二、经典题

排序

第1题

题目描述：

> 对输入的n 个数进行排序并输出。

输入：

> 输入的第一行包括一个整数n(1<=n<=100)。接下来的一行包括n 个整数。

输出：

> 可能有多组测试数据，对于每组数据，将排序后的n个整数输出，每个数后面都有一个空格。每组测试数据的结果占一行。

样例输入：

```
4
1 4 3 2
```

样例输出：

```
1 2 3 4
```

代码：

```
#include<stdio.h>
int main(){
    int n;
    int buff[100];
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < n; i++)
```

```
        {
            scanf("%d", &buff[i]);
        }
        for(int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (buff[j] > buff[j + 1]) {
                    int temp = buff[j];
                    buff[j] = buff[j + 1];
                    buff[j + 1] = temp;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            printf("%d ", buff[i]);
        }
    }
    return 0;
}
```

使用快速排序库函数：

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main(){
    int n;
    int buff[100];
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &buff[i]);
        }
        sort(buff, buff + n);
        for (int i = 0; i < n; i++) {
            printf("%d ", buff[i]);
        }
    }
    return 0;
}
```

如果需要重新规定编排规则

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

int cmp(int a, int b){//重新定义的编排规则
    return a > b;
}
```

```
int main(){
    int n;
    int buff[100];
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &buff[i]);
        }
        sort(buff, buff + n, cmp);
        for (int i = 0; i < n; i++) {
            printf("%d ", buff[i]);
        }
    }
    return 0;
}
```

C语言qsort

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

int compare(const void * a, const void * b){
    return *(int*) a - *(int*) b;
}

int main(){
    int n;
    int buff[100];
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &buff[i]);
        }
        qsort(buff, n, sizeof(int), compare);
        for (int i = 0; i < n; i++) {
            printf("%d ", buff[i]);
        }
        printf("\n");
    }
    return 0;
}
```

第2题

题目描述：

有N个学生的数据，将学生数据按成绩高低排序，如果成绩相同则按姓名字符的字母序排序，如果姓名的字母序也相同则按照学生的年龄排序，并输出N个学生排序后的信息。

输入：

测试数据有多组，每组输入第一行有一个整数N（N<=1000），接下来的N行包括N 个学生的数据。每个学生的数据包括姓名（长度不超过100 的字符串） 、年龄（整形数）、成绩（小于等于100 的正数）。

输出：

将学生信息按成绩进行排序，成绩相同的则按姓名的字母序进行排序。然后输出学生信息，按照如下格式：姓名年龄成绩

样例输入：

```
3
bcd 19 97
bed 20 97
abc 20 99
```

样例输出：

```
abc 20 99
bcd 19 97
bed 20 97
```

代码：

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;
struct Student
{
    char name[100];
    int age;
    int score;
} students[1000];

bool cmp(Student s1, Student s2)
{
```

```
    if (s1.score != s2.score)
        return s1.score > s2.score;
    else
    {
        int temp = strcmp(s1.name, s2.name);
        if (temp != 0)
            return temp < 0;
        else
        {
            return s1.age <= s2.age;
        }
    }
}

int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < n; i++)
        {
            scanf("%s", students[i].name);
            scanf("%d", &students[i].age);
            scanf("%d", &students[i].score);
        }
        sort(students, students + n, cmp);
        for (int i = 0; i < n; i++)
        {
            printf("%s %d %d\n", students[i].name,
students[i].age, students[i].score);
        }
    }
}
```

日期类问题

第6题

题目描述：

> 有两个日期，求两个日期之间的天数，如果两个日期是连续的我们规定他们之间的天数为两天

输入：

> 有多组数据，每组数据有两行，分别表示两个日期，形式为YYYYMMDD

输出：

```
每组数据输出一行，即日期差值
```

样例输入：

```
20110412
20110422
```

样例输出：

```
11
```

来源：

```
2009年上海交通大学计算机研究生机试题
```

代码：

```
#include<stdio.h>

int isLeapYear(int year) {
    if ((year % 4 == 0 && year % 100 != 0) || year % 400
== 0) return 0;
    else return 1;
}

int daysOfMonth[13][2] = {0, 0,
                          31, 31,
                          29, 28,
                          31, 31,
                          30, 30,
                          31, 31,
                          30, 30,
                          31, 31,
                          31, 31,
                          30, 30,
                          31, 31,
                          30, 30,
                          31, 31};
typedef struct Date {
    int year;
    int month;
```

```
    int day;
} Date;

void nextDay(Date *date) {
    date->day++;
    if (date->day > daysOfMonth[date->month]
[isLeapYear(date->year)]) {
        date->day = 1;
        date->month++;
        if (date->month > 12) {
            date->month = 1;
            date->year++;
        }
    }
}

int Abs(int a, int b) {
    return a > b ? a - b : b - a;
}
int daysTill0[5001][12][31] = {0};
int main() {
    Date date;
    date.year = 0;
    date.month = 1;
    date.day = 1;
    int count = 1;
    while (date.year != 5000) {
        daysTill0[date.year][date.month][date.day] =
count;
        nextDay(&date);
        count++;
    }
    int year1, year2, month1, month2, day1, day2;
    while (scanf("%4d%2d%2d", &year1, &month1, &day1) !=
EOF) {
        scanf("%4d%2d%2d", &year2, &month2, &day2);
        printf("%d\n", Abs(daysTill0[year1][month1][day1],
daysTill0[year2][month2][day2]) + 1);
    }
    return 0;
}
```

第7题

题目描述：

```
We now use the Gregorian style of dating in Russia. The leap years are years with number divisible by 4 but not divisible by 100, or divisible by 400.For example, years 2004, 2180 and 2400 are leap. Years 2004, 2181 and 2300 are not
leap.Your task is to write a program which will compute the day of week corresponding to a given date in the nearest past or in the future using today 's agreement about dating.
```

输入：

```
There is one single line contains the day number d, month name M and year number y(1000 ≤y≤3000). The month name is the corresponding English name starting from the capital letter.
```

输出：

```
Output a single line with the English name of the day of week corresponding to the date, starting from the capital letter. All other letters must be in lower case.
```

样例输入：

```
9 October 2001
14 October 2001
```

样例输出：

```
Tuesday
Sunday
```

来源：

```
2008 年上海交通大学计算机研究生机试真题
```

代码：

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

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

int daysOfMonth[13][2] = {0, 0,
                          31, 31,
                          29, 28,
                          31, 31,
                          30, 30,
                          31, 31,
                          30, 30,
                          31, 31,
                          31, 31,
                          30, 30,
                          31, 31,
                          30, 30,
                          31, 31};
typedef struct Date {
    int year;
    int month;
    int day;
} Date;

void nextDay(Date *date) {
    date->day++;
    if (date->day > daysOfMonth[date->month]
[isLeapYear(date->year)]) {
        date->day = 1;
        date->month++;
        if (date->month > 12) {
            date->month = 1;
            date->year++;
        }
    }
}
int Abs(int a, int b) {
    return a > b ? a - b : b - a;
}
char weekName[7][10] = {
"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday
","Saturday"};
char monthName[12][10] =
{"January","February","March","April","May","June","July",
"August","September","October","November","December"};
int daysTill0[3001][12][31] = {0};
int main() {
    Date date;
    date.year = 0;
    date.month = 1;
    date.day = 1;
```

```
    int count = 1;
    while (date.year != 3001) {
        daysTill0[date.year][date.month][date.day] =
count;
        nextDay(&date);
        count++;
    }
    int day,year;
    char month_name[10];
    while (scanf("%d%s%d", &day, month_name, &year) !=
EOF) {
        int month = -1;
        for (int i = 0; i < 12; i++) {
            if (strcmp(monthName[i],month_name) == 0) {
                month = i;
                break;
            }
        }
        if (month == -1) {
            printf("Month input error\n");
        } else
        {
            int days = daysTill0[year][month + 1][day] -
daysTill0[2019][6][9];
            printf("%s\n", weekName[(days % 7 + 7) % 7]);
        }
    }
    return 0;
}
```

Hash的应用

第10题

题目描述：

```
读入N 名学生的成绩，将获得某一给定分数的学生人数输出。
```

输入：

```
测试输入包含若干测试用例，每个测试用例的格式为
第1 行：N
第2 行：N名学生的成绩，相邻两数字用一个空格间隔。
第3 行：给定分数
当读到N=0时输入结束。其中N不超过1000，成绩分数为（包含）0到100
之间的一个整数。
```

输出：

```
对每个测试用例，将获得给定分数的学生人数输出。
```

样例输入：

```
3
80 60 90
60
2
85 66
0
5
60 75 90 55 75
75
0
```

样例输出：

```
1
0
2
```

来源：

```
2006年浙江大学计算机及软件工程研究生机试真题
```

代码：

```
#include<stdio.h>
int main(){
    int N;
    while(scanf("%d", &N) != EOF) {
        int hash[101] = {0};
        for (int i = 0; i < N; i++) {
            int score;
            scanf("%d", &score);
            hash[score]++;
        }
        int x;
        scanf("%d", &x);
        printf("%d\n",hash[x]);
    }
    return 0;
}
```

第11题

题目描述：

> 给你n个整数，请按从大到小的顺序输出其中前m大的数。

输入：

> 每组测试数据有两行，第一行有两个数n,m(0<n,m<1000000),第二行包含n个各不相同，且都处于区间[-500000,500000]的整数。

输出：

> 对每组测试数据按从大到小的顺序输出前m大的数。

样例输入：

```
5 3
3 -35 92 213 -644
```

样例输出：

```
213 92 3
```

代码：

```c
#include <stdio.h>
#define OFFSET 500000
int Hash[1000001];
int n, m;
int main()
{

    while (scanf("%d%d", &n, &m) != EOF)
    {
        //初始化Hash
        for (int i = 0; i < 1000000; i++)
        {
            Hash[i] = 0;
        }
        for (int i = 0; i < n; i++)
        {
            int x;
            scanf("%d", &x);
            Hash[x + OFFSET] = 1;
```

```
        }
        for (int i = 500000; i >= -500000; i--)
        {
            if (Hash[i + OFFSET] == 1)
            {
                printf("%d", i);
                m--;
                if (m != 0)
                {
                    printf(" ");
                } else
                {
                    printf("\n");
                    break;
                }


            }

        }
    }
    return 0;
}
```

排版题

第14题

题目描述：

```
输入一个高度h，输出一个高为h，上底边为h的梯形。
```

输入：

```
一个整数h(1<=h<=1000)。
```

输出：

```
h所对应的梯形。
```

样例输入：

```
4
```

样例输出：

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

来源：

```
2001年清华大学计算机研究生机试真题(第II套)
```

代码：

```
#include<stdio.h>
int main(){
    int h;
    while (scanf("%d", &h) != EOF)
    {
        int maxLen = h + (h - 1) * 2;
        for (int i = 0; i < h; i++)
        {
            int i_len = h + i * 2;
            for (int j = 0; j < maxLen - i_len; j++)
            {
                printf(" ");
            }
            for (int j = 0; j < i_len; j++)
            {
                printf("*");
            }
            printf("\n");
        }
    }
    return 0;
}
```

第15题

题目描述：

```
把一个个大小差一圈的筐叠上去，使得从上往下看时，边筐花色交错。这个工作现在要让计算机来完成，得看你的了。
```

输入：

```
输入是一个个的三元组，分别是外筐尺寸n（n为满足0<n<80 的奇整数），中心花色字符，外筐花色字符，后二者都为ASCII 可见字符；
```

输出：

输出叠在一起的筐图案，中心花色与外筐花色字符从内层起交错相叠，多筐相叠时，最外筐的角总是被打磨掉。叠筐与叠筐之间应有一行间隔。

样例输入：

```
11 B A
5 @ W
```

样例输出：

```
 AAAAAAAAA
ABBBBBBBBBA
ABAAAAAAABA
ABABBBBBABA
ABABAAABABA
ABABABABABA
ABABAAABABA
ABABBBBBABA
ABAAAAAAABA
ABBBBBBBBBA
 AAAAAAAAA
 @@@
@WWW@
@W@W@
@WWW@
 @@@
```

来源：

```

```

代码：

```
#include <stdio.h>
int main () {
int outPutBuf[82][82];
char a , b;
int n;
int firstCase = 1;
while (scanf ("%d %c %c" ,& n,& a,& b) == 3) {
if (firstCase == 1) {
firstCase = 0;
}
else printf ("\n" );
for (int i = 1,j = 1;i <= n;i += 2, j ++) {
int x = n / 2 + 1 , y = x;
x -= j - 1; y -= j - 1;
```

```
char c = j % 2 == 1 ? a : b;
for (int k = 1;k <= i;k ++) {
outPutBuf[x + k - 1][y] = c;
outPutBuf[x][y + k - 1] = c;
outPutBuf[x + i - 1][y + k - 1] = c;
outPutBuf[x + k - 1][y + i - 1] = c;
}
}
if (n != 1) {
outPutBuf [1][1] = ' ';
outPutBuf [n][1] = ' ' ;
outPutBuf [1][n] = ' ' ;
outPutBuf [n][n] = ' ' ;
}
for (int i = 1; i <= n;i ++) {
for (int j = 1;j <= n; j ++) {
printf ("%c" ,outPutBuf[i][j]);
}
printf ("\n" );
}
}
return 0;
}
```

查找

第17题

题目描述：

> 输入一个数n，然后输入n 个数值各不相同，再输入一个值x，输出这个值在这个数组中的下标（从0开始，若不在数组中则输出-1）。

输入：

> 测试数据有多组，输入n(1<=n<=200)，接着输入n 个数，然后输入x。

输出：

> 对于每组输入,请输出结果。

样例输入：

```
2
1 3
0
```

样例输出：

```
-1
```

来源：

```
2010年哈尔滨工业大学计算机研究生机试真题
```

代码：

```
#include <stdio.h>
int main()
{
    int buf[200];
    int n;
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &buf[i]);
        }
        int x, ans = -1;
        scanf("%d", &x);
        for (int i = 0; i < n; i++)
        {
            if (x == buf[i])
            {
                ans = i;
                break;
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

第18题

题目描述：

```
输入N 个学生的信息，然后进行查询。
```

输入：

输入的第一行为N，即学生的个数(N<=1000)
接下来的N行包括N个学生的信息，信息格式如下：
01 李江男21
02 刘唐男23
03 张军男19
04 王娜女19
然后输入一个M(M<=10000)，接下来会有M 行，代表M 次查询，每行输入一个学号，格式如下：
02
03
01
04

输出：

输出M 行，每行包括一个对应于查询的学生的信息。
如果没有对应的学生信息，则输出-No Answer!

样例输入：

```
4
01 李江 男 21
02 刘唐 男 23
03 张军 男 19
04 王娜 女 19
5
02
03
01
04
03
```

样例输出：

```
02 刘唐 男 23
03 张军 男 19
01 李江 男 21
04 王娜 女 19
03 张军 男 19
```

来源：

2003 年清华大学计算机研究生机试真题

代码：

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

```c
#include <string.h>
struct Student
{
    char id[100];
    char name[100];
    char sex[5];
    int age;
} students[1000];

int compare(const void *a, const void *b)
{
    return strcmp((*(struct Student *)a).id, (*(struct
Student *)b).id);
}

int main()
{
    int N;
    while (scanf("%d", &N) != EOF)
    {

        for (int i = 0; i < N; i++)
        {
            scanf("%s%s%s%d", students[i].id,
students[i].name, students[i].sex, &students[i].age);
        }
        qsort(students, N, sizeof(struct Student),
compare);
        int M;
        scanf("%d", &M);
        for (int i = 0; i < M; i++)
        {
            int ans = -1;
            char target[100];
            scanf("%s", target);
            int top = N - 1;
            int base = 0;
            int mid = (top + base) / 2;
            while (top >= base)
            {
                int cmp = strcmp(students[mid].id,
target);
                if (cmp == 0)
                {
                    ans = mid;
                    break;
                }
                else if (cmp > 0)
                {
                    top = mid - 1;
                    mid = (top + base) / 2;
                }
```

```
                else
                {
                    base = mid + 1;
                    mid = (top + base) / 2;
                }
            }
            if (ans != -1)
            {
                printf("%s %s %s %d\n", students[ans].id,
students[ans].name, students[ans].sex, students[ans].age);
            }
            else
            {
                printf("No answer\n");
            }
        }
    }
    return 0;
}
```

贪心算法

第21题

题目描述：

```
FatMouse prepared M pounds of cat food, ready to trade
with the cats guarding the warehouse containing his
favorite food, JavaBean.
The warehouse has N rooms. The i-th room contains J[i]
pounds of JavaBeans and requires F[i] pounds of cat food.
FatMouse does not have to trade for all the JavaBeans in
the room, instead, he may get J[i]* a% pounds of JavaBeans
if he pays F[i]* a% pounds of cat food. Here a is a real
number. Now he is assigning this homework to you: tell him
the maximum amount of JavaBeans he can obtain.
```

输入：

```
The input consists of multiple test cases. Each test case
begins with a line containing two non-negative integers M
and N. Then N lines follow, each contains two non-negative
integers J[i] and F[i] respectively. The last test case is
followed by two -1's. All integers are not greater than
1000.
```

输出：

```
For each test case, print in a single line a real number
accurate up to 3 decimal places, which is the maximum
amount of JavaBeans that FatMouse can obtain.
```

样例输入：

```
5 3
7 2
4 3
5 2
20 3
25 18
24 15
15 10
-1 -1
```

样例输出：

```
13.333
31.500
```

来源：

```
```

代码：

```
#include <stdio.h>

struct Goods
{
    double weight;
    double price;
    double performance;
}goods[1000];
int compare(const void *a, const void *b)
{
    return (*(struct Goods *)b).performance - (*(struct
Goods *)a).performance;
}

int main()
{
    double M;
    int N;
    while (scanf("%lf%d", &M, &N) != EOF)
    {
        if (M == -1 && N == -1) break;
        for (int i = 0; i < N; i++)
```

```
        {
            double weight, price, performance;
            scanf("%lf%lf", &weight, &price);
            performance = weight / price;
            goods[i].weight = weight;
            goods[i].price = price;
            goods[i].performance = performance;
        }
        qsort(goods, N, sizeof(struct Goods), compare);
        int count = 0;
        double ans = 0;
        while (M > 0)
        {
            if (count > N - 1) break;
            if (goods[count].price <= M)
            {
                M -= goods[count].price;
                ans += goods[count].weight;
            } else
            {
                ans += M * goods[count].performance;
                M = 0;
            }
            count++;
        }
        printf("%.3lf\n", ans);
    }
}
```

第22题

题目描述：

> 假设你已经知道了所有你喜欢看的电视节目的转播时间表，你会合理安排吗？（目标是能看尽量多的完整节目）

输入：

> 输入数据包含多个测试实例，每个测试实例的第一行只有一个整数n(n<=100)，表示你喜欢看的节目的总数，然后是n行数据，每行包括两个数据Ti_s,Ti_e (1<=i<=n)，分别表示第i 个节目的开始和结束时间，为了简化问题，每个时间都用一个正整数表示。n=0 表示输入结束，不做处理。

输出：

> 对于每个测试实例，输出能完整看到的电视节目的个数，每个测试实例的输出占一行。

样例输入：

```
12
1 3
3 4
0 7
3 8
15 19
15 20
10 15
8 18
6 12
5 10
4 14
2 9
0
```

样例输出：

```
5
```

来源：

```

```

代码：

```
#include<stdio.h>
struct Shows{
    int startTime;
    int endTime;
}shows[100];

int compare(const void * a, const void *b)
{
    return (*(struct Shows *)a).endTime - (*(struct Shows
*)b).endTime;
}

int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        if (n == 0) break;
        for (int i = 0; i < n; i++)
        {
            int startTime, endTime;
            scanf("%d%d", &startTime, &endTime);
            shows[i].startTime = startTime;
            shows[i].endTime = endTime;
        }
```

```
        qsort(shows, n, sizeof(struct Shows), compare);
        int ans = 0;
        int timeNow = 0;
        for (int i = 0; i < n; i++)
        {
            if (shows[i].startTime >= timeNow)
            {
                timeNow = shows[i].endTime;
                ans++;
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

三、数据结构

栈的应用

例3.1（括号匹配）

题目描述：

> 在某个字符串（长度不超过100）中有左括号、右括号和大小写字母；规定（与常见的算数式子一样）任何一个左括号都从内到外与在它右边且距离最近的右括号匹配。写一个程序，找到无法匹配的左括号和右括号，输出原来字符串，并在下一行标出不能匹配的括号。不能匹配的左括号用"$" 标注,不能匹配的右括号用"?" 标注.

输入：

> 输入包括多组数据，每组数据一行，包含一个字符串，只包含左右括号和大小写字母，字符串长度不超过100。

输出：

> 对每组输出数据，输出两行，第一行包含原始输入字符，第二行由"$","?"和空格组成， "$" 和"?" 表示与之对应的左括号和右括号不能匹配。

样例输入：

```
)(rttyy())sss)(
```

样例输出：

```
)(rttyy())sss)(
?           ?$
```

来源：

```
```

代码：

```
#include<stdio.h>
#include<stack>
#include<string.h>
using namespace std;
char input[100];
char output[100];
stack<int> S;
int main()
{
    while (scanf("%s", input) ! = EOF)
    {
        while (!S.empty()) S.pop();
        int len = strlen(input);
        for (int i = 0; i < len; i++)
        {
            output[i] = ' ';
            if (input[i] == '(') S.push(i);
            else if (input[i] == ')')
            {
                if (S.empty()) output[i] = '?';
                else S.pop();
            }
        }
        while (!S.empty())
        {
            int temp = S.top();
            S.pop();
            output[temp] = '$';
        }
        output[len] = 0;
        printf("%s\n%s\n", input, output);
    }
    return 0;
}
```

例3.2（表达式求值）

题目描述：

```
读入一个只包含+, -, *, / 的非负整数计算表达式，计算该表达式的值。
```

输入：

> 测试输入包含若干测试用例，每个测试用例占一行，每行不超过200 个字符，整数和运算符之间用一个空格分隔。没有非法表达式。当一行中只有0 时输入结束，相应的结果不要输出。

输出：

> 对每个测试用例输出1 行，即该表达式的值，精确到小数点后2 位。

样例输入：

```
1 + 2
4 + 2 * 5 - 7 / 11
0
```

样例输出：

```
3.00
13.36
```

来源：

> 2006 年浙江大学计算机及软件工程研究生机试真题

代码：

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

stack<int> opStack;
stack<double> numStack;

char s[200];

int loc;

int matrix[4][4] = {
    0, 0, 0, 0,
    0, 0, 0, 0,
    1, 1, 0, 0,
    1, 1, 0, 0};

void getNext(int &isNum, int &data)
{
    if (s[loc] >= '0' && s[loc] <= '9')
```

```
    {
        isNum = 1;
        data = 0;
        while (s[loc] != ' ' && s[loc] != 0)
        {
            data = data * 10 + (s[loc] - '0');
            loc++;
        }
        loc++;
    }
    else
    {
        isNum = 0;
        if (s[loc] == '+')
        {
            data = 0;
            loc = loc + 2;
        }
        else if (s[loc] == '-')
        {
            data = 1;
            loc = loc + 2;
        }
        else if (s[loc] == '*')
        {
            data = 2;
            loc = loc + 2;
        }
        else
        {
            data = 3;
            loc = loc + 2;
        }
    }
}

int main()
{
    while (gets(s))
    {
        if (s[0] == '0' && s[1] == 0) break;
        loc = 0;

        while (!numStack.empty())
            numStack.pop();
        while (!opStack.empty())
            opStack.pop();

        while (s[loc] != 0)
        {
            int isNum, data;
            getNext(isNum, data);
```

```
            if (isNum == 1)
                numStack.push((double)data);
            else
            {
                if (opStack.empty() || matrix[data]
[opStack.top()] == 1)
                    opStack.push(data);
                else
                {
                    while (!opStack.empty() &&
matrix[data][opStack.top()] == 0)
                    {
                        double num1 = numStack.top();
                        numStack.pop();
                        double num2 = numStack.top();
                        numStack.pop();
                        int op = opStack.top();
                        opStack.pop();
                        double result;
                        if (op == 0)
                            result = num1 + num2;
                        else if (op == 1)
                            result = num2 - num1;
                        else if (op == 2)
                            result = num1 * num2;
                        else
                            result = num2 / num1;
                        numStack.push(result);
                    }
                    opStack.push(data);
                }
            }
        }
        while (!opStack.empty())
        {
            double num1 = numStack.top();
            numStack.pop();
            double num2 = numStack.top();
            numStack.pop();
            int op = opStack.top();
            opStack.pop();
            double result;
            if (op == 0)
                result = num1 + num2;
            else if (op == 1)
                result = num2 - num1;
            else if (op == 2)
                result = num1 * num2;
            else
            {
                result = num2 / num1;
            }
```

```
            numStack.push(result);
        }
        printf("%.2lf\n", numStack.top());
        memset(s, '\0', 100 * sizeof(char));
    }
    return 0;
}
```

哈夫曼树

例3.3

题目描述：

> 哈夫曼树，第一行输入一个数n，表示叶结点的个数。需要用这些叶结点生成哈夫曼树，根据哈夫曼树的概念，这些结点有权值，即weight，题目需要输出所有结点的值与权值的乘积之和。

输入：

> 输入有多组数据。每组第一行输入一个数n，接着输入n 个叶节点（叶节点权值不超过100，2<=n<=1000）。

输出：

> 输出权值。

样例输入：

```
5
1 2 2 5 9
```

样例输出：

```
37
```

来源：

> 2010年北京邮电大学计算机研究生机试真题

代码：

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

```
using namespace std;
priority_queue<int, vector<int>, greater<int>> Q;
int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        while (Q.empty() == false)
            Q.pop();
        for (int i = 1; i <= n; i++)
        {
            int x;
            scanf("%d", &x);
            Q.push(x);
        }
        int ans = 0;
        while (Q.size() > 1)
        {
            int a = Q.top();
            Q.pop();
            int b = Q.top();
            Q.pop();
            ans += a + b;
            Q.push(a + b);
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

二叉树

例3.4

题目描述：

> 二叉树的前序、中序、后序遍历的定义：
> 前序遍历：对任一子树，先访问跟，然后遍历其左子树，最后遍历其右子树；
> 中序遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树；
> 后序遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访问根。
> 给定一棵二叉树的前序遍历和中序遍历，求其后序遍历（提示：给定前序遍历与中序遍历能够唯一确定后序遍历）。

输入：

> 两个字符串，其长度n 均小于等于26。第一行为前序遍历，第二行为中序遍历。二叉树中的结点名称以大写字母表示：A，B，C....最多26 个结点。

输出：

输入样例可能有多组，对于每组测试样例，输出一行，为后序遍历的字符串。

样例输入：

```
ABC
BAC
FDXEAG
XDEFAG
```

样例输出：

```
BCA
XEDGAF
```

来源：

2006年清华大学计算机研究生机试真题

代码：

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

struct Node
{
    struct Node *lchild;
    struct Node *rchild;
    char c;
} Tree[50];

int flag = 0;

struct Node *create()
{
    return &Tree[flag++];
}

char str1[26];
char str2[26];

void postOrder(struct Node *Tree)
{
    if (Tree->lchild != NULL)
    {
        postOrder(Tree->lchild);
    }
    if (Tree->rchild != NULL)
    {
        postOrder(Tree->rchild);
```

```
    }
    printf("%c", Tree->c);
}

struct Node *build(int s1, int d1, int s2, int d2)
{
    struct Node * root = create();
    root->c = str1[s1];
    int mid = -1;
    for (int i = s2; i <= d2; i++)
    {
        if (str2[i] == str1[s1])
        {
            mid = i;
            break;
        }
    }
    if (mid != s2)
    {
        root ->lchild = build(s1 + 1, mid + s1 - s2, s2,
mid - 1);
    }
    if (mid != d2)
    {
        root ->rchild = build(d1 - d2 + mid + 1, d1, mid +
1, d2);
    }
    return root;
}

int main()
{
    while (scanf("%s%s", str1, str2) != EOF)
    {
        flag = 0;
        int len1 = strlen(str1);
        int len2 = strlen(str2);
        struct Node *root = build(0, len1 - 1, 0, len2 -
1);
        postOrder(root);
        printf("\n");
    }
    return 0;
}
```

二叉排序树

例3.5

题目描述：

> 输入一系列整数，建立二叉排序数，并进行前序，中序，后序遍历。

输入：

> 输入第一行包括一个整数n(1<=n<=100)。接下来的一行包括n 个整数。

输出：

> 可能有多组测试数据，对于每组数据，将题目所给数据建立一个二叉排序树，并对二叉排序树进行前序、中序和后序遍历。每种遍历结果输出一行。每行最后一个数据之后有一个空格。

样例输入：

```
5
1 6 5 9 8
```

样例输出：

```
16598
15689
58961
```

来源：

> 2005年华中科技大学计算机保研机试真题

代码：

```
#include <stdio.h>

struct Node
{
    int data;
    struct Node *lchild;
    struct Node *rchild;
} Tree[100];

int flag = 0;
struct Node *create()
{
    return &Tree[flag++];
}
```

```
struct Node* insert(struct Node *root, int data)
{
    if (root == NULL)
    {
        struct Node *newNode = create();
        newNode->data = data;
        root = newNode;
    }
    else
    {
        if (data < root->data)
            root->lchild = insert(root->lchild, data);
        else if (data > root->data)
        {
            root->rchild = insert(root->rchild, data);
        }
    }
    return root;
}

void preOrder(struct Node * root)
{
    if (root == NULL) return;
    printf("%d", root->data);
    if (root->lchild != NULL) preOrder(root->lchild);
    if (root->rchild != NULL) preOrder(root->rchild);
}

void inOrder(struct Node * root)
{
    if (root->lchild != NULL) inOrder(root->lchild);
    printf("%d", root->data);
    if (root->rchild != NULL) inOrder(root->rchild);
}

void postOrder(struct Node * root)
{
    if (root->lchild != NULL) postOrder(root->lchild);
    if (root->rchild != NULL) postOrder(root->rchild);
    printf("%d", root->data);
}

int main()
{
    flag = 0;
    int n;
    while (scanf("%d", &n))
    {
        struct Node *root = NULL;
        for (int i = 0; i < n; i++)
        {
            int x;
```

```
            scanf("%d", &x);
            root = insert(root, x);
        }
        preOrder(root);
        printf("\n");
        inOrder(root);
        printf("\n");
        postOrder(root);
        printf("\n");
    }
}
```

例3.6

题目描述：

```
判断两序列是否为同一二叉搜索树序列
```

输入：

```
开始一个数n，(1<=n<=20) 表示有n 个需要判断， n= 0 的时候输入结束。接下去一行是一个序列，序列长度小于10，包含(0~9)的数字，没有重复数字，根据这个序列可以构造出一颗二叉搜索树。接下去的n 行有n 个序列，每个序列格式跟第一个序列一样，请判断这两个序列是否能组成同一颗二叉搜索树。
```

输出：

```
如果序列相同则输出YES，否则输出NO
```

样例输入：

```
2
567432
543267
576342
0
```

样例输出：

```
YES
NO
```

来源：

```
2010年浙江大学计算机及软件工程研究生机试真题
```

代码：

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

struct Node
{
    char data;
    struct Node * lchild;
    struct Node * rchild;
}Tree[200];

int flag = 0;

char * str;
int * size;

char str1[25];
int size1;
char str2[25];
int size2;

struct Node * create()
{
    return &Tree[flag++];
}

struct Node * insert(struct Node * root, char data)
{
    if (root == NULL)
    {
        struct Node * newNode = create();
        newNode->data = data;
        root = newNode;
    } else
    {
        if (root->data > data) root->lchild = insert(root-
>lchild, data);
        if (root->data < data) root->rchild = insert(root-
>rchild, data);
    }
    return root;
}

void preOrder(struct Node * root)
{
    str[(*size)++] = root->data;
    if (root->lchild != NULL) preOrder(root->lchild);
    if (root->rchild != NULL) preOrder(root->rchild);
}

void inOrder(struct Node * root)
```

```
{
    if (root->lchild != NULL) inOrder(root->lchild);
    str[(*size)++] = root->data;
    if (root->rchild != NULL) inOrder(root->rchild);
}

int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        if (n == 0) break;
        flag = 0;
        char temp[10];
        scanf("%s", temp);
        int str_len = strlen(temp);
        struct Node * root = NULL;
        for (int i = 0; i < str_len; i++)
        {
            root = insert(root, temp[i]);
        }
        size1 = 0;
        size = &size1;
        str = str1;
        preOrder(root);
        inOrder(root);
        str1[size1] = 0;

        for (int i = 0; i < n; i++)
        {
            scanf("%s", temp);
            str_len = strlen(temp);
            struct Node * root1 = NULL;
            for (int j = 0; j < str_len; j++)
            {
                root1 = insert(root1, temp[j]);
            }
            size2 = 0;
            str = str2;
            size = &size2;
            preOrder(root1);
            inOrder(root1);
            str2[size2] = 0;
            int ans = strcmp(str1, str2) == 0 ? 1 : 0;
            if (ans == 1) printf("YES\n");
            else printf("NO\n");
        }
    }
    return 0;
}
```

四、数学问题

%运算符

数位拆解

第39题

题目描述：

> 写个算法，对2 个小于1000000000 的输入，求结果。
> 特殊乘法举例： 123 * 45 = 1*4 +1*5 +2*4 +2*5 +3*4+3*5

输入：

> 两个小于1000000000的数

输出：

> 输入可能有多组数据，对于每一组数据，输出Input 中的两个数按照题目要求的方法进行运算后得到的结果。

样例输入：

```
123 45
```

样例输出：

```
54
```

来源：

> 2010年清华大学计算机研究生机试真题

代码：

```
#include <stdio.h>
int main()
{
    int a, b;
```

```
    while (scanf("%d%d", &a, &b) != EOF)
    {
        int a_size = 0, b_size = 0;
        int a_divide[10];
        int b_divide[10];
        while (a != 0)
        {
            a_divide[a_size++] = a % 10;
            a = a / 10;
        }
        while (b != 0)
        {
            b_divide[b_size++] = b % 10;
            b = b / 10;
        }
        int ans = 0;
        for (int i = 0; i < a_size; i++)
        {
            for (int j = 0; j < b_size; j++)
            {
                ans += a_divide[i] * b_divide[j];
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

进制转换

第43题

题目描述：

> 输入两个不超过整型定义的非负10 进制整数A 和B(<=231-1)，输出A+B的m (1 < m <10)进制数。

输入：

> 输入格式：测试输入包含若干测试用例。每个测试用例占一行，给出m 和A，B 的值。当m 为0 时输入结束。

输出：

> 输出格式：每个测试用例的输出占一行，输出A+B 的m 进制数。

样例输入：

```
8 1300 48
2 1 7
0
```

样例输出：

```
2504
1000
```

来源：

```
2008年浙江大学计算机及软件工程研究生机试真题
```

代码：

```
#include<stdio.h>
int main()
{
    int m;
    while(scanf("%d", &m))
    {
        if (m == 0) break;
        long a, b;
        scanf("%ld%ld", &a, &b);
        long sum = a + b;
        int result[100];
        int size = 0;
        while (sum != 0)
        {
            result[size++] = sum % m;
            sum = sum / m;
        }
        for (int i = size - 1; i >= 0; i--)
        {
            printf("%d",result[i]);
        }
        printf("\n");

    }
    return 0;
}
```

第44题

题目描述：

```
求任意两个不同进制非负整数的转换（ 2 进制～ 16 进制），所给整数在long所能表达的范围之内。不同进制的表示符号为（ 0，1，...，9，a，b，...，f）或者（ 0，1，...，9，A，B，...，F）。
```

输入：

> 输入只有一行，包含三个整数a，n，b。a 表示其后的n 是a 进制整数， b 表示欲将a 进制整数n 转换成b 进制整数。a，b 是十进制整数， 2 =< a，b <= 16。

输出：

> 可能有多组测试数据，对于每组数据，输出包含一行，该行有一个整数为转换后的b 进制数。输出时字母符号全部用大写表示，即（ 0，1，...，9，A，B，...，F）。

样例输入：

```
15 Aab3 7
```

样例输出：

```
210306
```

来源：

> 2008年北京大学图形实验室计算机研究生机试真题

代码：

```
#include<stdio.h>
#include<string.h>
int main()
{
    int a, b;
    char n[50];
    while (scanf("%d%s%d", &a, n, &b) != EOF)
    {
        int len = strlen(n);
        long temp = 1;
        long ans_mid = 0;
        for (int i = len - 1; i >= 0; i--)
        {
            int x;
            if (n[i] >= '0' && n[i] <= '9')
            {
                x = n[i] - '0';
            } else if (n[i] >= 'a' && n[i] <= 'z')
            {
                x = n[i] - 'a' + 10;
            } else
            {
```

```
                x = n[i] - 'A' + 10;
            }
            ans_mid += x * temp;
            temp *= a;
        }
        char ans[50];
        int count = 0;
        while (ans_mid != 0)
        {
            temp = ans_mid % b;
            if (temp > 9)
            {
                ans[count] = (char)('A' + temp - 10);
            }
            else
            {
                ans[count] = (char)('0' + temp);
            }
            count++;
            ans_mid /= b;
        }
        for(int i = count - 1; i >= 0; i--)
        {
            printf("%c",ans[i]);
        }
        printf("\n");
    }
    return 0;
}
```

最大公约数GCD

第47题

题目描述：

> 输入两个正整数，求其最大公约数。

输入：

> 测试数据有多组，每组输入两个正整数。

输出：

> 对于每组输入,请输出其最大公约数。

样例输入：

```
49 14
```

样例输出：

```
7
```

来源：

```
2011年哈尔滨工业大学计算机研究生机试真题
```

代码：

```
#include<stdio.h>
int gcd(int a, int b)
{
    if (b == 0) return a;
    else return gcd(b, a % b);
}
int main()
{
    int a, b;
    while (scanf("%d%d", &a, &b) != EOF)
    {
        printf("%d\n", gcd(a, b));
    }
    return 0;
}
```

最小公倍数LCM

第49题

题目描述：

```
给定两个正整数，计算这两个数的最小公倍数。
```

输入：

```
输入包含多组测试数据，每组只有一行，包括两个不大于1000 的正整数。
```

输出：

```
对于每个测试用例，给出这两个数的最小公倍数，每个实例输出一行。
```

样例输入：

```
10 14
```

样例输出：

```
70
```

来源：

```

```

代码：

```
#include <stdio.h>
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}
int main()
{
    int a, b;
    while (scanf("%d%d", &a, &b) != EOF)
    {
        int g =gcd(a, b);
        printf("%d\n",a * b / g);
    }
    return 0;
}
```

素数筛选

第50题

题目描述：

```
给定一个数n，要求判断其是否为素数（0,1，负数都是非素数）。
```

输入：

```
测试数据有多组，每组输入一个数n。
```

输出：

```
对于每组输入,若是素数则输出yes，否则输入no。
```

样例输入：

```
13
```

样例输出：

```
yes
```

来源：

```
2009年哈尔滨工业大学计算机研究生机试真题
```

代码：

```
#include<stdio.h>
#include<math.h>
int main()
{
    int n;
    while (scanf("%d",&n) != EOF)
    {
        int flag = 0;
        int t = (int)sqrt(n);
        for (int i = 2; i <= t; i++)
        {
            if(n % i == 0)
            {
                flag = 1;
                break;
            }
        }
        if (flag == 0) printf("yes\n");
        else printf("no\n");
    }

}
```

第51题

题目描述：

```
输入一个整数n(2<=n<=10000)，要求输出所有从1 到这个整数之间(不包括1和这个整数)个位为1 的素数，如果没有则输出-1。
```

输入：

```
输入有多组数据。每组一行，输入n。
```

输出：

> 输出所有从1 到这个整数之间(不包括1 和这个整数)个位为1 的素数(素数之间用空格隔开，最后一个素数后面没有空格)，如果没有则输出-1。

样例输入：

```
100
```

样例输出：

```
11 31 41 61 71
```

来源：

> 2008 年北京航空航天大学计算机研究生机试真题

代码：

```
#include<stdio.h>
int prime[10001];

void init()
{
    for (int i = 2; i < 10001; i++)
    {
        if (prime[i] == 0)
        {
            for (int j = i + i; j <= 10000; j += i)
            {
                prime[j] = 1;
            }
        }
    }
}

int main()
{
    init();
    int n;
    while (scanf("%d", &n) != EOF)
    {
        int primecount = 0;
        int first = 0;
        for (int i = 2; i <= n; i++)
        {
            if (prime[i] == 0 && i % 10 == 1)
            {
                if (first ==0)
```

```
                {
                    printf("%d", i);
                    first = 1;
                }
                else
                {
                    printf(" %d", i);
                }
            }
        }
        printf("\n");
    }
    return 0;
}
```

分解素因数

第54题

题目描述：

```
求正整数N(N>1) 的质因数的个数。
相同的质因数需要重复计算。如120=2*2*2*3*5，共有5个质因数。
```

输入：

```
可能有多组测试数据，每组测试数据的输入是一个正整数N，(1<N<10^9)。
```

输出：

```
对于每组数据，输出N 的质因数的个数。
```

样例输入：

```
120
```

样例输出：

```
5
```

来源：

```
2007年清华大学计算机研究生机试真题
```

代码：

```

```

```c
#include<stdio.h>
#define PRIME_LEN 100000

int prime[PRIME_LEN];

void init()
{
    for (int i = 2; i < PRIME_LEN; i++)
    {
        if (prime[i] == 0)
        {
            for (int j = i + i; j < PRIME_LEN; j = j + i)
            {
                prime[j] = 1;
            }
        }
    }
}
int main()
{
    init();
    int N;
    while (scanf("%d", &N))
    {
        int count_prime = 0;
        for (int i = 2; i < PRIME_LEN; i++)
        {
            if (prime[i] == 0 && N % i == 0)
            {
                while (N % i == 0)
                {
                    N /= i;
                    count_prime++;
                }
            }
            if (N == 1)
            {
                break;
            }
        }
        if (N != 1)
        {
            count_prime++;
        }
        printf("%d\n", count_prime);
    }
    return 0;
}
```

第55题

题目描述：

给定n，a 求最大的k，使n! 可以被a^k 整除但不能被a^(k+1)整除。

输入：

两个整数n(2<=n<=1000)，a(2<=a<=1000)

输出：

一个整数.

样例输入：

6 10

样例输出：

1

来源：

2011年上海交通大学计算机研究生机试真题

代码：

```
#include<stdio.h>
#include<string.h>
int mark[1010];
int prime[1010];
int primeSize;
void init()
{
    primeSize = 0;
    for (int i = 2; i <= 1000; i++)
    {
        if (mark[i])
            continue;
        mark[i] = 1;
        prime[primeSize++] = i;
        for (int j = i * i; j <= 1000; j += i)
        {
            mark[j] = 1;
        }
    }
}
int cnt[1010];
```

```
int cnt2[1010];
int main()
{
    int n, a;
    init();
    while (scanf("%d%d", &n, &a) == 2)
    {
        for (int i = 0; i < primeSize; i++)
            cnt[i] = cnt2[i] = 0;
        for (int i = 0; i < primeSize; i++)
        {
            int t = n;
            while (t)
            {
                cnt[i] += t / prime[i];
                t = t / prime[i];
            }
        }
        int ans = 123123123;
        for (int i = 0; i < primeSize; i++)
        {
            while (a % prime[i] == 0)
            {
                cnt2[i]++;
                a /= prime[i];
            }
            if (cnt2[i] == 0)
                continue;
            if (cnt[i] / cnt2[i] < ans)
                ans = cnt[i] / cnt2[i];
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

二分求幂

第57题

题目描述：

```
求A^B 的最后三位数表示的整数。说明：A^B的含义是A的B次方
```

输入：

```
输入数据包含多个测试实例，每个实例占一行，由两个正整数A 和B 组成
（1<=A,B<=10000），如果A=0， B=0，则表示输入数据的结束，不做处理。
```

输出：

```
对于每个测试实例，请输出A^B 的最后三位表示的整数，每个输出占一行。
```

样例输入：

```
2 3
12 6
6789 10000
0 0
```

样例输出：

```
8
984
1
```

来源：

```

```

代码：

```c
#include <stdio.h>
int main()
{
    int a, b;
    while (scanf("%d%d", &a, &b) != EOF)
    {
        if (a == 0 && b == 0)
            break;
        int ans = 1;
        while (b != 0)
        {
            if (b % 2 == 1)
            {
                ans *= a;
                ans %= 1000;
            }
            b /= 2;
            a *= a;
            a %= 1000;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

高精度整数

第60题

题目描述：

> 实现一个加法器，使其能够输出a+b的值。

输入：

> 输入包括两个数a和b，其中a和b的位数不超过1000位。

输出：

> 可能有多组测试数据，对于每组数据，输出a+b的值。

样例输入：

```
2 6
10000000000000000000 100000000000000000000000000000000
```

样例输出：

```
8
100000000000100000000000000000000
```

来源：

> 2010年华中科技大学计算机研究生机试真题

代码：

五、图论

并查集

例5.1

题目描述：

某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府–畅通工程的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

输入：

测试输入包含若干测试用例。每个测试用例的第1 行给出两个正整数，分别是城镇数目N (< 1000)和道路数目M；随后的M 行对应M 条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从1 到N 编号。当N 为0 时，输入结束，该用例不被处理。

输出：

对每个测试用例，在1 行里输出最少还需要建设的道路数目。

样例输入：

```
4 2
1 3
4 3
3 3
1 2
1 3
2 3
5 2
1 2
3 5
999 0
0
```

样例输出：

```
1
0
2
998
```

来源：

2005 年浙江大学计算机及软件工程研究生机试真题

代码：

```
#include<stdio.h>
#define MAX 1000

int Tree[MAX];
```

```
int findRoot(int i)
{
    if (Tree[i] == -1) return i;
    int root = findRoot(Tree[i]);
    Tree[i] = root;
    return root;
}

int main()
{
    int N, M;
    while (scanf("%d%d", &N, &M) != EOF)
    {
        for (int i = 1; i <= N; i++)
        {
            Tree[i] = -1;
        }
        for (int i = 0; i < M; i++)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            int rootA = findRoot(a);
            int rootB = findRoot(b);
            if (rootA != rootB) Tree[rootA] = rootB;
        }
        int ans = 0;
        for(int i = 1; i <= N; i++)
        {
            if(Tree[i] == -1) ans++;
        }
        printf("%d\n", ans - 1);
    }
    return 0;
}
```

例5.2

题目描述：

```
Mr Wang wants some boys to help him with a project.
Because the project is rather complex, the more boys come,
the better it will be. Of course there are certain
requirements.Mr Wang selected a room big enough to hold
the boys. The boy who are not been chosen has to leave the
room immediately. There are 10000000 boys in the room
numbered from 1 to 10000000 at the very beginning. After
Mr Wang's selection any two of them who are still in this
room should be friends (direct or indirect), or there is
only one boy left. Given all the direct friend-pairs, you
should decide the best way.
```

输入：

```
The first line of the input contains an integer n (0 ≤ n ≤ 100000)the number of direct friend-pairs. The following n lines each contains a pair of numbers A and B separated by a single space that suggests A and B are direct friends.(A ≠ B, 1 ≤ 10000000)
```

输出：

```
The output in one line contains exactly one integer equals to the maximum number of boys Mr Wang may keep.
```

样例输入：

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

样例输出：

```
4
2
```

来源：

```

```

代码：

```
#include<stdio.h>

#define N 10000001

int Tree[N];
int sum[N];

int findRoot(int a)
{
    if (Tree[a] == -1) return a;
    int b = findRoot(Tree[a]);
```

```
    Tree[a] = b;
    return b;
}

int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        for (int i = 0; i < N; i++)
        {
            Tree[i] = -1;
            sum[i] = 1;
        }

        for(int i = 0; i < n; i++)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            int rootA = findRoot(a);
            int rootB = findRoot(b);
            Tree[rootA] = rootB;
            sum[rootB] += sum[rootA];
        }
        int ans = 0;
        for (int i = 1; i < N; i++)
        {
            if (sum[i] > ans) ans = sum[i];
        }
        printf("%d\n", ans);
    }
}
```

最小生成树（Kruskal）

例5.3

题目描述：

> 某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府-畅通工程的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

输入：

> 测试输入包含若干测试用例。每个测试用例的第1 行给出村庄数目N (<100)；随后的N(N-1)/2 行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1 到N 编号。当N为0 时，输入结束，该用例不被处理。

输出：

```
对每个测试用例，在1 行里输出最小的公路总长度。
```

样例输入：

```
3
1 2 1
1 3 2
2 3 4
4
1 2 1
1 3 4
1 4 1
2 3 3
2 4 2
3 4 5
0
```

样例输出：

```
3
5
```

来源：

```
2006年浙江大学计算机及软件工程研究生机试真题
```

代码：

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

int Tree[100];

int findRoot(int a)
{
    if (Tree[a] == -1) return a;
    return findRoot(Tree[a]);
}

struct E
{
    int a, b;
    int cost;
} edge[5000];

int compare(const void *a, const void *b)
{
```

```
    return (*(struct E *)a).cost - (*(struct E *)b).cost;
}

int main()
{
    int N;
    while (scanf("%d", &N))
    {
        if (N == 0) break;
        for (int i = 0; i < N * (N - 1) / 2; i++)
        {
            scanf("%d%d%d", &edge[i].a, &edge[i].b,
&edge[i].cost);
        }
        for (int i = 1; i <= N; i++)
        {
            Tree[i] = -1;
        }

        qsort(edge, N * (N - 1) / 2, sizeof(struct E),
compare);

        int ans = 0;
        for (int i = 0; i < N * (N - 1) / 2; i++)
        {
            int rootA = findRoot(edge[i].a);
            int rootB = findRoot(edge[i].b);
            if (rootA != rootB)
            {
                Tree[rootA] = rootB;
                ans += edge[i].cost;
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

例5.4

题目描述：

```
In an episode of the Dick Van Dyke show, little Richie
connects the freckles on his Dad's back to form a picture
of the Liberty Bell. Alas, one of the freckles turns out
to be a scar, so his Ripley's engagement falls
through.Consider Dick's back to be a plane with freckles
at various (x,y) locations. Your job is to tell Richie how
to connect the dots so as to minimize the amount of ink
used. Richie connects the dots by drawing straight lines
between pairs, possibly lifting the pen between lines.
When Richie is done there must be a sequence of connected
lines from any freckle to any other freckle.
```

输入：

```
The first line contains 0 < n <= 100, the number of
freckles on Dick's back. For each freckle, a line follows;
each following line contains two real numbers indicating
the (x,y) coordinates of the freckle.
```

输出：

```
Your program prints a single real number to two decimal
places: the minimum total length of ink lines that can
connect all the freckles.
```

样例输入：

```
3
1.0 1.0
2.0 2.0
2.0 4.0
```

样例输出：

```
3.41
```

来源：

```
2009年北京大学计算机研究生机试真题
```

代码：

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
struct E{
    double x;
    double y;
```

```
} Point[101];

struct F
{
    int a;
    int b;
    double cost;
} Edge[5000];

int Tree[101];

int findRoot(int a)
{
    if (Tree[a] == -1) return a;
    int temp = findRoot(Tree[a]);
    Tree[a] = temp;
    return temp;
}

double getDistance(struct E e1, struct E e2)
{
    return sqrt((e1.x - e2.x) * (e1.x - e2.x) + (e1.y -
e2.y) * (e1.y - e2.y));
}

int compare(const void * a, const void * b)
{
    return (*(struct F*)a).cost - (*(struct F*)b).cost;
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%lf%lf", &Point[i].x, &Point[i].y);
        Tree[i] = -1;
    }

    int count = 0;
    for (int i = 1; i <= n - 1; i++)
    {
        for (int j = i + 1; j <= n; j++)
        {
            Edge[count].a = i;
            Edge[count].b = j;
            Edge[count].cost = getDistance(Point[i],
Point[j]);
            count++;
        }
    }
```

```
    qsort(Edge, n * (n - 1) / 2, sizeof(struct F),
compare);

    double ans = 0;
    for (int i = 0; i < count; i++)
    {
        int rootA = findRoot(Edge[i].a);
        int rootB = findRoot(Edge[i].b);
        if (rootA != rootB)
        {
            Tree[rootA] = rootB;
            ans += Edge[i].cost;
        }
    }
    printf("%.2lf\n", ans);
    system("pause");
    return 0;
}
```

最短路径

例5.5

题目描述：

> 在每年的校赛里，所有进入决赛的同学都会获得一件很漂亮的t-shirt。但是每当我们的工作人员把上百件的衣服从商店运回到赛场的时候，却是非常累的！所以现在他们想要寻找最短的从商店到赛场的路线，你可以帮助他们吗？

输入：

> 输入包括多组数据。每组数据第一行是两个整数N、M （N<=100，M<=10000），
> N 表示成都的大街上有几个路口，标号为1 的路口是商店所在地，标号为N 的路口是赛场所在地， M 则表示在成都有几条路。N=M=0 表示输入结束。接下来M 行，每行包括3 个整数A，B，C（1<=A,B<=N,1<=C<=1000 ）,表示在路口A与路口B 之间有一条路，我们的工作人员需要C 分钟的时间走过这条路。输入保证至少存在1 条商店到赛场的路线。当输入为两个0 时，输入结束。

输出：

> 对于每组输入，输出一行，表示工作人员从商店走到赛场的最短时间。

样例输入：

```
2 1
1 2 3
3 3
1 2 5
2 3 5
3 1 2
0 0
```

样例输出：

```
3
2
```

来源：

```
```

代码：

Floyd算法

```
#include <stdio.h>

int ans[101][101];

int main()
{
    int N, M;
    while (scanf("%d%d", &N, &M))
    {
        if (N == 0 && M == 0) break;
        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j <= N; j++)
            {
                ans[i][j] = -1;
            }
            ans[i][i] = 0;
        }

        for (int i = 0; i < M; i++)
        {
            int A, B, C;
            scanf("%d%d%d", &A, &B, &C);
            ans[A][B] = C;
            ans[B][A] = C;
        }

        for (int k = 1; k <= N; k++)
        {
```

```
            for (int i = 1; i <= N; i++)
            {
                for (int j = 1; j <= N; j++)
                {
                    if (ans[i][k] == -1 || ans[k][j] ==
-1) continue;
                    if (ans[i][j] == -1 || ans[i][j] >
(ans[i][k] + ans[k][j]))
                    {
                        ans[i][j] = ans[i][k] + ans[k][j];
                    }
                }
            }
        }
        printf("%d\n", ans[1][N]);
    }
    return 0;
}
```

Dijkstra算法

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

struct F
{
    int point;
    int distance;
    struct F* next;
};

struct E
{
    int Dis;
    int mark;
    struct F *point;
}edge[101];

int main()
{
    int N, M;
    while (scanf("%d%d", &N, &M))
    {
        if (N == 0 && M == 0) break;
        //初始化
        for (int i = 1; i <= N; i++)
        {
            edge[i].Dis = -1;
            edge[i].mark = 0;
            edge[i].point = NULL;
        }
```

```
        for (int i = 0; i < M; i++)
        {
            int A, B, C;
            scanf("%d%d%d", &A, &B, &C);

            struct F * newEdge= (struct F *)
malloc(sizeof(struct F));
            newEdge->point = B;
            newEdge->distance = C;
            newEdge->next = edge[A].point;
            edge[A].point = newEdge;

            struct F * newEdge2= (struct F *)
malloc(sizeof(struct F));
            newEdge2->point = A;
            newEdge2->distance = C;
            newEdge2->next = edge[B].point;
            edge[B].point = newEdge2;
        }
        edge[1].Dis = 0;
        edge[1].mark = 1;
        int newP = 1;
        for (int k = 1; k < N; k++)
        {
            struct F* temp = edge[newP].point;
            while (temp != NULL)
            {
                if (edge[temp->point].mark == 1)
                {
                    temp = temp->next;
                    continue;
                }
                if (edge[temp->point].Dis = -1 ||
edge[temp->point].Dis > (edge[newP].Dis + temp->distance))
                {
                    edge[temp->point].Dis = edge[newP].Dis
+ temp->distance;
                }
                temp = temp->next;
            }
            int min = 1001;
            for (int i = 1; i <= N; i++)
            {
                if (edge[i].mark == 1) continue;
                if (edge[i].Dis == -1) continue;
                if (edge[i].Dis < min)
                {
                    min = edge[i].Dis;
                    newP = i;
                }
            }
            edge[newP].mark = 1;
```

```
        }
        printf("%d\n", edge[N].Dis);
    }
    return 0;
}
```

例5.6

题目描述：

> 给你n 个点，m 条无向边，每条边都有长度d 和花费p，给你起点s 终点t，要求输出起点到终点的最短距离及其花费，如果最短距离有多条路线，则输出花费最少的。

输入：

> 输入n,m，点的编号是1~n,然后是m 行，每行4 个数a,b,d,p，表示a 和b之间有一条边，且其长度为d，花费为p。最后一行是两个数s,t;起点s，终点t。n 和m 为0 时输入结束。(1<n<=1000, 0<m<100000, s != t)

输出：

> 输出一行有两个数，最短距离及其花费。

样例输入：

```
3 2
1 2 5 6
2 3 4 5
1 3
0 0
```

样例输出：

```
9 11
```

来源：

> 2010年浙江大学计算机及软件工程研究生机试真题

代码：

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

```
struct F
{
    int point;
    int distance;
    int cost;
    struct F *next;
};

struct E
{
    int Dis;
    int costSum;
    int mark;
    struct F *point;
} edge[101];

int main()
{
    int N, M;
    while (scanf("%d%d", &N, &M))
    {
        if (N == 0 && M == 0)
            break;
        //初始化
        for (int i = 1; i <= N; i++)
        {
            edge[i].Dis = -1;
            edge[i].costSum = 0;
            edge[i].mark = 0;
            edge[i].point = NULL;
        }
        for (int i = 0; i < M; i++)
        {
            int A, B, C, D;
            scanf("%d%d%d%d", &A, &B, &C, &D);

            struct F *newEdge = (struct F
*)malloc(sizeof(struct F));
            newEdge->point = B;
            newEdge->distance = C;
            newEdge->cost = D;
            newEdge->next = edge[A].point;
            edge[A].point = newEdge;

            struct F *newEdge2 = (struct F
*)malloc(sizeof(struct F));
            newEdge2->point = A;
            newEdge2->distance = C;
            newEdge->cost = D;
            newEdge2->next = edge[B].point;
            edge[B].point = newEdge2;
        }
```

```
        int S, D;
        scanf("%d%d", &S, &D);
        edge[S].Dis = 0;
        edge[S].mark = 1;
        int newP = S;
        for (int k = 1; k < N; k++)
        {
            struct F *temp = edge[newP].point;
            while (temp != NULL)
            {
                if (edge[temp->point].mark == 1)
                {
                    temp = temp->next;
                    continue;
                }
                if (edge[temp->point].Dis = -1 ||
edge[temp->point].Dis > (edge[newP].Dis + temp->distance)
|| edge[temp->point].Dis == (edge[newP].Dis + temp-
>distance) && edge[temp->point].costSum >
(edge[newP].costSum + temp->cost))
                { //
                    edge[temp->point].Dis = edge[newP].Dis
+ temp->distance;
                    edge[temp->point].costSum =
edge[newP].costSum + temp->cost;
                }
                temp = temp->next;
            }
            int min = 123123123;
            for (int i = 1; i <= N; i++)
            {
                if (edge[i].mark == 1)
                    continue;
                if (edge[i].Dis == -1)
                    continue;
                if (edge[i].Dis < min)
                {
                    min = edge[i].Dis;
                    newP = i;
                }
            }
            edge[newP].mark = 1;
        }
        printf("%d %d\n", edge[D].Dis, edge[D].costSum);
    }
    return 0;
}
```

拓扑排序

例5.7

题目描述：

> ACM-DIY is a large QQ group where many excellent acmers get together. It is so harmonious that just like a big family. Every day,many "holy cows" like HH, hh, AC, ZT, lcc, BF, Qinz and so on chat on-line to exchange their ideas. When someone has questions, many warm-hearted cows like Lost will come to help. Then the one being helped will call Lost "master", and Lost will have a nice "prentice". By and by,
> there are many pairs of "master and prentice". But then problem occurs: there are too many masters and too many prentices, how can we know whether it is legal or not?We all know a master can have many prentices and a prentice may have a lot of masters too, it's legal. Nevertheless, some cows are not so honest, they hold illegal relationship. Take HH and 3xian for instant, HH is 3xian's master and, at the same time, 3xian is HH's master,which is quite illegal! To avoid this,please help us to judge whether their relationship is legal or not. Please note that the "master and prentice" relation is transitive. It means that if A is B's master ans B is C's master, then A is C's master.

输入：

> The input consists of several test cases. For each case, the first line contains two integers, N (members to be tested) and M (relationships to be tested)(2 <= N, M <= 100). Then M lines follow, each contains a pair of (x, y) which means x is y's master and y is x's prentice. The input is terminated by N = 0.TO MAKE IT SIMPLE, we give every one a number (0, 1, 2,..., N-1). We use their numbers instead of their names.

输出：

> For each test case, print in one line the judgement of the messy relationship.If it is legal, output "YES", otherwise "NO".

样例输入：

```
3 2
0 1
1 2
2 2
0 1
1 0
0 0
```

样例输出：

```
YES
NO
```

来源：

```
```

代码：

```
#include<stdio.h>
#include<stdlib.h>
struct E
{
    int data;
    struct E *next;
}node[101];

struct F
{
    int rear;
    int head;
    int data[101];
} queue;

int innode[101];

int main()
{
    int N, M;
    while (scanf("%d%d", &N, &M) != EOF)
    {
        if (N == 0) break;

        //队列初始化
        queue.head = -1;
        queue.rear = -1;

        //邻接表初始化
        for (int i = 0; i < N; i++)
        {
```

```
            innode[i] = 0;
            node[i].data = 0;
            node[i].next = NULL;
        }

        for (int i = 0; i < M; i++)
        {
            int x, y;
            scanf("%d%d", &x, &y);
            struct E *newY = (struct
E*)malloc(sizeof(struct E));
            newY->data = y;
            newY->next = node[x].next;
            node[x].next = newY;
            innode[y]++;
        }
        int count = 0;
        for (int i = 0; i < N; i++)
        {
            if (innode[i] == 0)
            {
                queue.data[++queue.rear] = i;
                count++;
            }
        }
        while (queue.head != queue.rear)
        {
            int target = queue.data[++queue.head];
            while (node[target].next != NULL)
            {
                struct E* temp = node[target].next;
                int a = temp->data;
                node[target].next = temp->next;
                free(temp);
                innode[a]--;
                if (innode[a] == 0)
                {
                    queue.data[++queue.rear] = a;
                    count++;
                }
            }

        }
        if (count == N) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

六、搜索

枚举

例6.1

题目描述：

> 用小于等于n元去买100只鸡，大鸡5元/只，小鸡3元/只，还有1/3 元每只的一种小鸡，分别记为x只，y只，z只。编程求解x，y，z 所有可能解。

输入：

> 测试数据有多组，输入n。

输出：

> 对于每组输入，请输出x，y，z 所有可行解，按照x，y，z 依次增大的顺序输出。

样例输入：

```
40
```

样例输出：

```
x=0,y=0,z=100
x=0,y=1,z=99
x=0,y=2,z=98
x=1,y=0,z=99
```

来源：

> 2009年哈尔滨工业大学计算机研究生机试真题

代码：

```
#include <stdio.h>
int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        for (int x = 0; x <= 100; x++)
        {
```

```
            for (int y = 0; y <= 100 - x; y++)
            {
                int z = 100 - x - y;
                if (x * 5 * 3+ y * 3 * 3 + z < n * 3)
                {
                    printf("x=%d,y=%d,z=%d\n", x , y, z);
                }
            }
        }
    }
    return 0;
}
```

广度优先搜索(BFS)

例6.2

题目描述：

> Ignatius被魔王抓走了,有一天魔王出差去了,这可是Ignatius逃亡的好机会.魔王住在一个城堡里,城堡是一个A*B*C 的立方体,可以被表示成A个B*C的矩阵,刚开始Ignatius被关在(0,0,0)的位置,离开城堡的门在(A-1,B-1,C-1)的位置,现在知道魔王将在T分钟后回到城堡,Ignatius每分钟能从一个坐标走到相邻的六个坐标中的其中一个.现在给你城堡的地图,请你计算出Ignatius能否在魔王回来前离开城堡(只要走到出口就算离开城堡,如果走到出口的时候魔王刚好回来也算逃亡成功),如果可以请输出需要多少分钟才能离开,如果不能则输出-1。

输入：

> 输入数据的第一行是一个正整数K,表明测试数据的数量.每组测试数据的第一行是四个正整数A,B,C 和T(1<=A,B,C<=50,1<=T<=1000), 它们分别代表城堡的大小和魔王回来的时间.然后是A 块输入数据(先是第0 块,然后是第1块,第2块.....),每块输入数据有B 行,每行有C 个正整数,代表迷宫的布局,其中0 代表路,1代表墙。

输出：

> 对于每组测试数据,如果Ignatius 能够在魔王回来前离开城堡,那么请输出他最少需要多少分钟,否则输出-1.

样例输入：

```
2
3 3 4 20
0 1 1 1
0 0 1 1
0 1 1 1
1 1 1 1
1 0 0 1
0 1 1 1
0 0 0 0
0 1 1 0
0 1 1 0
```

样例输出：

```
11
```

来源：

```
```

代码：

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

using namespace std;

int maze[50][50][50];
bool mark[50][50][50];

int go[6][3] = {
    1, 0, 0,
    0, 1, 0,
    0, 0, 1,
    -1, 0, 0,
    0, -1, 0,
    0, 0, -1
};

typedef struct node
{
    int x, y, z;
    int t;
}Node;

std::queue<Node> Q;

int BFS(int a, int b, int c)
{
    while (!Q.empty())
```

```
    {
        Node now = Q.front();
        Q.pop();
        for (int i = 0; i < 6; i++)
        {
            int nx = now.x + go[i][0];
            int ny = now.y + go[i][1];
            int nz = now.z + go[i][2];
            if (nx < 0 || nx >= a || ny < 0 || ny >= b ||
nz < 0 || nz >= c) continue;
            if (maze[nx][ny][nz] == 1) continue;
            if (mark[nx][ny][nz] == true) continue;
            Node newNode;
            newNode.x = nx;
            newNode.y = ny;
            newNode.z = nz;
            newNode.t = now.t + 1;
            Q.push(newNode);
            mark[nx][ny][nz] = true;
            if (nx == a - 1 && ny == b - 1 && nz == c - 1)
return newNode.t;
        }
    }
    return -1;
}

int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        int a, b, c, t;
        scanf("%d%d%d%d", &a, &b, &c, &t);
        for(int i = 0; i < a; i++)
        {
            for(int j = 0; j < b; j++)
            {
                for (int k = 0; k < c; k++)
                {
                    scanf("%d", &maze[i][j][k]);
                    mark[i][j][k] = false;
                }
            }
        }
        while (Q.empty() == false) Q.pop();
        mark[0][0][0] = true;
        Node node;
        node.x = node.y = node.z = node.t = 0;
        Q.push(node);
        int ans = BFS(a, b, c);
        if (ans <= t)printf("%d\n", ans);
```

```
        else printf("-1\n");
    }
    return 0;
}
```

例6.3

题目描述：

> 大家一定觉的运动以后喝可乐是一件很惬意的事情，但是see you却不这么认为。因为每次当see you买了可乐以后，阿牛就要求和see you一起分享这一瓶可乐，而且一定要喝的和see you一样多。但see you的手中只有两个杯子，它们的容量分别是N 毫升和M 毫升可乐的体积为S （S<101）毫升(正好装满一瓶），它们三个之间可以相互倒可乐(都是没有刻度的，且S==N+M ，101>S>0，N>0，M>0) 。聪明的ACMER 你们说他们能平分吗？如果能请输出倒可乐的最少的次数，如果不能输出"NO"。

输入：

> 三个整数: S可乐的体积,N和M是两个杯子的容量，以"0 0 0" 结束。

输出：

> 如果能平分的话请输出最少要倒的次数，否则输出"NO" 。

样例输入：

```
7 4 3
4 1 3
0 0 0
```

样例输出：

```
NO
3
```

来源：

代码：

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

typedef struct node
{
```

```
    int s, n, m, t;
}Node;

bool mark[101][101][101] = {false};

std::queue<Node> Q;

void AtoB(int &a, int sa, int &b, int sb)
{
    if (sb - b >= a)
    {
        b += a;
        a = 0;
    } else
    {
        a -= sb - b;
        b = sb;
    }

}

int BFS(int s, int n, int m)
{
    while (!Q.empty())
    {
        Node now = Q.front();
        Q.pop();
        int s1, n1, m1;
        //s to n
        s1 = now.s;
        n1 = now.n;
        m1 = now.m;
        AtoB(s1, s, n1, n);
        if (mark[s1][n1][m1] == false)
        {
            mark[s1][n1][m1] = true;
            Node newNode;
            newNode.s = s1;
            newNode.n = n1;
            newNode.m = m1;
            newNode.t = now.t + 1;
            if (s1 * 2 == s && n1 * 2 == s) return
newNode.t;
            if (s1 * 2 == s && m1 * 2 == s) return
newNode.t;
            if (n1 * 2 == s && m1 * 2 == s) return
newNode.t;
            Q.push(newNode);
        }
        //n to s
        s1 = now.s;
        n1 = now.n;
```

```
        m1 = now.m;
        AtoB(n1, n, s1, s);
        if (mark[s1][n1][m1] == false)
        {
            mark[s1][n1][m1] = true;
            Node newNode;
            newNode.s = s1;
            newNode.n = n1;
            newNode.m = m1;
            newNode.t = now.t + 1;
            if (s1 * 2 == s && n1 * 2 == s) return
newNode.t;
            if (s1 * 2 == s && m1 * 2 == s) return
newNode.t;
            if (n1 * 2 == s && m1 * 2 == s) return
newNode.t;
            Q.push(newNode);
        }
        //s to m
        s1 = now.s;
        n1 = now.n;
        m1 = now.m;
        AtoB(s1, s, m1, m);
        if (mark[s1][n1][m1] == false)
        {
            mark[s1][n1][m1] = true;
            Node newNode;
            newNode.s = s1;
            newNode.n = n1;
            newNode.m = m1;
            newNode.t = now.t + 1;
            if (s1 * 2 == s && n1 * 2 == s) return
newNode.t;
            if (s1 * 2 == s && m1 * 2 == s) return
newNode.t;
            if (n1 * 2 == s && m1 * 2 == s) return
newNode.t;
            Q.push(newNode);
        }
        //m to s
        s1 = now.s;
        n1 = now.n;
        m1 = now.m;
        AtoB(m1, m, s1, s);
        if (!mark[s1][n1][m1])
        {
            mark[s1][n1][m1] = true;
            Node newNode;
            newNode.s = s1;
            newNode.n = n1;
            newNode.m = m1;
            newNode.t = now.t + 1;
```

```
            if (s1 * 2 == s && n1 * 2 == s) return
newNode.t;
            if (s1 * 2 == s && m1 * 2 == s) return
newNode.t;
            if (n1 * 2 == s && m1 * 2 == s) return
newNode.t;
            Q.push(newNode);
        }
        //n to m
        s1 = now.s;
        n1 = now.n;
        m1 = now.m;
        AtoB(n1, n, m1, m);
        if (mark[s1][n1][m1] == false)
        {
            mark[s1][n1][m1] = true;
            Node newNode;
            newNode.s = s1;
            newNode.n = n1;
            newNode.m = m1;
            newNode.t = now.t + 1;
            if (s1 * 2 == s && n1 * 2 == s) return
newNode.t;
            if (s1 * 2 == s && m1 * 2 == s) return
newNode.t;
            if (n1 * 2 == s && m1 * 2 == s) return
newNode.t;
            Q.push(newNode);
        }
        //m to n
        s1 = now.s;
        n1 = now.n;
        m1 = now.m;
        AtoB(m1, m, n1, n);
        if (mark[s1][n1][m1] == false)
        {
            mark[s1][n1][m1] = true;
            Node newNode;
            newNode.s = s1;
            newNode.n = n1;
            newNode.m = m1;
            newNode.t = now.t + 1;
            if (s1 * 2 == s && n1 * 2 == s) return
newNode.t;
            if (s1 * 2 == s && m1 * 2 == s) return
newNode.t;
            if (n1 * 2 == s && m1 * 2 == s) return
newNode.t;
            Q.push(newNode);
        }
    }
    return -1;
```

```
}

int main()
{
    int s, n, m;
    while (scanf("%d%d%d", &s, &n, &m))
    {
        if (s == 0 && n == 0 && m == 0) break;
        if (s % 2 != 0)
        {
            printf("NO\n");
            continue;
        }
        for (int i = 0; i <= s; i++)
        {
            for (int j = 0; j <= n; j++)
            {
                for(int k = 0; k <= m; k++)
                {
                    mark[i][j][k] = false;
                }
            }
        }
        Node newNode;
        newNode.s = s;
        newNode.n = newNode.m = 0;
        newNode.t = 0;
        Q.push(newNode);
        int ans = BFS(s, n, m);
        if (ans == -1) printf("NO\n");
        else printf("%d\n", ans);
    }
    return 0;
}
```

递归

例6.4

题目描述：

> 约19世纪末，在欧州的商店中出售一种智力玩具，在一块铜板上有三根杆，最左边的杆上自上而下、由小到大顺序串着由64 个圆盘构成的塔。目的是将最左边杆上的盘全部移到右边的杆上，条件是一次只能移动一个盘，且不允许大盘放在小盘的上面。现在我们改变游戏的玩法，不允许直接从最左(右)边移到最右(左)边(每次移动一定是移到中间杆或从中间移出)， 也不允许大盘放到下盘的上面。Daisy 已经做过原来的汉诺塔问题和汉诺塔II ，但碰到这个问题时，她想了很久都不能解决，现在请你帮助她。现在有N个圆盘，她至少多少次移动才能把这些圆盘从最左边移到最右边？

输入：

```
包含多组数据，每次输入一个N 值(1<=N=35)。
```

输出：

```
对于每组数据，输出移动最小的次数。
```

样例输入：

```
1
3
12
```

样例输出：

```
2
26
531440
```

来源：

```

```

代码：

```
#include<stdio.h>
long long F(int n)
{
    if (n == 1) return 2;
    return 3 * F(n - 1) + 2;
}
int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        printf("%lld\n", F(n));
    }
    return 0;
}
```

递归的应用

例6.5（回溯法枚举）

题目描述：

```
A ring is compose of n circles as shown in diagram. Put
natural number 1, 2, ..., n into each circle separately,
and the sum of numbers in two adjacent circles should be a
prime.
Note: the number of first circle should always be 1.
```

输入：

```
n (1 < n < 17).
```

输出：

```
The output format is shown as sample below. Each row
represents a series of circle numbers in the ring
beginning from 1 clockwisely and anticlockwisely. The
order of numbers must satisfy the above requirements.
Print solutions in
lexicographical order.You are to write a program that
completes above process.Print a blank line after each
case.
```

样例输入：

```
6
8
```

样例输出：

```
Case 1:
143256
165234
Case 2:
12385674
12583476
14765832
16743852
```

来源：

```

```

代码：

```
#include <stdio.h>
#include <string.h>
int ans[22];
bool hash[22];
int n;
```

```
int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41};
bool judge(int x)
{
    for (int i = 0; i < 13; i++)
    {
        if (prime[i] == x)
            return true;
    }
    return false;
}
void check()
{
    if (judge(ans[n] + ans[1]) == false)
        return;
    for (int i = 1; i <= n; i++)
    {
        if (i != 1)
            printf(" ");
        printf("%d", ans[i]);
    }
    printf("\n");
}

void DFS(int num)
{
    if (num > 1)
        if (judge(ans[num] + ans[num - 1]) == false)
            return;
    if (num == n)
    {
        check();
        return;
    }
    for (int i = 2; i <= n; i++)
    {
        if (hash[i] == false)
        {
            hash[i] = true;
            ans[num + 1] = i;
            DFS(num + 1);
            hash[i] = false;
        }
    }
}
int main()
{
    int cas = 0;
    while (scanf("%d", &n) != EOF)
    {
        cas++;
        for (int i = 0; i < 22; i++)
```

```
            hash[i] = false;
        ans[1] = 1;
        printf("Case %d:\n", cas);
        hash[1] = true;
        DFS(1);
        printf("\n");
    }
    return 0;
}
```

例6.6（图的遍历）

题目描述：

```
The GeoSurvComp geologic survey company is responsible for
detecting underground oil deposits. GeoSurvComp works with
one large rectangular region of land at a time, and
creates a grid that divides the land into numerous square
plots. It
then analyzes each plot separately, using sensing
equipment to determine whether or not the plot contains
oil. A plot containing oil is called a pocket. If two
pockets are
adjacent, then they are part of the same oil deposit. Oil
deposits can be quite large and may contain numerous
pockets. Your job is to determine how many different oil
deposits are contained in a grid.
```

输入：

```
The input file contains one or more grids. Each grid
begins with a line containing m and n, the number of rows
and columns in the grid, separated by a single space. If m
= 0 it signals the end of the input; otherwise 1 <= m <=
100 and 1 <= n <= 100.Following this are m lines of n
characters each (not counting the end-of-line characters).
Each character corresponds to one plot, and is either `*',
representing the absence of oil, or `@', representing an
oil pocket.
```

输出：

```
For each grid, output the number of distinct oil deposits.
Two different pockets are part of the same oil deposit if
they are adjacent horizontally, vertically, or diagonally.
An oil deposit will not contain more than 100 pockets.
```

样例输入：

```
```

```
1 1
*
3 5
*@*@*
**@**
*@*@*
1 8
@ @ * * * * @ *
5 5
****@
*@@*@
*@**@
@@@*@
@@**@
0 0
```

样例输出：

```
0
1
2
2
```

来源：

```
```

代码：

```
#include<stdio.h>

char maze[101][101];
int mark[101][101];
int n, m;
int go[8][2] = {
    1, 0,
    0, 1,
    -1, 0,
    0, -1,
    1, 1,
    1, -1,
    -1, 1,
    -1, -1
};

void DFS(int x, int y)
{
    for (int i = 0; i < 8; i++)
    {
        int nx = x + go[i][0];
```

```
        int ny = y + go[i][1];
        if (nx < 0 || nx >= m || ny < 0 || ny >= n)
continue;
        if (mark[nx][ny] == 1) continue;
        if (maze[nx][ny] == '*') continue;
        mark[nx][ny] = 1;
        DFS(nx, ny);
    }
}

int main()
{
    while (scanf("%d%d", &m, &n) != EOF)
    {
        int ans = 0;
        if (m == 0 && n == 0) break;

        for (int i = 0; i < m; i++)
        {
            scanf("%s", maze[i]);
        }

        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                mark[i][j] = 0;
            }
        }

        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (mark[i][j] == 0 && maze[i][j] == '@')
                {
                    ans++;
                    DFS(i, j);
                }
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

DFS

例6.6

题目描述：

```
The doggie found a bone in an ancient maze, which
fascinated him a lot. However, when he picked it up, the
maze began to shake, and the doggie could feel the ground
sinking. He realized that the bone was a trap, and he
tried desperately to get out of this maze. The maze was a
rectangle with sizes N by M. There was a door in the maze.
At the beginning, the door was closed and it would open at
the T-th second for a short period of time (less than 1
second). Therefore the doggie had to arrive at the door on
exactly the T-th second. In every second, he could move
one block to one of the upper,lower, left and right
neighboring blocks. Once he entered a block, the ground of
this block would start to sink and disappear in the next
second. He could not stay at one block for more than one
second, nor could he move into a visited block. Can the
poor doggie survive? Please help him.
```

输入：

```
The input consists of multiple test cases. The first line
of each test case contains three integers N, M, and T (1 <
N, M < 7; 0 < T < 50), which denote the sizes of the
maze and the time at which the door will open,
respectively. The next N lines give the maze layout, with
each line containing M characters. A character is one of
the
following:
'X': a block of wall, which the doggie cannot enter;
'S': the start point of the doggie;
'D': the Door; or
'.': an empty block.
The input is terminated with three 0's. This test case is
not to be processed.
```

输出：

```
For each test case, print in one line "YES" if the doggie
can survive, or "NO" otherwise.
```

样例输入：

```
4 4 5
S.X.
..X.
..XD
....
3 4 5
S.X.
..X.
...D
0 0 0
```

样例输出：

```
NO
YES
```

来源：

```
```

代码：

```
#include<stdio.h>

char maze[7][7];

int success = 0;

int n, m, t;

int go[4][2] = {
    0, 1,
    1, 0,
    -1, 0,
    0, -1
};

void DFS(int x, int y, int time)
{
    for(int i = 0; i < 4; i++)
    {
        int nx = x + go[i][0];
        int ny = y + go[i][1];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m)
continue;
        if (maze[nx][ny] == 'X') continue;
        if (maze[nx][ny] == 'D')
        {
            if (time + 1 == t)
            {
```

```
                success = 1;
                return;
            } else
            {
                continue;
            }
        }
        maze[nx][ny] = 'X';
        DFS(nx, ny, time + 1);
        maze[nx][ny] = '.';

    }
}

int main()
{
    while(scanf("%d%d%d", &n, &m, &t) != EOF)
    {
        success = 0;
        int nx, ny;
        if (n == 0 && m == 0 && t == 0) break;
        for (int i = 0; i < n; i++)
        {
            scanf("%s", maze[i]);
        }
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (maze[i][j] == 'S')
                {
                    nx = i;
                    ny = j;
                }
            }
        }

        DFS(nx, ny, 0);
        if (success == 1) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

七、动态规划

递推求解

例7.1

题目描述：

> N阶楼梯上楼问题：一次可以走两阶或一阶，问有多少种上楼方式。（要求采用非递归）

输入：

> 输入包括一个整数N,(1<=N<90) 。

输出：

> 可能有多组测试数据，对于每组数据，输出当楼梯阶数是N 时的上楼方式个数。

样例输入：

```
4
```

样例输出：

```
5
```

来源：

> 2008 年华中科技大学计算机保研机试真题

代码：

```
#include<stdio.h>
long long F[91];

int main()
{
    F[1] = 1;
    F[2] = 2;
    for (int i = 3; i <= 90; i++)
    {
        F[i] = F[i - 1] + F[i - 2];
    }
    int n;
    while (scanf("%d", &n) != EOF)
    {
        printf("%lld\n", F[n]);
    }
    return 0;
}
```

最长递增子序列

题目描述：

> 某国为了防御敌国的导弹袭击，开发出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，并观测到导弹依次飞来的高度，请计算这套系统最多能拦截多少导弹。拦截来袭导弹时，必须按来袭导弹袭击的时间顺序，不允许先拦截后面的导弹，再拦截前面的导弹。

输入：

> 每组输入有两行，
> 第一行，输入雷达捕捉到的敌国导弹的数量k（k<=25），
> 第二行，输入k 个正整数，表示k 枚导弹的高度，按来袭导弹的袭击时间顺序给出，以空格分隔。

输出：

> 每组输出只有一行，包含一个整数，表示最多能拦截多少枚导弹。

样例输入：

```
8
300 207 155 300 299 170 158 65
```

样例输出：

```
6
```

来源：

> 2007 年北京大学计算机研究生机试真题

代码：

```
#include<stdio.h>

int list[26];
int dp[26];

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

```
            scanf("%d", &list[i]);
        }
        for (int i = 1; i <= n; i++)
        {
            int tmax = 1;
            for (int j = 1; j < i; j++)
            {
                if (list[j] >= list[i])
                {
                    tmax = dp[j] + 1 > tmax ? dp[j] + 1 :
tmax;
                }
            }
            dp[i] = tmax;
        }
        printf("%d\n", dp[n]);
    }
    return 0;
}
```

最长公共子序列

题目描述：

```
Find a longest common subsequence of two strings.
```

输入：

```
First and second line of each input case contain two
strings of lowercase character a …z. There are no spaces
before, inside or after the strings. Lengths of strings do
not exceed 100.
```

输出：

```
For each case, output k-the length of a longest common
subsequence in one line.
```

样例输入：

```
abcd
cxbydz
```

样例输出：

```
2
```

来源：

> 2008年上海交通大学计算机研究生机试真题

代码：

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

int max(int a, int b)
{
    return a > b ? a : b;
}
int dp[101][101];

int main()
{
    char s1[101], s2[101];
    while (scanf("%s%s", s1, s2) != EOF)
    {
        int l1 = strlen(s1);
        int l2 = strlen(s2);
        for (int i = 0; i <= l1; i++) dp[i][0] = 0;
        for (int i = 0; i <= l2; i++) dp[0][i] = 0;
        for (int i = 1; i <= l1; i++)
        {
            for (int j = 1; j <= l2; j++)
            {
                if (s1[i - 1] == s2[j - 1])
                {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else
                {
                    dp[i][j] = max(dp[i][j - 1], dp[i - 1]
[j]);
                }
            }
        }
        printf("%d\n", dp[l1][l2]);
    }
    return 0;
}
```

背包问题

例7.7(01背包)

题目描述：

辰辰是个很有潜能、天资聪颖的孩子，他的梦想是称为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到个到处都是草药的山洞里对他说： –孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。如果你是辰辰，你能完成这个任务吗？

输入：

输入的第一行有两个整数T（1 <= T <= 1000）和M（1 <= M <= 100 ），T 代表总共能够用来采药的时间， M 代表山洞里的草药的数目。接下来的M行每行包括两个在1 到100 之间（包括1 和100）的的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出：

可能有多组测试数据，对于每组数据，输出只包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

样例输入：

```
70 3
71 100
69 1
1 2
```

样例输出：

```
3
```

来源：

2008年北京大学图形实验室计算机研究生机试真题

代码：

```
#include<stdio.h>

struct E
{
    int w;
    int v;
} list[101];

int max(int a, int b)
{
    return a > b ? a : b;
}
```

```
int dp[101][1001];

int main()
{
    int T, M;
    while (scanf("%d%d", &T, &M))
    {
        for (int i = 1; i <= M; i++)
        {
            scanf("%d%d", &list[i].w, &list[i].v);
        }

        for (int i = 0; i <= T; i++)
        {
            dp[0][T] = 0;
        }

        for (int i = 1; i <= M;i ++)
        {
            for (int j = T; j >= list[i].w; j--)
            {
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j -
list[i].w] + list[i].v);
            }
            for (int j = 0; j < list[i].w; j++)
            {
                dp[i][j] = dp[i - 1][j];
            }
        }
        printf("%d\n", dp[M][T]);
    }
    return 0;
}
```

或：

```
#include<stdio.h>

struct E
{
    int w;
    int v;
} list[101];

int max(int a, int b)
{
    return a > b ? a : b;
}

int dp [1001];
```

```
int main()
{
    int T, M;
    while (scanf("%d%d", &T, &M))
    {
        for (int i = 1; i <= M; i++)
        {
            scanf("%d%d", &list[i].w, &list[i].v);
        }

        for (int i = 0; i <= T; i++)
        {
            dp[T] = 0;
        }

        for (int i = 1; i <= M;i ++)
        {
            for (int j = T; j >= list[i].w; j--)
            {
                dp[j] = max(dp[j], dp[j - list[i].w] +
list[i].v);
            }
        }
        printf("%d\n", dp[T]);
    }
    return 0;
}
```

例子7.8(完全背包)

题目描述：

Before ACM can do anything, a budget must be prepared and the necessary financial support obtained. The main income for this action comes from Irreversibly Bound Money (IBM). The idea behind is simple. Whenever some ACM member has any small money, he takes all the coins and throws them into a piggy-bank. You know that this process is irreversible, the coins cannot be removed without breaking the pig. After a sufficiently long time, there should be enough cash in the piggy-bank to pay everything that needs to be paid.

But there is a big problem with piggy-banks. It is not possible to determine how much money is inside. So we might break the pig into pieces only to find out that there is not enough money. Clearly, we want to avoid this unpleasant situation. The only possibility is to weigh the piggy-bank and try to guess how many coins are inside. Assume that we are able to determine the weight of the pig exactly and that we know the weights of all coins of a given currency. Then there is some minimum amount of money in the piggy-bank that we can guarantee. Your task is to find out this worst case and determine the minimum amount of cash inside the piggy-bank. We need your help. No more prematurely broken pigs!

输入：

The input consists of T test cases. The number of them (T) is given on the first line of the input file. Each test case begins with a line containing two integers E and F. They indicate the weight of an empty pig and of the pig filled with coins. Both weights are given in grams. No pig will weigh more than 10 kg, that means 1 <= E <= F <= 10000. On the second line of each test case, there is an integer number N (1 <= N <= 500) that gives the number of various coins used in the given currency. Following this are exactly N lines, each specifying one coin type. These lines contain
two integers each, P and W (1 <= P <= 50000, 1 <= W <=10000). P is the value of the coin in monetary units, W is it's weight in grams.

输出：

Print exactly one line of output for each test case. The line must contain the sentence "The minimum amount of money in the piggy-bank is X." where X is the minimum amount of money that can be achieved using coins with the given total weight. If the weight cannot be reached exactly, print a line "This is impossible.".

样例输入：

```
3
10 110
2
1 1
30 50
10 110
2
1 1
50 30
1 6
2
10 3
20 4
```

样例输出：

```
The minimum amount of money in the piggy-bank is 60.
The minimum amount of money in the piggy-bank is 100.
This is impossible.
```

来源：

```

```

代码：

```
#include <stdio.h>
#define INF 0x7fffffff
int min(int a, int b)
{
    return a < b ? a : b;
}
struct test
{
    int v;
    int w;
} list[501];

int dp[10001];

int main()
{
    int T;
    scanf("%d", &T);
    for (int i = 0; i < T; i++)
    {
        int E, F;
        scanf("%d%d", &E, &F);
```

```
        int N;
        scanf("%d", &N);
        for (int i = 1; i <= N; i++)
        {
            scanf("%d%d", &list[i].v, &list[i].w);
        }
        int coinweight = F - E;
        dp[0] = 0;
        for (int i = 1; i <= coinweight; i++)
        {
            dp[i] = INF;
        }
        for (int i = 1; i <= N; i++)
        {
            for (int j = list[i].w; j <= coinweight; j++)
            {
                if (dp[j - list[i].w] != INF)
                {
                    dp[j] = min(dp[j], dp[j - list[i].w] +
list[i].v);
                }
            }
        }
        if (dp[coinweight] != INF)
            printf("The minimum amount of money in the
piggy-bank is %d\n", dp[coinweight]);
        else
            printf("This is impossible");
    }
    return 0;
}
```

例7.8(多重背包问题)

题目描述：

> 为了挽救灾区同胞的生命，心系灾区同胞的你准备自己采购一些粮食支援灾区，现在假设你一共有资金n 元，而市场有m 种大米，每种大米都是袋装产品，其价格不等，并且只能整袋购买。请问：你用有限的资金最多能采购多少公斤粮食呢？

输入：

> 输入数据首先包含一个正整数C，表示有C 组测试用例，每组测试用例的第一行是两个整数n 和m(1<=n<=100, 1<=m<=100),分别表示经费的金额和大米的种类， 然后是m 行数据， 每行包含3 个数p ， h 和 c(1<=p<=20,1<=h<=200,1<=c<=20)，分别表示每袋的价格、每袋的重量以及对应

输出：

对于每组测试数据，请输出能够购买大米的最多重量，你可以假设经费买不光所有的大米，并且经费你可以不用完。每个实例的输出占一行。

样例输入：

```
2
8 2
2 100 4
4 100 2
```

样例输出：

```
400
```

来源：

```
```

代码：

```
#include<stdio.h>

int max(int a, int b)
{
    return a > b ? a : b;
}

struct E
{
    int v;
    int w;
    int c;
}list[101];

int dp[101];

int main()
{
    int C;
    scanf("%d", &C);
    for (int i = 0; i < C; i++)
    {
        int n, m;
        scanf("%d%d", &n, &m);
        int cnt = 0;
        for (int j = 1;j <= m; j++)
        {
            int w, v, k;
            scanf("%d%d%d", &w, &v, &k);
```

```
            int c = 1;
            while (k - c > 0)
            {
                k -= c;
                list[++cnt].w = c * w;
                list[cnt].v = c * v;
                c *= 2;
            }
            list[++cnt].w = w * k;
            list[cnt].v = v * k;
        }
        for (int j = 0; j <= n; j++)
        {
            dp[j] = 0;
        }
        for (int j = 1; j <= cnt; j++)
        {
            for (int z = n; z >= list[j].w; z--)
            {
                dp[z] = max(dp[z], dp[z - list[j].w] +
list[j].v);
            }
        }
        printf("%d\n", dp[n]);
    }
    return 0;
}
```