目录

1 离散化

1.1 一维线离散化

```
struct LINE{
    int l,r;
}line[SIZE*16];
int lisan[SIZE*16];
void Discrete(int N){//bl存的是编号为[1,N]的线,在每个线段的尾部插入一个断点
    int lisantot=0;
    for(int i=1;i<=N;i++){
        lisan[lisantot++]=line[i].l;
        lisan[lisantot++]=line[i].r;
        lisan[lisantot++]=line[i].r+1;
    }
    sort(lisan,lisan+lisantot);
    int lisanlen=unique(lisan,lisan+lisantot)-lisan;
    for(int i=1;i<=N;i++){
        line[i].l=lower_bound(lisan,lisan+lisanlen,line[i].l)-lisan+1;
        line[i].r=lower_bound(lisan,lisan+lisanlen,line[i].r)-lisan+1;
    }
}
```

1.2 二维点离散化

```
#define ll long long
const int SIZE=1e5;
struct point{
    ll x,y;
}p[SIZE];
bool cmp_x(point a,point b){return a.x < b.x;}
bool cmp_y(point a,point b){return a.y < b.y;}
void Discrete(int n){//n个点,下标[1,n]
    sort(p+1,p+n+1,cmp_x);
    int last=p[1].x,num=1;
    p[1].x = num = 1;
    for(int i=2;i<=n;i++){
        if(p[i].x == last)p[i].x=num;
        else{
            last = p[i].x;
            p[i].x=++num;
```

```
        }
    }
    sort(p+1,p+n+1,cmp_y);
    last=p[1].y,num=1;
    p[1].y=num=1;
    for(int i=2;i<=n;i++){
        if(p[i].y == last)p[i].y=num;
        else{
            last=p[i].y;
            p[i].y=++num;
        }
    }
}
```

2 矩阵快速幂

```
//矩阵快速幂
class mat{
public:
    int n,m;
    ll v[maxn][maxn];
    mat(int n,int m):n(n),m(m){}
    void init()
    {
        memset(v,0,sizeof(v));
    }
    void init1()
    {
        for(int i=0;i<maxn;i++)
            for(int j=0;j<maxn;j++)
                v[i][j]=(i==j); //单位矩阵
    }
    mat operator* (const mat B) const//矩阵乘法 A(n,k)*B(k,m)=C(n,m);
    {
        mat C(n,B.m);
        C.init();
        for(int i=0;i<n;i++)
        for(int j=0;j<B.m;j++)
        for(int k=0;k<m;k++)
            C.v[i][j]=(C.v[i][j]+v[i][k]*B.v[k][j])%Mod;//Mod
        return C;
    }
```

```
    mat operator ^ (int t)//矩阵快速幂 n=m时可用
    {
        mat ans(n,n),now(n,n);
        ans.init1();
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                now.v[i][j]=v[i][j];
        while(t>0)
        {
            if(t&1) ans=ans*now;
            now=now*now;
            t>>=1;
        }
        return ans;
    }
};
```

3 二进制枚举

```
for(int i=0;i<(1<<n);i++)//n个物品取或不取
{

    for(int j=0;j<n;j++)
    {
        if( i & (1<<j) )//取
        {

        }
        else//不取
        {

        }
    }
 }
```

4 状态压缩

```
//判断一个数字x二进制下第i位是不是等于1,反之为0
if(((1<<(i-1))&x)>0)

//将一个数字x二进制下第i位更改成1
```

```
x=x|(1<<(i-1))

//将一个数字x二进制下第i位更改成0
x=x^i(1<<(i-1));

//把一个数字二进制下最靠右的第一个1去掉
x=x&(x-1)
```

5 bitset

```
/*
C++的 bitset 在 bitset 头文件中，它是一种类似数组的结构，
它的每一个元素只能是 0 或 1，每个元素仅用 1 bit空间。
*/
//————————————构造方法————————————
bitset<4> bitset1;    //无参构造，长度为 4，默认每一位为 0
bitset<8> bitset2(12);    //长度为 8，二进制保存，前面用 0 补充
string s = "100101";
bitset<10> bitset3(s);    //长度为10，前面用 0 补充
char s2[] = "10101";
bitset<13> bitset4(s2);    //长度为13，前面用 0 补充
cout << bitset1 << endl;    //0000
cout << bitset2 << endl;    //00001100
cout << bitset3 << endl;    //0000100101
cout << bitset4 << endl;    //0000000010101
//————————————可用函数————————————
bitset<8> foo ("10011011");

cout << foo.count() << endl;    //5    count函数用来求bitset中1的位数
cout << foo.size() << endl;    //8    size函数用来求bitset的大小

cout << foo.test(0) << endl;    //true    test函数用来查下标处的元素是 0 还是 1
cout << foo.test(2) << endl;    //false

cout << foo.any() << endl;    //true    any函数检查bitset中是否有 1
cout << foo.none() << endl;    //false    none函数检查bitset中是否没有 1
cout << foo.all() << endl;    //false    all函数检查bitset中是全部为 1


bitset<8> foo ("10011011");

cout << foo.flip(2) << endl;    //10011111    flip函数传参数时，用于将参数位取反
```

```
cout << foo.flip() << endl;
   //01100000    flip函数不指定参数时，将bitset每一位全部取反

cout << foo.set() <<
   endl;        //11111111    set函数不指定参数时，将bitset的每一位全部置为 1
cout << foo.set(3,0) <<
   endl;    //11110111    set函数指定两位参数时，将第一参数位的元素置为第二参数的值
cout << foo.set(3) << endl;
   //11111111    set函数只有一个参数时，将参数下标处置为 1

cout << foo.reset(4) <<
   endl;    //11101111    reset函数传一个参数时将参数下标处置为 0
cout << foo.reset() << endl;
   //00000000    reset函数不传参数时将bitset的每一位全部置为 0



bitset<8> foo ("10011011");

string s = foo.to_string();    //将bitset转换成string类型
unsigned long a = foo.to_ulong();    //将bitset转换成unsigned long类型
unsigned long long b = foo.to_ullong();    //将bitset转换成unsigned long
   long类型

cout << s << endl;    //10011011
cout << a << endl;    //155
cout << b << endl;    //155
```