目录

        - 1.线段树套线段树
        - 2.树状数组套动态开点线段树
        - 3.线段树套平衡树
        - 4.平衡树套线段树
            - 暴力重构O(nlogn^3)
            - 线段树合并
    - KD-Tree
        - 通用
        - 邻域查询
        - 带插入删除(替罪羊树重构思想)
        - 高维询问/修改
- 字符串
    - 字符串哈希
        - 模板
            - 1.自然溢出
            - 2.二维矩阵hash
            - 3.双hash
    - 序列自动机
        - 长度为 k 的本质不同子序列个数
    - 树状数组维护字符串哈希
    - KMP
        - 资料
        - 模板
        - 应用
            - 1.求字符串的最小循环节
            - 2.求字符串 S 每一个前缀在字符串 T 中的出现次数
    - 扩展kmp
        - 资料
        - 模板
    - 字符串最小表示法
        - 循环同构
        - 最小表示
        - 资料
        - 模板
    - manacher
    - AC自动机
        - 1.求以文本串每个位置匹配的模式串个数
        - 2.求每个模式串的出现次数
    - 后缀数组
        - 数组含义
        - 性质
        - 模板
        - 统计字符串中相同的子串的对数 或 字符串中每一对后缀的LCP之和
            - 模板
        - 字符串本质不同的子串个数
    - 后缀自动机

Head

```
#include "bits/stdc++.h"

using namespace std;

typedef unsigned long long ULL;
typedef long long LL;
typedef pair<int, int> pr;
typedef pair<LL, LL> lpr;
typedef pair<double, double> dpr;
typedef pair<long double, long double> ldpr;
typedef double db;
typedef long double ld;
// typedef complex<double> cp;
// typedef pair<ld,ld> pt;

#define dbgs(x) #x << " = " << x
#define dbgs2(x, y) dbgs(x) << ", " << dbgs(y)
#define dbgs3(x, y, z) dbgs2(x, y) << ", " << dbgs(z)
#define dbgs4(w, x, y, z) dbgs3(w, x, y) << ", " << dbgs(z)

#define mst(s, x) memset(s, x, sizeof(s))
#define fi first
#define se second
#define pb push_back
#define pf push_front
#define ppb pop_back
#define ppf pop_front
#define mp make_pair
#define all(x) x.begin(), x.end()
#define unq(x) x.erase(unique(x.begin(), x.end()), x.end())
#define rg register
#define fp(i, a, b) for (rg int i = (a), I = (b) + 1; i < I; ++i)
#define fd(i, a, b) for (rg int i = (a), I = (b)-1; i > I; --i)
#define ub upper_bound
#define lb lower_bound
#define bitcount __builtin_popcount
#define sqr(x) ((x)*(x))

#define go(u) for (rg int i = pre[u], v = e[i].v; i; i = e[i].next, v = e[i].v)
#define gow(u) for (rg LL i = pre[u], v = e[i].v, w = e[i].w; i; i = e[i].next, v
= e[i].v, w = e[i].w)
#define EDGE(N,M) struct edge{int u,v,w,next;}e[M];int pre[N],edge_cnt=0;void
addedge(int u,int v,int w=0)
{edge_cnt=edge_cnt+1,e[edge_cnt].u=u,e[edge_cnt].v=v,e[edge_cnt].w=w,e[edge_cnt].n
ext=pre[u],pre[u]=edge_cnt;}
#define CLEAR_EDGE(N) {edge_cnt=0,memset(pre,0,sizeof(int)*(N+1));}

#define MAXSIZE (1 << 20)
#define isdigit(x) (x >= '0' && x <= '9')

#define MC int _CASE;scanf("%d",&_CASE);fp(case_num,1,_CASE)

const int next_x[8] = {0, 1, -1, 0, 1, 1, -1, -1};
```

```
const int next_y[8] = {1, 0, 0, -1, 1, -1, -1, 1};
const int inf = 1e9+5;//7;
const LL linf = 1e18+5;
const double PI = acos(-1.0);

const int MAXN = 81;
const int N    = 2e6+5;
const int fix  = 500*500+5;

#define mid ((l+r)/2)
#define ls (cnt<<1)
#define rs (cnt<<1|1)

// #define ls(x) ch[x][0]
// #define rs(x) ch[x][1]

/*----------head----------*/

int work(){};

// 小数 二分/三分 注意break条件
// 浮点运算 sqrt(a^2-b^2) 可用 sqrt(a+b)*sqrt(a-b) 代替，避免精度问题
// long double -> %Lf 别用C11 (C14/16)
// 控制位数 cout << setprecision(10) << ans;
// reverse vector 注意判空 不然会re
// 分块注意维护块上标记 来更新块内数组a[]
// vector+lower_bound常数 < map/set/(unordered_map)
// map.find不会创建新元素 map[]会 注意空间
// 别对指针用memset
// 用位运算表示2^n注意加LL 1LL<<20
// 注意递归爆栈
// 注意边界
// 注意memset 多组会T

//#define ONLINE_JUDGE

int main()
{
    // ios::sync_with_stdio(false);
    // cin.tie(0);
#ifdef ONLINE_JUDGE
    // freopen("triatrip.in", "r",stdin );
    // freopen("triatrip.out","w",stdout);
#endif // ONLINE_JUDGE
#ifndef ONLINE_JUDGE
    // gcc 本地扩栈
    int size = 512 << 20; // 512MB
    char *p = (char*)malloc(size) + size;
    __asm__("movl %0, %%esp\n" :: "r"(p));
    freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);
#endif
    // make();
    work();
```

```
#ifndef ONLINE_JUDGE
    cout << "time:" << clock() << "ms" << endl;
    freopen("CON", "r", stdin);
    freopen("CON", "w", stdout);
    system("pause");
    fflush(stdout);
#endif
    return 0;
}
```

python

读入

```
n,m,k = map(int,raw_input().split())
```

二维数组

```
f = [[0 for i in range(10)] for i in range(10)]
```

读入挂

普通

```
inline int read()
{
    int sum=0;char ch=getchar();
    while(!(ch>='0'&&ch<='9'))ch=getchar();
    while(ch>='0'&&ch<='9')sum=sum*10+ch-48,ch=getchar();
    return sum;
}
inline void out(int x)
{
   if(x>9)out(x/10);
   putchar(x%10+'0');
}
```

二逼

```
// #define DEBUG 1  //调试开关
struct IO {
#define MAXSIZE (1 << 20)
```

```
#define isdigit(x) (x >= '0' && x <= '9')
  char buf[MAXSIZE], *p1, *p2;
  char pbuf[MAXSIZE], *pp;
#if DEBUG
#else
  IO() : p1(buf), p2(buf), pp(pbuf) {}
  ~IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
#endif
  inline char gc() {
#if DEBUG  //调试，可显示字符
    return getchar();
#endif
    if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
    return p1 == p2 ? ' ' : *p1++;
  }
  inline bool blank(char ch) {
    return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
  }
  template <class T>
  inline void read(T &x) {
    register double tmp = 1;
    register bool sign = 0;
    x = 0;
    register char ch = gc();
    for (; !isdigit(ch); ch = gc())
      if (ch == '-') sign = 1;
    for (; isdigit(ch); ch = gc()) x = x * 10 + (ch - '0');
    if (ch == '.')
      for (ch = gc(); isdigit(ch); ch = gc())
        tmp /= 10.0, x += tmp * (ch - '0');
    if (sign) x = -x;
  }
  inline void read(char *s) {
    register char ch = gc();
    for (; blank(ch); ch = gc())
      ;
    for (; !blank(ch); ch = gc()) *s++ = ch;
    *s = 0;
  }
  inline void read(char &c) {
    for (c = gc(); blank(c); c = gc())
      ;
  }
  inline void push(const char &c) {
#if DEBUG  //调试，可显示字符
    putchar(c);
#else
    if (pp - pbuf == MAXSIZE) fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
    *pp++ = c;
#endif
  }
  template <class T>
  inline void write(T x) {
    if (x < 0) x = -x, push('-');  // 负数输出
```

```
    static T sta[35];
    T top = 0;
    do {
      sta[top++] = x % 10, x /= 10;
    } while (x);
    while (top) push(sta[--top] + '0');
  }
  template <class T>
  inline void write(T x, char lastChar) {
    write(x), push(lastChar);
  }
} io;
```

fread

```
char ibuf[40000000], *ih = ibuf, obuf[15000000], *oh = obuf;
inline void read( int &x ) //fread(ibuf,1,40000000,stdin);
{
    int f=1;
    for (; !isdigit( *ih ); ++ih )if(*ih == '-') f=-1;
    for ( x = 0; isdigit( *ih ); x = x * 10 + *ih++ - 48 );
    x*=f;
}
inline void out( LL x ) // fwrite( obuf, 1, oh - obuf, stdout );
{
    if ( !x ){*oh++ = '0'; return;}
    if(x<0)*oh++='-',x=-x;
    static int buf[30]; int xb = 0;
    for (; x; x /= 10 )
        buf[++xb] = x % 10;
    for (; xb; )
        *oh++ = buf[xb--] | 48;
}
```

数据结构

维护树上 dis(*，u) 之和

$dis(u,v)=d[u]+d[v]-d[lca(u,v)]$ ，考虑维护所有点对的 lca 之和，加入一个点对它到根节点的链区间加，询问 $dis(*,u)$ 时对 u 到根节点的链求和即为 $d[lca(u,v)]$ 之和。

笛卡尔树 && Treap 线性构造

```
int stk[MAXN],top,root;
void build(int p[])
{
    top=0;
```

```
    fp(i,0,n-1)
    {
        ls[i]=rs[i]=-1;if(d[i]==p[i]%10)continue;

        while(top&&p[i]<p[stk[top]])ls[i]=stk[top--];
        if(top)rs[stk[top]]=i;
        stk[++top]=i;
    }

    if(!top)root=-1;
    else root=stk[1];
}
```

std::list O(1) 合并

stl 链表 O(1) 合并

```
int work()
{
    list<int> l1,l2;
    fp(i,1,100000)l1.push_back(i),l2.push_back(i);
    l1.splice(l1.begin(),l2); //将 l2 接到 l1 后面 O(1)
    return 0;
}
```

并查集 & 带权并查集

数颜色

1.莫队 nsqrt(n)

小Z的袜子

2.主席树 or 二维数点 mlogn

设 p[i] = max{j|j<i&&a[j]==a[i]} (不存在这样的 j 时p[i]=0)

那么区间不重复的个数等价于 [l,r] 范围内 p[i]<l 的i的个数，二维数点 or 主席树维护即可。

主席树

```
int cnt=0,s[MAXN*40],ls[MAXN*40],rs[MAXN*40];
void update(int pre,int &rt,int l,int r,int val)
{
    if(!rt)rt=++cnt; s[rt]=s[pre]+1; if(l==r)return;
    if(val<=mid)update(ls[pre],ls[rt],l,mid,val),rs[rt]=rs[pre];
    else update(rs[pre],rs[rt],mid+1,r,val),ls[rt]=ls[pre];
}
```

```
int query(int pre,int rt,int l,int r,int nl,int nr)
{
    if(!rt) return 0;
    if(l==nl&&r==nr)return s[rt]-s[pre];
    if(nr<=mid)return query(ls[pre],ls[rt],l,mid,nl,nr);
    if(nl>mid) return query(rs[pre],rs[rt],mid+1,r,nl,nr);
    return query(ls[pre],ls[rt],l,mid,nl,mid)+
        query(rs[pre],rs[rt],mid+1,r,mid+1,nr);
}
void debug(int rt,int l,int r)
{
    cout << dbgs4(rt,l,r,s[rt]) << endl;
    if(l==r) return;
    debug(ls[rt],l,mid),debug(rs[rt],mid+1,r);
}
int n,m;
int a[MAXN],pre[MAXN],root[MAXN];
int work()
{
    io.read(n);
    fp(i,1,n)io.read(a[i]);
    fp(i,1,n)update(root[i-1],root[i],0,N,pre[a[i]]),pre[a[i]]=i;
    io.read(m);
    while(m--)
    {
        int l,r; io.read(l),io.read(r);
        int ans=query(root[l-1],root[r],0,N,0,l-1);
        io.write(ans),io.push('\n');
    }
    return 0;
}
```

二维数点(离线sort+树状数组)

```
int n,m,head;
int a[MAXN],c[MAXN],ans[MAXN];
int pre[MAXN];

namespace BIT
{
    int c[MAXN];
    int lowbit(int x){return x&-x;}
    void update(int x,int v){x++;for(int i=x;i<MAXN;i+=lowbit(i))c[i]+=v;}
    int sum(int x)
    {
        x++;
        int ans=0;
        for(int i=x;i;i-=lowbit(i))
            ans+=c[i];
        return ans;
    }
```

```
    int query(int l,int r){return sum(r)-sum(l-1);}
}
using namespace BIT;

struct point
{
    int x,y,index,f;
    bool operator <(const point &t)
    {
        if(x!=t.x) return x<t.x;
        if(y!=t.y) return y>t.y;
        return index<t.index;
    }
}q[MAXN*2];

int work()
{
    io.read(n);//,io.read(m);
    fp(i,1,n)io.read(a[i]);
    fp(i,1,n)::c[i]=pre[a[i]],pre[a[i]]=i,q[++head]={i,::c[i],0};
    io.read(m);
    fp(i,1,m)
    {
        int l,r; io.read(l),io.read(r);
        ans[i]=-(r-l+1);
        //q[++head]={l-1,l,i,-1};
        q[++head]={r,l,i, 1};
    }
    sort(q+1,q+1+head);
    fp(i,1,head)
    {
        if(q[i].index) ans[q[i].index]+=q[i].f*query(q[i].y,n);
        else update(q[i].y,1);
    }
    fp(i,1,m) io.write(-ans[i]),io.push('\n');
    //fp(i,1,m)printf("%d ",-ans[i]);printf("\n");
    return 0;
}
```

3.树状数组 mlogn (常数小，且可用主席树维护)

考虑对于每个 r 维护一个不重复的 a[i] 的集合，且 a[i] 重复出现时保留最靠右的。 那么对于固定 r 的询问 [l,r] 只需要求当前集合中下标在 [l,r] 范围内的数的个数，树状数组维护即可。 当需要保存每个 r 对应的集合时可以考虑用主席树维护。

```
//换掉vector能更快
int n,m;
int l[MAXN],r[MAXN],a[MAXN],pre[MAXN],ans[MAXN],c[MAXN];
vector<int> q[MAXN];
void up(int x,int v){for(int i=x;i<MAXN;i+=(i&-i))c[i]+=v;}
int qr(int x,int ans=0){for(int i=x;i;i-=(i&-i))ans+=c[i];return ans;}
```

```
int work()
{
    scanf("%d",&n);
    fp(i,1,n)scanf("%d",&a[i]);
    scanf("%d",&m);
    fp(i,1,m)scanf("%d%d",&l[i],&r[i]),q[r[i]].pb(i);
    fp(i,1,n)
    {
        if(pre[a[i]])up(pre[a[i]],-1); up(i,1),pre[a[i]]=i;
        for(auto j:q[i])ans[j]=qr(i)-qr(l[j]-1);
    }
    fp(i,1,m)printf("%d\n",ans[i]);
    return 0;
}
```

珂朵莉树

资料

我的数据结构不可能这么可爱！——珂朵莉树(ODT)详解

珂朵莉树详解

例题

CF896C

维护一个数据结构，支持区间加法，区间赋值，区间k大，和区间幂次和 $\sum_{i=l}^{r} a_i^x$

将一段 a_i 相同的区间合并起来，用一个set来维护，在操作随机的情况下各种操作的复杂度为 $log(n)$

模板

```
LL vmax,seed;
LL rnd()
{
    LL ret=seed;
    seed=(seed*7+13)%(M);
    return ret;
}
LL fpow(LL x,LL k,LL MOD)
{
    x%=MOD; LL ans=1;
    while(k)
    {
        if(k&1)ans=ans*x%MOD;
        x=x*x%MOD,k=k>>1;
    }
    return ans;
}
```

```
int n,m;
LL a[MAXN];
struct node
{
    int l,r;
    mutable LL t;
    bool operator<(const node &t)const
    {
        if(l==t.l) return r<t.r;
        return l<t.l;
    }
};
set<node> s;
typedef set<node>::iterator I;
I split(int pos)
{
    I it=s.lb({pos,-1,0});
    if(it!=s.end()&&it->l==pos)return it;
    it--; int l=it->l,r=it->r;LL t=it->t;
    s.erase(it),s.insert({l,pos-1,t});
    return s.insert({pos,r,t}).fi;
}
void td(int l,int r,LL t)
{
    I R=split(r+1),L=split(l); //先拆右边，防止迭代器失效
    s.erase(L,R),s.insert({l,r,t});
}
void add(int l,int r,LL v)
{
    I R=split(r+1),L=split(l);
    for(I it=L;it!=R;it++)it->t+=v;
}
vector<lpr> t;
void kth(int l,int r,int k)
{
    t.clear();
    I R=split(r+1),L=split(l);
    for(I it=L;it!=R;it++)
        t.pb({it->t,it->r-it->l+1});
    sort(all(t));
    int cnt=0;
    for(auto x:t)
    {
        cnt+=x.se;
        if(cnt>=k){printf("%lld\n",x.fi);return;}
    }
}
void sum(int l,int r,LL x,LL MOD)
{
    LL ans=0;
    I R=split(r+1),L=split(l);
    for(I it=L;it!=R;it++)
        ans+=LL(it->r-it->l+1)*fpow(it->t,x,MOD)%MOD,ans%=MOD;
    printf("%lld\n",ans);
```

```
}

int work()
{
    scanf("%d%d%lld%lld",&n,&m,&seed,&vmax);
    fp(i,1,n)a[i]=rnd()%vmax+1;
    for(int cnt=1,i=2;i<=n+1;i++)
    {
        if(a[i]!=a[i-1]||i==n+1)
            s.insert({i-cnt,i-1,a[i-1]}),cnt=1;
        else cnt++;
    }
    s.insert({n+1,n+1,0});
    fp(i,1,m)
    {
        int op=rnd()%4+1,l=rnd()%n+1,r=rnd()%n+1,x,y;
        if(l>r)swap(l,r);
        if(op==3)x=(rnd()%(r-l+1))+1;
        else x=(rnd()%vmax)+1;
        if(op==4)y=(rnd()%vmax)+1;

        if(op==1)add(l,r,x);
        if(op==2)td(l,r,x);
        if(op==3)kth(l,r,x);
        if(op==4)sum(l,r,(LL)x,(LL)y);
    }
    return 0;
}
```

树状数组

资料

"高级"数据结构——树状数组!

二维树状数组

单点修改 矩阵求和

```
int n,m;
LL c[MAXN][MAXN];

int lowbit(int x){return x&-x;}
void update(int x,int y,int v)
{
    for(int i=x;i<=n;i+=lowbit(i))
        for(int j=y;j<=m;j+=lowbit(j))
            c[i][j]+=v;
}
LL sum(int x,int y,LL ans=0)
{
```

```
    for(int i=x;i;i-=lowbit(i))
        for(int j=y;j;j-=lowbit(j))
            ans+=c[i][j];
    return ans;
}
LL query(int x1,int y1,int x2,int y2)
{
    x1--,y1--;
    return sum(x2,y2)+sum(x1,y1)-sum(x1,y2)-sum(x2,y1);
}

int work()
{
    scanf("%d%d",&n,&m);
    int opt,a,b,c,d;
    while(scanf("%d",&opt)==1)
    {
        scanf("%d%d%d",&a,&b,&c);
        if(opt==1) update(a,b,c);
        if(opt==2) scanf("%d",&d),printf("%lld\n",query(a,b,c,d));
    }
    return 0;
}
```

矩形修改 单点求值

考虑维护差分数组

$$d[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1]$$

给矩形 $(x1,y1)$ $(x2,y2)$ 加上某个值等价于

$$d[x1][y1]+=v,d[x2+1][y1]-=v;$$

$$d[x1][y2+1]-=v,d[x2+1][y2+1]+=v;$$

单点求值

$$a[i][j]=\sum_{i=1}^{n}\sum_{j=1}^{m}d[i][j]$$

用普通的二维树状数组维护即可

```
namespace bit
{
    LL c[MAXN][MAXN];
    void update(int x,int y,int val)
    {
        for(int i=x;i<MAXN;i+=(i&-i))
            for(int j=y;j<MAXN;j+=(j&-j))
                c[i][j]+=val;
```

```
    }
    void add(int x1,int y1,int x2,int y2,int val)
    {
        x2++,y2++;
        update(x1,y1, val);
        update(x2,y1,-val);
        update(x1,y2,-val);
        update(x2,y2, val);
    }
    LL query(int x,int y)
    {
        int s=0;
        for(int i=x;i;i-=(i&-i))
            for(int j=y;j;j-=(j&-j))
                s+=c[i][j];
        return s;
    }
}
```

矩形修改 矩形求和(待补))

线段树

李超树

给出 m 条直线 $y=kx+b$ 询问在整点 x' 上最大的 y' 是多少。

```
double K[N*4],B[N*4];
bool cov[N*4];
void build(int l,int r,int cnt)
{
    cov[cnt]=K[cnt]=B[cnt]=0;
    if(l==r) return;
    build(l,mid,ls);
    build(mid+1,r,rs);
}
void insert(int l,int r,double nk,double nb,int cnt)
{
    if(!cov[cnt]){cov[cnt]=1,K[cnt]=nk,B[cnt]=nb;return;}
    if(nk>K[cnt]) swap(nk,K[cnt]),swap(nb,B[cnt]);
    if(K[cnt]*mid+B[cnt]>nk*mid+nb)
        {if(l!=r) insert(l,mid,nk,nb,ls);}
    else
    {
        swap(nk,K[cnt]),swap(nb,B[cnt]);
        if(l!=r) insert(mid+1,r,nk,nb,rs);
    }
}
double query(int l,int r,int x,int cnt)
{
    double Y=K[cnt]*x+B[cnt];
```

```
    if(l==r) return Y;
    if(x<=mid) return max(Y,query(l,mid,x,ls));
    if(x> mid) return max(Y,query(mid+1,r,x,rs));
}
```

扫描线

1.扫描线求矩形面积交

```
vector<int> Y;

//线段树中第i个点表示 Y[i]~Y[i+1] 这条线段
LL sum[MAXN*4];
int cov[MAXN*4]; //标记永久化
void pushup(int l,int r,int cnt)
{
    if(cov[cnt]>0) sum[cnt]=Y[r+1]-Y[l];
    else sum[cnt]=sum[ls]+sum[rs];
}
void update(int l,int r,int nl,int nr,int f,int cnt)
{
    if(l==nl&&r==nr)
    {
        cov[cnt]+=f;
        if(cov[cnt]>0) sum[cnt]=(Y[r+1]-Y[l]);
        else sum[cnt]=sum[ls]+sum[rs];
        return;
    }
    if(nr<=mid) update(l,mid,nl,nr,f,ls);
    else if(nl>mid) update(mid+1,r,nl,nr,f,rs);
    else update(l,mid,nl,mid,f,ls),update(mid+1,r,mid+1,nr,f,rs);
    pushup(l,r,cnt);
}
int n;
struct line
{
    int x,y1,y2,f;
    bool operator <(const line &t)const{return x<t.x;}
};
vector<line> L;
LL ans=0;

int work()
{
    n=read();
    for(int i=1;i<=n;i++)
    {
        int x1=read(),y1=read(); //左下
        int x2=read(),y2=read(); //右上
        Y.pb(y1),Y.pb(y2);
        L.pb({x1,y1,y2, 1});
```

```
        L.pb({x2,y1,y2,-1});
    }
    sort(all(Y)),unq(Y),sort(all(L));

    int m=Y.size()-2;
    for(int i=0;i<L.size();i++)
    {
        L[i].y1=lower_bound(all(Y),L[i].y1)-Y.begin();
        L[i].y2=lower_bound(all(Y),L[i].y2)-Y.begin();
        if(i) ans+=LL(L[i].x-L[i-1].x)*sum[1];
        update(0,m,L[i].y1,L[i].y2-1,L[i].f,1);
    }
    return write(ans),putchar('\n');
}
```

线段树合并

多用于权值线段树，均摊 $logn$

```
int merge(int x,int y,int l,int r)
{
    if(x*y==0)return x+y;
    if(l==r){mx[x]+=mx[y];return x;}
    ls[x]=merge(ls[x],ls[y],l,mid);
    rs[x]=merge(rs[x],rs[y],mid+1,r);
    up(x);return x;
}
```

线段树分裂

线段树优化建图

$[a,b]$ 的点到 $[c,d]$ 的点有边权为 w 的边

```
//点数 n*4*2+m*2*2
//边数 n*4*2+n+(1+2*logn)*m*2
EDGE(N,M);
int n,m,S,node=0;
int in[MAXN*4],out[MAXN*4],dis[N];
bool v[MAXN];
vector<int> q1,q2;
void build(int l,int r,int cnt)
{
    in[cnt]=++node,out[cnt]=++node;
    if(cnt/2)
    {
        addedge(in[cnt/2],in[cnt],0);
        addedge(out[cnt],out[cnt/2],0);
    }
```

```
    if(l==r){addedge(in[cnt],out[cnt],0);return;}
    else build(l,mid,ls),build(mid+1,r,rs);
}
void query(int l,int r,int nl,int nr,int id[],vector<int> &v,int cnt)
{
    if(l==nl&&r==nr){v.pb(id[cnt]);return;}
    if(nr<=mid)query(l,mid,nl,nr,id,v,ls);
    else if(nl>mid)query(mid+1,r,nl,nr,id,v,rs);
    else query(l,mid,nl,mid,id,v,ls),query(mid+1,r,mid+1,nr,id,v,rs);
}
void link(int a,int b,int c,int d)
{
    query(1,n,a,b,out,q1,1),query(1,n,c,d,in,q2,1);
    int t1=++node,t2=++node;
    addedge(t1,t2,1);
    for(auto x:q1)addedge(x,t1,0);
    for(auto x:q2)addedge(t2,x,0);
    q1.clear(),q2.clear();
}
void bfs()
{
    query(1,n,S,S,in,q1,1); S=q1.back();
    deque<int> q; mst(dis,-1); dis[S]=0,q.pb(S);
    while(!q.empty())
    {
        int u=q.front(); q.ppf();
        if(v[u])continue;
        else v[u]=1;
        gow(u)if(dis[v]==-1||dis[u]+w<dis[v])
            dis[v]=dis[u]+w,(w?q.pb(v):q.pf(v));
    }
}
void print(int l,int r,int cnt)
{
    if(l==r){printf("%d\n",dis[in[cnt]]);return;}
    print(l,mid,ls),print(mid+1,r,rs);
}
int work()
{
    scanf("%d%d%d",&n,&m,&S);
    build(1,n,1);
    while(m--)
    {
        int a,b,c,d;scanf("%d%d%d%d",&a,&b,&c,&d);
        link(a,b,c,d),link(c,d,a,b);
    }
    bfs(),print(1,n,1);
    return 0;
}
```

(待补)ZKW线段树

Segment Tree Beats(吉司机线段树)

Segment Tree Beats!

1.区间>=k的数变成k

HDU5306

```
LL fimax[MAXN*4],semax[MAXN*4],num[MAXN*4],sum[MAXN*4];
LL setmax[MAXN*4];

void doSetMax(int cnt,LL v)
{
    if(fimax[cnt]>v)
    {
        setmax[cnt] =min(setmax[cnt],v);
        sum   [cnt]-=(fimax[cnt]-v)*num[cnt];
        fimax [cnt] =v;
    }
}
void pushup(int cnt)
{
    sum[cnt]=sum[ls]+sum[rs];
    if(fimax[ls]>fimax[rs])
    {
        fimax[cnt]=fimax[ls];
        semax[cnt]=max(fimax[rs],semax[ls]);
        num  [cnt]=num  [ls];
    }
    else if(fimax[rs]>fimax[ls])
    {
        fimax[cnt]=fimax[rs];
        semax[cnt]=max(fimax[ls],semax[rs]);
        num  [cnt]=num  [rs];
    }
    else
    {
        fimax[cnt]=fimax[ls];
        num  [cnt]=num[ls]+num[rs];
        semax[cnt]=max(semax[ls],semax[rs]);
    }
}
void pushdown(int cnt)
{
    if(setmax[cnt]!=linf)
    {
        doSetMax(ls,setmax[cnt]);
        doSetMax(rs,setmax[cnt]);
        setmax[cnt]=linf;
    }
}
void build(LL a[],int l,int r,int cnt)
```

```
{
    setmax[cnt]=linf;
    if(l==r)
    {
        fimax[cnt]=sum[cnt]=a[l];
        num  [cnt]= 1;
        semax[cnt]=-1;
        return;
    }
    build(a,l,mid,ls);
    build(a,mid+1,r,rs);
    pushup(cnt);
}
void update(int l,int r,int nl,int nr,LL v,int cnt)
{
    if(l==nl&&r==nr)
    {
        if(v>=fimax[cnt]) return;
        if(v>semax[cnt]&&v<fimax[cnt])
            {doSetMax(cnt,v);return;}
    }
    pushdown(cnt);
    if(nr<=mid) update(l,mid,nl,nr,v,ls);
    else if(nl>mid) update(mid+1,r,nl,nr,v,rs);
    else update(l,mid,nl,mid,v,ls),update(mid+1,r,mid+1,nr,v,rs);
    pushup(cnt);
}
LL querySum(int l,int r,int nl,int nr,int cnt)
{
    if(l==nl&&r==nr) return sum[cnt];
    pushdown(cnt);
    if(nr<=mid) return querySum(l,mid,nl,nr,ls);
    if(nl> mid) return querySum(mid+1,r,nl,nr,rs);
    return querySum(l,mid,nl,mid,ls)+querySum(mid+1,r,mid+1,nr,rs);
}
LL queryMax(int l,int r,int nl,int nr,int cnt)
{
    if(l==nl&&r==nr) return fimax[cnt];
    pushdown(cnt);
    if(nr<=mid) return queryMax(l,mid,nl,nr,ls);
    if(nl> mid) return queryMax(mid+1,r,nl,nr,rs);
    return max(queryMax(l,mid,nl,mid,ls),queryMax(mid+1,r,mid+1,nr,rs));
}
void debug(int l,int r,int cnt)
{
/*
    int fimax[MAXN*4],semax[MAXN*4],num[MAXN*4],sum[MAXN*4];
    int setmax[MAXN*4];
*/
    cout << dbgs3(l,r,cnt) << endl;
    cout << dbgs4(fimax[cnt],semax[cnt],num[cnt],sum[cnt]) << " " <<
dbgs(setmax[cnt]) <<endl;
    if(l==r) return;
    debug(l,mid,ls);
```

```
    debug(mid+1,r,rs);
}

int T,n,m;
LL  a[MAXN];

int work()
{
    io.read(T);
    while(T--)
    {
        io.read(n),io.read(m);
        fp(i,1,n) io.read(a[i]);
        build(a,1,n,1);
        while(m--)
        {
            int t,x,y,v;
            io.read(t),io.read(x),io.read(y);
            if(t==0)
                io.read(v),update(1,n,x,y,v,1);
            else
            {
                if(t==1) io.write(queryMax(1,n,x,y,1));
                if(t==2) io.write(querySum(1,n,x,y,1));
                io.push('\n');
            }
            //cout << dbgs(m) << endl << endl;
            //debug(1,n,1);
        }
    }
    return 0;
}
```

2.区间or 区间and 区间max

当区间内所有数受影响的位上相同时可以直接打标记，维护区间内共同1的位置和0的位置和区间max

注意下标记关系，&类似于乘法，|类似于加法

(a&b|c)&d = a&(b&d)|(c&d)

(a&b|c)|d = a&b|(c|d)

```
const  int _all1 = ((1<<21)-1);
inline int flip(int x){return  _all1 ^ x;}

//all1 区间内所有数1的共同位置
//all0 区间内所有数0的共同位置
int n,m,a[MAXN];
int mx[MAXN*4],all1[MAXN*4],all0[MAXN*4];
int orTag[MAXN*4],andTag[MAXN*4];
```

```
//a&b|c a*b+c
void doAnd(int cnt,int v)
{
    all1[cnt]&=v;
    all0[cnt]|=flip(v);
    mx[cnt]&=v,andTag[cnt]&=v,orTag[cnt]&=v;
}
void doOr(int cnt,int v)
{
    all1[cnt]|=v;
    all0[cnt]&=flip(v);
    mx[cnt]|=v,orTag[cnt]|=v;
}
void pushup(int cnt)
{
    mx[cnt]=max(mx[ls],mx[rs]);
    all1[cnt]=all1[ls]&all1[rs];
    all0[cnt]=all0[ls]&all0[rs];
}
void pushdown(int cnt)
{
    if(andTag[cnt]!=_all1)
    {
        doAnd(ls,andTag[cnt]);
        doAnd(rs,andTag[cnt]);
        andTag[cnt]=_all1;
    }
    if(orTag[cnt])
    {
        doOr(ls,orTag[cnt]);
        doOr(rs,orTag[cnt]);
        orTag[cnt]=0;
    }
}
void build(int l,int r,int cnt)
{
    andTag[cnt]=_all1;
    orTag [cnt]=0;
    if(l==r)
    {
        mx  [cnt]=a[l];
        all1[cnt]=a[l];
        all0[cnt]=flip(a[l]);
        return;
    }
    build(l,mid,ls);
    build(mid+1,r,rs);
    pushup(cnt);
}
int query(int l,int r,int nl,int nr,int cnt)
{
    if(l==nl&&r==nr) return mx[cnt];
    pushdown(cnt);
```

```
    if(nr<=mid) return query(l,mid,nl,nr,ls);
    if(nl> mid) return query(mid+1,r,nl,nr,rs);
    return max(query(l,mid,nl,mid,ls),query(mid+1,r,mid+1,nr,rs));
}
void updateAnd(int l,int r,int nl,int nr,int v,int cnt)
{
    if(l==nl&&r==nr)
    {
        if(((all1[cnt]|all0[cnt])&flip(v))==flip(v))
            {doAnd(cnt,v);return;}
    }
    pushdown(cnt);
    if(nr<=mid) updateAnd(l,mid,nl,nr,v,ls);
    else if(nl>mid) updateAnd(mid+1,r,nl,nr,v,rs);
    else updateAnd(l,mid,nl,mid,v,ls),updateAnd(mid+1,r,mid+1,nr,v,rs);
    pushup(cnt);
}
void updateOr(int l,int r,int nl,int nr,int v,int cnt)
{
    if(l==nl&&r==nr)
    {
        if(((all1[cnt]|all0[cnt])&v)==v)
            {doOr(cnt,v);return;}
    }
    pushdown(cnt);
    if(nr<=mid) updateOr(l,mid,nl,nr,v,ls);
    else if(nl>mid) updateOr(mid+1,r,nl,nr,v,rs);
    else updateOr(l,mid,nl,mid,v,ls),updateOr(mid+1,r,mid+1,nr,v,rs);
    pushup(cnt);
}
```

可持久化线段树(主席树)

模板

1.静态区间Kth

```
int n,m,cnt=0,a[N+5],b[N+5];
int root[MAXN],ls[MAXN],rs[MAXN],s[MAXN];
void pushup(int cnt){s[cnt]=s[ls[cnt]]+s[rs[cnt]];}
void update(int pre,int &rt,int l,int r,int v)
{
    rt=++cnt;
    if(l==r){s[rt]=s[pre]+1;return;}
    if(v<=mid) rs[rt]=rs[pre],update(ls[pre],ls[rt],l,mid,v);
    else ls[rt]=ls[pre],update(rs[pre],rs[rt],mid+1,r,v);
    pushup(rt);
}
int query(int lcnt,int rcnt,int l,int r,int k)
{
    if(l==r) return l;
```

```
    int sum=s[ls[rcnt]]-s[ls[lcnt]];
    if(k<=sum) return query(ls[lcnt],ls[rcnt],l,mid,k);
    else return query(rs[lcnt],rs[rcnt],mid+1,r,k-sum);
}
void init()
{
    for(int i=1;i<=n;i++) b[i]=a[i];
    sort(b+1,b+1+n);
    for(int i=1;i<=n;i++) a[i]=lower_bound(b+1,b+1+n,a[i])-b;
}
int main()
{
    n=read(),m=read();
    for(int i=1;i<=n;i++) a[i]=read();
    for(int i=1;i<=n;i++) update(root[i-1],root[i],0,N,a[i]);
    while(m--)
    {
        int l=read(),r=read(),k=read();
        printf("%d\n",b[query(root[l-1],root[r],0,N,k)]);
    }
    return 0;
}
```

(待补) 2.区间修改(标记永久化) & 历史版本询问 hdu4348

```
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <cstring>
#define N 100005
#define MAXN (100005*50)
#define mid ((l+r)>>1)
#define LL long long
using namespace std;
int n,m,cnt=0,a[N],T=0;
int root[N+5],ls[MAXN],rs[MAXN];
LL s[MAXN],add[MAXN];
void build(int &rt,int l,int r)
{
    rt=++cnt;
    if(l==r){s[rt]=a[l];return;}
    build(ls[rt],l,mid);
    build(rs[rt],mid+1,r);
    s[rt]=s[ls[rt]]+s[rs[rt]];
}
void update(int pre,int &rt,int l,int r,int nl,int nr,LL v)
{
    rt=++cnt;
    s[rt]=s[pre]+(LL)(nr-nl+1)*v;
    add[rt]=add[pre];//！！继承标记
    if(l==nl&&r==nr)
```

```
    {
        ls[rt]=ls[pre];
        rs[rt]=rs[pre];
        add[rt]+=v;
        return;
    }
    if(nl>mid) ls[rt]=ls[pre],update(rs[pre],rs[rt],mid+1,r,nl,nr,v);
    else if(nr<=mid) rs[rt]=rs[pre],update(ls[pre],ls[rt],l,mid,nl,nr,v);
    else
update(ls[pre],ls[rt],l,mid,nl,mid,v),update(rs[pre],rs[rt],mid+1,r,mid+1,nr,v);
}
LL query(int rt,int l,int r,int nl,int nr)
{
    if(l==nl&&r==nr) return s[rt];
    LL tmp=(LL)add[rt]*(nr-nl+1);
    if(nl>mid) return query(rs[rt],mid+1,r,nl,nr)+tmp;
    else if(nr<=mid) return query(ls[rt],l,mid,nl,nr)+tmp;
    else return query(ls[rt],l,mid,nl,mid)+query(rs[rt],mid+1,r,mid+1,nr)+tmp;
}
void init()
{
    cnt=T=0;
    memset(root,0,sizeof(root));
    memset(ls,0,sizeof(ls));
    memset(rs,0,sizeof(rs));
    memset(add,0,sizeof(add));
    memset(s,0,sizeof(s));
}
int main()
{
    bool f=1;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        if(f) f=0;
        else printf("\n"),init();
        for(int i=1;i<=n;i++) scanf("%d",&a[i]);
        build(root[0],1,n);
        while(m--)
        {
            char t[5]; scanf("%s",t);
            if(t[0]=='C')
            {
                T++;
                int l,r,v; scanf("%d%d%d",&l,&r,&v);
                update(root[T-1],root[T],1,n,l,r,v);
            }
            else if(t[0]=='B') scanf("%d",&T);
            else
            {
                int l,r,h; scanf("%d%d",&l,&r);
                if(t[0]=='Q') printf("%lld\n",query(root[T],1,n,l,r));
                else if(t[0]=='H')
scanf("%d",&h),printf("%lld\n",query(root[h],1,n,l,r));
            }
```

```
        }
    }
    return 0;
}
```

可持久化 Trie

```
int n,m,root[MAXN],a[MAXN];
int ls[MAXN*35],rs[MAXN*35],s[MAXN*35],cnt;
void insert(int &rt,int pre,int x,int i)
{
    if(!rt) rt=++cnt,s[rt]=s[pre]; s[rt]++;
    if(i==-1) return;
    bool v=((1<<i)&x);
    if(v) ls[rt]=ls[pre],insert(rs[rt],rs[pre],x,i-1);
    else rs[rt]=rs[pre],insert(ls[rt],ls[pre],x,i-1);
}
int query(int lcnt,int rcnt,int x,int i)
{
    if(i==-1||s[rcnt]-s[lcnt]==0) return 0;
    if((1<<i)&x)
    {
        int sum=s[ls[rcnt]]-s[ls[lcnt]];
        if(!sum) return query(rs[lcnt],rs[rcnt],x,i-1)+(1<<i);
        else return query(ls[lcnt],ls[rcnt],x,i-1);
    }
    else
    {
        int sum=s[rs[rcnt]]-s[rs[lcnt]];
        if(sum) return query(rs[lcnt],rs[rcnt],x,i-1)+(1<<i);
        else return query(ls[lcnt],ls[rcnt],x,i-1);
    }
}
```

平衡树

带删除优先队列

```
struct que
{
    heap<int,vector<int>,less<int>> a,b;
    int size(){return a.size()-b.size();}
    bool empty(){return size()==0;}
    void push(int x){a.push(x);}
    void del(int x){b.push(x);}
    void clear(){while(!b.empty()&&a.top()==b.top())a.pop(),b.pop();}
    int top(){clear();return a.top();}
    void pop(){clear();a.pop();}
    int setop()
```

```
    {
        int temp=this->top(),ans; this->pop();
        ans=this->top(); this->push(temp);
        return ans;
    }
}q;
```

1.Splay

2.Treap

3.FHQ_treap(非旋Treap)

排名分裂

```
namespace fhq
{
    int cnt,root;
    int ch[MAXN][2],fix[MAXN],siz[MAXN];
    LL add[MAXN],val[MAXN],mx[MAXN];
    bool rev[MAXN];
    void debug(int x=root)
    {
        if(ls) debug(ls);
        cout << dbgs4(x,ls,rs,fix[x]) << endl;
        cout << dbgs3(val[x],siz[x],mx[x]) << endl;
        if(rs) debug(rs);
    }
    int newnode()
    {
        int x=++cnt;
        val[x]=mx[x]=0,siz[x]=1;
        fix[x]=rand();
        return x;
    }
    void pushup(int x)
    {
        siz[x]=siz[ls]+siz[rs]+1;
        mx[x]=val[x];
        if(ls) mx[x]=max(mx[x],mx[ls]);
        if(rs) mx[x]=max(mx[x],mx[rs]);
    }
    void doAdd(int x,LL v){add[x]+=v,val[x]+=v,mx[x]+=v;}
    void doRev(int x){rev[x]^=1,swap(ls,rs);}
    void pushdown(int x)
    {
        if(rev[x])
        {
            if(ls) doRev(ls);
            if(rs) doRev(rs);
            rev[x]=0;
```

```
        }
        if(add[x])
        {
            if(ls) doAdd(ls,add[x]);
            if(rs) doAdd(rs,add[x]);
            add[x]=0;
        }
    }
    pr split(int x,int k)
    {
        if(!x) return {0,0};
        pr t; pushdown(x);
        if(siz[ls]>=k) t=split(ls,k),ls=t.se,t.se=x;
        else t=split(rs,k-siz[ls]-1),rs=t.fi,t.fi=x;
        pushup(x); return t;
    }
    int merge(int x,int y)
    {
        if(x*y==0) return x+y;
        pushdown(x),pushdown(y);
        if(fix[x]<fix[y]){ch[x][1]=merge(ch[x][1],y);pushup(x);return x;}
        else{ch[y][0]=merge(x,ch[y][0]);pushup(y);return y;}
    }
    void update(int l,int r,LL v)
    {
        pr t1=split(root,l-1);
        pr t2=split(t1.se,r-l+1);
        doAdd(t2.fi,v),pushup(t2.fi);
        root=merge(t1.fi,merge(t2.fi,t2.se));
    }
    void reverse(int l,int r)
    {
        pr t1=split(root,l-1);
        pr t2=split(t1.se,r-l+1);
        doRev(t2.fi),pushup(t2.fi);
        root=merge(t1.fi,merge(t2.fi,t2.se));
    }
    LL query(int l,int r)
    {
        pr t1=split(root,l-1);
        pr t2=split(t1.se,r-l+1);
        LL ans=mx[t2.fi];
        root=merge(t1.fi,merge(t2.fi,t2.se));
        return ans;
    }
    void init(int n)
    {
        fp(i,1,n) root=merge(root,newnode());
    }
}
```

权值分裂

```
namespace fhq
{
    int root,node_cnt;
    int ch[MAXN][2],val[MAXN],siz[MAXN],add[MAXN],cnt[MAXN],fix[MAXN];
    vector<int> rec;

    void debug(int x=root)
    {
        if(ls) debug(ls);
        cout << dbgs3(x,ls,rs) << " " << dbgs4(val[x],siz[x],add[x],cnt[x]) <<
endl;
        if(rs) debug(rs);
    }
    int newnode(int v)
    {
        int x;
        if(!rec.empty()) x=rec.back(),rec.ppb();
        else x=++node_cnt;
        ls=rs=add[x]=0;
        siz[x]=cnt[x]=1;
        fix[x]=rand();
        val[x]=v;
        return x;
    }
    void doAdd(int x,int v){val[x]+=v,add[x]+=v;}
    void pushdown(int x)
    {
        if(add[x])
        {
            if(ls) doAdd(ls,add[x]);
            if(rs) doAdd(rs,add[x]);
            add[x]=0;
        }
    }
    void pushup(int x){siz[x]=siz[ls]+siz[rs]+cnt[x];}
    pr split(int x,int v) //分裂出<=v的
    {
        if(!x) return {0,0};
        pr t; pushdown(x);
        if(val[x]<=v) t=split(rs,v),rs=t.fi,t.fi=x;
        else t=split(ls,v),ls=t.se,t.se=x;
        pushup(x);
        return t;
    }
    int merge(int x,int y)
    {
        if(x*y==0) return x+y;
        pushdown(x),pushdown(y);
        if(fix[x]<fix[y]){ch[x][1]=merge(ch[x][1],y);pushup(x);return x;}
        else{ch[y][0]=merge(x,ch[y][0]);pushup(y);return y;}
    }
    void insert(int v)
```

```
    {
        pr x=split(root,v);
        pr y=split(x.fi,v-1);
        if(y.se)
        {
            cnt[y.se]++,pushup(y.se);
            root=merge(merge(y.fi,y.se),x.se);
            return;
        }
        else root=merge(merge(y.fi,newnode(v)),x.se);
    }
    void del(int x)
    {
        if(!x) return;
        rec.pb(x);
        if(ls) del(ls);
        if(rs) del(rs);
    }
    int kth(int x,int k)
    {
        pushdown(x);
        if(k>siz[ls]&&k<=siz[ls]+cnt[x]) return val[x];
        if(siz[ls]>=k) return kth(ls,k);
        else return kth(rs,k-siz[ls]-cnt[x]);
    }
    void remove(int line)
    {
        pr x=split(root,line-1);
        if(x.fi) ans+=siz[x.fi];
        del(x.fi),root=x.se;
    }
};
```

4.替罪羊树

当插入后某颗子树不平衡，就重构这颗子树

```
namespace Scapegoat
{
    const double alpha = 0.75;
    int root,node_cnt,last=0;
    int ch[MAXN][2],fa[MAXN];
    int siz[MAXN],cnt[MAXN],val[MAXN];
    int realsiz[MAXN];
    vector<int> rec;

    void debug(int x=root)
    {
        if(ls) debug(ls);
        cout << dbgs4(x,val[x],siz[x],cnt[x]) << " "  << dbgs3(ls,rs,fa[x]) <<
endl;
```

```
        if(rs) debug(rs);
    }
    int newnode(int v,int par)
    {
        int x;
        if(!rec.empty()) x=rec.back(),rec.ppb();
        else x=++node_cnt;
        realsiz[x]=cnt[x]=siz[x]=1;
        ls=rs=0,val[x]=v,fa[x]=par;
        return x;
    }
    void pushup(int x)
    {
        siz[x]=siz[ls]+cnt[x]+siz[rs];
        realsiz[x]=realsiz[ls]+realsiz[rs]+1;
    }
    bool balance(int x)
    {
        return realsiz[ls]<=realsiz[x]*alpha&&realsiz[rs]<=realsiz[x]*alpha;
    }
    void init(){node_cnt=0;rec.clear();}
    void insert(int &x,int v,int par)
    {
        if(!x){x=newnode(v,par);return;}
        if(val[x]==v){cnt[x]++,siz[x]++;return;}
        if(v<val[x]) insert(ls,v,x);
        else insert(rs,v,x);
        pushup(x); if(!balance(x)) last=x;
    }
    void remove(int x,int v)
    {
        if(val[x]==v){if(cnt[x])cnt[x]--,siz[x]--;return;}
        if(v<val[x]) remove(ls,v);
        else remove(rs,v);
        pushup(x);
    }
    inline bool chk(int x){return ch[fa[x]][1]==x;}
    int seq[MAXN],n=0;
    void dfs(int x)
    {
        if(ls) dfs(ls);
        if(cnt[x]) seq[++n]=x;
        else rec.pb(x);
        if(rs) dfs(rs);
    }
    int build(int l,int r,int par)
    {
        if(r<l) return 0;
        int x=seq[mid];
        ls=build(l,mid-1,x);
        rs=build(mid+1,r,x);
        fa[x]=par,pushup(x);
        return x;
    }
```

```
    void rebuild()
    {
        if(!last) return;
        n=0,dfs(last);
        if(!fa[last]) root=build(1,n,0);
        else ch[fa[last]][chk(last)]=build(1,n,fa[last]);
        last=0;
    }

    int rank(int x,int v)
    {
        if(!x) return 0;
        if(val[x]==v) return siz[ls];
        if(v<val[x]) return rank(ls,v);
        else return siz[ls]+cnt[x]+rank(rs,v);
    }
    int kth(int x,int k)
    {
        if(k>siz[ls]&&k<=siz[ls]+cnt[x]) return val[x];
        if(siz[ls]>=k) return kth(ls,k);
        else return kth(rs,k-siz[ls]-cnt[x]);
    }
    int pre(int v)
    {
        int k=rank(root,v);
        return kth(root,k);
    }
    int aft(int v)
    {
        int k=rank(root,v+1);
        return kth(root,k+1);
    }
}
using namespace Scapegoat;

int work()
{
    init();
    int T;
    scanf("%d",&T);
    while(T--)
    {
        int opt,x; scanf("%d%d",&opt,&x);
        if(opt==1) insert(root,x,0),rebuild();
        if(opt==2) remove(root,x);
        if(opt==3) printf("%d\n",Scapegoat::rank(root,x)+1);
        if(opt==4) printf("%d\n",kth(root,x));
        if(opt==5) printf("%d\n",pre(x));
        if(opt==6) printf("%d\n",aft(x));
    }
    return 0;
}
```

(待补)5.可持久化平衡树

LCT

```
#define ls(x) ch[x][0]
#define rs(x) ch[x][1]
bool rev[MAXN];
int fa[MAXN],ch[MAXN][2],val[MAXN],sum[MAXN];
int h,q[MAXN];
bool isroot(int x){return !fa[x]||(ls(fa[x])!=x&&rs(fa[x])!=x);}
void dorev(int x){rev[x]^=1,swap(ls(x),rs(x));}
void pushdown(int x)
{
    if(rev[x])
    {
        if(ls(x))dorev(ls(x));
        if(rs(x))dorev(rs(x));
        rev[x]=0;
    }
}
void pushup(int x)
{
    sum[x]=val[x];
    if(ls(x))sum[x]^=sum[ls(x)];
    if(rs(x))sum[x]^=sum[rs(x)];
}
void rotate(int x)
{
    int y=fa[x],z=fa[y],w=(ch[y][1]==x);
    ch[y][w]=ch[x][w^1];
    if(ch[x][w^1])fa[ch[x][w^1]]=y;
    if(z)
    {
        if(ls(z)==y)ls(z)=x;
        else if(rs(z)==y)rs(z)=x;
    }
    fa[x]=fa[y],fa[y]=x,ch[x][w^1]=y;
    up(y),up(x);if(z)up(z);
}
void splay(int x)
{
    int t=x; while(!isroot(t)) q[++h]=t,t=fa[t]; q[++h]=t;
    while(h) pushdown(q[h--]);
    while(!isroot(x))
    {
        int y=fa[x];
        if(!isroot(y))
        {
            if((ls(fa[y])==y)^(ls(y)==x))rotate(x);
            else rotate(y);
        }
        rotate(x);
```

```
    }
    pushup(x);
}
void access(int x){for(int y=0;x;y=x,x=fa[x])splay(x),rs(x)=y,pushup(x);}
void makeroot(int x){access(x),splay(x),dorev(x);}
int findroot(int x){access(x),splay(x);while(ls(x))x=ls(x);return x;}
void link(int x,int y){makeroot(x);if(findroot(y)==x)return;fa[x]=y,access(x);}
void cutfa(int x){access(x),splay(x);fa[ls(x)]=0,ls(x)=0,pushup(x);}
void cut(int x,int y){makeroot(x),access(y),splay(y);if(ls(y)!=x)return;cutfa(y);}
void change(int x,int v){splay(x),val[x]=v;}
void query(int x,int y){makeroot(x),access(y),splay(y);printf("%d\n",sum[y]);} //
别忘记splay
```

用法总结参考自：

1. LCT总结——应用篇（附题单）（LCT）FlashHu
2. LCT模板 LCT题型大荟萃 snowy_smile

虚边 u->v(fa[u]=v&&ls(v)!=u&&rs(v)!=u) 对应树上的边为 findroot(u) -> v

LCT 维护边双

BZOJ2959

LCT 维护内向基环树

LCT 维护虚子树信息

假设维护树的大小 $siz[x]$，为每个点新增一个 ${}_{s}iz[x]$ 表示所有虚儿子的 siz 之和

对 LCT 的模板操作进行一些修改：

1. $pushup$: $siz[x] = siz[ls] + siz[rs] +_{s} iz[x]$
2. $splay$: 无影响，不改变虚边连接，信息会经由 $pushup$ 正确更新
3. $access$: x->rs(x) 虚变实, x->y 实变虚

```
void access(int x)
{
    for(int y=0;x;y=x,x=fa[x])
    {
        splay(x);
        _siz[x]+=siz[rs(x)];
        _siz[x]-=siz[y];
        rs(x)=y,up(x);
    }
}
```

4. $makeroot$：无影响

5. $findroot$：无影响

6. $link$：x->y 会使 y 新增一个虚儿子，注意要把 y 转到根($access+splay$)，不然会对 y 到根的路径上的 siz 与 ${}_{s}iz$ 产生影响

```
inline void link(int x,int y)
{
    makeroot(x),makeroot(y),
    splay(y),
    fa[x]=y,
    _siz[y]+=siz[x],
    up(y),
    access(x);
}
```

7. cut: x->y 断掉一条实边，$pushup$即可

若维护的是最大值等信息，则需用 set 等数据结构来维护虚儿子的答案，复杂度多一个 log

LCT 维护树上路径信息

LCT 维护同色联通块

左偏树(可并堆)

```
int n,m,v;
struct {int dis,val,ch[2],rt;}t[MAXN];
int find(int x){return t[x].rt==x?x:t[x].rt=find(t[x].rt);}
int merge(int x,int y)
{
    if(x*y==0)return x+y;
    if((t[x].val>t[y].val)||(t[x].val==t[y].val&&x>y))swap(x,y);
    rs=merge(rs,y);
    if(t[ls].dis<t[rs].dis)swap(ls,rs);
    t[x].dis=t[rs].dis+1,t[ls].rt=t[rs].rt=x;
    return x;
}
void pop(int x){t[x].val=-1,t[ls].rt=ls,t[rs].rt=rs,t[x].rt=merge(ls,rs);}
int work()
{
    scanf("%d%d",&n,&m),t[0].dis=-1;
    fp(i,1,n)scanf("%d",&v),t[i].rt=i,t[i].val=v;
    fp(i,1,m)
    {
        int opt,x,y;scanf("%d",&opt);
        if(opt==1)
        {
            scanf("%d%d",&x,&y);
            if(t[x].val==-1||t[y].val==-1)continue;
            x=find(x),y=find(y);
```

```
            if(x!=y)t[x].rt=t[y].rt=merge(x,y);
        }
        if(opt==2)
        {
            scanf("%d",&x);
            if(t[x].val==-1)puts("-1");
            else x=find(x),printf("%d\n",t[x].val),pop(x);
        }
    }
    return 0;
}
```

单调栈 & 单调队列

树链剖分

1.重链剖分

```
int n,m,root;
LL  a[MAXN],w[MAXN];
int pre[MAXN];

int st[MAXN],ed[MAXN],cnt; //维护子树和
int top[MAXN],deep[MAXN],fa[MAXN],son[MAXN],siz[MAXN];

struct edge{int u,v,next;};
vector<edge> e = {edge()};
void addedge(int u,int v){e.pb({u,v,pre[u]}),pre[u]=e.size()-1;}

void dfs1(int u)
{
    siz[u]=1;
    go(u)if(v!=fa[u])
    {
        deep[v]=deep[u]+1,fa[v]=u,dfs1(v),siz[u]+=siz[v];
        if(siz[v]>siz[son[u]]) son[u]=v;
    }
}
void dfs2(int u,int tp)
{
    st[u]=++cnt,top[u]=tp;
    if(!son[u]){ed[u]=cnt;return;}
    dfs2(son[u],tp);
    go(u)if(v!=fa[u]&&v!=son[u])dfs2(v,v);
    ed[u]=  cnt;
}

void query(int u,int v)
{
    LL ans=0;
    int fu=top[u],fv=top[v];
```

```
    while(fu!=fv)
    {
        if(deep[fu]<deep[fv])
            swap(u,v),swap(fu,fv);
        ans+=sgt::query(1,n,st[fu],st[u],1),ans%=MOD;
        u=fa[fu],fu=top[u];
    }
    if(st[u]>st[v]) swap(u,v);
    io.write((ans+sgt::query(1,n,st[u],st[v],1))%MOD),io.push('\n');
}

void update(int u,int v,LL add)
{
    int fu=top[u],fv=top[v];
    while(fu!=fv)
    {
        if(deep[fu]<deep[fv])
            swap(u,v),swap(fu,fv);
        sgt::update(1,n,st[fu],st[u],add,1);
        u=fa[fu],fu=top[u];
    }
    if(st[u]>st[v]) swap(u,v);
    sgt::update(1,n,st[u],st[v],add,1);
}

void init()
{
    deep[root]=1;
    dfs1(root),dfs2(root,root);
    fp(i,1,n) w[st[i]]=a[i];
    sgt::build(w,1,n,1);
}
int work()
{
    io.read(n),io.read(m),io.read(root),io.read(MOD);
    fp(i,1,n) io.read(a[i]);
    fp(i,1,n-1)
    {
        int u,v;
        io.read(u),io.read(v);
        addedge(u,v),addedge(v,u);
    }
    init();
    while(m--)
    {
        int opt,x,y; LL z;
        io.read(opt);
        if(opt==1) io.read(x),io.read(y),io.read(z),update(x,y,z);
        if(opt==2) io.read(x),io.read(y),query(x,y);
        if(opt==3) io.read(x),io.read(z),sgt::update(1,n,st[x],ed[x],z,1);
        if(opt==4)
io.read(x),io.write(sgt::query(1,n,st[x],ed[x],1)),io.push('\n');
    }
```

```
    return 0;
}
```

2.dsu on tree

树上启发式合并 用来解决一类子树询问问题

```
int n;
vector<int> G[MAXN];
int d[MAXN],sz[MAXN],son[MAXN],c[MAXN];
LL ans[MAXN];
bool big[MAXN];
void getson(int u,int fa)
{
    sz[u]=1;
    for(auto v:G[u])if(v!=fa)
    {
        getson(v,u),sz[u]+=sz[v];
        if(sz[v]>sz[son[u]])son[u]=v;
    }
}
int MX,cnt[MAXN];LL num;
void add(int u,int fa,int val)
{
    cnt[c[u]]+=val;
    if(cnt[c[u]]>MX) MX=cnt[c[u]],num=c[u];
    else if(cnt[c[u]]==MX) num+=c[u];
    for(auto v:G[u])
        if(v!=fa&&!big[v])
            add(v,u,val);
}
void solve(int u,int fa,bool keep)
{
    for(auto v:G[u])
        if(v!=fa&&v!=son[u])
            solve(v,u,0);
    if(son[u]) solve(son[u],u,1),big[son[u]]=1;
    add(u,fa,1),big[son[u]]=0,ans[u]=num;
    if(!keep) add(u,fa,-1),MX=num=0;
}

int work()
{
    scanf("%d",&n);
    fp(i,1,n)scanf("%d",&c[i]);
    fp(i,1,n-1)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        G[u].pb(v),G[v].pb(u);
    }
```

```
    getson(1,0),solve(1,0,0);
    fp(i,1,n)printf("%lld ",ans[i]);
    return 0;
}
```

RMQ & 二维RMQ (ST表)

1.一维ST表

```
int n,m,a[MAXN],f[MAXN][20];
void init_st()
{
    for(int i=1;i<=n;i++) f[i][0]=a[i];
    for(int j=1;(1<<j)<=n;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
}
int query(int l,int r)
{
    int k=log2(r-l+1);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
```

2.二维ST表

```
//询问矩形内最大值
int n,m,q;
int a[MAXN][MAXN];
int f[MAXN][MAXN][9][9];
void init_st()
{
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            f[i][j][0][0]=a[i][j];
    for(int x=0;(1<<x)<=n;x++)
    {
        for(int y=0;(1<<y)<=m;y++)
        {
            if(x+y==0) continue;
            for(int i=1;i+(1<<x)-1<=n;i++)
            {
                for(int j=1;j+(1<<y)-1<=m;j++)
                {
                    if(x) f[i][j][x][y]=max(f[i][j][x-1][y],
                        f[i+(1<<(x-1))][j][x-1][y]);
                    else f[i][j][x][y]=max(f[i][j][x][y-1],
                       f[i][j+(1<<(y-1))][x][y-1]);
                }
            }
```

```
        }
    }
}
int query(int x1,int y1,int x2,int y2)
{
    int x=log2(x2-x1+1),y=log2(y2-y1+1);
    x2=x2-(1<<x)+1;
    y2=y2-(1<<y)+1;
    return max(max(f[x1][y1][x][y],f[x2][y1][x][y]),
       max(f[x1][y2][x][y],f[x2][y2][x][y]));
}
```

莫队

```
namespace MO
{
    const int SZ = 1500;
    int id(int x){return (x-1)/SZ + 1;}
    struct query
    {
        int l,r,index,ans;
        bool operator <(const query &t)
        {
            if(id(l)!=id(t.l)) return id(l)<id(t.l);
            if(id(l)&1) return r<t.r;
            else return r>t.r;
        }
    }q[MAXN];
    void solve()
    {
        sort(q+1,q+1+m);
        int l=1,r=0;
        fp(i,1,m)
        {
            while(r<q[i].r) update(a[r+1],1),r++;
            while(r>q[i].r) update(a[r], -1),r--;
            while(l<q[i].l) update(a[l], -1),l++;
            while(l>q[i].l) update(a[l-1],1),l--;
            q[i].ans=Div::query();
        }
        sort(q+1,q+1+m,[](query &a,query &b){return a.index<b.index;});
        fp(i,1,m) io.write(q[i].ans),io.push('\n');
    }
}
using namespace MO;
```

带修莫队

考虑在莫队的基础上再增加一维时间，修改相当于时间维度的变化

$O(n^{\frac{5}{3}})$，块大小取 $n^{\frac{2}{3}}$

带修数颜色

```
int n,m,tim,k,ans;
int a[MAXN],cnt[N];
int BLOCK;
inline int id(int x)
{
    return x/BLOCK;
}
struct update{int x,from,to;}b[MAXN];
struct query
{
    int l,r,t;
    int i,ans;
    bool operator <(query &x)const
    {
        if(id(l)!=id(x.l))return id(l)<id(x.l);
        if(id(r)!=id(x.r))return id(r)<id(x.r);
        return t<x.t;
    }
}q[MAXN];
inline void add(int val,int f)
{
    if(cnt[val]==0&&f== 1)ans++;
    if(cnt[val]==1&&f==-1)ans--;
    cnt[val]+=f;
}
int work()
{
    scanf("%d%d",&n,&m); BLOCK = max(1,(int)pow(n,2.0/3.0));
    //O(n^(5/3)) BLOCK = n^(2/3)
    fp(i,1,n)scanf("%d",&a[i]);
    fp(i,1,m)
    {
        char s[5];int x,y;
        scanf("%s%d%d",s,&x,&y);
        if(s[0]=='Q')q[++k]={x,y,tim,i};
        if(s[0]=='R')b[++tim]={x,a[x],y},a[x]=y;
    }
    sort(q+1,q+1+k);
    rg int l=1,r=0,cur=tim;
    fp(i,1,k)
    {
        while(r<q[i].r)r++,add(a[r],1);
        while(l>q[i].l)l--,add(a[l],1);
        while(r>q[i].r)add(a[r],-1),r--;
        while(l<q[i].l)add(a[l],-1),l++;
        while(cur<q[i].t)
        {
            cur++;
```

```
            if(b[cur].x>=l&&b[cur].x<=r)
                add(b[cur].from,-1),add(b[cur].to,1);
            a[b[cur].x]=b[cur].to;
        }
        while(cur>q[i].t)
        {
            if(b[cur].x>=l&&b[cur].x<=r)
                add(b[cur].from,1),add(b[cur].to,-1);
            a[b[cur].x]=b[cur].from;
            cur--;
        }
        q[i].ans=ans;
    }
    sort(q+1,q+1+k,[](query &a,query &b){return a.i<b.i;});
    fp(i,1,k)printf("%d\n",q[i].ans);
    return 0;
}
```

树上莫队

树上数颜色

王室联邦分块

```
const int B = 500;
int par[MAXN];
int bel[MAXN],block_cnt;
int s[MAXN],top;
void dfs(int u=1,int fa=0)
{
    int bottom = top; par[u]=fa;
    go(u)if(v!=fa)
    {
        dfs(v,u);
        if(top-bottom>=B)
        {
            block_cnt++;
            fp(i,bottom+1,top)
                bel[s[i]]=block_cnt;
            top=bottom;
        }
    }
    s[++top]=u;
}
int n,m;
int a[MAXN],cnt[MAXN],cur_ans;
bool v[MAXN];
inline void add(int u)
{
    int f=v[u]?-1:1,x=a[u];
    if(cnt[x]==0&&f== 1)cur_ans++;
```

```
    if(cnt[x]==1&&f==-1)cur_ans--;
    cnt[x]+=f,v[u]^=1;
}
inline void move(int cl,int tl)
{
    int l=lca::query(cl,tl);
    while(cl!=l)add(cl),cl=par[cl];
    while(tl!=l)add(tl),tl=par[tl];
}
vector<int> X;
inline void init(){mst(cnt,0),mst(v,0),cur_ans=block_cnt=0,X.clear();}
struct query{int l,r,i,ans;}q[MAXN];
int work()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        init();
        fp(i,1,n)scanf("%d",&a[i]),X.pb(a[i]);
        sort(all(X)),unq(X);
        fp(i,1,n)a[i]=lb(all(X),a[i])-X.begin()+1;
        fp(i,1,n-1)
        {
            int u,v;scanf("%d%d",&u,&v);
            addedge(u,v),addedge(v,u);
        }
        dfs();block_cnt++;while(top)s[top--]=block_cnt;
        lca::init();
        fp(i,1,m)scanf("%d%d",&q[i].l,&q[i].r),q[i].i=i;
        sort(q+1,q+1+m,[](const query &x,const query &y)
        {
            if(bel[x.l]!=bel[y.l])return bel[x.l]<bel[y.l];
            return bel[x.r]<bel[y.r];
        });
        int cl=1,cr=1;
        fp(i,1,m)
        {
            move(cl,q[i].l),move(cr,q[i].r),cl=q[i].l,cr=q[i].r;
            int l=lca::query(cl,cr); add(l),q[i].ans=cur_ans,add(l);
        }
        sort(q+1,q+1+m,[](const query &x,const query &y){return x.i<y.i;});
        fp(i,1,m)printf("%d\n",q[i].ans);
    }
    return 0;
}
```

回滚莫队

具体步骤

1. 对询问离线并排序，以左端点所在块号为第一关键字，右端点大小为第二关键字进行排序
2. 以块号递增的顺序处理询问，每次只考虑左端点在该块内的询问

3. 暴力回答所有左右端点都在块内的询问
4. 记当前块号为 k，区间为 $[st[k],ed[k]]$，考虑剩下的询问，显然询问的右端点都大于 $ed[k]$ 并且递增
5. 记当前已求得答案的区间为 $[l,r]$，答案为 pre，若该询问是该块内的第一次询问则初始化区间为 $[l=ed[k]+1,r=ed[k]]$
6. 扩展 r 至当前询问的右边界，并更新答案 pre
7. 建立临时变量 $ans=pre$，扩展 l 至当前询问的左边界，并更新答案 ans，扩展完毕后用 ans 回答当前询问
8. 回撤左端点 l 至 $ed[k]+1$，并**撤销扩展 l 时对辅助变量的更改**，保留 pre 用于下次询问（即回撤区间至 $[ed[k],r]$）

这样我们就实现了区间只增不减的莫队，步骤 8 中撤销左边界的具体实现视所用辅助变量而定，例如桶的话就 $cnt[value]--$，并查集的话用按秩合以支持撤销操作。

分析下复杂度，假定 n 和 m 同数量级，取块大小 $B=\sqrt{n}$，暴力处理块内询问 $O(\sqrt{n})$，每个块内右端点至多扩展 n 次 $O(n\sqrt{n})$，左端点每次最多扩展和撤销 $\sqrt{n}$ 次，因此总复杂度仍为 $O(n\sqrt{n})$。

模板

AtCoder1219 历史研究

扩展右端点：`cnt[x]++,pre=max(pre,cnt[x]*x)`

扩展右端点：`cnt[x]++,ans=max(ans,cnt[x]*x)`

回撤左端点：`cnt[x]--`

```
const int SZ = sqrt(MAXN);
vector<int> X;
int n,m,z;
int a[MAXN],b[MAXN],cnt[MAXN],id[MAXN],st[MAXN],ed[MAXN],t[MAXN];
struct query
{
    int l,r,i; LL ans;
    bool operator <(const query &t)const
    {
        if(id[t.l]!=id[l])return id[l]<id[t.l];
        return r<t.r;
    }
}q[MAXN];
inline LL baoli(int l,int r)
{
    LL ans=0;
    fp(i,l,r)t[b[i]]++,ans=max(ans,1LL*t[b[i]]*a[i]);
    fp(i,l,r)t[b[i]]--;
    return ans;
}
int work()
```

```
{
    scanf("%d%d",&n,&m);
    fp(i,1,n)scanf("%d",&a[i]),X.pb(a[i]);
    fp(i,1,n)id[i]=(i-1)/SZ+1; z=id[n];
    fp(i,1,n)if(!st[id[i]])st[id[i]]=i;
    fd(i,n,1)if(!ed[id[i]])ed[id[i]]=i;
    sort(all(X)),unq(X);
    fp(i,1,n)b[i]=lb(all(X),a[i])-X.begin()+1;
    fp(i,1,m)scanf("%d%d",&q[i].l,&q[i].r),q[i].i=i;
    sort(q+1,q+1+m);
    LL ans=0,pre=0;
    for(int i=1,l,r;i<=m;i++)
    {
        int k=id[q[i].l];
        if(id[q[i].l]!=id[q[i-1].l])mst(cnt,0),ans=pre=0,r=ed[k],l=ed[k]+1;
        if(id[q[i].l]==id[q[i].r])q[i].ans=baoli(q[i].l,q[i].r);
        else
        {
            while(l<=ed[k])cnt[b[l]]--,l++;
            while(r<q[i].r)r++,cnt[b[r]]++,pre=max(pre,1LL*cnt[b[r]]*a[r]);
            ans=pre;
            while(l>q[i].l)l--,cnt[b[l]]++,ans=max(ans,1LL*cnt[b[l]]*a[l]);
            q[i].ans=ans;
        }
    }
    sort(q+1,q+1+m,[](const query &a,const query &b){return a.i<b.i;});
    fp(i,1,m)printf("%lld\n",q[i].ans);
    return 0;
}
```

二次离线莫队

求区间内满足某性质的 $pair$ 数可以考虑

```
const int SZ = sqrt(MAXN);
int n,m,k;
int a[MAXN];
int sa[MAXN],pa[MAXN],cnt[16384*8];
LL  res[MAXN],ans[MAXN];
inline int id(int x){return (x-1)/SZ+1;}
struct q1
{
    int l,r,i;
    bool operator <(const q1 &t)const
    {
        if(id(l)!=id(t.l))return id(l)<id(t.l);
        if(id(l)&1)return r<t.r;
        return r>t.r;
    }
}q[MAXN];
struct q2{int i,l,r,f;};
```

```
vector<q2> pre[MAXN],suf[MAXN];
vector<int> b;
inline void insert_pre(int i,int pos,int l,int r,int f)
{if(pos>=1&&pos<=n)pre[pos].pb({i,l,r,f});}
inline void insert_suf(int i,int pos,int l,int r,int f)
{if(pos>=1&&pos<=n)suf[pos].pb({i,l,r,f});}
int work()
{
    io.read(n),io.read(m),io.read(k);
    fp(i,1,n)io.read(a[i]);
    fp(i,0,16384)if(bitcount(i)==k)b.pb(i);
    fp(i,1,m)io.read(q[i].l),io.read(q[i].r),q[i].i=i;
    sort(q+1,q+1+m);
    rg int r=1,l=0;
    fp(i,1,m)
    {
        if(r<q[i].r){insert_pre(i,l-
1,r+1,q[i].r,-1);while(r<q[i].r)++r,res[i]+=pa[r];}
        if(l>q[i].l){insert_suf(i,r+1,q[i].l,l-1,-1);while(l>q[i].l)--
l,res[i]+=sa[l];}
        if(r>q[i].r){insert_pre(i,l-1,q[i].r+1,r, 1);while(r>q[i].r)res[i]-
=pa[r],--r;}
        if(l<q[i].l){insert_suf(i,r+1,l,q[i].l-1, 1);while(l<q[i].l)res[i]-
=sa[l],++l;}
    }
    fp(i,1,n)
    {
        pa[i]=cnt[a[i]];
        for(auto x:b)cnt[a[i]^x]++;
        for(auto q:pre[i])fp(j,q.l,q.r)if(j)res[q.i]+=q.f*cnt[a[j]];
    }
    mst(cnt,0);
    fd(i,n,1)
    {
        sa[i]=cnt[a[i]];
        for(auto x:b)cnt[a[i]^x]++;
        for(auto q:suf[i])fp(j,q.l,q.r)if(j)res[q.i]+=q.f*cnt[a[j]];
    }
    r=1,l=0;
    fp(i,1,m)
    {
        if(r<q[i].r){while(r<q[i].r)++r,res[i]+=pa[r];}
        if(l>q[i].l){while(l>q[i].l)--l,res[i]+=sa[l];}
        if(r>q[i].r){while(r>q[i].r)res[i]-=pa[r],--r;}
        if(l<q[i].l){while(l<q[i].l)res[i]-=sa[l],++l;}
    }
    fp(i,1,m)res[i]+=res[i-1],ans[q[i].i]=res[i];
    fp(i,1,m)io.write(ans[i]),io.push('\n');
    return 0;
}
```

分块

询问分块

区间加，区间求值：

考虑对询问分块，维护前缀和，每 $\sqrt{n}$ 次修改便重建前缀数组，询问时暴力查询当前维护的修改对本次询问的影响。

```
int n;
LL  a[MAXN],b[MAXN];
struct modify{int l,r;LL v;};
vector<modify> v;
void build()
{
    mst(b,0);
    for(auto x:v)b[x.l]+=x.v,b[x.r+1]-=x.v;
    v.clear();
    fp(i,1,n)b[i]+=b[i-1];
    fp(i,1,n)b[i]+=b[i-1];
    fp(i,1,n)a[i]+=b[i];
}
pr inter(int a,int b,int c,int d)
{
    if(b<c||a>d)return {0,0};
    return {max(a,c),min(b,d)};
}
void query(int l,int r,int MOD)
{
    LL ans=a[r]-a[l-1];
    for(auto x:v)
    {
        pr seg=inter(x.l,x.r,l,r);
        if(seg.fi)ans+=1LL*(seg.se-seg.fi+1)*x.v,ans%=(MOD+1);
    }
    ans%=(MOD+1),printf("%lld\n",ans);
}
int work()
{
    scanf("%d",&n);
    fp(i,1,n)scanf("%lld",&a[i]),a[i]+=a[i-1];
    fp(i,1,n)
    {
        int t,l,r,c;scanf("%d%d%d%d",&t,&l,&r,&c);
        if(!t){v.pb({l,r,c});if(v.size()>=sqrt(n))build();}
        else query(l,r,c);
    }
    return 0;
}
```

维护 ans=kx+y 的最大值(x,y常数且多个 k可变)

将 (a,b) 看作是平面上的点

ans = kx + y

y = -kx + ans

相当于一条斜率固定的直线 从上向下平移碰到的第一个点即为ans的最大值 维护凸包即可

区间加：分块 块内维护凸包

```
struct convexhull //ans=kx+y, x y为常数,k可变(询问),求ans的最大值
{
    vector<lpr> p;
    inline void clear(){p.clear();}
    inline double slope(lpr &a,lpr &b)
    {
        if(b.fi==a.fi)return 1.0/0,0;
        return double(b.se-a.se)/double(b.fi-a.fi);
    }
    inline void insert(lpr x){p.pb(x);}
    inline LL val(int k,lpr &x){return 1LL*k*x.fi+x.se;}
    inline void build() //构建凸包 需保证插入的点按x排好序且p非空
    {
        //sort(all(p)),unq(p);
        vector<lpr> a; //swap(a,p);

        LL maxy=p.front().se;
        fp(i,1,p.size())
        {
            if(i==p.size()||p[i].fi!=p[i-1].fi)
            {
                a.pb({p[i-1].fi,maxy});
                if(i!=p.size())maxy=p[i].se;
            }
            else maxy=max(maxy,p[i].se);
        }
        p.clear();

        for(int i=0;i<a.size();i++)
        {
            while(p.size()>=2&&slope(p[p.size()-2],a[i])>
                slope(p[p.size()-2],p[p.size()-1]))
                    p.ppb();
            p.pb(a[i]);
        }
        reverse(all(p));
    }
    inline LL query(int k) //询问
    {

        while(p.size()>=2&&val(k,p[p.size()-2])>val(k,p[p.size()-1]))
            p.ppb();
        return val(k,p.back());
/*      //斜率不单调时二分用
        if(p.size()==1)return val(k,p[0]);
```

```
        if(p.size()==2)return max(val(k,p[0]),val(k,p[1]));
        if(val(k,p[0])>=val(k,p[1]))return val(k,p[0]);
        if(val(k,p[p.size()-1])>=val(k,p[p.size()-2]))return val(k,p[p.size()-1]);
        int l=1,r=p.size()-1,ans=1;
        while(l<=r)
        {
            if(val(k,p[mid])>=val(k,p[mid-1]))
                ans=mid,l=mid+1;
            else r=mid-1;
        }
        return val(k,p[ans]);
*/
    }
};

EDGE(MAXN,MAXN*2);
int l[MAXN],r[MAXN],num[MAXN],cnt;
LL sa[MAXN],sb[MAXN],a[MAXN],b[MAXN];
void dfs(int u,int fa)
{
    sa[u]+=a[u],sb[u]+=b[u],l[u]=++cnt,num[cnt]=u;
    go(u)if(v!=fa)sa[v]=sa[u],sb[v]=sb[u],dfs(v,u);
    r[u]=cnt;
}

int n,m,z;
int id[MAXN],st[MAXN],ed[MAXN];

int SZ;
LL add[MAXN];
convexhull h[MAXN];
vector<int> v[MAXN];

inline void rebuild(int k)
{
    h[k].clear();
    fp(i,st[k],ed[k])sa[num[i]]+=add[k];
    for(int t=v[k].size()-1;t>=0;t--)
    {
        int i=v[k][t],u=num[i];
        h[k].insert({-sb[u],-sa[u]*sb[u]});
    }
    for(int t=0;t<v[k].size();t++)
    {
        int i=v[k][t],u=num[i];
        h[k].insert({sb[u],sa[u]*sb[u]});
    }
    h[k].build(),add[k]=0;
}
inline void init()
{
    fp(i,1,n)id[i]=(i-1)/SZ+1; z=id[n];
    fp(i,1,n)if(!st[id[i]])st[id[i]]=i;
    fd(i,n,1)if(!ed[id[i]])ed[id[i]]=i;
```

```
    fp(k,1,z)
    {
        fp(i,st[k],ed[k])v[k].pb(i);
        sort(all(v[k]),[](int i,int j)
        {
            return sb[num[i]]<sb[num[j]];
        });
        rebuild(k);
    }
}
inline LL val(int i)
{
    int u=num[i],k=id[i];
    return abs(sa[u]+add[k])*sb[u];
}
inline LL query(int l,int r)
{
    LL ans=-linf;
    if(id[l]==id[r])
    {
        fp(i,l,r)ans=max(ans,val(i));
        return ans;
    }
    int L=id[l]+1,R=id[r]-1;
    fp(i,l,st[L]-1)ans=max(ans,val(i));
    fd(i,r,ed[R]+1)ans=max(ans,val(i));
    fp(k,L,R)ans=max(ans,h[k].query(add[k]));
    return ans;
}
inline void update(int l,int r,LL v)
{
    if(id[l]==id[r])
    {
        fp(i,l,r)sa[num[i]]+=v;
        rebuild(id[l]);
        return;
    }
    int L=id[l]+1,R=id[r]-1;
    fp(i,l,st[L]-1)sa[num[i]]+=v;
    fd(i,r,ed[R]+1)sa[num[i]]+=v;
    fp(k,L,R)add[k]+=v;
    rebuild(id[l]),rebuild(id[r]);
}

FASTIO;

inline int work()
{
    io.read(n),io.read(m);
    SZ=max(1,(int)sqrt(n/10));
    fp(i,2,n)
    {
        int fa;
        io.read(fa);
```

```
            addedge(fa,i);
        }
        fp(i,1,n)io.read(a[i]);
        fp(i,1,n)io.read(b[i]);
        dfs(1,0);
        fp(i,1,n)sb[i]=abs(sb[i]);
        init();
        while(m--)
        {
            int t,x,y; io.read(t);
            if(t==1)io.read(x),io.read(y),update(l[x],r[x],y);
            if(t==2)
            {
                io.read(x);
                printf("%lld\n",query(l[x],r[x]));
            }
        }
        return 0;
    }
```

值域分块

$O(1)$修改，$O(sqrt(n))$查询，有时可用于莫队

树分快

随即散点

从一个关键点跳到另一个关键点期望距离 $\sqrt{n}$

王室联邦

保证块的大小 $[B,3B]$，直径和个数，不保证连通性

```
EDGE(MAXN,MAXN*2);
int n,B;
int s[MAXN],top;
int bel[MAXN],root[MAXN],cnt;
void dfs(int u,int fa)
{
    int bottom = top;
    go(u)if(v!=fa)
    {
        dfs(v,u);
        if(top-bottom>=B)
        {
            cnt++,root[cnt]=u;
            while(top!=bottom)
                bel[s[top--]]=cnt;
        }
    }
```

```
    s[++top]=u;
}
int work()
{
    scanf("%d%d",&n,&B);
    fp(i,1,n-1)
    {
        int u,v;scanf("%d%d",&u,&v);
        addedge(u,v),addedge(v,u);
    }
    dfs(1,0);while(top)bel[s[top--]]=cnt;
    printf("%d\n",cnt);
    fp(i,1,n)printf("%d ",bel[i]);printf("\n");
    fp(i,1,cnt)printf("%d ",root[i]);printf("\n");
    return 0;
}
```

height分块

保证直径 联通

```
int s[MAXN],h=0;
int dfs(int u=1,int fa=0)
{
    d[u]=d[fa]+1,s[++h]=u,update(root[fa],root[u],a[u],d[u]),::fa[u]=fa;
    int mx=d[u]; go(u)if(v!=fa)mx=max(mx,dfs(v,u));
    if(u==1||mx-d[u]>=H)
    {
        ++tree_block_cnt;
        while(s[h]!=u)bel[s[h]]=tree_block_cnt,h--;
        bel[u]=tree_block_cnt,h--,rt[tree_block_cnt]=u;
        return d[u]-1;
    }
    return mx;
}
```

CDQ分治

1.二维偏序

一维sort cdq时用双指针统计答案

```
namespace cdq
{
    int n;
    struct opt
    {
        LL x,y,val,f;
        bool query;
```

```
        bool operator < (opt & other){return y<other.y;}
        void print()
        {
            cout << dbgs3(x,y,query) << endl;
            cout << dbgs2(val,f) << endl;
        }
    }q[MAXN];

    void sort(opt a[],int l,int r)
    {
        if(l==r) return;
        static opt t[MAXN];
        int head=0,i=l,j=mid+1;
        while(i<=mid&&j<=r) t[++head]=a[i]<a[j]?a[i++]:a[j++];
        while(i<=mid) t[++head]=a[i++];
        while(j<=r  ) t[++head]=a[j++];
        fp(i,1,head) a[l+i-1]=t[i];
    }
    void solve(LL ans[],int l=1,int r=n)
    {
        if(l==r) return;

        solve(ans,l,mid),solve(ans,mid+1,r);
/*
        cout << dbgs2(l,r) << endl;
        fp(i,l,r) q[i].print();
        cout << endl;
*/
        int i=l;
        LL sum=0;
        fp(j,mid+1,r)
        {
            while(i<=mid&&q[i].y<=q[j].y)
                sum+=q[i++].val;
            if(q[j].query) ans[q[j].x]+=q[j].f*sum;
        }
        sort(q,l,r);
    }
    void push(opt x){q[++n]=x;}
}

int n,m;
LL a[MAXN],ans[MAXN];
bool query[MAXN];

int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,n) scanf("%lld",&a[i]);
    fp(i,1,n) a[i]+=a[i-1];
    fp(i,1,m)
    {
        static int opt,l,r;
        scanf("%d%d%d",&opt,&l,&r);
```

```
        if(opt==1) cdq::push({i,l,r,0,0});
        else
        {
            query[i]=1;
            ans[i]=a[r]-a[l-1];
            cdq::push({i,r,0,1,1});
            cdq::push({i,l-1,0,-1,1});
        }
    }
    cdq::solve(ans);
    fp(i,1,m) if(query[i]) printf("%lld\n",ans[i]);
    return 0;
}
```

2.三维偏序

陌上花开

```
namespace bit
{
    int c[N];
    vector<pr> q;
    int lowbit(int x){return x&-x;}
    int sum(int x)
    {
        int s=0;
        for(int i=x;i;i-=lowbit(i)) s+=c[i];
        return s;
    }
    void update(int x,int v,bool save=1)
    {
        if(save) q.pb({x,v});
        for(int i=x;i<N;i+=lowbit(i)) c[i]+=v;
    }
    void undo()
    {
        while(!q.empty())
            update(q.back().fi,-q.back().se,0),q.ppb();
    }
    int query(int l,int r){return sum(r)-sum(l-1);}
}

namespace cdq
{
    int n,m;
    struct query
    {
        int x,y,z;
        int index,cnt,ans;
        bool operator ==(const query &t)
        {
```

```
            return x==t.x&&y==t.y&&z==t.z;
        }
        bool operator <(const query &t)
        {
            if(y!=t.y) return y<t.y;
            return z<t.z;
        }
    }q[MAXN],t[MAXN];

    void sort(int l,int r)
    {
        static query t[MAXN];
        int i=l,j=mid+1,head=0;
        while(i<=mid&&j<=r) t[++head]=(q[i]<q[j])?q[i++]:q[j++];
        while(i<=mid) t[++head]=q[i++];
        while(j<=  r) t[++head]=q[j++];
        fp(i,1,head) q[l+i-1]=t[i];
    }
    void solve(int l=1,int r=n)
    {
        if(l==r) return;
        solve(l,mid),solve(mid+1,r);
/*
        cout << dbgs2(l,r) << endl;
        fp(i,l,r) cout << dbgs4(i,q[i].x,q[i].y,q[i].z) << dbgs(q[i].cnt) << endl;
        cout << endl;
*/
        for(int i=l,j=mid+1;j<=r;j++)
        {
            while(i<=mid&&q[i].y<=q[j].y)
                bit::update(q[i].z,q[i].cnt),i++;
            q[j].ans+=bit::query(1,q[j].z);
        }

        bit::undo(),sort(l,r);
    }
    void push(const query x){q[++n]=x;}
    bool cmp(const cdq::query &a,const cdq::query &b)
    {
        if(a.x!=b.x) return a.x<b.x;
        if(a.y!=b.y) return a.y<b.y;
        return a.z<b.z;
    }
    void init()
    {
        std::sort(q+1,q+1+n,cmp);
        query cur=q[1];
        for(int i=2,cnt=1;i<=n+1;i++)
        {
            if(i==n+1||!(q[i]==cur))
                cur.cnt=cnt,t[++m]=cur,cur=q[i],cnt=1;
            else cnt++;
        }
        swap(t,q),swap(n,m);
```

```
    }

};

int n,k;
int f[MAXN];

vector<int> X;

int work()
{
    io.read(n),io.read(k);
    fp(i,1,n)
    {
        static int x,y,z;
        io.read(x),io.read(y),io.read(z);
        cdq::push({x,y,z,i,0});
    }
    cdq::init();
    cdq::solve();
    fp(i,1,cdq::n) f[cdq::q[i].ans+cdq::q[i].cnt-1]+=cdq::q[i].cnt;
    fp(i,0,n-1) io.write(f[i]),io.push('\n');
    return 0;
}
```

3.四维偏序 CDQ套CDQ

solve1(l,r): 考虑处理完 solve1(l,mid) 与 solve2(mid+1,r) 后,仅有跨越 mid 的偏序关系尚未处理

因此可以将 (l,mid) 的点第一维都设为 0, (mid+1,r) 的点的第一维都设为 1, 继续 cdq 分治处理三维偏序即可

```
namespace bit
{
    int c[N],vis[N],tim=1;
    int lowbit(int x){return x&-x;}
    void update(int x,int v)
    {
        for(int i=x;i<N;i+=lowbit(i))
        {
            if(vis[i]!=tim) c[i]=0;
            vis[i]=tim,c[i]+=v;
        }
    }
    int sum(int x)
    {
        int s=0;
        for(int i=x;i;i-=lowbit(i))
            if(vis[i]==tim)
                s+=c[i];
        return s;
    }
```

```
    int query(int l,int r){return sum(r)-sum(l-1);}
};

struct node
{
    int a,b,c,d,id,f;
}a[MAXN],b[MAXN],t[MAXN];

int n,ans[MAXN];

void solve1(int l,int r);
void solve2(int l,int r);

void solve1(int l,int r)
{

    if(l==r) return;
    solve1(l,mid),solve1(mid+1,r);

    //cout << "solve 1 " << dbgs2(l,r) << endl;
    //debug(a,l,r);

    fp(i,l,mid)   a[i].a=0;
    fp(j,mid+1,r) a[j].a=1;

    int i=l,j=mid+1,head=l;
    while(i<=mid&&j<=r) b[head++]=(a[i].b<=a[j].b)?a[i++]:a[j++];
    while(i<=mid) b[head++]=a[i++];
    while(j<=  r) b[head++]=a[j++];
/*
    fp(i,l,mid)   b[i].a=0;
    fp(j,mid+1,r) b[j].a=1;
*/
    fp(i,l,r) a[i]=b[i];

    solve2(l,r);
}

void solve2(int l,int r)
{

    if(l==r) return;
    solve2(l,mid),solve2(mid+1,r);

    //cout << "solve2 " << dbgs2(l,r) << endl;
    //debug(b,l,r);

    bit::tim++;

    int i=l,j=mid+1,head=l;
    while(i<=mid&&j<=r)
    {
        if(b[i].c<=b[j].c)
        {
```

```
            t[head++]=b[i];
            if(!b[i].a&&!b[i].f) bit::update(b[i].d,1);
            i++;
        }
        else
        {
            t[head++]=b[j];
            if(b[j].a&&b[j].f) ans[b[j].id]+=b[j].f*bit::query(1,b[j].d);
            j++;
        }
    }
    while(i<=mid) t[head++]=b[i++];
    while(j<=r)
    {
        t[head++]=b[j];
        if(b[j].a&&b[j].f) ans[b[j].id]+=b[j].f*bit::query(1,b[j].d);
        j++;
    }
    fp(i,l,r) b[i]=t[i];
}

int T,q;
vector<int> X;

int work()
{
    io.read(T);
    while(T--)
    {
        bit::tim=n=0,X.clear();
        io.read(q);
        fp(i,1,q)
        {
            int opt,x1,x2,y1,y2,z1,z2;
            io.read(opt);
            io.read(x1),io.read(y1),io.read(z1);
            if(opt==1) a[++n]={i,x1,y1,z1,i,0},ans[i]=-1,X.pb(z1);
            else
            {
                io.read(x2),io.read(y2),io.read(z2);
                x1--,y1--,z1--;
                a[++n]={i,x1,y1,z1,i,-1};
                a[++n]={i,x1,y2,z1,i, 1};
                a[++n]={i,x2,y1,z1,i, 1};
                a[++n]={i,x2,y2,z1,i,-1};
                a[++n]={i,x1,y1,z2,i, 1};
                a[++n]={i,x1,y2,z2,i,-1};
                a[++n]={i,x2,y1,z2,i,-1};
                a[++n]={i,x2,y2,z2,i, 1};
                X.pb(z1),X.pb(z1+1),X.pb(z2);
                ans[i]=0;
            }
        }
```

```
        sort(all(X)),unq(X);
        fp(i,1,n) a[i].d=lower_bound(all(X),a[i].d)-X.begin()+1;
        //debug(a,1,n);

        solve1(1,n);
        fp(i,1,q) if(ans[i]!=-1) io.write(ans[i]),io.push('\n');
    }
    return 0;
}
```

4.带插入曼哈顿距离最近点对(天使玩偶)

Luogu P4169

插入:在二维平面上添加一个点，询问：给定一个点，查询已插入点中与他曼哈顿距离最近的点

SOL:假设有两个点 (X_i,Y_i) 和 (X_j,Y_j)，点 j 在 点 i 的左下方，则他们的曼哈顿距离为 $X_i+Y_j-(X_j+Y_j)$，即对于每一个点 i 查询插入时间在他之前的，在他左下角的所有点 j 的 X_j+Y_j 的最大值，是个三维偏序问题。

其他方向的点(左上，右下，右上)可以通过坐标变换(沿 x/y 轴对称)转到左下角

```
vector<int> X,Y;
int c[N],t[N],curt;
inline int query(int x)
{
    int ans=-inf;
    for(rg int i=x;i;i-=(i&(-i)))
        if(t[i]==curt)
            ans=max(ans,c[i]);
    return ans;
}
inline void update(int x,int v)
{
    for(rg int i=x;i<N;i+=(i&(-i)))
        if(t[i]!=curt)c[i]=v,t[i]=curt;
        else c[i]=max(c[i],v);
    //cout << dbgs(v) << endl;
}

int n,m,tim,h;
int ans[MAXN];
struct point
{
    int t,x,y;
    bool query;
}p[MAXN],q[MAXN],a[MAXN];
void solve(int l,int r)
{
    if(l==r)return; solve(l,mid),solve(mid+1,r);
```

```
    curt++;
    for(int i=l,j=mid+1;j<=r;j++)
    {
        while(i<=mid&&p[i].x<=p[j].x)
        {
            if(!p[i].query)update(p[i].y,p[i].x+p[i].y);
            i++;
        }
        if(p[j].query)
        {
            ans[p[j].t]=min(ans[p[j].t],p[j].x+p[j].y-query(p[j].y));
            //cout << dbgs(query(p[j].y)) << endl;
        }
    }
    int i=l,j=mid+1,h=0;
    while(i<=mid&&j<=r)q[++h]=(p[i].x<p[j].x)?p[i++]:p[j++];
    while(i<=mid)q[++h]=p[i++];
    while(j<=  r)q[++h]=p[j++];
    fp(i,l,r)p[i]=q[i-l+1];
}
inline void gao()
{
    fp(i,1,n)
    {
        p[i]=a[i];
        if(p[i].x<=0)p[i].x+=1e6;
        if(p[i].y<=0)p[i].y+=1e6;
        //cout << dbgs4(p[i].t,p[i].x,p[i].y,p[i].query) << endl;
        //p[i].x=lb(all(X),p[i].x)-X.begin()+1;
        //p[i].y=lb(all(Y),p[i].y)-Y.begin()+1;
    }
    //cout << endl;
    solve(1,n);
}
FASTIO;
int work()
{
    io.read(n),io.read(m);
    fp(i,1,n)
    {
        int x,y;io.read(x),io.read(y);
        a[++h]={++tim,x,y};
    }
    fp(i,1,m)
    {
        int t,x,y;io.read(t),io.read(x),io.read(y);
        if(t==1)a[++h]={++tim,x,y};
        if(t==2)a[++h]={++tim,x,y,1};
    }

    //for(auto x:X)printf("%d ",x);printf("\n");
    //for(auto x:Y)printf("%d ",x);printf("\n");

    n=h;fp(i,1,n)ans[i]=inf;gao();
```

```
    fp(i,1,n)a[i].x=-a[i].x;gao();
    fp(i,1,n)a[i].y=-a[i].y;gao();
    fp(i,1,n)a[i].x=-a[i].x;gao();

    fp(i,1,n)if(a[i].query)
    printf("%d\n",ans[i]);
    //io.write(ans[i]),io.push('\n');
    return 0;
}
```

(待补)整体二分

树套树

1.线段树套线段树

2.树状数组套动态开点线段树

单点修改区间k大

```
vector<int> X;

namespace sgt
{
    int ls[MAXN*400],rs[MAXN*400],s[MAXN*400],cnt;
    void pushup(int cnt){s[cnt]=s[ls[cnt]]+s[rs[cnt]];}
    void update(int &rt,int l,int r,int pos,int v)
    {
        if(!rt) rt=++cnt;
        if(l==r){s[rt]+=v;return;}
        if(pos<=mid) update(ls[rt],l,mid,pos,v);
        else update(rs[rt],mid+1,r,pos,v);
        pushup(rt);
    }
}

int n,m,root[MAXN],a[MAXN];
int lowbit(int x){return x&-x;}
void update(int pos,int v,bool init=0)
{
    if(!init)
    for(int i=pos;i<=n;i+=lowbit(i))
        sgt::update(root[i],1,X.size(),a[pos],-1);
    a[pos]=v;
    for(int i=pos;i<=n;i+=lowbit(i))
        sgt::update(root[i],1,X.size(),a[pos], 1);
}
vector<int> add,sub;
int query(int L,int R,int k)
{
    add.clear(),sub.clear();
```

```
    for(int i=R  ;i;i-=lowbit(i)) add.pb(root[i]);
    for(int i=L-1;i;i-=lowbit(i)) sub.pb(root[i]);
    int l=1,r=X.size();
    while(l!=r)
    {
        int s=0;
        for(auto x:add) s+=sgt::s[sgt::ls[x]];
        for(auto x:sub) s-=sgt::s[sgt::ls[x]];
        if(s>=k)
        {
            for(auto &x:add) x=sgt::ls[x];
            for(auto &x:sub) x=sgt::ls[x];
            r=mid;
        }
        else
        {
            for(auto &x:add) x=sgt::rs[x];
            for(auto &x:sub) x=sgt::rs[x];
            k-=s,l=mid+1;
        }
    }
    return X[l-1];
}

int L[MAXN],R[MAXN],K[MAXN];

int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,n) scanf("%d",&a[i]),X.pb(a[i]);
    fp(i,1,m)
    {
        char t[5];
        scanf("%s",t);
        scanf("%d%d",&L[i],&R[i]);
        if(t[0]=='Q') scanf("%d",&K[i]);
        else K[i]=0,X.pb(R[i]);
    }
    sort(all(X)),unq(X);
    fp(i,1,m) if(!K[i]) R[i]=lower_bound(all(X),R[i])-X.begin()+1;
    fp(i,1,n) a[i]=lower_bound(all(X),a[i])-X.begin()+1;
    fp(i,1,n) update(i,a[i],1);
    fp(i,1,m)
    {
        if(!K[i]) update(L[i],R[i]);
        else printf("%d\n",query(L[i],R[i],K[i]));
    }
    return 0;
}
```

3.线段树套平衡树

线段树套FHQ

```
int n, m, w[MAXN];

namespace fhq_treap
{
#define ls ch[x][0]
#define rs ch[x][1]

    vector<int> rec;

    int cnt;
    int ch[N][2], siz[N], fix[N], val[N];

    void init() { srand(20000713), cnt = 0, rec.clear(); }
    int newnode(int v)
    {
        int x;
        if (!rec.empty()) x = rec.back(), rec.pop_back();
        else x = ++cnt;
        fix[x] = rand(), ls = rs = 0;
        siz[x] = 1, val[x] = v;
        return x;
    }

    struct fhq_treap
    {
        int root;

        void init() { root = 0; }
        void pushup(int x) { siz[x] = siz[ls] + siz[rs] + 1; }
        void pushdown(int x) {}
        fhq_treap() { init(); }

        pr split(int x, int k)
        {
            if (!x) return { 0,0 };
            pr t; pushdown(x);
            if (k <= siz[ls]) t = split(ls, k), ls = t.second, t.second = x;
            else t = split(rs, k - siz[ls] - 1), rs = t.first, t.first = x;
            pushup(x); return t;
        }

        int merge(int x, int y)
        {
            if (x*y == 0) return x + y;
            pushdown(x), pushdown(y);
            if (fix[x] < fix[y]) { ch[x][1] = merge(ch[x][1], y), pushup(x);
return x; }
            else { ch[y][0] = merge(x, ch[y][0]), pushup(y); return y; }
        }

        int rank(int x, int v)
```

```
        {
            if (!x) return 0;
            pushdown(x);
            if (val[x] < v) return siz[ls] + 1 + rank(rs, v);
            else return rank(ls, v);
        }

        void insert(int v)
        {
            int k = rank(root, v);
            pr t = split(root, k);
            root = merge(merge(t.fi, newnode(v)), t.se);
        }

        int kth(int k)
        {
            pr a = split(root, k - 1);
            pr b = split(a.se, 1);
            root = merge(merge(a.fi, b.fi), b.se);
            return val[b.fi];
        }

        void remove(int v)
        {
            pr a = split(root, rank(root, v));
            pr b = split(a.se, 1);
            root = merge(a.fi, b.se);
            rec.push_back(b.fi);
        }

        int pre(int v) { return kth(rank(root, v)); }
        int suc(int v) { return kth(rank(root, v + 1) + 1); }

        void debug(int x)
        {
            if (!x) return;
            if (ls) debug(ls);
            cout << dbgs2(x, val[x]) << endl;
            if (rs) debug(rs);
        }

    }tr[MAXN * 4];

#undef ls
#undef rs
}

namespace seg_tree
{
#define ls (x<<1)
#define rs ((x<<1)|1)
#define mid ((l+r)>>1)

    void build(int l, int r, int x)
```

```
    {
        for (int i = l; i <= r; i++)
            fhq_treap::tr[x].insert(w[i]);
        if (l == r) return;
        build(l, mid, ls), build(mid + 1, r, rs);
    }

    void modify(int l, int r, int pos, int v, int x)
    {
        fhq_treap::tr[x].remove(w[pos]);
        fhq_treap::tr[x].insert(v);
        if (l == r) return;
        if (pos <= mid) modify(l, mid, pos, v, ls);
        else modify(mid + 1, r, pos, v, rs);
    }

    int rank(int l, int r, int nl, int nr, int v, int x)
    {
        if (l == nl && r == nr)
            return fhq_treap::tr[x].rank(fhq_treap::tr[x].root, v);
        if (nr <= mid) return rank(l, mid, nl, nr, v, ls);
        if (nl > mid) return rank(mid + 1, r, nl, nr, v, rs);
        return rank(l, mid, nl, mid, v, ls) + rank(mid + 1, r, mid + 1, nr, v,
rs);
    }

    int pre(int l, int r, int nl, int nr, int v, int x)
    {
        if (l == nl && r == nr)
        {
            int res = fhq_treap::tr[x].pre(v);
            return res < v ? res : -inf;
        }
        if (nr <= mid) return pre(l, mid, nl, nr, v, ls);
        if (nl > mid) return pre(mid + 1, r, nl, nr, v, rs);
        return max(pre(l, mid, nl, mid, v, ls), pre(mid + 1, r, mid + 1, nr, v,
rs));
    }

    int suc(int l, int r, int nl, int nr, int v, int x)
    {
        if (l == nl && r == nr)
        {
            int res = fhq_treap::tr[x].suc(v);
            return res > v ? res : inf;
        }
        if (nr <= mid) return suc(l, mid, nl, nr, v, ls);
        if (nl > mid) return suc(mid + 1, r, nl, nr, v, rs);
        return min(suc(l, mid, nl, mid, v, ls), suc(mid + 1, r, mid + 1, nr, v,
rs));
    }

#undef ls
#undef rs
```

```
#undef mid
}

inline int  rank(int nl, int nr, int v) { return seg_tree::rank(1, n, nl, nr, v,
1) + 1; }
inline void modify(int pos, int v) { seg_tree::modify(1, n, pos, v, 1), w[pos] =
v; }
inline int  pre(int nl, int nr, int v) { return seg_tree::pre(1, n, nl, nr, v, 1);
}
inline int  suc(int nl, int nr, int v) { return seg_tree::suc(1, n, nl, nr, v, 1);
}

#define mid ((l+r)>>1)
inline int  kth(int nl, int nr, int k)
{
    int l = 0, r = 1e8, ans;
    while (l <= r)
    {
        int s = seg_tree::rank(1, n, nl, nr, mid + 1, 1);
        if (s >= k) ans = mid, r = mid - 1;
        else l = mid + 1;
    }
    return ans;
}
#undef mid

int work()
{
    n = read(), m = read();
    for (int i = 1; i <= n; i++) w[i] = read();
    fhq_treap::init();
    seg_tree::build(1, n, 1);
    while (m--)
    {
        int opt = read(), l, r, k;
        if (opt == 3) l = read(), r = read(), modify(l, r);
        else
        {
            l = read(), r = read(), k = read();
            switch (opt)
            {
                case 1:printf("%d\n", ::rank(l, r, k)); break;
                case 2:printf("%d\n", ::kth(l, r, k)); break;
                case 4:printf("%d\n", ::pre(l, r, k)); break;
                case 5:printf("%d\n", ::suc(l, r, k)); break;
            }
        }
    }
    return 0;
}
```

4.平衡树套线段树

BZOJ3065 带插入区间K小值

暴力重构O(nlogn^3)

```
#define ls ch[x][0]
#define rs ch[x][1]

namespace segment_tree
{
    int node_cnt;
    int s[20000010],ch[20000010][2];
    int tre[MAXN];
    vector<int> rec;
    int newnode()
    {
        int x;
        if(!rec.empty()) x=rec.back(),rec.ppb();
        else x=++node_cnt;
        s[x]=ls=rs=0;
        return x;
    }
    void remove(int &x) //垃圾回收
    {
        if(!x) return;
        if(ls) remove(ls);
        if(rs) remove(rs);
        rec.pb(x),x=0;
    }
    void pushup(int x)
    {
        s[x]=0;
        if(ls) s[x]+=s[ls];
        if(rs) s[x]+=s[rs];
    }
    void update(int &x,int l,int r,int pos,int v)
    {
        if(!x) x=newnode();
        if(l==r){s[x]+=v;return;}
        if(pos<=mid) update(ls,l,mid,pos,v);
        else update(rs,mid+1,r,pos,v);
        pushup(x); if(!s[x]) remove(x);
    }
/*
    void debug(int x,int l,int r)
    {
        if(l==r){if(s[x])printf("%d ",l);return;}
        debug(ls,l,mid);
        debug(rs,mid+1,r);
    }
*/
};
using namespace segment_tree;
```

```
namespace scapegoat
{
    const double alpha = 0.75;

    int node_cnt,root,last;
    int siz[MAXN],val[MAXN],ch[MAXN][2],fa[MAXN];
/*
    void debug(int x=root)
    {
        if(ls) debug(ls);
        //cout << val[x] << " ";
        //cout << dbgs4(x,fa[x],ls,rs) << " ";
        //cout << dbgs3(val[x],siz[x],tre[x]) << endl;

        if(rs) debug(rs);
    }
*/
    int newnode(int v,int par)
    {
        int x=++node_cnt;
        siz[x]=1,fa[x]=par,val[x]=v,ls=rs=0,tre[x]=0;
        return x;
    }
    void pushup(int x){siz[x]=siz[ls]+siz[rs]+1;}
    bool balance(int x){return max(siz[ls],siz[rs])<=siz[x]*alpha;}
    bool chk(int x){return ch[fa[x]][1]==x;}
    int seq[MAXN],head;
    int build(int l,int r,int par)
    {
        if(r<l) return 0;
        int x=seq[mid];
        remove(tre[x]),tre[x]=segment_tree::newnode();
        fp(i,l,r) update(tre[x],0,N,val[seq[i]],1);
        ls=build(l,mid-1,x);
        rs=build(mid+1,r,x);
        fa[x]=par,pushup(x);
        return x;
    }
    void dfs(int x)
    {
        if(ls) dfs(ls);
        seq[++head]=x;
        if(rs) dfs(rs);
    }
    void rebuild()
    {
        if(!last) return;
        head=0,dfs(last);
        if(!fa[last]) root=build(1,head,0);
        else ch[fa[last]][chk(last)]=build(1,head,fa[last]);
        last=0;
    }
    void init(int a[],int n)
```

```
{
    node_cnt+=n;
    fp(i,1,n) val[i]=a[i],seq[i]=i;
    root=build(1,n,0);
}
int change(int x,int pos,int v)
{
    if(siz[ls]+1==pos)
    {
        int t=val[x]; val[x]=v;
        update(tre[x],0,N,t,-1);
        update(tre[x],0,N,v, 1);
        return t;
    }
    int pre;
    if(siz[ls]>=pos) pre=change(ls,pos,v);
    else pre=change(rs,pos-siz[ls]-1,v);
    update(tre[x],0,N,pre,-1);
    update(tre[x],0,N,v  , 1);
    return pre;
}
void insert(int &x,int pos,int v,int par)
{
    if(!x)
    {
        x=newnode(v,par);
        update(tre[x],0,N,v,1);
        return;
    }
    if(siz[ls]>=pos) insert(ls,pos,v,x);
    else insert(rs,pos-siz[ls]-1,v,x);
    update(tre[x],0,N,v,1),pushup(x);
    if(!balance(x)) last=x;
}

vector<int> cnt,a;
void find(int x,int l,int r)
{
    if(!x||l>r) return;
    if(l==1&&r==siz[x]){cnt.pb(tre[x]);return;}
    if(r<=siz[ls]) find(ls,l,r);
    else if(l>siz[ls]+1) find(rs,l-siz[ls]-1,r-siz[ls]-1);
    else a.pb(val[x]),find(ls,l,siz[ls]),find(rs,1,r-siz[ls]-1);
}
int query(int nl,int nr,int k)
{
    cnt.clear(),a.clear(),find(root,nl,nr),sort(all(a));
    int l=0,r=N;
    while(l<r)
    {
        int sum=0;
        for(auto &x:cnt) sum+=s[segment_tree::ls];
        sum+=upper_bound(all(a),mid)-lower_bound(all(a),l);
        if(sum>=k){for(auto &x:cnt) cnt[i]=segment_tree::ls;r=mid;}
```

```
            else{for(auto &x:cnt)cnt[i]=segment_tree::rs;}k-=sum,l=mid+1;}
        }
        return l;
    }
}
using namespace scapegoat;

int n,m,ans=0;
int arry[MAXN];

int work()
{
    scanf("%d",&n);
    fp(i,1,n) scanf("%d",&arry[i]);
    init(arry,n);

    scanf("%d",&m);
    while(m--)
    {
        char s[5];int x,y,k=0;
        scanf("%s",s),scanf("%d%d",&x,&y),x^=ans,y^=ans;
        if(s[0]=='Q') scanf("%d",&k),k^=ans,printf("%d\n",ans=query(x,y,k));
        if(s[0]=='M') change(root,x,y);
        if(s[0]=='I') insert(root,x-1,y,0),rebuild();
    }
    return 0;
}
```

线段树合并

KD-Tree

注意：除了矩阵覆盖以外的查询(邻域查询等)最坏复杂度都是 $O(n)$ 的，需保证 随机数据 / 出题人想让考 kd-tree

通用

```
#define ls ch[x][0]
#define rs ch[x][1]
const int K = 3; //维度
const float alpha = 0.85; //平衡因子
int n,m,last,root;
int ch[MAXN][2],fa[MAXN],cmpd[MAXN]; //节点固有属性
int siz[MAXN],mx[MAXN][K],mn[MAXN][K]; //pushup维护

bool balance(int x){return siz[ls]<=siz[x]*alpha&&siz[rs]<=siz[x]*alpha;}
int cnt,h,rec[MAXN];
int newnode()
{
    int x=h?rec[h--]:++cnt;
    ls=rs=fa[x]=cmpd[x]=siz[x]=0;
```

```
    fp(k,0,K-1)mx[x][k]=mn[x][k]=0;
    return x;
}
struct point
{
    int d[K],num=1,v=0;
    bool operator==(const point &t)const
    {
        fp(i,0,K-1)
            if(d[i]!=t.d[i])
                return 0;
        return 1;
    }
    bool operator!=(const point &t)const{return !(*this==t);}
}p[MAXN];
void pushup(int x)
{
    siz[x]=siz[ls]+siz[rs]+bool(p[x].num);
    fp(k,0,K-1)
    {
        if(p[x].num)mx[x][k]=mn[x][k]=p[x].d[k];
        else mx[x][k]=-inf,mn[x][k]=inf;
        if(ls)
        {
            mx[x][k]=max(mx[x][k],mx[ls][k]);
            mn[x][k]=min(mn[x][k],mn[ls][k]);
        }
        if(rs)
        {
            mx[x][k]=max(mx[x][k],mx[rs][k]);
            mn[x][k]=min(mn[x][k],mn[rs][k]);
        }
    }
}
int curd;bool cmp(const point &a,const point &b){return a.d[curd]<b.d[curd];}
int build(int l,int r,int par,point t[])
{
    if(l>r)return 0;
    int x=newnode();
    //curd=cmpd[x]=(cmpd[par]+1)%K; //交替选法 容易被卡

    //选方差大的一维
    db a[K]={0},v[K]={0};
    fp(k,0,K-1)
    {
        fp(i,l,r)a[k]+=t[i].d[k];
        a[k]/=(r-l+1);
        fp(i,l,r)v[k]+=sqr(a[k]-t[i].d[k]);
    }
    curd=cmpd[x]=(max_element(v,v+K)-v);

    fa[x]=par;
    nth_element(t+l,t+mid,t+r+1,cmp),p[x]=t[mid];
    ls=build(l,mid-1,x,t);
```

```
    rs=build(mid+1,r,x,t);
    pushup(x);
    return x;
}
void init(){cnt=root=last=h=0;} //多组初始化
```

邻域查询

查询 欧几里德/曼哈顿 最近/最远 点

曼哈顿最近/最远点有能保证复杂度的 cdq 分治的解法，优先考虑

```
//邻域查询 最坏O(n)
LL eudis(point a,point b){return 1LL*sqr(a.d[0]-b.d[0])+1LL*sqr(a.d[1]-b.d[1]);}
LL eumin(int x,point q) //欧几里得下界
{
    LL dis=0;
    fp(k,0,1)
    {
        if(q.d[k]<=mn[x][k])dis+=sqr(q.d[k]-mn[x][k]);
        if(q.d[k]>=mx[x][k])dis+=sqr(q.d[k]-mx[x][k]);
    }
    return dis;
}
LL eumax(int x,point q) //欧几里得上界
{
    LL dis=0;
    fp(k,0,1)dis+=max(sqr(q.d[k]-mx[x][k]),sqr(q.d[k]-mn[x][k]));
    return dis;
}
LL ans;
void qeumin(int x,point q)
{
    if(!x)return;
    LL ld=eumin(ls,q),rd=eumin(rs,q),d=eudis(p[x],q);
    if(p[x].num)ans=min(ans,d);
    if(ld<rd)
    {
        if(ld<ans)qeumin(ls,q);
        if(rd<ans)qeumin(rs,q);
    }
    else
    {
        if(rd<ans)qeumin(rs,q);
        if(ld<ans)qeumin(ls,q);
    }
}
void qeumax(int x,point q)
{
    if(!x)return;
    LL ld=eumax(ls,q),rd=eumax(rs,q),d=eudis(p[x],q);
```

```
    if(p[x].num)ans=max(ans,d);
    if(ld>rd)
    {
        if(ld>ans)qeumax(ls,q);
        if(rd>ans)qeumax(rs,q);
    }
    else
    {
        if(rd>ans)qeumax(rs,q);
        if(ld>ans)qeumax(ls,q);
    }
}

//曼哈顿距离
LL mandis(point a,point b){return abs(a.d[0]-b.d[0])+abs(a.d[1]-b.d[1]);}
LL manmin(int x,point q)
{
    LL dis=0;
    fp(k,0,1)
    {
        if(q.d[k]<=mn[x][k])dis+=abs(q.d[k]-mn[x][k]);
        if(q.d[k]>=mx[x][k])dis+=abs(q.d[k]-mx[x][k]);
    }
    return dis;
}
LL manmax(int x,point q)
{
    LL dis=0;
    fp(k,0,1)dis+=max(abs(q.d[k]-mx[x][k]),abs(q.d[k]-mn[x][k]));
    return dis;
}
//LL ans;
void qmanmin(int x,point q)
{
    if(!x)return;
    LL ld=manmin(ls,q),rd=manmin(rs,q),d=mandis(p[x],q);
    if(p[x].num)ans=min(ans,d);
    if(ld<rd)
    {
        if(ld<ans)qmanmin(ls,q);
        if(rd<ans)qmanmin(rs,q);
    }
    else
    {
        if(rd<ans)qmanmin(rs,q);
        if(ld<ans)qmanmin(ls,q);
    }
}
void qmanmax(int x,point q)
{
    if(!x)return;
    LL ld=manmax(ls,q),rd=manmax(rs,q),d=mandis(p[x],q);
    if(p[x].num)ans=max(ans,d);
    if(ld>rd)
```

```
    {
        if(ld>ans)qmanmax(ls,q);
        if(rd>ans)qmanmax(rs,q);
    }
    else
    {
        if(rd>ans)qmanmax(rs,q);
        if(ld>ans)qmanmax(ls,q);
    }
}
```

带插入删除(替罪羊树重构思想)

```
void insert(int &x,int par,point val) //val.num默认为1
{
    if(!x) //开辟新节点
    {
        x=newnode();
        p[x]=val;
        fa[x]=par;
        cmpd[x]=(cmpd[par]+1)%K;
        pushup(x);
        return;
    }
    if(val==p[x]) //处理重点
    {
        p[x].v=val.v;
        p[x].num+=val.num;
        pushup(x);
        return;
    }
    curd=cmpd[x];
    if(cmp(val,p[x]))insert(ch[x][0],x,val);
    else insert(ch[x][1],x,val);
    pushup(x);
    if(!balance(x))last=x;
}
void remove(int x,point val) //删除节点 默认p[x].num--
{
    if(!x)return;
    if(p[x]==val)
    {
        p[x].num-=val.num;
        pushup(x);
        return;
    }
    curd=cmpd[x];
    if(cmp(val,p[x]))remove(ls,val);
    else remove(rs,val);
    pushup(x);
    if(!balance(x))last=x;
```

```
}
int tail; point t[MAXN];
void dfs(int x)
{
    if(!x)return;
    if(p[x].num)t[++tail]=p[x];
    rec[++h]=x;
    dfs(ls);
    dfs(rs);
}
void rebuild() //每次 insert 或 remove 后调用一下
{
    if(!last)return;
    tail=0,dfs(last);
    int x=fa[last],w=(ch[x][1]==last),&p=x?ch[x][w]:root;
    p=build(1,tail,x,t);
    last=0;
}
```

高维询问/修改

复杂度 $O(\log_2 n)$ ~ $O(\sqrt{n})$

询问矩阵和

```
int X1,X2,Y1,Y2;
LL query(int x)
{
    if(!x)return 0;
    if(X1<=mn[x][0]&&X2>=mx[x][0]&&Y1<=mn[x][1]&&Y2>=mx[x][1])return sum[x];
    if(X2<mn[x][0]||X1>mx[x][0]||Y1>mx[x][1]||Y2<mn[x][1])return 0;
    return (p[x].d[0]>=X1&&p[x].d[0]<=X2&&p[x].d[1]>=Y1&&p[x].d[1]<=Y2?p[x].v:0)+
        query(ls)+query(rs);
}
```

区间修改(YY)：类似线段树/平衡树，维护懒标记

重构时注意 $pushdown$

字符串

单模式串 -> KMP

多模式串 -> AC自动机

单文本串 -> SA/SAM

多文本串 -> 广义SAM

字符串哈希

模板

1.自然溢出

```
const ULL BASE = 13331;
ULL p[MAXN];
void init()
{
    p[0]=1;
    for(int i=1;i<MAXN;i++)
        p[i]=p[i-1]*BASE;
}

struct myHash
{
    ULL h[MAXN];
    hash(const char str[]) {init(str);}
    void init(const char str[])
    {
        int n=strlen(str);
        h[n]=0;
        for(int i=n-1;i>=0;i--) //倒着hash i位为最低位 n为最高位 方便获取子串hash值
            h[i]=h[i+1]*BASE+str[i]-'A'+1;
    }
    ULL get_hash(int i,int length) //从下标i开始长度为length的字串的hash值
    {
        return h[i]-h[i+length]*p[length];
    }
};
```

2.二维矩阵hash

```
//二维hash 取两个base
//统计b矩阵在a矩阵中的出现次数

const ULL base1 = 13331;
const ULL base2 = 23333;
ULL p1[MAXN],p2[MAXN];
void pre()
{
    p1[0]=p2[0]=1;
    for(int i=1;i<MAXN;i++)
        p1[i]=p1[i-1]*base1,
        p2[i]=p2[i-1]*base2;
}
int T,n1,m1,n2,m2;
char a[MAXN][MAXN],b[MAXN][MAXN];
```

```
ULL h[MAXN][MAXN];
void init()
{
    for(int i=1;i<=n1;i++)
        for(int j=1;j<=m1;j++)
            h[i][j]=h[i][j-1]*base2+h[i-1][j]*base1
                -h[i-1][j-1]*base1*base2+(a[i][j]-'A'+1);
}
ULL get_hash(int x,int y,int lx,int ly)
{
    return  +h[x+lx-1][y+ly-1]
            -h[x+lx-1][y-1]*p2[ly]
            -h[x-1][y+ly-1]*p1[lx]
            +h[x-1][y-1]*p1[lx]*p2[ly];
}

int work()
{
    pre();
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n1,&m1);
        for(int i=1;i<=n1;i++) scanf("%s",a[i]+1);
        scanf("%d%d",&n2,&m2);
        for(int i=1;i<=n2;i++) scanf("%s",b[i]+1);

        init();

        ULL t=0;
        for(int i=1;i<=n2;i++)
            for(int j=1;j<=m2;j++)
                t+=(b[i][j]-'A'+1)*p1[n2-i]*p2[m2-j];//注意下标

        int ans=0;
        for(int i=1;i<=n1-n2+1;i++)
            for(int j=1;j<=m1-m2+1;j++)
                if(get_hash(i,j,n2,m2)==t)
                    ans++;
        printf("%d\n",ans);
    }
    return 0;
}
```

3.双hash

```
//备用 模数&base
//const ULL MOD1 = 1610612741;
//const ULL MOD2 = 1222827239;

//const ULL BASE1 = 1331;
```

```
//const ULL BASE2 = 131;

const ULL MOD1 = 402653189;
const ULL MOD2 = 805306457;

const ULL BASE1 = 13331;
const ULL BASE2 = 23333;

ULL p[MAXN][2];
void init()
{
    p[0][0]=p[0][1]=1;
    for(int i=1;i<MAXN;i++) p[i][0]=p[i-1][0]*BASE1%MOD1;
    for(int i=1;i<MAXN;i++) p[i][1]=p[i-1][1]*BASE2%MOD2;
}
struct myHash
{
    ULL h[MAXN][2];
    void init(char s[],int n=0)
    {
        n=strlen(s),h[n][0]=h[n][1]=0;
        for(int i=n-1;i>=0;i--) h[i][0]=(h[i+1][0]*BASE1%MOD1+s[i]-'A'+1)%MOD1;
        for(int i=n-1;i>=0;i--) h[i][1]=(h[i+1][1]*BASE2%MOD2+s[i]-'A'+1)%MOD2;
    }
    upr get_hash(int pos,int length)
    {
        return {
            ((h[pos][0]-h[pos+length][0]*p[length][0]%MOD1)+MOD1)%MOD1,
            ((h[pos][1]-h[pos+length][1]*p[length][1]%MOD2)+MOD2)%MOD2
        };
    }
};
upr make_hash(char s[])
{
    int n=strlen(s);
    upr ans;
    for(int i=n-1;i>=0;i--)
    {
        ans.fi=(ans.fi*BASE1%MOD1+s[i]-'A'+1)%MOD1;
        ans.se=(ans.se*BASE2%MOD2+s[i]-'A'+1)%MOD2;
    }
    return ans;
}
```

序列自动机

维护 $nx[i][c]$ 表示在 i 之后的最近的字符 c 的下标

若将 i 看作是点，$nx[i][c]$ 看作是转移边，那么原字符串的每一个子序列都对唯一对应着 DAG 上的一条路径

值域过大时可以考虑使用主席树来存边: 【模板】子序列自动机

也可以使用 vector 存下所有元素出现的位置，贪心匹配

长度为 k 的本质不同子序列个数

DAG 上 dp

```
int n,k;
int nx[MAXN][26],t[26];
LL f[MAXN][MAXN],ans;
char s[MAXN];
int work()
{
    scanf("%d%d",&n,&k);
    scanf("%s",s+1);
    if(k==0)return printf("1\n");
    fp(i,0,25)t[i]=n+1;
    fd(i,n,0)
    {
        memcpy(nx[i],t,sizeof(t));
        if(i)t[s[i]-'a']=i;
    }
    f[0][0]=1;
    fp(i,0,n)fp(j,0,i)if(f[i][j])
        fp(c,0,25)if(nx[i][c]<=n)
            f[nx[i][c]][j+1]+=f[i][j],
            f[nx[i][c]][j+1]%=MOD;
    fp(i,1,n)ans+=f[i][k],ans%=MOD;
    return printf("%lld\n",ans);
}
```

树状数组维护字符串哈希

支持单点修改，查询子串哈希值。

```
const ULL BASE = 4238197ll;
ULL p[N+5];
void init(){p[0]=1;fp(i,1,N-1)p[i]=p[i-1]*BASE;}
namespace BIT
{
    ULL h[N];
    inline int lowbit(int x){return x&-x;}
    inline void update(int x,ULL v)
    {
        for(register int i=x;i<N;i+=i&-i)
            h[i]+=p[i-x]*v;
    }
    inline ULL sum(int x)
    {
        ULL ans = 0;
        for(register int i=x;i;i-=i&-i)
            ans+=p[x-i]*h[i];
        return ans;
```

```
    }
    inline ULL query(int l,int r)
    {
        if(r<l) return 0;
        return sum(r)-sum(l-1)*p[r-l+1];
    }
}
```

KMP

资料

前缀函数与 KMP 算法 OI Wiki

模板

```
namespace kmp
{
    string a;
    int n,nx[MAXN];
    void get_nx(string &str)
    {
        a=str,n=str.length(),nx[0]=0;
        for(int i=1;i<n;i++)
        {
            int j=nx[i-1];
            while(j&&a[i]!=a[j]) j=nx[j-1];
            if(a[i]==a[j]) j++;
            nx[i]=j;
        }
    }
    vector<int> match(string &b,vector<int> pos={})
    {
        int m=b.length();
        for(int i=0,j=0;i<m;i++)
        {
            while(j&&b[i]!=a[j]) j=nx[j-1];
            if(b[i]==a[j]) j++;
            if(j==n) pos.pb(i-n+1),j=nx[j-1];
        }
        return pos;
    }
};
```

应用

1.求字符串的最小循环节

设字符串长度为 N 下标从0开始，设 $L=N-next[N-1]$，若 $N\ \%\ L=0$则存在最小循环节且长度为 L，否则最少需要补充 $N-N\ \%\ L$ 个字符串使其存在循环节。

若求一个字符串的所有循环节则需要不断跳 $nx[j-1]$ 直到 j 变为 0， 因为最小循环节不一定能构造出所有可行的循环节，如字符串 $ababaaaabab$ 中 7 和 9 都是一个可行的循环节。

2.求字符串 S 每一个前缀在字符串 T 中的出现次数

将字符串 S 和 T 用不出现在两个字符串内的字符(如'#')拼接起来，对于 $i>=n+1$ 的 $nx[i]$ 用以下代码计算。

```
vector<int> ans(n + 1);
for (int i = 0; i < n; i++) ans[nx[i]]++;
for (int i = n - 1; i > 0; i--) ans[nx[i - 1]] += ans[i];
for (int i = 0; i <= n; i++) ans[i]++;
```

扩展kmp

扩展kmp可在线性时间复杂度内求解字符串 S 的每一个后缀与字符串 T 的最长公共前缀。

$next[i]$ 表示 $T[1...i-1]$ 与 $T[i...m-1]$ 的最长公共前缀

$ext[i]$ 表示 S 的后缀 $S[i...n-1]$ 与字符串 T 的最长公共前缀

资料

扩展KMP算法

模板

```
namespace exkmp //字符串下标从0开始
{
    string T;
    int m,nx[MAXN],ext[MAXN];
    void get_nx(string &str)
    {
        int a=0,p=0;
        m=str.length(),nx[0]=m,T=str;
        for(int i=1;i<m;i++)
        {
            if(i>=p||i+nx[i-a]>=p)
            {
                if(i>=p) p=i;
                while(p<m&&T[p]==T[p-i]) p++;
                nx[i]=p-i,a=i;
            }
            else nx[i]=nx[i-a];
        }
    }
```

```
    void get_ext(string &S)
    {
        int a=0,p=0,n=S.length();
        for(int i=0;i<n;i++)
        {
            if(i>=p||i+nx[i-a]>=p)
            {
                if(i>=p) p=i;
                while(p<n&&p-i<m&&S[p]==T[p-i]) p++;
                ext[i]=p-i,a=i;
            }
            else ext[i]=nx[i-a];
        }

    }
};
```

字符串最小表示法

循环同构

当字符串 S 中可以选定一个位置 i 满足

$$S[i...n]+S[1...i-1]=T$$

则称字符串 S 与 T 循环同构。

最小表示

与字符串 S 循环同构的字符串中字典序最小(大)的字符串，判断两个字符串是否循环同构时可以求出它们最小表示并比较。

资料

最小表示法 OI Wiki

模板

```
int get_min_rep(string &s)
{
    int n=s.length();
    int i=0,j=1,k=0;
    while(i<n&&j<n&&k<n)
    {
        int l=(i+k)%n,r=(j+k)%n;
        if(s[l]==s[r]) k++;
        else
        {
            s[l]>s[r]?(i+=k+1):(j+=k+1);
            if(i==j) j++;
            k=0;
```

```
        }
    }
    return min(i,j);
}

int get_max_rep(string &s)
{
    int n=s.length();
    int i=0,j=1,k=0;
    while(i<n&&j<n&&k<n)
    {
        int l=(i+k)%n,r=(j+k)%n;
        if(s[l]==s[r]) k++;
        else
        {
            s[l]<s[r]?(i+=k+1):(j+=k+1);
            if(i==j) j++;
            k=0;
        }
    }
    return min(i,j);
}
```

manacher

$d1[i]$ 代表以 $S[i]$ 为中心的最长奇回文的长度，$d2[i]$ 代表以 $S[i]$ ~ $S[i+1]$ 中间的空位开始的最长偶回文的长度。

注意 $d1$ $d2$ 均代表的是回文串的长度，而非回文串的*半径*。

```
namespace manacher
{
    int p[MAXN*2];
    string a;
    void run(string &s,int d1[],int d2[])
    {
        a.clear(),a.pb('$'),a.pb('#');
        for(int i=0;i<s.size();i++)
            a.pb(s[i]),a.pb('#');
        a.pb('\0');
        int n=a.size()-1,id=0,mx=0;
        for(int i=1;i<=n;i++)
        {
            if(mx>i) p[i]=min(p[2*id-i],mx-i-1);
            else p[i]=1;
            while(a[i+p[i]]==a[i-p[i]]) p[i]++;
            if(i+p[i]-1>mx) mx=i+p[i]-1,id=i;
        }
        for(int i=0;i<s.size();i++)   d1[i]=p[(i+1)*2]-1;
        for(int i=0;i<s.size()-1;i++) d2[i]=p[(i+1)*2+1]-1;
```

```
    }
}
```

AC自动机

一个状态的 $fail$ 连向的是这个状态在自动机($Trie$)上的最长真后缀。

1.求以文本串每个位置匹配的模式串个数

定义 $f(i)$ 为文本串以 $S[i]$ 为最后一个字符所能匹配到的模式串的个数，以下是求 $f(i)$ 的模板。

```
namespace ACM
{
    const int SZ=26;
    int root,cnt;
    int nx[N][SZ],fail[N],s[N];
    //s表示该节点能匹配多少个模式串

    int newnode()
    {
        cnt++;
        for(int i=0;i<SZ;i++)
            nx[cnt][i]=0;
        fail[cnt]=s[cnt]=0;
        return cnt;
    }
    void init(){cnt=0,root=newnode();}
    void insert(string &s)
    {
        int n=s.length(),now=root;
        for(int i=0;i<n;i++)
        {
            if(!nx[now][s[i]-'a'])
                nx[now][s[i]-'a']=newnode();
            now=nx[now][s[i]-'a'];
        }
        s[now]++;
    }
    void build()
    {
        queue<int> q;
        fail[root]=root;
        for(int i=0;i<SZ;i++)
        {
            if(!nx[root][i]) nx[root][i]=root;
            else fail[nx[root][i]]=root,q.push(nx[root][i]);
        }
        while(!q.empty())
        {
            int now=q.front(); q.pop();
            for(int i=0;i<SZ;i++)
```

```
            {
                if(nx[now][i])
                {
                    fail[nx[now][i]]=nx[fail[now]][i];
                    s[nx[now][i]]+=s[fail[nx[now][i]]];
                    //继承fail的s(结尾)
                    q.push(nx[now][i]);
                }
                else nx[now][i]=nx[fail[now]][i];
            }
        }
    }
    //f[i]代表以i为结尾能匹配的模式串的个数
    void match(string &s,LL f[])
    {
        int now=root,n=s.length(),t;
        for(int i=0;i<n;i++)
        {
            now=nx[now][s[i]-'a'];
            f[i]=s[t];
        }
    }
};
```

2.求每个模式串的出现次数

在自动机上记录每个状态能被文本串到达的次数，匹配完后让每个节点和他的 $fail$ 节点连边建成 $fail$ 树，统计作为模式串结尾的节点的子树中有多少次匹配。

```
const int K=26;
namespace ACM
{
    int root,cnt;
    int nx[N][K],fail[N],add[N];
    vector<int> id[N];
    void newnode(int &x)
    {
        x=++cnt;
        for(int i=0;i<K;i++)
            nx[cnt][i]=0;
        fail[cnt]=add[cnt]=0,id[cnt].clear();
    }
    void init(){cnt=0,newnode(root);}
    void insert(string &s,int index)
    {
        int n=s.length(),now=root;
        for(int i=0;i<n;i++)
        {
            int c=s[i]-'a';
            if(!nx[now][c])
                newnode(nx[now][c]);
```

```
            now=nx[now][c];
        }
        id[now].pb(index);
    }
    void build()
    {
        queue<int> q;
        fail[root]=root;
        for(int i=0;i<K;i++)
        {
            if(!nx[root][i]) nx[root][i]=root;
            else fail[nx[root][i]]=root,q.push(nx[root][i]);
        }
        while(!q.empty())
        {
            int now=q.front();q.pop();
            for(int i=0;i<K;i++)
            {
                if(nx[now][i])
                    fail[nx[now][i]]=nx[fail[now]][i],q.push(nx[now][i]);
                else nx[now][i]=nx[fail[now]][i];
            }
        }
    }
    void match(string &s)
    {
        int now=root,n=s.length(),t;
        for(int i=0;i<n;i++)
        {
            now=nx[now][s[i]-'a'];
            add[now]++;
        }
    }
    void debug(int x=root,char c='#')
    {
        cout << dbgs2(x,c) << endl;
        for(int i=0;i<K;i++)
            if(nx[x][i])
                debug(nx[x][i],i+'a');
    }
};

int n;
struct edge{int u,v,next;};
vector<edge> e={edge()};
int pre[N];
void addedge(int u,int v){e.pb({u,v,pre[u]}),pre[u]=e.size()-1;}
void dfs(int u,int ans[])
{
    go(u) dfs(v,ans),ACM::add[u]+=ACM::add[v];
    for(int i:ACM::id[u])
        ans[i]+=ACM::add[u];
}
void solve(int ans[])
```

```
{
    for(int i=2;i<=ACM::cnt;i++)
        addedge(ACM::fail[i],i);
    dfs(ACM::root,ans);
}

int ans[MAXN];
string s;

int work()
{
    cin >> n,ACM::init();
    fp(i,1,n) cin >> s,ACM::insert(s,i);
    cin>>s,ACM::build(),ACM::match(s),solve(ans);
    fp(i,1,n)cout << ans[i] << endl;
    return 0;
}
```

后缀数组

下标从 1 开始

用数字当字符时注意不要用 0

数组含义

$sa[i]$ 排名为 i 的后缀的起始位置.

$rk[i]$ 以i为起点的后缀的排名。

$height[i]$ 排名为 i 的后缀和排名为 $i-1$ 的后缀的 LCP。

性质

$$LCP(sa_i,sa_j)=minheight_{i+1},\ldots,height_j$$

即排名为i的后缀和排名为j的后缀的LCP为 $height_{i+1},\ldots,height_j$ 中的最小值

模板

```
namespace SA
{
    int n,m;
    int sa[MAXN],c[MAXN],rk[MAXN],y[MAXN],t[MAXN];
    inline bool cmp(int i,int j,int k)
    {
        static int a,b;
        a = i+k>n ? -1 : rk[i+k];
        b = j+k>n ? -1 : rk[j+k];
        return (a==b)&&(rk[i]==rk[j]);
    }
```

```
void init(char s[])
{
    n=strlen(s+1),m=256;
    for(int i=1;i<=n;i++) rk[i]=s[i];

    for(int i=0;i<=m;i++) c[i]=0;
    for(int i=1;i<=n;i++) c[rk[i]]++;
    for(int i=1;i<=m;i++) c[i]+=c[i-1];
    for(int i=n;i;i--) sa[c[rk[i]]--]=i;
    for(int k=1,p;k<n;k=k<<1)
    {
        p=0;
        //y[]用来暂存第二关键字排序的结果
        for(int i=n;i>n-k;i--) y[++p]=i;
        for(int i=1;i<=n;i++) if(sa[i]>k) y[++p]=sa[i]-k;

        //按第一关键字排序
        for(int i=0;i<=m;i++) c[i]=0;
        for(int i=1;i<=n;i++) c[rk[i]]++;
        for(int i=1;i<=m;i++) c[i]+=c[i-1];

        //按y的逆序提取计数排序的结果
        //以保证第一关键字相同时第二关键字较小的在后面
        for(int i=1;i<=n;i++) t[i]=rk[y[i]];



        for(int i=n;i;i--) sa[c[t[i]]--]=y[i];

        //构建以本次排序为基准的新rk[]数组
        p=y[sa[1]]=1;
        for(int i=2;i<=n;i++)
            y[sa[i]]=cmp(sa[i],sa[i-1],k) ? p : ++p;
        swap(rk,y),m=p;

        if(p==n) break;
    }
}
int height[MAXN];
//height[i]代表后缀rank[i]和后缀rank[i-1]的LCP lcp(sa[i],sa[i-1])
void get_height(char s[])
{
    int k=0;
    for(int i=1;i<=n;i++)
    {
        if(k) k--;
        int j=sa[rk[i]-1];
        while(s[i+k]==s[j+k]) k++;
        height[rk[i]]=k;
    }
}
void debug(char s[])
{
    fp(i,1,n)
```

```
        {
            cout << dbgs4(i,sa[i],rk[i],h[i]) << endl;
            fp(j,sa[i],n)putchar(s[j]);putchar('\n');
        }
    }
}
```

统计字符串中相同的子串的对数 或 字符串中每一对后缀的LCP之和

若用 T_i 表示字符串从i开始的后缀，相同子串对数等价于求

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1}lcp(T_i,T_j)$$

即字符串中任意两个后缀的 lcp 之和

考虑后缀 T_{sa_i} 和排名小于它的后缀的 lcp 之和 S_{sa_i}

$$S_{sa_i}=\sum_{j=1}^{i}minheight_j...height_i$$

用单调栈维护以下每个 $height_j$ 作为最小值的范围和它的贡献即可 $O(n)$ 来求 $\sum_{i=1}^{n}S_{sa_i}$

模板

配合上面的SA板子食用

```
struct myStack
{
    stack<lpr> s1;
    stack<LL > s2;
    LL sum;
    void push(lpr p)
    {
        while(p<s1.top())
        {
            sum-=s2.top();
            s1.pop(),s2.pop();
        }
        s2.push(p.fi*(p.se-s1.top().se));
        s1.push(p);
        sum+=s2.top();
    }
    void init()
    {
        while(!s1.empty())s1.pop();
        while(!s2.empty())s2.pop();
        s1.push({0,0}),s2.push(0),sum=0;
    }
}stk;
```

```
LL solve()
{
    LL ans=0;
    stk.init();
    fp(i,1,n) stk.push({SA::height[i],i}),ans+=stk.sum;
    return ans;
}
```

字符串本质不同的子串个数

$$ans=\sum_{i=1}^{n}n-sa[i]+1-height[i]$$

后缀自动机

资料

1. 陈立杰冬令营SAM讲稿
2. 后缀自动机（SAM）学习笔记 ouuan
3. OI wiki SAM
4. 后缀自动机学习笔记 Menci

笔记

后缀自动机上每个点代表的是一些 $endpos$ 相同且长度连续的一些字符串，设状态 u 表示的字符串的长度范围为 $[min(u),max(v)]$

每个结点的 par 是所有满足 $endpos[u]\in endpos[v]$ 的 v 中且 $endpos[v]$ 大小最小的 v。

有关 par 的一些性质：

1. $min(u)=max(par[u])+1$
2. $par[u]$ 所表示的字符串都是 u 的后缀。

自动机接受的字符串为 S 的所有子串。

表示字符串前缀 Pre 的状态 u 的 $max(u)=|Pre|$，从 u 跳 par 的话可以遍历 Pre 的所有后缀。

CF200C

模板

```
namespace SAM
{
    int root,cnt,last;
    int nx[N][26],par[N],len[N],sum[N];
    int buc[N],seq[N];
    int newnode()
    {
```

```
        cnt++;
        mst(nx[cnt],0);
        par[cnt]=len[cnt]=sum[cnt]=0;
        return cnt;
    }
    void init(){cnt=0,last=root=newnode();}
    void insert(int c)
    {
        int np=newnode(),p=last;
        len[np]=len[last]+1;
        while(p&&!nx[p][c])nx[p][c]=np,p=par[p];
        if(p)
        {
            int q=nx[p][c];
            if(len[q]==len[p]+1)par[np]=q;
            else
            {
                int nq=newnode();
                len[nq]=len[p]+1;
                par[nq]=par[q];
                memcpy(nx[nq],nx[q],sizeof(nx[q]));
                while(p&&nx[p][c]==q)nx[p][c]=nq,p=par[p];
                par[q]=par[np]=nq;
            }
        }
        else par[np]=root;
        last=np;
    }
    void sort() //seq 倒序为 叶子 -> 根
    {
        mst(buc,0);
        fp(i,1,cnt)buc[len[i]]++;
        fp(i,1,N-1)buc[i]+=buc[i-1];
        fp(i,1,cnt)seq[buc[len[i]]--]=i;
    }
    void solve()
    {
        fp(i,1,n)insert(s[i]-'a');
        int cur=root;
        fp(i,1,n)cur=nx[cur][s[i]-'a'],sum[cur]++;
        sort();
        fd(i,cnt,1)cur=seq[i],sum[par[cur]]+=sum[cur];
    }
}
```

技巧 & 应用

1.求 $endpos$ 集合的大小/最大值/最小值

考虑 par 树的所有节点，$endpos$ 中 含有 i 的深度最大的节点一定是表示前缀 Pre_i 的状态，标记这个节点后用 dfs/拓扑/基排 逆推即可。

如果要求每个点 $endpos$ 的具体值可能需要 线段树合并/平衡树启发式合并。

```
int seq[MAXN*2],buc[N];
void sort_by_len()
{
    mst(buc,0);
    fp(i,1,cnt) buc[len[i]]++;
    fp(i,1,N-1) buc[i]+=buc[i-1];
    fp(i,1,cnt) seq[buc[len[i]]--]=i;
}
void solve()
{
    fd(i,cnt,1)
    {
        int cur=seq[i];
        f[par[cur]]+=f[cur];
        //f[par[cur]]=min(f[par[cur]],f[cur]);
        //f[par[cur]]=max(f[par[cur]],f[cur]);
    }
}
```

2. $logn$ 查找某个子串在 SAM 上对应的状态

考虑倍增，预处理出每个节点在 par 树上的2^j 祖先，从表示 $S[1...r]$ 的节点往上跳找到深度最小且满足 $max(u) >= (r-l+1)$ 的 u 即可。

```
void init()
{
    for(int i=1;i<=cnt;i++) nx[i][0]=par[i];
    fp(k,1,20)fp(i,1,cnt)nx[i][k]=nx[nx[i][k-1]][k-1];
}
LL count(int l,int r)
{
    int cur=suf[r];
    fd(k,20,0)
        if(nx[cur][k]&&len[nx[cur][k]]>=r-l+1)
            cur=nx[cur][k];
    return sum[cur];
}
```

3. 两个串的LCS

对 S 串建立 SAM，拿 T 串到 SAM 上“运行”

具体来讲就是考虑 T 的每个前缀 pre_i 有多长的后缀是 S 的一个子串，从 pre_i 转移到 pre_{i+1} 时看下当前状态是否存在到 $T[i+1]$ 的转移边，若不存在就不断跳 par (删掉前面的一部分) 直到找到存在转移边的状态，同时维护在每个状态下的匹配长度，最后取个 max 就是答案。

匹配过程十分类似于 kmp 和 AC自动机 啊，，，

```
void solve(char s[])
{
    int L=0,ans=0,cur=root,n=strlen(s+1);
    for(int i=1;i<=n;i++)
    {
        int c=s[i]-'a';
        while(cur!=root&&!nx[cur][c])
            cur=par[cur],L=len[cur];
        if(nx[cur][c]) cur=nx[cur][c],L++;
        ans=max(ans,L);
    }
    printf("%d\n",ans);
}
```

4.本质不同的子串个数

相同的子串一定被同一个状态所表示，所以对后缀自动机上每个状态所表示的子串长度求和即是答案。

$$ans=\sum_{i=1}^{cnt}len[i]-len[par[i]]$$

5.最小表示法

先将字符串 S 复制一倍后得到 T，可以发现 T 的长度为 n 的子串中涵盖了 S 循环移位所能得到的所有可能，所以问题转化为求 T 的长度为 n 的子串中字典序最小的一个。

对 T 建立 SAM 后从 $root$ 开始贪心走转移边中字符最小的那条即可。

6.字典序k大子串

统计出在每个状态通过转移边能到达的状态的数量 $f[u]$ ，从根节点开始从小到大枚举转移边 v 若 $f[v]>=k$ 则从 u 转移到 v，否则将 k 减去 $f[v]$ 后继续尝试字符更大的转移边。

```
void print(int cur,int k,int t,LL f[])
{
    if(!k) return;
    for(auto trans:nx[cur])
    {
        if(f[trans.se]>=k)
        {
            putchar(trans.fi),print(trans.se,k-(t?sum[trans.se]:1),t,f);
            break;
        }
        else k-=f[trans.se];
    }
}
void solve(int k,int t)
```

```
{
    fp(i,1,cnt) buc[len[i]]++;
    fp(i,1,N-1) buc[i]+=buc[i-1];
    fp(i,1,cnt) seq[buc[len[i]]--]=i;
    fd(i,cnt,1) sum[par[seq[i]]]+=sum[seq[i]];

    fd(i,cnt,1)
    {
        int cur=seq[i];
        f[0][cur]=1,f[1][cur]=sum[cur];
        for(auto trans:nx[cur])
        {
            f[0][cur]+=f[0][trans.se];
            f[1][cur]+=f[1][trans.se];
        }
    }

    if(f[t][root]-(t==1?sum[root]:1)<k) printf("-1\n");
    else print(root,k,t,f[t]),putchar('\n');
}
```

7.多个串的LCS

对其中一个串建立 SAM 把剩下所有串放到 SAM 上运行，类似于求两个串的 LCS 的过程，运行每一个串的时候记录它在当前状态上的最大匹配长度，注意不同串在同一个状态上的匹配长度要取 min,最后统计答案即可。

```
void run(string &s,int id)
{
    mst(f,0);
    int cur=root,L=0;
    fp(i,0,s.size()-1)
    {
        int c=s[i]-'a';
        while(cur!=root&&!nx[cur][c])
            cur=par[cur],L=len[cur];
        if(nx[cur][c]) cur=nx[cur][c],L++;
        f[cur]=max(f[cur],L);
    }
    fd(i,cnt,1)
    {
        int cur=seq[i];
        if(f[cur]) f[par[cur]]=len[par[cur]];
    }
    if(id==2) fp(i,1,cnt) ans[i]=f[i];
    else fp(i,1,cnt) ans[i]=min(ans[i],f[i]);
}
void solve()
{
    int L=0;
    fp(i,1,cnt) L=max(ans[i],L);
```

```
    cout << L << endl;
}
```

8.询问某一子串的后缀自动机

考虑用线段树合并维护 $right$ 集合，询问一个节点是否在子串 $[L,R]$ 构成的后缀自动机上可以通过查询节点的 $right$ 集合是否在区间 $[L+len,R]$ 中有值，其中 len 为当前匹配长度。

[NOI2018]你的名字

```
namespace seg
{
    int cnt,s[N],ls[N],rs[N];
    int query(int x,int l,int r,int nl,int nr)
    {
        if(!x||nl>nr||nr<l||nl>r)return 0;
        if(l==nl&&r==nr)return s[x];
        if(nr<=mid)return query(ls[x],l,mid,nl,nr);
        if(nl> mid)return query(rs[x],mid+1,r,nl,nr);
        return query(ls[x],l,mid,nl,mid)+query(rs[x],mid+1,r,mid+1,nr);
    }
    void insert(int &x,int l,int r,int pos)
    {
        if(!x)x=++cnt; s[x]++; if(l==r)return;
        if(pos<=mid)insert(ls[x],l,mid,pos);
        else insert(rs[x],mid+1,r,pos);
    }
    int merge(int x,int y,int l,int r)
    {
        if(!x||!y)return x+y; int cur=++cnt;
        if(l==r)s[cur]=s[x]+s[y];
        else
        {
            ls[cur]=merge(ls[x],ls[y],l,mid);
            rs[cur]=merge(rs[x],rs[y],mid+1,r);
            s[cur]=s[ls[cur]]+s[rs[cur]];
        }
        return cur;
    }
}
template<int N>
struct SAM
{
    int root,cnt,last;
    int nx[N][26],par[N],len[N];
    int rt[N],mx[N],buc[N],seq[N];
    int newnode()
    {
        cnt++;
        mst(nx[cnt],0);
        par[cnt]=len[cnt]=rt[cnt]=mx[cnt]=0;
```

```
        return cnt;
    }
    void init(){cnt=0,last=root=newnode();}
    void insert(int c)
    {
        int np=newnode(),p=last;
        len[np]=len[last]+1;
        while(p&&!nx[p][c])nx[p][c]=np,p=par[p];
        if(p)
        {
            int q=nx[p][c];
            if(len[q]==len[p]+1)par[np]=q;
            else
            {
                int nq=newnode();
                len[nq]=len[p]+1;
                par[nq]=par[q];
                memcpy(nx[nq],nx[q],sizeof(nx[q]));
                while(p&&nx[p][c]==q)nx[p][c]=nq,p=par[p];
                par[q]=par[np]=nq;
            }
        }
        else par[np]=root;
        last=np;
    }
    void sort()
    {
        int mx=0;
        fp(i,1,cnt)buc[len[i]]++,mx=max(mx,len[i]);
        fp(i,1,mx)buc[i]+=buc[i-1];
        fp(i,1,cnt)seq[buc[len[i]]--]=i;
        fp(i,0,mx)buc[i]=0;
    }
    void build(char s[])
    {
        init();
        int n=strlen(s+1);
        fp(i,1,n)insert(s[i]-'a');
    }
    void build_mx(char s[])
    {
        int n=strlen(s+1),cur=root;
        fp(i,1,n)cur=nx[cur][s[i]-'a'],mx[cur]=i;
        sort();
        fd(i,cnt,1)
        {
            int x=seq[i];
            if(par[x])mx[par[x]]=max(mx[par[x]],mx[x]);
        }
    }
    void build_right(char s[])
    {
        int n=strlen(s+1),cur=root;
        fp(i,1,n)cur=nx[cur][s[i]-'a'],seg::insert(rt[cur],1,n,i);
```

```
        sort();
        fd(i,cnt,1)
        {
            int x=seq[i];
            if(par[x])rt[par[x]]=seg::merge(rt[par[x]],rt[x],1,n);
        }
    }
    void move(int &l,int &cur,int n,int L,int R,int c)
    {
        while(1)
        {
            if(nx[cur][c]&&seg::query(rt[nx[cur][c]],1,n,L+l,R)>0)
                {cur=nx[cur][c],l++;return;}
            if(!l)return;
            if(--l==len[par[cur]])cur=par[cur];
        }
    }
    LL calc(int p[],LL ans=0)
    {
        fp(i,2,cnt)
            ans+=max(0,len[i]-max(len[par[i]],p[mx[i]]));
        return ans;
    }
};
int n,m,L,R;
char s[MAXN],t[MAXN];
int  p[MAXN];
SAM<1000005> S;
SAM<2000005> T;
int work()
{
    scanf("%s",s+1),n=strlen(s+1),S.build(s),S.build_right(s);
    MC
    {
        scanf("%s %d %d",t+1,&L,&R),m=strlen(t+1),T.build(t),T.build_mx(t);
        int cur=S.root,l=0;
        fp(i,1,m)S.move(l,cur,n,L,R,t[i]-'a'),p[i]=l;
        printf("%lld\n", T.calc(p));
    }
    return 0;
}
```

广义SAM

将 SAM 推广到字典树上的产物，目前不太理解，，，

定义 $Trie$ 树的一个子串为节点 u 到它子树中的节点 v 的路径所形成的字符串，那么广义 SAM 接受的状态为 $Tire$ 树上的所有子串。

把 $endpos$ 视为 $Trie$ 树上的节点，par 的定义不变。

模板

离线构造，需要预先知道 $Tire$ 树的形态，从根节点开始 bfs，建立新节点时拿他父亲的状态作为 $last$ 执行普通 SAM 的 $extend$ 即可。

[ZJOI2015]诸神眷顾的幻想乡 求多个串本质不同的子串个数

```
const int K = 10;

namespace SAM
{
    int root,cnt;
    int len[MAXN*2],par[MAXN*2];
    int nx[MAXN*2][K+1];
    void newnode(int &x){x=++cnt;}
    void init(){cnt=0,newnode(root);}
    int extend(int c,int last)
    {
        int cur;
        newnode(cur);
        len[cur]=len[last]+1;
        int p=last;
        while(p&&!nx[p][c])
            nx[p][c]=cur,p=par[p];
        if(!p) par[cur]=root;
        else
        {
            int q=nx[p][c];
            if(len[q]==len[p]+1) par[cur]=q;
            else
            {
                int n;
                newnode(n);
                len[n]=len[p]+1;
                par[n]=par[q];
                memcpy(nx[n],nx[q],sizeof(nx[q]));
                while(p&&nx[p][c]==q)
                    nx[p][c]=n,p=par[p];
                par[q]=par[cur]=n;
            }
        }
        return cur;
    }
    void solve()
    {
        LL ans=0;
        fp(i,1,cnt) ans+=len[i]-len[par[i]];
        io.write(ans);
        io.push('\n');
    }
}

namespace Trie
{
```

```
    int root,cur,cnt;
    int nx[MAXN][K+1],fa[MAXN],last[MAXN];
    void newnode(int &x){x=++cnt;}
    void init(){cnt=0,newnode(root),cur=root;}
    void insert(int c)
    {
        if(!nx[cur][c])
            newnode(nx[cur][c]),fa[nx[cur][c]]=cur;
        cur=nx[cur][c];
    }
    void back(){cur=fa[cur];}
    void build_sam()
    {
        SAM::init();
        queue<int> q;
        q.push(root);
        last[root]=SAM::root;
        while(!q.empty())
        {
            int cur=q.front(); q.pop();
            fp(i,0,K)if(nx[cur][i])
                last[nx[cur][i]]=SAM::extend(i,last[cur]),
                    q.push(nx[cur][i]);
        }
    }
};

struct edge{int u,v,next;};
vector<edge> e;
int n,c;
int pre[MAXN],col[MAXN],in[MAXN];
void addedge(int u,int v){e.pb({u,v,pre[u]}),pre[u]=e.size()-1;}
void dfs(int u,int fa)
{
    Trie::insert(col[u]);
    go(u) if(v!=fa) dfs(v,u);
    Trie::back();
}

int work()
{
    io.read(n),io.read(c),e.pb(edge());
    fp(i,1,n) io.read(col[i]);
    fp(i,1,n-1)
    {
        static int u,v;
        io.read(u),io.read(v);
        addedge(u,v),addedge(v,u);
        in[u]++,in[v]++;
    }
    Trie::init();
    fp(i,1,n) if(in[i]==1) dfs(i,-1);
    Trie::build_sam();
    SAM::solve();
```

```
    return 0;
}
```

维护字符串在分组中的出现情况

set + 启发式合并 o(nlognlogn)

可以用线段树合并去1log

多个串的LCS

```
const int K = 26;
int n,ans;
char s[MAXN];

namespace SAM
{
    int root,cnt;
    int len[MAXN*2],par[MAXN*2],nx[MAXN*2][K+1];
    vector<int> G[MAXN*2];
    set<int> s[MAXN*2];
    void newnode(int &x)
    {
        x=++cnt;
        G[x].clear();
        s[x].clear();
        len[x]=par[x]=0;
        mst(nx[x],0);
    }
    void init(){cnt=0,newnode(root);}
    int extend(int c,int last)
    {
        int cur;
        newnode(cur);
        len[cur]=len[last]+1;
        int p=last;
        while(p&&!nx[p][c])
            nx[p][c]=cur,p=par[p];
        if(!p) par[cur]=root;
        else
        {
            int q=nx[p][c];
            if(len[q]==len[p]+1) par[cur]=q;
            else
            {
                int n;
                newnode(n);
                len[n]=len[p]+1;
                par[n]=par[q];
                memcpy(nx[n],nx[q],sizeof(nx[q]));
                while(p&&nx[p][c]==q)
                    nx[p][c]=n,p=par[p];
```

```
                par[q]=par[cur]=n;
            }
        }
        return cur;
    }
    void dfs(int u)
    {
        int son=u;
        for(auto v:G[u])
        {
            dfs(v);
            if(s[v].size()>s[son].size())
                son=v;
        }
        s[u].swap(s[son]);
        for(auto v:G[u])
            for(auto id:s[v])
                s[u].insert(id);
        if(s[u].size()==n)
            ans=max(ans,len[u]);
    }
    void solve()
    {
        fp(i,1,cnt)
            G[par[i]].pb(i);
        dfs(root);
    }
}

namespace Trie
{
    int root,cnt;
    int nx[MAXN][K+1],fa[MAXN],last[MAXN];
    set<int> s[MAXN];
    void newnode(int &x)
    {
        x=++cnt;
        s[x].clear();
        fa[x]=last[x]=0;
        mst(nx[x],0);
    }
    void init(){cnt=0,newnode(root);}
    void insert(int id,char str[])
    {
        int cur=root,n=strlen(str+1);
        fp(i,1,n)
        {
            int c=str[i]-'a';
            if(!nx[cur][c])newnode(nx[cur][c]);
            cur=nx[cur][c],s[cur].insert(id);
        }
    }
    void dfs(int u,int cur)
    {
```

```
        for(auto id:s[u])
            SAM::s[cur].insert(id);
        fp(i,0,K)if(nx[u][i])
            dfs(nx[u][i],SAM::nx[cur][i]);
    }
    void solve()
    {
        SAM::init();
        queue<int> q;
        q.push(root);
        last[root]=SAM::root;
        while(!q.empty())
        {
            int cur=q.front(); q.pop();
            fp(i,0,K)if(nx[cur][i])
                last[nx[cur][i]]=SAM::extend(i,last[cur]),
                    q.push(nx[cur][i]);
        }
        dfs(root,SAM::root);
        SAM::solve();
    }
};

int work()
{
    MC
    {
        scanf("%d",&n),ans=0;
        Trie::init();
        fp(i,1,n)scanf("%s",s+1),Trie::insert(i,s);
        Trie::solve();
        printf("%d\n",ans);
    }
    return 0;
}
```

回文自动机(回文树)

$root1$ 奇根 $root2$ 偶根

```
const int K = 26;

namespace PAM
{
    int cnt,root1,root2,last=0;
    int nx[MAXN*2][K],len[MAXN*2],fail[MAXN*2],sum[MAXN*2];
    string str;
    void newnode(int &x)
    {
        x=++cnt;
        fp(i,0,K-1) nx[x][i]=0;
```

```
        len[x]=fail[x]=0;
    }
    void init()
    {
        cnt=0,str.clear(),str.pb('$');
        newnode(root1),len[root1]=-1;
        newnode(root2),len[root2]= 0;
        fail[root2]=root1,last=root1;
    }
    void extend(int pos)
    {
        int c=str[pos]-'a',p=last,np;
        while(str[pos-len[p]-1]!=str[pos]) p=fail[p];
        if(nx[p][c]) np=nx[p][c];
        else
        {
            newnode(np);
            nx[p][c]=np;
            len[np]=len[p]+2;
            if(p==root1) fail[np]=root2;
            else
            {
                p=fail[p];
                while(str[pos-len[p]-1]!=str[pos]) p=fail[p];
                fail[np]=nx[p][c];
            }
        }
        last=np,sum[last]++;
    }
    void build(char s[])
    {
        int n=strlen(s+1);
        fp(i,1,n) str.pb(s[i]),extend(i);
    }
    void solve()
    {
        LL ans=0;
        fd(i,cnt,1) sum[fail[i]]+=sum[i];
        fp(i,1,cnt) ans=max(ans,(LL)len[i]*sum[i]);
        io.write(ans),io.push('\n');
    }
    void debug()
    {
        fp(x,1,cnt)
        {
            cout << dbgs3(x,len[x],fail[x]) << endl;
            fp(i,0,K-1) if(nx[x][i]) cout << dbgs2(char(i+'a'),nx[x][i]) << endl;
        }
        cout << endl;
    }
};
```

Shift-And

可以处理一些带限制的子串匹配问题，$O(n*m/64)$

hdu5716

```
int n,m,len;
char s[MAXN],t[65];
bitset<N> f,b[65];
inline int H(char c)
{
    if(c>='0'&&c<='9')return c-'0'+1;
    if(c>='a'&&c<='z')return c-'a'+11;
    if(c>='A'&&c<='Z')return c-'A'+37;
    return 0;
}
int work()
{
    while(gets(s+1))
    {
        m=strlen(s+1),scanf("%d",&n),mst(b,0);
        //fp(i,1,62)b[i].reset();
        fp(i,1,n)
        {
            scanf("%d %s",&len,t+1);
            fp(j,1,len)b[H(t[j])][i]=1;
        }
        f.reset(); bool flag=1;
        fp(i,1,m)
        {
            int k=H(s[i]);
            if(k)
            {
                f[0]=1;
                f<<=1;
                f&=b[k];
                if(f[n])printf("%d\n",i-n+1),flag=0;
            }
            else f.reset();
        }
        if(flag)puts("NULL"); getchar();
    }
    return 0;
}
```

数学 & 数论

二次剩余

模意义下开根号

```
LL n,p,w;
struct num {LL x, y;};
num mul(num a, num b, LL p)
{
    num ans = {0, 0};
    ans.x = ((a.x * b.x % p + a.y * b.y % p * w % p) % p + p) % p;
    ans.y = ((a.x * b.y % p + a.y * b.x % p) % p + p) % p;
    return ans;
}
LL binpow_real(LL a, LL b, LL p)
{
    LL ans = 1;
    while (b)
    {
        if (b & 1) ans = ans * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return ans % p;
}
LL binpow_imag(num a, LL b, LL p)
{
    num ans = {1, 0};
    while (b)
    {
        if (b & 1) ans = mul(ans, a, p);
        a = mul(a, a, p);
        b >>= 1;
    }
    return ans.x % p;
}
LL cipolla(LL n, LL p)
{
    n %= p;
    if (p == 2) return n;
    if (binpow_real(n, (p - 1) / 2, p) == p - 1) return -1;
    LL a;
    while (1)
    {
        a = rand() % p;
        w = ((a * a % p - n) % p + p) % p;
        if (binpow_real(w, (p - 1) / 2, p) == p - 1) break;
    }
    num x = {a, 1};
    return binpow_imag(x, (p + 1) / 2, p);
}

int work()
{
    srand(time(0));
    MC
    {
```

```
        scanf("%lld%lld",&n,&p);
        if(!n) printf("0\n");
        else
        {
            LL ans1=cipolla(n,p),ans2=p-ans1;
            if(ans1==-1)printf("Hola!\n");
            else
            {
                if(ans1==ans2)printf("%lld\n",ans1);
                else printf("%lld %lld\n",min(ans1,ans2),max(ans1,ans2));
            }

        }
    }
    return 0;
}
```

斐波那契通项公式

$$a_n = \frac{1}{\sqrt{5}}[(\frac{\sqrt{5}+1}{2})^n - (\frac{\sqrt{5}-1}{2})^n]$$

在模 $1e9+9$ 意义下有:

$$\frac{1}{\sqrt{5}} = 383008016$$

$$\frac{\sqrt{5}+1}{2} = 691504013$$

$$\frac{\sqrt{5}-1}{2} = 308495997$$

可以用来快速求 fib k 次方之和

```
const LL a = 691504013, b = 308495997, t = 383008016, MOD = 1e9+9;

LL n; int k;
LL f[MAXN],fi[MAXN],inv[MAXN];
LL pa[MAXN],pb[MAXN];
LL fpow(LL x,LL k)
{
    LL ans=1; x%=MOD;
    while(k)
    {
        if(k&1)ans=ans*x%MOD;
        x=x*x%MOD,k=k>>1;
    }
    return ans;
}
LL get_inv(LL x){return fpow(x,MOD-2);}
LL C(int n,int m){return f[n]*fi[n-m]%MOD*fi[m]%MOD;}
```

```
LL A(int n,int m){return f[n]*fi[n-m]%MOD;}
void init(int n)
{
    f[0]=fi[0]=1;fp(i,1,n)f[i]=f[i-1]*i%MOD;
    fi[n]=get_inv(f[n]);
    fd(i,n-1,1)fi[i]=fi[i+1]*(i+1)%MOD;
    fp(i,1,n)inv[i]=fi[i]*f[i-1]%MOD;
}
LL sum(LL a1, LL q, LL n)
{
    if(q==1)return n*a1%MOD;
    return a1*(1-fpow(q,n))%MOD*get_inv(1-q)%MOD;
}
int work()
{
    init(1e5);
    pa[0]=1;fp(i,1,1e5)pa[i]=pa[i-1]*a%MOD;
    pb[0]=1;fp(i,1,1e5)pb[i]=pb[i-1]*b%MOD;
    MC
    {
        scanf("%lld%d",&n,&k);
        LL ans=0;
        fp(j,0,k)
            ans+=C(k,j)*sum(pa[k-j]*pb[j]%MOD,pa[k-j]*pb[j]%MOD,n)%MOD*((j&1)?
-1:1),ans%=MOD;
        ans*=fpow(get_inv(t),k),ans%=MOD,ans+=MOD,ans%=MOD;
        printf("%lld\n",ans);
    }
    return 0;
}
```

等比求和

$$q=1,S_n=n*a_1$$

$$q\neq 1,S_n=\frac{a_1(1-q^n)}{1-q}=\frac{a_1-a_nq}{1-q}$$

等差求和

$$S_n=na_1+\frac{n(n-1)}{2}d=\frac{n(a_1+a_n)}{2}$$

GCD & EXGCD

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

```
bool solve(int a,int b,int c,int &x,int &y)
{
    int d=gcd(a,b);
    if(c%d) return 0;//无解
    a/=d,b/=d,c/=d;
    exgcd(a,b,x,y,d);
    x*=c,y*=c;
    //x+=b y-=a
    return 1;
}
```

1.解方程 $ax+by=c$

设 $g=gcd(a,b)$，首先方程有解的前提是 $c\ \%\ g=0$

令 $a'=a/g$，$b'=b/g$，$c'=c/g$，则方程 $ax+by=c$ 的解与方程 $a'x+b'y=c'$ 的解一一对应，同时满足 $gcd(a',b')=1$（即 a', b' 互质）

利用 $exgcd$ 我们可以求得方程$a'x+b'y=gcd(a',b')=1$ 的一组特解 x_0,y_0，那么**原方程的通解**可以表示为：

$c'x_{0}+kb',c'y_{0}-ka',k\in{Z}$

快速幂

```
LL fpow(LL x,LL k)
{
    LL ans=1;
    while(k)
    {
        if(k&1) ans=ans*x%MOD;
        x=x*x%MOD,k=k>>1;
    }
    return ans;
}
```

因子求和

若X分解质因数的结果是

$$p_1^{c_1}p_2^{c_2}\dots p_n^{c_n}$$

那么X所有因子的和就是

$$(1+p_1^1+\dots+p_1^{c_1})(1+p_2^1+\dots+p_2^{c_2})\dots(1+p_n^1+p_n^2+\dots+p_n^{c_n})$$

等比数列求和即可，注意下 $1-p$ 和模数互质的情况，这时

$$(1+p^1+p^2+\dots+p^n)\equiv(n+1)$$

```
//求a^b的所有因子之和
LL inv(LL x){return fpow(x,MOD-2);}
LL a,b;
vector<lpr> p;
int work()
{
    a=read(),b=read();
    if(a==1) return printf("1\n");
    if(a==0) return printf("0\n");
    if(b==0) return printf("1\n");
    for(int i=2;(LL)i<=a;i++)
    {
        LL cnt=0;
        while(a%i==0) a/=i,cnt++;
        if(cnt) p.pb({i,cnt*b});
        if(a==1) break;
    }
    LL ans=1;
    for(auto x:p)
    {
        LL t=x.fi,cnt=x.se;
        if((1-t)%MOD==0) ans=ans*(cnt+1)%MOD;
        else ans=(ans*(1-fpow(t,cnt+1))%MOD*inv(1-t)%MOD+MOD)%MOD;
    }
    write(ans),putchar('\n');
    return 0;
}
```

线性筛素数

```
vector<int> prime;
bool flag[N];
void init_prime(int n)
{
    flag[1]=1;
    for(int i=2;i<=n;i++)
    {
        if(!flag[i]) prime.pb(i);
        for(auto p:prime)
        {
            if((LL)p*i>n) break;
            flag[p*i]=1;
            if(i%p==0) break;
        }
    }
}
```

(待补)阶乘分解质因数

$O(n\log n)$

组合数递推

$C(n,m)=C(n-1,m-1)+C(n-1,m)$

组合数预处理

O(n+nlogn)

```
const LL MOD = 998244353;
LL f[MAXN],ff[MAXN];
LL fpow(LL x,LL k)
{
    LL ans=1; x%=MOD;
    while(k)
    {
        if(k&1)ans=ans*x%MOD;
        x=x*x%MOD,k=k>>1;
    }
    return ans;
}
inline LL inv(LL x){return fpow(x,MOD-2);}
inline LL A(LL n,LL m){return f[n]*ff[n-m]%MOD;}
inline LL C(LL n,LL m){return A(n,m)*ff[m]%MOD;}
inline void init(){f[0]=ff[0]=1;fp(i,1,MAXN-1)f[i]=f[i-1]*i%MOD,ff[i]=ff[i-
1]*inv(i)%MOD;}
```

FFT

当所求的值可以写作

$$h(j)=\sum_{i=0}^{j}f(i)*g(j-i)$$

或

$$h(j)=\sum_{i=j+1}^{n}f(i)*g(i-j)$$

的卷积形式时，可以考虑使用 FFT ，第二种类型的多项式需要翻转并扩展上下界来变成第一类的形式。

资料

1. 快速傅里叶变换(FFT)详解
2. FFT 学习笔记
3. FFT模板以及简单的FFT入门题总结

模板

常规

```
typedef cp complex<double>;
namespace fft
{
    cp o[4*MAXN],oi[4*MAXN];
    void init(int n)
    {
        for(int i=0;i<n;i++)
        {
            o[i] =cp(cos(2*PI/n*i),sin(2*PI/n*i));
            oi[i]=conj(o[i]);
        }
    }
    void transform(cp a[],const int n,cp o[])
    {
        //二进制翻转
        int k=0;
        while((1<<k)<n)k++;
        for(int i=0;i<n;i++)
        {
            int t=0;
            for(int j=0;j<k;j++)
                if(i&(1<<j))
                    t|=(1<<(k-j-1));
            if(i<t) swap(a[i],a[t]);
        }
        for(int l=2;l<=n;l*=2)
        {
            int m=l/2;
            //将两个长度为 m 的序列的答案合并为长度为 l 的序列的答案
            for(cp *p=a;p!=a+n;p+=l)
            {
                for(int i=0;i<m;i++)
                {
                    cp t=o[n/l*i]*p[m+i];
                    p[m+i]=p[i]-t,p[i]+=t;
                }
            }
        }
    }
    void dft (cp a[],int n){transform(a,n,o);}
    void idft(cp a[],int n){transform(a,n,oi);for(int i=0;i<n;i++) a[i]/=n;}
};

//n1 n2为多项式a1 a2系数的个数 不是最高次数(注意+1)

void mul(int a1[],int n1,int a2[],int n2,int res[])
{
    int n=1;
    while(n<n1+n2) n=n<<1;
    static cp c1[4*MAXN],c2[4*MAXN];
```

```
    for(int i=0;i< n;i++) c1[i]=c2[i]=0;
    for(int i=0;i<n1;i++) c1[i].real(a1[i]);
    for(int i=0;i<n2;i++) c2[i].real(a2[i]);
    fft::init(n);
    fft::dft(c1,n),fft::dft(c2,n);
    for(int i=0;i<n;i++) c1[i]*=c2[i];
    fft::idft(c1,n);
    for(int i=0;i<n1+n2-1;i++)
        res[i]=int(floor(c1[i].real()+0.5));
}

void mul(double a1[],int n1,double a2[],int n2,double res[])
{
    int n=1;
    while(n<n1+n2) n=n<<1;
    static cp c1[4*MAXN],c2[4*MAXN];
    for(int i=0;i< n;i++) c1[i]=c2[i]=0;
    for(int i=0;i<n1;i++) c1[i].real(a1[i]);
    for(int i=0;i<n2;i++) c2[i].real(a2[i]);
    fft::init(n);
    fft::dft(c1,n),fft::dft(c2,n);
    for(int i=0;i<n;i++) c1[i]*=c2[i];
    fft::idft(c1,n);
    for(int i=0;i<n1+n2-1;i++)
        res[i]=c1[i].real();
}
```

卡常

```
struct cp
{
    double r,i;
    cp(double _r = 0,double _i = 0){r = _r; i = _i;}
    cp operator +(const cp  &b){return cp(r+b.r,i+b.i);}
    cp operator -(const cp  &b){return cp(r-b.r,i-b.i);}
    cp operator *(const cp  &b){return cp(r*b.r-i*b.i,r*b.i+i*b.r);}
    cp operator /(const int &n){return cp(r/n,i/n);}
};

namespace fft
{
    int R[MAXN*4];
    void init(int n)
    {
        int L=0;
        while((1<<L)<n) L++;
        for(int i=0;i<n;i++)
            R[i]=(R[i>>1]>>1)|((i&1)<<(L-1));
    }
    void transform(cp a[],const int n,int f)
    {
```

```
        for(int i=0;i<n;i++)
            if(i<R[i])
                swap(a[i],a[R[i]]);
        for(int l=2;l<=n;l=l<<1)
        {
            int m=l/2;
            cp wn=cp(cos(2*PI/l),f*sin(2*PI/l));
            for(cp *p=a;p!=a+n;p+=l)
            {
                cp w=cp(1,0);
                for(int i=0;i<m;i++,w=w*wn)
                {
                    cp t=w*p[m+i];
                    p[m+i]=p[i]-t,p[i]=p[i]+t;
                }
            }
        }
    }
    void dft (cp a[],int n){transform(a,n, 1);}
    void idft(cp a[],int n){transform(a,n,-1);for(int i=0;i<n;i++) a[i]=a[i]/n;}
};

cp c1[MAXN*4],c2[MAXN*4];
void mul(LL a1[],int n1,LL a2[],int n2,LL res[])
{
    int n=1;
    while(n<n1+n2) n=n<<1;
    for(int i=0;i<=n;i++) c1[i]=c2[i]=cp(0,0);
    for(int i=0;i<n1;i++) c1[i]=cp(a1[i],0);
    for(int i=0;i<n2;i++) c2[i]=cp(a2[i],0);
    fft::init(n);
    fft::dft(c1,n),fft::dft(c2,n);
    for(int i=0;i<n;i++) c1[i]=c1[i]*c2[i];
    fft::idft(c1,n);
    for(int i=0;i<n1+n2-1;i++)
        res[i]=LL(floor(c1[i].r+0.5));
}
```

高斯消元

普通

```
bool guass(int **a,int n)//高斯消元
{
    int i=1,to;double t;//now为当前处理的行数
    for(int i=1;i<=n;i++) //消去并回代Xi
    {
        for(to=i;to<=n;to++)
            if(fabs(a[to][i])>eps) break;//找到Xi系数非0的一行
        if(to>n) continue;//Xi系数全为0
        if(to!=i)for(int j=1;j<=n+1;j++)
```

```
            swap(a[to][j],a[i][j]);//交换
        t=a[i][i];
        for(int j=1;j<=n+1;j++) a[i][j]/=t;//将Xi行所有的系数变为1
        for(int j=1;j<=n;j++)//j为1到n 回代与消去同时进行
        {
            if(j==i) continue;
            t=a[j][i];//倍数
            for(int k=1;k<=n+1;k++)
                a[j][k]-=t*a[i][k];
        }
    }
    for(int i=1;i<=n;i++)
        if(fabs(a[i][n+1])>eps) return 0;//出现 0=常数 的情况无解
    return 1;//有解
}
```

模意义

```
void guass(LL a[][sz], int n, int MOD) //模意义下的高斯消元
{
    LL t;
    int i = 1, to;
    for (int i = 1;i <= n;i++)
    {
        for (to = i;to <= n;to++)
            if (a[to][i]) break;

        if (to > n) continue;
        if (to != i)
            for (int j = 1;j <= n + 1;j++)
                swap(a[to][j], a[i][j]);

        t = get_inv(a[i][i], MOD);
        for (int j = 1;j <= n + 1;j++)
            a[i][j] *= t, a[i][j] %= MOD;

        for (int j = 1;j <= n;j++)
        {
            if (j == i) continue;
            t = a[j][i];
            for (int k = 1;k <= n + 1;k++)
                a[j][k] = ((a[j][k] - t * a[i][k] % MOD)
                    % MOD + MOD) % MOD;
        }
    }
}
```

Lucas

```
int C(int n,int m,int p)//C(n,m) mod p p为素数
{
    if(n<m) return 0;//注意当n<m时 C(a, b) = 0
    if(m>n-m) m=n-m;
    int s1=1,s2=1;
    for(int i=1;i<=m;i++) s1=s1*(n-i+1)%p;
    for(int i=1;i<=m;i++) s2=s2*i%p;
    return s1*qpow(s2,p-2,p)%p;//逆元
}
int lucas(int n,int m,int p)//lucas定理
{
    if(!n||!m) return 1;
    return lucas(n/p,m/p,p)*C(n%p,m%p,p)%p;
}
```

欧拉降幂

```
LL euler(LL a, string b, LL m) // 扩展欧拉定理 a^b % m = a^(b%phi(m)+phi(m)) % m b为高精度
{
    bool flag = 0;
    LL t = 0, phi_m = get_phi(m);
    for (string::iterator p = b.begin();p != b.end();p++)
    {
        t = t * 10 + (*p) - '0';
        if (t >= phi_m) // 当b<phi(m)时直接用快速幂
            t %= phi_m, flag = 1;
    }
    return flag ? fast_pow(a, t % phi_m + phi_m, m) : fast_pow(a, t, m);
}
```

(待补)中国剩余定理

线性求逆元

```
int n;
LL MOD,f[MAXN],fi[MAXN],inv[MAXN];
void init()
{
    f[0]=1;fp(i,1,n)f[i]=f[i-1]*i%MOD;
    fi[n]=fpow(f[n],MOD-2);
    fd(i,n-1,1)fi[i]=fi[i+1]*(i+1)%MOD;
    fp(i,1,n)inv[i]=fi[i]*f[i-1]%MOD;
}
```

数论分块(整除分块)

$\lfloor \frac{n}{i} \rfloor$ 的取值最多 $sqrt(n)$ 种

```
for(int l=1,r=1;l<=n;l=r+1,r=min(n,n/(n/l)))
{
    ...
    if(r==n)break;
}
```

O(1)组合数套装

预处理 $O(n)$

```
#undef fi
int n,m;
const LL MOD = 998244353;
LL f[MAXN],fi[MAXN],inv[MAXN];
LL fpow(LL x,LL k)
{
    LL ans=1; x%=MOD;
    while(k)
    {
        if(k&1)ans=ans*x%MOD;
        x=x*x%MOD,k=k>>1;
    }
    return ans;
}
LL get_inv(LL x){return fpow(x,MOD-2);}
LL C(int n,int m){return f[n]*fi[n-m]%MOD*fi[m]%MOD;}
LL A(int n,int m){return f[n]*fi[n-m]%MOD;}
void init()
{
    f[0]=fi[0]=1;fp(i,1,n)f[i]=f[i-1]*i%MOD;
    fi[n]=get_inv(f[n]);
    fd(i,n-1,1)fi[i]=fi[i+1]*(i+1)%MOD;
    fp(i,1,n)inv[i]=fi[i]*f[i-1]%MOD;
}
```

线性基

资料

1. 线性基学习笔记 Menci
2. 线性基学习笔记 ouuan

性质

1. 线性基内的元素线性无关
2. 原集合的任意一个元素都可唯一的表示为线性基中若干个元素异或起来的结果

3. 线性基合并只需要将一个线性基中的所有元素插入到另一个线性基中即可。

模板

1.线性基求最大异或和

```
namespace LinearBasis
{
    const int mn = 60;
    LL a[MAXN];
    LL maxXor()//最大异或和
    {
        LL ans=0;
        for(int i=mn;i>=0;i--)
            if((ans^a[i])>ans)
                ans^=a[i];
        return ans;
    }
    void insert(LL x)//插入一个数
    {
        for(int i=mn;i>=0;i--)
        {
            if((1LL<<i)&x)
            {
                if(a[i]) x^=a[i];
                else{a[i]=x;break;}
            }
        }
    }
    bool check(LL x)//判断x是否可由线性基构造出来
    {
        for(int i=mn;i>=0;i--)
            if(((1LL<<i)&x) && a[i])
                x^=a[i];
        return x==0;
    }
};
```

2.前缀线性基

前缀线性基可以解决用给定区间内的元素所能异或出的最大和问题，本质上是贪心的选择��置更靠右的元素作为基底。

```
int a[MAXN][31];//线性基
int pos[MAXN][31];//线性基中的元素在原序列中的位置
void insert(int x,int index)
{
    for(int i=0;i<=mx;i++)
    {
```

```
            a[index][i]  =a[index-1][i];
            pos[index][i]=pos[index-1][i];
        }
        int curp=index;
        for(int i=mx;i>=0;i--)
        {
            if((1<<i)&x)
            {
                if(!a[index][i])//该位上没有元素 直接插入
                {
                    a[index][i]=x;
                    pos[index][i]=curp;
                    break;
                }
                if(curp>pos[index][i])//当前元素作为基时位置更靠右
                {
                    swap(x,a[index][i]);
                    swap(curp,pos[index][i]);
                }
                x^=a[index][i];
            }
        }
    }
    int query(int l,int r)
    {
        int ans=0;
        for(int i=mx;i>=0;i--)
            if(a[r][i]&&pos[r][i]>=l)
                if((ans^a[r][i])>ans)
                    ans^=a[r][i];
        return ans;
    }
```

dp

树上背包上下界优化 O(nm)

参考

树上背包的上下界优化 ouuan

模板

Nowcoder 蓝魔法师

```
const LL MOD = 998244353;
EDGE(MAXN,MAXN*2);
int n,k;
LL  f[MAXN][MAXN],s[MAXN],g[MAXN];
int sz[MAXN];
```

```
void dfs(int u,int fa)
{
    f[u][1]=sz[u]=1;
    go(u)if(v!=fa)
    {
        dfs(v,u),mst(g,0);
        fp(i,1,min(k,sz[u]))
        {
            fp(j,1,min(k-i,sz[v]))
                g[i+j]+=f[u][i]*f[v][j]%MOD,g[i+j]%=MOD;
            g[i]+=f[u][i]*s[v]%MOD,g[i]%=MOD;
        }
        fp(i,1,k)f[u][i]=g[i];
        sz[u]+=sz[v];
    }
    fp(i,1,k)s[u]+=f[u][i],s[u]%=MOD;
}
int work()
{
    scanf("%d%d",&n,&k);
    fp(i,1,n-1)
    {
        int u,v;scanf("%d%d",&u,&v);
        addedge(u,v),addedge(v,u);
    }
    dfs(1,0);
    LL ans=0;
    fp(i,1,k)ans+=f[1][i],ans%=MOD;
    return printf("%lld\n",ans);
}
```

NlogN求LIS

模板

```

int n,ans1,ans2;
int f1[MAXN],f2[MAXN];
int s2[MAXN],s1[MAXN];
int a[MAXN];

int work()
{
    while(scanf("%d",&a[++n])!=EOF);n--;
    for(int i=1;i<=n;i++) s1[i]=s2[i]=inf;
    for(int i=1;i<=n;i++)
    {
        //s[i]代表长度为i的子序列最后一个元素的最小值
        int j =lower_bound(s1+1,s1+1+n,a[i])-s1;
        f1[i]=j,s1[j]=a[i];
        ans1=max(ans1,f1[i]);
    }
```

```
    for(int i=n;i;i--)
    {
        int j =upper_bound(s2+1,s2+1+n,a[i])-s2;
        f2[i]=j,s2[j]=a[i];
        ans2=max(ans2,f2[i]);
    }
    return printf("%d\n%d\n",ans2,ans1);
}
```

数位DP 记忆化搜索

模板

```
LL cnt,d[20],f[20][...];

//limit 是否有最高位限制
//lead  是否有前导零
//state 状态
//状态只与len之前的位有关 因此len之前的状态相同时可用记忆化来计算
//** 请用-1作为没算过的标记 **
LL dfs(int len,bool limit,bool lead,int state1 ...)
{
    if(len==0) return ...; //边界值
    if(!lead && !limit  && f[len][state1][...] != -1)  return f[len][state1][...];
    int maxn=limit?d[len]:9;
    LL ans=0;
    for(int i=0;i<=maxn;i++)
    {
        if(lead&&i==0)
            ans+=dfs(len-1,limit&&i==maxn,1,...);
        else
        {
            if(....)
                ans+=dfs(len-1,limit&&i==maxn,0,...);
        }
    }
    if(!lead && !limit) f[len][state1][...]=ans;
    //cout << dbgs(len) << dbgs4(limit,lead,num,s) << endl;
    return ans;
}

LL solve(LL x) //计算[0,x]内符合条件的数
{
    cnt=0;
    while(x) d[++cnt]=x%10,x=x/10;
    LL ans=0;
    for(...) ans+=dfs(cnt,1,1,...);
    return ans;
}

int main()
```

```
{
    mst(f,-1);
    ...
}
```

子集枚举(SOS-DP)

O(3^n)

```
for(int s=0;s<(1<<n);s++)
{
    for(int i=s;;i=(i-1)&s)
    {
        //code here...
        if(!i) break;
    }
}
```

O(n*2^n)

类似于:

$$S_i = \sum_{j \subset i} A_j$$

$$F_i = \min A_j, j \subset i$$

$$F_i = \max A_j, j \subset i$$

的式子都可以用高维前缀和优化

```
for(int i=1;i<=n;i++)
    for(int s=0;s<(1<<n);s++)
        if((1<<(i-1))&s)
            f[s]+=f[s^(1<<(i-1))];
```

决策单调性

可离线

分治

例: CF321E

```
void solve(int s,int l=1,int r=n,int nl=0,int nr=n-1)
{
```

```
    if(l>r) return;
    int m = mid,from; f[s][m]=inf;
    fp(i,nl,min(nr,m-1))
        if(f[s-1][i]+cost(i+1,m)<f[s][m])
            f[s][m]=f[s-1][i]+cost(i+1,m),from=i;
    solve(s,l,m-1,nl,from),solve(s,m+1,r,from,nr);
}
```

不可离线

写法参考自：浅析1D1D动态规划的优化

确定 $f[3]$ 之前所有状态的最优决策表：

111111111111222222222222

加入 $f[3]$ 后决策表只能有三种类型

111111111111222222222222 不变

111111111111222223333333 占据 2 的一部分

111111333333333333333333 完全覆盖 2

考虑使用栈来维护每个 i 作为决策点的起始位置

设老决策的起点为 j，从栈顶向下依次考虑：

1. 若新决策在 j 优于老决策，则退栈并抛弃老决策
2. 否则，转折点一定在当前这个老决策的区间中，二分这个位置
3. 新决策入栈

例： [NOI2009]诗人小G

```
pr s[MAXN]; int h=0;
LL calc(int i,int j){return f[j]+cost(i,j);}
fp(i,1,n)
{
    pr t=*(ub(s+1,s+h+1,mp(i,inf))-1);

    f[i]=calc(i,t.se),from[i]=t.se;

    if(i==n||calc(n,s[h].se)<calc(n,i))continue;

    while(h&&s[h].fi>i&&calc(s[h].fi,s[h].se)>calc(s[h].fi,i))h--;
    int l=max(i+1,s[h].fi),r=n,ans=n;
    while(l<=r)
    {
        if(calc(mid,s[h].se)>calc(mid,i))ans=mid,r=mid-1;
        else l=mid+1;
    }
```

```
        s[++h]={ans,i};
    }
```

斜率优化

$$f[i]=x(j)+k(i)*y(j)+b(i)$$

x , y 与 j 相关，k 与 i 相关，b 为常数/与 i 相关

x 单调 k 单调 -> 单调队列/单调栈 (看 k 递减/增)

x 单调 k 不单调 -> 单调队列上二分

x 不单调 -> 平衡树(请)

图论

树 hash

```
ULL seed = 13331;
int n;
int sz[MAXN];
ULL f[MAXN],p[MAXN];
vector<int> G[MAXN];
void dfs(int u,int fa)
{
    vector<int> ve; sz[u]=1;
    for(auto v:G[u])if(v!=fa)dfs(v,u),sz[u]+=sz[v],ve.pb(v);
    sort(all(ve),[](int u,int v){return f[u]<f[v];});
    for(int i=0;i<ve.size();i++)
    {
        int v=ve[i];
        f[u]+=f[v]*p[i];
    }
    f[u]*=sz[u];
    if(sz[u]==1)f[u]=1; // 叶子
    // cout << dbgs2(u,f[u]) << endl;
}
void init(){p[0]=1;fp(i,1,MAXN-1)p[i]=p[i-1]*seed;}
```

2-sat

n 个变量 $x[1]$ ~ $x[n]$，取值为 0 或 1，满足若干二元限制关系，判断并求出可行解。

$x[i]$ & $x[j]$ = 0/1

$x[i]$ | $x[j]$ = 0/1

$x[i]$ ^ $x[j]$ = 0/1

$x[i]$ = 1

$x[i]$ = 0

建图，每个变量 $x[i]$ 拆成两个点 i 和 $i+n$，代表 $x[i]=0$ 和 $x[i]=1$

有向边 $u->v$ 表示当 u 成立时，v 一定成立，那么一条二元限制一定可以分解成若干有向边

之后缩点，判断是否存在 i 和 $i+n$ 属于同一联通份量，若存在则无解

否则有解，$x[i]=(bel[i]>bel[i+n])$，bel 为缩点后的编号

luogu P4782

```
int n,m;
vector<int> G[MAXN];
int tim=0,num=0;
int low[MAXN],dfn[MAXN],bel[MAXN],siz[MAXN];
bool vis[MAXN]; //in stack
vector<int> s;
void tarjan(int u)
{
    dfn[u]=low[u]=++tim,vis[u]=1,s.pb(u);
    for(auto v:G[u])
    {
        if(!dfn[v]) tarjan(v),low[u]=min(low[u],low[v]);
        else if(vis[v]) low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u])
    {
        int cur; num++;
        do{
            cur=s.back(),s.ppb();
            vis[cur]=0,bel[cur]=num,siz[num]++;
        }while(cur!=u);
    }
}
void addedge(int u,int v){G[u].pb(v);}
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,m)
    {
        int x,y,a,b;
        scanf("%d%d%d%d",&x,&a,&y,&b);
        if(a==0&&b==0)      addedge(x+n,y),addedge(y+n,x);
        else if(a==0&&b==1) addedge(x+n,y+n),addedge(y,x);
        else if(a==1&&b==0) addedge(x,y),addedge(y+n,x+n);
        else if(a==1&&b==1) addedge(x,y+n),addedge(y,x+n);
    }
    fp(i,1,2*n)if(!dfn[i])tarjan(i);
    fp(i,1,n)if(bel[i]==bel[i+n])return puts("IMPOSSIBLE");
    puts("POSSIBLE");
```

```
    fp(i,1,n)printf("%d ",bel[i]>bel[i+n]);
    return 0;
}
```

POJ3678

```
int n,m;
vector<int> G[MAXN];
int tim=0,num=0;
int low[MAXN],dfn[MAXN],bel[MAXN],siz[MAXN];
bool vis[MAXN]; //in stack
vector<int> s;
void tarjan(int u)
{
    dfn[u]=low[u]=++tim,vis[u]=1,s.pb(u);
    fp(i,0,G[u].size()-1)
    // for(auto v:G[u])
    {
        int v=G[u][i];
        if(!dfn[v]) tarjan(v),low[u]=min(low[u],low[v]);
        else if(vis[v]) low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u])
    {
        int cur; num++;
        do{
            cur=s.back(),s.ppb();
            vis[cur]=0,bel[cur]=num,siz[num]++;
        }while(cur!=u);
    }
}
void addedge(int u,int v){G[u].pb(v);}
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,m)
    {
        int a,b,c; char op[5];
        scanf("%d %d %d %s",&a,&b,&c,op+1),a++,b++;
        if(op[1]=='A') //and
        {
            if(c) addedge(a,a+n),addedge(b,b+n);
            else  addedge(a+n,b),addedge(b+n,a);
        }
        if(op[1]=='O') //or
        {
            if(c) addedge(a,b+n),addedge(b,a+n);
            else  addedge(a+n,a),addedge(b+n,b);
        }
        if(op[1]=='X') //xor
        {
            if(c)
```

```
            {
                addedge(a,b+n),addedge(a+n,b);
                addedge(b,a+n),addedge(b+n,a);
            }
            else
            {
                addedge(a,b),addedge(a+n,b+n);
                addedge(b,a),addedge(b+n,a+n);
            }
        }
    }
    fp(i,1,2*n)if(!dfn[i])tarjan(i);
    bool flag=1;
    fp(i,1,n)flag&=(bel[i]!=bel[i+n]);
    puts(flag?"YES":"NO");
    return 0;
}
```

支配树

支配：以某个点为起点，当删除 u 时 v 变的不可达，则称点 u 支配了点 v

支配关系构成一种有根树，起点为根，树上每个点支配他子树的所有节点

$idom[u]$ 最近支配点，即每个点在支配树上的父亲

$sdom[u]$ 半支配点，即从 v 点为起点在 dfs 树上不走树边可以到达 u 的 v 中 dfn 最小的点

```
int n,m,root,tim;
int dfn[MAXN],sdom[MAXN],idom[MAXN],fa[MAXN],p[MAXN];
//数组下标为原图编号
//idom sdom 记录点的dfn序 其他记录原图编号
vector<int> G[MAXN],V[MAXN],c[MAXN],d[MAXN];
//G正向边 V反向边 c={x|sdom[x]==x} d支配树

void add(int u,int v){G[u].pb(v),V[v].pb(u);}
namespace dsu
{
    int f[MAXN],home[MAXN];
    // home[x] 记录x到根上最小的sdom所在的点
    int find(int x)
    {
        if(x==f[x])return x;
        int t=f[x];
        f[x]=find(f[x]);
        if(sdom[home[t]]<sdom[home[x]])
            home[x]=home[t];
        return f[x];
    }
    int get(int x){find(x);return home[x];}
}
void dfs(int u,int par)
```

```
{
    dfn[u]=++tim,p[tim]=u,fa[u]=par;
    for(auto v:G[u])
        if(v!=par&&!dfn[v])
            dfs(v,u);
}
int ans[MAXN];
void get_ans(int u,int par)
{
    ans[u]=1;
    for(auto v:d[u])if(v!=par)
        get_ans(v,u),ans[u]+=ans[v];
}
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,m)
    {
        int u,v;scanf("%d%d",&u,&v);
        add(u,v);
    }
    dfs(root=1,0);
    fd(i,tim,1)
    {
        int x=p[i];
        dsu::f[x]=dsu::home[x]=x;
        sdom[x]=i;
        idom[x]=0;
    }
    fd(i,tim,2)
    {
        int u=p[i];
        for(auto v:V[u])if(dfn[v])
            sdom[u]=min(sdom[u],sdom[dsu::get(v)]);
        c[p[sdom[u]]].pb(u);
        dsu::f[u]=fa[u];
        for(auto v:c[fa[u]])
        {
            int x=dsu::get(v);
            if(sdom[x]==dfn[fa[u]])idom[v]=dfn[fa[u]];
            else idom[v]=dfn[x];
        }
        c[fa[u]].clear();
    }
    fp(i,1,tim)
    {
        int x=p[i];
        if(sdom[x]==idom[x])idom[x]=p[idom[x]];
        else idom[x]=idom[p[idom[x]]];
        d[idom[x]].pb(x);
        // cout << dbgs2(idom[x],x) << endl;
    }
    get_ans(root,0);
    fp(i,1,n)printf("%d ",ans[i]);
```

```
    return 0;
}
```

虚树

模板

```
namespace virtual_tree //虚树
{
    int *inner_dfn;

    void dfs(int u, int fa, graph &G)
    {
        for (int i = G.pre[u]; i; i = G.e[i].next)
            if (G.e[i].to != fa)
                dfs(G.e[i].to, u, G);
        G.pre[u] = 0;
    }
    void clear_edge(int root, graph &G) { G.ecnt = 0, dfs(root, -1, G); }
    //别用memset 手动清边表保证复杂度

    typedef int func(int u, int v);
    bool cmp(int u, int v) { return inner_dfn[u] < inner_dfn[v]; }

    //建立虚树 返回dfn最小的节点 需预处理出dfs序(dfn[])与get_lca函数
    int build_virtual_tree(vector<int> &p, int dfn[], graph &G, func get_lca, int
s[])
    {
        inner_dfn = dfn;
        p.push_back(1); //默认加入1作为根节点
        sort(p.begin(), p.end());
        p.erase(unique(p.begin(), p.end()), p.end());
        sort(p.begin(), p.end(), cmp);

        //使用 栈s 保存树链
        int top = 0;
        s[++top] = p.front();

        for (int i = 1; i < p.size(); i++)
        {
            int &cur = p[i], lca = get_lca(cur, s[top]);
            if (lca == s[top]) s[++top] = cur; //当前节点可以加入到链的尾部
            else //栈中所表示的树链已经有一部分处理完了 退栈并连边
            {
                int t = top;
                while (dfn[s[t]] > dfn[lca]) t--;
                for (int j = t + 1; j < top; j++)
                    G.add_edge(s[j], s[j + 1], 0, 1);
                top = t, G.add_edge(lca, s[top + 1], 0, 1);
                if (s[top] != lca) s[++top] = lca;
                s[++top] = cur;
```

```
            }
        }
        for (int i = 1; i < top; i++)
            G.add_edge(s[i], s[i + 1], 0, 1);
        return s[1];
    }
}
```

Kruskal重构树

资料

Kruskal重构树入门 自为风月马前卒

模板

构建方法：初始时有 n 个叶子节点，运行 $kruskal$ 最小生成树算法，对于用边权为 w 连接两个节点 (u,v)，新建一个权值为 w 的节点 x，并连边 $(x,root(u))$ 和 $(x,root(v))$，并将 x 置为这颗树的根节点。

这样建出来的其实是一个大根堆，可以用来求从点 x 出发只走边权 $<=w$ 的边能到达的点的极大集合

做法就是从 x 倍增往上面跳直到点权 $>w$，设跳到的点为 u，那么 u 的所有叶子就是所求的极大集合

还有一个性质是：任意两个点路径上边权的最大值为它们的LCA的点权

例题

Peaks加强版

> 在Bytemountains有N座山峰，每座山峰有他的高度hi。
>
> 有些山峰之间有双向道路相连，共M条路径，每条路径有一个困难值，这个值越大表示越难走.
>
> 现在有Q组询问，每组询问询问从点v开始只经过困难值小于等于x的路径所能到达的山峰中第k高的山峰，如果无解输出-1

kruskal重构树 + 主席树

```
vector<int> X;
EDGE(MAXN,MAXN*2);
int n,m,q,cnt,last=-1;
int h[MAXN],a[MAXN],b[MAXN],fa[MAXN],st[MAXN],ed[MAXN],f[MAXN][21];
struct temp{int u,v,w;}edg[N];
int find(int x){return fa[x]==x?x:fa[x]=find(fa[x]);}
void dfs(int u,int fa)
{
    st[u]=++cnt,b[cnt]=h[u],f[u][0]=fa;
    go(u)if(v!=fa)dfs(v,u);
    ed[u]=cnt;
}
int root[MAXN],s[MAXN*10],ls[MAXN*10],rs[MAXN*10],cnt_;
void update(int pre,int &rt,int l,int r,int pos)
```

```
{
    if(!rt)rt=++cnt_; s[rt]=s[pre]+1; if(l==r)return;
    if(pos<=mid)update(ls[pre],ls[rt],l,mid,pos),rs[rt]=rs[pre];
    else update(rs[pre],rs[rt],mid+1,r,pos),ls[rt]=ls[pre];
}
void query(int pre,int rt,int l,int r,int k)
{
    if(l==r){printf("%d\n",X[l-1]),last=X[l-1];return;}
    int sum=s[rs[rt]]-s[rs[pre]];
    if(sum>=k)query(rs[pre],rs[rt],mid+1,r,k);
    else query(ls[pre],ls[rt],l,mid,k-sum);
}
bool cmp(const temp &a,const temp &b){return a.w<b.w;}
int work()
{
    scanf("%d%d%d",&n,&m,&q);
    fp(i,1,n)scanf("%d",&h[i]),X.pb(h[i]); sort(all(X)),unq(X);
    fp(i,1,n)h[i]=lb(all(X),h[i])-X.begin()+1;
    fp(i,1,2*n)fa[i]=i;
    fp(i,1,m)scanf("%d%d%d",&edg[i].u,&edg[i].v,&edg[i].w);
    sort(edg+1,edg+1+m,cmp);
    fp(i,1,m)if(find(edg[i].u)!=find(edg[i].v))
    {
        a[++n]=edg[i].w; int fu=find(edg[i].u),fv=find(edg[i].v);
        addedge(n,fu),addedge(n,fv),fa[fu]=fa[fv]=n;
    }
    fp(i,1,n)if(!st[i])dfs(find(i),0);
    fp(j,1,20)fp(i,1,n)f[i][j]=f[f[i][j-1]][j-1];
    fp(i,1,n)
        if(b[i])update(root[i-1],root[i],1,X.size(),b[i]);
        else root[i]=root[i-1];
    while(q--)
    {
        int u,x,k;scanf("%d%d%d",&u,&x,&k);
        if(last!=-1)u^=last,x^=last,k^=last;
        fd(i,20,0)if(f[u][i]&&a[f[u][i]]<=x)u=f[u][i];
        int l=st[u],r=ed[u];
        if(s[root[r]]-s[root[l-1]]<k)printf("-1\n"),last=-1;
        else query(root[l-1],root[r],1,X.size(),k);
    }
    return 0;
}
```

传递闭包

若有关系 a->b 和 b->c，则有关系 a->c

初始给定相邻的关系，求最终的关系矩阵

bitset 优化 floyd，$O(n^3/64)$

```
fp(k,1,n)fp(i,1,n)
    if(f[i][k])f[i]|=f[k];
```

最小环

无边权

bfs O(n^2+nm)

考虑枚举每个点为起点做一遍 bfs，当搜索树上出现平行边或前向边的时候更新答案

需特判自环

```
void bfs(int S)
{
    queue<int> q; mst(dis,0),dis[S]=inq[S]=1,q.push(S);
    int aa=0,bb=0;
    while(!q.empty())
    {
        int u=q.front(); q.pop(); inq[u]=0;
        go(u)
            if(!dis[v])dis[v]=dis[u]+1,q.push(v),inq[v]=1;
            else if(inq[v])ans=min(ans,dis[u]+dis[v]-1);
    }
}
int main()
{
    fp(i,1,n)bfs(i);
}
```

有边权

floyd n^3

```
int n,m,ans=inf;
int d[MAXN][MAXN],w[MAXN][MAXN];
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,n)fp(j,1,n)w[i][j]=inf;
    fp(i,1,n)w[i][i]=0;
    fp(i,1,m)
    {
        int u,v,x;
        scanf("%d%d%d",&u,&v,&x);
        w[u][v]=min(w[u][v],x);
        w[v][u]=min(w[v][u],x);
    }
```

```
    fp(i,1,n)fp(j,1,n)d[i][j]=w[i][j];
    fp(k,1,n)
    {
        fp(i,1,k-1)fp(j,1,k-1)if(i!=j)ans=min(ans,d[i][j]+w[i][k]+w[j][k]);
        fp(i,1,n)fp(j,1,n)d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
    }
    if(ans==inf)puts("No solution.");
    else printf("%d\n",ans);
    return 0;
}
```

KM

O(n^3)

```
int n,m;
LL w[MAXN][MAXN],lx[MAXN],ly[MAXN],slack[MAXN];
int lk[MAXN],pre[MAXN];
bool visy[MAXN];
void bfs(int k)
{
    int x=0,y=0,yy=0; LL delta;
    fp(i,1,n)slack[i]=linf;
    mst(pre,0),lk[y]=k;
    while(1)
    {
        x=lk[y],delta=linf,visy[y]=1;
        fp(i,1,n)if(!visy[i])
        {
            if(slack[i]>lx[x]+ly[i]-w[x][i])
                slack[i]=lx[x]+ly[i]-w[x][i],pre[i]=y;
            if(slack[i]<delta) delta=slack[i],yy=i;
        }
        fp(i,0,n)
            if(visy[i]) lx[lk[i]]-=delta,ly[i]+=delta;
            else slack[i]-=delta;
        y=yy;
        if(lk[y]==-1) break;
    }
    while(y) lk[y]=lk[pre[y]],y=pre[y];
}
void solve()
{
    mst(lx,0),mst(ly,0),mst(lk,-1);
    fp(i,1,n) mst(visy,0),bfs(i);
}
int work()
{
    while(scanf("%d",&n)!=EOF)
    {
        fp(i,1,n)fp(j,1,n)scanf("%lld",&w[i][j]);
```

```
        solve();
        LL ans=0;
        fp(i,1,n)ans+=lx[i]+ly[i];
        static int Case = 0;
        printf("%lld\n",ans);
    }
    return 0;
}
```

tarjan

1.无向图 割点

```
struct edge{int u,v,next;};
vector<edge> e;
vector<int> ans;

int n,m;
int fa[MAXN],dfn[MAXN],low[MAXN],tim;
int pre[MAXN];

void addedge(int u,int v){e.pb({u,v,pre[u]}),pre[u]=e.size()-1;}

void tarjan(int u)
{
    int child=0;
    bool flag=0;
    dfn[u]=low[u]=++tim;
    go(u)
    {
        if(!dfn[v])
        {
            fa[v]=u,tarjan(v);
            low[u]=min(low[u],low[v]);
            child++;
            if(fa[u]) flag|=(low[v]>=dfn[u]);
            else flag|=(child>=2);
        }
        else if(fa[u]!=v) low[u]=min(low[u],dfn[v]);
    }
    if(flag) ans.pb(u);
}

void solve()
{
    ans.clear();
    fp(i,1,n)if(!dfn[i])tarjan(i);
    sort(all(ans));
    printf("%d\n",ans.size());
    for(auto u:ans) printf("%d ",u);
}
```

```
int work()
{
    io.read(n),io.read(m);
    fp(i,1,m)
    {
        int u,v;
        io.read(u),io.read(v);
        addedge(u,v),addedge(v,u);
    }
    solve();
    return 0;
}
```

2.无向图 割边

```
struct edge
{
    int u,v;
    bool flag;
};
int n,m;
int fa[MAXN],dfn[MAXN],low[MAXN],tim;
vector<int>  G[MAXN],ans;
vector<edge> e;
map<pr,int> h; //判重边

void tarjan(int u)
{
    dfn[u]=low[u]=++tim;
    for(auto i:G[u])
    {
        int v=(e[i].u==u)?e[i].v:e[i].u;
        if(!dfn[v])
        {
            fa[v]=u;
            tarjan(v);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]&&!e[i].flag)
                ans.pb(i+1);
        }
        else if(fa[u]!=v) low[u]=min(low[u],dfn[v]);
    }
}

void init()
{
    e.clear(),h.clear(),tim=0;
    fp(i,1,n) G[i].clear();
    fp(i,1,n) dfn[i]=low[i]=fa[i]=0;
}
```

```
void solve()
{
    ans.clear();
    for(auto &t:e)if(h[{t.u,t.v}]>1)t.flag=1;
    fp(i,1,n)if(!dfn[i])tarjan(i);
    sort(all(ans));
    printf("%d\n",ans.size());
    for(int i=0;i<ans.size();i++)
    {
        printf("%d",ans[i]);
        putchar(i==ans.size()-1?'\n':' ');
    }
}

int work()
{
    int T;
    io.read(T);
    while(T--)
    {
        io.read(n),io.read(m),init();
        fp(i,1,m)
        {
            int u,v;
            io.read(u),io.read(v);
            if(u>v) swap(u,v);
            e.pb({u,v,0});
            G[u].pb(e.size()-1);
            G[v].pb(e.size()-1);
            h[{u,v}]++;
        }
        solve();
        if(T) printf("\n");
    }
    return 0;
}
```

3.无向图 点双缩点

4.无向图 边双缩点

例题：逃不掉的路

边双连通分量： 不包含桥 或 任意两点都存在至少两条独立路可以相互到达 的图

求法：去掉所有桥后能互达的点构成一个边双分量

缩点之后不存在环，是一棵树

```
struct edge
{
```

```
    int u,v;
    bool flag;
    bool isBridge;
};
int n,m,q;
int fa[MAXN],dfn[MAXN],low[MAXN],tim;
vector<int>  G[MAXN],M[XN];
vector<edge> e;
map<pr,int> h; //判重边
void tarjan(int u)
{
    dfn[u]=low[u]=++tim;
    for(auto i:G[u])
    {
        int v=(e[i].u==u)?e[i].v:e[i].u;
        if(!dfn[v])
        {
            fa[v]=u;
            tarjan(v);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]&&!e[i].flag)
                e[i].isBridge=1;
        }
        else if(fa[u]!=v) low[u]=min(low[u],dfn[v]);
    }
}

int cnt,bel[MAXN];
void dfs(int u)
{
    if(!bel[u]) bel[u]=cnt;
    for(auto i:G[u])
    {
        if(e[i].isBridge) continue;
        int v=(e[i].u==u)?e[i].v:e[i].u;
        if(!bel[v]) dfs(v);
    }
}

void solve()
{
    for(auto &t:e)if(h[{t.u,t.v}]>1)t.flag=1;
    fp(i,1,n)if(!dfn[i])tarjan(i);
    fp(i,1,n)if(!bel[i])cnt++,dfs(i);
    for(auto t:e) //建图
    {
        int &u=bel[t.u],&v=bel[t.v];
        if(v!=u) M[u].pb(v),M[v].pb(u);
    }
    //lca::init();
}

int work()
{
```

```
    scanf("%d%d",&n,&m);
    fp(i,1,m)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        if(u>v) swap(u,v);
        e.pb({u,v,0,0});
        G[u].pb(e.size()-1);
        G[v].pb(e.size()-1);
        h[{u,v}]++;
    }
    return 0;
}
```

5.有向图强连通分量缩点

```
int n,m;
vector<int> G[MAXN];
int tim=0,num=0;
int low[MAXN],dfn[MAXN],bel[MAXN],siz[MAXN];
bool vis[MAXN]; //in stack
vector<int> s;
void init()
{
    s.clear();
    fp(i,1,n)low[i]=dfn[i]=bel[i]=siz[i]=0;
    fp(i,1,n)G[i].clear(); q
    tim=num=0;
}
void tarjan(int u)
{
    dfn[u]=low[u]=++tim,vis[u]=1,s.pb(u);
    for(auto v:G[u])
    {
        if(!dfn[v]) tarjan(v),low[u]=min(low[u],low[v]);
        else if(vis[v]) low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u])
    {
        int cur; num++;
        do{
            cur=s.back(),s.ppb();
            vis[cur]=0,bel[cur]=num,siz[num]++;
        }while(cur!=u);
    }
}
int d[MAXN];
void solve()
{
    fp(i,1,n)if(!dfn[i])tarjan(i);
    fp(u,1,n)for(auto v:G[u])
```

```
            if(bel[u]!=bel[v])
                d[bel[u]]++;
        int ans;
        fp(i,1,num)if(!d[i])ans=i;
        if(siz[ans]==n) puts("No");
        else
        {
            puts("Yes");
            printf("%d %d\n",siz[ans],n-siz[ans]);
            fp(i,1,n)if(bel[i]==ans)printf("%d ",i);printf("\n");
            fp(i,1,n)if(bel[i]!=ans)printf("%d ",i);printf("\n");
        }
    }
```

处理仙人掌

仙人掌找环：

```
void build()
{
    if(a.size()==2) //树边
    {
        ...
    }
    else //环
    {
        ...
    }
}
void tarjan(int u,int fa)
{
    low[u]=dfn[u]=++tim,s.pb(u);
    for(auto e:G1[u])
    {
        int v=e.fi;
        if(v!=fa)
        {
            if(!dfn[v])
            {
                tarjan(v,u),low[u]=min(low[u],low[v]);
                if(low[v]>=dfn[u])
                {
                    a.clear();
while(!s.empty()&&s.back()!=v)a.pb(s.back()),s.ppb();
                    a.pb(s.back()),s.ppb(),a.pb(u),reverse(all(a)),build();
                }
            }
            else low[u]=min(low[u],dfn[v]);
        }
    }
}
```

求环外分支大小

```
int n,m,tim;
int dfn[MAXN],low[MAXN],sz[MAXN];
LL ans=0;
EDGE(MAXN,MAXN*2);
vector<int> s,a;
void solve(vector<int> &a)
{
    if(a.size()==2)return void(ans=(ans+1LL*a[0]*a[1]%MOD)%MOD); // 树边
    // 环边
    int len=a.size(); LL s=0,sj=0,sjj=0,ans=0;
    fp(i,0,a.size()-1)
    {
        LL si=a[i];

        ans+=si*i%MOD*len%MOD*s%MOD,ans%=MOD;
        ans-=si*i%MOD*i%MOD*s%MOD,ans%=MOD;
        ans+=si*i%MOD*sj%MOD,ans%=MOD;
        ans-=si*len%MOD*sj%MOD,ans%=MOD;
        ans+=si*i%MOD*sj%MOD,ans%=MOD;
        ans-=si*sjj%MOD,ans%=MOD;

        s+=si,sj+=si*i%MOD,sjj+=si*i%MOD*i%MOD,s%=MOD,sj%=MOD,sjj%=MOD;
    }
    ans*=inv(len),ans%=MOD,::ans+=ans,::ans%=MOD; return;
}
void dfs(int u,int fa)
{
    low[u]=dfn[u]=++tim,sz[u]=1,s.pb(u);
    go(u)if(v!=fa)
    {
        if(!dfn[v])
        {
            dfs(v,u),low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u])
            {
                a.clear(); int left=n;
                while(!s.empty()&&s.back()!=v)
                    sz[u]+=sz[s.back()],
                    a.pb(sz[s.back()]), // a.pb(s.back()),
                    left-=sz[s.back()],
                    s.ppb();

                sz[u]+=sz[s.back()],
                a.pb(sz[s.back()]), // a.pb(s.back()),
                left-=sz[s.back()],
                s.ppb();

                a.pb(left), // a.pb(u),
                solve(a);
```

```
            }
        }
        else low[u]=min(low[u],dfn[v]);
    }
}
int work()
{
    MC
    {
        scanf("%d%d",&n,&m);
        edge_cnt=tim=ans=0,mst(pre,0),mst(dfn,0),mst(low,0),s.clear();
        fp(i,1,m)
        {
            int u,v;scanf("%d%d",&u,&v);
            addedge(u,v),addedge(v,u);
        }
        dfs(1,0),printf("%lld\n",(ans%MOD+MOD)%MOD);
    }
    return 0;
}
```

仙人掌最短路

考虑将仙人掌缩成(有根)园方树，原点到原点的边权不变，原点到方点的距离为0，方点 u 到原点 v 的距离为环上 v 到 $fa[u]$ 的最短路。

求两点最短路首先求lca，若lca是原点则直接输出距离，若是方点则输出 dis(tu,u)+dis(tv,v)+(u,v)两点在环上的最短路距离。

```
int n,m,q,tim,sz;
vector<pr> G1[MAXN],G2[MAXN];
map<pr,int> weight;
unordered_map<int,int> pos[MAXN];
int f[MAXN][25],low[MAXN],dfn[MAXN],sum[MAXN],d[MAXN],h[MAXN];
vector<int> s,a,pre[MAXN];
int dis(int u,int v){return weight[{min(u,v),max(u,v)}];}
void addedge(int u,int v,int w,vector<pr> G[MAXN]){G[u].pb({v,w}),G[v].pb({u,w});}
void build()
{
    if(a.size()==2)return addedge(a[0],a[1],dis(a[0],a[1]),G2);
    ++n,pre[n].resize(a.size());
    fp(i,1,pre[n].size()-1)
    {
        int d=dis(a[i],a[i-1]);
        pre[n][i]=pre[n][i-1]+d;
        sum[n]+=d;
    }
    sum[n]+=dis(a[0],a.back());
    fp(i,1,a.size()-1)
        addedge(n,a[i],min(pre[n][i],sum[n]-pre[n][i]),G2);
    addedge(a[0],n,0,G2);
```

```
    fp(i,0,a.size()-1)pos[n][a[i]]=pre[n][i];
}
void tarjan(int u,int fa)
{
    low[u]=dfn[u]=++tim,s.pb(u);
    for(auto e:G1[u])
    {
        int v=e.fi;
        if(v!=fa)
        {
            if(!dfn[v])
            {
                tarjan(v,u),low[u]=min(low[u],low[v]);
                if(low[v]>=dfn[u])
                {
                    a.clear();
while(!s.empty()&&s.back()!=v)a.pb(s.back()),s.ppb();
                    a.pb(s.back()),s.ppb(),a.pb(u),reverse(all(a)),build();
                }
            }
            else low[u]=min(low[u],dfn[v]);
        }
    }
}
void dfs(int u,int par)
{
    f[u][0]=par,h[u]=h[par]+1;
    for(auto e:G2[u])
    {
        int v=e.fi,w=e.se;
        if(v!=par)
        {
            d[v]=d[u]+w,dfs(v,u);
        }
    }
}
int dis(int u,int v,int sq)
{
    int d=abs(pos[sq][u]-pos[sq][v]);
    return min(d,sum[sq]-d);
}
int query(int u,int v)
{
    int tu=u,tv=v; if(h[u]<h[v]) swap(u,v),swap(tu,tv);
    for(int k=log2(h[u])+1;k>=0;k--)
        if(h[f[u][k]]>=h[v])
            u=f[u][k];
    if(u==v) return d[tu]-d[tv];
    for(int k=log2(h[u])+1;k>=0;k--)
        if(f[u][k]!=f[v][k])
            u=f[u][k],v=f[v][k];
    int l=f[u][0];
    if(l<=sz)return d[tu]+d[tv]-2*d[l];
    return d[tu]-d[u]+d[tv]-d[v]+dis(u,v,f[u][0]);
```

```
}
int work()
{
    scanf("%d%d%d",&n,&m,&q),sz=n;
    fp(i,1,m)
    {
        int u,v,w;scanf("%d%d%d",&u,&v,&w);
        if(u>v)swap(u,v);addedge(u,v,w,G1),weight[{u,v}]=w;
    }
    tarjan(1,0),dfs(1,0);
    for(int k=1;k<=20;k++)
        for(int i=1;i<=n;i++)
            f[i][k]=f[f[i][k-1]][k-1];
    while(q--)
    {
        int u,v;scanf("%d%d",&u,&v);
        printf("%d\n",query(u,v));
    }
    return 0;
}
```

仙人掌直径

定义仙人掌上两点距离为他们之间的最短路，求仙人掌的直径

```
EDGE(MAXN,N*2);
int n,m,tim,ans=0;
int low[MAXN],dfn[MAXN],f[MAXN],t[MAXN];
vector<int> a,s;
void solve()
{
    if(a.size()==2)
    {
        ans=max(ans,f[a[0]]+f[a[1]]+1);
        f[a[0]]=max(f[a[0]],f[a[1]]+1);
        return;
    }
    int len=a.size();
    fp(i,0,len-1)a.pb(a[i]);
    deque<int> q;
    fp(i,0,a.size()-1)
    {
        while(!q.empty()&&i-q.front()>len/2)q.ppf();
        if(!q.empty()){int j=q.front();ans=max(ans,f[a[i]]+i-j+f[a[j]]);}
        while(!q.empty()&&f[a[i]]-i>=f[a[q.back()]]-q.back())q.ppb();q.pb(i);
    }
    fp(i,1,len-1)f[a[0]]=max(f[a[0]],f[a[i]]+min(i,len-i));
}
void dfs(int u,int fa)
{
    low[u]=dfn[u]=++tim,s.pb(u);
```

```
    go(u)if(v!=fa)
    {
        if(!dfn[v])
        {
            dfs(v,u),low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u])
            {
                a.clear();while(!s.empty()&&s.back()!=v)a.pb(s.back()),s.ppb();
                a.pb(s.back()),s.ppb(),a.pb(u),reverse(all(a)),solve();
            }
        }
        else low[u]=min(low[u],dfn[v]);
    }
}
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,m)
    {
        int k;scanf("%d",&k);
        fp(j,1,k)
        {
            scanf("%d",&t[j]);
            if(j!=1)addedge(t[j],t[j-1]),addedge(t[j-1],t[j]);
        }
    }
    dfs(1,0),printf("%d\n",ans);
    return 0;
}
```

树上 k 级祖先

O(nlogn+qlogn) 在线

倍增

O(n+q) 离线

dfs过程中维护个栈

O(nlogn+q) 在线

长链剖分

```
EDGE(MAXN,MAXN*2);
int n,q;
int h[MAXN],son[MAXN],f[MAXN][21],bel[MAXN],mx[MAXN];
int cnt,top[MAXN];
vector<int> up[MAXN],down[MAXN];
int highbit[MAXN];
void dfs1(int u,int fa=0)
```

```
{
    f[u][0]=fa,h[u]=h[fa]+1,son[u]=0,mx[u]=h[u];
    go(u)if(v!=fa)
    {
        dfs1(v,u),mx[u]=max(mx[u],mx[v]);
        if(mx[v]>mx[son[u]])son[u]=v;
    }
}
void dfs2(int u,int tp)
{
    if(!bel[tp])++cnt,bel[tp]=cnt,top[cnt]=tp;
    bel[u]=bel[tp];
    if(son[u])dfs2(son[u],tp);
    go(u)if(v!=f[u][0]&&v!=son[u])dfs2(v,v);
}
void init()
{
    fp(i,1,n)fd(j,20,0)if((1<<j)&i){highbit[i]=j;break;}
    fp(j,1,20)fp(i,1,n)f[i][j]=f[f[i][j-1]][j-1];
    fp(i,1,cnt)
    {
        int u=top[i],len=0;
        for(int x=u;x;x=son[x]) down[i].pb(x),len++;
        for(int x=u;x&&h[u]-h[x]<=len;x=f[x][0])up[i].pb(x);
    }
}
int query(int u,int k)
{
    if(!k)return u;
    int j=highbit[k];
    u=f[u][j],k-=(1<<j);
    int tp=top[bel[u]];
    if(h[u]-h[tp]>=k) return down[bel[u]][h[u]-h[tp]-k];
    else return up[bel[u]][k-(h[u]-h[tp])];
}
```

lca

1.倍增

```
namespace lca //记得初始化 lca::n
{
    //判断点x在u到v的路径上 -> dis(u,x)+dis(x,v)==dis(u,v)
    int n,f[MAXN][25],d[MAXN];
    void dfs(int u)
    {
        go(u)if(v!=f[u][0])
            f[v][0]=u,d[v]=d[u]+1,dfs(v);
    }
    void init(int sz)
    {
```

```
        n=sz,d[1]=1,dfs(1);
        for(int k=1;k<=20;k++)
            for(int i=1;i<=n;i++)
                f[i][k]=f[f[i][k-1]][k-1];
    }
    pr query(int u,int v) //返回{lca(u,v),dis(u,v)}
    {
        int tu=u,tv=v;
        if(d[u]<d[v]) swap(u,v),swap(tu,tv);
        for(int k=log2(d[u])+1;k>=0;k--)
            if(d[f[u][k]]>=d[v])
                u=f[u][k];
        if(u==v) return {tv,d[tu]-d[tv]};
        for(int k=log2(d[u])+1;k>=0;k--)
            if(f[u][k]!=f[v][k])
                u=f[u][k],v=f[v][k];
        int l=f[u][0];
        return {l,d[tu]+d[tv]-2*d[l]};
    }
    int kth_fa(int u,int k) //k级祖先 需保证存在
    {
        fd(i,20,0)if((1<<i)&k)u=f[u][i];
        return u;
    }
    int move(int u,int v,int k) //u向v走k步 需保证dis(u,v)>=k
    {
        pr t=query(u,v); int l=t.fi;
        if(d[u]-d[l]>=k)return kth_fa(u,k);
        return kth_fa(v,t.se-k);
    }
}
```

2.欧拉序+RMQ

```
namespace lca //不用初始化n 。。。
{
    int n,seq[MAXN*2],pos[MAXN],d[MAXN],f[MAXN*2][25];
    void dfs(int u,int fa)
    {
        seq[++n]=u,pos[u]=n;
        gow(u)if(v!=fa)
            d[v]=d[u]+1,dfs(v,u),seq[++n]=u;
    }
    void init(int S)
    {
        n=0,d[S]=1,dfs(S,0);
        fp(i,1,n)f[i][0]=seq[i];
        for(int k=1;(1<<k)<=n;k++)
            for(int i=1;i+(1<<k)-1<=n;i++)
                if(d[f[i][k-1]]<d[f[i+(1<<(k-1))][k-1]])
                        f[i][k]=f[i][k-1];
```

```
                    else
                        f[i][k]=f[i+(1<<(k-1))][k-1];
    }
    int query(int u,int v)
    {
        if(pos[v]<pos[u]) swap(u,v);
        int k=log2(pos[v]-pos[u]+1);
        if(d[f[pos[u]][k]]<d[f[pos[v]-(1<<k)+1][k]])
            return f[pos[u]][k];
        else
            return f[pos[v]-(1<<k)+1][k];
    }
}
```

Spfa 判负环

```
//找到一个从顶点1能到达的负环
//注意图不连通时负环的判断
int n,m,S;
int d[MAXN],in[MAXN];
bool vis[MAXN];
int pre[MAXN];
struct edge{int u,v,w,next;};
vector<edge> e={edge()};
void addedge(int u,int v,int w){e.pb({u,v,w,pre[u]}),pre[u]=e.size()-1;}
bool spfa()
{
    queue<int> q;
    fp(i,1,n)d[i]=inf,vis[i]=0,in[i]=0;
    d[S]=0,in[S]=1;
    q.push(S),vis[S]=1;
    while(!q.empty())
    {
        int u=q.front();
        q.pop(),vis[u]=0;
        gow(u)if(d[u]+w<d[v])
        {
            d[v]=d[u]+w;
            in[v]++; if(in[v]>=n) return 1;
            if(!vis[v]) q.push(v);
        }
    }
    return 0;
}
int work()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&m);
```

```
        fp(i,1,n)pre[i]=0;
        e.clear(),e.pb(edge());
        fp(i,1,m)
        {
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            addedge(u,v,w);
            if(w>=0) addedge(v,u,w);
        }
        S=1,puts(spfa()?"YE5":"N0");
    }
    return 0;
}
```

点分治

普通

```
int n,m,root,sum;
int siz[MAXN],mx[MAXN],cnt[3];
vector<int> G[MAXN];
bool vis[MAXN];
LL ans=0,d[MAXN];

int pre[MAXN];
struct edge{int u,v,w,next;};
vector<edge> e;
void addedge(int u,int v,int w){e.pb({u,v,w,pre[u]}),pre[u]=e.size()-1;}

void find_root(int u,int fa) //找重心
{
    siz[u]=1,mx[u]=0;
    gow(u)if(v!=fa&&!vis[v])
    {
        find_root(v,u);
        siz[u]+=siz[v];
        mx[u]=max(mx[u],siz[v]);
    }
    mx[u]=max(mx[u],sum-siz[u]);
    if(!root||mx[u]<mx[root]) root=u;
}
void calc_deep(int u,int fa)
{
    cnt[d[u]%3]++;
    gow(u)if(v!=fa&&!vis[v])
        d[v]=d[u]+w,calc_deep(v,u);
}
LL calc(int u,LL dis)
{
    cnt[0]=cnt[1]=cnt[2]=0,d[u]=dis,calc_deep(u,0);
    return cnt[0]*cnt[0]+2*cnt[1]*cnt[2];
```

```
}
void solve(int u)
{
    vis[u]=1,ans+=calc(u,0);
    gow(u)if(!vis[v])ans-=calc(v,w);
    gow(u)if(!vis[v])root=0,sum=siz[v],find_root(v,u),solve(root);
}
LL gcd(LL a,LL b){return b==0?a:gcd(b,a%b);}

int work()
{
    e.pb(edge());
    scanf("%d",&n);
    fp(i,1,n-1)
    {
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        addedge(u,v,w),addedge(v,u,w);
    }
    sum=n,root=0,find_root(1,0),solve(root);
    LL a=ans,b=n*n,g=gcd(a,b); a/=g,b/=g;
    return printf("%lld/%lld\n",a,b);
}
```

动态(点分树)

通常套路：用数据结构维护点分树中 u 这颗子树(分治联通快)所有点到 u 的信息($info1$)，和所有点到 $par[u]$ 的信息($info2$)用于去重。

计算答案时跳点分树用 $info1$ 计算当前联通快内答案，用 $info2$ 去重。

震波

求距离点 u 不超过 k 的点的点权和，单点修改。

```
EDGE(MAXN,MAXN*2);
namespace lca; //欧拉序lca模板
int cnt,ls[N],rs[N],s[N];
void update(int &x,int l,int r,int pos,int val)
{
    if(!x)x=++cnt; if(l==r){s[x]+=val;return;}
    if(pos<=mid)update(ls[x],l,mid,pos,val);
    else update(rs[x],mid+1,r,pos,val);
    s[x]=s[ls[x]]+s[rs[x]];
}
int query(int x,int l,int r,int nl,int nr)
{
    if(!x||nl>nr||nl>r||nr<l)return 0;
    if(l==nl&&r==nr)return s[x];
    if(nr<=mid)return query(ls[x],l,mid,nl,nr);
    if(nl> mid)return query(rs[x],mid+1,r,nl,nr);
    return query(ls[x],l,mid,l,mid)+query(rs[x],mid+1,r,mid+1,nr);
```

```
}
int n,m,root,sum;
int a[MAXN],siz[MAXN],mx[MAXN],sz[MAXN];
int dis[MAXN],par[MAXN],fa[MAXN];
bool vis[MAXN]; vector<int> G[MAXN];
void find_root(int u,int fa)
{
    siz[u]=1,mx[u]=0;
    go(u)if(v!=fa&&!vis[v])find_root(v,u),siz[u]+=siz[v],mx[u]=max(mx[u],siz[v]);
    mx[u]=max(mx[u],sum-siz[u]);
    if(!root||mx[u]<mx[root])root=u;
}
void solve(int u,int fa)
{
    vis[u]=1; if(fa)G[fa].pb(u),::fa[u]=fa;
    go(u)if(!vis[v])root=0,sum=siz[v],find_root(v,u),solve(root,u);
}
void update(int x,int u,int val)
{
    update(dis[u],0,sz[u],lca::dis(u,x),val);
    if(fa[u])
        update(par[u],0,sz[u]+1,lca::dis(fa[u],x),val),
        update(x,fa[u],val);
}
void calc(int x,int u,int v,int k,int &ans)
{
    int d=k-lca::dis(u,x);
    if(d>=0)
    {
        ans+=query(dis[u],0,sz[u],0,d);
        if(v)ans-=query(par[v],0,sz[v]+1,0,d);
    }
    if(fa[u])calc(x,fa[u],u,k,ans);
}
void dfs(int u){sz[u]=1;for(auto v:G[u])dfs(v),sz[u]+=sz[v];}
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,n)scanf("%d",&a[i]);
    fp(i,1,n-1)
    {
        int u,v;scanf("%d%d",&u,&v);
        addedge(u,v),addedge(v,u);
    }
    sum=n,find_root(1,0);int t=root;find_root(t,0);solve(t,0);lca::init(t);dfs(t);
    fp(i,1,n)update(i,i,a[i]);
    int last=0;
    while(m--)
    {
        int t,x,y;scanf("%d%d%d",&t,&x,&y);
        x^=last,y^=last;
        if(!t) last=0,calc(x,x,0,y,last),printf("%d\n",last);
        else update(x,x,-a[x]+y),a[x]=y;
    }
```

```
    return 0;
}
```

二分图匹配

最大流

转为网络流跑 Dinic，复杂度为 $O(\sqrt{n}m)$，$n=m=1e5$ 时可以考虑

匈牙利

$O(nm)$

```
EDGE(MAXN,MAXN*2);
int n,m;
int lk[MAXN];
bool vis[MAXN];
bool dfs(int u)
{
    go(u)if(!vis[v])
    {
        vis[v]=1;
        if(!lk[v]||dfs(lk[v]))
            {lk[v]=u;return 1;}
    }
    return 0;
}
int solve()
{
    int ans=0;
    fp(i,1,n)mst(vis,0),ans+=dfs(i);
    return ans;
}
```

二分图最大权匹配(KM)

$O(N^3)$

```
int n,m;
LL w[MAXN][MAXN],lx[MAXN],ly[MAXN],slack[MAXN];
int lk[MAXN],pre[MAXN];
bool visy[MAXN];

void bfs(int k)
{
    int x=0,y=0,yy=0; LL delta;
    fp(i,1,n)slack[i]=linf;
    mst(pre,0),lk[y]=k;
    while(1)
```

```
    {
        x=lk[y],delta=linf,visy[y]=1;
        fp(i,1,n)if(!visy[i])
        {
            if(slack[i]>lx[x]+ly[i]-w[x][i])
                slack[i]=lx[x]+ly[i]-w[x][i],pre[i]=y;
            if(slack[i]<delta) delta=slack[i],yy=i;
        }
        fp(i,0,n)
            if(visy[i]) lx[lk[i]]-=delta,ly[i]+=delta;
            else slack[i]-=delta;
        y=yy;
        if(lk[y]==-1) break;
    }
    while(y) lk[y]=lk[pre[y]],y=pre[y];
}
void solve()
{
    mst(lx,0),mst(ly,0),mst(lk,-1);
    fp(i,1,n) mst(visy,0),bfs(i);
}

int work()
{
    while(scanf("%d",&n)!=EOF)
    {
        fp(i,1,n)fp(j,1,n)scanf("%lld",&w[i][j]);
        solve();
        LL ans=0;
        fp(i,1,n)ans+=lx[i]+ly[i];
        static int Case = 0;
        printf("%lld\n",ans);
    }
    return 0;
}
```

一般图最大匹配(带花树)

```
EDGE(MAXN,MAXN*2);
int n,m,ans,tim;
int f[MAXN],nxt[MAXN],match[MAXN],vis[MAXN],dfn[MAXN];
queue<int> q;
int find(int x){return f[x]==x?x:f[x]=find(f[x]);}
int lca(int u,int v)
{
    ++tim,u=find(u),v=find(v);
    while(dfn[u]!=tim)
    {
        dfn[u]=tim;
        u=find(nxt[match[u]]);
        if(v)swap(u,v);
```

```
    }
    return u;
}
void blossom(int x,int y,int l)
{
    while(find(x)!=l)
    {
        nxt[x]=y,y=match[x];
        if(vis[y]==2)vis[y]=1,q.push(y);
        if(find(x)==x)f[x]=l;
        if(find(y)==y)f[y]=l;
        x=nxt[y];
    }
}
bool aug(int S)
{
    fp(i,1,n)f[i]=i,vis[i]=nxt[i]=0;
    while(!q.empty())q.pop();q.push(S),vis[S]=1;
    while(!q.empty())
    {
        int u=q.front(); q.pop();
        go(u)
        {
            if(find(u)==find(v)||vis[v]==2)continue;
            if(!vis[v])
            {
                vis[v]=2,nxt[v]=u;
                if(!match[v])
                {
                    for(int x=v,last;x;x=last)
                        last=match[nxt[x]],match[x]=nxt[x],match[nxt[x]]=x;
                    return 1;
                }
                vis[match[v]]=1,q.push(match[v]);
            }
            else
            {
                int l=lca(u,v);
                blossom(u,v,l);
                blossom(v,u,l);
            }
        }
    }
    return 0;
}
int work()
{
    scanf("%d%d",&n,&m);
    fp(i,1,m)
    {
        int u,v;scanf("%d%d",&u,&v);
        addedge(u,v),addedge(v,u);
    }
    fp(i,1,n)if(!match[i])ans+=aug(i);
```

```
    printf("%d\n",ans);
    fp(i,1,n)printf("%d ",match[i]);
    return 0;
}
```

一般图最大权匹配

最大流

Dinic 用容量不为 0 的边构建层次图，在层次图上有方向的进行多路增广。

```
int n,m,S,T;
struct edge
{
    int u,v,next;
    LL flow;
}e[N*2];
int pre[MAXN],cur[MAXN],cnt=1;
void add(int u,int v,LL flow)
{
    e[++cnt]={u,v,pre[u],flow},pre[u]=cnt;
}
int dis[MAXN],q[MAXN];
bool bfs() //构建分层图
{
    int h=1,t=0;
    mst(dis,0),q[++t]=S,dis[S]=1;
    while(h<=t)
    {
        int u=q[h++];
        for(int i=pre[u];i;i=e[i].next)
        {
            int v=e[i].v;
            if(!dis[v]&&e[i].flow)
                dis[v]=dis[u]+1,q[++t]=v;
        }
    }
    return dis[T];
}
LL dfs(int u,LL flow) //多路增广
{
    if(u==T||!flow)return flow;
    LL ret=0,d=0;
    for(int i=cur[u];i;i=e[i].next)
    {
        int v=e[i].v; cur[u]=i; //当前弧优化
        if(dis[v]==dis[u]+1&&e[i].flow)
        {
            d=dfs(v,min(flow-ret,e[i].flow));
            if(d)ret+=d,e[i].flow-=d,e[i^1].flow+=d;
            if(ret==flow)return ret;
```

```
        }
    }
    return ret;
}
LL dinic()
{
    LL ans=0;
    while(bfs())
    {
        fp(i,1,n)cur[i]=pre[i];
        ans+=dfs(S,linf);
    }
    return ans;
}
int work()
{
    scanf("%d%d%d%d",&n,&m,&S,&T);
    fp(i,1,m)
    {
        int u,v;LL flow;
        scanf("%d%d%lld",&u,&v,&flow);
        add(u,v,flow),add(v,u,0);
    }
    return printf("%lld\n",dinic());
}
```

费用流

1. Dinic Bfs改最短路算法

将 Dinic 中的 bfs 构建层次图改成最短路算法（Spfa或带势函数的Dijkstra），每次沿着最短路图的方向进行增广。

Spfa

```
int n,m,S,T;
struct edge{int u,v,next;LL flow,w;}e[N];
int pre[MAXN],cur[MAXN],cnt=1;
void add(int u,int v,LL flow,LL w){e[++cnt]={u,v,pre[u],flow,w},pre[u]=cnt;}

bool vis[MAXN];
int q[MAXN],h,t; LL dis[MAXN];
bool spfa()//建议用stl queue 手写要换成循环队列防越界
{
    fp(i,1,n)dis[i]=linf; mst(vis,0);
    h=1,t=0,vis[S]=1,dis[S]=0,q[++t]=S;
    while(h<=t)
    {
        int u=q[(h++)%MAXN]; vis[u]=0;
        gow(u)if(e[i].flow&&dis[v]>dis[u]+w)
        {
```

```
            dis[v]=dis[u]+w;
            if(!vis[v])vis[v]=1,q[(++t)%MAXN]=v;
        }
    }
    return dis[T]<linf;
}
lpr ans;
LL dfs(int u,LL flow)
{
    if(u==T||!flow) return ans.fi+=flow,ans.se+=dis[u]*flow,flow;
    LL ret=0,d; vis[u]=1; //vis标记 防止在费用为 0 的边上反复跳
    for(int i=cur[u];i;i=e[i].next)
    {
        int v=e[i].v;LL w=e[i].w; cur[u]=i;
        if(e[i].flow&&dis[u]+w==dis[v]&&!vis[v])
        {
            d=dfs(v,min(e[i].flow,flow-ret));
            if(d) e[i].flow-=d,e[i^1].flow+=d,ret+=d;
            if(ret==flow)return ret;
        }
    }
    return ret;
}
void mcf()
{
    while(spfa())
    {
        mst(vis,0);
        fp(i,1,n)cur[i]=pre[i];
        dfs(S,linf);
    }
}
int work()
{
    scanf("%d%d",&n,&m); scanf("%d%d",&S,&T);
    fp(i,1,m)
    {
        int u,v; LL f,w;
        scanf("%d%d%lld%lld",&u,&v,&f,&w);
        add(u,v,f,w),add(v,u,0,-w);
    }
    mcf();
    return printf("%lld %lld\n",ans.fi,ans.se);
}
```

Dijkstra

```
int n,m,S,T;
struct edge{int u,v,next;LL flow,w;}e[N];
int pre[MAXN],cur[MAXN],cnt=1;
void add(int u,int v,LL flow,LL w)
```

```
{
    e[++cnt]={u,v,pre[u],flow,w},pre[u]=cnt;
}
bool vis[MAXN];
LL dis[MAXN],h[MAXN];
bool dij()
{
    priority_queue<lpr> q;
    fp(i,1,n)dis[i]=linf;
    dis[S]=0,q.push({0,S});
    while(!q.empty())
    {
        int u=q.top().se; LL d=-q.top().fi; q.pop();
        if(d!=dis[u])continue;
        gow(u)if(e[i].flow&&dis[u]+w+h[u]-h[v]<dis[v])
            dis[v]=dis[u]+w+h[u]-h[v],q.push({-dis[v],v});
    }
    //fp(i,1,n)cout<<dbgs2(i,dis[i])<<endl;
    return dis[T]<linf;
}
lpr ans;
LL dfs(int u,LL flow)
{
    //cout << dbgs(u) << endl;
    if(u==T||!flow) return ans.fi+=flow,ans.se+=(dis[u]+h[u])*flow,flow;
    LL ret=0,d; vis[u]=1;
    for(int i=cur[u];i;i=e[i].next)
    {
        int v=e[i].v;LL w=e[i].w; cur[u]=i;
        if(e[i].flow&&dis[u]+w+h[u]-h[v]==dis[v]&&!vis[v])
        {
            d=dfs(v,min(e[i].flow,flow-ret));
            if(d) e[i].flow-=d,e[i^1].flow+=d,ret+=d;
            if(ret==flow)return ret;
        }
    }
    return ret;
}
void mcf()
{
    while(dij())
    {
        mst(vis,0);
        fp(i,1,n)cur[i]=pre[i];
        dfs(S,linf);
        fp(i,1,n)if(dis[T]<linf)h[i]+=dis[i]; //维护势函数
    }
}
int work()
{
    scanf("%d%d",&n,&m);
    scanf("%d%d",&S,&T);
    fp(i,1,m)
    {
```

```
        int u,v; LL f,w;
        scanf("%d%d%lld%lld",&u,&v,&f,&w);
        add(u,v,f,w),add(v,u,0,-w);
    }
    mcf();
    return printf("%lld %lld\n",ans.fi,ans.se);
}
```

2.EK

Dijkstra

```
int n,m,S,T;
struct edge{int u,v,next;LL flow,w;}e[N];
int pre[MAXN],cur[MAXN],cnt=1;
void add(int u,int v,LL flow,LL w)
{
    e[++cnt]={u,v,pre[u],flow,w},pre[u]=cnt;
}
bool vis[MAXN];
LL dis[MAXN],h[MAXN];
int from[MAXN];
bool dij()
{
    priority_queue<lpr> q;
    fp(i,1,n)dis[i]=linf;
    dis[S]=0,q.push({0,S});
    while(!q.empty())
    {
        int u=q.top().se; LL d=-q.top().fi; q.pop();
        if(d!=dis[u])continue;
        gow(u)if(e[i].flow&&dis[u]+w+h[u]-h[v]<dis[v])
            dis[v]=dis[u]+w+h[u]-h[v],q.push({-dis[v],v}),from[v]=i;
    }
    return dis[T]<linf;
}
lpr ans;
void mcf()
{
    while(dij())
    {
        LL flow=linf;
        for(int u=T;u!=S;u=e[from[u]].u)
            flow=min(flow,e[from[u]].flow);
        ans.fi+=flow,ans.se+=(dis[T]+h[T])*flow;
        for(int u=T;u!=S;u=e[from[u]].u)
            e[from[u]].flow-=flow,
            e[from[u]^1].flow+=flow;
        fp(i,1,n)if(dis[i]<linf)h[i]+=dis[i];
    }
}
```

```
int work()
{
    scanf("%d%d",&n,&m);
    scanf("%d%d",&S,&T);
    fp(i,1,m)
    {
        int u,v; LL f,w;
        scanf("%d%d%lld%lld",&u,&v,&f,&w);
        add(u,v,f,w),add(v,u,0,-w);
    }
    mcf();
    return printf("%lld %lld\n",ans.fi,ans.se);
}
```

欧拉回路

欧拉路径：奇度节点只有两个，连边做一边欧拉回路

```
int n,m;
struct edge{int i,next;}t[N];
int head[MAXN],cnt;
bool vis[MAXN],con[MAXN];
int d[MAXN];
pr e[MAXN];
vector<int> ans;
void addedge(int u,int i)
{
    ++cnt;
    t[cnt].i=i;
    t[cnt].next=head[u];
    head[u]=cnt;
}
void dfs(int u,int from)
{
    con[u]=1;
    for(rg int k=head[u];k;k=t[k].next)
    {
        int i=t[k].i;
        if(!vis[i])
        {
            vis[i]=1;
            dfs(e[i].fi!=u?e[i].fi:e[i].se,i);
        }
        while(t[k].next&&vis[t[t[k].next].i])
            t[k].next=t[t[k].next].next;
    }
    if(from)ans.pb((e[from].se==u?1:-1)*from);
}
bool eula(int type) // 1无向 2有向
{
    ans.clear();
```

```
    mst(con,0),mst(vis,0),mst(head,0),mst(d,0);
    cnt=0;
    fp(i,1,m)
    {
        if(type==1)
        {
            d[e[i].fi]++;
            d[e[i].se]++;
            addedge(e[i].fi,i);
            addedge(e[i].se,i);
        }
        else
        {
            d[e[i].fi]++;
            d[e[i].se]--;
            addedge(e[i].fi,i);
        }
    }
    fp(i,1,n)
    {
        if(type==1){if(d[i]&1)return 0;}
        else{if(d[i]!=0)return 0;}
    }
    bool flag=1;
    fp(i,1,n)if(head[i]&&!con[i])
    {
        if(flag)dfs(i,0),flag=0;
        else return 0; //不连通
    }
    reverse(all(ans));
    return 1;
}
int work()
{
    int type;
    scanf("%d",&type);
    scanf("%d%d",&n,&m);
    fp(i,1,m)scanf("%d%d",&e[i].fi,&e[i].se);
    if(!eula(type))puts("NO");
    else
    {
        puts("YES");
        for(auto x:ans)printf("%d ",x);
    }
    return 0;
}
```

计算几何

二维计算几何

```
int sgn(double x)
{
    if(x<-eps)return -1;
    if(x> eps)return  1;
    return 0;
}
struct vec
{
    double x,y;
    vec(){x=y=0;}
    vec(double _x,double _y){x=_x,y=_y;}
    vec operator +(vec v){return vec(x+v.x,y+v.y);}
    vec operator -(vec v){return vec(x-v.x,y-v.y);}
    vec operator *(double k){return vec(k*x,k*y);}
    vec operator /(double k){return vec(x/k,y/k);}
    double operator *(vec v){return x*v.x+y*v.y;} //点积
    double len(){return hypot(x,y);}
};

double cross(vec a,vec b) //叉积
{
    return a.x*b.y-a.y*b.x;
}

bool point_on_seg(vec p,vec a,vec b) //点在线段上
{
    return sgn((p-a).len()+(p-b).len()-(a-b).len())==0;
    //return sgn(cross(b-a,p-a))==0&&sgn((p-a)*(p-b))<=0;
}
int seg_have_inter(vec a,vec b,vec p,vec q) //线段判交
{
    int d1=sgn(cross(b-a,p-a)),d2=sgn(cross(b-a,q-a));
    int d3=sgn(cross(q-p,a-p)),d4=sgn(cross(q-p,b-p));
    if(d1*d2<0&&d3*d4<0) return 1; //有交点且不在端点
    if(
        (d1==0&&point_on_seg(p,a,b))||
        (d2==0&&point_on_seg(q,a,b))||
        (d3==0&&point_on_seg(a,p,q))||
        (d4==0&&point_on_seg(b,p,q))
    ) return -1; //交在线段端点上
    return 0;
}
bool line_parallel(vec a,vec b,vec c,vec d) //直线平行
{
    return sgn(cross(b-a,d-c))==0;
}
bool line_same(vec a,vec b,vec c,vec d) //直线重叠
{
    return line_parallel(a,b,c,d)&&sgn(cross(a-c,a-d))==0;
}
vec line_inter(vec a,vec b,vec c,vec d) //直线求交点
{
    vec base=d-c;
```

```
    double d1 = cross(base,a-c);
    double d2 = cross(base,b-c);
    return vec(
        (a.x*d2-b.x*d1)/(d2-d1),
        (a.y*d2-b.y*d1)/(d2-d1)
    );
}
```

二维凸包

多边形判凸：所有点都在凸包上

```
ld slope(pt x,pt y)
{
    if(y.fi-x.fi==0)return 1.0/0.0;
    return (y.se-x.se)/(y.fi-x.fi);
}
ld dis(pt x,pt y){return sqrt(sqr(y.se-x.se)+sqr(y.fi-x.fi));}
void solve()
{
    sort(all(p)),unq(p);
    for(auto x:p) //求上凸包
    {
        while(u.size()>=2&&
            slope(u[u.size()-1],u[u.size()-2])<slope(u[u.size()-2],x))
                u.ppb();
        u.pb(x);
    }
    reverse(all(p));while(p.back()!=u.front())p.ppb();reverse(all(p));
    sort(all(p),[](const pt &a,const pt &b)
    {
        if(a.fi!=b.fi)return a.fi<b.fi;
        if(a.fi==u.front().fi)return a.se==u.front().se;
        return a.se>b.se;
    });
    for(auto x:p) //下凸包
    {
        //cout << dbgs2(x.fi,x.se) << endl;
        while(l.size()>=2&&slope(l[l.size()-1],l[l.size()-2])>
            slope(l[l.size()-2],x))l.ppb();
        l.pb(x);
        if(x==u.back())break;
    }
}
```

最小圆覆盖

最小球覆盖

模拟退火 n=100

```
const double eps = 1e-6;

int n;
struct point{double x,y,z;}p[MAXN];

double dis(point a,point b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z));
}

void solve()
{
    const double step = 0.998;
    double ans=1e20,temp=10000;
    point cur={0,0,0};
    while(temp>eps)
    {
        point far=p[1];
        fp(i,1,n)if(dis(p[i],cur)>dis(far,cur))far=p[i];
        if(dis(far,cur)<ans)ans=dis(far,cur);
        cur.x+=(far.x-cur.x)*(temp/10000);
        cur.y+=(far.y-cur.y)*(temp/10000);
        cur.z+=(far.z-cur.z)*(temp/10000);
        temp*=step;
    }
    cout << setprecision(10) << ans << endl; //include<iomanip>
}

int work()
{
    while(cin >> n && n)
    {
        fp(i,1,n) cin>>p[i].x>>p[i].y>>p[i].z;
        solve();
    }
    return 0;
}
```

其他

模拟退火

```
const db T0    = 1e7;
const db T_end = 1e-1;
const db D     = 1-(3e-3);
const int L    = 100;

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int n,r,ans=0;
vector<pr> v;
int p[10],t[10];
map<int,int> mn,mx;
int dis(pr x,pr y){return sqr(x.fi-y.fi)+sqr(x.se-y.se);}
int cost(int p[])
{
    int s=0;
    fp(i,1,n)
        fp(j,1,i-1)
            s+=dis(v[p[i]],v[p[j]]);
    return s;
}
inline db rand_f()
{
    const int MAX = 1000000000;
    return db(rng()%MAX)/db(MAX);
}
inline bool check(db delta,db T){return delta>0||rand_f()<exp(delta/T);} //保证
delta<0
void SA()
{
    fp(i,1,n)p[i]=rng()%v.size();
    for(db T=T0;T>T_end;T=T*D)
    {
        fp(tim,1,L)
        {
            // fp(i,1,n)t[i]=(p[i]+rng())%v.size();

            //构造新解
            int pos=rng()%n+1;
            t[pos]=rng()%v.size();

            if(check(cost(t)-cost(p),T))swap(t,p);
            ans=max({ans,cost(t),cost(p)});
        }
        // cout << dbgs2(T,cost(p)) << endl;
    }
    // cout << dbgs(cost(p)) << endl;
}
int solve()
{
    v.clear(),ans=0;
    fp(i,-r,r)mn[i]=inf,mx[i]=-inf;
    fp(x,-r,r)fp(y,-r,r)
        if(x*x+y*y<=r*r)
```

```
            mx[x]=max(mx[x],y),
            mn[x]=min(mn[x],y);
    fp(x,-r,r)v.pb({x,mn[x]}),v.pb({x,mx[x]});
    // int num=min({int(v.size()),4*r+10});
    // sort(all(v),[](pr x,pr y){return dis(x,{0,0})>dis(y,{0,0});});
    // while(v.size()>num)v.ppb();
    SA();
    return ans;
}

int work()
{
    // freopen("output.txt", "w", stdout);
    // n=7,r=29;
    // int tim = 20;
    // while(tim--)
        // cout << solve() << endl;


    fp(i,1,8)
    {
        fp(j,1,30)
        {
            n=i,r=j;
            // cout << dbgs3(n,r,solve()) << endl;
            int ans=0;
            fp(tim,1,1)
                ans=max(ans,solve());
            printf("a[%d][%d] = %d;\n",n,r,ans);
        }
    }
    return 0;
}
```

unordered_map 防爆

Blowing up unordered_map, and how to stop getting hacked on it

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
```

```
    }
};
//Now we can simply define our unordered_map or our gp_hash_table as follows:
unordered_map<long long, int, custom_hash> safe_map;
gp_hash_table<long long, int, custom_hash> safe_hash_table;
```

stl 更快的 hash table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
```

分数类

```
LL gcd(LL a,LL b){return b==0?a:gcd(b,a%b);}
struct fraction
{
    LL a,b; bool zero;
    void reduce()
    {
        if(!a){a=b=0,zero=1;return;}
        LL g=gcd(abs(a),abs(b));a/=g,b/=g;
    }
    fraction(LL _a=0,LL _b=0,bool _z=1){a=_a,b=_b,zero=_z;}
    void operator +=(const fraction &t){*this = *this + t;}
    fraction operator +(const fraction &t)const
    {
        if(zero)   return t;
        if(t.zero) return *this;
        fraction res = {a*t.b+b*t.a,b*t.b,0};
        res.reduce();
        return res;
    }
    void operator -=(const fraction &t){*this = *this - t;}
    fraction operator -(const fraction &t)const
    {
        if(zero)   return t;
        if(t.zero) return *this;
        fraction res = {a*t.b-b*t.a,b*t.b,0};
        res.reduce();
        return res;
    }
    void operator *=(const fraction &t){*this = *this * t;}
    fraction operator *(const fraction &t)const
    {
        if(zero||t.zero)return fraction();
        fraction res = {a*t.a,b*t.b,0};
        res.reduce();
        return res;
```

```
    }
    void operator /=(const fraction &t){*this = *this / t;}
    fraction operator /(const fraction &t)const
    {
        if(zero||t.zero)throw -1;
        fraction res = {abs(a*t.b),abs(b*t.a),0};
        if((a>0)^(t.a>0))res.a=-res.a;
        res.reduce();
        return res;
    }
    void print()
    {
        cout << dbgs2(a,b) << endl;
    }
};
```

mt19937

```
#include <cstdio>
#include <chrono>
#include <random>
using namespace std;

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int main()
{
    shuffle(a, a + n, rng); //random_shuffle()
    printf("%u\n", rng());
    return 0;
}
```

预处理log2

$O(logn+n)$

```
    lg2[1]=0;
    for(rg int i=2;i<=n;i<<=1)lg2[i]=1;
    fp(i,1,n)lg2[i]+=lg2[i-1];
```

基数排序

模板

```
//二进制每8为作为一个关键字
const int N = 256;
```

```
void radix_sort(int a[],int n)
{
    static int r1[N],r2[N],r3[N],r4[N];
    static int t1[MAXN],t2[MAXN];
    mst(r1,0),mst(r2,0),mst(r3,0),mst(r4,0);
    fp(i,1,n)
    {
        ++r1[(a[i]    )&0xff]; //255
        ++r2[(a[i]>>8 )&0xff];
        ++r3[(a[i]>>16)&0xff];
        ++r4[(a[i]>>24)&0xff];
    }
    fp(i,1,N-1)
    {
        r1[i]+=r1[i-1];
        r2[i]+=r2[i-1];
        r3[i]+=r3[i-1];
        r4[i]+=r4[i-1];
    }
    fp(i,1,n)t1[i]=i;
    fd(i,n,1)t2[r1[(a[t1[i]]    )&0xff]--]=t1[i];
    fd(i,n,1)t1[r2[(a[t2[i]]>>8 )&0xff]--]=t2[i];
    fd(i,n,1)t2[r3[(a[t1[i]]>>16)&0xff]--]=t1[i];
    fd(i,n,1)t1[r4[(a[t2[i]]>>24)&0xff]--]=t2[i];
    fp(i,1,n)t1[i]=a[t1[i]];
    fp(i,1,n)a[i]=t1[i];
}
```

(待补)舞蹈链

动态二维数组

模板

```

template<class T>
class array2 //动态二维数组
{
private:
    vector<vector<T>> a;

public:
    void resize(int n, int m){a.resize(n+1,vector<T>(m+1));}
    //设置数组大小

    void release(){a.swap(*(new vector<vector<T>>));}
    //释放空间防止MLE

    vector<T> & operator [](int row){return a[row];}

};
```

二分答案

模板

```
int binary_search_result(int l, int r, checker check)
{
    int mid, ans = l;
    while (l <= r)
    {
        mid = (l + r) >> 1;
        if (check(mid))
            ans = mid, l = mid + 1;
        else r = mid - 1;
    }
    return ans;
}
```

三分

```
int n;
double l,r;
double a[MAXN];

double calc(double x)
{
    double ans=0;
    fp(i,0,n)ans+=a[i]*pow(x,i);
    return ans;
}
int work()
{
    cin >> n >> l >> r;
    fd(i,n,0)cin>>a[i];
    while(fabs(r-l)>eps)
    {
        double mid1=(l+r)/2;
        double mid2=(mid1+r)/2;
        if(calc(mid1)>calc(mid2)) r=mid2;
        else l=mid1;
    }
    cout << fixed << setprecision(5) << l << endl;
    return 0;
}
```

单调栈 最大子矩阵

模板

```
int n,m;
bool a[MAXN][MAXN],s[MAXN][MAXN];
int h[MAXN],l[MAXN],r[MAXN];
int ans1=0,ans2=0;

vector<pr> q;

void init(int i)
{
    fp(j,1,m)
    {
        if(a[i][j]) h[j]=0;
        else h[j]=(a[i][j]==a[i-1][j])?(h[j]+1):1;
    }
    fp(j,1,m)
    {
        pr cur={h[j],j};
        while(!q.empty()&&cur<q.back())
            r[q.back().se]=j,q.ppb();
        q.pb(cur);
    }
    for(auto t:q) r[t.se]=m+1;
    q.clear();
    fd(j,m,1)
    {
        pr cur={h[j],j};
        while(!q.empty()&&cur<q.back())
            l[q.back().se]=j,q.ppb();
        q.pb(cur);
    }
    for(auto t:q) l[t.se]=0;
    q.clear();
}

void solve()
{
    mst(h,0);
    fp(i,1,n)
    {
        init(i);
        fp(j,1,m)
        {
            int w=r[j]-l[j]-1;
            ans1=max(ans1,min(h[j],w)*min(h[j],w));
            ans2=max(ans2,w*h[j]);
        }
    }
}
```