

Nantong University ICPC Team Notebook (2018-19)

thirtiseven wanggan

目录

第一章	输入输出	3
1.1	取消同步	3
1.2	浮点数输出格式	3
1.3	整型快速输入	3
1.4	字符串快速输入	4
1.5	整型快速输出	4
1.6	字符串快速输出	5
1.7	python 输入	5
第二章	动态规划	6
2.1	背包问题	6
2.2	最长单调子序列 (nlogn)	6
第三章	数学	8
3.1	暴力判素数	8
3.2	埃氏筛	8
3.3	欧拉筛	8
3.4	分解质因数	9
3.5	暴力判回文数	9
3.6	最大公约数	10
3.7	最小公倍数	10
3.8	扩展欧几里得	10
3.9	中国剩余定理	10
3.10	欧拉函数	11
3.11	求逆元	11
3.12	$C(n,m) \bmod p$ (n 很大 p 可以很大)	11
3.13	Lucas 定理	11
3.14	快速乘法取模	12
3.15	快速幂取模	12
3.16	计算从 $C(n, 0)$ 到 $C(n, p)$ 的值	12
3.17	二分分数树 (Stern-Brocot Tree)	13
第四章	图论	14
4.1	并查集	14
4.2	可撤销并查集 (按秩合并)	14
4.3	Kruskal 最小生成树	15
4.4	Prim 最小生成树	16
4.5	SPFA 最短路	17
4.6	dijkstra 最短路	17
4.7	Floyd 任意两点间最短路	19
4.8	Dinic 最大流	19
4.9	2-SAT 问题	21
第五章	数据结构	23
5.1	树状数组	23
5.2	二维树状数组	23
5.3	堆	24
5.4	RMQ	24
5.5	线段树	25
5.6	ZKW 线段树	26
5.7	莫队算法	27

第六章	字符串	29
6.1	前缀树	29
6.2	后缀数组	29
6.3	后缀自动机	30
6.4	最长回文子串	32
第七章	类	33
7.1	点类	33
7.2	分数类	33
7.3	矩阵	34
7.4	01 矩阵	35
第八章	黑科技	37
8.1	位运算	37
8.2	珂朵莉树 (Old Driver Tree)	37

第一章 输入输出

1.1 取消同步

```
1 std::ios::sync_with_stdio(false);
2 std::cin.tie(0);
```

1.2 浮点数输出格式

```
1 //include <iomanip>
2
3 std::cout << std::fixed << std::setprecision(12) << ans << std::endl;
```

1.3 整型快速输入

```
1 // 整型
2 //若读入不成功，返回false
3 //ios::sync_with_stdio(true)
4 //include <cctype>
5 bool quick_in(int &x) {
6     char c;
7     while((c = getchar()) != EOF && !isdigit(c));
8     if(c == EOF) {
9         return false;
10    }
11    x = 0;
12    do {
13        x *= 10;
14        x += c - '0';
15    } while((c = getchar()) != EOF && isdigit(c));
16    return true;
17 }
18
19 //带符号整型
20 //直接=返回值
21 //include <cctype>
22 int read() {
23     int x = 0, l = 1; char ch = getchar();
24     while (!isdigit(ch)) {if (ch=='-') l=-1; ch=getchar();}
```

```
25     while (isdigit(ch)) x=x*10+(ch^48),ch=getchar();
26     return x*1;
27 }
28
29 template <class T>
30 inline bool Read(T &ret) {
31     char c; int sgn;
32     if(c=getchar(),c==EOF) return 0; //EOF
33     while(c!='-'&&(c<'0' || c>'9')) c=getchar();
34     sgn=(c=='-') ?-1:1 ;
35     ret=(c=='-') ?0:(c-'0');
36     while(c=getchar(),c>='0'&&c<='9')
37         ret=ret*10+(c-'0');
38     ret*=sgn;
39     return 1;
40 }
```

1.4 字符串快速输入

```
1 bool quick_in(char *p) {
2     char c;
3     while((c = getchar()) != EOF && (c == '\u' || c == '\n'));
4     if(c == EOF) {
5         return false;
6     }
7     do {
8         *p++ = c;
9     } while((c=getchar()) != EOF && c != '\u' && c != '\n');
10    *p = 0;
11    return true;
12 }
```

1.5 整型快速输出

```
1 void quick_out(int x) {
2     char str[13];
3     if(x) {
4         int i;
5         for(i = 0; x; ++i) {
6             str[i] = x % 10 + '0';
7             x /= 10;
8         }
9         while(i--) {
10            putchar(str[i]);
11        }
12    } else {
13        putchar('0');
14    }
15 }
```

1.6 字符串快速输出

```
1 void quick_out(char *p) {  
2     while(*p) {  
3         putchar(*p++);  
4     }  
5 }
```

1.7 python 输入

```
1 a, b, c =map(int,input().split(' '))
```

第二章 动态规划

2.1 背包问题

```
1  const int maxn=100005;
2  int w[maxn],v[maxn],num[maxn];
3  int W,n;
4  int dp[maxn];
5
6  void ZOP(int weight, int value) {
7      for(int i = W; i >= weight; i--) {
8          dp[i]=std::max(dp[i],dp[i-weight]+value);
9      }
10 }
11
12 void CP(int weight, int value){
13     for(int i = weight; i <= W; i++) {
14         dp[i] = std::max(dp[i], dp[i-weight]+value);
15     }
16 }
17
18 void MP(int weight, int value, int cnt){
19     if(weight*cnt >= W) {
20         CP(weight, value);
21     } else {
22         for(int k = 1; k < cnt; k <= 1) {
23             ZOP(k*weight, k*value), cnt -= k;
24         }
25         ZOP(cnt*weight, cnt*value);
26     }
27 }
```

2.2 最长单调子序列 (nlogn)

```
1  int arr[maxn], n;
2
3  template<class Cmp>
4  int LIS (Cmp cmp) {
5      static int m, end[maxn];
6      m = 0;
7      for (int i=0; i<n; i++) {
8          int pos = lower_bound(end, end+m, arr[i], cmp)-end;
9          end[pos] = arr[i], m += pos==m;
```

```
10     }
11     return m;
12 }
13
14 bool greater1(int value) {
15     return value >=1;
16 }
17
18 /*****
19     std::cout << LIS(std::less<int>()) << std::endl;           //严格上升
20     std::cout << LIS(std::less_equal<int>()) << std::endl;     //非严格上升
21     std::cout << LIS(std::greater<int>()) << std::endl;        //严格下降
22     std::cout << LIS(std::greater_equal<int>()) << std::endl;  //非严格下降
23     std::cout << count_if(a,a+7,std::greater1) << std::endl;  //计数
24 *****/
```


第三章 数学

3.1 暴力判素数

```
1 bool is_prime(int n) {
2     if(n < 2) return false;
3     for(int i = 2; i * i <= n; i++) {
4         if(n % i == 0) return false;
5     }
6     return true;
7 }
```

3.2 埃氏筛

```
1 bool prime_or_not[maxn];
2 for (int i = 2; i <= int(sqrt(maxn)); i++) {
3     if (!prime_or_not[i]) {
4         for (int j = i * i; j <= maxn; j = j+i) {
5             prime_or_not[j] = 1;
6         }
7     }
8 }
```

3.3 欧拉筛

```
1 #include <iostream>
2
3 const int maxn = 1234;
4 int flag[maxn], primes[maxn], totPrimes;
5
6 void euler_sieve(int n) {
7     totPrimes = 0;
8     memset(flag, 0, sizeof(flag));
9     for (int i = 2; i <= n; i++) {
10         if (!flag[i]) {
11             primes[totPrimes++] = i;
12         }
13         for (int j = 0; i * primes[j] <= n; j++) {
14             flag[i * primes[j]] = true;
15             if (i % primes[j] == 0)
16                 break;
17         }
18     }
19 }
```

```
17     }
18 }
19 }
```

3.4 分解质因数

```
1 int cnt[maxn]; // 存储质因子是什么
2 int num[maxn]; // 该质因子的个数
3 int tot = 0; // 质因子的数量
4 void factorization(int x) // 输入x, 返回cnt数组和num数组
5 {
6     for(int i=2; i*i<=x; i++)
7     {
8         if(x%i==0)
9         {
10             cnt[tot]=i;
11             num[tot]=0;
12             while(x%i==0)
13             {
14                 x/=i;
15                 num[tot]++;
16             }
17             tot++;
18         }
19     }
20     if(x!=1)
21     {
22         cnt[tot]=x;
23         num[tot]=1;
24         tot++;
25     }
26 }
```

3.5 暴力判回文数

```
1 bool is_palindrome(int bob) {
2     int clare = bob, dave = 0;
3     while (clare){
4         dave = dave * 10 + clare % 10;
5         clare /= 10;
6     }
7     if(bob == dave) {
8         return true;
9     } else {
10        return false;
11    }
12 }
```

3.6 最大公约数

```
1 ll gcd(ll a, ll b) {
2     ll t;
3     while(b != 0) {
4         t=a%b;
5         a=b;
6         b=t;
7     }
8     return a;
9 }
```

3.7 最小公倍数

```
1 ll lcm(ll a, ll b) {
2     return a * b / gcd(a, b);
3 }
```

3.8 扩展欧几里得

```
1 void Gcd(int a, int b, int &d, int &x, int &y){
2     if(!b) {
3         d=a;
4         x=1;
5         y=0;
6     } else {
7         Gcd(b, a%b, d, y, x);
8         y-=x*(a/b);
9     }
10 }
```

3.9 中国剩余定理

```
1 LL Crt(LL *div, LL *rmd, LL len) {
2     LL sum = 0;
3     LL lcm = 1;
4     //lcm为除数们的最小公倍数, 若div互素, 则如下一行计算lcm
5     for (int i = 0; i < len; ++i)
6         lcm *= div[i];
7     for (int i = 0; i < len; ++i) {
8         LL bsn = lcm / div[i];
9         LL inv = Inv(bsn, div[i]);
10        // dvd[i] = inv[i] * bsn[i] * rmd[i]
11        LL dvd = MulMod(MulMod(inv, bsn, lcm), rmd[i], lcm);
12        sum = (sum + dvd) % lcm;
13    }
```

```
14     return sum;
15 }
```

3.10 欧拉函数

```
1 LL EulerPhi(LL n){
2     LL m = sqrt(n + 0.5);
3     LL ans = n;
4     for(LL i = 2; i <= m; ++i)
5         if(n % i == 0) {
6             ans = ans - ans / i;
7             while(n % i == 0)
8                 n/=i;
9         }
10    if(n > 1)
11        ans = ans - ans / n;
12    return ans;
13 }
```

3.11 求逆元

```
1 LL Inv(LL a, LL n){
2     return PowMod(a, EulerPhi(n) - 1, n);
3     //return PowMod(a,n-2,n); //n为素数
4 }
```

3.12 $C(n,m) \bmod p$ (n 很大 p 可以很大)

```
1 LL C(const LL &n, const LL &m, const int &pr) {
2     LL ans = 1;
3     for (int i = 1; i <= m; i++) {
4         LL a = (n - m + i) % pr;
5         LL b = i % pr;
6         ans = (ans * (a * Inv(b, pr)) % pr) % pr;
7     }
8     return ans;
9 }
```

3.13 Lucas 定理

```
1 //C(n, m) mod p (n 很大 p 较小 (不知道能不能为非素数))
2 LL Lucas(LL n, LL m, const int &pr) {
3     if (m == 0) return 1;
4     return C(n % pr, m % pr, pr) * Lucas(n / pr, m / pr, pr) % pr;
5 }
```

3.14 快速乘法取模

```
1 //by sevenkplus
2 #define ll long long
3 #define ld long double
4 ll mul(ll x,ll y,ll z){return (x*y-(ll)(x/(ld)z*y+1e-3)*z+z)%z;}
5
6 //by Lazer2001
7 inline long long mmul (long long a, long long b, const long long& Mod) {
8     long long lf = a * (b >> 25LL) % Mod * (1LL << 25) % Mod;
9     long long rg = a * ( b & ( ( 1LL << 25 ) - 1 ) ) % Mod ;
10    return (lf + rg) % Mod ;
11 }
```

3.15 快速幂取模

```
1 using LL = long long;
2
3 LL PowMod(LL a, LL b, const LL &Mod) {
4     a %= Mod;
5     LL ans = 1;
6     while(b) {
7         if (b & 1){
8             ans = (ans * a) % Mod;
9         }
10        a = (a * a) % Mod;
11        b >>= 1;
12    }
13    return ans;
14 }
```

3.16 计算从 $C(n, 0)$ 到 $C(n, p)$ 的值

```
1 //by Yuhao Du
2 int p;
3 std::vector<int> gao(int n) {
4     std::vector<int> ret(p+1,0);
5     if (n==0) {
6         ret[0]=1;
7     } else if (n%2==0) {
8         std::vector<int> c = gao(n/2);
9         for(int i = 0; i <= p+1; i++) {
10             for(int j = 0; j <= p+1; j++) {
11                 if (i+j<=p) ret[i+j]+=c[i]*c[j];
12             }
13         }
14     }
```

```
14     } else {
15         std::vector<int> c = gao(n-1);
16         for(int i = 0; i <= p+1; i++) {
17             for(int j = 0; j <= 2; j++) {
18                 if (i+j<=p) ret[i+j]+=c[i];
19             }
20         }
21     }
22     return ret;
23 }
```

3.17 二分分数树 (Stern-Brocot Tree)

```
1  //Author:CookieC
2  //未做模板调整, 请自行调整
3  #include <cmath>
4  #define LL long long
5  #define LD long double
6
7  void SternBrocot(LD X, LL &A, LL &B) {
8      A=X+0.5;
9      B=1;
10     if(A==X)
11         return;
12     LL la=X, lb=1, ra=X+1, rb=1;
13     long double C=A, a, b, c;
14     do {
15         a = la+ra;
16         b = lb+rb;
17         c = a/b;
18         if(std::abs(C-X) > std::abs(c-X)) {
19             A=a;
20             B=b;
21             C=c;
22             if(std::abs(X-C) < 1e-10) {
23                 break;
24             }
25         }
26         if(X<c) {
27             ra=a;
28             rb=b;
29         } else {
30             la=a;
31             lb=b;
32         }
33     } while(lb+rb<=1e5);
34 }
```

第四章 图论

4.1 并查集

```
1 int fa[N];
2
3 void init(int n) {
4     for (int i = 1; i <= n; i++) fa[i] = i;
5 }
6
7 int find(int u) {
8     return fa[u] == u ? fa[u] : fa[u] = find(fa[u]);
9 }
10
11 void unin(int u, int v) {
12     fa[find(v)] = find(u);
13 }
```

4.2 可撤销并查集（按秩合并）

```
1 #include <iostream>
2 #include <stack>
3 #include <utility>
4
5 class UFS {
6     private:
7         int *fa, *rank;
8         std::stack <std::pair <int*, int> > stk ;
9     public:
10         UFS() {}
11         UFS(int n) {
12             fa = new int[(const int)n + 1];
13             rank = new int[(const int)n + 1];
14             memset (rank, 0, n+1);
15             for (int i = 1; i <= n; ++i) {
16                 fa [i] = i;
17             }
18         }
19         inline int find(int x) {
20             while (x ^ fa[x]) {
21                 x = fa[x];
22             }
23             return x;
24         }
25         inline void merge(int x, int y) {
26             x = find(x);
27             y = find(y);
28             if (x == y) return;
29             if (rank[x] < rank[y]) {
30                 fa[x] = y;
31             } else {
32                 fa[y] = x;
33                 if (rank[x] == rank[y]) rank[x]++;
34             }
35         }
36         inline void split(int x, int y) {
37             x = find(x);
38             y = find(y);
39             if (x == y) return;
40             stk.push({fa[x], rank[x]});
41             fa[x] = y;
42             rank[x] = 0;
43         }
44         inline void undo() {
45             if (stk.empty()) return;
46             auto [fa_x, rank_x] = stk.top();
47             stk.pop();
48             fa[fa_x] = x;
49             rank[rank_x] = rank_x;
50         }
51 }
```

```

22     }
23     return x ;
24 }
25 inline int Join (int x, int y) {
26     x = find(x), y = find(y);
27     if (x == y) {
28         return 0;
29     }
30     if (rank[x] <= rank[y]) {
31         stk.push(std::make_pair (fa + x, fa[x]));
32         fa[x] = y;
33         if (rank[x] == rank[y]) {
34             stk.push(std::make_pair (rank + y, rank[y]));
35             ++rank[y];
36             return 2;
37         }
38         return 1 ;
39     }
40     stk.push(std::make_pair(fa + y, fa [y]));
41     return fa[y] = x, 1;
42 }
43 inline void Undo ( ) {
44     *stk.top( ).first = stk.top( ).second ;
45     stk.pop( ) ;
46 }
47 }T;

```

4.3 Kruskal 最小生成树

```

1  #include <vector>
2  #include <algorithm>
3
4  #define maxm 1000
5  #define maxn 1000
6
7  class Kruskal {
8      struct UdEdge {
9          int u, v, w;
10         UdEdge(){}
11         UdEdge(int u,int v,int w):u(u), v(v), w(w){}
12     };
13     int N, M;
14     UdEdge pool[maxm];
15     UdEdge *E[maxm];
16     int P[maxn];
17     int Find(int x){
18         if(P[x] == x)
19             return x;
20         return P[x] = Find(P[x]);
21     }
22     public:
23     static bool cmp(const UdEdge *a, const UdEdge *b) {

```



```

24     return a->w < b->w;
25 }
26 void Clear(int n) {
27     N = n;
28     M = 0;
29 }
30 void AddEdge(int u, int v, int w) {
31     pool[M] = UEdge(u, v, w);
32     E[M] = &pool[M];
33     ++M;
34 }
35 int Run() {
36     int i, ans=0;
37     for(i = 1; i <= N; ++i)
38         P[i] = i;
39     std::sort(E, E+M, cmp);
40     for(i = 0; i < M; ++i) {
41         UEdge *e = E[i];
42         int x = Find(e->u);
43         int y = Find(e->v);
44         if(x != y) {
45             P[y] = x;
46             ans += e->w;
47         }
48     }
49     return ans;
50 }
51 };

```

4.4 Prim 最小生成树

```

1  int d[maxn][maxn];
2  int lowc[maxn];
3  int vis[maxn];
4
5  int prim(int n) {
6      int ans = 0;
7      memset(vis, 0, sizeof(vis));
8      for (int i = 2; i <= n; i++)
9          lowc[i] = d[1][i];
10     vis[1] = 1;
11     for (int i = 1; i < n; i++) {
12         int minc = INF;
13         int p = -1;
14         for (int j = 1; j <= n; j++) {
15             if (!vis[j] && minc > lowc[j]) {
16                 minc = lowc[j];
17                 p = j;
18             }
19         }
20         vis[p] = 1;
21         ans += minc;

```

```
22     for (int j = 1; j <= n; j++) {
23         if (!vis[j] && lowc[j] > d[p][j])
24             lowc[j] = d[p][j];
25     }
26 }
27 return ans;
28 }
```

4.5 SPFA 最短路

```
1  #include <queue>
2  #include <cstring>
3  #include <vector>
4  #define maxn 10007
5  #define INF 0x7FFFFFFF
6  using namespace std;
7  struct Edge{
8      int v,w;
9      Edge(int v,int w):v(v),w(w){}
10 };
11 int d[maxn];
12 bool inq[maxn];
13 vector<Edge> G[maxn];
14 void SPFA(int s){
15     queue<int> q;
16     memset(inq,0,sizeof(inq));
17     for(int i=0;i<maxn;++i)
18         d[i]=INF;
19     d[s]=0;
20     inq[s]=1;
21     q.push(s);
22     int u;
23     while(!q.empty()){
24         u=q.front();
25         q.pop();
26         inq[u]=0;
27         for(vector<Edge>::iterator e=G[u].begin();e!=G[u].end();++e) {
28             if(d[e->v]>d[u]+e->w){
29                 d[e->v]=d[u]+e->w;
30                 if(!inq[e->v]){
31                     q.push(e->v);
32                     inq[e->v]=1;
33                 }
34             }
35         }
36     }
37 }
```

4.6 dijkstra 最短路

```
1 #include <vector>
2 #include <queue>
3 #define INF 0x7FFFFFFF
4 #define maxn 1000
5 using namespace std;
6 class Dijkstra{
7 private:
8     struct HeapNode{
9         int u;
10        int d;
11        HeapNode(int u, int d) :u(u), d(d){}
12        bool operator < (const HeapNode &b) const{
13            return d > b.d;
14        }
15    };
16    struct Edge{
17        int v;
18        int w;
19        Edge(int v, int w) :v(v), w(w){}
20    };
21    vector<Edge>G[maxn];
22    bool vis[maxn];
23 public:
24    int d[maxn];
25    void clear(int n){
26        int i;
27        for(i=0;i<n;++i)
28            G[i].clear();
29        for(i=0;i<n;++i)
30            d[i] = INF;
31        memset(vis, 0, sizeof(vis));
32    }
33    void AddEdge(int u, int v, int w){
34        G[u].push_back(Edge(v, w));
35    }
36    void Run(int s){
37        int u;
38        priority_queue<HeapNode> q;
39        d[s] = 0;
40        q.push(HeapNode(s, 0));
41        while (!q.empty()){
42            u = q.top().u;
43            q.pop();
44            if (!vis[u]){
45                vis[u] = 1;
46                for (vector<Edge>::iterator e = G[u].begin(); e != G[u].end(); ++e)
47                    if (d[e->v] > d[u] + e->w){
48                        d[e->v] = d[u] + e->w;
49                        q.push(HeapNode(e->v, d[e->v]));
50                    }
51            }
52        }
53    }
54};
```

4.7 Floyd 任意两点间最短路

```
1 // #define inf maxn*maxw+10
2 for(int i = 0; i < n; i++) {
3     for(int j = 0; j < n; j++) {
4         d[i][j] = inf;
5     }
6 }
7 d[0][0] = 0;
8 for(int k = 0; k < n; k++) {
9     for(int i = 0; i < n; i++) {
10        for(int j = 0; j < n; j++) {
11            d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
12        }
13    }
14 }
```

4.8 Dinic 最大流

```
1 #include <queue>
2 #include <vector>
3 #include <cstring>
4
5 #define INF 0x7FFFFFFF
6 #define maxn 1010
7
8 using namespace std;
9 struct Edge{
10     int c,f;
11     unsigned v,flip;
12     Edge(unsigned v,int c,int f,unsigned flip):v(v),c(c),f(f),flip(flip){}
13 };
14
15 /*
16  *b: BFS使用 ,
17  *a: 可改进量 , 不会出现负数可改进量。
18  *p[v]: u到v的反向边, 即v到u的边。 *cur[u]: i开始搜索的位置 , 此位置前所有路已满载。 *s: 源点。
19  *t: 汇点 。
20  */
21
22 class Dinic{
23 private:
24     bool b[maxn];
25     int a[maxn];
26     unsigned p[maxn],cur[maxn],d[maxn];
27     vector<Edge> G[maxn];
28 public:
29     unsigned s,t;
30     void Init(unsigned n){
```

```
31     for(int i=0;i<=n;++i)
32         G[i].clear();
33 }
34 void AddEdge(unsigned u,unsigned v,int c){
35     G[u].push_back(Edge(v,c,0,G[v].size()));
36     G[v].push_back(Edge(u,0,0,G[u].size()-1)); //使用无向图时将0改为c即可
37 }
38 bool BFS(){
39     unsigned u,v;
40     queue<unsigned> q;
41     memset(b,0,sizeof(b));
42     q.push(s);
43     d[s]=0;
44     b[s]=1;
45     while(!q.empty()){
46         u=q.front();
47         q.pop();
48         for(auto it=G[u].begin();it!=G[u].end();++it) {
49             Edge &e=*it;
50             if(!b[e.v]&&e.c>e.f){
51                 b[e.v]=1;
52                 d[e.v]=d[u]+1;
53                 q.push(e.v);
54             }
55         }
56     }
57     return b[t];
58 }
59 int DFS(unsigned u,int a){
60     if(u==t || a==0)
61         return a;
62     int flow=0,f;
63     for(unsigned &i=cur[u];i<G[u].size();++i){
64         Edge &e=G[u][i];
65         if(d[u]+1==d[e.v]&&(f=DFS(e.v,min(a,e.c-e.f)))>0){
66             a-=f;
67             e.f+=f;
68             G[e.v][e.flip].f-=f;
69             flow+=f;
70             if(!a) break;
71         }
72     }
73     return flow;
74 }
75 int MaxFlow(unsigned s,unsigned t){
76     int flow=0;
77     this->s=s;
78     this->t=t;
79     while(BFS()){
80         memset(cur,0,sizeof(cur));
81         flow+=DFS(s,INF);
82     }
83     return flow;
84 }
```

```
85 };
```

4.9 2-SAT 问题

```
1  class TwoSAT{
2      private:
3          const static int maxm=maxn*2;
4
5          int S[maxm],c;
6          vector<int> G[maxm];
7
8          bool DFS(int u){
9              if(vis[u^1])
10                 return false;
11              if(vis[u])
12                 return true;
13              vis[u]=1;
14              S[c++]=u;
15              for(auto &v:G[u])
16                 if(!DFS(v))
17                     return false;
18              return true;
19          }
20
21      public:
22          int N;
23          bool vis[maxm];
24
25          void Clear(){
26              for(int i=2;i<(N+1)*2;++i)
27                  G[i].clear();
28              memset(vis,0,sizeof(bool)*(N+1)*2);
29          }
30
31          void AddClause(int x,int xv,int y,int yv){
32              x=x*2+xv;
33              y=y*2+yv;
34              G[x].push_back(y);
35              G[y].push_back(x);
36          }
37
38          bool Solve(){
39              for(int i=2;i<(N+1)*2;i+=2)
40                  if(!vis[i]&&!vis[i+1]){
41                      c=0;
42                      if(!DFS(i)){
43                          while(c>0)
44                              vis[S[--c]]=0;
45                          if(!DFS(i+1))
46                              return false;
47                      }
48                  }
```

```
49         return true;
50     }
51 };
```

第五章 数据结构

5.1 树状数组

```
1 void add(int i, int x) {
2     for(; i <= n; i += i & -i)
3         tree[i] += x;
4 }
5
6 int sum(int i) {
7     int ret = 0;
8     for(; i; i -= i & -i) ret += tree[i];
9     return ret;
10 }
```

5.2 二维树状数组

```
1 int N;
2 int c[maxn][maxn];
3
4 inline int lowbit(int t) {
5     return t & (-t);
6 }
7
8 void update(int x, int y, int v) {
9     for (int i=x; i<=N; i+=lowbit(i)) {
10         for (int j=y; j<=N; j+=lowbit(j)) {
11             c[i][j] += v;
12         }
13     }
14 }
15
16 int query(int x, int y) {
17     int s = 0;
18     for (int i=x; i>0; i-=lowbit(i)) {
19         for (int j=y; j>0; j-=lowbit(j)) {
20             s += c[i][j];
21         }
22     }
23     return s;
24 }
25
26 int sum(int x, int y, int xx, int yy) {
```



```

27     x--, y--;
28     return query(xx, yy) - query(xx, y) - query(x, yy) + query(x, y);
29 }

```

5.3 堆

```

1  const int N = 1000;
2
3  template <class T>
4  class Heap {
5      private:
6          T h[N];
7          int len;
8      public:
9          Heap() {
10             len = 0;
11         }
12         inline void push(const T& x) {
13             h[++len] = x;
14             std::push_heap(h+1, h+1+len, std::greater<T>());
15         }
16         inline T pop() {
17             std::pop_heap(h+1, h+1+len, std::greater<T>());
18             return h[len--];
19         }
20         inline T& top() {
21             return h[1];
22         }
23         inline bool empty() {
24             return len == 0;
25         }
26 };

```

5.4 RMQ

```

1  //A为原始数组, d[i][j]表示从i开始, 长度为(1<<j)的区间最小值
2
3  int A[maxn];
4  int d[maxn][30];
5
6  void init(int A[], int len) {
7      for (int i = 0; i < len; i++) d[i][0] = A[i];
8      for (int j = 1; (1 << j) <= len; j++) {
9          for (int i = 0; i + (1 << j) - 1 < len; i++) {
10             d[i][j] = min(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
11         }
12     }
13 }
14

```

```

15 int query(int l, int r) {
16     int p = 0;
17     while ((1 << (p + 1)) <= r - l + 1)p++;
18     return min(d[l][p], d[r - (1 << p) + 1][p]);
19 }

```

5.5 线段树

```

1 //A为原始数组, sum记录区间和, Add为懒惰标记
2
3 int A[maxn], sum[maxn << 2], Add[maxn << 2];
4
5 void pushup(int rt) {
6     sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
7 }
8
9 void pushdown(int rt, int l, int r) {
10     if (Add[rt]) {
11         int mid = (l + r) >> 1;
12         Add[rt << 1] += Add[rt];
13         Add[rt << 1 | 1] += Add[rt];
14         sum[rt << 1] += (mid - l + 1)*Add[rt];
15         sum[rt << 1 | 1] += (r - mid)*Add[rt];
16         Add[rt] = 0;
17     }
18 }
19
20 void build(int l, int r, int rt) {
21     if (l == r) {
22         sum[rt] = A[l];
23         return;
24     }
25     int mid = (l + r) >> 1;
26     build(l, mid, rt << 1);
27     build(mid + 1, r, rt << 1 | 1);
28     pushup(rt);
29 }
30
31 // 区间加值
32 void update(int L, int R, int val, int l, int r, int rt) {
33     if (L <= l && R >= r) {
34         Add[rt] += val;
35         sum[rt] += (r - l + 1)*val;
36         return;
37     }
38     pushdown(rt, l, r);
39     int mid = (l + r) >> 1;
40     if (L <= mid)update(L, R, val, l, mid, rt << 1);
41     if (R > mid)update(L, R, val, mid + 1, r, rt << 1 | 1);
42     pushup(rt);
43 }
44

```

```

45 //点修改
46 void update(int index, int val, int l, int r, int rt) {
47     if (l == r) {
48         sum[rt] = val;
49         return;
50     }
51     int mid = (l + r) >> 1;
52     if (index <= mid) update(index, val, l, mid, rt << 1);
53     else update(index, val, mid + 1, r, rt << 1 | 1);
54     pushup(rt);
55 }
56
57 //区间查询
58 int query(int L, int R, int l, int r, int rt) {
59     if (L <= l && R >= r) {
60         return sum[rt];
61     }
62     pushdown(rt, l, r);
63     int mid = (l + r) >> 1;
64     int ret = 0;
65     if (L <= mid) ret += query(L, R, l, mid, rt << 1);
66     if (R > mid) ret += query(L, R, mid + 1, r, rt << 1 | 1);
67     return ret;
68 }

```

5.6 ZKW 线段树

```

1  const int maxn = 100;
2  int M, T[maxn*2+2], n;
3
4  void build(int x) {
5      for(M=1; M<=n+1; M<=1);
6
7      for(int i=1; i<=n; i++)
8          scanf("%d", &T[i+M]);
9
10     for(int i=M-1; i; i--)
11         T[i] = T[i<<1] + T[i<<1|1];
12 }
13 void updata(int n, int val) {
14     T[n+=M] = val; //这个地方是单点修改的哟
15     for(n>>=1; n>>=1)
16         T[n] = T[n<<1] + T[n<<1|1];
17 }
18 int query(int l, int r) {
19     int ans = 0;
20     l = l+M-1, r = r+M+1;
21     for (; l^r^1; l>>=1, r>>=1) {
22         if (~l&1) ans += T[l^1];
23         if (r&1) ans += T[r^1];
24     }
25     return ans;

```

26 }

5.7 莫队算法

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int maxn = 1000005;
4  inline int read()
5  {
6      int x=0,f=1;char ch=getchar();
7      while(ch>'9' || ch<'0'){if(ch=='-')f=-1;ch=getchar();}
8      while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
9      return x*f;
10 }
11 int a[maxn],pos[maxn],c[maxn],Ans[maxn];
12 int ans,n,m;
13 struct query
14 {
15     int l,r,id;
16 }Q[maxn];
17 bool cmp(query a,query b)
18 {
19     if(pos[a.l]==pos[b.l])
20         return a.r<b.r;
21     return pos[a.l]<pos[b.l];
22 }
23 void Update(int x)
24 {
25     c[x]++;
26     if(c[x]==2)ans++;
27 }
28 void Delete(int x)
29 {
30     c[x]--;
31     if(c[x]==1)ans--;
32 }
33 int main()
34 {
35     n=read(),m=read(),m=read();
36     int sz =ceil(sqrt(1.0*n));
37     for(int i=1;i<=n;i++)
38     {
39         a[i]=read();
40         pos[i]=(i-1)/sz;
41     }
42     for(int i=1;i<=m;i++)
43     {
44         Q[i].l=read();
45         Q[i].r=read();
46         Q[i].id = i;
47     }
48     sort(Q+1,Q+1+m,cmp);
```

```
49  int L=1,R=0;ans=0;
50  for(int i=1;i<=m;i++)
51  {
52      int id = Q[i].id;
53      while(R<Q[i].r)R++,Update(a[R]);
54      while(L>Q[i].l)L--,Update(a[L]);
55      while(R>Q[i].r)Delete(a[R]),R--;
56      while(L<Q[i].l)Delete(a[L]),L++;
57      Ans[id]=ans;
58  }
59  for(int i=1;i<=m;i++)
60      printf("%d\n",Ans[i]);
61 }
```

第六章 字符串

6.1 前缀树

```
1 #include <cstring>
2
3 const int maxn = 10000*50+10;
4 const int max_stringlen = 26+2;
5 int trie[maxn][max_stringlen];
6 int val[maxn];
7 int trie_index;
8
9 int index_of(const char &c) {
10     return c - 'a';
11 }
12 void trie_init() {
13     trie_index = 0;
14     memset(val, 0, sizeof(val));
15     memset(trie, 0, sizeof(trie));
16 }
17 void trie_insert(char *s, int v) { //要求v!=0
18     int len = strlen(s);
19     int now = 0;
20     for (int i = 0; i < len; ++i) {
21         int idx = index_of(s[i]);
22         int &tr = trie[now][idx];
23         if (!tr) {
24             tr = ++trie_index;
25         }
26         now = tr;
27     }
28     val[now] += v;
29 }
```

6.2 后缀数组

```
1 //Author: CookiC
2 #include <cstring>
3 const int maxn = 10010;
4
5 char str[maxn];
6 int s[maxn], si[maxn], n;
7
```

```

8 void BuildSi(int m) {
9     //si为第一关键字排在第i位的后缀在s中的下标
10    //y为第二关键字排在第i位的后缀在s中的下标
11    //m为字母的种类
12    static int t1[maxn], t2[maxn], c[maxn];
13    int *x=t1, *y=t2;
14    int i;
15    //基数排序
16    memset(c, 0, sizeof(int)*m);
17    for(i=0; i<n; ++i) ++c[x[i]=s[i]];
18    for(i=1; i<m; ++i) c[i]+=c[i-1];
19    for(i=n-1; i>=0; --i) si[--c[x[i]]]=i;
20    for(int k=1; k<=n; k<=1) {
21        int p=0;
22
23        //第二关键字排序
24        for(i=n-k; i<n; ++i) y[p++]=i;
25        for(i=0; i<n; ++i) if(si[i]>=k) y[p++]=si[i]-k;
26
27        //第一关键字与第二关键字合并排序
28        memset(c, 0, sizeof(int)*m);
29        for(i=0; i<n; ++i)
30            ++c[x[y[i]]];
31        for(i=0; i<m; ++i)
32            c[i]+=c[i-1];
33        for(i=n-1; i>=0; --i)
34            si[--c[x[y[i]]]]=y[i];
35
36        //判断相邻元素是否等价，等价则标上同等大小的数字。
37        swap(x, y);
38        p=1;
39        x[si[0]]=0;
40        for(i=1; i<n; ++i)
41            x[si[i]]=y[si[i-1]]==y[si[i]]&&y[si[i-1]+k]==y[si[i]+k]?p-1:p++;
42        if(p>=n)
43            break;
44        m=p;
45    }
46 }

```

6.3 后缀自动机

```

1 //Author: CookiC
2 #include <cstring>
3 #define MAXN 10000
4
5 struct State{
6     State *f, *c[26];
7     int len;
8 };
9
10 State *root, *last, *cur;

```

```
11 State StatePool[MAXN];
12
13 State* NewState(int len){
14     cur->len=len;
15     cur->f=0;
16     memset(cur->c,0,sizeof(cur->c));
17     return cur++;
18 }
19
20 void Init(){
21     cur=StatePool;
22     last=StatePool;
23     root=NewState(0);
24 }
25
26 void Extend(int w){
27     State *p = last;
28     State *np = NewState(p->len+1);
29     while(p&&!p->c[w]) {
30         p->c[w] = np;
31         p = p->f;
32     }
33     if(!p) {
34         np->f=root;
35     } else {
36         State *q=p->c[w];
37         if(p->len+1==q->len) {
38             np->f=q;
39         } else {
40             State *nq = NewState(p->len+1);
41             memcpy(nq->c, q->c, sizeof(q->c));
42             nq->f = q->f;
43             q->f = nq;
44             np->f = nq;
45             while(p&&p->c[w]==q) {
46                 p->c[w]=nq;
47                 p=p->f;
48             }
49         }
50     }
51     last=np;
52 }
53
54 bool Find(char *s,int len) {
55     int i;
56     State *p=root;
57     for(i=0;i<len;++i) {
58         if(p->c[s[i]-'a']) {
59             p=p->c[s[i]-'a'];
60         } else {
61             return false;
62         }
63     }
64     return true;
```


65 }

6.4 最长回文子串

```
1 using namespace std;
2 const int MAXN=110010;
3 char Ma[MAXN*2];
4 int Mp[MAXN*2];
5 void Manacher(char s[],int len) {
6     int l=0;
7     Ma[l++] = '$';
8     Ma[l++] = '#';
9     for(int i=0;i<len;i++) {
10         Ma[l++] =s[i] ;
11         Ma[l++] = '#';
12     }
13     Ma[l]=0 ;
14     int mx=0,id=0;
15     for(int i=0;i<l;i++) {
16         Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
17         while(Ma[i+Mp[i]] == Ma[i-Mp[i]]) Mp[i]++;
18         if(i+Mp[i]>mx) {
19             mx=i+Mp[i];
20             id=i;
21         }
22     }
23 }
24 /*
25 * abaaba
26 * i:
27 * Ma[i]:$#a#b#a#a$b # a # *Mp[i]:11214127214 1 2 1
28 */
29 char s[MAXN];
30 int main() {
31     while(scanf( "%s", s)== 1) {
32         int len=strlen(s);
33         Manacher(s,len);
34         int ans=0;
35         for(int i=0;i<2*len+2;i++)
36             ans=max(ans, Mp[i] -1);
37         printf( "%d\n",ans );
38     }
39     return 0;
40 }
```

第七章 类

7.1 点类

```
1 struct point {
2     double x, y;
3     point() { };
4     point(double x, double y) :x(x), y(y) { }
5     point operator - (const point &b) const {
6         return point(x - b.x, y - b.y);
7     }
8     point operator + (const point &b) const {
9         return point(x + b.x, y + b.y);
10    }
11    point operator * (const double k) const {
12        return point(k * x, k * y);
13    }
14    point operator / (const double k) const {
15        return point(x / k, y / k);
16    }
17    double slope() {
18        return y / x;
19    }
20 };
```

7.2 分数类

```
1 inline int gcd(int x, int y){return y?gcd(y,x%y):x;}
2 struct frac{
3     int a,b;
4     frac(int _a,int _b){int g=gcd(_a,_b); a=_a/g; b=_b/g;}
5     frac(){}
6     friend bool operator<(frac a, frac b) {
7         return (ll)a.a*b.b < (ll)b.a*a.b;
8     }
9     friend bool operator==(frac a, frac b) {
10        return a.a==b.a && a.b==b.b;
11    }
12 }
```

7.3 矩阵

```

1  #define maxm 10
2  typedef long long LL;
3
4  const LL Mod=1e9+7;
5  struct Matrix {
6      int n, m;
7      LL mat[maxm][maxm];
8      void clear() {
9          memset(mat, 0, sizeof(mat));
10     }
11
12     Matrix(int n, int m) :n(n), m(m) {
13         //不要设置默认构造函数, 让编译器检查初始化遗漏
14         clear();
15     }
16
17     Matrix operator +(const Matrix &M) const {
18         Matrix res(n, m);
19         for (LL i = 0; i < n; ++i) for (LL j = 0; j < m; ++j) {
20             res.mat[i][j] = (mat[i][j] + M.mat[i][j]) % Mod;
21         }
22         return res;
23     }
24
25     Matrix operator *(const Matrix &M) const {
26         if (m != M.n){
27             std::cout << "Wrong!" << std::endl;
28             return Matrix(-1, -1);
29         }
30         Matrix res(n, M.m);
31         res.clear();
32         int i,j,k;
33         for (i = 0; i < n; ++i)
34             for (j = 0; j < M.m; ++j)
35                 for (k = 0; k < m; ++k) {
36                     res.mat[i][j] += mat[i][k] * M.mat[k][j]%Mod;
37                     res.mat[i][j] %= Mod;
38                 }
39         return res;
40     }
41
42     Matrix operator *(const LL &x) const {
43         Matrix res(n,m);
44         int i,j;
45         std::cout << n << 'x' << m << std::endl;
46         for (i = 0; i < n; ++i)
47             for (j = 0; j < m; ++j)
48                 res[i][j] = mat[i][j] * x % Mod;
49         return res;
50     }
51
52     Matrix operator ^(LL b) const { // 矩阵快速幂, 取余Mod
53         if (n != m)

```

```

53         return Matrix(-1, -1);
54     Matrix a(*this);
55     Matrix res(n, n);
56     res.clear();
57     for (LL i = 0; i < n; ++i)
58         res.mat[i][i] = 1;
59     for (; b; b >>= 1) {
60         if (b & 1) {
61             res = a * res;
62         }
63         a = a * a;
64     }
65     return res;
66 }
67
68 LL* operator [] (int i) {
69     return mat[i];
70 }
71
72 void Print() const {
73     for (int i = 0; i < n; ++i) {
74         for (int j = 0; j < m; ++j)
75             std::cout << mat[i][j] << ' ';
76         std::cout << '\n';
77     }
78 }
79 };

```

7.4 01 矩阵

```

1  #include <bitset>
2  #define maxn 1000
3  struct Matrix01{
4      int n,m;
5      std::bitset<maxn> a[maxn];
6      void Resize(int x,int y){
7          n=x;
8          m=y;
9      }
10     std::bitset<maxn>& operator [] (int n) {
11         return a[n];
12     }
13     void print(){
14         for(int i = 0; i < n; ++i)
15             std::cout << a[i] << std::endl;
16     }
17 };
18
19 Matrix01 operator & (Matrix01 &a,Matrix01 &b){ int i,j,k;
20     Matrix01 c;
21     c.Resize(a.n,b.m);
22     for(i = 0; i < a.n; ++i) {

```

```
23     c[i].reset();
24     for(j = 0; j < b.m; ++j)
25         if(a[i][j])
26             c[i]|=b[j];
27     }
28     return c;
29 }
```

第八章 黑科技

8.1 位运算

```
1 //去掉最后一位
2 x >> 1
3 //在最后加一个0
4 x << 1
5 //在最后加一个1
6 x << 1 + 1
7 //把最后一位变成1
8 x | 1
9 //把最后一位变成0
10 x | 1 - 1
11 //最后一位取反
12 x ^ 1
13 //把右数第k位变成1
14 x | (1 << (k-1))
15 //把右数第k位变成0
16 x & ~ (1 << (k-1))
17 //右数第k位取反
18 x ^ (1 << (k-1))
19 //取末三位
20 x & 7
21 //取末k位
22 x & (1 << k-1)
23 //取右数第k位
24 x >> (k-1) & 1
25 //把末k位变成1
26 x | (1 << k-1)
27 //末k位取反
28 x ^ (1 << k-1)
29 //把右边连续的1变成0
30 x & (x+1)
31 //x个1
32 ((1<<x)-1)
33 //二进制里1的数量
34 (x>>16)+(x&((1<<16)-1))
```

8.2 珂朵莉树 (Old Driver Tree)

```
1 #include <set>
2 #include <algorithm>
```

```
3
4 using LL = long long;
5
6 struct node {
7     int l, r;
8     mutable LL v;
9     node(int L, int R = -1, LL V = 0) : l(L), r(R), v(V) {}
10    bool operator < (const node& o) const {
11        return l < o.l;
12    }
13 };
14
15 std::set<node> s;
16
17 //分割SET 返回一个pos位置的迭代器
18 std::set<node>::iterator split(int pos) {
19     auto it = s.lower_bound(node(pos));
20     if (it != s.end() && it->l == pos) return it;
21     --it;
22     if (pos > it->r) return s.end();
23     int L = it->l, R = it->r;
24     LL V = it->v;
25     s.erase(it);
26     s.insert(node(L, pos - 1, V));
27     return s.insert(node(pos, R, V)).first;
28 }
29
30 //区间加值
31 void add(int l, int r, LL val=1) {
32     split(l);
33     auto itr = split(r+1), itl = split(l);
34     for (; itl != itr; ++itl) itl->v += val;
35 }
36
37 //区间赋值
38 void assign(int l, int r, LL val = 0) {
39     split(l);
40     auto itr = split(r+1), itl = split(l);
41     s.erase(itl, itr);
42     s.insert(node(l, r, val));
43 }
```