Nantong University ICPC Team Notebook (2018-19)

thirtiseven wanggann

目录

第一章　输入输出

1.1　取消同步

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

1.2　浮点数输出格式

```
//include <iomanip>

std::cout << std::fixed << std::setprecision(12) << ans << std::endl;
```

1.3　整型快速输入

```
//整型
//若读入不成功，返回false
//ios::sync_with_stdio(true)
//#include <cctype>
bool quick_in(int &x) {
    char c;
    while((c = getchar()) != EOF && !isdigit(c));
    if(c == EOF) {
        return false;
    }
    x = 0;
    do {
        x *= 10;
        x += c - '0';
    } while((c = getchar()) != EOF && isdigit(c));
    return true;
}

//带符号整型
//直接=返回值
//#include <cctype>
int read() {
    int x = 0, l = 1; char ch = getchar();
    while (!isdigit(ch)) {if (ch=='-') l=-1; ch=getchar();}
    while (isdigit(ch)) x=x*10+(ch^48),ch=getchar();
    return x*1;
}

template <class T>
inline bool Read(T &ret) {
    char c; int sgn;
    if(c=getchar(),c==EOF) return 0; //EOF
    while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    sgn=(c=='-') ?-1:1 ;
    ret=(c=='-') ?0:(c -'0');
    while(c=getchar(),c>='0'&&c<='9')
        ret=ret*10+(c-'0');
    ret*=sgn;
    return 1;
}
```

1.4 字符串快速输入

```
bool quick_in(char *p) {
    char c;
    while((c = getchar()) != EOF && (c == ' ' || c == '\n'));
    if(c == EOF) {
        return false;
    }
    do {
        *p++ = c;
    } while((c=getchar()) != EOF && c != ' ' && c != '\n');
    *p = 0;
    return true;
}
```

1.5 整型快速输出

```
void quick_out(int x) {
    char str[13];
    if(x) {
        int i;
        for(i = 0; x; ++i) {
            str[i] = x % 10 + '0';
            x /= 10;
        }
        while(i--) {
            putchar(str[i]);
        }
    } else {
        putchar('0');
    }
}
```

1.6 字符串快速输出

```
void quick_out(char *p) {
    while(*p) {
        putchar(*p++);
    }
}
```

1.7 python 输入

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

第二章　动态规划

2.1　背包问题

```cpp
const int maxn=100005;
int w[maxn],v[maxn],num[maxn];
int W,n;
int dp[maxn];

void ZOP(int weight, int value) {
    for(int i = W; i >= weight; i--) {
        dp[i]=std::max(dp[i],dp[i-weight]+value);
    }
}

void CP(int weight, int value){
    for(int i = weight; i <= W; i++) {
        dp[i] = std::max(dp[i], dp[i-weight]+value);
    }
}

void MP(int weight, int value, int cnt){
    if(weight*cnt >= W) {
         CP(weight, value);
    } else {
        for(int k = 1; k < cnt; k <<= 1) {
            ZOP(k*weight, k*value), cnt -= k;
        }
        ZOP(cnt*weight, cnt*value);
    }
}
```

2.2　最长单调子序列（nlogn）

```cpp
int arr[maxn], n;

template<class Cmp>
int LIS (Cmp cmp) {
    static int m, end[maxn];
    m = 0;
    for (int i=0; i<n; i++) {
        int pos = lower_bound(end, end+m, arr[i], cmp)-end;
        end[pos] = arr[i], m += pos==m;
    }
    return m;
}

bool greater1(int value) {
    return value >=1;
}

/*********
    std::cout << LIS(std::less<int>()) << std::endl;         //严格上升
    std::cout << LIS(std::less_equal<int>()) << std::endl;   //非严格上升
    std::cout << LIS(std::greater<int>()) << std::endl;      //严格下降
    std::cout << LIS(std::greater_equal<int>()) << std::endl;//非严格下降
    std::cout << count_if(a,a+7,std::greater1) << std::endl; //计数
**********/
```

第三章 数学

3.1 暴力判素数

```cpp
bool is_prime(int n) {
    if(n < 2) return false;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) return false;
    }
    return true;
}
```

3.2 埃氏筛

```cpp
bool prime_or_not[maxn];
for (int i = 2; i <= int(sqrt(maxn)); i++) {
    if (!prime_or_not[i]) {
        for (int j = i * i; j <= maxn; j = j+i) {
            prime_or_not[j] = 1;
        }
    }
}
```

3.3 欧拉筛

```cpp
#include <iostream>

const int maxn = 1234;
int flag[maxn], primes[maxn], totPrimes;

void euler_sieve(int n) {
    totPrimes = 0;
    memset(flag, 0, sizeof(flag));
    for (int i = 2; i <= n; i++) {
        if (!flag[i]) {
            primes[totPrimes++] = i;
        }
        for (int j = 0; i * primes[j] <= n; j++) {
            flag[i * primes[j]] = true;
            if (i % primes[j] == 0)
            break;
        }
    }
}
```

3.4 分解质因数

```
int cnt[maxn];//存储质因子是什么
int num[maxn];//该质因子的个数
int tot = 0;//质因子的数量
void factorization(int x)//输入x，返回cnt数组和num数组
{
    for(int i=2;i*i<=x;i++)
    {
        if(x%i==0)
        {
            cnt[tot]=i;
            num[tot]=0;
            while(x%i==0)
            {
                x/=i;
                num[tot]++;
            }
            tot++;
        }
    }
    if(x!=1)
    {
        cnt[tot]=x;
        num[tot]=1;
        tot++;
    }
}
```

3.5 暴力判回文数

```
bool is_palindrome(int bob) {
    int clare = bob, dave = 0;
    while (clare){
        dave = dave * 10 + clare % 10;
        clare /= 10;
    }
    if(bob == dave) {
        return true;
    } else {
        return false;
    }
}
```

3.6 最大公约数

```
ll gcd(ll a, ll b) {
    ll t;
    while(b != 0) {
        t=a%b;
        a=b;
        b=t;
    }
    return a;
}
```

3.7 最小公倍数

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

3.8 扩展欧几里得

```
void Gcd(int a,int b,int &d,int &x,int &y){
    if(!b) {
        d=a;
        x=1;
        y=0;
    } else {
        Gcd(b,a%b,d,y,x);
        y-=x*(a/b);
    }
}
```

3.9 中国剩余定理

```
LL Crt(LL *div, LL *rmd, LL len) {
    LL sum = 0;
    LL lcm = 1;
    //lcm为除数们的最小公倍数，若div互素，则如下一行计算lcm
    for (int i = 0; i < len; ++i)
        lcm *= div[i];
    for (int i = 0; i < len; ++i) {
        LL bsn = lcm / div[i];
        LL inv = Inv(bsn, div[i]);
        // dvd[i] = inv[i] * bsn[i] * rmd[i]
        LL dvd = MulMod(MulMod(inv, bsn, lcm), rmd[i], lcm);
        sum = (sum + dvd) % lcm;
    }
    return sum;
}
```

3.10 欧拉函数

```
LL EulerPhi(LL n){
    LL m = sqrt(n + 0.5);
    LL ans = n;
    for(LL i = 2; i <= m; ++i)
    if(n % i == 0) {
        ans = ans - ans / i;
    while(n % i == 0)
        n/=i;
    }
    if(n > 1)
        ans = ans - ans / n;
    return ans;
}
```

3.11 求逆元

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

3.12 C(n,m) mod p (n 很大 p 可以很大)

```
LL C(const LL &n, const LL &m, const int &pr) {
    LL ans = 1;
    for (int i = 1; i <= m; i++) {
        LL a = (n - m + i) % pr;
        LL b = i % pr;
        ans = (ans * (a * Inv(b, pr)) % pr) % pr;
    }
    return ans;
}
```

3.13 Lucas 定理

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

3.14 快速乘法取模

```cpp
//by sevenkplus
#define ll long long
#define ld long double
ll mul(ll x,ll y,ll z){return (x*y-(ll)(x/(ld)z*y+1e-3)*z+z)%z;}

//by Lazer2001
inline long long mmul (long long a, long long b, const long long& Mod) {
    long long lf = a * (b >> 25LL) % Mod * (1LL << 25) % Mod;
    long long rg = a * ( b & ( ( 1LL << 25 ) - 1 ) ) % Mod ;
    return (lf + rg) % Mod ;
}
```

3.15 快速幂取模

```cpp
using LL = long long;

LL PowMod(LL a, LL b, const LL &Mod) {
    a %= Mod;
    LL ans = 1;
    while(b) {
        if (b & 1){
            ans = (ans * a) % Mod;
        }
        a = (a * a) % Mod;
        b >>= 1;
    }
    return ans;
}
```

3.16 计算从 C(n, 0) 到 C(n, p) 的值

```cpp
//by Yuhao Du
int p;
std::vector<int> gao(int n) {
    std::vector<int> ret(p+1,0);
    if (n==0) {
        ret[0]=1;
    } else if (n%2==0) {
        std::vector<int> c = gao(n/2);
        for(int i = 0; i <= p+1; i++) {
            for(int j = 0; j <= p+1; j++) {
                if (i+j<=p) ret[i+j]+=c[i]*c[j];
            }
        }
    } else {
        std::vector<int> c = gao(n-1);
        for(int i = 0; i <= p+1; i++) {
            for(int j = 0; j <= 2; j++) {
                if (i+j<=p) ret[i+j]+=c[i];
            }
        }
    }
    return ret;
}
```

3.17 二分分数树（Stern-Brocot Tree)

```
//Author:CookiC
//未做模板调整，请自行调整
#include <cmath>
#define LL long long
#define LD long double

void SternBrocot(LD X, LL &A, LL &B) {
    A=X+0.5;
    B=1;
    if(A==X)
        return;
    LL la=X, lb=1, ra=X+1, rb=1;
    long double C=A, a, b, c;
    do {
        a = la+ra;
        b = lb+rb;
        c = a/b;
        if(std::abs(C-X) > std::abs(c-X)) {
            A=a;
            B=b;
            C=c;
            if(std::abs(X-C) < 1e-10) {
                break;
            }
        }
        if(X<c) {
            ra=a;
            rb=b;
        } else {
            la=a;
            lb=b;
        }
    } while(lb+rb<=1e5);
}
```

第四章 图论

4.1 并查集

```
int fa[N];

void init(int n) {
    for (int i = 1; i <= n; i++) fa[i] = i;
}

int find(int u) {
    return fa[u] == u ? fa[u] : fa[u] = find(fa[u]);
}

void unin(int u, int v) {
    fa[find(v)] = find(u);
}
```

4.2 可撤销并查集（按秩合并）

```
#include <iostream>
#include <stack>
#include <utility>

class UFS {
    private:
        int *fa, *rank;
        std::stack <std::pair <int*, int> > stk ;
    public:
        UFS() {}
        UFS(int n) {
            fa = new int[(const int)n + 1];
            rank = new int[(const int)n + 1];
            memset (rank, 0, n+1);
            for (int i = 1; i <= n; ++i) {
                fa [i] = i;
            }
        }
        inline int find(int x) {
            while (x ^ fa[x]) {
                x = fa[x];
            }
            return x ;
        }
        inline int Join (int x, int y) {
            x = find(x), y = find(y);
            if (x == y) {
                return 0;
            }
            if (rank[x] <= rank[y]) {
                stk.push(std::make_pair (fa + x, fa[x]));
                fa[x] = y;
                if (rank[x] == rank[y]) {
                    stk.push(std::make_pair (rank + y, rank[y]));
                    ++rank[y];
                    return 2;
                }
                return 1 ;
            }
            stk.push(std::make_pair(fa + y, fa [y]));
            return fa[y] = x, 1;
        }
        inline void Undo ( )  {
            *stk.top( ).first = stk.top( ).second ;
            stk.pop( ) ;
        }
}T;
```

4.3 Kruskal 最小生成树

```
#include <vector>
#include <algorithm>

#define maxm 1000
#define maxn 1000

class Kruskal {
    struct UdEdge {
        int u, v, w;
        UdEdge(){}
        UdEdge(int u,int v,int w):u(u), v(v), w(w){}
    };
    int N, M;
    UdEdge pool[maxm];
    UdEdge *E[maxm];
    int P[maxn];
    int Find(int x){
        if(P[x] == x)
            return x;
        return P[x] = Find(P[x]);
    }
    public:
    static bool cmp(const UdEdge *a, const UdEdge *b) {
        return a->w < b->w;
    }
    void Clear(int n) {
        N = n;
        M = 0;
    }
    void AddEdge(int u, int v, int w) {
        pool[M] = UdEdge(u, v, w);
        E[M] = &pool[M];
        ++M;
    }
    int Run() {
        int i, ans=0;
        for(i = 1; i <= N; ++i)
            P[i] = i;
        std::sort(E, E+M, cmp);
        for(i = 0; i < M; ++i) {
            UdEdge *e = E[i];
            int x = Find(e->u);
            int y = Find(e->v);
            if(x != y) {
                P[y] = x;
                ans += e->w;
            }
        }
        return ans;
    }
};
```

4.4 Prim 最小生成树

```
int d[maxn][maxn];
int lowc[maxn];
int vis[maxn];

int prim(int n) {
    int ans = 0;
    memset(vis, 0, sizeof(vis));
    for (int i = 2; i <= n; i++)
        lowc[i] = d[1][i];
    vis[1] = 1;
    for (int i = 1; i < n; i++) {
        int minc = INF;
        int p = -1;
        for (int j = 1; j <= n; j++) {
            if (!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        }
        vis[p] = 1;
        ans += minc;
        for (int j = 1; j <= n; j++) {
            if (!vis[j] && lowc[j] > d[p][j])
                lowc[j] = d[p][j];
        }
    }
    return ans;
}
```

4.5 SPFA 最短路

```
#include <queue>
#include <cstring>
#include <vector>
#define maxn 10007
#define INF 0x7FFFFFFF
using namespace std;
struct Edge{
    int v,w;
    Edge(int v,int w):v(v),w(w){}
};
int d[maxn];
bool inq[maxn];
vector<Edge> G[maxn];
void SPFA(int s){
    queue<int> q;
    memset(inq,0,sizeof(inq));
    for(int i=0;i<maxn;++i)
        d[i]=INF;
    d[s]=0;
    inq[s]=1;
    q.push(s);
    int u;
    while(!q.empty()){
        u=q.front();
        q.pop();
        inq[u]=0;
        for(vector<Edge>::iterator e=G[u].begin();e!=G[u].end();++e) {
            if(d[e->v]>d[u]+e->w){
                d[e->v]=d[u]+e->w;
                if(!inq[e->v]){
                    q.push(e->v);
                    inq[e->v]=1;
                }
            }
        }
    }
}
```

4.6 dijkstra 最短路

```cpp
#include <vector>
#include <queue>
#define INF 0x7FFFFFFF
#define maxn 1000
using namespace std;
class Dijkstra{
private:
    struct HeapNode{
        int u;
        int d;
        HeapNode(int u, int d) :u(u), d(d){}
        bool operator < (const HeapNode &b) const{
            return d > b.d;
        }
    };
    struct Edge{
        int v;
        int w;
        Edge(int v, int w) :v(v), w(w){}
    };
    vector<Edge>G[maxn];
    bool vis[maxn];
public:
    int d[maxn];
    void clear(int n){
        int i;
        for(i=0;i<n;++i)
            G[i].clear();
        for(i=0;i<n;++i)
            d[i] = INF;
        memset(vis, 0, sizeof(vis));
    }
    void AddEdge(int u, int v, int w){
        G[u].push_back(Edge(v, w));
    }
    void Run(int s){
        int u;
        priority_queue<HeapNode> q;
        d[s] = 0;
        q.push(HeapNode(s, 0));
        while (!q.empty()){
            u = q.top().u;
            q.pop();
            if (!vis[u]){
                vis[u] = 1;
                for (vector<Edge>::iterator e = G[u].begin(); e != G[u].end(); ++e)
                    if (d[e->v] > d[u] + e->w){
                        d[e->v] = d[u] + e->w;
                        q.push(HeapNode(e->v, d[e->v]));
                    }
            }
        }
    }
};
```

4.7 Floyd 任意两点间最短路

```
//#define inf maxn*maxw+10
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        d[i][j] = inf;
    }
}
d[0][0] = 0;
for(int k = 0; k < n; k++) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
```

4.8 Dinic 最大流

```
#include <queue>
#include <vector>
#include <cstring>

#define INF 0x7FFFFFFF
#define maxn 1010

using namespace std;
struct Edge{
    int c,f;
    unsigned v,flip;
    Edge(unsigned v,int c,int f,unsigned flip):v(v),c(c),f(f),flip(flip){}
};

/*
*b:BFS使用 ,
*a:可改进量 ， 不会出现负数可改进量。
*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。
*t:汇点 。
*/

class Dinic{
private:
    bool b[maxn];
    int a[maxn];
    unsigned p[maxn],cur[maxn],d[maxn];
    vector<Edge> G[maxn];
public:
    unsigned s,t;
    void Init(unsigned n){
        for(int i=0;i<=n;++i)
            G[i].clear();
    }
    void AddEdge(unsigned u,unsigned v,int c){
        G[u].push_back(Edge(v,c,0,G[v].size()));
        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
    }
    bool BFS(){
        unsigned u,v;
        queue<unsigned> q;
        memset(b,0,sizeof(b));
        q.push(s);
        d[s]=0;
        b[s]=1;
        while(!q.empty()){
            u=q.front();
            q.pop();
            for(auto it=G[u].begin();it!=G[u].end();++it) {
                Edge &e=*it;
                if(!b[e.v]&&e.c>e.f){
                    b[e.v]=1;
                    d[e.v]=d[u]+1;
                    q.push(e.v);
                }
            }
        }
        return b[t];
    }
    int DFS(unsigned u,int a){
        if(u==t||a==0)
            return a;
        int flow=0,f;
        for(unsigned &i=cur[u];i<G[u].size();++i){
            Edge &e=G[u][i];
            if(d[u]+1==d[e.v]&&(f=DFS(e.v,min(a,e.c-e.f)))>0){
                a-=f;
                e.f+=f;
                G[e.v][e.flip].f-=f;
                flow+=f;
                if(!a) break;
            }
        }
        return flow;
    }
    int MaxFlow(unsigned s,unsigned t){
        int flow=0;
        this->s=s;
        this->t=t;
        while(BFS()){
            memset(cur,0,sizeof(cur));
            flow+=DFS(s,INF);
        }
        return flow;
    }
```

```
};
```

4.9 2-SAT 问题

```
class TwoSAT{
    private:
        const static int maxm=maxn*2;

        int S[maxm],c;
        vector<int> G[maxm];

        bool DFS(int u){
            if(vis[u^1])
                return false;
            if(vis[u])
                return true;
            vis[u]=1;
            S[c++]=u;
            for(auto &v:G[u])
                if(!DFS(v))
                    return false;
            return true;
        }

    public:
        int N;
        bool vis[maxm];

        void Clear(){
            for(int i=2;i<(N+1)*2;++i)
                G[i].clear();
            memset(vis,0,sizeof(bool)*(N+1)*2);
        }

        void AddClause(int x,int xv,int y,int yv){
            x=x*2+xv;
            y=y*2+yv;
            G[x].push_back(y);
            G[y].push_back(x);
        }

        bool Solve(){
            for(int i=2;i<(N+1)*2;i+=2)
                if(!vis[i]&&!vis[i+1]){
                    c=0;
                    if(!DFS(i)){
                        while(c>0)
                            vis[S[--c]]=0;
                        if(!DFS(i+1))
                            return false;
                    }
                }
            return true;
        }
    };
```

第五章　数据结构

5.1　树状数组

```cpp
void add(int i, int x) {
    for(;i <= n; i += i & -i)
        tree[i] += x;
}

int sum(int i) {
    int ret = 0;
    for(; i; i -= i & -i) ret += tree[i];
    return ret;
}
```

5.2　二维树状数组

```cpp
int N;
int c[maxn][maxn];

inline int lowbit(int t) {
    return t&(-t);
}

void update(int x, int y, int v) {
    for (int i=x; i<=N; i+=lowbit(i)) {
        for (int j=y; j<=N; j+=lowbit(j)) {
            c[i][j]+=v;
        }
    }
}

int query(int x, int y) {
    int s = 0;
    for (int i=x; i>0; i-=lowbit(i)) {
        for (int j=y; j>0; j-=lowbit(j)) {
            s += c[i][j];
        }
    }
    return s;
}

int sum(int x, int y, int xx, int yy) {
    x--, y--;
    return query(xx, yy) - query(xx, y) - query(x, yy) + query(x, y);
}
```

5.3 堆

```
const int N = 1000;

template <class T>
class Heap {
    private:
        T h[N];
        int len;
    public:
        Heap() {
            len = 0;
        }
        inline void push(const T& x) {
            h[++len] = x;
            std::push_heap(h+1, h+1+len, std::greater<T>());
        }
        inline T pop() {
            std::pop_heap(h+1, h+1+len, std::greater<T>());
            return h[len--];
        }
        inline T& top() {
            return h[1];
        }
        inline bool empty() {
            return len == 0;
        }
};
```

5.4 RMQ

```
//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

int A[maxn];
int d[maxn][30];

void init(int A[], int len) {
    for (int i = 0; i < len; i++)d[i][0] = A[i];
    for (int j = 1; (1 << j) <= len; j++) {
        for (int i = 0; i + (1 << j) - 1 < len; i++) {
            d[i][j] = min(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
        }
    }
}

```

```cpp
int query(int l, int r) {
    int p = 0;
    while ((1 << (p + 1)) <= r - l + 1)p++;
    return min(d[l][p], d[r - (1 << p) + 1][p]);
}
```

5.5 线段树

```cpp
//A为原始数组，sum记录区间和，Add为懒惰标记

int A[maxn], sum[maxn << 2], Add[maxn << 2];

void pushup(int rt) {
    sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}

void pushdown(int rt, int l, int r) {
    if (Add[rt]) {
        int mid = (l + r) >> 1;
        Add[rt << 1] += Add[rt];
        Add[rt << 1 | 1] += Add[rt];
        sum[rt << 1] += (mid - l + 1)*Add[rt];
        sum[rt << 1 | 1] += (r - mid)*Add[rt];
        Add[rt] = 0;
    }
}

void build(int l, int r, int rt) {
    if (l == r) {
        sum[rt] = A[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
    pushup(rt);
}

//区间加值
void update(int L, int R, int val, int l, int r, int rt) {
    if (L <= l && R >= r) {
        Add[rt] += val;
        sum[rt] += (r - l + 1)*val;
        return;
    }
    pushdown(rt, l, r);
    int mid = (l + r) >> 1;
    if (L <= mid)update(L, R, val, l, mid, rt << 1);
    if (R > mid)update(L, R, val, mid + 1, r, rt << 1 | 1);
    pushup(rt);
}

```

```
//点修改
void update(int index, int val, int l, int r, int rt) {
    if (l == r) {
        sum[rt] = val;
        return;
    }
    int mid = (l + r) >> 1;
    if (index <= mid)update(index, val, l, mid, rt << 1);
    else update(index, val, mid + 1, r, rt << 1 | 1);
    pushup(rt);
}

//区间查询
int query(int L, int R, int l, int r, int rt) {
    if (L <= l && R >= r) {
        return sum[rt];
    }
    pushdown(rt, l, r);
    int mid = (l + r) >> 1;
    int ret = 0;
    if (L <= mid)ret += query(L, R, l, mid, rt << 1);
    if (R > mid)ret += query(L, R, mid + 1, r, rt << 1 | 1);
    return ret;
}
```

5.6 ZKW 线段树

```
const int maxn = 100;
int M, T[maxn*2+2], n;

void build(int x) {
    for(M=1; M<=n+1; M<<=1);

    for(int i=1; i<=n; i++)
        scanf("%d",&T[i+M]);

    for(int i=M-1;i;i--)
        T[i]=T[i<<1]+T[i<<1|1];
}
void updata(int n,int val) {
    T[n+=M]=val;//这个地方是单点修改的哟
    for(n>>=1;n;n>>=1)
        T[n]=T[n<<1]+T[n<<1|1];
}
int query(int l,int r) {
    int ans=0;
    l=l+M-1, r=r+M+1;
    for (;l^r^1;l>>=1,r>>=1) {
        if (~l&1)ans+=T[l^1];
        if (r&1) ans+=T[r^1];
    }
    return ans;
}
```

5.7 莫队算法

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1000005;
inline int read()
{
    int x=0,f=1;char ch=getchar();
    while(ch>'9'||ch<'0'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
    return x*f;
}
int a[maxn],pos[maxn],c[maxn],Ans[maxn];
int ans,n,m;
struct query
{
    int l,r,id;
}Q[maxn];
bool cmp(query a,query b)
{
    if(pos[a.l]==pos[b.l])
        return a.r<b.r;
    return pos[a.l]<pos[b.l];
}
void Update(int x)
{
    c[x]++;
    if(c[x]==2)ans++;
}
void Delete(int x)
{
    c[x]--;
    if(c[x]==1)ans--;
}
int main()
{
    n=read(),m=read(),m=read();
    int sz =ceil(sqrt(1.0*n));
    for(int i=1;i<=n;i++)
    {
        a[i]=read();
        pos[i]=(i-1)/sz;
    }
    for(int i=1;i<=m;i++)
    {
        Q[i].l=read();
        Q[i].r=read();
        Q[i].id = i;
    }
    sort(Q+1,Q+1+m,cmp);
    int L=1,R=0;ans=0;
    for(int i=1;i<=m;i++)
    {
        int id = Q[i].id;
        while(R<Q[i].r)R++,Update(a[R]);
        while(L>Q[i].l)L--,Update(a[L]);
        while(R>Q[i].r)Delete(a[R]),R--;
        while(L<Q[i].l)Delete(a[L]),L++;
        Ans[id]=ans;
    }
    for(int i=1;i<=m;i++)
        printf("%d\n",Ans[i]);
}
```

第六章　字符串

6.1　前缀树

```cpp
#include <cstring>

const int maxn = 10000*50+10;
const int max_stringlen = 26+2;
int trie[maxn][max_stringlen];
int val[maxn];
int trie_index;

int index_of(const char &c) {
    return c - 'a';
}
void trie_init() {
    trie_index = 0;
    memset(val, 0, sizeof(val));
    memset(trie, 0, sizeof(trie));
}
void trie_insert(char *s, int v) { //要求v!=0
    int len = strlen(s);
    int now = 0;
    for (int i = 0; i < len; ++i) {
        int idx = index_of(s[i]);
        int &tr = trie[now][idx];
        if (!tr) {
            tr = ++trie_index;
        }
        now = tr;
    }
    val[now] += v;
}
```

6.2　后缀数组

```cpp
//Author:CookiC
#include <cstring>
const int maxn = 10010;

char str[maxn];
int s[maxn], si[maxn], n;

```

```
void BuildSi(int m) {
    //si为第一关键字排在第i位的后缀在s中的下标
    //y为第二关键字排在第i位的后缀在s中的下标
    //m为字母的种类
    static int t1[maxn], t2[maxn], c[maxn];
    int *x=t1, *y=t2;
    int i;
    //基数排序
    memset(c, 0, sizeof(int)*m);
    for(i=0; i<n; ++i)  ++c[x[i]=s[i]];
    for(i=1; i<m; ++i)  c[i]+=c[i-1];
    for(i=n-1; i>=0; --i)   si[--c[x[i]]]=i;
    for(int k=1; k<=n; k<<=1) {
        int p=0;

        //第二关键字排序
        for(i=n-k;i<n;++i)  y[p++]=i;
        for(i=0;i<n;++i)    if(si[i]>=k)    y[p++]=si[i]-k;

        //第一关键字与第二关键字合并排序
        memset(c,0,sizeof(int)*m);
        for(i=0;i<n;++i)
            ++c[x[y[i]]];
        for(i=0;i<m;++i)
            c[i]+=c[i-1];
        for(i=n-1;i>=0;--i)
            si[--c[x[y[i]]]]=y[i];

        //判断相邻元素是否等价, 等价则标上同等大小的数字。
        swap(x,y);
        p=1;
        x[si[0]]=0;
        for(i=1;i<n;++i)
            x[si[i]]=y[si[i-1]]==y[si[i]]&&y[si[i-1]+k]==y[si[i]+k]?p-1:p++;
        if(p>=n)
            break;
        m=p;
    }
}
```

6.3 后缀自动机

```
//Author:CookiC
#include<cstring>
#define MAXN 10000

struct State{
    State *f,*c[26];
    int len;
};

State *root,*last,*cur;
State StatePool[MAXN];

State* NewState(int len){
    cur->len=len;
    cur->f=0;
    memset(cur->c,0,sizeof(cur->c));
    return cur++;
}

void Init(){
    cur=StatePool;
    last=StatePool;
    root=NewState(0);
}

void Extend(int w){
    State *p = last;
    State *np = NewState(p->len+1);
    while(p&&!p->c[w]) {
        p->c[w] = np;
        p = p->f;
    }
    if(!p) {
        np->f=root;
    } else {
        State *q=p->c[w];
        if(p->len+1==q->len) {
            np->f=q;
        } else {
            State *nq = NewState(p->len+1);
            memcpy(nq->c, q->c, sizeof(q->c));
            nq->f = q->f;
            q->f = nq;
            np->f = nq;
            while(p&&p->c[w]==q) {
                p->c[w]=nq;
                p=p->f;
            }
        }
    }
    last=np;
}

bool Find(char *s,int len) {
    int i;
    State *p=root;
    for(i=0;i<len;++i) {
        if(p->c[s[i]-'a']) {
            p=p->c[s[i]-'a'];
        } else {
            return false;
        }
    }
    return true;
}
```

6.4 最长回文子串

```
using namespace std;
const int MAXN=110010;
char Ma[MAXN*2];
int Mp[MAXN*2];
void Manacher(char s[],int len) {
    int l=0;
    Ma[l++] ='$';
    Ma[l++] ='#';
    for(int i=0;i<len;i++) {
        Ma[l++] =s[i] ;
        Ma[l++] ='#';
    }
    Ma[l]=0 ;
    int mx=0,id=0;
    for(int i=0;i<l;i++) {
        Mp[i]=mx>i?min(Mp[2*id−i],mx−i):1;
        while(Ma[i+Mp[i]] == Ma[i−Mp[i]]) Mp[i]++;
        if(i+Mp[i]>mx) {
            mx=i+Mp [i];
            id=i;
        }
    }
}
/*
* abaaba
* i:
* Ma[i]:$#a#b#a#a$b # a # *Mp[i]:11214127214 1 2 1
*/
char s[MAXN];
int main() {
    while(scanf( "%s", s)== 1) {
        int len=strlen(s);
        Manacher(s,len);
        int ans=0;
        for(int i=0;i<2*len+2;i++)
            ans=max(ans, Mp[i] −1);
        printf( "%d\n␣",ans );
    }
    return 0;
}
```

第七章　类

7.1　点类

```cpp
struct point {
    double x, y;
    point() { };
    point(double x, double y) :x(x), y(y) { }
    point operator - (const point &b) const {
        return point(x - b.x, y - b.y);
    }
    point operator + (const point &b) const {
        return point(x + b.x, y + b.y);
    }
    point operator * (const double k) const {
        return point(k * x, k * y);
    }
    point operator / (const double k) const {
        return point(x / k, y / k);
    }
    double slope() {
        return y / x;
    }
};
```

7.2　分数类

```cpp
inline int gcd(int x,int y){return y?gcd(y,x%y):x;}
struct frac{
    int a,b;
    frac(int _a,int _b){int g=gcd(_a,_b); a=_a/g; b=_b/g;}
    frac(){}
    friend bool operator<(frac a, frac b) {
        return (ll)a.a*b.b < (ll)b.a*a.b;
    }
    friend bool operator==(frac a, frac b) {
        return a.a==b.a && a.b==b.b;
    }
}
```

7.3 矩阵

```cpp
#define maxm 10
typedef long long LL;

const LL Mod=1e9+7;
struct Matrix {
    int n, m;
    LL mat[maxm][maxm];
    void clear() {
        memset(mat, 0, sizeof(mat));
    }

    Matrix(int n, int m) :n(n), m(m) {
        //不要设置默认构造函数，让编译器检查初始化遗漏
        clear();
    }

    Matrix operator +(const Matrix &M) const {
        Matrix res(n, m);
        for (LL i = 0; i < n; ++i) for (LL j = 0; j < m; ++j) {
            res.mat[i][j] = (mat[i][j] + M.mat[i][j]) % Mod;
        }
        return res;
    }

    Matrix operator *(const Matrix &M) const {
        if (m != M.n){
            std::cout << "Wrong!" << std::endl;
            return Matrix(-1, -1);
        }
        Matrix res(n, M.m);
        res.clear();
        int i,j,k;
        for (i = 0; i < n; ++i)
            for (j = 0; j < M.m; ++j)
                for (k = 0; k < m; ++k) {
                    res.mat[i][j] += mat[i][k] * M.mat[k][j]%Mod;
                    res.mat[i][j] %= Mod;
                }
        return res;
    }
    Matrix operator *(const LL &x) const {
        Matrix res(n,m);
        int i,j;
        std::cout << n << '␣' << m << std::endl;
        for (i = 0; i < n; ++i)
            for (j = 0; j < m; ++j)
                res[i][j] = mat[i][j] * x % Mod;
        return res;
    }

    Matrix operator ^(LL b) const { // 矩阵快速幂 ， 取余Mod
        if (n != m)
```

```cpp
            return Matrix(-1, -1);
        Matrix a(*this);
        Matrix res(n, n);
        res.clear();
        for (LL i = 0; i < n; ++i)
            res.mat[i][i] = 1;
        for (; b; b >>= 1) {
            if (b & 1) {
                res = a * res;
            }
            a = a * a;
        }
        return res;
    }

    LL* operator [](int i) {
        return mat[i];
    }

    void Print() const {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j)
                std::cout << mat[i][j] << '␣';
            std::cout << '\n';
        }
    }
};
```

7.4 01 矩阵

```cpp
#include <bitset>
#define maxn 1000
struct Matrix01{
    int n,m;
    std::bitset<maxn> a[maxn];
    void Resize(int x,int y){
        n=x;
        m=y;
    }
    std::bitset<maxn>& operator [] (int n) {
        return a[n];
    }
    void print(){
        for(int i = 0; i < n; ++i)
            std::cout << a[i] << std::endl;
        }
};

Matrix01 operator & (Matrix01 &a,Matrix01 &b){ int i,j,k;
    Matrix01 c;
    c.Resize(a.n,b.m);
    for(i = 0; i < a.n; ++i) {
```

```
    c[i].reset();
    for(j = 0; j < b.m; ++j)
        if(a[i][j])
            c[i]|=b[j];
        }
    return c;
}
```

第八章　黑科技

8.1　位运算

```
//去掉最后一位
 x >> 1
//在最后加一个0
x << 1
//在最后加一个1
x << 1 + 1
//把最后一位变成1
 x | 1
//把最后一位变成0
x | 1 - 1
//最后一位取反
 x ^ 1
//把右数第k位变成1
 x | (1 <<(k-1))
//把右数第k位变成0
 x & ~ (1 << (k-1))
//右数第k位取反
x ^ (1 << (k-1))
//取末三位
x & 7
//取末k位
 x & (1 << k-1)
//取右数第k位
 x >>(k-1) & 1
//把末k位变成1
x |(1 << k-1)
//末k位取反
 x ^ (1 << k-1)
//把右边连续的1变成0
x & (x+1)
//x个1
((1<<x-1)
//二进制里1的数量
(x>>16)+(x&((1<<16)-1))
```

8.2　珂朵莉树（Old Driver Tree)

```
#include <set>
#include <algorithm>
```

```cpp

using LL = long long;

struct node {
    int l, r;
    mutable LL v;
    node(int L, int R = -1, LL V = 0) : l(L), r(R), v(V) {}
    bool operator < (const node& o) const {
        return l < o.l;
    }
};

std::set<node> s;

//分割SET 返回一个pos位置的迭代器
std::set<node>::iterator split(int pos) {
    auto it = s.lower_bound(node(pos));
    if (it != s.end() && it->l == pos) return it;
    --it;
    if (pos > it->r) return s.end();
    int L = it->l, R = it->r;
    LL V = it->v;
    s.erase(it);
    s.insert(node(L, pos - 1, V));
    return s.insert(node(pos, R, V)).first;
}

//区间加值
void add(int l, int r, LL val=1) {
    split(l);
    auto itr = split(r+1), itl = split(l);
    for (; itl != itr; ++itl) itl->v += val;
}

//区间赋值
void assign(int l, int r, LL val = 0) {
    split(l);
    auto itr = split(r+1), itl = split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, val));
}
```