基础算法-位运算、递推与递归

一、AcWing 90. 64位整数乘法

【题目描述】

求a乘b对p取模的值。

【输入格式】

第一行输入整数a，第二行输入整数b，第三行输入整数p。

【输出格式】

输出一个整数，表示$a * b \ mod \ p$的值。

【数据范围】

$1 \leq a,b,p \leq 10^{18}$

【输入样例】

```
3
4
5
```

【输出样例】

```
2
```

【分析】

直接乘会爆精度，将a加b次会爆时间，因此需要使用快速乘（龟速乘）。

将b看成二进制数，例如：$(11010)_B$，则$a * b \iff a * 2^1 + a * 2^3 + a * 2^4$，因此我们在从小到大枚举$b$的每一位时，不断将$a$乘$2$，当$b$的某一位为$1$时，答案加上当前的$a$即可。

【代码】

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

```

```cpp
typedef long long LL;

LL qmul(LL a, LL b, LL p)
{
    LL res = 0;
    while (b)
    {
        if (b & 1) res = (res + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return res;
}

int main()
{
    LL a, b, p;
    cin >> a >> b >> p;
    cout << qmul(a, b, p) << endl;
    return 0;
}
```

二、AcWing 95. 费解的开关

【题目描述】

你玩过“拉灯”游戏吗？

25盏灯排成一个5×5的方形。

每一个灯都有一个开关，游戏者可以改变它的状态。

每一步，游戏者可以改变某一个灯的状态。

游戏者改变一个灯的状态会产生连锁反应：和这个灯上下左右相邻的灯也要相应地改变其状态。

我们用数字1表示一盏开着的灯，用数字0表示关着的灯。

例如下面这种状态：

```
10111
01101
10111
10000
11011
```

在改变了最左上角的灯的状态后将变成：

```
01111
11101
10111
10000
11011
```

再改变它正中间的灯后状态将变成：

```
01111
11001
11001
10100
11011
```

给定一些游戏的初始状态，编写程序判断游戏者是否可能在6步以内使所有的灯都变亮。

【输入格式】

第一行输入正整数n，代表数据中共有n个待解决的游戏初始状态。

以下若干行数据分为n组，每组数据有5行，每行5个字符。

每组数据描述了一个游戏的初始状态。

各组数据间用一个空行分隔。

【输出格式】

一共输出n行数据，每行有一个小于等于6的整数，它表示对于输入数据中对应的游戏状态最少需要几步才能使所有灯变亮。

对于某一个游戏初始状态，若6步以内无法使所有灯变亮，则输出-1。

【数据范围】

$0 < n \leq 500$

【输入样例】

```
3
00111
01011
10001
11010
11100

11101
11101
11110
11111
11111

01111
11111
11111
11111
11111
```

【输出样例】

```
3
2
-1
```

【分析】

本题有两种解题思路：

1. 使用BFS将终点状态（全为1）反推6步所能到达的所有状态搜索出来，然后根据每次输入的状态直接查表判断是否合法即可。
2. 枚举第一行所有开关的全部可能的状态，可以使用一个二进制数$state$，它的第i位表示第i个开关是否按下。当第一行的状态确定了，我们从第二行开始逐行枚举每一个开关，如果当前开关(i,j)的上一个开关状态是`0`，即$g[i-1][j]==='0'$，那么当前开关一定要按，因为是逐行枚举的，上一行的状态已经是确定的了，也就是无法再按了，只有当前开关能改变上一行开关的状态。那么我们枚举完后面四行时，第$1\sim4$行一定已经全为`1`了，这时候我们枚举最后一行的每个开关，如果每个开关的状态都为`1`，那么用记录下的操作次数去更新最终答案。最后判断最终答案是否小于6即可。

方法一的时间复杂度难以计算，因此选用第二种方法。

PS：字符`0`（ASCII码为48）与字符`1`（ASCII码为49）相互转换的方式为`^=1`即可，因为二进制表示中只有最后一位不同。

【代码】

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

const int N = 10;
char g[N][N], backup[N][N];
int n;
int dx[5] = { -1, 0, 1, 0, 0 }, dy[5] = { 0, 1, 0, -1, 0 };

void op(int x, int y)//将开关(x,y)按下
{
    for (int i = 0; i < 5; i++)
    {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < 5 && ny >= 0 && ny < 5) g[nx][ny] ^= 1;
    }
}

int main()
{
    cin >> n;
    while (n--)
    {
        int res = 10;//答案,初值取一个大于6的数即可
        for (int i = 0; i < 5; i++) cin >> backup[i];
        for (int state = 0; state < 32; state++)
        {
            memcpy(g, backup, sizeof backup);
            int cnt = 0;//操作次数
            //根据当前状态操作第0行的开关
            for (int i = 0; i < 5; i++)
                if (state >> i & 1) op(0, i), cnt++;
            //根据第0行的状态递推第1~4行的状态
            for (int i = 1; i < 5; i++)
                for (int j = 0; j < 5; j++)

                    if (g[i - 1][j] == '0') op(i, j), cnt++;
```

```
            //递推完之后前四行一定全为1,检验最后一行是否全为1,若是则更新答案
            for (int i = 0; i < 5; i++)
                if (g[4][i] == '0') break;
                else if (i == 4) res = min(res, cnt);
        }
        if (res > 6) cout << -1 << endl;
        else cout << res << endl;
    }
    return 0;
}
```

三、AcWing 97. 约数之和

【题目描述】

假设现在有两个自然数A和B，S是A^B的所有约数之和。

请你求出$S \bmod 9901$的值是多少。

【输入格式】

在一行中输入用空格隔开的两个整数A和B。

【输出格式】

输出一个整数，代表$S \bmod 9901$的值。

【数据范围】

$0 \le A,B \le 5 \times 10^7$

【输入样例】

```
2 3
```

【输出样例】

```
15
```

注意：A和B不会同时为0。

【分析】

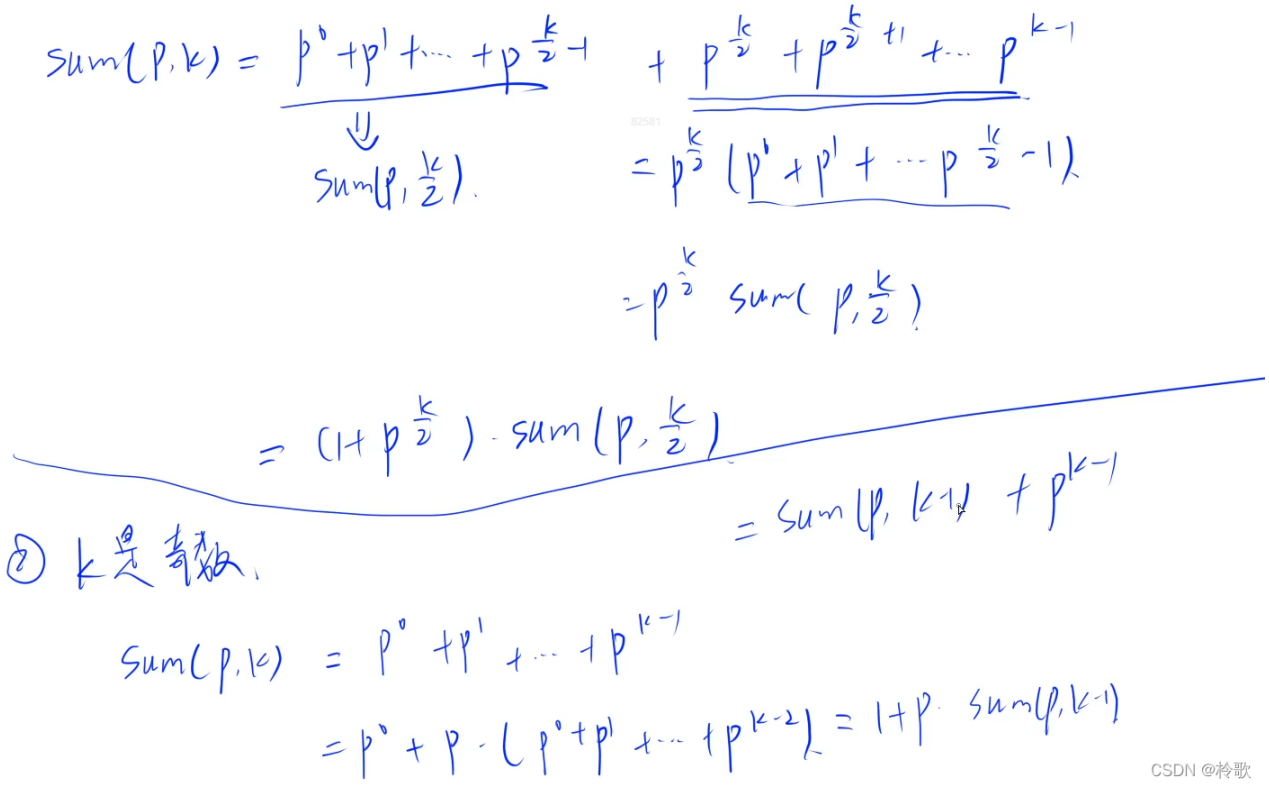
我们将A分解质因数后为：$A = p_1^{\alpha_1} * p_2^{\alpha_2} * \ldots * p_k^{\alpha_k}$

则$A^B = p_1^{\alpha_1 B} * p_2^{\alpha_2 B} * \ldots * p_k^{\alpha_k B}$

因此约数之和为：$(p_1^0+p_1^1+\cdots+p_1^{\alpha_1 B})*\ \ldots\ *(p_k^0+p_k^1+\cdots+p_k^{\alpha_k B})$

如果直接累加那么一定会爆时间复杂度，因此我们设$sum(p,k)=p^0+p^1+\cdots+p^{k-1}$，则原式为：$sum(p_1,\alpha_1 B+1)*\cdots*sum(p_k,\alpha_k B+1)$。那么如何快速求解$sum(p,k)$呢？可以使用递归分治的思想：

- 当k为偶数时，总共有偶数个项，那么我们可以分成两半，前半部分为$p^0+p^1+\cdots+p^{\frac{k}{2}-1}=sum(p,\frac{k}{2})$，后半部分为$p^{\frac{k}{2}}+p^{\frac{k}{2}+1}+\cdots+p^{k-1}=p^{\frac{k}{2}}(p^0+p^1+\cdots+p^{\frac{k}{2}-1})=p^{\frac{k}{2}}sum(p,\frac{k}{2})$。因此$sum(p,k)=(1+p^{\frac{k}{2}})*sum(p,\frac{k}{2})$
- 当k为奇数时，总共有奇数个项，那么我们将最后一项拆出来，前$k-1$项为$p^0+p^1+\cdots+p^{k-2}=sum(p,k-1)$，最后一项为$p^{k-1}$，则原式为$sum(p,k-1)+p^{k-1}$



【代码】

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

typedef long long LL;
const int MOD = 9901;

LL qmi(LL a, LL b)

{
```

```
    LL res = 1;
    while (b)
    {
        if (b & 1) res = res * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return res;
}

//计算p^0+p^1+...+p^(k-1)
LL sum(LL p, LL k)
{
    if (k == 1) return 1;//只有p^0的情况
    else if (k % 2 == 0)//项数为偶数的时候可以分成两部分
        return (1 + qmi(p, k / 2)) * sum(p, k / 2) % MOD;
    return (sum(p, k - 1) + qmi(p, k - 1)) % MOD;//为奇数就把最后一项单拿出来加
}

int main()
{
    LL a, b;
    cin >> a >> b;
    LL res = 1;
    for (int i = 2; i * i <= a; i++)//对a分解质因数
        if (a % i == 0)
        {
            int s = 0;//表示当前因数i的个数
            while (a % i == 0) a /= i, s++;
            res = res * sum(i, s * b + 1) % MOD;
        }
    if (a > 1) res = res * sum(a, b + 1) % MOD;//还有最后一个大于sqrt(a)的因数
    else if (!a) res = 0;//a可能为0
    cout << res << endl;
    return 0;
}
```

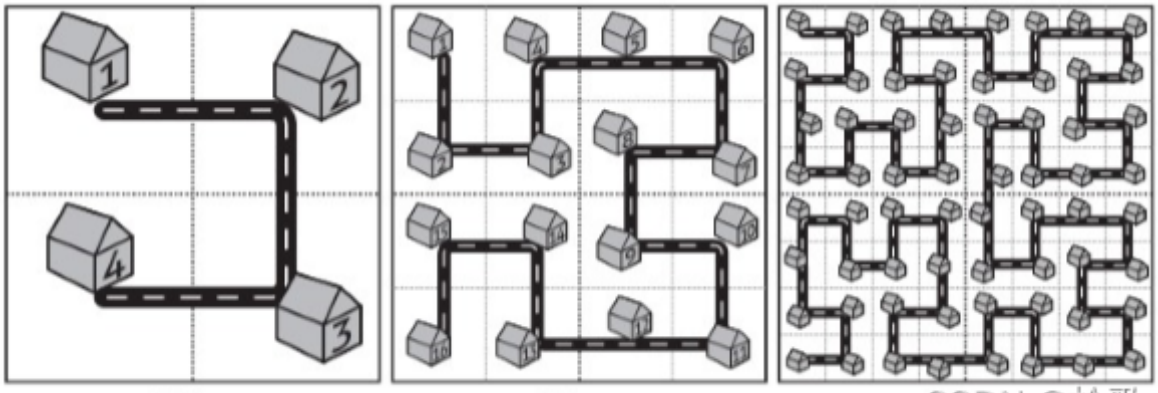
四、AcWing 98. 分形之城

【题目描述】

城市的规划在城市建设中是个大问题。

不幸的是，很多城市在开始建设的时候并没有很好的规划，城市规模扩大之后规划不合理的问题就开始显现。

而这座名为Fractal的城市设想了这样的一个规划方案，如下图所示：



当城区规模扩大之后，Fractal的解决方案是把和原来城区结构一样的区域按照图中的方式建设在城市周围，提升城市的等级。

对于任意等级的城市，我们把正方形街区从左上角开始按照道路标号。

虽然这个方案很烂，Fractal规划部门的人员还是想知道，如果城市发展到了等级N，编号为A和B的两个街区的直线距离是多少。

街区的距离指的是街区的中心点之间的距离，每个街区都是边长为10米的正方形。

【输入格式】

第一行输入正整数n，表示测试数据的数目。

以下n行，输入n组测试数据，每组一行。

每组数据包括三个整数N, A, B，表示城市等级以及两个街区的编号，整数之间用空格隔开。

【输出格式】

一共输出n行数据，每行对应一组测试数据的输出结果，结果四舍五入到整数。

【数据范围】

$1 \leq N \leq 31$

$1 \leq A, B \leq 2^{2N}$

$1 \leq n \leq 1000$

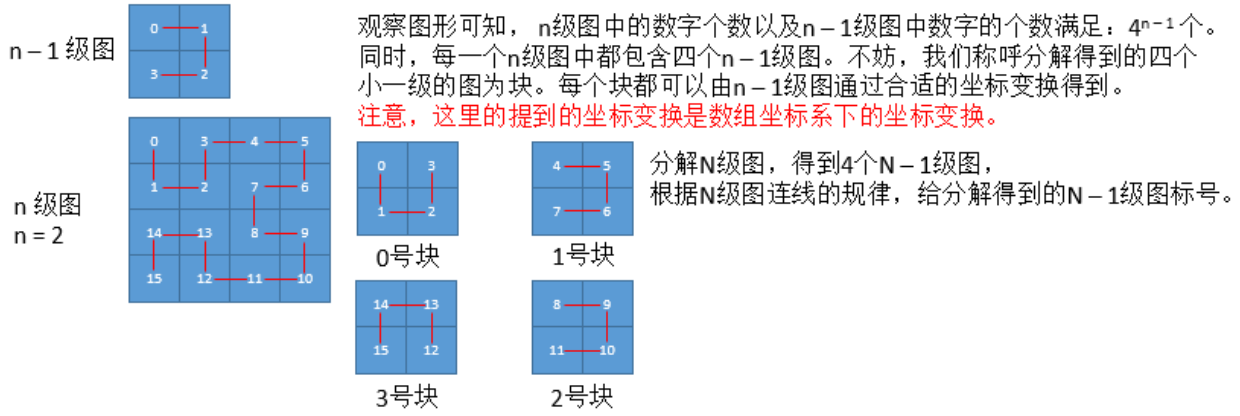
【输入样例】

```
3
1 1 2
2 16 1
3 4 33
```

【输出样例】

```
10
30
50
```

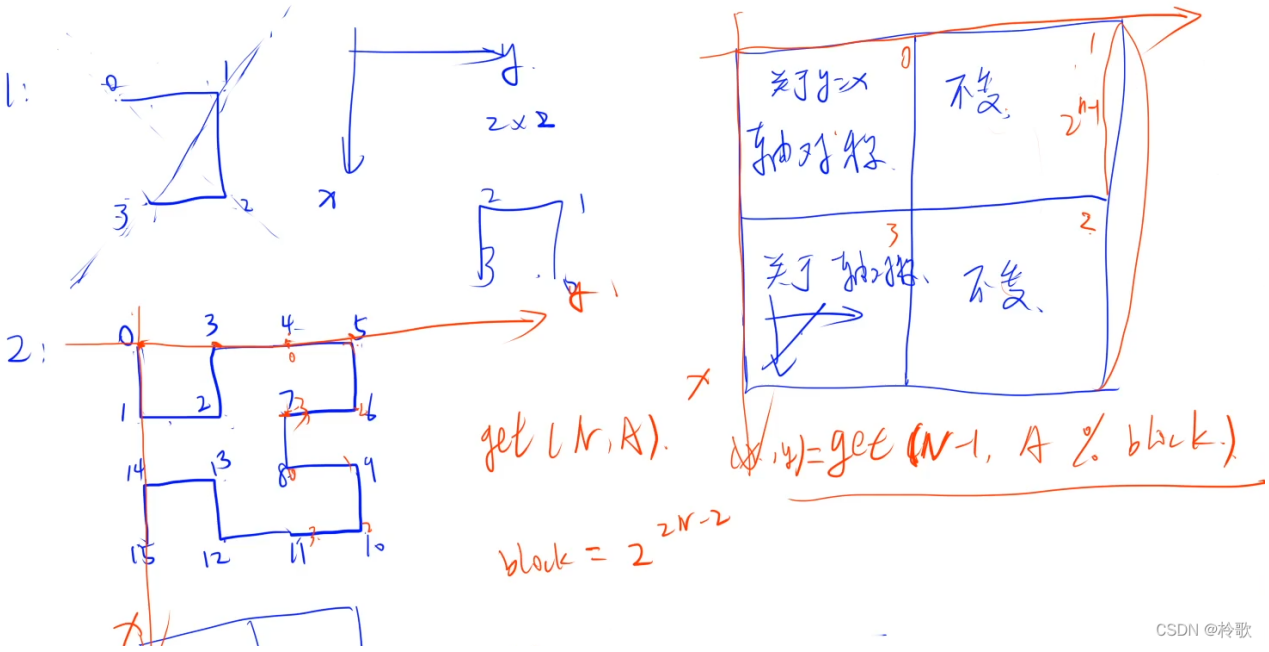
【分析】



显然，要求n级图的数字个数要先求n – 1级图中数字的个数。令n – 1级图的数字的个数为cnt。
那么有：cnt = 1 << 2 * (n – 1)。
n级图中的任意一个数字的编号为m，令块的编号为cnk。那么，数字m所在块的编号是：cnk = m / cnt。
n级图中的任意一个数字对应n – 1级图上的数字的编号是：idx = m % cnt。
因为n – 1级图会在n级图上做平移才能得到某些块。所以，要确定平移的单位长度。令这个长度为len，
那么有：len = 1 << (n – 1)。

剩下的事情就是确定，N – 1级图与N级图的每个块之间的坐标变换。

首先我们根据$N=2$与$N=1$的图进行分析：

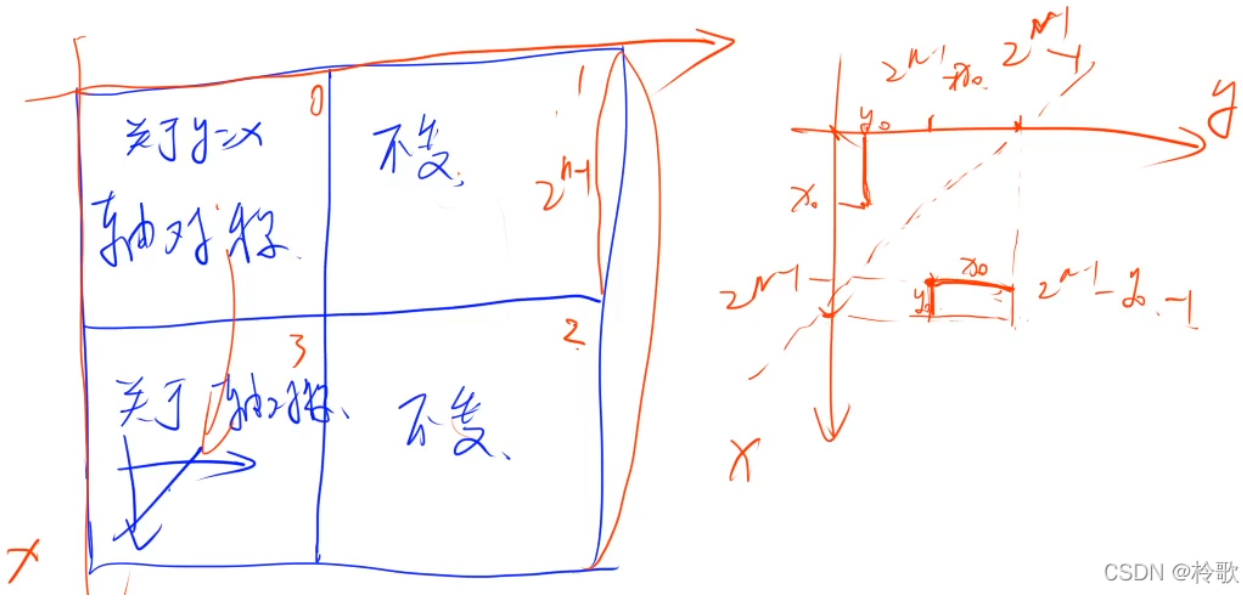


- N级图形的0号子块的图形与$N-1$级图形关于直线$y=x$轴对称，对称后没有进行平移
- N级图形的1号子块的图形是由$N-1$级图形向右平移2^{N-1}个单位得到的
- N级图形的2号子块的图形是由$N-1$级图形向右平移2^{N-1}个单位后再向下平移2^{N-1}得到的
- N级图形的3号子块的图形与$N-1$级图形关于直线$y=-x$轴对称，对称后向下平移2^{N-1}个单位

因此我们求出编号A的点所在的子块编号$block=A/cnt$与A在子块中的坐标(x,y)后，进行如下坐标变换：

- $block==0$时，$(x,y)\to(y,x)$
- $block==1$时，$(x,y)\to(x,y+2^{N-1})=(x,y+len)$
- $block==2$时，$(x,y)\to(x+2^{N-1},y+2^{N-1})=(x+len,y+len)$
- $block==3$时，$(x,y)\to(2^{N-1}-1-y+2^{N-1},2^{N-1}-1-x)=(2*len-1-y,len-1-x)$

$block==3$时可以画图辅助找规律：



【代码】

```cpp
#include <iostream>
#include <cmath>
#define X first
#define Y second
using namespace std;

typedef long long LL;
typedef pair<LL, LL> PLL;
LL n, a, b;

PLL get(LL N, LL A)
{
    if (N == 0) return { 0, 0 };
    LL cnt = (LL)1 << 2 * N - 2, len = (LL)1 << N - 1;//cnt表示子块中点的数量,len表示子块的边长
    PLL p = get(N - 1, A % cnt);//递归在子块中求解,A%cnt表示A在子块中的编号
    int block = A / cnt;//表示子块的编号
    LL x = p.X, y = p.Y;
    if (block == 0) return { y, x };
    else if (block == 1) return { x, y + len };
    else if (block == 2) return { x + len, y + len };
    return { len - 1 - y + len, len - 1 - x };
}

int main()
{
    int T;
```

```
    cin >> T;
    while (T--)
    {
        cin >> n >> a >> b;
        PLL pa = get(n, a - 1), pb = get(n, b - 1);//将1~N映射为0~N-1
        double dx = pa.X - pb.X, dy = pa.Y - pb.Y;
        printf("%.0lf\n", sqrt(dx * dx + dy * dy) * 10);//注意边长为10
    }
    return 0;
}
```