Nantong University ICPC Team Notebook (2018-19)

thirtiseven wanggann

May 29 2018

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
**********/
```

2.3 单调队列优化 DP

```
//单调队列求区间最小值
int a[maxn], q[maxn], num[maxn] = {0};
int Fmin[maxn];
int k, n, head, tail;

void DPmin() {
    head = 1, tail = 0;
    for (int i = 1; i <= n; i++) {
        while (num[head] < i-k+1 && head <= tail) head++;
        while (a[i] <= q[tail] /*区间最大值此处改为>=*/ && head <= tail) tail--;
        num[++tail] = i;
        q[tail] = a[i];
        Fmin[i] = q[head];
    }
}
```

第三章　数学

3.1　暴力判素数

```cpp
bool is_prime(int  u) {
    if(u ==  0 || u ==  1) return false;
    if(u ==  2)      return true;
    if(u%2 == 0)      return false;
    for(int i=3; i <=  sqrt(u) ; i+=2)
        if(u%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

```
//如果GCD(a,b) = d, 则存在x, y, 使d =  ax +  by
// extended_euclid(a, b) = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
    int d;
    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    d = extended_euclid(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

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
}

int Inv(int a, int n) {
    int d, x, y;
    d = extended_euclid(a, n, x, y);
    if(d == 1)  return (x%n + n) % n;
    else     return -1; // no solution
}
```

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

```
//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

```
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

```
//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);
}
```

3.18 计算莫比乌斯函数

```
const int n=1<<20;
int mu[n];
int getMu() {
    for(int i=1;i<=n;i++) {
        int target=i==1?1:0;
        int delta=target-mu[i];
        mu[i]=delta;
        for(int j=i+i;j<=n;j+=i) {
            mu[j]+=delta;
        }
    }
}
```

3.19 博弈论

```
Nim Game
    最经典最基础的博弈.
    n堆石子,双方轮流从任意一堆石子中取出至少一个,不能取的人输.
    对于一堆x个石子的情况,容易用归纳法得到SG(x)=x.
    所以所有石子个数的异或和为0是必败态,否则为必胜态.

Bash Game
    每人最多一次只能取m个石子,其他规则同Nim Game.
    依旧数学归纳…SG(x)=xmod(m+1).

NimK Game
    每人一次可以从最多K堆石子中取出任意多个,其他规则同Nim Game.
    结论:在二进制下各位上各堆石子的数字之和均为(K+1)的倍数的话则为必败态,否则为必胜态.
    这个证明要回到原始的方法上去.
    补:这个游戏还可以推广,即一个由n个子游戏组成的游戏,每次可以在最多K个子游戏中进行操作.
    然后只要把结论中各堆石子的个数改为各个子游戏的SG值即可,证明也还是一样的.

Anti-Nim Game
    似乎又叫做Misère Nim.
    不能取的一方获胜,其他规则同Nim Game.
    关于所谓的"Anti-SG游戏"及"SJ定理"贾志鹏的论文上有详细说明,不过似乎遇到并不多.
    结论是一个状态是必胜态当且仅当满足以下条件之一:
    SG值不为0且至少有一堆石子数大于1;
    SG值为0且不存在石子数大于1的石子堆.

Staircase Nim
```

每人一次可以从第一堆石子中取走若干个,或者从其他石子堆的一堆中取出若干个放到左边一堆里(没有石子的石子堆不会消失),其他规则同Nim Game.

这个游戏的结论比较神奇:

当且仅当奇数编号堆的石子数异或和为0时为必败态.

简单的理解是从偶数编号堆中取石子对手又可以放回到奇数编号堆中,而且不会让对手不能移动.比较意识流,然而可以归纳证明.

Wythoff Game

有两堆石子,双方轮流从某一堆取走若干石子或者从两堆中取走相同数目的石子,不能取的人输.

容易推理得出对任意自然数k,都存在唯一的一个必败态使得两堆石子数差为k,设其为Pk=(ak,bk),表示石子数分别为ak,bk(ak<=bk).

那么ak为在Pk0(k0<k)中未出现过的最小自然数,bk=ak+k.

数学班的说,用Betty定理以及显然的单调性就可以推出神奇的结论:

ak=floor(k*5√+12),bk=floor(k*5√+32).

Take & Break

有n堆石子,双方轮流取出一堆石子,然后新增两堆规模更小的石子堆(可以没有石子),无法操作者输.

这个游戏似乎只能暴力SG,知道一下就好.

树上删边游戏

给出一个有n个结点的树，有一个点作为树的根节点,双方轮流从树中删去一条边边，之后不与根节点相连的部分将被移走,无法操作者输.

结论是叶子结点的SG值为0,其他结点SG值为其每个儿子结点SG值加1后的异或和,证明也并不复杂.

翻硬币游戏

n枚硬币排成一排，有的正面朝上，有的反面朝上。

游戏者根据某些约束翻硬币（如：每次只能翻一或两枚，或者每次只能翻连续的几枚），但他所翻动的硬币中，最右边的必须是从正面翻到反面。

谁不能翻谁输。

需要先开动脑筋把游戏转化为其他的取石子游戏之类的,然后用如下定理解决:

局面的 SG 值等于局面中每个正面朝上的棋子单一存在时的 SG 值的异或和。

无向图删边游戏

一个无向连通图，有一个点作为图的根。

游戏者轮流从图中删去边， 删去一条边后，不与根节点相连的部分将被移走。

谁无路可走谁输。

对于这个模型，有一个著名的定理——Fusion Principle:

我们可以对无向图做如下改动：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。 这样的改动不会影响图的 SG 值。

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;
                    }
                }
```

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

4.10 tarjan 强连通分量

```
1  #define maxn 5010
2  #define maxm 30000
3  int top;//栈顶位置
4  int Bcnt;//强连通分量编号
5  int Index;//时间顺序
6  int DFN[maxn];//时间戳
7  int LOW[maxn];
8  int belong[maxn];//顶点i属于哪个强连通分量
9  int Stack[maxn];//栈
10 int instack[maxn];//是否在栈内
11 int n,m;
12 struct node {
13     int to;
14     int next;
15 } edge[maxm];
16 int head[maxn];
17 bool Judge[maxn];
18 int ansi;
19 void init() {
20     std::fill(head,head+n+1,-1);
21     std::fill(DFN,DFN+n+1,0);
22     std::fill(Judge,Judge+n+1,true);
23     ansi=0;
24     top=0;
25     Bcnt=0;
26     Index=0;
27 }
28 void add(int a,int b) {
29     edge[ansi].to=b;
30     edge[ansi].next=head[a];
31     head[a]=ansi++;
32 }
33 void read() {
34     int a,b;
35     for(int i=0; i<m; i++) {
36         scanf("%d%d",&a,&b);
37         add(a,b);
38     }
39 }
40 void tarjan(int i) {
41     int j,k;
42     DFN[i]=LOW[i]=++Index;
43     instack[i]=true;
44     top++;
45     Stack[top]=i;
46     for (k=head[i]; k!=-1; k=edge[k].next) {
```

```
47          j=edge[k].to;
48          if (!DFN[j]) {//j未访问，用dfn值标记是否已访问过
49              tarjan(j);
50              if (LOW[j]<LOW[i])
51                  LOW[i]=LOW[j];
52          }
53          else if (instack[j] && DFN[j]<LOW[i])
54              LOW[i]=DFN[j];
55      }
56      if (DFN[i]==LOW[i]) {//dfn和low相等，递归打印强连通分量
57          Bcnt++;//强连通分量编号
58          do {
59              j=Stack[top--];
60              instack[j]=false;
61              belong[j]=Bcnt;
62          }
63          while (j!=i);
64      }
65  }
66  void judge() {
67      for(int i=1; i<=n; i++) {
68          for (int k=head[i]; k!=-1; k=edge[k].next) {
69              if(belong[i]!=belong[edge[k].to]) {
70                  Judge[belong[i]]=false;
71              }
72          }
73      }
74  }
75
76  void solve() {
77      init();
78      read();
79      for (int i=1; i<=n; i++) {
80          if (!DFN[i]) {
81              tarjan(i);
82          }
83      }
84      judge();
85      int ss;
86      for (int i=n; i>=1; i--) {
87          if(Judge[belong[i]]) {
88              ss=i;
89              break;
90          }
91      }
92      for (int i=1; i<=n; i++) {
93          if(Judge[belong[i]]) {
94              printf("%d",i);
95              if(i!=ss) {
96                  printf("␣");
97              }
98          }
99      }
100     printf("\n");
```

```
}
```

4.11 点双联通分量

```
//Author:CookiC
#include<stack>
#include<vector>
#define maxn 1000
using namespace std;

class BCC{
private:
    int clk, cnt;
    int pre[maxn];
    stack<int> s;

    int DFS(int u,int f){
        int lowu = pre[u] = ++clk;
        int child = 0;
        s.push(u);
        for (auto it = G[u].begin(); it != G[u].end(); ++it){
            int v = *it;
            if (!pre[v]){
                s.push(u);
                ++child;
                int lowv = DFS(v, u);
                if (lowu > lowv)
                    lowu = lowv;
                if (lowv >= pre[u]){
                    iscut[u] = 1;
                    ++cnt;
                    int x;
                    do{
                        x = s.top();
                        s.pop();
                        if (num[x] != cnt)
                            num[x] = cnt;
                    }while (x != u);
                }
            }
            else if (pre[v] < pre[u] && v != f){
                if (lowu > pre[v])
                    lowu = pre[v];
            }
        }
        if (f < 0 && child == 1)
            iscut[u] = 0;
        return lowu;
    }
public:
    vector<int> G[maxn];
    bool iscut[maxn];
```

```
    int num[maxn];

    void Clear(int n){
        for (int i = 0; i < n; ++i)
            G[i].clear();
    }

    void AddEdge(int u,int v){
        G[u].push_back(v);
        G[v].push_back(u);
    }

    void Find(){
        int i;
        memset(pre, 0, sizeof(pre));
        memset(iscut, 0, sizeof(iscut));
        memset(num,0,sizeof(num));
        clk = cnt = 0;
        for (i = 0; i < n; ++i)
            if (!pre[i]){
                while(!s.empty())
                    s.pop();
                DFS(i,-1);
            }
    }
};
```

4.12 边双联通分量

```
//Author: XieNaoban
//在求桥的基础上修改
#include<algorithm>
#include<cstring>
#include<vector>
#include<cmath>
#include<set>

class CutEdge {
private:
    int N;
    int clk, pre[Maxn];

    int DFS(int u, int f) {
        int lowu = pre[u] = ++clk;
        for (auto e = G[u].begin(); e != G[u].end(); ++e) {
            int v = *e;
            if (!pre[v]) {
                int lowv = DFS(v, u);
                lowu = min(lowu, lowv);
                if (lowv > pre[u]) {
                    Cut[u].insert(v);
                    Cut[v].insert(u);
```

```
                }
            }
            else if (pre[u] > pre[v]) {
                int cnt = 0; //重复边的处理
                for (const auto &e : G[u]) if (e == v) ++cnt;
                if (cnt > 1 || v != f) {
                    lowu = min(lowu, pre[v]);
                }
            }
        }
        return lowu;
    }

    void DFS2(int u, int id) {
        ID[u] = id;
        for (const auto &v : G[u]) if (!ID[v]) {
            if (Cut[u].count(v)) {
                ++Num;
                G2[id].push_back(Num);
                G2[Num].push_back(id);
                DFS2(v, Num);
            }
            else DFS2(v, id);
        }
    }

public:
    vector<int> G[Maxn];
    set<int> Cut[Maxn];

    vector<int> G2[Maxn]; //缩点后的图（以ID为结点）
    int ID[Maxn]; //每个点所在的子图
    int Num; //ID个数

    void Clear(int n) {
        N = n;
        memset(ID, 0, sizeof(ID));
        memset(pre, 0, sizeof(pre));
        for (int i = 1; i <= N; ++i) {
            G[i].clear();
            G2[i].clear();
            Cut[i].clear();
        }
        clk = 0;
        Num = 1;
    }

    void AddEdge(int u, int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }

    void Find() {
        for (int i = 1; i <= N; ++i)
```

```
78              if (!pre[i])
79                  DFS(i, -1);
80      }
81
82      //求边双联通部分
83      int BCC() { //要求先运行Find
84          DFS2(1, Num);
85          return Num;
86      }
87  };
```

第五章　数据结构

5.1　树状数组

```
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　差分数组

```
//Author:CookiC
/*
*a为原数组
*C为差分数组
*/
int a[]={0, 1, 1, 1, 1, 1, 1};
int N, C[maxn];

int Sum(unsigned n) {
    int sum = 0;
    while(n>0){
        sum += C[n];
        n -= lowbit(n);
    }
    return sum;
}

void Add(unsigned n, int d) {
    while(n<=N){
        C[n]+=d;
        n+=lowbit(n);
    }
}

void Add(int L,int R, int d) {
    Add(L,d);
```

```
27      Add(R+1,-d);
28  }
29
30  void Init() {
31      memset(C, 0, sizeof(C));
32      Add(1, a[1]);
33      for(int i=2; i<=N; ++i)
34          Add(i, a[i]-a[i-1]);
35  }
36
37  void Update() {
38      for(int i=1; i<=N; ++i)
39          a[i] = Sum(i);
40  }
```

5.3 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  }
```

5.4 堆

```cpp
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;
        }
};
```

5.5 RMQ

```cpp
//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]);
        }
    }
}

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]);
}
```

5.6 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;
}
```

5.7 Treap 树

```
typedef int value;

enum { LEFT, RIGHT };
struct node {
    int size, priority;
    value x, subtree;
    node *child[2];
    node(const value &x): size(1), x(x), subtree(x) {
        priority = rand();
        child[0] = child[1] = nullptr;
    }
};

inline int size(const node *a) { return a == nullptr ? 0 : a->size; }

inline void update(node *a) {
    if (a == nullptr) return;
    a->size = size(a->child[0]) + size(a->child[1]) + 1;
    a->subtree = a->x;
    if (a->child[LEFT] != nullptr) a->subtree = a->child[LEFT]->subtree + a->subtree;
    if (a->child[RIGHT] != nullptr) a->subtree = a->subtree + a->child[RIGHT]->subtree;
}

node *rotate(node *a, bool d) {
    node *b = a->child[d];
    a->child[d] = b->child[!d];
    b->child[!d] = a;
    update(a); update(b);
    return b;
}

node *insert(node *a, int index, const value &x) {
    if (a == nullptr && index == 0) return new node(x);
```

```
34      int middle = size(a->child[LEFT]);
35      bool dir = index > middle;
36      if (!dir) a->child[LEFT]  = insert(a->child[LEFT], index, x);
37      else      a->child[RIGHT] = insert(a->child[RIGHT], index - middle - 1, x);
38      update(a);
39      if (a->priority > a->child[dir]->priority) a = rotate(a, dir);
40      return a;
41  }
42
43  node *erase(node *a, int index) {
44      assert(a != nullptr);
45      int middle = size(a->child[LEFT]);
46      if (index == middle) {
47          if (a->child[LEFT] == nullptr && a->child[RIGHT] == nullptr) {
48              delete a;
49              return nullptr;
50          } else if (a->child[LEFT] == nullptr) a = rotate(a, RIGHT);
51          else if (a->child[RIGHT] == nullptr) a = rotate(a, LEFT);
52          else a = rotate(a, a->child[LEFT]->priority < a->child[RIGHT]->priority);
53          a = erase(a, index);
54      } else {
55          bool dir = index > middle;
56          if (!dir) a->child[LEFT] = erase(a->child[LEFT], index);
57          else      a->child[RIGHT] = erase(a->child[RIGHT], index - middle - 1);
58      }
59      update(a);
60      return a;
61  }
62
63  void modify(node *a, int index, const value &x) {
64      assert(a != nullptr);
65      int middle = size(a->child[LEFT]);
66      if (index == middle) a->x = x;
67      else {
68          bool dir = index > middle;
69          if (!dir) modify(a->child[LEFT], index, x);
70          else      modify(a->child[RIGHT], index - middle - 1, x);
71      }
72      update(a);
73  }
74
75  value query(node *a, int l, int r) {
76      assert(a != nullptr);
77      if (l <= 0 && size(a) - 1 <= r) return a->subtree;
78      int middle = size(a->child[LEFT]);
79      if (r < middle) return query(a->child[LEFT], l, r);
80      if (middle < l) return query(a->child[RIGHT], l - middle - 1, r - middle - 1);
81      value res = a->x;
82      if (l < middle && a->child[LEFT] != nullptr)
83          res = query(a->child[LEFT], l, r) + res;
84      if (middle < r && a->child[RIGHT] != nullptr)
85          res = res + query(a->child[RIGHT], l - middle - 1, r - middle - 1);
86      return res;
87  }
```

5.8 Splay 树

```
typedef int value;

enum { LEFT, RIGHT };
struct node {
    node * child[2], * parent;
    value v, subtree;
    int size;
} pool[MAXN], * pool_next = pool;

node * allocate(const value & v) {
    node * x = pool_next++;
    x->parent = x->child[LEFT] = x->child[RIGHT] = nullptr;
    x->subtree = x->v = v;
    x->size = 1;
    return x;
}

struct tree {
    node * root;
    tree(): root(allocate(0)) {}

    bool child_dir(const node * x, const node * y) { return (x->child[LEFT] == y) ? LEFT :
        RIGHT; }
    bool is_child(const node * x, const node * y) { return x->child[LEFT] == y || x->child[
        RIGHT] == y; }

    void update(node * x) {
        x->size = 1;
        x->subtree = x->v;
        FOR (d, 2) if (x->child[d] != nullptr) {
            x->size += x->child[d]->size;
            if (d == LEFT) x->subtree = x->child[LEFT]->subtree + x->subtree;
            else x->subtree = x->subtree + x->child[RIGHT]->subtree;
        }
    }

    void set_child(node * x, bool dir, node * y) {
        if ((x->child[dir] = y) != nullptr) y->parent = x;
        update(x);
    }

    node * rotate(node * x, bool dir) {
        node * parent = x->parent, * y = x->child[dir];
        set_child(x, dir, y->child[!dir]);
        set_child(y, !dir, x);
        set_child(parent, child_dir(parent, x), y);
        return y;
    }

```

```
48      node * splay(node * x) {
49          node * old_p = nullptr;
50          while (x->parent != nullptr) {
51              node * p = x->parent;
52              x = rotate(p, child_dir(p, x));
53              if (old_p != nullptr && is_child(p, old_p)) rotate(p, child_dir(p, old_p));
54              old_p = p;
55          }
56          return x;
57      }
58
59      node * insert(int order, const value & v) { // order is 0-indexed
60          bool dir = LEFT;
61          node * parent = root, * x = parent->child[LEFT];
62          while (x != nullptr) {
63              int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
64              parent = x;
65              if (order <= left_size) x = x->child[dir = LEFT];
66              else {
67                  order -= left_size + 1;
68                  x = x->child[dir = RIGHT];
69              }
70          }
71          set_child(parent, dir, x = allocate(v));
72          return splay(x);
73      }
74
75      node * find(int order) {
76          node * x = root->child[LEFT];
77          while (true) {
78              int left_size = (x->child[LEFT] == nullptr) ? 0 : x->child[LEFT]->size;
79              if (order < left_size) x = x->child[LEFT];
80              else if (order == left_size) break;
81              else {
82                  order -= left_size + 1;
83                  x = x->child[RIGHT];
84              }
85          }
86          return splay(x);
87      }
88
89      void erase(const int& order) {
90          node * x = find(order);
91          if (x->child[LEFT] == nullptr) set_child(root, LEFT, x->child[RIGHT]);
92          else if (x->child[RIGHT] == nullptr) set_child(root, LEFT, x->child[LEFT]);
93          else {
94              node * y = x->child[RIGHT];
95              while (y->child[LEFT] != nullptr) y = y->child[LEFT];
96              y = splay(y);
97              set_child(y, LEFT, x->child[LEFT]);
98              set_child(root, LEFT, y);
99          }
100     }
101
```

```
    value query(int e) { // e is the prefix length desired.
        node * x = root->child[LEFT];
        if (e <= 0) return 0;
        if (e >= x->size) return x->subtree;
        x = find(e - 1);
        if (x->child[LEFT] != nullptr) return x->child[LEFT]->subtree * x->v;
        else return x->v;
    }
};
```

5.9 莫队算法

```
//Author:marszed
/*
*离线区间处理问题。
*从区间[l,r]得到区间[l+1,r+1] [l-1,r-1]信息的转移复杂度为O(1)。
*siz为块大小。
*cnt为位于第几个块。
*modify()函数为转移函数。
*/

#include <iostream>
#include <algorithm>

const int maxn = 2e5 + 10;

int n, siz, q;
int a[maxn];

struct Node {
    int id, l, r, val, cnt;

    int operator<(const Node& b)
    {
        return cnt == b.cnt ? r < b.r : cnt < b.cnt;
    }
}nod[maxn];

void modify(int i, int flag) {

}

void mo() {
    cin >> n >> q;
    siz = sqrt(n);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    for (int i = 1; i <= q; i++) {
        cin >> nod[i].l >> nod[i].r;
        nod[i].id = i;
        nod[i].cnt = nod[i].l / siz;
```

```
    }
    std::sort(nod + 1, nod + q + 1);
    int l = 0, r = 0;
    for (int i = 1; i <= q; i++) {
        while (l < nod[i].l - 1) modify(++l);
        while (l >= nod[i].l) modify(l--);
        while (r < nod[i].r) modify(++r);
        while (r > nod[i].r) modify(r--);
        ans[nod[i].id] = Ans;
    }
}
```

第六章　字符串

6.1　KMP

```cpp
//Author:CookiC
//返回下标最大的匹配串
#include<cstring>

void getFail(char *P, int *f) {
    int i, j;
    f[0] = 0;
    f[1] = 0;
    for(i=1; P[i]; ++i) {
        j = f[i];
        while(j && P[i]!=P[j]) {
            j = f[j];
        }
        f[i+1] = P[i]==P[j]? j+1: 0;
    }
}

int KMP(char *T, char *P) {
    int ans = -1;
    int n = strlen(T), m = strlen(P);
    int *f = new int[m+1];
    getFail(P, f);
    int j = 0;
    for(int i=0; i<n; ++i){
        while(j && P[j]!=T[i])
        j = f[j];
        if(P[j]==T[i]) {
            ++j;
        }
        if(j==m) {
            j = f[j];
            ans = i-m+1;
        }
    }
    return ans;
}
```

6.2　TRIE

```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;
}
```

6.3 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;
    }
}
```

6.4 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;
}
```

6.5 最长回文子串

```
const int maxn=2000005;
int f[maxn];
std::string a, s;
int manacher() {
    int n=0, res=0, maxr=0, pos=0;
    for (int i=0; a[i]; i++) {
        s[++n] = '#', s[++n] = a[i];
```

```
8           s[++n] = '#';
9       }
10      for (int i=1; i<=n; i++) {
11          f[i] = (i<maxr? std::min(f[pos*2-i], maxr-i+1): 1);
12          while (i-f[i]>0 && i+f[i]<=n && s[i-f[i]]==s[i+f[i]]) {
13              f[i]++;
14          }
15          if (i+f[i]-1 > maxr) {
16              maxr=i+f[i]-1;
17              pos=i;
18          }
19          res = std::max(res,f[i]-1);
20      }
21      return res;
22  }
```

第七章　几何

7.1　平面几何公式

```
三角形:
    1. 半周长 P=(a+b+c)/2
    2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
    3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
    4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
    5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
    6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
                                =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
                                =Ptan(A/2)tan(B/2)tan(C/2)
    7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))


    四边形:
    D1,D2为对角线,M对角线中点连线,A为对角线夹角
    1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
    2. S=D1D2sin(A)/2
    (以下对圆的内接四边形)
    3. ac+bd=D1D2
    4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长


    正n边形:
    R为外接圆半径,r为内切圆半径
    1. 中心角 A=2PI/n
    2. 内角 C=(n-2)PI/n
    3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
    4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))


    圆:
    1. 弧长 l=rA
    2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
    3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
    4. 扇形面积 S1=rl/2=r^2A/2
    5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2


    棱柱:
    1. 体积 V=Ah,A为底面积,h为高
    2. 侧面积 S=lp,l为棱长,p为直截面周长
    3. 全面积 T=S+2A

```

```
棱锥:
1. 体积 V=Ah/3,A为底面积,h为高
(以下对正棱锥)
2. 侧面积 S=lp/2,l为斜高,p为底面周长
3. 全面积 T=S+A


棱台:
1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高
(以下为正棱台)
2. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高
3. 全面积 T=S+A1+A2


圆柱:
1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h


圆锥:
1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3


圆台:
1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3


球:
1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3


球台:
1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6


球扇形:
1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径
2. 体积 V=2PIr^2h/3
```

第八章　类

8.1　点类

```
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;
    }
};
```

8.2　分数类

```
struct Fraction {
    long long num;
    long long den;
    Fraction(long long num=0,long long den=1) {
        if(den<0) {
            num=-num;
            den=-den;
        }
        assert(den!=0);
        long long g=gcd(abs(num),den);
        this->num=num/g;
        this->den=den/g;
    }
    Fraction operator +(const Fraction &o)const {
        return Fraction(num*o.den+o.num,den*o.den);
    }
```

```
    Fraction operator -(const Fraction &o)const {
        return Fraction(num*o.den-den*o.num,den*o.den);
    }
    Fraction operator *(const Fraction &o)const {
        return Fraction(num*o.num,den*o.den);
    }
    Fraction operator /(const Fraction &o)const {
        return Fraction(num*o.den,den*o.num);
    }
    bool operator <(const Fraction &o)const {
        return num*o.den< den*o.num;
    }
    bool operator ==(const Fraction &o)const {
        return num*o.den==den*o.num;
    }
};
```

8.3 矩阵

```
#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)
            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';
        }
    }
};
```

8.4 01 矩阵

```
#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;
}
```

第九章　黑科技

9.1　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))
```

9.2　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));
}
```