读的格式输入一行数据，参见输入样例，其中“ $ABR=[,]$ ”中的每个字符都是关键分割符，输入中无空格，其他均为任意正整数，输入 A 和 B 已经排好序，A 和 B 的大小不超过 50，正整数范围不会超过 65535。

输出描述：

(A_i, B_j) 数对序列，排列顺序满足序列中前面的 $A_x \leq$ 后面的 A_y ，前面的 $B_x \leq$ 后面的 B_y ，因为输入 A 和 B 已经排好序，所以实际上输出结果不用特意排序，排序不是考察点。

示例 1

输入

$A=[1,3,5], B=[2,4,6], R=1$

输出

(1,2)(3,4)(5,6)

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

void printArray(int* arr, int len)
{
    for (int j = 0; j < len; j++)
    {
        if (arr[j] == 0)
        {
            break;
        }
        printf("%d ", arr[j]);
    }
    printf("\n");
    return;
}

int getData(char* p, int* arr)
{
    printf("%s\n", p);
    if (strchr(p, '{') == NULL)
    {
        // C 库函数 char *strrchr(const char *str, int c)
        // 在参数 str 所指向的字符串中搜索最后一次出现字符 c（一个无符号字符）的位置。
        p = strchr(p, '=') + 1;
        sscanf(p, "%d", arr);
        // C 库函数 int sscanf(const char *str, const char *format, ...)
        // 从字符串读取格式化输入。
        printArray(arr, 1);
        return 1;
    }
    p = strchr(p, '{') + 1;
    printf("%s\n", p);
    int i = 0;
    while (*p != '\0')
    {
        char tmp[] = "";
        char* p1 = p;
        int n = 0;
        while (*p != ',' && *p != '\0')
        {
            n++;
            p++;
        }
        strncat(tmp, p1, n);
        // C 库函数 char *strncat(char *dest, const char *src, size_t n)
        // 把 src 所指向的字符串追加到 dest 所指向的字符串的结尾，直到 n 字符长度为止。
        sscanf(tmp, "%d", (arr + i));
        i++;
        if (*p == '\0')
        {
            break;
        }
        p++;
    }
    int arr_len = i;
    printArray(arr, arr_len);
    return arr_len;
}

int main(int argc, char** argv)
{
    int A[50] = { 0 };
    int B[50] = { 0 };
    int R = 0;
```

<pre> char str[210]; char* p = str; scanf("%s", str); printf("%s\n", str); const char s[2] = "}"; p = strtok(str, s); // C 库函数 char *strtok(char *str, const char *delim) // 分解字符串 str 为一组字符串, delim 为分隔符。 int len_A = getData(p, A); p = strtok(NULL, s); int len_B = getData(p, B); p = strtok(NULL, s); getData(p, &R); // A[i]<=B[j] // distance(A[i],B[j])<=R // print all (A[i],B[j]) size_t noFind = 1; for (int i = 0; i < len_A; i++) { for (int j = 0; j < len_B; j++) { if (A[i] - B[j] > 0) { continue; } if (B[j] - A[i] <= R) { printf("(%d,%d)", A[i], B[j]); noFind = 0; continue; } if (noFind) { printf("(%d,%d)", A[i], B[j]); } break; } } system("pause"); return 0; } </pre>	<pre> #01 原数据 line = "A={1,3,5},B={2,4,7,8},R=1" # 最终输出: (1,2)(3,4)(5,6) #02 划分数据: 在时间复杂度和空间复杂度的处理上, 使用列表生成式的好处 line = Line.split('=') # ['A', '{1,3,5},B', '{2,4,6},R', '1'] A = [int(i) for i in list(line[1]) if i != '{' and i != '}' and i != ',' and i != 'B'] B = [int(i) for i in list(line[2]) if i != '{' and i != '}' and i != ',' and i != 'R'] R = int(line[-1]) #03 满足条件: L_list = [] for a in A: for b in B: # 第一个条件: a <= b if a<=b: dis_ab = abs(a-b) # 在R范围内 if dis_ab<=R: L_list.append((a,b)) break # 找到即结束当前循环 # 不在R范围内 else: if dis_ab > R: # 由于A、B序列为排好序的序列, >R的第一个数即为最近的, 找到该数后, break循环 L_list.append((a,b)) break # 找到即结束当前循环 #05 格式化输出 L_str = [str(i) for i in L_list] s = "".join(L_str) s = s.replace(" ", "") # 去除空格 print(s) </pre>
--	---

93、【勾股数元祖】

题目描述:
 如果 3 个正整数(a,b,c)满足 $a^2+b^2=c^2$ 的关系,则称(a,b, c)为勾股数(著名的勾三股四弦五),为了探索勾股数的规律,我们定义如果勾股数(a,b,c)之间两两互质(即 a 与 b, a 与 c, b 与 c 之间均互质),则其为勾股数元祖(例(3,4,5)是勾股数元祖, (6,8,10)则不是勾股数元祖).请求出始范围[N,M]内,所有的勾股数元祖。
 输入描述

起始范围 N, $1 \leq N \leq 200$

结束范围 M, $N \leq M \leq 200$

输出描述:

- 1.a,b, c 请保证 $a < b < c$,输出格式:a b c;
- 2.多组勾股数元祖请按照 a 升序,b 升序,最后 c 升序的方式排序输出;
- 3.给定范围中如果找不到勾股数元祖时,输出"NA".

备注:

1.[N,M]范围表示 a,b,c 三个数均在[N,M]范围。

示例 1

输入

1

20

输出

3 4 5

5 12 13

8 15 17

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// #include <unistd.h>
#include <iostream>
using namespace std;
#include <math.h>

/* 判断两个数字是否互质, 返回值是1时表明互质, 其它值则不互质 */
int is_coprime(int src1, int src2)
{
    if (0 == src2)
    {
        return src1;
    }
    else
    {
        return is_coprime(src2, src1 % src2);
    }
}

void printResult(int a, int b, int c)
{
    // print the three number orderedly
    if (a < b && b < c)
    {
        printf("%d %d %d\n", a, b, c);
    }
    else if (a > b && b > c)
    {
        printf("%d %d %d\n", c, b, a);
    }
    else if (b < c && a > c)
    {
        printf("%d %d %d\n", b, c, a);
    }
    else
    {
        printf("%d %d %d\n", b, a, c);
    }
}

int GogutanPrimitive(int N, int M)
{
    int countor = 0;
    int m = 0;

    if (0 < N && N < M)
    {
        m = sqrt(M);
    }
    else
    {
        return -1;
    }

    for (int i = N; i <= M; i++)
    {
        for (int j = i + 1; j <= M; j++)
        {
            for (int k = j + 1; k <= M; k++)
            {
                if (i * i + j * j == k * k)
                {
                    printResult(i, j, k);
                    countor++;
                }
            }
        }
    }

    if (countor == 0)
    {
        printf("NA\n");
    }
}
```

```

        {
            if (1 == is_coprime(i, j) && 1 == is_coprime(i, k) && 1 == is_coprime(j, k))
            {
                countor++;
                printResult(i, j, k);
                //printf("%d %d %d\n", a, b, c);
            }
        }
    }
}

if (countor == 0)
{
    printf("NA");
}
return 0;
}

int main(int argc, char** argv)
{
    int N = 0, M = 0;
    scanf_s("%d %d", &N, &M);

    GogutanPrimitive(N, M);

    return 0;
}

```

94、【素勾股数】

题目描述：勾股数，是由三个正整数组成的数组：能符合勾股定理 $a^2 + b^2 = c^2$ ，(a,b,c) 的正整数解。

如果 (a,b,c) 是勾股数，它们的正整数倍数，也是勾股数。

如果 (a,b,c) 互质，它们就称为素勾股数。

给定正整数 N，计算出小于或等于 N 的素勾股数个数。

输入描述：输一个正整数

输出描述：素勾股数

示例 1:

输入

10

输出

```

1
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

/* 判断两个数字是否互质，返回值是1时表明互质，其它值则不互质 */
int is_coprime(int src1, int src2)
{
    if (0 == src2)
        return src1;
    else
        return is_coprime(src2, src1 % src2);
}

int main(int argc, char** argv)
{
    int input = 0;
    int i, j;
    int a, b, c;
    int countor = 0;
    int m;
    scanf_s("%d", &input);

    if (0 < input)
    {
        m = sqrt(input);
    }
    else
    {
        return -1;
    }

    for (i = 1; i <= input; i++)
    {
        for (j = i + 1; j <= input; j++)
        {

```

```

        a = j * j - i * i;
        b = 2 * i * j;
        c = i * i + j * j;
        if (c <= input)
        {
            if (1 == is_coprime(a, b) && 1 == is_coprime(b, c) && 1 == is_coprime(a, c))
            {
                countor++;
                printf("a=%d,b=%d,c=%d\n", a, b, c);
            }
        }
    }
}

printf("%d\n", countor);
return 0;
}

```

95、【等式变换】

输入一个正整数 x ，在下面的等式左边的数字之间添加+号或者-号，使得等式成立。

1 2 3 4 5 6 7 8 9 = x

比如：

12-34+5-67+89 = 5

1+23+4-5+6-7-8-9 = 5

请编写程序，统计满足输入整数的所有整数个数。

输入描述：

正整数，等式右边的数字

输出描述： 使该等式成立的个数

示例 1

输入：5

输出：21

```

#include<iostream>
#include<cmath>
using namespace std;
int dp(int before, int des, int res, int ex)
{
    if (before == 0)
    {
        if (des == res)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
    else
    {
        return dp(before - 1, before, res + des, 1) + dp(before - 1, before, res - des, 1) + dp(before - 1, before * pow(10, ex) + des, res, ex + 1);
    }
}

int main(int argc, char** argv)
{
    int n; cin >> n;
    cout << dp(8, 9, n, 1);
    return 0;
}

```

96、【平安果】

题目描述：

给定一个 M 行 N 列的矩阵 ($M \times N$ 个格子)，每个格子中放着一定数量的平安果。

你从左上角的格子开始，只能向下或者向右走，目的地是右下角的格子。

每走过一个格子，就把格子上的平安果都收集起来。求你最多能收集到多少平安果。

注意：当经过一个格子时，需要一次性把格子里的平安果都拿走。

限制条件: $1 \leq N, M \leq 50$; 每个格子里的平安果数量是 0 到 1000 (包含 0 和 1000)。

输入描述:

输入包含两部分:

第一行 M, N

接下来 M 行, 包含 N 个平安果数量

输出描述:

一个整数, 最多拿走的平安果的数量

示例 1:

输入

2 4

1 2 3 40

6 7 8 90

输出

136

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;
int dp(int m, int n, int apple[][50])
{
    if (m == 0 && n == 0)
    {
        return apple[0][0];
    }
    else if (m == 0)
    {
        return apple[m][n] + dp(m, n - 1, apple);
    }
    else if (n == 0)
    {
        return apple[m][n] + dp(m - 1, n, apple);
    }
    else
    {
        return max(apple[m][n] + dp(m, n - 1, apple), apple[m][n] + dp(m - 1, n, apple));
    }
}

int main(int argc, char** argv)
{
    int m, n; cin >> m >> n;
    int apple[50][50];

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            int tmp; cin >> tmp;
            apple[i][j] = tmp;
        }
    }

    cout << dp(1, 3, apple);
    return 0;
}
```

97、√靠谱的数学

数字从 1 开始, 遇到数字 7 就会跳过, 比如 6 后边直接是 8, 69 后边直接是 80, 现在给你一个数字, 问是第几位?

比如输入 8, 输出 7, 就是第 7 个数。那 89 那? 请你编程输出。

```
#include <iostream>
#include <string>
#include <stack>
#include <unordered_map>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    int number = 0;
    vector<int> vec;
    cin >> number;
```



```

    for (int i = 1; i < 2 * number; i++)
    {
        string temp = to_string(i);
        string::size_type pos = temp.find("7");
        if (pos == string::npos)
        {
            vec.push_back(i);
        }
    }

    for (int i = 0; i < vec.size(); i++)
    {
        if (vec[i] == number)
        {
            cout << i + 1 << endl;
        }
    }

    return 0;
}

```

98、✓ 输出匹配字符串的开始下标

给出母串和子串，输出子串能够在母串完全匹配的开始位置。

比如 asdfasdfa, fas 输出 3，就是最小下标。

```

#include <iostream>
#include <string>
#include <stack>
#include <unordered_map>
#include <algorithm>
using namespace std;

void Split(string input, vector<string>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(one_word);
            one_word = "";
            j = i + 1;
        }
    }
}

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        vector<string> vec;
        Split(input_string, vec, ',');
        int pos = vec[0].find(vec[1]);
        cout << pos << endl;
    }

    return 0;
}

```

99、金字塔

微商模式比较典型，下级每赚 100 元就要上交 15 元，给出每个级别的收入，求出金字塔尖上的人收入。

比如：

（代理商代号） （上级代理商代号） （代理商转的钱）

1 0 223

2 0 323

3 2 1203

输出是 0（金字塔顶代理商）105（最终的钱数）

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
using namespace std;

double GetSumPerLayer(vector<int>& array_Renamed, int layer)
{
    double sum = 0;
    for (int i = 0; i < array_Renamed.size(); i++)
    {
        sum += array_Renamed[i];
    }
    sum = sum * pow(0.15, layer);
    return sum;
}

void PyramidTopMoney(vector <vector<int>> martix)
{
    vector<int> money(10);
    for (int i = 0; i < martix.size(); i++)
    {
        money[i] = martix[i][2];
    }

    int layer = 0;
    int cur_count = 0;
    for (; cur_count <= martix.size(); )
    {
        cur_count += pow(2, layer);
        layer++;
    }

    vector<int> array1(2);
    vector<int> array2(4);
    vector<int> array3(8);

    for (int i = 0; i < 2; i++)
    {
        array1[i] = money[i + 1];
    }
}
```

```

    for (int i = 0; i < 4; i++)
    {
        array2[i] = money[i + 3];
    }
    for (int i = 0; i < 3; i++)
    {
        array3[i] = money[i + 7];
    }

    double result = GetSumPerLayer(array1, 1) + GetSumPerLayer(array2, 2) +
    GetSumPerLayer(array3, 3);

    cout << static_cast<int>(result) << endl;
}

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}

int main(int argc, char** argv)
{
    vector <vector<int>> martix;
    string line;
    while (getline(cin, line))
    {

```

```

        if (!line.empty())
        {
            vector<int> vec;
            Split(line, vec, ' ');
            martix.push_back(vec);
        }
        else
        {
            break;
        }
    }
    PyramidTopMoney(martix);
    return 0;
}

```

100、✓组最大数字

给出几组字符串的数字，需要获得组成的最大数字。

比如输入 123, 546, 8, 32, 输出 854632123

```

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <string>
#include <vector>
using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i+1;
        }
    }
}

void ComposeBiggest(vector<int>& nums_vec)
{
    int len = nums_vec.size();
    int tempNum;
    string temp1, temp2;
    for (int i = 0; i < len; i++)
    {
        for (int j = i + 1; j < len; j++)
        {
            temp1 = to_string(nums_vec[i]) + to_string(nums_vec[j]);
            temp2 = to_string(nums_vec[j]) + to_string(nums_vec[i]);
            if (stoi(temp1) - stoi(temp2) < 0)
            {
                tempNum = nums_vec[i];
                nums_vec[i] = nums_vec[j];
                nums_vec[j] = tempNum;
            }
        }
    }
}

int main(int argc, const char* argv[])
{
    string input;
    getline(cin, input);
}

```

```

vector<int> nums_vec;
Split(input, nums_vec, ',');
ComposeBiggest(nums_vec);
for (int i = 0; i < nums_vec.size(); i++)
{
    cout << nums_vec[i];
}
return 0;
}

```

101、√特殊计算

特殊符号代替普通的计算方式比如 $x\#y = 2 * x + y$, $x\$y = x + 3y$, #优先级高于\$。
 比如输入 5#2\$6 输出结果就是 30, 因为先算 $5\#2 = 12$, 再算 $12\$6 = 30$

```

#include <iostream>
#include <cmath>
#include <stack>
#include <string>
#include <vector>
using namespace std;

// #
int XHY(int x, int y)
{
    return 2 * x + y;
}

// $
int XRY(int x, int y)
{
    return x + 3*y;
}

int main(int argc, char** argv)
{
    string input_string;
    cin >> input_string;

    stack<int> stk;
    string number;

    int operand1 = -1;
    int operand2 = -1;
    for (int i = 0; i < input_string.size(); i++)
    {
        if (input_string[i] >= '0' && input_string[i] <= '9')
        {
            number.push_back(input_string[i]);
        }
        else
        {
            if (!number.empty())
            {
                stk.push(stoi(number));
                number.erase(number.begin(), number.end());
            }

            if (operand1 >= 0)
            {
                operand2 = stk.top();
                stk.pop();
                stk.push(XRY(operand1, operand2));
                operand1 = -1;
            }

            if (input_string[i] == '$')
            {
                if (operand1 < 0)
                {
                    operand1 = stk.top();
                    stk.pop();
                }
            }
        }
    }

    if (!number.empty())
    {
        stk.push(stoi(number));
        if (operand1 >= 0)
        {

```

```

        operand2 = stk.top();
        stk.pop();
        stk.push(XRY(operand1, operand2));
        operand1 = -1;
    }
}

stack<int> dst;
while (!stk.empty())
{
    dst.push(stk.top());
    stk.pop();
}

int out_value = -1;
while (!dst.empty())
{
    if (out_value < 0)
    {
        out_value = dst.top();
    }
    else
    {
        out_value = XHY(out_value, dst.top());
    }
    dst.pop();
}

cout << out_value;

return 0;
}

```

102、✓ 数组元素划分

给出一个数组,按照这种规则划分,规则是在同一类里的都能被这个类里的最小的数字整除。比如[2,3,4], 输出是 2 个类划分, 即[2,4]和[3]

```

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i + 1;
        }
    }
}

void Classify(vector<int> nums_vec, vector<string>& output)
{
    while (!nums_vec.empty())
    {
        int temp = nums_vec[0];
        string temp_string = "[";
        for (int i = 0; i < nums_vec.size(); i++)
        {
            if (nums_vec[i] % temp == 0)
            {
                temp_string += to_string(nums_vec[i]);
                if (i < nums_vec.size() - 1)
                {
                    temp_string += ",";
                }
                nums_vec.erase(nums_vec.begin() + i);
            }
        }
        if (temp_string[temp_string.size() - 1] == ',')
    }
}

```

```

        {
            temp_string.erase(temp_string.begin() + (temp_string.size() - 1));
        }
        temp_string += "]";
        output.push_back(temp_string);
    }
}

int main(int argc, char** argv)
{
    string input;
    getline(cin, input);
    vector<int> nums_vec;
    string array_string = input.substr(1, input.size() - 2);
    Split(array_string, nums_vec, ',');
    sort(nums_vec.begin(), nums_vec.end());

    vector<string> outputs;
    Classify(nums_vec, outputs);
    cout << outputs.size() << endl;
    for (int i = 0; i < outputs.size(); i++)
    {
        cout << outputs[i] << endl;
    }
    return 0;
}

```

103、√ 到达终点

给定一个数组，从第一个开始，正好走到数组最后，所使用的最少步骤数。要求：第一步从第一元素开始，第一步小于 $\text{len}/2$ （ len 为数组的长度）。从第二步开始，只能以所在成员的数字走相应的步数，不能多也不能少，如果目标不可达返回-1，输出最少的步骤数，不能往回走。

输入 7 5 9 4 2 6 8 3 5 4 3 9 输出 2

输入 1 2 3 7 1 5 9 3 2 1 输出-1

```

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

int main(int argc, char** argv)
{
    int input_array[100] = { 0 };
    int num = 0;
    int max = 0;
    while (scanf("%d", &input_array[num]) != EOF)
    {
        if (max < input_array[num])
        {
            max = input_array[num];
        }
        num += 1;
    }

    num = num;
    int length = (num + 1) / 2;
    int time = 0;
    int i = 0;
    int flag = 0;
    int max_step = 0;
    int max_index = 0;

    while (num >= 0)
    {
        max_step = 0;
        max_index = 0;
        for (i = 1; i <= max + 1 && num - i >= 0; i++)
        {
            if (input_array[num - i] == i - 1)
            {
                max_index = i;
                max_step = input_array[num - i];
            }
        }
        if (i == 1)
        {
            break;
        }
        time++; //往前走
        num = num - max_index;
    }
}

```

```

        if ((input_array[num] == 0 || max_index == 0) && num >= input_array[0])
        {
            flag = 1;
            break;
        }
        else if (max_index == 0)
        {
            break;
        }
    }
    if ((num == 0 || input_array[0] >= num) && flag != 1)
    {
        printf("%d\n", time);
    }
    else
    {
        printf("-1\n");
    }
    return 0;
}

```

104、能量消耗

给出一批用户，每个用户有 3 种选择 A\B\C，但是价值不同，相临的用户不能选同一个，求出所有用户选择后总价值最大。

3 个用户

30 8 4

50 20 9

11 7 6

输出 65，因为选 8 50 7

```

#include <iostream>
#include <unordered_map>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void SplitInt(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}

int MaxEnergy(vector<vector<int>> martix)
{
    int sum = 0;
    for (int i = 0; i < martix.size(); i++)
    {
        for (int j = 0; j < martix[i].size(); j++)
        {
            if (j == i)
            {
                continue;
            }
            for (int k = 0; k < martix[i].size(); k++)
            {
                if (k == j)
                {
                    continue;
                }
                int temp = martix[0][i] + martix[1][j] + martix[2][k];
            }
        }
    }
}

```



```

        if (temp > sum)
        {
            sum = temp;
        }
    }
}
return sum;
}

int main(int argc, char** argv)
{
    while (1)
    {
        string input;
        vector<vector<int>> martix;
        while (getline(cin, input))
        {
            if (!input.empty())
            {
                vector<int> vec;
                SplitInt(input, vec, ' ');
                martix.push_back(vec);
            }
            else
            {
                break;
            }
        }
        cout << MaxEnergy(martix) << endl;
    }

    return 0;
}

```

```

#include <iostream>
#include <unordered_map>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void SplitInt(string input, vector<int>&output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}

int MaxEnergy(vector<vector<int>> martix)
{
    int line = 0;
    int sum = 0;
    int flag = 0;
    for (; line < martix.size(); )
    {
        int sum_temp = 0;
        for (int i = 0; i < martix[line].size(); i++)
        {
            if (line >= 3)
            {
                if (i == flag)
                {
                    continue;
                }
            }
            for (int j = 0; j < martix[line].size(); j++)
            {
                for (int k = 0; k < martix[line].size(); k++)
                {
                    if ((j == i) || (k == j))
                    {
                        continue;
                    }
                }
            }
        }
    }
}

```

```

        int first_line = line;
        int second_line = line + 1;
        int third_line = line + 2;
        int temp = martix[first_line][i] + martix[second_line][j] + martix[third_line][k];

        if (temp > sum_temp)
        {
            sum_temp = temp;
        }
        flag = k;
    }

    }
    line = line + 3;
    sum += sum_temp;
}
return sum;
}

int main(int argc, char** argv)
{
    while (1)
    {
        string input;
        vector<vector<int>> martix;
        while (getline(cin, input))
        {
            if (!input.empty())
            {
                vector<int> vec;
                SplitInt(input, vec, ' ');
                martix.push_back(vec);
            }
            else
            {
                break;
            }
        }
        cout << MaxEnergy(martix) << endl;
    }

    return 0;
}

```

105、√数字分解

给你一个数可以分解成几组的连续的自然数之和？

比如 8 可以分解成 8，即 1 组，比如 6 可以分解成 6,123，即 2 组。

```

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

int Add(int m, int n)
{
    int sum = 0;
    for (int i = m; i <= n; i++)
    {
        sum += i;
    }
    return sum;
}

void Divide(int num, vector<int>& result)
{
    int i = 1, j = 2;
    int sum = 0;
    while (i <= num / 2)
    {
        sum = Add(i, j);
        while (sum != num) //一定存在解，所以不会死循环
        {
            if (sum > num)
            {
                i++;
            }
            else
            {
                j++;
            }
            sum = Add(i, j);
        }
    }
}

```

```

    }
    for (int k = i; k <= j; k++)
    {
        result.push_back(k);
    }
    ++i;
}
}

int main(int argc, char** argv)
{
    int num;
    vector<int> result;
    cin >> num;
    Divide(num, result);
    cout << result[result.size() - 1];
    if (result.size() > 1)
    {
        cout << ",";
        for (int i = 0; i < result.size()-1;i++)
        {
            cout << result[i];
        }
        cout << endl;
    }

    return 0;
}

```

106、√喊 7 的次数重排

喊 7 是一个传统的聚会游戏，N 个人围成一圈，按顺时针从 1 到 N 编号。编号为 1 的人从 1 开始喊数，下一个人喊的数字为上一个人的数字加 1，但是当数字是 7 的倍数或者数字本身含有 7 的话，要喊"过"。现给定一个长度为 N 的数组，存储了打乱顺序的每个人喊"过"的次数，请把它还原成正确的顺序，即数组的第 i 个元素存储编号 i 的人喊"过"的次数。

输入为一行，为空格分隔的喊"过"的次数，

示例 1

输入

0 1 0

输出

1 0 0

说明

一共只有一次喊"过"，那只会发生在需要喊 7 时，按顺序，编号为 1 的人会遇到 7，故输出 1 0 0。注意，结束时的 K 不一定是 7，也可以是 8、9 等，喊过的次数都是 1 0 0。

示例 2

输入

0 0 0 2 1

输出

0 2 0 1 0

说明

一共有三次喊"过"，发生在 7 14 17，按顺序，编号为 2 的人会遇到 7 17，编号为 4 的人会遇到 14，故输出 0 2 0 1 0。

```

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

bool IsSeven(int num)
{
    if (0 == num % 7)
    {
        return true;
    }
    while (0 != num)
    {

```

```

        if (7 == num % 10)
        {
            return true;
        }
        num /= 10;
    }
    return false;
}

int main(int argc, char** argv)
{
    string input_string;
    while (1)
    {
        getline(cin, input_string);
        if (input_string.empty())
        {
            break;
        }
        int n = 0; //读取单个数值
        int counts = 0; //喊7次数
        int people_count[10] = { 0 }; //存值数组, 用于存放每个人喊7的次数。
        int peoples = 0; //一共几个人
        stringstream sstrcin(input_string);
        while (sstrcin >> n)
        {
            counts += n;
            peoples++;
        }

        if (0 == counts)
        {
            cout << input_string << endl;
            continue;
        }
        int num = 0;
        int i = 1;
        while (num < counts)
        {
            if (true == IsSeven(i))
            {
                people_count[i % peoples - 1]++;
                num++;
            }
            i++;
        }
        for (int i = 0; i < peoples; i++)
        {
            cout << people_count[i] << " ";
        }
        cout << endl;
        input_string.clear();
    }
}

```

107、✓找出出错的数字方阵求和 不一致问题

数字方阵就是 $N \times N$ 数组成的, 其中有一个特殊的方阵, 每行、每列、对角线的和都是一样, 现在其中一个数字错误, 导致的不在满足和相等, 现在要找出这个位置。

8 1 6

3 5 7

4 0 2

可以看到出错的位置在<3,2>, 并且数字应该是 9。

```

#include <iostream>
#include <string>
#include <vector>
using namespace std;

typedef struct PointPos
{
    int row;
    int cow;
}PointPos;

bool FindErrorPos(vector<int> vec_row, vector<int> vec_cow, PointPos& point_pos)
{
    int row_temp = vec_row[0];

```

```

bool flag = true;
for (int i = 0; i < vec_row.size(); i++)
{
    if (vec_row[i] != row_temp)
    {
        flag = false;
        if (i < vec_row.size() - 1)
        {
            if (vec_row[i] == vec_row[i + 1])
            {
                point_pos.row = 1;
                break;
            }
            else
            {
                point_pos.row = i + 1;
                break;
            }
        }
        if (i == vec_row.size() - 1)
        {
            point_pos.row = i + 1;
            break;
        }
    }
}
int cow_temp = vec_cow[0];

for (int i = 0; i < vec_cow.size(); i++)
{
    if (vec_cow[i] != cow_temp)
    {
        flag = false;
        if (i < vec_cow.size() - 1)
        {
            if (vec_cow[i] == vec_cow[i + 1])
            {
                point_pos.cow = 1;
                break;
            }
            else
            {
                point_pos.cow = i + 1;
                break;
            }
        }
        if (i == vec_cow.size() - 1)
        {
            point_pos.cow = i + 1;
            break;
        }
    }
}
return flag;
}

void MagicMatrix(vector<vector<int>> input, vector<int>& rows_sum, vector<int>& cows_sum)
{
    for (int i = 0; i < input[0].size(); i++)
    {
        int row_sum = 0;
        int cow_sum = 0;
        for (int j = 0; j < input[0].size(); j++)
        {
            row_sum += input[i][j];
            cow_sum += input[j][i];
        }
        rows_sum.push_back(row_sum);
        cows_sum.push_back(cow_sum);
    }
}

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
        }
    }
}

```

```

        one_word = "";
        j = i;
    }
}

int CorrectValue(vector<vector<int>> matrix, vector<int> rows_sum, PointPos pos)
{
    int correctValue;
    if (pos.row - 1 < rows_sum.size() - 1)
    {
        if (rows_sum[pos.row - 1] < rows_sum[pos.row])
        {
            correctValue = matrix[pos.row - 1][pos.cow - 1] + (rows_sum[pos.row] - rows_sum[pos.row - 1]);
        }
        else
        {
            correctValue = matrix[pos.row - 1][pos.cow - 1] - (rows_sum[pos.row - 1] - rows_sum[pos.row]);
        }
    }
    else
    {
        if (rows_sum[pos.row - 1] < rows_sum[pos.row - 2])
        {
            correctValue = matrix[pos.row - 1][pos.cow - 1] + (rows_sum[pos.row - 2] - rows_sum[pos.row -
1]);
        }
        else
        {
            correctValue = matrix[pos.row - 1][pos.cow - 1] - (rows_sum[pos.row - 1] - rows_sum[pos.row -
2]);
        }
    }

    return correctValue;
}

int main(int argc, char** argv)
{
    while (1)
    {
        string line;

        vector<string> lines;
        while (getline(cin, line))
        {
            if (!line.empty())
            {
                lines.push_back(line);
            }
            else
            {
                break;
            }
        }
        int line_count = lines.size();
        vector<vector<int>> matrix(line_count);

        for (int i = 0; i < line_count; i++)
        {
            Split(lines[i], matrix[i], ' ');
        }

        PointPos pos = { -1, -1 };
        vector<int> rows_sum;
        vector<int> cows_sum;
        MagicMatrix(matrix, rows_sum, cows_sum);
        bool flag = FindErrorPos(rows_sum, cows_sum, pos);
        if (!flag)
        {
            cout << "<" << pos.row << ", " << pos.cow << ">" << endl;

            cout << "value = " << CorrectValue(matrix, rows_sum, pos) << endl;
        }
    }
    return 0;
}

```

108、✓贪吃蛇

现在有一个 $N \times M$ ($N, M=100$) 的方形矩形，在这个矩形的每一个方格上都放有一个随机值，

一条可爱的小蛇从矩形的 左上角开始出发，每次移动都只能移动一格，向右或向下，而每到达一格贪吃的小蛇都会把该位置上的值吃个一干二净，直到到达右下角时停止。而贪吃的小蛇不怕撑死，它只想吃到最多，并输出路径。

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <stack>
#include <ctime>
using namespace std;

stack<int> istack;
int value[100][100], best[100][100], n, m;
//value里面存储每个格子的值
//best 存储走到当前格子的最大值
//m,n为格子大小100*100

void InitMatrixValue()
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 1; j < n; j++)
        {
            srand(unsigned(time(NULL)));
            value[i][j] = (rand() % (1000 - 1 + 1)) + 1; //[a,b]
        }
        value[0][0] = 0;
    }
}

int SearchMaxValue()
{
    best[0][0] = 0;
    for (int i = 1; i < m; i++)
    {
        best[i][0] = value[i][0] + best[i - 1][0];
    }
    for (int j = 1; j < n; j++)
    {
        best[0][j] += value[0][j] + best[0][j - 1];
    }
    for (int i = 1; i < m; i++)
    {
        for (int j = 1; j < n; j++)
        {
            best[i][j] = value[i][j] + max(best[i - 1][j], best[i][j - 1]);
        }
    }
    return best[m - 1][n - 1];
}

void StackPush(int n, int m)
{
    istack.push(n);
    istack.push(m);
    if (n == 0 && m == 0)
    {
        return;
    }
    else if (n == 0 && m > 0)
    {
        StackPush(n, m - 1);
    }
    else if (n > 0 && m == 0)
    {
        StackPush(n - 1, m);
    }
    else
    {
        if (best[n - 1][m] > best[n][m - 1])
        {
            StackPush(n - 1, m);
        }
        else
        {
            StackPush(n, m - 1);
        }
    }
}

void PrintRoute()
{
    StackPush(99, 99);
    while (!istack.empty())
    {

```

```

        cout << istack.top() << "___";
        istack.pop();
        cout << istack.top();
        istack.pop();
        cout << endl;
    }
}

int main(int argc, char** argv)
{
    m = 100;
    n = 100;
    InitMatrixValue();
    int bestvalue = SearchMaxValue();

    //cout << bestvalue;
    PrintRoute();
    //先初始化价值矩阵之后，计算最大价值矩阵，然后输出路径
    return 0;
}

```

109、✓特定和的子序列

给出一个数组和一个固定 SUM，求出连续元素和为 sum。例如：[1,2,3,4,5,6,7,8] sum = 10
输出是 1 2 3 4；sum= 15 输出是 1 2 3 4 5； 4 5 6； 7 8。

```

#include <iostream>
#include <cstdlib>
#include <map>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i + 1;
        }
    }
}

void DivideArrayBySum(vector<int> vec_int, int sum, vector<string>& output)
{
    int flag = 0;

    while (flag < vec_int.size())
    {
        int i = 0;
        int temp_sum = 0;
        string arr_string = "[";
        for (i = flag; i < vec_int.size(); i++)
        {
            temp_sum += vec_int[i];
            arr_string += to_string(vec_int[i]);
            if (temp_sum < sum)
            {
                arr_string += " ";
            }
            else if (temp_sum == sum)
            {
                arr_string += "]";
                output.push_back(arr_string);
                break;
            }
            else
            {
                break;
            }
        }
        flag++;
    }
}

```



```

}

int main(int argc, char** argv)
{
    vector<int> vec_int = { 1,2,3,4,5,6,7,8 };
    int sum;
    while (cin >> sum)
    {
        vector<string> outpus;
        DivideArrayBySum(vec_int, sum, outpus);
        for (int i = 0; i < outpus.size(); i++)
        {
            cout << outpus[i] << endl;
        }
    }
    return 0;
}

```

110、√勾股定理

给定一个区间，找出所有的可以组成 $a^2 + b^2 = c^2$ 的组合，例如[2,10]，可以找出两组：3 4 5；6 8 10。

```

#include <iostream>
#include <string>
#include <stdio.h>
#include <queue>

using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i + 1;
        }
    }
}

int main(int argc, char** argv)
{
    string input = "";
    vector<int> temp_vec;
    cin >> input;
    Split(input, temp_vec, ',');

    int start = temp_vec[0];
    int end = temp_vec[1];

    for (int i = start; i <= end; i++)
    {
        for (int j = i+1; j <= end; j++)
        {
            for (int k = j+1; k <= end; k++)
            {
                if (i*i + j*j == k*k)
                {
                    cout << i << " " << j << " " << k << endl;
                }
            }
        }
    }
    return 0;
}

```

111、√找出可以围城正方形的边长

给一组数据，找出可以围城正方形的边长，例如[1,2,2,3,4,4]，可以组成边长为 4 的正方形。

```

#include <iostream>
#include <cstdlib>

```

```

#include <map>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i + 1;
        }
    }
}

void Square(vector<int> vec_int, int& length)
{
    int sum = 0;
    for (int i = 0; i < vec_int.size(); i++)
    {
        sum += vec_int[i];
    }
    length = sum / 4;
}

int main(int argc, char** argv)
{
    string input;
    while (getline(cin, input))
    {
        if (!input.empty())
        {
            input = input.substr(1, input.size() - 2);
            vector<int> vec_int;
            Split(input, vec_int, ',');
            int length;
            Square(vec_int, length);
            cout << length << endl;
        }
    }
    return 0;
}

```

112、✓数组拼接

给出多个一维数组，每次依次取固定的个数放到同一个数组中。这个主要考察动态的输入多个一维数组，和数组的长度

例如：如下，输出 1,2,3,1,2,3,1,2,3, 4,5,6, 4

3 (一次取的固定数目)

3 (有几个数组)

1,2,3,4,5,6

1,2,3

1,2,3,4

输出 1,2,3,1,2,3,1,2,3, 4,5,6, 4

```

#include <iostream>
#include <string>
#include <stdio.h>
#include <queue>

using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

```

```

for (int i = 0; i < input.size(); i++)
{
    if (input[i] == patten)
    {
        one_word = input.substr(j, i - j);
        output.push_back(stoi(one_word));
        one_word = "";
        j = i + 1;
    }
}

int main(int argc, char** argv)
{
    queue<int> que[10000];
    int res[100005];
    int num;

    int cnt = 0;
    string str;

    int k,n;
    cin >> k;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        string input = "";
        vector<int> temp_vec;
        cin>> input;
        Split(input, temp_vec, ',');
        for (int j = 0; j < temp_vec.size(); j++)
        {
            que[cnt].push(temp_vec[j]);
        }
        cnt++;
    }

    num = 0;
    while (true)
    {
        int flag = 0;
        for (int i = 0; i < cnt; i++)
        {
            int l = 0;
            while (!que[i].empty() && l < k)
            {
                flag = 1;
                int ans = que[i].front();
                que[i].pop();
                l++;
                res[num++] = ans;
            }
        }
        if (flag == 0)
            break;
    }

    for (int i = 0; i < num; i++) //以 1, 2, 3, 4 的格式输出合并后的新数组
    {
        cout << res[i];
        if (i == num - 1)
            printf("\n");
        else
            printf(",");
    }

    return 0;
}

```

113、动态规划固定的容积水

固定的容积，可以最多装多少？

例如：[5 3 4 6 1 2 9]，容积是 15，输出为：5，说明是 3 4 1 2

114、✓小兔子繁殖

小兔子每三个月可以长大，长大后每月生一个，到达 N 天时有几只兔子
例如 4 月，有 2 只，5 月有 3 只，。。。7 月有 6 只。

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

int RabbitBreed(int month)
{
    int start_cout = 1;
    if (month <= 3)
    {
        int rabbit_counts = start_cout;
        return rabbit_counts;
    }
    else if ((month > 3) && (month <= 4))
    {
        int rabbit_counts = month - 2;
        return rabbit_counts;
    }
    else
    {
        int temp = month - 4;
        int rabbit_counts = (temp*temp)/2 - temp/2 + 3;
        return rabbit_counts;
    }
}

int main(int argc, char** argv)
{
    int month;
    while (cin >> month)
    {
        cout << RabbitBreed(month) << endl;
    }
    return 0;
}
```

115、✓字符串匹配（正则）

给出一个字符串，给出一个正则白表达式子串，找出可以找到子串的第一个位置。
例如:字符串 asdfas，子串是 d[sf]，([]表示里边任意一个元素)，输出为 3，因为 ds 或 df 去匹配字符串，找到 df 输出位置为 3。

```
#include <iostream>
#include <regex>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

int RegexSubStringPos(vector<string> input)
{
    regex sub_string(input[1]);
    smatch m;
    int pos = -1;
    auto ret = regex_search(input[0], m, sub_string);
    if (ret)
    {
        pos = m.position();
    }
    return pos + 1;
}

void Split(string input, vector<string>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(one_word);
            }
            j = i + 1;
            one_word = "";
        }
    }
}
```

```

        output.push_back(one_word);
    }
    one_word = "";
    j = i + 1;
}
}

int main(int argc, char** argv)
{
    string input;
    while (getline(cin, input))
    {
        if (!input.empty())
        {
            vector<string> vec;
            Split(input, vec, ' ');
            cout << RegexSubStringPos(vec) << endl;
        }
    }
    return 0;
}

```

116、✓找到比自己强的人数

[[2,1],[3,2]], 每组表示师徒关系，第一个元素是第二个元素的老师，数字代表排名，**现在找出比自己强的徒弟**

输入

[[2,1],[3,2]]

输出

[0,1,2]

第一行数据[2,1]表示排名第 2 的员工是排名第 1 员工的导师，后面的数据以此类推。

输出结果：

第一个元素 0 表示成绩排名第一的导师，没有徒弟考试超过他；

第二个元素 1 表示成绩排名第二的导师，有 1 个徒弟成绩超过他

第三个元素 2 表示成绩排名第二的导师，有 2 个徒弟成绩超过他

```

#include <iostream>
#include <cstdlib>
#include <unordered_map>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void SplitInt(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}

void SplitString(string input, vector<string>& output, string pattern)
{
    string::size_type pos;
    input += pattern;
    for (int i = 0; i < input.size(); i++)
    {
        pos = input.find(pattern, i);
    }
}

```

```

        if (pos < input.size())
        {
            string temp = input.substr(i, pos - i);
            if ((temp != pattern) && (!temp.empty()))
            {
                output.push_back(temp);
            }
            i = pos + pattern.size() - 1;
        }
    }
}

vector<int> FindSomeoneBetterThanMyself(vector<vector<int>> martix)
{
    unordered_map<int,int> u_map;
    int master;
    int prentice;
    for (int i = 0; i < martix.size(); i++)
    {
        master = martix[i][0];
        for (int j = 0; j < martix.size(); j++)
        {
            prentice = martix[j][1];
            if (master > prentice)
            {
                u_map[master] += 1;
                u_map[prentice] += 0;
            }
        }
    }
    vector<int> vec;
    unordered_map<int, int>::iterator it = u_map.begin();
    for (; it != u_map.end(); it++)
    {
        vec.push_back(it->second);
    }
    sort(vec.begin(), vec.end());
    return vec;
}

int main(int argc, char** argv)
{
    string input;

    while (getline(cin,input))
    {
        if (!input.empty())
        {
            input = input.substr(2, input.size() - 2);
            vector<string> groups;
            string pattern = "],[ ";
            SplitString(input, groups, pattern);
            vector<vector<int>> martix;
            for (int i = 0; i < groups.size(); i++)
            {
                vector<int> vec;
                SplitInt(groups[i], vec, ',');
                martix.push_back(vec);
            }
            vector<int> outputs = FindSomeoneBetterThanMyself(martix);
            cout << "[";
            for (int i = 0; i < outputs.size(); i++)
            {
                cout << outputs[i];
                if (i < outputs.size()-1)
                {
                    cout << ",";
                }
            }
            cout << "]" << endl;
        }
    }
    return 0;
}

```

117、✓找出比自己大的数

给一个数组，首尾相连，按照顺序找到比自己大的第一个数，找不到的存-1。

例如[35,25,48,37] ->输出[48,48,-1,48]

```

#include <iostream>
#include <vector>
#include <string>

```

```

#include <algorithm>
using namespace std;

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i + 1;
        }
    }
}

int main(int argc, char** argv)
{
    while (1)
    {
        string input_string;
        vector<int> input_vec;
        input_vec.clear();
        cin >> input_string;
        input_string = input_string.substr(1, input_string.size() - 1);
        Split(input_string, input_vec, ',');
        auto max_pos = max_element(input_vec.begin(), input_vec.end());
        int max_value = *max_pos;
        cout << "[";
        for (int i = 0; i < input_vec.size(); i++)
        {
            if (input_vec[i] < max_value)
            {
                cout << max_value;
            }
            else
            {
                cout << -1;
            }
            if (i != input_vec.size() - 1)
            {
                cout << ",";
            }
        }
        cout << "]" << endl;
    }

    return 0;
}

```

118、✓输出制定字母在字符串的中的索引

给定一个字符串，把字符串按照大写在前小写在后排序，输出排好后的第 K 个字母在原来字符串的索引。相同字母输出第一个出现的位置。

例如：输入 hAkDAjByBq 4 输出 6 说明：排好序后 AABBDhjkqy，第 4 个是 B，第一个出现的在原字符串 6 这个位置。（索引是从 0 开始）

```

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int FindSourcePos(string source_string, int new_pos)
{
    string string_tempUp = "";
    string string_tempLow = "";
    for (int i = 0; i < source_string.size(); i++)
    {
        if ((source_string[i] >= 'A') && (source_string[i] <= 'Z'))
        {
            string_tempUp += source_string[i];
        }
        else
        {
            string_tempLow += source_string[i];
        }
    }
}

```

```

    }
    sort(string_tempUp.begin(), string_tempUp.end());
    sort(string_tempLow.begin(), string_tempLow.end());
    string new_string = string_tempUp + string_tempLow;
    int old_pos = source_string.find(new_string[new_pos-1]);
    return old_pos;
}

int main(int argc, char** argv)
{
    while (1)
    {
        string source_string;
        int new_pos;
        cin >> source_string >> new_pos;
        cout << FindSourcePos(source_string, new_pos) << endl;
    }
    return 0;
}

```

119、✓ 匹配子字符串的个数

给出一个母串，一个子串，母串中包含空格。计算出子串在母串中出现的次数。

例如: 母串: Akk bhah ahb bd, 子串 ha, 输出是 2

```

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

// 去掉字符串中所有空格
void RemoveStringAllSpace(string& source_string)
{
    int index = 0;
    if (!source_string.empty())
    {
        while ((index = source_string.find(' ', index)) != string::npos)
        {
            source_string.erase(index, 1);
        }
    }
}

// 字符串中子串出现次数
int GetSunStringCount(string source_string, string sub_string)
{
    RemoveStringAllSpace(source_string);
    int index = 0;    // 下标
    int count = 0;    // 次数

    // b.find(a); 这句代码的意思就是从b字符串中查找a字符串
    // 返回值的类型为int类型，返回的是字符串的下标
    // 如果没找到，返回一个特别的标志c++中用npos表示，string::npos很大的一个数，转成int值是-1

    while ((index = source_string.find(sub_string, index)) < source_string.length())
    {
        count++;
        index++;
    }
    return count;
}

int main(int argc, char** argv)
{
    while (1)
    {
        string source_string;
        string sub_string;
        getline(cin, source_string);
        getline(cin, sub_string);
        cout << GetSunStringCount(source_string, sub_string) << endl;
    }
    return 0;
}

```


120、求解特定以元音字母开头和结尾的子串长度 --比较难

给定一个字符串，找出元音字母开头和结尾的字符串，并且满足包含非元音字母个数为 n ，输出这个子串的长度，不满足的输出 0；

例如 adfei $n=2$ 输出为 5，本身这个串就是元音开头和结尾，并且 df 非元音，满足 2 个个数

121、✓ 求出排列的指定排序的序列

例如由前 k 个数组成的排列中的第 n 个排列

例如 $k=4$, $n=4$ 输出 1342 说明 1234 1243 1324 1342 1423 ...

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

string GetPermutation(int n, int k)
{
    vector<int> factorial(n);
    factorial[0] = 1;
    for (int i = 1; i < n; ++i)
    {
        factorial[i] = factorial[i - 1] * i;
    }
    --k;
    string output;
    vector<int> valid(n + 1, 1);
    for (int i = 1; i <= n; ++i)
    {
        int order = k / factorial[n - i] + 1;
        for (int j = 1; j <= n; ++j)
        {
            order -= valid[j];
            if (!order)
            {
                output += (j + '0');
                valid[j] = 0;
                break;
            }
        }
        k %= factorial[n - i];
    }
    return output;
}

int main(int argc, char** argv)
{
    while (1)
    {
        int k, n;
        cin >> k >> n;
        cout << GetPermutation(k, n) << endl;
    }
    return 0;
}
```

122、✓ 求子矩阵的最大和

这个题比较经典，给定一个矩阵，求出任意子矩阵的和，输出最大值，用到动态求解，网上都有实现的代码。主要是 $sum = p[j][m] - p[j][k-1] - p[i-1][m] + p[i-1][k-1]$ ，首先初始化 $p[i][j] = p[i-1][j] + p[i][j-1] - p[i-1][j-1] + a[i][j]$ 。

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstdlib>
#include <algorithm>
```

```

using namespace std;
long long a[550][550];
int main(int argc, char** argv)
{
    int N, M, t;
    long long sum, ans = 0;
    scanf("%d %d", &M, &N);
    memset(a, 0, sizeof(a));
    for (int i = 1; i <= N; i++)
    {
        for (int j = 1; j <= M; j++)
        {
            scanf("%d", &t);
            a[i][j] = a[i - 1][j] + t;
        }
    }
    for (int i = 1; i <= N; i++)
    {
        for (int j = i; j <= N; j++)
        {
            sum = 0;
            for (int k = 1; k <= M; k++)
            {
                sum = sum + (a[j][k] - a[i - 1][k]);
                if (sum < 0) sum = 0;
                if (sum > ans) ans = sum;
            }
        }
    }
    printf("%lld\n", ans);
    return 0;
}
}

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int MaxArray(int* a, int n);
int MaxMatrix(int** a, int m, int n)//int **a 指针数组,m*n矩阵
{
    int max_sum = 0;
    int* b = new int[n];
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            b[j] = 0;
        }
        for (int j = i; j < m; j++)
        {
            for (int k = 0; k < n; k++)
            {
                b[k] += a[j][k];
            }
            int sum = MaxArray(b, n);
            if (sum > max_sum)
            {
                max_sum = sum;
            }
        }
    }
    return max_sum;
}

int MaxArray(int* a, int n)
{
    int sum = 0;
    int b = 0;
    for (int i = 0; i < n; i++)
    {
        if (b < 0)
            b = a[i];
        else
            b += a[i];
        if (b > sum)
            sum = b;
    }
    return sum;
}

int main(int argc, char** argv)
{
    int b[5] = { 1, -1, 0, -3, 5 };
    cout << MaxArray(b, 5) << endl; //测试MaxArray方法是否正确
    //getchar();
    int m, n;
    cout << "输入矩阵的行数m:" << endl;

```

```

cin >> m;
cout << "输入矩阵的列数n:" << endl;
cin >> n;
int** a = new int* [m];
for (int i = 0; i < m; i++)
{
    a[i] = new int[n];
    for (int j = 0; j < n; j++)
    {
        cin >> a[i][j];
    }
}
cout << MaxMatrix(a, m, n);
return 0;
}

```

123、✓找最小值、最大值（移掉k位数字）

给出一个数字组成的字符串，去除指定个数的字符，剩余的组成最小值

这个解题思路可以是递归，按照顺序取剩余个数的字符，组成数字，依次比较获取，但是可能超时。

例如 6525441，去除 4 个字符，得到 241。

网上有一个更简单的办法，最优解是删除出现的第一个左边>右边的数，例如第一次删除 6，因为 6>5，变成 525441，第二次删除 5，5>2，变成 25441，第三次是 5，第四次是 4，结果为 241。

```

#include<iostream>
#include<vector>
#include<string>
#include<queue>
using namespace std;

//移掉k位数字使得到的最小
string RemoveKdigitsMinValue(string num, int k)
{
    vector<char> stk;
    for (auto& digit : num)
    {
        while (stk.size() > 0 && stk.back() > digit && k)
        {
            stk.pop_back();
            k--;
        }
        stk.push_back(digit);
    }

    for (; k > 0; --k)
    {
        stk.pop_back();
    }

    string ans = "";
    bool isLeadingZero = true;
    for (auto& digit : stk)
    {
        if (isLeadingZero && digit == '0')
        {
            continue;
        }
        isLeadingZero = false;
        ans += digit;
    }
    return ans == "" ? "0" : ans;
}

//保留k位最大的子序列（移除n-k位）
string ReserveKdigitsMaxValue(string num, int k)
{
    string ans = "";
    int n = num.Length();
    if (k >= n)
    {
        ans = num;
    }
    vector<int> stk;
    int pop = n - k; //要移除n-k个数
}

```

```

    for (int i = 0; i < n; i++)
    {
        int number = num[i] - '0';
        while (!stk.empty() && number > stk.back() && pop > 0)
        {
            stk.pop_back();
            pop--;
        }
        stk.push_back(number);
    }
    stk.resize(k);

    for (int i = 0; i < k; i++)
    {
        ans += to_string(stk[i]);
    }
    return ans;
}

int main(int argc, char** argv)
{
    string res1 = RemoveKdigitsMinValue("6525441", 4);
    cout << res1 << endl;
    string res2 = ReserveKdigitsMaxValue("6525441", 3); // 求保留3位的最大值，也就是移除4位
    cout << res2 << endl;
    return 0;
}

```

124、✓ 括号深度题

现有一字符串仅由 '(', ')', '{', '}', '[', ']' 六种括号组成。

若字符串满足以下条件之一，则为无效字符串：

- ①任一类型的左右括号数量不相等；
- ②存在未按正确顺序（先左后右）闭合的括号。

输出括号的最大嵌套深度，若字符串无效则输出 0。

0 ≤ 字符串长度 ≤ 100000

输入描述：

一个只包括 '(', ')', '{', '}', '[', ']' 的字符串

输出描述：

一个整数，最大的括号深度

示例 2

输入

([{}])

输出

3

```

#include <iostream>
#include <string>
#include <stack>
#include <unordered_map>
#include <algorithm>
using namespace std;

bool ValidMaxDepth(string s, int& depth)
{
    int n = s.size();
    if (n % 2 == 1)
    {
        return false;
    }

    unordered_map<char, char> pairs = {
        {'}', '('},
        {'}', '('},
        {'}', '('},
        {'}', '('}
    };
    stack<char> stk;
}

```

```

    for (char ch : s)
    {
        if (pairs.count(ch))
        {
            if (stk.empty() || stk.top() != pairs[ch])
            {
                return false;
            }
            depth = max(depth, (int)stk.size());
            stk.pop();
        }
        else
        {
            stk.push(ch);
        }
    }
    return stk.empty();
}

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        int depth = 0;
        bool flag = ValidMaxDepth(input_string, depth);
        if (flag)
        {
            cout << depth << endl;
        }
        else
        {
            cout << 0 << endl;
        }
    }

    return 0;
}

```

125、✓ 翻转文章片段

输入一个英文文章片段，翻转指定区间的单词顺序，标点符号和普通字母一样处理。例如输入字符串"I am a developer. "，区间[0,3]，则输出"developer. a am I"。

输入描述:

使用换行隔开三个参数，第一个参数为英文文章内容即英文字符串，第二个参数为翻转起始单词下标(下标从 0 开始)，第三个参数为结束单词下标。

输出描述:

翻转后的英文文章片段所有单词之间以一个半角空格分隔进行输出

示例 1

输入

I am a developer.

1

2

输出

I a am developer.

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

void Split(string input, vector<string>& output, char patten);

int main(int argc, char** argv)
{
    string str, temp;

```

```

getline(cin, str);
int pos1, pos2;
cin >> pos1;
cin >> pos2;
vector<string> s_vector;

Split(str, s_vector, ' ');

for (int i = 0; i < pos1; i++)
{
    cout << s_vector[i] << " ";
}

for (int i = pos2; i >= pos1; i--)
{
    cout << s_vector[i] << " ";
}

for (int i = pos2+1; i < s_vector.size(); i++)
{
    cout << s_vector[i] << " ";
}

return 0;
}

void Split(string input, vector<string>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(one_word);
            one_word = "";
            j = i;
        }
    }
}

```

126、连连 check（连连看）

连连看游戏大家应该都玩过，选择图案相同的两个格子，如果这两个格子可以用三根以内的线段连在一起，则这两个格子可以消除。

给定一个 $N \times M$ 的数组 `arr`，代表 $N \times M$ 个格子，若 `arr[i][j] == 0`，表示该格子已经被清空；若 `arr[i][j] == k > 0`，表示该格子的图案为 `k`。现要求实现一个算法，输入一个连连看内容数组 `arr` 和一系列消除步骤，判断这些消除步骤是否合法。若全部消除步骤都合法，输出 `ok`；若首次遇到不合法的步骤，输出该步骤的编号（步骤从 0 开始编号）

输入描述：

第一行是数组 `arr` 的行数 `N`，接下来的 `N` 行每行是‘,’分隔的 `M` 个数字，表示每行的 `M` 列
 接下来一行是消除步骤的数量 `S`，接下来的 `S` 行每行是‘,’分隔的 4 个数字，`x1,y1,x2,y2`，
 表示这一步选择 `arr[y1][x1]`和 `arr[y2][x2]`这两个格子尝试消除

输出描述：

若全部消除步骤都合法，输出 `ok`，否则输出首次出现的不合法步骤的编号

输入

2

4,5,4

5,4,5

2

0,0,2,0

1,0,0,1

输出

ok

下面给出示例1和示例2的图示

示例1

1. 一开始的图案分布

4	5	4
5	4	5

2. 0,0,2,0选取如图的两个格子，它们的图案都是4，而且可以通过3根线段连接，故消除步骤合法

4	5	4
5	4	5

3. 第1步消除后的图案分布

	5	
5	4	5

4. 1,0,0,1选取如图的两个格子，它们的图案都是5，而且可以通过2根线段连接，故消除步骤合法

	5	
5	4	5

<https://blog.csdn.net/ins9991>

127、✓String 加密

给你一串未加密的字符串 `str`，通过对字符串的每一个字母进行改变来实现加密，加密方式是在每一个字母 `str[i]` 偏移特定数组元素 `a[i]` 的量，数组 `a` 前三位已经赋值：`a[0]=1,a[1]=2,a[2]=4`。当 $i \geq 3$ 时，数组元素 $a[i]=a[i-1]+a[i-2]+a[i-3]$ ，例如：原文 `abcde` 加密后 `bdgkr`，其中偏移量分别是 1,2,4,7,13。

输入描述:

第一行为一个整数 n ($1 \leq n \leq 1000$)，表示有 n 组测试数据，每组数据包含一行，原文 `str` (只含有小写字母， $0 < \text{长度} \leq 50$)。

输出描述:

每组测试数据输出一行，表示字符串的密文

示例 1

输入

1

xy

输出

ya

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void encrypt(string& str)
{
    vector<int> array;
    array.push_back(1);
    array.push_back(2);
    array.push_back(4);
    int length = str.size();
    for (int i = 0; i < 3; i++)
    {
        str[i] = str[i] + array[i];
        if (str[i] > 'z')
        {
            str[i] = 'a';
        }
    }
    for (int i = 3; i < length; i++)
    {
        array[i] = array[i - 1] + array[i - 2] + array[i - 3];
        str[i] += array[i];
        if (str[i] > 'z')
        {
            str[i] = 'a';
        }
    }
}
```

```

}

int main(int argc, char** argv)
{
    int numbers = 0;
    string str;
    vector<string> input;
    while (1)
    {
        input.clear();
        cin >> numbers;
        for (int i = 0; i < numbers; i++)
        {
            cin >> str;
            input.push_back(str);
        }
        for (int i = 0; i < numbers; i++)
        {
            string temp = input[i];
            encrypt(temp);
            cout << temp << endl;
        }
    }

    return 0;
}

```

128、✓ 区间重叠问题

给定一组闭区间，其中部分区间存在交集。任意两个给定区间的交集，称为公共区间（如：[1,2],[2,3]的公共区间为[2,2]，[3,5],[3,6]的公共区间为[3,5]）。公共区间之间若存在交集，则需要合并（如：[1,3],[3,5]区间存在交集[3,3]，须合并为[1,5]）。按升序排列输出合并后的区间列表。

输入描述:

一组区间列表，

区间数为 N:

$0 \leq N \leq 1000$;

区间元素为 X:

$-10000 \leq X \leq 10000$ 。

输出描述:

升序排列的合并后区间列表

示例 1

输入

0 3

1 3

3 5

3 6

输出

1 5

说明

[0,3]和[1,3]的公共区间为[1,3]，[0,3]和[3,5]的公共区间为[3,3]，[0,3]和[3,6]的公共区间为[3,3]，[1,3]和[3,5]的公共区间为[3,3]，[1,3]和[3,6]的公共区间为[3,3]，[3,5]和[3,6]的公共区间为[3,5]，公共区间列表为[[1,3],[3,3],[3,5]]；[1,3],[3,3],[3,5]存在交集，须合并为[1,5]。

```

#include <bits/stdc++.h>
using namespace std;
int main(int argc, char** argv)
{
    int n, ans = 0; // ans存储最终结果
    scanf("%d", &n);
    vector<pair<int, int>> v1(n), v2(n); // 分别存储两个区间
    for (int i = 0; i < n; ++i)

```



```

{
    scanf("%d%d", &v1[i].first, &v1[i].second);
}
for (int i = 0; i < n; ++i)
{
    scanf("%d%d", &v2[i].first, &v2[i].second);
}
for (auto p1 : v1)
{
    for (auto p2 : v2)
    {
        if (p1.first <= p2.second && p1.second >= p2.first)
        {
            ans += min(p1.second, p2.second) - max(p1.first, p2.first); //加上重叠区间
        }
    }
}
printf("%d", ans);
return 0;
}

```

```

#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <algorithm>
using namespace std;

// 交集
vector<vector<int>> Intersection(vector<vector<int>> intervals)
{
    if (intervals.empty())
    {
        return vector<vector<int>> {};
    }

    vector<vector<int>> res;
    int left = intervals[0][0];
    int right = intervals[0][1];
    for (int i = 0; i < intervals.size(); ++i)
    {
        if (i == 0)
        {
            continue;
        }
        if (intervals[i][0] < right)
        {
            left = max(intervals[i][0], left);
            right = min(intervals[i][1], right);

            res.push_back(vector<int>{left, right});
        }
        if (intervals[i][0] >= right)
        {
            left = intervals[i][0];
            right = intervals[i][1];
        }
    }
    return res;
}

// 合并
vector<vector<int>> Merge(vector<vector<int>> intervals)
{
    if (intervals.size() == 0)
    {
        return {};
    }
    vector<vector<int>> merged;
    for (int i = 0; i < intervals.size(); ++i)
    {
        int L = intervals[i][0], R = intervals[i][1];
        if (!merged.size() || merged.back()[1] < L)
        {
            merged.push_back({ L, R });
        }
        else
        {
            merged.back()[1] = max(merged.back()[1], R);
        }
    }
    return merged;
}

void Split(string input, vector<int>& output, char patten)
{

```

```

string one_word = "";
int j = 0;
input += patten;
for (int i = 0; i < input.size(); i++)
{
    if (input[i] == patten)
    {
        one_word = input.substr(j, i - j);
        output.push_back(stoi(one_word));
        one_word = "";
        j = i + 1;
    }
}

int main(int argc, char** argv)
{
    vector<vector<int>> intervals;
    string input;
    while (getline(cin, input))
    {
        if (!input.empty())
        {
            vector<int> vec;
            Split(input, vec, ' ');
            intervals.push_back(vec);
        }
        else
        {
            break;
        }
    }
    vector<vector<int>> temps = Intersection(intervals);
    vector<vector<int>> outputs = Merge(temps);
    for (int i = 0; i < outputs.size(); i++)
    {
        for (int j = 0; j < outputs[i].size(); j++)
        {
            cout << outputs[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}

```

129、√ 面试题（最少面试官数）

某公司组织一场公开招聘活动，假设由于人数和场地的限制，每人每次面试的时长不等，并已经安排给定，用(S1,E1)、(S2,E2)、(Sj,Ej)...(Si < Ei, 均为非负整数)表示每场面试的开始和结束时间。面试采用一对一的方式，即一名面试官同时只能面试一名应试者，一名面试官完成一次面试后可以立即进行下一场面试，且每个面试官的面试人次不超过 m。

为了支撑招聘活动高效顺利进行，请你计算至少需要多少名面试官

输入描述:

输入的第一行为面试官的最多面试人次 m，第二行为当天总的面试场次 n，接下来的 n 行为每场面试的起始时间和结束时间，起始时间和结束时间用空格分隔。

其中，1 ≤ n, m ≤ 500

输出描述:

输出一个整数，表示至少需要的面试官数量。

示例 1

输入

2

5

1 2

2 3

3 4

4 5

5 6

输出

3

说明

总共有 5 场面试，且面试时间都不重叠，但每个面试官最多只能面试 2 人次，所以需要 3 名面试官。

```
#include<iostream>
#include<vector>
#include<string>
#include<queue>
#include <algorithm>
using namespace std;

int MinInterviewers(vector<vector<int>> intervals,int max_meets)
{
    if (intervals.empty())
    {
        return 0;
    }
    int len = intervals.size();
    int start_time = intervals[0][0];
    int end_time = intervals[0][1];
    int min_interviewers = 1;
    for (int i = 1; i < intervals.size(); ++i)
    {
        if (intervals[i][0] < end_time)
        {
            min_interviewers++;
        }
        else
        {
            if ((i + 1) % max_meets == 0)
            {
                min_interviewers++;
            }
        }
    }
    return min_interviewers;
}

int main(int argc, char** argv)
{
    vector<vector<int>> intervals;

    int max_meets = 0;
    int meets = 0;
    cin >> max_meets;
    cin >> meets;
    for (int i = 0; i < meets; i++)
    {
        int start, end;
        vector<int> vec;
        cin >> start >> end;
        vec.push_back(start);
        vec.push_back(end);
        intervals.push_back(vec);
    }

    cout << MinInterviewers(intervals, max_meets) << endl;

    return 0;
}
```

130、✓ Directory 删除

某文件系统中存在 N 个目录，每个目录都有一个独一无二的 ID。每个目录只有一个父目录，但每个父目录下可以有零个或者多个子目录，目录结构呈树状结构。

假设，根目录的 ID 为 0，且根目录没有父目录，其他所有目录的 ID 用唯一的正整数表示，并统一编号。

现给定目录 ID 和其父目录 ID 的对应父子关系表[子目录 ID，父目录 ID]，以及一个待删除

的目录 ID，请计算并返回一个 ID 序列，表示因为删除指定目录后剩下的所有目录，返回的 ID 序列以递增序输出。

注意：

- 1、被删除的目录或文件编号一定在输入的 ID 序列中；
- 2、当一个目录删除时，它所有的子目录都会被删除。

输入描述：

输入的第一行为父子关系表的长度 **m**；接下来的 **m** 行为 **m** 个父子关系对；最后一行为待删除的 ID。序列中的元素以空格分割，参见样例。

输出描述：

输出一个序列，表示因为删除指定目录后，剩余的目录 ID。

示例 1

输入

```
5
8 6
10 8
6 0
20 8
2 6
8
```

输出

```
2 6
```

目录结构如下所示：

```
  6
 / \
2   8
 / \
10  20
```

删除目录8，同时它的子目录10也被删除，剩余2和6两个目录。

```
#include<iostream>
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;

int main(int argc, char** argv)
{
    int n;
    int temp;
    int delete_value = 0;
    int delete_index = 0;
    vector<vector<int>> > count;    //to creat a array
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        vector<int> MiddleUse;
        cin >> temp;
        MiddleUse.push_back(temp);
        cin >> temp;
        MiddleUse.push_back(temp);
        count.push_back(MiddleUse); //input the array
    }
    cin >> delete_value;
    sort(count.begin(), count.end()); //sort to calculate

    for (int i = 0; i < n - 1; i++)
    {
        if (count[i][0] == delete_value)
        {
            delete_index = i;
        }
    }

    for (int k = 0; k < delete_index; k++)
    {
```

```

        if (count[k][1] == 0)
        {
            continue;
        }
        cout << count[k][0] << " " << count[k][1] << endl;
    }
    return 0;
}

```

131、✓连续数列

已知连续正整数数列 $\{K\}=K_1, K_2, K_3 \dots K_i$ 的各个数相加之和为 S ， $i=N$ ($0 < S < 100000$, $0 < N < 100000$)，求此数列 K 。

输入描述:

输入包含两个参数，1) 连续正整数数列和 S ，2) 数列里数的个数 N 。

输出描述:

如果有解输出数列 K ，如果无解输出-1

示例 1

输入

525 6

输出

85 86 87 88 89 90

```

#include<iostream>
#include<vector>
#include<string>
#include<queue>
#include <algorithm>
using namespace std;

vector<vector<int>> FindContinuousSequence(int target,int N)
{
    vector<vector<int>> vec;
    vector<int> res;
    int sum = 0, limit = (target - 1) / 2;
    for (int i = 1; i <= limit; ++i)
    {
        for (int j = i; ++j)
        {
            sum += j;
            if (sum > target)
            {
                sum = 0;
                break;
            }
            else if (sum == target)
            {
                res.clear();
                for (int k = i; k <= j; ++k)
                {
                    res.emplace_back(k);
                }
                if (res.size()==N)
                {
                    vec.emplace_back(res);
                }
                sum = 0;
                break;
            }
        }
    }
    return vec;
}

int main(int argc, char** argv)
{
    vector<vector<int>> intervals;

    int sum = 0;
    int N = 0;
    cin >> sum >> N;
}

```

```

intervals = FindContinuousSequence(sum, N);
for (int i = 0; i < intervals.size(); i++)
{
    cout << "[";
    for (int j = 0; j < intervals[i].size(); j++)
    {
        cout << intervals[i][j];
        if (j < intervals[i].size() - 1)
        {
            cout << ",";
        }
    }
    cout << "]";
}

return 0;
}

```

132、✓ 数字排列组合

小明负责公司年会，想出一个趣味游戏：

屏幕给出 1~9 中任意 3 个不重复的数字，大家以最快时间给出这几个数字可拼成的数字从小到大排列位于第 N 位置的数字，其中 N 为给出的数字中最大的（如果不到这么多个数字则给出最后一个即可）。

注意：

- 1) 2 可以当做 5 来使用，5 也可以当做 2 来使用进行数字拼接，且屏幕不能同时给出 2 和 5；
- 2) 6 可以当做 9 来使用，9 也可以当做 6 来使用进行数字拼接，且屏幕不能同时给出 6 和 9。

如给出：1，4，8，则可以拼成的数字为：

1，4，8，14，18，41，48，81，84，148，184，418，481，814，841

那么最第 N（即 8）个的数字为 81。

输入描述：

输入为以逗号分隔的描述 3 个 int 类型整数的字符串。

输出描述：

输出为这几个数字可拼成的数字从小到大排列位于第 N（N 为输入数字中最大的数字）位置的数字，如果输入的数字不在范围内或者有重复，则输出-1。

示例 1

输入

1,4,8

输出

81

说明

可以构成的数字按从小到大排序为 1，4，8，14，18，41，48，81，84，148，184，418，481，814，841，故第 8 个为 81

```

#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

bool CeckRepet(vector<int> vec)
{
    int repet_check2_5 = count(vec.begin(), vec.end(), 2);
    if (repet_check2_5 >= 2)
    {
        return true;
    }
    int repet_check5_2 = count(vec.begin(), vec.end(), 5);
    if (repet_check5_2 >= 2)

```

```

{
    return true;
}
if ((repet_check2_5 == 1)&&(repet_check5_2 == 1))
{
    return true;
}
int repet_check6_9 = count(vec.begin(), vec.end(), 6);
if (repet_check6_9 >= 2)
{
    return true;
}
int repet_check9_6 = count(vec.begin(), vec.end(), 9);
if (repet_check9_6 >= 2)
{
    return true;
}
if ((repet_check6_9 == 1) && (repet_check9_6 == 1))
{
    return true;
}
return false;
}

int MaxPosPermutationValue(vector<int> vec)
{
    bool flag = CeckRepet(vec);
    if (flag)
    {
        return -1;
    }
    int max_value_pos = *max_element(vec.begin(), vec.end());

    vector<int> permutation;
    int repet_check = vec[0];

    for (int i = 0; i < vec.size(); i++)
    {
        if (i > 0)
        {
            if (repet_check == vec[i])
            {
                return -1;
            }
        }
        permutation.push_back(vec[i]);
    }
    for (int i = 0; i < vec.size(); i++)
    {
        for (int j = 0; j < vec.size(); j++)
        {
            if (i != j)
            {
                string temp_string = to_string(vec[i]) + to_string(vec[j]);
                int temp_int = stoi(temp_string);
                permutation.push_back(temp_int);
            }
        }
    }

    do {
        string temp_string = "";
        for (int i = 0; i < vec.size(); i++)
        {
            temp_string += to_string(vec[i]);
        }
        int temp_int = stoi(temp_string);
        permutation.push_back(temp_int);
    } while (next_permutation(vec.begin(), vec.end()));

    sort(permutation.begin(), permutation.end());
    if ((max_value_pos-1) >= permutation.size())
    {
        return -1;
    }
    else
    {
        return permutation[max_value_pos - 1];
    }
}

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;

```

```

        input += patten;
        for (int i = 0; i < input.size(); i++)
        {
            if (input[i] == patten)
            {
                one_word = input.substr(j, i - j);
                if (!one_word.empty())
                {
                    output.push_back(stoi(one_word));
                }
                one_word = "";
                j = i + 1;
            }
        }
    }
}

int main(int argc, char** argv)
{
    string line;
    while (getline(cin, line))
    {
        if (!line.empty())
        {
            vector<int> vec;
            Split(line, vec, ',');
            cout << MaxPosPermutationValue(vec) << endl;
        }
    }
    return 0;
}

```

133、✓阿里集五福

集五福作为近年来大家喜闻乐见迎新春活动，集合爱国福、富强福、和谐福、友善福、敬业福即可分享超大红包

以 0 和 1 组成的长度为 5 的字符串代表每个人所得到的福卡，每一位代表一种福卡，1 表示已经获得该福卡，单类型福卡不超过 1 张，随机抽取一个不超过 10 人团队，求该团队最多可以集齐多少套五

输入描述:

输入类似 11010,001110，由 0、1 组成的长度为 5 的字符串，代表指定团队中每个人福卡获得情况

注意 1: 1 人也可以是一个团队

注意 2: 多人之间的福卡以英文逗号隔开

输出描述:

输出该团队能凑齐多少套五福

示例 1

输入

11001,11100

输出

0

说明

第 4 个福卡无人获得，无法凑齐五福

```

#include<iostream>
#include<string>
#include<vector>
using namespace std;

void CollectionWufu(string input_string, int& output_counts);
void Split(string input, vector<string>& output, char patten);

int main(int argc, char** argv)
{
    string input_string;
    int wufu_counts = 0;
    while (cin >> input_string)

```



```

    {
        CollectionWufu(input_string, wufu_counts);
        cout << wufu_counts << endl;
    }

    return 0;
}

void CollectionWufu(string input_string, int& output_counts)
{
    output_counts = 0;
    vector<int> positions;
    vector<string> vector_string;

    Split(input_string, vector_string, ',' );

    for (int i = 0; i < vector_string.size(); i++)
    {
        int pos = 0;
        bool flag = true;
        if (pos = vector_string[i].find("0", 0) != string::npos)
        {
            flag = false;
        }
        if (flag == true)
        {
            output_counts++;
        }
    }
}

void Split(string input, vector<string>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(one_word);
            one_word = "";
            j = i + 1;
        }
    }
}

```

134、✓堆栈中的剩余数字

向一个空栈中依次存入正整数，假设入栈元素 $n(1 \leq n \leq 2^{31}-1)$ 按顺序依次为 $n_x \dots n_4, n_3, n_2, n_1$ ，每当元素入栈时，如果 $n_1 = n_2 + \dots + n_y$ (y 的范围 $[2, x]$, $1 \leq x \leq 1000$)，则 $n_1 \sim n_y$ 全部元素出栈，重新入栈新元素 $m(m = 2 * n_1)$ 。

如：依次向栈存入 6、1、2、3，当存入 6、1、2 时，栈底至栈顶依次为 [6、1、2]；当存入 3 时， $3=2+1$ ，3、2、1 全部出栈，重新入栈元素 6 ($6=2*3$)，此时栈中有元素 6；因为 $6=6$ ，所以两个 6 全部出栈，存入 12，最终栈中只剩一个元素 12。

输入描述：

使用单个空格隔开的正整数的字符串，如 "5 6 7 8"，左边的数字先入栈，输入的正整数个数为 x , $1 \leq x \leq 1000$ 。

输出描述：

最终栈中存留的元素值，元素值使用空格隔开，如 "8 7 6 5"，栈顶数字在左边。6 1 2 3

示例 1

输入

5 10 20 50 85 1

输出

1 170

说明

5+10+20+50=85，输入 85 时，5、10、20、50、85 全部出栈，入栈 170，最终依次出栈的数字为 1 和 170。

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

void Push(vector<int>& intLists)
{
    int sum = 0;
    int pop_index = 0;
    for (int j = 0; j < intLists.size(); j++)
    {
        sum += intLists[j];
        if (j < intLists.size() - 1)
        {
            if (sum == intLists[j + 1])
            {
                pop_index = j + 1;
                int added_integer = intLists[j + 1] * 2;
                for (int k = 0; k <= pop_index; k++)
                {
                    intLists.erase(intLists.begin());
                }
                intLists.push_back(added_integer);
                j = 0;
                sum = 0;
            }
        }
    }

    for (auto result : intLists)
    {
        cout << result << " ";
    }
}

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            if (!one_word.empty())
            {
                output.push_back(stoi(one_word));
            }
            one_word = "";
            j = i + 1;
        }
    }
}

int main(int argc, char** argv)
{
    string input_string;
    while (getline(cin, input_string))
    {
        vector<int> vec;
        Split(input_string, vec, ' ');
        Push(vec);
    }
    return 0;
}
```

135、数据分组

给一个数组对元素分组，要求只有满足 1、每组里的数据都相等；2、每组的元素个数都是 N（N>=2），满足两个条件才允许分组，返回从小到大的分组。

例如例 1、[1,3,3,1] ->[[1,1],[2,2]]；例 2、[1,1,4,3,3] -> []

136、✓求子集的最大值

给一个数组，找出相邻元素的和最大。

例如 例 1、[1,-2,4,8,-1,3,-4] -> 最大为 14 因为相邻的 4,8,-1,3 之和

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

int MaxSubArraySum(vector<int> nums)
{
    int sum = nums[0];
    for (int i = 1; i < nums.size(); ++i)
    {
        if (nums[i - 1] > 0)
        {
            nums[i] = nums[i - 1] + nums[i];
        }
        sum = max(sum, nums[i]);
    }
    return sum;
}

void Split(string input, vector<int>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(stoi(one_word));
            one_word = "";
            j = i + 1;
        }
    }
}

int main(int argc, char** argv)
{
    string input_string;
    while (cin >> input_string)
    {
        input_string = input_string.substr(1, input_string.size() - 2);
        vector<int> vec;
        Split(input_string, vec, ',');
        cout << MaxSubArraySum(vec) << endl;
    }

    return 0;
}
```

137、找出距离远点最大的距离

给一串字符串，里边有合法的坐标，找到最远的坐标，注意判断坐标的不合法((1,01),(0,100)等)，如果多个最远输出最早的一次。

例如 例 1、faaa(3,01)art53(7,4)dfgh4w(5,10) ->(5,10)

138、回文串 -题目比较难

先给出一个不是回文串的字符串，并且子串也是不包含回文串（大于等于 2），请找出下一个字典序，满足子串也是不包含回文串的字符串。

cba→NO，说明长度相同的字符串 cbb 和 cbc，由于都有回文串不可接受，所以输出 NO。

139、✓找出过特定点的路劲长度

输入一个字符串，都是以大写字母组成，每个相邻的距离是 1，第二个输入一个字符串，表示必过的点。说明每个点可过多次。

输入

ANTSEDXQOKPUVGIFWHJLYMCRZB

ABC

输出 28

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    while (1)
    {
        string source_string;
        string path_string;
        cin >> source_string;
        cin >> path_string;
        vector<char> points;
        for (int i = 0; i < path_string.size(); i++)
        {
            points.push_back(path_string[i]);
        }
        int distance_path = 0;
        for (int i = 0; i < points.size()-1; i++)
        {
            int start = source_string.find(points[i]);
            int end = source_string.find(points[i + 1]);
            int temp = abs(end - start);
            distance_path += temp;
        }
        cout << distance_path << endl;
    }
    return 0;
}
```

140、找出一条线上的相同元素个数 -比较难

给一个矩阵，里边包含两种不同的字符，需要你找出水平、垂直、对角线、反对角线上相同元素最多的个数。

输入

3,4

F,G,G,F

F,G,G,F

F,F,F,G

输出

3

141、✓按字符串里的数字排序

给出的字符串结构是前边是字母、后边为数字，例如 wew0145，要求给出一组数据，按照后边的数字从小到大排序。

输入

wr0068,lt01,qhy0027828,gh20425256,xj0033258,zmj00505562

输出

lt01,wr0068,qhy0027828,xj0033258,zmj00505562,gh20425256

说明

按照报名名单中员工工号中的数字部分升序排序输出。

备注:输入名单中工号中数字都是唯一的。

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

typedef pair<int, string> PAIR;

struct CmpByKey
{
    bool operator()(const PAIR& lhs, const PAIR& rhs)
    {
        return lhs.first < rhs.first;
    }
};

void Split(string input, vector<string>& output, char patten)
{
    string one_word = "";
    int j = 0;
    input += patten;

    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == patten)
        {
            one_word = input.substr(j, i - j);
            output.push_back(one_word);
            one_word = "";
            j = i+1;
        }
    }
}

vector<PAIR> Sort(vector<string> vec_jobIds)
{
    map<int, string> mp;
    for (int i = 0; i < vec_jobIds.size(); i++)
    {
        string number = "";
        for (int j = 0; j < vec_jobIds[i].size(); j++)
        {
            if ((vec_jobIds[i][j] >= '0') && (vec_jobIds[i][j] <= '9'))
            {
                number += vec_jobIds[i][j];
            }
        }
        mp[stoi(number)] = vec_jobIds[i];
    }
    vector<PAIR> vec(mp.begin(), mp.end());
    sort(vec.begin(), vec.end(), CmpByKey());
    return vec;
}

int main(int argc, char** argv)
{
    while (1)
    {
        string line;
        vector<string> vec_jobIds;
        while (getline(cin, line))
        {
```

```

        if (!line.empty())
        {
            Split(line, vec_jobIds, ',');
        }
        else
        {
            break;
        }
    }

    vector<PAIR> vec = Sort(vec_jobIds);

    for (int i = 0; i < vec.size(); i++)
    {
        cout << vec[i].second;
        if (i != vec.size() - 1)
        {
            cout << ", ";
        }
        cout << endl;
    }
    return 0;
}

```

142、√整数分解之素数

给定一个 32 位正整数，对其进行因数分解，分解成两个素数的乘积，如果不能分解输出-1，-1。

输入

15

输出

3, 5

```

#include <iostream>
#include <string>
#include <vector>
#include <math.h>

using namespace std;
bool IsPrime(int n)
{
    if (n <= 1)
    {
        return false;
    }
    else if (n == 2)
    {
        return true;
    }
    else if (n % 2 == 0)
    {
        return false;
    }
    else
    {
        int maxsq = (int)sqrt(n) + 1;
        for (int i = 3; i < maxsq; i = i + 2)
        {
            if (n % i == 0)
            {
                return false;
            }
        }
        return true;
    }
}

void Prime(int n, vector<int>& vec)
{
    if (n <= 1)
    {
        return;
    }
    for (int i = 2; i < n; ++i)
    {
        if (IsPrime(i))
        {
            vec.push_back(i);
        }
    }
}

```

```

    }
    return;
}

void IsDecode(int n, vector<int> prime_vec, vector<int>& decode_prime_vec)
{
    for (int i = 0; i < prime_vec.size(); i++)
    {
        if (n % prime_vec[i] == 0)
        {
            decode_prime_vec.push_back(prime_vec[i]);
            n = n / prime_vec[i];
        }
    }
}

int main(int argc, char** argv)
{
    vector<int> prime_vec;
    vector<int> decode_prime_vec;
    int n;
    cin >> n;
    Prime(n, prime_vec);
    IsDecode(n, prime_vec, decode_prime_vec);
    if (decode_prime_vec.empty())
    {
        cout << -1 << " " << -1 << endl;
        return 0;
    }
    for (int i = 0; i < decode_prime_vec.size(); i++)
    {
        cout << decode_prime_vec[i];
        if (i != decode_prime_vec.size()-1)
        {
            cout << " ";
        }
    }
    cout << endl;
    return 0;
}

```

143、替换字母，找出字典序最小的回文串

给定一个只包含小写字母的字符串，你可以修改任意位置的字符（变换为 a-z 中任一），然后重新排列字符串。现在要求用最少数次的修改得到一个回文串，若有多种方案，输出字典序最小的方案。

```

#include <iostream>
#include <stdio.h>
#include <string.h>
#include <cstring>
using namespace std;
const int N = 2e5 + 5;
char s[N];
int num[26] = { 0 };
// int mark[26]={0};
char trans[26] = { 0 };

int main(int argc, char** argv)
{
    scanf("%s", s);
    int len = strlen(s);
    int unuse = 0; char mid = '0';
    for (int i = 0; i < len; i++)
    {
        num[s[i] - 'a']++;
    }
    for (int i = 0; i < 26; i++)
    {
        if (num[i] % 2)
        {
            trans[++unuse] = 'a' + i;
        }
    }
    /**cout << unuse << endl;*/
    for (int i = 1; i <= unuse / 2; i++)
    {
        num[trans[i] - 'a']++;
        num[trans[unuse - i + 1] - 'a']--;
    }
    if (len % 2) mid = trans[unuse / 2 + 1];
    int F = 0, B = len - 1; // 我们最后是通过记录改变完字母后的每个字母的出现次数来按照字典序重排字符串的
}

```

```

for (int i = 0; i < 26; i++)
{
    /**cout << num[i] << " ";*/
    for (int j = F; j < F + num[i] / 2; j++)
    {
        s[j] = 'a' + i;
    }
    F += num[i] / 2;
    for (int j = B; j > B - num[i] / 2; j--)
    {
        s[j] = 'a' + i;
    }
    B -= num[i] / 2;
}
if (len % 2)
{
    s[len / 2] = mid;
}
cout << s << endl;
return 0;
}

```

144、字符串“漂亮度”

给出一个名字，该名字有 26 个字符串组成，定义这个字符串的“漂亮度”是其所有字母“漂亮度”的总和。

每个字母都有一个“漂亮度”，范围在 1 到 26 之间。没有任何两个字母拥有相同的“漂亮度”。字母忽略大小写。

给出多个名字，计算每个名字最大可能的“漂亮度”。

输入：

zhangsan

lisi

输出：

192

101

```

//26个字符串组成
#include<vector>
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
vector<pair<int, char> > v;
int cnt[30] = { 0 };
int main(int argc, char** argv)
{
    int num;
    while (cin >> num)
    {
        while (num--)
        {
            string s;
            cin >> s;
            for (int i = 0; i < s.size(); i++)
            {
                s[i] = toupper(s[i]); //全部转化为大写处理
            }
            for (int i = 0; i < s.size(); i++) //字符个数统计
            {
                cnt[s[i] - 65]--; //对个数取反，后面就可以逆序排序
            }
            sort(cnt, cnt + 30);
            int sum = 0, k = 26;
            for (int i = 0; i < 30 && cnt[i] != 0; i++)
            {
                sum += (0 - cnt[i]) * k;
                k--;
            }
            cout << sum << "\n";
            for (int i = 0; i < 30; i++)
            {
                cnt[i] = 0;
            }
        }
    }
}

```



```
}  
    return 0;  
}
```