

常用模板

1. 快速幂

$$a^k \bmod p$$

复杂度: $O(\log(n))$

代码

递归

```
1 long long fastpow(long long a, long long k, long long p)
2 {
3     if (k == 0) return 1 % p;
4     if (k & 1) {
5         return fastpow(a, k - 1, p) * a % p;
6     }
7     else {
8         long long temp = fastpow(a, k >> 1, p);
9         return temp*temp % p;
10    }
11 }
```

非递归

```
1 typedef long long ll;
2 ll fastpow(ll a,ll k,ll p)
3 {
4     ll res = 1;
5     while (k) {
6         if (k & 1) res = res * a % p;
7         a = a * a % p;
8         k >>= 1;
9     }
10    return res;
11 }
```

解释

$$a^k \bmod p$$

```

1  long long fastpow(long long a, long long k, long long p)
2  {
3      if (k == 0) return 1 % p;
4      if (k % 2) //用位运算会快一些 k&1
5      {
6          return fastpow(a, k - 1, p) * a % p;
7      }
8      else {
9          long long temp = fastpow(a, k / 2, p);
10         //注意需要保存该值，不能算两次，否则复杂度变回o(n)
11         return temp*temp % p;
12     }
13 }

```

2. 匈牙利算法

复杂度: $O(n^3)$

代码

```

1  int con[510][510];
2  int vis[510];
3  int link[510];
4  int n, m, k;
5
6  int find(int x)
7  {
8      for (int i = 1; i <= m; i++)\{
9          if (vis[i] == 0 && con[x][i] == 1){
10             vis[i] = 1;
11             if (link[i] == -1 || find(link[i])){
12                 link[i] = x;
13                 return 1;
14             }
15         }
16     }
17     return 0;
18 }
19 int hungary()
20 {
21     int res = 0;
22     memset(link, -1, sizeof(link));
23     for (int x = 1; x <= n; x++) {
24         memset(vis, 0, sizeof(vis));
25         if (find(x)) res++;
26     }
27     return res;
28 }
29 void solve()
30 {
31     memset(con, 0, sizeof(con));
32     cin >> n >> m >> k;
33     int x, y;
34     for (int i = 1; i <= k; i++) {

```

```

35         cin>>x>>y;
36         con[x][y]=1;
37     }
38     cout<<hungary()<<endl;
39 }
40 int main()
41 {
42     ios::sync_with_stdio(false);
43     cin.tie(0); cout.tie(0);
44     int tt;
45     while(tt--)>0 solve();
46     return 0;
47 }

```

解释

```

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

```

```

1  int find(int x)
2  {
3      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*/
4      {
5          if (vis[i] == 0 && con[x][i] == 1)
6              /*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*/
7              /*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*/
8              {
9                  vis[i] = 1;
10                 if (link[i] == -1 || find(link[i]))
11                     /*link表示该点有没有被连接，-1表示没有，储存的连接的信息*/
12                     /*find(link[i])就是进行下一步寻找*/
13                     {
14                         link[i] = x; /*找到了，连接上*/
15                         return 1;
16                     }
17             }
18     }
19     return 0; /*没找到，循环continue*/
20 }

```

```

1  int hungary()
2  {
3      int res = 0;
4      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*/
5      for (int x = 1; x <= n; x++) /*具体是从1到n还是0到n-1看题意*/
6      {
7          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*/
8          if (find(x)) res++; /*成功一次多一个匹配数*/
9      }
10     return res;
11 }

```

```

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

```

3. 求MEX

复杂度: $O(n \log(n))$

代码

```

1 const int N=200010;
2 int a[N];
3 int ct[N];
4 int r[N];
5 //int b[N];
6 int n;
7 void solve()
8 {
9     cin >> n;
10    memset(ct, 0, sizeof(ct));
11    //memset(b, 0, sizeof(b));
12    memset(r, 0, sizeof(r));
13    for (int i = 1; i <= n; i++) {
14        cin >> a[i];
15        ct[a[i]]++;
16    }
17    int id = 1;
18    int m = 0;
19    for (int i = 1; i <= n; i++) {
20        ct[a[i]]--;
21        r[a[i]] = id;
22        while (r[m] == id)
23            m++;
24        if (ct[m] == 0)
25        {
26            //b[id] = m;
27            //id++;
28            //m = 0;
29            break;
30        }
31    }
32    cout<<m<<endl;
33 }

```

解释

```

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

```

```

5  int b[N];
6  int n;
7  void solve()
8  {
9      cin >> n;
10     memset(ct, 0, sizeof(ct));
11     memset(b, 0, sizeof(b));
12     memset(r, 0, sizeof(r));
13     for (int i = 1; i <= n; i++) {
14         cin >> a[i];
15         ct[a[i]]++;
16     }
17     int id = 1;
18     int m = 0;
19     for (int i = 1; i <= n; i++) {
20         ct[a[i]]--;
21         r[a[i]] = id; //代表a[i]这个数存在且在第一轮
22         while (r[m] == id)
23             m++;
24         if (ct[m] == 0) //说明mex已经是最大的了
25         {
26             b[id] = m;
27             id++;
28             m = 0;
29         }
30     }
31     cout << id-1 << endl;
32     for (int i = 1; i <= id-1; i++)
33         cout << b[i] << ' ';
34     cout << endl;
35 }

```

4. 矩阵快速幂

复杂度: $O(\log(n))$

代码

```

1  typedef long long ll;
2  const int N = 20, mod = 9973;
3  const int n;
4  struct mat {
5      int m[N][N];
6  };
7  mat a, ans, s;
8  mat multi(mat a, mat b)
9  {
10     mat c;
11     for(int i=0;i<n;i++)
12         for (int j = 0; j < n; j++) {
13             c.m[i][j] = 0;
14             for (int k = 0; k < n; k++){
15                 c.m[i][j] = (c.m[i][j] + (a.m[i][k] * b.m[k][j]) % mod) %
mod;

```

```

16         }
17         c.m[i][j] %= mod;
18     }
19     return c;
20 }
21 mat fastpow(mat a, int k)
22 {
23     mat res;
24     for (int i = 0; i < n; i++)
25         for (int j = 0; j < n; j++) {
26             if (i == j) res.m[i][j] = 1;
27             else res.m[i][j] = 0;
28         }
29     while (k) {
30         if (k & 1) res = multi(res, a);
31         a = multi(a, a);
32         k >>= 1;
33     }
34     return res;
35 }
36 void init()
37 {
38     a.m[0][0]=1;
39
40     s.m[0][0]=1;
41 }
42 void solve()
43 {
44     int k;
45     cin >> n >> k;
46     ans = fastpow(a, k);
47     ans = multi(ans, s);
48     cout<< ans.m[0][0]<< endl;
49 }

```

解释

```

1  const int N = 20, mod = 9973; //方阵大小, %的值
2  int n; //方阵大小
3  struct mat {
4      int m[N][N];
5  };
6  mat a, ans; //初始矩阵, 结果矩阵
7  mat multi(mat a, mat b) //矩阵乘法
8  {
9      mat c;
10     for(int i=0;i<n;i++)
11         for (int j = 0; j < n; j++) {
12             c.m[i][j] = 0; //注意清零
13             for (int k = 0; k < n; k++){
14                 c.m[i][j] = (c.m[i][j] + (a.m[i][k] * b.m[k][j]) % mod) %
mod;
15                 //注意每步取%
16             }
17

```

```

18     }
19     return c;
20 }
21 mat fastpow(mat a, int k)//矩阵快速幂
22 {
23     mat res;
24     //注意初始化为单位矩阵
25     for (int i = 0; i < n; i++)
26         for (int j = 0; j < n; j++) {
27             if (i == j)res.m[i][j] = 1;
28             else res.m[i][j] = 0;
29         }
30     while (k) {
31         if (k & 1)res = multi(res, a);
32         a = multi(a, a);
33         k >>= 1;
34     }
35     return res;
36 }
37 void solve()
38 {
39     int k;
40     cin >> n >> k;
41     for (int i = 0; i < n; i++)
42         for (int j = 0; j < n; j++)
43             cin >> a.m[i][j];
44     ans = fastpow(a, k);
45     //下面是输出矩阵的迹
46     /*int sum = 0;
47     for (int i = 0; i < n; i++) {
48         sum = (sum + ans.m[i][i] )% mod;
49     }
50     cout << sum << endl;*/
51 }

```

5. 二分查找

复杂度: $o(\log n)$

代码

```

1  long long l = 0, r = h;
2  long long Min = 0;
3  while (l <= r)
4  {
5      long long mid = (l + r) / 2;
6      if (check(mid))
7      {
8          r = mid - 1;
9          Min = mid;
10     }
11     else
12     {
13         l = mid + 1;

```

```

14     }
15 }
16 cout << Min << endl;

```

解释

```

1  long long l = 0, r = h;
2  long long Min = 0;
3  while (l <= r)
4  {
5      long long mid = (l + r) / 2;
6      if (check(mid))//判断条件 大于
7      {
8          r = mid - 1;
9          Min = mid;
10     }
11     else
12     {
13         l = mid + 1;
14     }
15 }
16 cout << Min << endl;

```

6. 树状数组区间查询

复杂度: $O(n \log(n))$

代码

```

1  typedef long long ll;
2  int c[1000010];
3  int n;
4  int lowbit(int x)
5  {
6      return x & -x;
7  }
8  ll sum(int i)
9  {
10     ll ret = 0;
11     while (i > 0) {
12         ret += c[i];
13         i -= lowbit(i);
14     }
15     return ret;
16 }
17 void update(int i, int val)
18 {
19     while (i <= n) {
20         c[i] += val;
21         i += lowbit(i);
22     }
23 }

```


解释

```
1 typedef long long ll;
2 int c[1000010];
3 int n;
4 int lowbit(int x)
5 {
6     return x & -x;
7 }
8 ll sum(int i)//求前缀和
9 {
10     ll ret = 0;
11     while (i > 0) {
12         ret += c[i];
13         i -= lowbit(i);
14     }
15     return ret;
16 }
17 void update(int i, int val)//更新树状数组
18 {
19     while (i <= n) {
20         c[i] += val;
21         i += lowbit(i);
22     }
23 }
```

7. sort排序

复杂度: $O(n\log(n))$

代码

```
1 bool cmp(const node& x, const node& y)
2 {
3     return x.num < y.num;
4 }
5 sort(a+1,a+n+1,greater<int>());
```

解释

```
1 //结构体成员排序
2 bool cmp(const node& x, const node& y)
3 {
4     return x.num < y.num;
5 }
6 //递减排序
7 sort(a+1,a+n+1,greater<int>());
```

8. 区间修改+单点查询

复杂度: $O(n\log(n))$

代码

```
1  typedef long long ll;
2  int n;
3  int c[100010];
4  void add(int p, int x)
5  {
6      while (p <= n)c[p] += x, p += p & -p;
7  }
8  void range_add(int l, int r, int x)
9  {
10     add(l, x), add(r + 1, -x);
11 }
12 int ask(int p)
13 {
14     int res = 0;
15     while (p)res += c[p], p -= p & -p;
16     return res;
17 }
```

解释

```
1  typedef long long ll;
2  int n;
3  int c[100010];
4  void add(int p, int x)
5  {
6      while (p <= n)c[p] += x, p += p & -p;
7  }
8  void range_add(int l, int r, int x)//区间修改
9  {
10     add(l, x), add(r + 1, -x);
11 }
12 int ask(int p)
13 {
14     int res = 0;
15     while (p)res += c[p], p -= p & -p;
16     return res;
17 }
18
```

9. 区间修改+区间查询

复杂度: $O(n\log(n))$

代码

```
1 typedef long long ll;
2 int n;
3 int c1[100010];
4 int c2[100010];
5 void add(int p, int x)
6 {
7     for (int i = p; i <= n; i += i & -i)
8         c1[i] += x, c2[i] += p * x;
9 }
10 void range_add(int l, int r, int x)
11 {
12     add(l, x), add(r + 1, -x);
13 }
14 int ask(int p)
15 {
16     int res = 0;
17     for (int i = p; i; i -= i & -i)
18         res += (p + 1) * c1[i] - c2[i];
19     return res;
20 }
21 int range_ask(int l, int r)
22 {
23     return ask(r) - ask(l - 1);
24 }
```

解释

```
1 typedef long long ll;
2 int n;
3 int c1[100010];
4 int c2[100010];
5 void add(int p, int x)//维护两个树状数组
6 {
7     for (int i = p; i <= n; i += i & -i)
8         c1[i] += x, c2[i] += p * x;
9 }
10 void range_add(int l, int r, int x)
11 {
12     add(l, x), add(r + 1, -x);
13 }
14 int ask(int p)//前缀和查询，查询1到p的和
15 {
16     int res = 0;
17     for (int i = p; i; i -= i & -i)
18         res += (p + 1) * c1[i] - c2[i];
19     return res;
20 }
21 int range_ask(int l, int r)//区间查询
22 {
23     return ask(r) - ask(l - 1);
24 }
```

