常用模板

1. 快速幂

$a^k \ mod \ \ p$

复杂度： $o(log(n))$

代码

递归

```
long long fastpow(long long a, long long k, long long p)
{
    if (k == 0)return 1 % p;
    if (k & 1) {
        return fastpow(a, k - 1, p) * a % p;
    }
    else {
        long long  temp = fastpow(a, k >> 1, p);
        return temp*temp % p;
    }
}
```

非递归

```
typedef long long ll;
ll fastpow(ll a,ll k,ll p)
{
    ll res = 1;
    while (k) {
        if (k & 1)res = res * a % p;
        a = a * a % p;
        k >>= 1;
    }
    return res;
}
```

解释

$a^k \ mod \ \ p$

```
long long fastpow(long long a, long long k, long long p)
{
    if (k == 0)return 1 % p;
    if (k % 2) //用位运算会快一些 k&1
    {
        return fastpow(a, k - 1, p) * a % p;
    }
    else {
        long long  temp = fastpow(a, k / 2, p);
        //注意需要保存该值，不能算两次，否则复杂度变回o(n)
        return temp*temp % p;
    }
}
```

---

2. 匈牙利算法

复杂度： $o(n^3)$

代码

```
int con[510][510];
int vis[510];
int link[510];
int n, m, k;

int find(int x)
{
    for (int i = 1; i <= m; i++)\{
        if (vis[i] == 0 && con[x][i] == 1){
            vis[i] = 1;
            if (link[i] == -1 || find(link[i])){
                link[i] = x;
                return 1;
            }
        }
    }
    return 0;
}
int hungary()
{
    int res = 0;
    memset(link, -1, sizeof(link));
    for (int x = 1; x <= n; x++) {
        memset(vis, 0, sizeof(vis));
        if (find(x))res++;
    }
    return res;
}
void solve()
{
    memset(con,0,sizeof(con));
    cin>>n>>m>>k;
    int x,y;
    for(int i=1;i<=k;i++){
```

```cpp
        cin>>x>>y;
        con[x][y]=1;
    }
    cout<<hungary()<<endl;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    int tt;
    while(tt--)solve();
    return 0;
}
```

解释

```cpp
int con[510][510];/*连接关系 connect*/
int vis[510];/*右边点使用与否*/
int link[510];/*右边点连接的左边点*/
int n, m, k;/*左端点个数，右端点个数，匹配关系数*/
```

```cpp
int find(int x)
{
    for (int i = 1; i <= m; i++)/*遍历右边的点，具体是从1到n还是0到n-1看题意*/
    {
        if (vis[i] == 0 && con[x][i] == 1)
            /*vis判断右边的点在本轮中是否访问过了，可以减少一些运算量*/
            /*con储存的是可以匹配的点的对应关系，点x和y能匹配，则con[x][y]=1*/
        {
            vis[i] = 1;
            if (link[i] == -1 || find(link[i]))
                /*link表示该点有没有被连接，-1表示没有，储存的连接的点的信息*/
                /*find(link[i])就是进行下一步寻找*/
            {
                link[i] = x;/*找到了，连接上*/
                return 1;
            }
        }
    }
    return 0;/*没找到，循环continue*/
}
```

```cpp
int hungary()
{
    int res = 0;
    memset(link, -1, sizeof(link));/*注意连线是需要存储的，不然下一步就没有方向了*/
    for (int x = 1; x <= n; x++) /*具体是从1到n还是0到n-1看题意*/
    {
        memset(vis, 0, sizeof(vis));/*注意这是每轮清空，表示需要抢之前已经匹配的*/
        if (find(x))res++;/*成功一次多一个匹配数*/
    }
    return res;
}
```

```
//主函数
memset(con, 0, sizeof(con));
//随后存入匹配关系
cout<<hungary()<<endl;
```

---

3. 求MEX

复杂度: $o(nlog(n))$

代码

```
const int N=200010;
int a[N];
int ct[N];
int r[N];
//int b[N];
int n;
void solve()
{
    cin >> n;
    memset(ct, 0, sizeof(ct));
    //memset(b, 0, sizeof(b));
    memset(r, 0, sizeof(r));
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        ct[a[i]]++;
    }
    int id = 1;
    int m = 0;
    for (int i = 1; i <= n; i++) {
        ct[a[i]]--;
        r[a[i]] = id;
        while (r[m] == id)
            m++;
        if (ct[m] == 0)
        {
            //b[id] = m;
            //id++;
            //m = 0;
            break;
        }
    }
    cout<<m<<endl;
}
```

解释

```
const int N=200010;
int a[N];
int ct[N];//该数字剩余次数
int r[N];//a中的第几个数字使用在第几轮
```

```
int b[N];
int n;
void solve()
{
    cin >> n;
    memset(ct, 0, sizeof(ct));
    memset(b, 0, sizeof(b));
    memset(r, 0, sizeof(r));
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        ct[a[i]]++;
    }
    int id = 1;
    int m = 0;
    for (int i = 1; i <= n; i++) {
        ct[a[i]]--;
        r[a[i]] = id;//代表a[i]这个数存在且在第一轮
        while (r[m] == id)
            m++;
        if (ct[m] == 0) //说明mex已经是最大的了
        {
            b[id] = m;
            id++;
            m = 0;
        }
    }
    cout << id-1 << endl;
    for (int i = 1; i <= id-1; i++)
        cout << b[i] << ' ';
    cout << endl;
}
```

4. 矩阵快速幂

复杂度：$o(log(n))$

代码

```
typedef long long ll;
const int N = 20, mod = 9973;
const int n;
struct mat {
    int m[N][N];
};
mat a, ans, s;
mat multi(mat a, mat b)
{
    mat c;
    for(int i=0;i<n;i++)
        for (int j = 0; j < n; j++) {
            c.m[i][j] = 0;
            for (int k = 0; k < n; k++){
                c.m[i][j] = (c.m[i][j] + (a.m[i][k] * b.m[k][j]) % mod) % mod;
```

```
            }
            c.m[i][j] %= mod;
        }
    return c;
}
mat fastpow(mat a, int k)
{
    mat res;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            if (i == j)res.m[i][j] = 1;
            else res.m[i][j] = 0;
        }
    while (k) {
        if (k & 1)res = multi(res, a);
        a = multi(a, a);
        k >>= 1;
    }
    return res;
}
void init()
{
    a.m[0][0]=1;

    s.m[0][0]=1;
}
void solve()
{
    int k;
    cin >> n >> k;
    ans = fastpow(a, k);
    ans = multi(ans, s);
    cout<< ans.m[0][0]<< endl;
}
```

解释

```
const int N = 20, mod = 9973;//方阵大小，%的值
int n ;//方阵大小
struct mat {
    int m[N][N];
};
mat a, ans;//初始矩阵，结果矩阵
mat multi(mat a, mat b)//矩阵乘法
{
    mat c;
    for(int i=0;i<n;i++)
        for (int j = 0; j < n; j++) {
            c.m[i][j] = 0;//注意清零
            for (int k = 0; k < n; k++){
                c.m[i][j] = (c.m[i][j] + (a.m[i][k] * b.m[k][j]) % mod) % mod;
                //注意每步取%
            }

```

```
        }
    return c;
}
mat fastpow(mat a, int k)//矩阵快速幂
{
    mat res;
    //注意初始化为单位矩阵
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            if (i == j)res.m[i][j] = 1;
            else res.m[i][j] = 0;
        }
    while (k) {
        if (k & 1)res = multi(res, a);
        a = multi(a, a);
        k >>= 1;
    }
    return res;
}
void solve()
{
    int k;
    cin >> n >> k;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> a.m[i][j];
    ans = fastpow(a, k);
    //下面是输出矩阵的迹
    /*int sum = 0;
    for (int i = 0; i < n; i++) {
        sum = (sum + ans.m[i][i] )% mod;
    }
    cout << sum << endl;*/
}
```

---

5. 二分查找

复杂度: $o(logn)$

代码

```
long long l = 0, r = h;
    long long  Min = 0;
    while (l <= r)
    {
        long long  mid = (l + r) / 2;
        if (check(mid))
        {
            r = mid - 1;
            Min = mid;
        }
        else
        {
            l = mid + 1;
```

```
        }
    }
    cout << Min << endl;
```

解释

```
long long l = 0, r = h;
    long long  Min = 0;
    while (l <= r)
    {
        long long  mid = (l + r) / 2;
        if (check(mid))//判断条件 大于
        {
            r = mid - 1;
            Min = mid;
        }
        else
        {
            l = mid + 1;
        }
    }
    cout << Min << endl;
```

---

6. 树状数组区间查询

复杂度: $o(nlog(n))$

代码

```
typedef long long ll;
int c[1000010];
int n;
int lowbit(int x)
{
    return x & -x;
}
ll sum(int i)
{
    ll ret = 0;
    while (i > 0) {
        ret += c[i];
        i -= lowbit(i);
    }
    return ret;
}
void update(int i, int val)
{
    while (i <= n) {
        c[i] += val;
        i += lowbit(i);
    }
}
```

解释

```cpp
typedef long long ll;
int c[1000010];
int n;
int lowbit(int x)
{
    return x & -x;
}
ll sum(int i)//求前缀和
{
    ll ret = 0;
    while (i > 0) {
        ret += c[i];
        i -= lowbit(i);
    }
    return ret;
}
void update(int i, int val)//更新树状数组
{
    while (i <= n) {
        c[i] += val;
        i += lowbit(i);
    }
}
```

---

7. sort排序

复杂度：$o(nlog(n))$

代码

```cpp
bool cmp(const node& x, const node& y)
{
    return x.num < y.num;
}
sort(a+1,a+n+1,greater<int>());
```

解释

```cpp
//结构体成员排序
bool cmp(const node& x, const node& y)
{
    return x.num < y.num;
}
//递减排序
sort(a+1,a+n+1,greater<int>());
```

---

8. 区间修改+单点查询

复杂度：$o(nlog(n))$

代码

```
typedef long long ll;
int n;
int c[100010];
void add(int p, int x)
{
    while (p <= n)c[p] += x, p += p & -p;
}
void range_add(int l, int r, int x)
{
    add(l, x), add(r + 1, -x);
}
int ask(int p)
{
    int res = 0;
    while (p)res += c[p], p -= p & -p;
    return res;
}
```

解释

```
typedef long long ll;
int n;
int c[100010];
void add(int p, int x)
{
    while (p <= n)c[p] += x, p += p & -p;
}
void range_add(int l, int r, int x)//区间修改
{
    add(l, x), add(r + 1, -x);
}
int ask(int p)
{
    int res = 0;
    while (p)res += c[p], p -= p & -p;
    return res;
}

```

---

9. 区间修改+区间查询

复杂度：$o(nlog(n))$

代码

```
typedef long long ll;
int n;
int c1[100010];
int c2[100010];
void add(int p, int x)
{
    for (int i = p; i <= n; i += i & -i)
        c1[i] += x, c2[i] += p * x;
}
void range_add(int l, int r, int x)
{
    add(l, x), add(r + 1, -x);
}
int ask(int p)
{
    int res = 0;
    for (int i = p; i; i -= i & -i)
        res += (p + 1) * c1[i] - c2[i];
    return res;
}
int range_ask(int l, int r)
{
    return ask(r) - ask(l - 1);
}
```

解释

```
typedef long long ll;
int n;
int c1[100010];
int c2[100010];
void add(int p, int x)//维护两个树状数组
{
    for (int i = p; i <= n; i += i & -i)
        c1[i] += x, c2[i] += p * x;
}
void range_add(int l, int r, int x)
{
    add(l, x), add(r + 1, -x);
}
int ask(int p)//前缀和查询，查询1到p的和
{
    int res = 0;
    for (int i = p; i; i -= i & -i)
        res += (p + 1) * c1[i] - c2[i];
    return res;
}
int range_ask(int l, int r)//区间查询
{
    return ask(r) - ask(l - 1);
}
```