

目录

1	离散化	1
1.1	一维线离散化 . . . . .	1
1.2	二维点离散化 . . . . .	1
2	矩阵快速幂	2
3	二进制枚举	3
4	状态压缩	3
5	bitset	3

# 1 离散化

## 1.1 一维线离散化

```
1 struct LINE{
2     int l,r;
3 }line[SIZE*16];
4 int lisan[SIZE*16];
5 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
6     int lisantot=0;
7     for(int i=1;i<=N;i++){
8         lisan[lisantot++]=line[i].l;
9         lisan[lisantot++]=line[i].r;
10        lisan[lisantot++]=line[i].r+1;
11    }
12    sort(lisan,lisan+lisantot);
13    int lisanlen=unique(lisan,lisan+lisantot)-lisan;
14    for(int i=1;i<=N;i++){
15        line[i].l=lower_bound(lisan,lisan+lisanlen,line[i].l)-lisan+1;
16        line[i].r=lower_bound(lisan,lisan+lisanlen,line[i].r)-lisan+1;
17    }
18 }
```

## 1.2 二维点离散化

```
1 #define ll long long
2 const int SIZE=1e5;
3 struct point{
4     ll x,y;
5 }p[SIZE];
6 bool cmp_x(point a,point b){return a.x < b.x;}
7 bool cmp_y(point a,point b){return a.y < b.y;}
8 void Discrete(int n){//n个点,下标[1,n]
9     sort(p+1,p+n+1,cmp_x);
10    int last=p[1].x,num=1;
11    p[1].x = num = 1;
12    for(int i=2;i<=n;i++){
13        if(p[i].x == last)p[i].x=num;
14        else{
15            last = p[i].x;
16            p[i].x=++num;
17        }
18    }
19    sort(p+1,p+n+1,cmp_y);
20    last=p[1].y,num=1;
```

```

21     p[1].y=num=1;
22     for(int i=2;i<=n;i++){
23         if(p[i].y == last)p[i].y=num;
24         else{
25             last=p[i].y;
26             p[i].y=++num;
27         }
28     }
29 }

```

## 2 矩阵快速幂

```

1  //矩阵快速幂
2  class mat{
3  public:
4      int n,m;
5      ll v[maxn][maxn];
6      mat(int n,int m):n(n),m(m){}
7      void init()
8      {
9          memset(v,0,sizeof(v));
10     }
11     void init1()
12     {
13         for(int i=0;i<maxn;i++)
14             for(int j=0;j<maxn;j++)
15                 v[i][j]=(i==j); //单位矩阵
16     }
17     mat operator* (const mat B) const//矩阵乘法  $A(n,k)*B(k,m)=C(n,m)$ ;
18     {
19         mat C(n,B.m);
20         C.init();
21         for(int i=0;i<n;i++)
22             for(int j=0;j<B.m;j++)
23                 for(int k=0;k<m;k++)
24                     C.v[i][j]=(C.v[i][j]+v[i][k]*B.v[k][j])%Mod; //Mod
25         return C;
26     }
27     mat operator ^ (int t)//矩阵快速幂  $n=m$ 时可用
28     {
29         mat ans(n,n),now(n,n);
30         ans.init1();
31         for(int i=0;i<n;i++)
32             for(int j=0;j<n;j++)
33                 now.v[i][j]=v[i][j];
34         while(t>0)
35         {
36             if(t&1) ans=ans*now;

```

```

37         now=now*now;
38         t>>=1;
39     }
40     return ans;
41 }
42 };

```

### 3 二进制枚举

```

1  for(int i=0;i<(1<<n);i++)//n个物品取或不取
2  {
3
4      for(int j=0;j<n;j++)
5      {
6          if( i & (1<<j) )//取
7          {
8
9              }
10         else//不取
11         {
12
13             }
14     }
15 }

```

### 4 状态压缩

```

1  //判断一个数字x二进制下第i位是不是等于1,反之为0
2  if(((1<<(i-1))&x)>0)
3
4  //将一个数字x二进制下第i位更改成1
5  x=x|(1<<(i-1))
6
7  //将一个数字x二进制下第i位更改成0
8  x=x^i(1<<(i-1));
9
10 //把一个数字二进制下最靠右的第一个1去掉
11 x=x&(x-1)

```

### 5 bitset

```

1  /*
2  C++的 bitset 在 bitset 头文件中，它是一种类似数组的结构，
3  它的每一个元素只能是 0 或 1，每个元素仅用 1 bit空间。

```

```

4  */
5  //—————构造方法—————
6  bitset<4> bitset1;    //无参构造，长度为4，默认每一位为0
7  bitset<8> bitset2(12); //长度为8，二进制保存，前面用0补充
8  string s = "100101";
9  bitset<10> bitset3(s); //长度为10，前面用0补充
10 char s2[] = "10101";
11 bitset<13> bitset4(s2); //长度为13，前面用0补充
12 cout << bitset1 << endl;    //0000
13 cout << bitset2 << endl;    //00001100
14 cout << bitset3 << endl;    //0000100101
15 cout << bitset4 << endl;    //0000000010101
16 //—————可用函数—————
17 bitset<8> foo ("10011011");
18
19 cout << foo.count() << endl;    //5    count函数用来求bitset中1的位数
20 cout << foo.size() << endl;    //8    size函数用来求bitset的大小
21
22 cout << foo.test(0) << endl;    //true    test函数用来查下标处的元素是0还是1
23 cout << foo.test(2) << endl;    //false
24
25 cout << foo.any() << endl;    //true    any函数检查bitset中是否有1
26 cout << foo.none() << endl;    //false    none函数检查bitset中是否没有1
27 cout << foo.all() << endl;    //false    all函数检查bitset中是全部为1
28
29
30 bitset<8> foo ("10011011");
31
32 cout << foo.flip(2) << endl;    //10011111    flip函数传参数时，用于将参数位取反
33 cout << foo.flip() << endl;    //01100000    flip函数不指定参数时，将bitset每一位全部取反
34
35 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
36 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
37 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
38
39 cout << foo.reset(4) << endl;    //11101111    reset函数传一个参数时将参数下标处置为0
40 cout << foo.reset() << endl;    //00000000    reset函数不传参数时将bitset的每一位全部置为0
41
42
43
44 bitset<8> foo ("10011011");
45
46 string s = foo.to_string();    //将bitset转换成string类型
47 unsigned long a = foo.to_ulong();    //将bitset转换成unsigned long类型
48 unsigned long long b = foo.to_ullong();    //将bitset转换成unsigned long long类型
49
50 cout << s << endl;    //10011011
51 cout << a << endl;    //155
52 cout << b << endl;    //155

```