第十三周实验报告

沈家成

3022 二哥要翘课

动态规划

根据题目描述，连续三天翘课才会被教务处发现，也就是不能有连续三天翘课。

那么，从动态规划的角度，决定每天翘不翘课的因素，就只有前两天有没有翘课。

如果前两天已经翘课了，那么无论如何都不能再翘课了，必须上课。如果前面两天没有都翘课，就意味着这天可翘可不翘，就有两个分支。

用 0 表示翘课，用 1 表示上课，则前两天的情况可以总结成如下表格。

前两天翘课情况	数字表示
第一天翘课，第二天翘课	0 0
第一天翘课，第二天上课	0 1
第一天上课，第二天翘课	1 0
第一天上课，第二天上课	1 1

用第三天到第四天的决策举例：

第三天	第四天
~~000~~	~~0001~~
001	0011\0010
010	0101\0100
011	0111\0110
100	1001\ ~~1000~~
101	1011\1010
110	1101\1100
111	1111\1110

划线的表示被教务处发现了

统计一下这两天安全的情况中最后两位的情况：

第三天

00	**01**	**10**	**11**
1	2	2	2

第四天

00	01	10	11
2	3	4	4

统计的时候可以发现，第四天的分布与第三天有紧密的联系。

用$A_{00}^n, A_{01}^n, A_{10}^n, A_{11}^n$ 分布表示第 n 天时各种情况的分布，就可以得到如下的递推关系：

$$A_{00}^{n+1} = A_{10}^n$$

$$A_{01}^{n+1} = A_{00}^n + A_{10}^n$$

$$A_{10}^{n+1} = A_{01}^n + A_{11}^n$$

$$A_{11}^{n+1} = A_{01}^n + A_{11}^n$$

因此，想要知道第 N 天有多少种不被抓住的方式，就是$A_{00}^N + A_{01}^N + A_{10}^N + A_{11}^N$

unsigned long long 都不够表示的大数

题目中有提示，最终结果会很大，原本以为 unsigned long long 就已经足够了，但是测试 10000 天的情况时，还是发生了溢出，因此需要自己设计一个适应的大数。

设计了一个叫做 BigInt 的类，最主要的是加法函数的设计。用运算符重载的确可以提高通用性，但是因为还要用到赋值运算符，肯定会有性能的牺牲，因此最终还是选择直接用一个 add 函数完成这个功能。

```
void add(BigInt & add1, BigInt & add2)
{
    int carry = 0;
    for (int i = 0; i < max_size; ++i) {
        data[i] = add1.data[i] + add2.data[i] + carry;
        if ((carry = data[i] / 10))
            data[i] %= 10;
    }
}
```

使用一个 carry 变量储存进位。如果产生了进位，carry会储存进位数，同时会对该位数字取余，这样就完成了加法的操作。

最终进行测试，10000 天的结果是一个 2500+ 位数，的确不是 unsigned long long 能够表示的。

1994 二哥的地图

极大连通域

题目要求我们求出最多有多少个国家。为了国家数最多，就是需要每一块单独的陆地只属于一个国家，那么问题就转换成了求有多少个极大连通域。

图的遍历

如何求极大连通域的个数呢？如果把每个坐标看作结点，那么陆地相邻就是结点直接有边相接，求极大连通域的个数就转换成了求图遍历的森林有多少棵树。

因此，就是设计一个相适应的遍历算法。

```
void search(int x, int y)
{
    map[x][y] = false;
    if (map[x][y-1])
        search(x, y-1);
    if (map[x][y+1])
        search(x, y+1);
    if (map[x-1][y])
```

```
        search(x-1, y);
    if (map[x+1][y])
        search(x+1, y);
}
```

利用递归的方法，搜索所有与点(x, y)相邻的点，并置为已搜索，就找出了一个极大连通域。

```
for (int i = 1; i <= N; ++i) {
    for (int j = 1; j <= M; ++j) {
        if (map[i][j]) {
            ++ctr;
            search(i, j);
        }
    }
}
```

主函数中，遍历所有点，确保所有点都已经访问。ctr为计数器，每次找到一个连通域就累加。循环结束，ctr中的值就是极大连通域的个数。