



程序实践报告

2018.8.24

曾慧敏

专业：2017 电子信息工程

学号 17020022049

目录

1、放苹果-----2

1.1 问题描述-----2

1.2 问题分析-----2

1.2.1 动态分析-----2

1.2.2 递归做法-----2

1.3 流程图-----3

1.4 程序实现-----4

1.5 结果分析-----4

1.6 反思总结-----5

2、不吉利日期-----6

1.1 问题描述-----6

1.2 问题分析-----6

1.3 流程图-----7

1.4 程序实现-----8

1.5 结果分析-----9

1.6 反思总结-----9

3、魔兽备战-----10

1.1 问题描述-----10

1.2 问题分析-----11

1.3 流程图-----11

1.4 程序实现-----12

1.5 结果分析-----15

1.6 反思总结-----16

放苹果

1、问题描述

把 M 个同样的苹果放在 N 个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用 K 表示）
5, 1, 1 和 1, 5, 1 是同一种分法。

Input

第一行是测试数据的数目 t ($0 \leq t \leq 20$)。以下每行均包含二个整数 M 和 N ，以空格分开。 $1 \leq M, N \leq 10$

Output

对输入的每组数据 M 和 N ，用一行输出相应的 K

Sample Input 1

1
7 3

Sample Output 1

8

2、问题分析

设有 m 个苹果， n 个盘，用 $\text{map}[m][n]$ 来记录放置的情况。

- 动态规划(1ms)

$m < n$: 此种情况下总会有多出来的盘子，则可以只考虑 m 个苹果， m 个盘子的情况

$m \geq n$: 两种情况：

1、至少有一个盘子为空，则有 $\text{map}[m][n-1]$ 种放法。

2、 n 个盘子都装有苹果，放置步骤如下：

(1) 将 n 个苹果分别放到 n 个盘， $\text{map}[n][n]$ 中放法

(2) 将剩下的 $m-n$ 个苹果放到 n 个盘子中。此种情况和把 $m-n$ 个苹果放 n 个盘子里面相同，有 $\text{map}[m-n][n]$ 种放法。

则此部分的状态转移方程：

if ($i \geq j$)

$\text{map}[i][j] = \text{map}[i][j-1] + \text{map}[i-j][j]$

if ($i < j$)

$\text{map}[i][j] = \text{map}[i][i];$

- 递归实现(2ms)

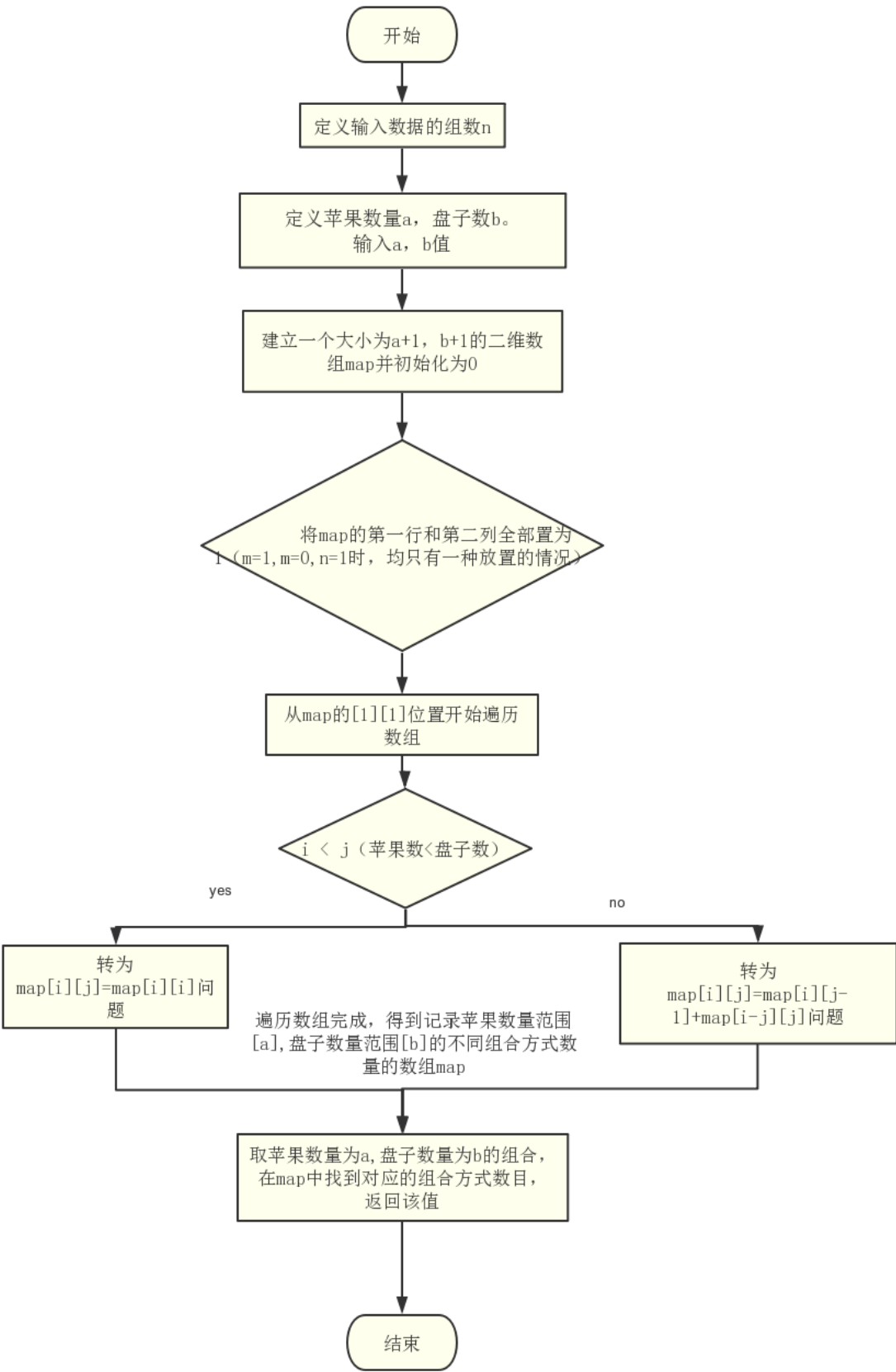
从题目中可以得出如下结论：

(1) 当 $m=1, m=0, n=1$ 时，均只有一种放置的情况

(2) $m < n$: 可以只考虑 m 个苹果， m 个盘子的情况，则为 $\text{map}[m][m]$ 的情况

(3) $m \geq n$: 则成为了上述的第二种情况

3、流程图



4、程序实现

动态规划:

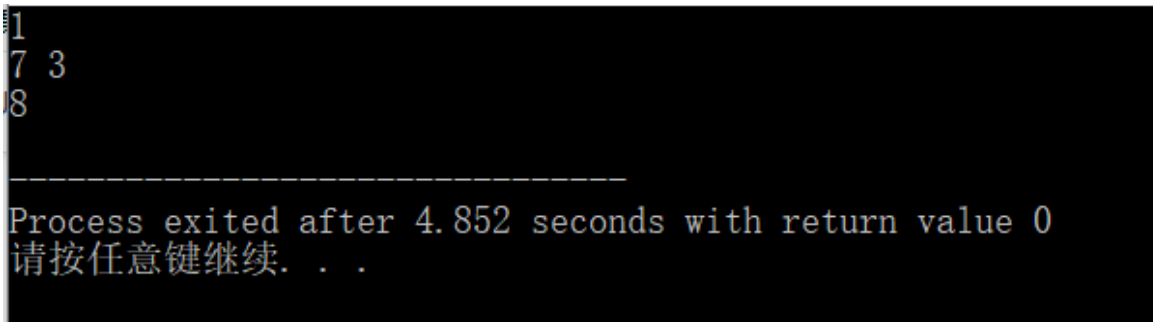
```
#include<iostream>
using namespace std;
int main()
{
    int n;
    int a,b;
    cin>>n;
    for(int k=0;k<n;k++){

        cin>>a>>b;
        int map[a+1][b+1]={0};
        for(int i=0;i<b+1;i++)
            map[0][i]=1;
        for(int i=1;i<a+1;i++)
            map[i][1]=1;
        for(int i=1;i<a+1;i++){
            for(int j=1;j<b+1;j++){
                if(i<j)
                    map[i][j]=map[i][i];
                else
                    map[i][j]=map[i][j-1]+map[i-1][j];
            }
        }
        cout<<map[a][b]<<endl;
    }
}
```

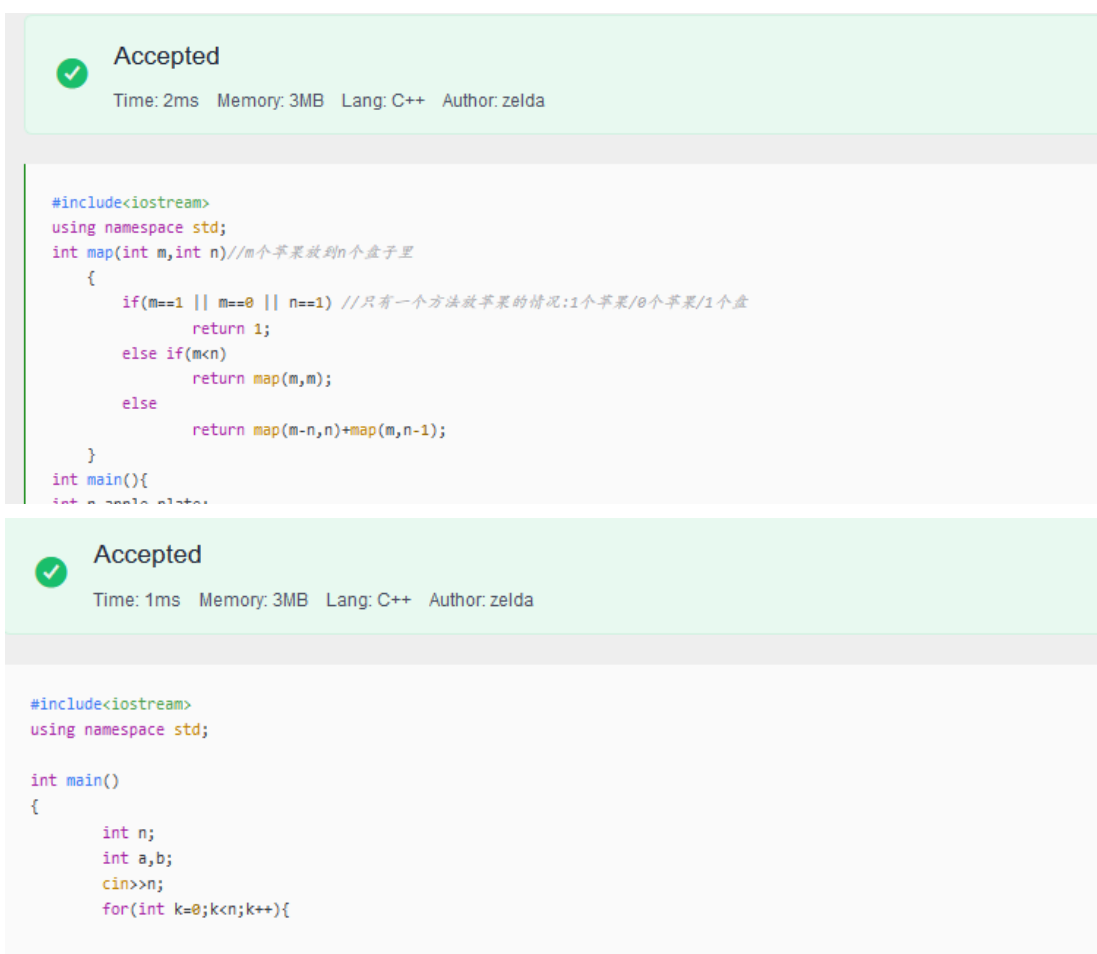
递归做法:

```
#include<iostream>
using namespace std;
int map(int m,int n)//m个苹果放到n个盘子里
{
    if(m==1 || m==0 || n==1) //只有一个方法放苹果的情况:1个苹果/0个苹果/1个盘
        return 1;
    else if(m<n)
        return map(m,m);
    else
        return map(m-n,n)+map(m,n-1);
}
int main() {
    int n,apple,plate;
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>apple>>plate;
        int all=map(apple,plate);
        cout<<all<<endl;
    }
    return 0;
}
```

5、结果分析



```
1
7 3
8
-----
Process exited after 4.852 seconds with return value 0
请按任意键继续. . .
```



6、总结思考

- 此题用动态规划和递归都可以解出来，但在思路我觉得递归的方法更容易理解
- 动态规划（dynamic programming）在解决这类问题时有着巨大的优势，应该重点掌握，比较经典的题还有计算棋盘格子中左上到右下的行走路径

不吉利日期

1、问题描述

在国外，每月的 13 号和每周的星期 5 都是不吉利的。特别是当 13 号那天恰好是星期 5 时，更不吉利。已知某年的一月一日是星期 w ，并且这一年一定不是闰年，求出这一年所有 13 号那天是星期 5 的月份，按从小到大的顺序输出月份数字。(w=1..7)

Input

输入有一行，即一月一日星期几 (w)。(1 ≤ w ≤ 7)

Output

输出有一到多行，每行一个月份，表示该月的 13 日是星期五。

Sample Input 1

7

Sample Output 1

1
10

Hint

1、3、5、7、8、10、12 月各有 31 天 4、6、9、11 月各有 30 天 2 月有 28 天

2、问题分析

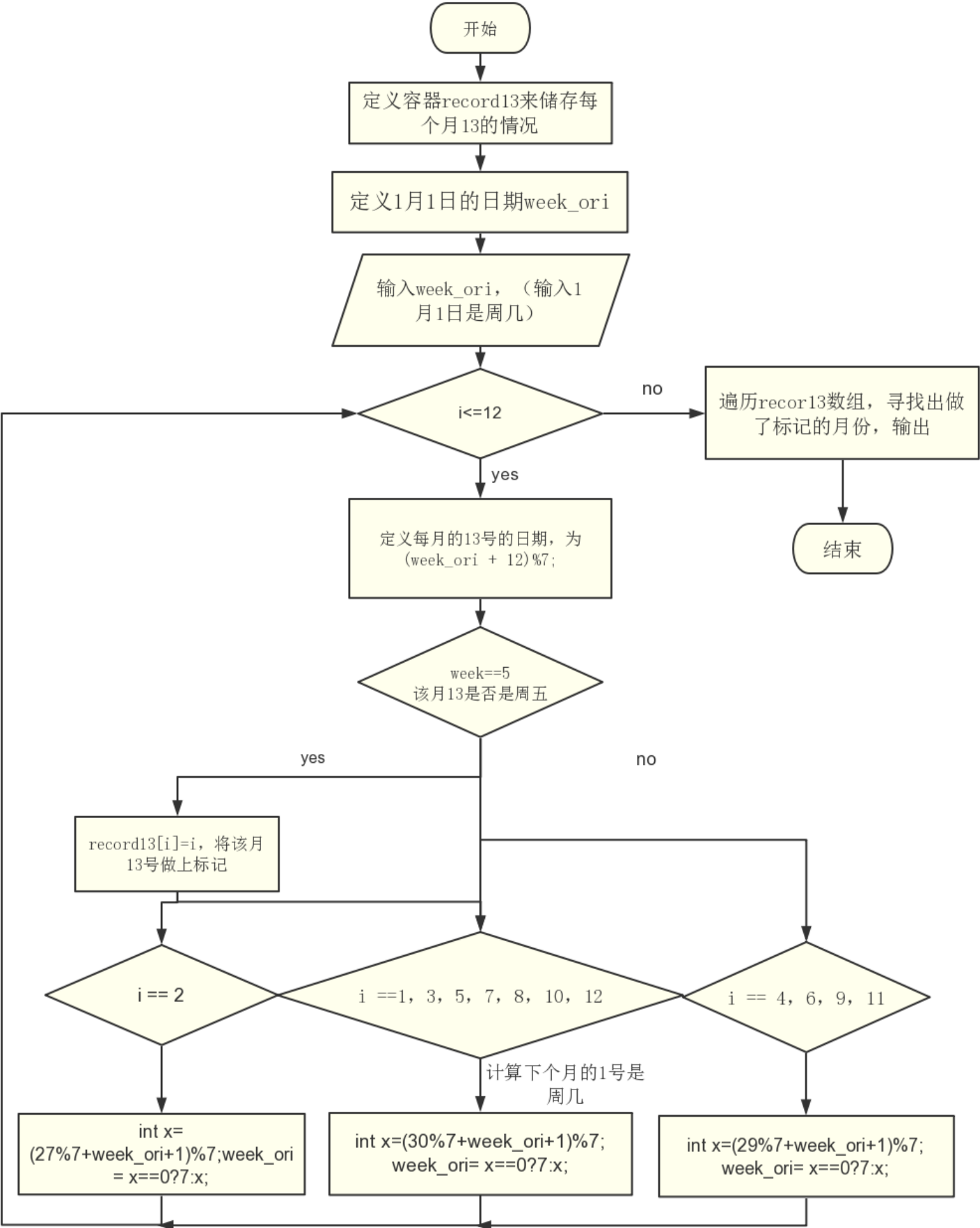
(1) 日期的推算：

此题进行简单计算即可。设每个月的第一天时间是 $week_ori$ ($week_ori$ 是周一到周日的某一天)，可以推算出该月的 13 号是否是周五，根据每个月的天数的情况计算出下个月的第一天的起始时间 $week_ori$ ，然后再判断下个月的 13 号的情况，如此循环下去直到计算完 12 个月。

(2) 日期的记录：

可以先建立一个大小为 13 的 $vector$ 容器用来储存每个月的 13 号是否是周五的情况，将 $vector$ 的元素先全部初始化为 0，然后从第 2 个位置开始往后记录（这样数组的序号直接对应月份比较方便），若该月 13 号为周五，则对应 $vector$ 位置的元素为 1，最后遍历 $vector$ 数组，输出元素为 1 的序号

3、流程图



4、程序实现 (2ms)

```
#include<iostream>
#include<vector>
using namespace std;
int record13[13]={0};
int main() {
    int week_ori;
    cin>>week_ori;
    for(int i=1;i<=12;i++) {
        int week= (week_ori + 12)%7;
        if(week==5)
            record13[i]=i;
        if(i==1||i==3||i==5||i==7||i==8||i==10||i==12)
        {
            int x=(30%7+week_ori+1)%7;//计算下一个月的1号是周几
            week_ori= x==0?7:x;
        }
        else if(i==2) {
            int x=(27%7+week_ori+1)%7;
            week_ori= x==0?7:x;
        }
        else if(i==4||i==6||i==9||i==11) {
            int x=(29%7+week_ori+1)%7;
            week_ori= x==0?7:x;
        }
    }
    for(int i=1;i<=12;i++)
    {
        if(record13[i]!=0)
            cout<<record13[i]<<endl;
    }
}
```

5、结果分析

```
7
1
10
```

```
-----
Process exited after 1.929 seconds with return value 0
请按任意键继续. . .
```



Accepted

Time: 2ms Memory: 3MB Lang: C++ Author: zelda

```
#include<iostream>
#include<vector>
using namespace std;
int record13[13]={0};

int main(){
    int week_ori;
    cin>>week_ori;
    for(int i=1;i<=12;i++){
        int week= (week_ori + 12)%7;
        if(week==5)
            record13[i]=i;
        if(i==1||i==3||i==5||i==7||i==8||i==10||i==12)
        {
            int x=(30%7+week_ori+1)%7;
            week_ori= x==0?7:x;
        }
        else if(i==2){
            int x=(27%7+week_ori+1)%7;
            week_ori= x==0?7:x;
        }
        else if(i==4||i==6||i==9||i==11){
            int x=(29%7+week_ori+1)%7;
            week_ori= x==0?7:x;
        }
    }
    for(int i=1;i<=12;i++)
    {
        if(record13[i]!=0)
            cout<<record13[i]<<endl;
    }
}
```

6、总结思考

这道题通过上面的步骤做起来比较繁琐，特别是在判断本月是几月份的时候。另有一种做法如下：

可以先初始化一个容器记录下 12 个月对应的天数，然后遍历该容器，寻找当前月的 13 号为周五的月份，然后在容器相应位置做出标记即可。

———魔兽世界一： 备战——— ———魔兽战斗一： 开始———

1、问题描述

魔兽世界的西面是红魔军的司令部，东面是蓝魔军的司令部。两个司令部之间是依次排列的若干城市。红司令部，City 1, City 2, ……，City n, 蓝司令部

两军的司令部都会制造武士。武士一共有 dragon、ninja、iceman、lion、wolf 五种。每种武士都有编号、生命值、攻击力这三种属性。

双方的武士编号都是从 1 开始计算。红方制造出来的第 n 个武士，编号就是 n。同样，蓝方制造出来的第 n 个武士，编号也是 n。

武士在刚降生的时候有一个生命值。

在每个整点，双方的司令部中各有一个武士降生。

红方司令部按照 iceman、lion、wolf、ninja、dragon 的顺序循环制造武士。

蓝方司令部按照 lion、dragon、ninja、iceman、wolf 的顺序循环制造武士。

制造武士需要生命元。

制造一个初始生命值为 m 的武士，司令部中的生命元就要减少 m 个。

如果司令部中的生命元不足以制造某个按顺序应该制造的武士，那么司令部就试图制造下一个。如果所有武士都不能制造了，则司令部停止制造武士。

给定一个时间，和双方司令部的初始生命元数目，要求你将从 0 点 0 分开始到双方司令部停止制造武士为止的所有事件按顺序输出。一共有两种事件，其对应的输出样例如下：

1) 武士降生输出样例：004 blue lion 5 born with strength 5,2 lion in red headquarter 表示在 4 点整，编号为 5 的蓝魔 lion 武士降生，它降生时生命值为 5，降生后蓝魔司令部里共有 2 个 lion 武士。（为简单起见，不考虑单词的复数形式）注意，每制造出一个新的武士，都要输出此时司令部里共有多少个该种武士。

2) 司令部停止制造武士输出样例：010 red headquarter stops making warriors 表示在 10 点整，红方司令部停止制造武士

输出事件时：

首先按时间顺序输出；

同一时间发生的事件，先输出红司令部的，再输出蓝司令部的。

Input

第一行是一个整数，代表测试数据组数。

每组测试数据共两行。

第一行：一个整数 M。其含义为，每个司令部一开始都有 M 个生命元 ($1 \leq M \leq 10000$)。

第二行：五个整数，依次是 dragon、ninja、iceman、lion、wolf 的初始生命值。它们都大于 0 小于等于 10000。

Output

对每组测试数据，要求输出从 0 时 0 分开始，到双方司令部都停止制造武士为止的所有事件。

对每组测试数据，首先输出"Case:n" n 是测试数据的编号，从 1 开始。

接下来按恰当的顺序和格式输出所有事件。

每个事件都以事件发生的时间开头，时间以小时为单位，有三位。

Sample Input 1

```
1
20
3 4 5 6 7
```

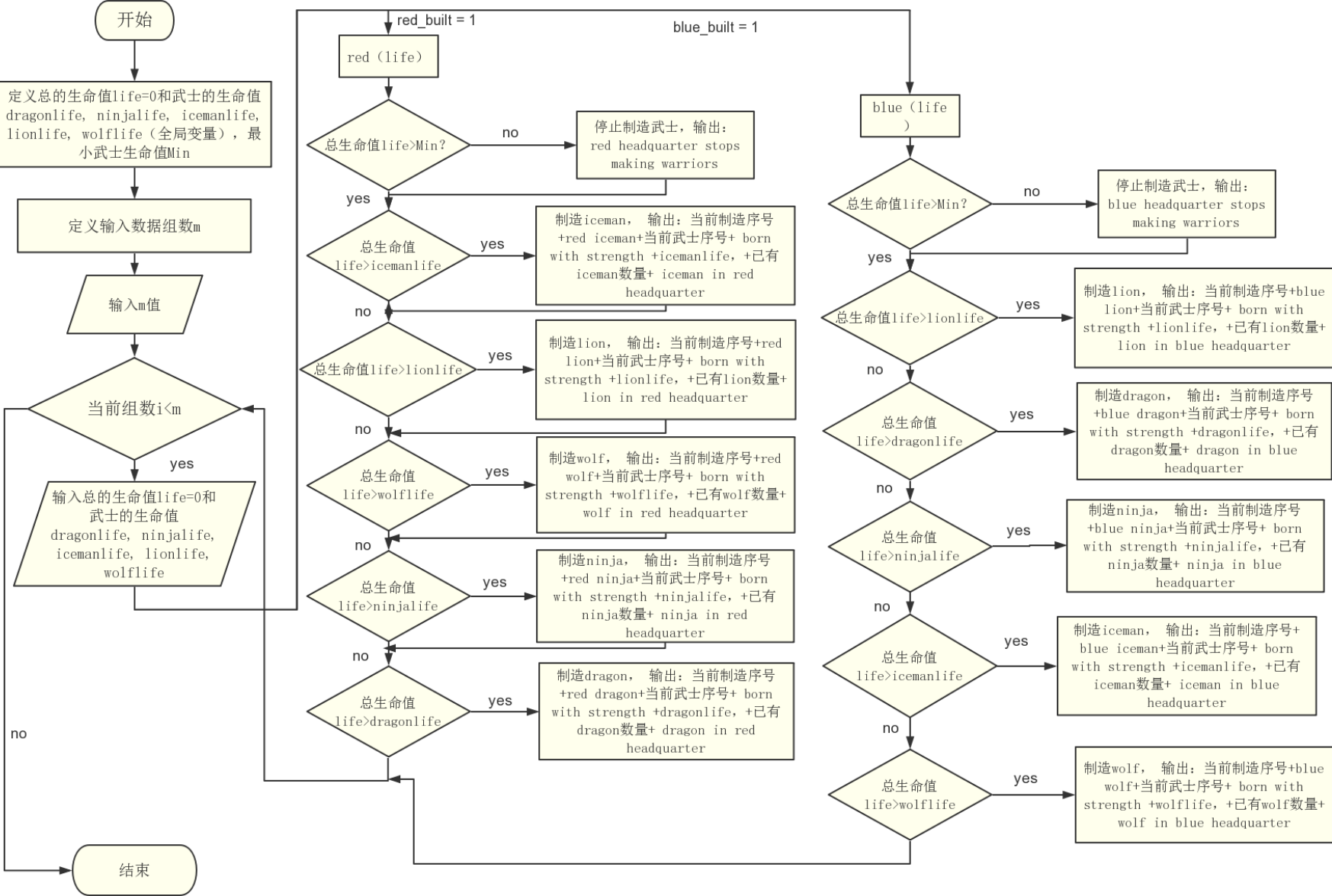
Sample Output 1

```
Case:1
000 red iceman 1 born with strength 5,1 iceman in red headquarter
000 blue lion 1 born with strength 6,1 lion in blue headquarter
001 red lion 2 born with strength 6,1 lion in red headquarter
001 blue dragon 2 born with strength 3,1 dragon in blue headquarter
002 red wolf 3 born with strength 7,1 wolf in red headquarter
002 blue ninja 3 born with strength 4,1 ninja in blue headquarter
003 red headquarter stops making warriors
003 blue iceman 4 born with strength 5,1 iceman in blue headquarter
004 blue headquarter stops making warriors
```

2、问题分析

由于本题的红蓝两方在制造武士时是相互独立的两个部分，因此此题可以完全拆分成 red 和 blue 两个部分来做，红蓝双方各自按照自己的方法和顺序制造武士，只需要设定一个 bool 型的 red_built 和 blue_built 来控制当前该是哪一方制造武士，并输出相应的武士信息即可

3、流程图（见下图）



4、程序实现

```
#include<iostream>
#include<iomanip>
#define min(a,b) (a<b?a:b)
using namespace std;
int life=0, dragonlife, ninjalife,
icemanlife, lionlife, wolflife;
int Min;
class soldier {
public:
    soldier() {}
    int number;
int life=0;
    soldier(int n):number(n) {}
};
```

```
class dragon :public soldier {
public:
    dragon(int n) :soldier(n) {
    };
};
class ninja :public soldier {
public:
    ninja(int n) :soldier(n) {
    };
};
class iceman :public soldier {
public:
    iceman(int n) :soldier(n) {}
};
```

```
class lion :public soldier {
public:
    lion(int n) :soldier(n) {
    };
};
class wolf :public soldier {
public:
    wolf(int n) :soldier(n) {}
};
```

```

//-----RED-----
//红方制造武士
class red :public soldier {
public:
    int m, n = 0, dragonnumber = 0, ninjanumber =0, icemannumber
=0, lionnumber =0, wolfnnumber =0;
    int num = 0;
    red(int c) :m(c) {}

    int judge() {
        if (m < Min) {
            cout << "red headquarter stops making warriors" <<
endl;

            return 1;
        }
        if (num % 5 == 0 && m >= icemanlife) {
            build(0);
            return 0;
        }
        if (num % 5 == 1 && m >= lionlife) {
            build(1);
            return 0;
        }
        if (num % 5 == 2 && m >= wolflife) {
            build(2);
            return 0;
        }
        if (num % 5 == 3 && m >= ninjalife) {
            build(3);
            return 0;
        }
        if (num % 5 == 4 && m >= dragonlife) {
            build(4);
            return 0;
        }
    }
};
else {
    num++;
    judge();
}
}

int build(int d) {
    num++; n++;
    if (d == 0) {
        m = m - icemanlife;
        icemannumber++;
        cout << "red iceman " << n << " born with strength " <<
icemanlife << ", " << icemannumber << " iceman in red headquarter" <<
endl;

        iceman a(n);
    }
    if (d == 1) {
        m = m - lionlife;
        lionnumber++;
        cout << "red lion " << n << " born with strength " <<
lionlife << ", " << lionnumber << " lion in red headquarter" << endl;
        lion a(n);
    }
    if (d == 2) {
        m = m - wolflife;
        wolfnnumber++;
        cout << "red wolf " << n << " born with strength " <<
wolflife << ", " << wolfnnumber << " wolf in red headquarter" << endl;
        wolf a(n);
    }
    if (d == 3) {
        m = m - ninjalife;
        ninjanumber++;
        cout << "red ninja " << n << " born with strength " <<
ninjalife << ", " << ninjanumber << " ninja in red headquarter" <<
endl;

        iceman a(n);
    }
    if (d == 4) {
        m = m - dragonlife;
        dragonnumber++;
        cout << "red dragon " << n << " born with strength " <<
dragonlife << ", " << dragonnumber << " dragon in red headquarter" <<
endl;

        dragon a(n);
    }
}

```

```
//-----BLUE-----
//蓝方制造武士
class blue :public soldier {
public:
    int m, n = 0, dragonnumber = 0, ninjanumber = 0,
    icemannumber = 0, lionnumber = 0, wolfnnumber = 0;
    int num = 0;
    blue(int c) :m(c) {};
    int build() {};
    int judge() {
        if (m < Min) {
            cout << "blue headquarter stops making
warriors" << endl;
            return 1;
        }
        if (num % 5 == 0 && m >= lionlife) {
            build(0);
            return 0;
        }
        if (num % 5 == 1 && m >= dragonlife) {
            build(1);
            return 0;
        }
        if (num % 5 == 2 && m >= ninjalife) {
            build(2);
            return 0;
        }
        if (num % 5 == 3 && m >= icemanlife) {
            build(3);
            return 0;
        }
        if (num % 5 == 4 && m >= wolflife) {
            build(4);
            return 0;
        }
        else {
            num++;
            judge();
        }
    }
}
```

```
int build(int d) {
    num++; n++;
    if (d == 0) {
        m = m - lionlife;
        lionnumber++;
        cout << "blue lion " << n << " born with strength " << lionlife << ", "
        << lionnumber << " lion in blue headquarter" << endl;
        lion a(n);
    }
    if (d == 1) {
        m = m - dragonlife;
        dragonnumber++;
        cout << "blue dragon " << n << " born with strength "
        << dragonlife << ", " << dragonnumber << " dragon in blue headquarter" <<
        endl;
        dragon a(n);
    }
    if(d==2){
        m = m - ninjalife;
        ninjanumber++;
        cout << "blue ninja " << n << " born with strength " <<
        ninjalife << ", " << ninjanumber << " ninja in blue headquarter" << endl;
        ninja a(n);
    }
    if (d == 3) {
        m = m - icemanlife;
        icemannumber++;
        cout << "blue iceman " << n << " born with strength "
        << icemanlife << ", " << icemannumber << " iceman in blue headquarter" <<
        endl;
        iceman a(n);
    }
    if (d == 4) {
        m = m - wolflife;
        wolfnnumber++;
        cout << "blue wolf " << n << " born with strength " <<
        wolflife << ", " << wolfnnumber << " wolf in blue headquarter" << endl;
        wolf a(n);
    }
}
};
```

```

int main()//主函数
{
    int m;
    int recent = 1;
    cin>>m;
    for(int i=1;i<=m;i++) {
        int time = 0;
        cin >> life >> dragonlife >> ninjalife >> icemanlife >> lionlife >> wolflife;
        red redl(life);
        blue blue1(life);
        bool red_built = 1, blue_built = 1;
        Min = min(min(dragonlife, ninjalife), min(min(icemanlife, lionlife), wolflife));
        cout << "Case:" <<i<< endl;

        while (1) {
if (red_built) {
    cout << setw(3) << setfill('0') << time << " ";
    red_built= 1 - redl.judge();
        }

        if (blue_built) {
    cout << setw(3) << setfill('0') << time << " ";
    blue_built = 1 - blue1.judge();}

        if (red_built + blue_built == 0)
            break;
            time++;
        }
    }
    return 0;
}

```

5、结果分析

```

1
20
3 4 5 6 7
Case:1
000 red iceman 1 born with strength 5,1 iceman in red headquarter
000 blue lion 1 born with strength 6,1 lion in blue headquarter
001 red lion 2 born with strength 6,1 lion in red headquarter
001 blue dragon 2 born with strength 3,1 dragon in blue headquarter
002 red wolf 3 born with strength 7,1 wolf in red headquarter
002 blue ninja 3 born with strength 4,1 ninja in blue headquarter
003 red headquarter stops making warriors
003 blue iceman 4 born with strength 5,1 iceman in blue headquarter
004 blue headquarter stops making warriors

-----
Process exited after 6.793 seconds with return value 0
请按任意键继续. . .

```


**Accepted**

Time: 2ms Memory: 3MB Lang: C++ Author: zelda

```
#include<iostream>
#include<iomanip>
#define min(a,b)(a<b?a:b)
using namespace std;
int life=0, dragonlife, ninjalife, icemanlife, lionlife, wolflife;
int Min;
class soldier {
public:
    soldier(){}
    int number;
    int life=0;
```

6、总结思考

这道题难度倒不是特别大，只是要注意很多细节，比如红蓝方制造武士时交替进行，两方造武士的顺序不一样等等。同时由于题目太长，在编写算法时要注意审题。

注：流程图可能比较模糊，pdf 格式已上传至 GitHub

<https://github.com/ZeldaM1/programing-practice-2018>