CMakeLists(in Clion)

C#

```
add_definitions(-D LOCAL)
```

C#

```
#include<bits/stdc++.h>
#include<numeric>
using namespace std;
template<typename typC,typename typD> bool cmin(typC &x,const typD
&y) { if (y<x) { x=y; return 1; } return 0; }
template<typename typC,typename typD> bool cmax(typC &x,const typD
&y) { if (x<y) { x=y; return 1; } return 0; }
#define ll long long
#define pb emplace_back
#define fs first
#define sc second
#define mpi make_pair
#define re(a) {cout<<a<<endl;return;}
#define all(v) v.begin(),v.end()
//#define all(v, n) v.begin()+1,v.begin()+n+1
#define fr(i, a, n)  for(int i = a; i <= n; i++)
const int N = 2e5 + 7;
const int M = 1e18 + 7;
const ll inf = 1e10;
const int mod = 998244353;
//function<void(int,int)>dfs=[&](int x,int y)->void{};
//sort(last.begin(),last.begin()+n,[&](pair<ll,ll>x,pair<ll,ll>y){
//if(x.first==y.first)    return x.second<y.second;
//return x.first>y.first;
//});
void solve(){
    int n;  cin >> n;
}
signed main(){
#ifdef  LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif
    ios::sync_with_stdio(0);    cin.tie(0);
    int _ = 1;
    cin >> _;//处理多组样例T
    while (_--) solve();
}
```

快读

```C#
inline int read()
{
    int x=0,f=1;char ch=getchar();
    while (ch<'0'||ch>'9'){if (ch=='-') f=-1;ch=getchar();}
    while (ch>='0'&&ch<='9'){x=x*10+ch-48;ch=getchar();}
    return x*f;
}
```

int128

```C#
char buf[1<<23],*p1=buf,*p2=buf,obuf[1<<23],*O=obuf;
#define getchar() (p1==p2&&(p2=
(p1=buf)+fread(buf,1,1<<21,stdin),p1==p2)?EOF:*p1++)
inline int rd() {
    int x=0,f=1;char ch=getchar();
    while(!isdigit(ch)){if(ch=='-') f=-1;ch=getchar();}
    while(isdigit(ch)) x=x*10+(ch^48),ch=getchar();
    return x*f;
}

inline void write(__int128_t x) {
    if(x<0) putchar('-'),x=-x;
    if(x>9) write(x/10);
    putchar(x%10+'0');
}
```

预处理

阶乘预处理

阶乘以及其逆元预处理

```C#
ll fac[N + 3], invfac[N + 3];
long long binpow(long long a, long long b) {
    a %= mod;
    long long res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
ll C(ll n, ll m){
    ll ans = (fac[n] * invfac[m] % mod) * invfac[n - m] % mod;
    return ans;
```

```
}
fac[0] = 1;
for(int i = 1; i < N; i++)  fac[i] =(fac[i - 1] * i) % mod;
invfac[N - 1] = binpow(fac[N - 1], mod - 2);
for(int i = N - 2; i >= 0; i--) invfac[i] = invfac[i + 1] * (i + 1)
% mod;
```

逆元预处理

1-n分母逆元预处理

```C#
ll inv[N + 2];
inv[1] = 1;
for(ll i = 2; i <= n; i++){
    inv[i] = (m - m / i) * inv[m % i] % m;
}
```

二进制相关

```C#
//1.统计数字二进制中1的数量
int x;
x = __builtin_popcount(6);  //输出6所对应二进制中1的数量
//2.三进制数中最高位的2的幂次
x = __lg(7) // 输出2
x = __lg(8) // 输出3
```

打表

全排列打表

一个数组，要求输出它从小到大的全排列

```C#
vc<int>a(n);
fr(i, 0, n - 1) cin >> a[i];
sort(all(a));
fr(i, 0, n - 1) cout << a[i] <<" ";
cout << "\n";
do{fr(i, 0, n)  cout << a[i] <<" ";
   cout <<"\n";
}while(next_permutation(all(a)));
```

排列组合

从一个集合里取出它所有的非空子集

C#

```
fr(i, 0, (1 << n) - 1){
    fr(j, 0, n - 1)
        if(i & (1 << j))    cout << a[j] <<" ";
    cout <<"\n";
}
```

从一个集合里取出它所有大小为k的子集

C#

```
int num, kk;
fr(i, 0, (1 << n) - 1){
    num = 0,kk = i;
    while(kk){
        kk = kk & (kk - 1);
        num ++;
    }
    if(num == k){
        fr(j, 0, n - 1)
            if(i & (1 << j))    cout << a[j] <<" ";
        cout <<"\n";
    }
}
```

数据结构

ST表

C#

```
const int N = 1e5 + 10, M = 20;
int st[N][M];
int n, m;

int query(int l, int r)
{
    int x = log2(r - l + 1);
    return max(st[l][x], st[r - (1 << x) + 1][x]);
}

void solve()
{
    cin >> n >> m;
    fer(i, 1, n)cin >> st[i][0];
    fer(j, 1, 18)
    {
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
        {
            st[i][j] = max(st[i][j - 1], st[i + (1 << j - 1)][j -
1]);
        }
    }
    while (m--)
```

```
    {
        int l, r;
        cin >> l >> r;
        cout << query(l, r) << endl;
    }
}
```

并查集

C#

```
struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

树状数组

C#

```
const int N = 1e5 + 7;
ll Btree[N];
int lowbit(int x){
    return x & -x;
}
ll getsum(int x){
    int ans = 0;
    while(x > 0){
        ans += Btree[x];
        x -= lowbit(x);
    }
    return ans;
}
void add(int x, ll k){
    while(x <= n){
        Btree[x] += k;
        x += lowbit(x);
    }
}
//O(n)建树
void init() {
    for (int i = 1; i <= n; ++i) {
        Btree[i] += a[i];
        int j = i + lowbit(i);
        if (j <= n) Btree[j] += Btree[i];
    }
}
```

单调栈

C#

```
ll n;   cin>>n;
vector<ll>h(n+2);
for(int i=1;i<=n;i++)   cin>>h[i];
h[n+1]=0;
stack<ll>s;
ll maxans=0;
for(int i=1;i<=n+1;i++){
    while(!s.empty()){
        if(h[i]>=h[s.top()])    break;
        ll id=s.top();  s.pop();
        ll area=h[id]*(s.empty()?i-1:i-s.top()-1);
        maxans=max(maxans,area);
    }
    s.push(i);
}
cout<<maxans<<"\n";
```

单调队列

用途:滑动窗口求Min/Max

C#

```
struct Mx/Mi_deque{
    vector<pair<int, int>>dq;
    int s = 1, t = 0, sz, cnt = 0;
    Mx/Mi_deque(int n, int k){
        dq.resize(n + 2);
        sz = k;
    }
    void push(int x){
        while(s <= t && dq[t].first <= x)   t--;     //修改单调顺序,维护Max。
      //while(s <= t && dq[t].first >= x)   t--;     //修改单调顺序,维护Min。
        dq[++t].first = x;
        dq[t].second = cnt++;
        if(dq[t].second - dq[s].second + 1 > sz)   s++;
    }
    int front(){
        return dq[s].first;
    }
};
```

C#

```
int q[maxn], a[maxn];
int n, k;
void getmin() {
    // 得到这个队列里的最小值，直接找到最后的就行了
    int head = 0, tail = 0;
    for (int i = 1; i < k; i++) {
        while (head <= tail && a[q[tail]] >= a[i]) tail--;
        q[++tail] = i;
    }
    for (int i = k; i <= n; i++) {
        while (head <= tail && a[q[tail]] >= a[i]) tail--;
        q[++tail] = i;
        while (q[head] <= i - k) head++;
        printf("%d ", a[q[head]]);
    }
}

void getmax() {
    // 和上面同理
    int head = 0, tail = 0;
    for (int i = 1; i < k; i++) {
        while (head <= tail && a[q[tail]] <= a[i]) tail--;
        q[++tail] = i;
    }
    for (int i = k; i <= n; i++) {
        while (head <= tail && a[q[tail]] <= a[i]) tail--;
        q[++tail] = i;
        while (q[head] <= i - k) head++;
```

```
            printf("%d ", a[q[head]]);
        }
    }
```

线段树

1、区间加，区间SUM

C#

```
LL n, a[100005], d[270000], b[270000];

void build(LL l, LL r, LL p) {  // l:区间左端点 r:区间右端点 p:节点标号
  if (l == r) {
    d[p] = a[l];  // 将节点赋值
    return;
  }
  LL m = l + ((r - l) >> 1);
  build(l, m, p << 1), build(m + 1, r, (p << 1) | 1);  // 分别建立子
树
  d[p] = d[p << 1] + d[(p << 1) | 1];
}

void update(LL l, LL r, LL c, LL s, LL t, LL p) {
  if (l <= s && t <= r) {
    d[p] += (t - s + 1) * c, b[p] += c;  // 如果区间被包含了，直接得出
答案
    return;
  }
  LL m = s + ((t - s) >> 1);
  if (b[p])
    d[p << 1] += b[p] * (m - s + 1), d[(p << 1) | 1] += b[p] * (t -
m),
        b[p << 1] += b[p], b[(p << 1) | 1] += b[p];
  b[p] = 0;
  if (l <= m)
    update(l, r, c, s, m, p << 1);  // 本行和下面的一行用来更新p*2和
p*2+1的节点
  if (r > m) update(l, r, c, m + 1, t, (p << 1) | 1);
  d[p] = d[p << 1] + d[(p << 1) | 1];  // 计算该节点区间和
}

LL getsum(LL l, LL r, LL s, LL t, LL p) {
  if (l <= s && t <= r) return d[p];
  LL m = s + ((t - s) >> 1);
  if (b[p])
    d[p << 1] += b[p] * (m - s + 1), d[(p << 1) | 1] += b[p] * (t -
m),
        b[p << 1] += b[p], b[(p << 1) | 1] += b[p];
  b[p] = 0;
  LL sum = 0;
  if (l <= m)
    sum =
        getsum(l, r, s, m, p << 1);  // 本行和下面的一行用来更新p*2和
p*2+1的答案
  if (r > m) sum += getsum(l, r, m + 1, t, (p << 1) | 1);
  return sum;
```

```
}

int main() {
  std::ios::sync_with_stdio(0);
  LL q, i1, i2, i3, i4;
  std::cin >> n >> q;
  for (LL i = 1; i <= n; i++) std::cin >> a[i];
  build(1, n, 1);
  while (q--) {
    std::cin >> i1 >> i2 >> i3;
    if (i1 == 2)
      std::cout << getsum(i2, i3, 1, n, 1) << std::endl;  // 直接调用
操作函数
    else
      std::cin >> i4, update(i2, i3, i4, 1, n, 1);
  }
  return 0;
}
```

2、区间加乘区间SUM

C#

```
#define ll long long
ll read() {
    ll w = 1, q = 0;
    char ch = ' ';
    while (ch != '-' && (ch < '0' || ch > '9')) ch = getchar();
    if (ch == '-') w = -1, ch = getchar();
    while (ch >= '0' && ch <= '9') q = (ll) q * 10 + ch - '0', ch =
getchar();
    return (ll) w * q;
}

int n, m;
ll mod;
ll a[100005], sum[400005], mul[400005], laz[400005];

void up(int i) { sum[i] = (sum[(i << 1)] + sum[(i << 1) | 1]) % mod;
}

void pd(int i, int s, int t) {
    int l = (i << 1), r = (i << 1) | 1, mid = (s + t) >> 1;
    if (mul[i] != 1) {  // 懒标记传递，两个懒标记
        mul[l] *= mul[i];   mul[l] %= mod;  mul[r] *= mul[i];
mul[r] %= mod;
        laz[l] *= mul[i];   laz[l] %= mod;  laz[r] *= mul[i];
laz[r] %= mod;
        sum[l] *= mul[i];   sum[l] %= mod;  sum[r] *= mul[i];
sum[r] %= mod;
        mul[i] = 1;
    }
    if (laz[i]) {  // 懒标记传递
        sum[l] += laz[i] * (mid - s + 1);   sum[l] %= mod;
        sum[r] += laz[i] * (t - mid);   sum[r] %= mod;
        laz[l] += laz[i];   laz[l] %= mod;  laz[r] += laz[i];
laz[r] %= mod;
```

```
        laz[i] = 0;
    }
    return;
}

void build(int s, int t, int i) {
    mul[i] = 1;
    if (s == t) {
        sum[i] = a[s];
        return;
    }
    int mid = s + ((t - s) >> 1);
    build(s, mid, i << 1);  // 建树
    build(mid + 1, t, (i << 1) | 1);
    up(i);
}

void chen(int l, int r, int s, int t, int i, ll z) {
    int mid = s + ((t - s) >> 1);
    if (l <= s && t <= r) {
        mul[i] *= z;    mul[i] %= mod;
        laz[i] *= z;    laz[i] %= mod;
        sum[i] *= z;    sum[i] %= mod;
        return;
    }
    pd(i, s, t);
    if (mid >= l) chen(l, r, s, mid, (i << 1), z);
    if (mid + 1 <= r) chen(l, r, mid + 1, t, (i << 1) | 1, z);
    up(i);
}

void add(int l, int r, int s, int t, int i, ll z) {
    int mid = s + ((t - s) >> 1);
    if (l <= s && t <= r) {
        sum[i] += z * (t - s + 1);  sum[i] %= mod;
        laz[i] += z;    laz[i] %= mod;
        return;
    }
    pd(i, s, t);
    if (mid >= l) add(l, r, s, mid, (i << 1), z);
    if (mid + 1 <= r) add(l, r, mid + 1, t, (i << 1) | 1, z);
    up(i);
}

ll getans(int l, int r, int s, int t,
          int i) {  // 得到答案，可以看下上面懒标记助于理解
    int mid = s + ((t - s) >> 1);
    ll tot = 0;
    if (l <= s && t <= r) return sum[i];
    pd(i, s, t);
    if (mid >= l) tot += getans(l, r, s, mid, (i << 1));
    tot %= mod;
    if (mid + 1 <= r) tot += getans(l, r, mid + 1, t, (i << 1) | 1);
    return tot % mod;
}

int main() {  // 读入
    int i, j, x, y, bh;
```

```
    ll z;
    cin >> n >> m;
    mod = read();
    for (i = 1; i <= n; i++) a[i] = read();
    build(1, n, 1);  // 建树
    for (i = 1; i <= m; i++) {
        bh = read();
        if (bh == 1) {
            cin >> x >> y >> z;
            chen(x, y, 1, n, 1, z);
        } else if (bh == 2) {
            cin >> x >> y >> z;
            add(x, y, 1, n, 1, z);
        } else if (bh == 3) {
            cin >> x >> y;
            printf("%lld\n", getans(x, y, 1, n, 1));
        }
    }
    return 0;
}
```

3、区间修改成一个定值

C#

```
int n, a[100005], d[270000], b[270000];

void build(int l, int r, int p) {  // 建树
  if (l == r) {
    d[p] = a[l];
    return;
  }
  int m = l + ((r - l) >> 1);
  build(l, m, p << 1), build(m + 1, r, (p << 1) | 1);
  d[p] = d[p << 1] + d[(p << 1) | 1];
}

void update(int l, int r, int c, int s, int t,
            int p) {  // 更新，可以参考前面两个例题
  if (l <= s && t <= r) {
    d[p] = (t - s + 1) * c, b[p] = c;
    return;
  }
  int m = s + ((t - s) >> 1);
  if (b[p]) {
    d[p << 1] = b[p] * (m - s + 1), d[(p << 1) | 1] = b[p] * (t -
m);
    b[p << 1] = b[(p << 1) | 1] = b[p];
    b[p] = 0;
  }
  if (l <= m) update(l, r, c, s, m, p << 1);
  if (r > m) update(l, r, c, m + 1, t, (p << 1) | 1);
  d[p] = d[p << 1] + d[(p << 1) | 1];
}

int getsum(int l, int r, int s, int t, int p) {  // 取得答案，和前面一
样
```

```
  if (l <= s && t <= r) return d[p];
  int m = s + ((t - s) >> 1);
  if (b[p]) {
    d[p << 1] = b[p] * (m - s + 1), d[(p << 1) | 1] = b[p] * (t -
m);
    b[p << 1] = b[(p << 1) | 1] = b[p];
    b[p] = 0;
  }
  int sum = 0;
  if (l <= m) sum = getsum(l, r, s, m, p << 1);
  if (r > m) sum += getsum(l, r, m + 1, t, (p << 1) | 1);
  return sum;
}

int main() {
  std::ios::sync_with_stdio(0);
  std::cin >> n;
  for (int i = 1; i <= n; i++) std::cin >> a[i];
  build(1, n, 1);
  int q, i1, i2, i3, i4;
  std::cin >> q;
  while (q--) {
    std::cin >> i1 >> i2 >> i3;
    if (i1 == 0)
      std::cout << getsum(i2, i3, 1, n, 1) << std::endl;
    else
      std::cin >> i4, update(i2, i3, i4, 1, n, 1);
  }
  return 0;
}
```

4.维护区间的最小值，及最小值的数量

C#

```
struct info{
    int cnt, minn;
};
struct node{
    int lazy, len;
    info val;
} seg[N << 2];
info operator+(const info &a, const info &b){
    info c;
    c.minn = min(a.minn, b.minn);
    c.cnt = 0;
    a.minn == c.minn ? c.cnt += a.cnt : c.cnt;
    b.minn == c.minn ? c.cnt += b.cnt : c.cnt;
    return c;
}
void settag(int id, int tag){
    seg[id].val.minn += tag;
    seg[id].lazy += tag;
}
void up(int id){
    seg[id].val = seg[id << 1].val + seg[id << 1 | 1].val;
}
```

```
void down(int id)
{
    if (seg[id].lazy == 0)
        return;
    settag(id << 1, seg[id].lazy);
    settag(id << 1 | 1, seg[id].lazy);
    seg[id].lazy = 0;
}
void build(int id, int l, int r){
    seg[id].len = r - l + 1;
    if(l == r){
        seg[id].val.minn = 0;
        seg[id].val.cnt = 1;
        seg[id].lazy = 0;
        return;
    }
    int mid = (l + r) >> 1;
    build(id << 1, l, mid);
    build(id << 1 | 1, mid + 1, r);
    up(id);
}
void modify(int id, int l, int r, int ql, int qr, int val){
    if(ql <= l && r <= qr){
        settag(id, val);
        return;
    }
    down(id);
    int mid = (l + r) >> 1;
    if(qr <= mid)
        modify(id << 1, l, mid, ql, qr, val);
    else if(ql > mid)
        modify(id << 1 | 1, mid + 1, r, ql, qr, val);
    else{
        modify(id << 1, l, mid, ql, qr, val);
        modify(id << 1 | 1, mid + 1, r, ql, qr, val);
    }
    up(id);
}
info query(int id, int l, int r, int ql, int qr)
{
    if (ql <= l && r <= qr)
    {
        return seg[id].val;
    }
    down(id);
    int mid = (l + r) >> 1;
    if (qr <= mid)
        return query(id << 1, l, mid, ql, qr);
    else if (ql > mid)
        return query(id << 1 | 1, mid + 1, r, ql, qr);
    else
        return query(id << 1, l, mid, ql, qr) + query(id << 1 | 1,
mid + 1, r, ql, qr);
}
void solve() {
    int n;  cin >> n;
    build(1, 1, n);
    int t;  cin >> t;
```

```
    while(t--){
        char q; cin >> q;
        if(q == 'q'){
            int l, r;   cin >> l >> r;
            info ans = query(1, 1, n, l, r);
            cout << ans.minn <<" "<< ans.cnt;
        }else{
            int l, r, add;  cin >> l >> r >> add;
            modify(1, 1, n, l, r, add);
        }
    }
}
```

数论

GCD

C#

```
// Version 1
int gcd(int a, int b) {
  if (b == 0) return a;
  return gcd(b, a % b);
}

// Version 2
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
```

快速幂

一般快速幂

C#

```
long long binpow(long long a, long long b, long long m) {
  a %= m;
  long long res = 1;
  while (b > 0) {
    if (b & 1) res = res * a % m;
    a = a * a % m;
    b >>= 1;
  }
  return res;
}
```

筛法

线性筛

使空间中每个合数都被标记一次，并且只被每个数的最小质因子标记。

C#

```
void init(int n) {
```

```
  for (int i = 2; i <= n; ++i) {
    if (!vis[i]) {
      pri[cnt++] = i;
    }
    for (int j = 0; j < cnt; ++j) {
      if (1ll * i * pri[j] > n) break;
      vis[i * pri[j]] = 1;
      if (i % pri[j] == 0) {
        // i % pri[j] == 0s
        // 换言之，i 之前被 pri[j] 筛过了
        // 由于 pri 里面质数是从小到大的，所以 i乘上其他的质数的结果一定会被
        // pri[j]的倍数筛掉，就不需要在这里先筛一次，所以这里直接 break
        // 掉就好了
        break;
      }
    }
  }
}
```

在vis中填充他们的最小质因数

```C#
ll pri[N], vis[N];
int cnt = 0;
void init(int n) {
    for (int i = 2; i <= n; ++i) {
        if (!vis[i]) {
            pri[cnt++] = i;
        }
        for (int j = 0; j < cnt; ++j) {
            if (1ll * i * pri[j] > n) break;
            vis[i * pri[j]] = pri[j];
            if (i % pri[j] == 0) {
                break;
            }
        }
    }
    fr(i, 1, n) if(!vis[i]) vis[i] = i;
}
```

线性基

重要性质

1、原序列任何数都可以通过线性基异或得到。

2、线性基内部任何数异或都不为0。

3、线性基个数唯一且最少。

插入

C#

```
inline void insert(ll x){
    for(int i = 52; i >= 0; i--){
        if(x >> i & 1){
            if(!p[i]){
                p[i] = x;
                break;
            }
            x ^= p[i];
        }
    }
}
```

求Max

C#

```
ll getmx(){
    ll ans = 0;
    for(int i = 52; i >= 0; i--)
        if((ans ^ p[i]) > ans)  ans ^= p[i];
    return ans;
}
```

求k小值

C#

```
inline void prework() { //预处理线性基p[i]，如果p[i]二进制第j位为1则异或p[j - 1]
    for(int i=1;i<=60;++i)
        for(int j=1;j<=i;++j)
            if(p[i]&(1LL<<j-1)) p[i]^=p[j-1];
}
inline ll getkth(int k) {//第k小则是ans异或线性基中每一位为1的元素
    if(k==1&&size<n) return 0;
    if(size<n) --k;
    //n表示序列长度，size表示线性基内部元素个数
    prework();
    ll ans=0;
    for(int i=0;i<=60;++i)
        if(p[i]) {
            if(k&1) ans^=p[i];
            k>>=1;
        }
    return ans;
}
```

图论以及树上问题

LCA

倍增

C#

```
/*
input
5 5 4   //5个点，5次LCA询问，4为树的根节点。
3 1
2 4
5 1
1 4
2 4
3 2
3 5
1 2
4 5
*/

vc<int> depth(N + 2),edge[N + 2];
vc<vc<int>>fa(N + 2,vc(22 , 0));
void solve() {
    int n, q ,root;  cin >> n >> q >> root;
    int u, v;
    fr(i, 2, n){
        cin >> u >> v;
        edge[u].pb(v);
        edge[v].pb(u);
    }
    vector<int>lg(n);
    fr(i, 1, n) lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
    function<void(int, int)>dfs = [&](int now,int fath)->void{
        fa[now][0] = fath;//第一个父亲节点是自己
        depth[now] = depth[fath] + 1;
        fr(i, 1, lg[depth[now]])    fa[now][i] = fa[fa[now][i - 1]]
[i - 1];
        for(auto i:edge[now])   if(i != fath) dfs(i, now);
    };
    function<int(int, int)>lca = [&](int x,int y)->int{
        if(depth[x] < depth[y]) swap(x, y);
        while(depth[x] > depth[y])  x = fa[x][lg[depth[x] -
depth[y]] - 1];
        if(x == y)  return x;
        for(int k = lg[depth[x]] - 1;k >= 0; k--)
            if(fa[x][k] != fa[y][k])    x = fa[x][k], y = fa[y][k];
        return fa[x][0];
    };
    dfs(root, 0);
    rep(q){
        cin >> u >> v;
        cout << lca(u, v) <<"\n";
    }
}
```

树剖写法

1.判断两点是否在同一条链上，是则能直接得到答案

2.否则，令深度较大的点变成它重链端的父节点，得到新的两点，重复以上步骤。

树剖步骤

定义siz[x] 为以x为根结点的子树节点个数

对x的每个子节点，找到其最大的节点y，使得siz[y] >= siz[z]。

C#

```
vc<int>dep(N + 2),edge[N + 2],fa(N + 2),son(N + 2),siz(N + 2), top(N
+ 2);
//son存最大的儿子子树
void solve() {
    int n, q ,root;  cin >> n >> q >> root;
    int u, v;
    fr(i, 2, n){
        cin >> u >> v;
        edge[u].pb(v);
        edge[v].pb(u);
    }
    function<void(int)>dfs1 = [&](int x)->void{
        siz[x] = 1;
        dep[x] = dep[fa[x]] + 1;
        for(auto i : edge[x]){
            if(i == fa[x]) continue;
            fa[i] = x;
            dfs1(i);
            siz[x] += siz[i];
            if(!son[x] || siz[son[x]] < siz[i]) son[x] = i;
        }
    };
    function<void(int, int)>dfs2 = [&](int x, int tv)->void{
        top[x] = tv;
        if(son[x])  dfs2(son[x], tv);
        for(auto i:edge[x]){
            if(i == fa[x] || i == son[x]) continue;
            dfs2(i,i);
        }
    };
    dfs1(root);
    dfs2(root, root);
    int ans;
    rep(q){
        cin >> u >> v;
        while(top[u] != top[v])
            dep[top[u]] >= dep[top[v]] ? u = fa[top[u]] : v =
fa[top[v]];
        ans = (dep[u] < dep[v] )? u : v;
        cout<<ans <<"\n";
    }
}
```

树的直径

DFS

选定一个点，进行dfs，找到最深的点，在最深的点dfs，最大路径便是直径

C#

```
void solve() {
    int n;  cin >> n;
    vc<pair<int,int>>edge[n + 2];//pair<v,w> u->v,value = w;
    vc<int>pre(n + 2);
    fr(i, 2, n){
         int v, w;   cin >> v >>w;
         edge[i].pb(mpi(v,w));
         edge[v].pb(mpi(i,w));
    }
    int mx = 1;
    function<void(int, int)>dfs=[&](int x,int fa){
        for(auto [v, w]: edge[x]){
            if(v == fa) continue;
            pre[v] = pre[x] + w;
            if(pre[v] > pre[mx])    mx = v;
            dfs(v,x);
        }
    };
    dfs(1, 0);
    fill(all(pre),0);
    dfs(mx, 0);
    cout << pre[mx] << "\n";
}
```

树形DP求

在任意节点跑树形DP维护这些节点最大的两条链的值，D = max(D, d[i].first + d[i].second)。

C#

```
void solve() {
    int n;  cin >> n;
    vc<pair<int, int>> d(n + 2);//pair<max_edge,nextmax_edge>;
    vc<pair<int, int>> edge[n + 2];//pair<v,w> u->v,value = w;
    fr(i, 2, n){
        int v, w;   cin >> v >>w;
        edge[i].pb(mpi(v,w));
        edge[v].pb(mpi(i,w));
    }
    int mx = 1;
    function<void(int,int)>dfs=[&](int x,int fa){
        d[x].first = d[x].second = 0;
        for(auto [v,w]: edge[x]){
            if(v == fa) continue;
            dfs(v, x);
            int now = d[v].first + w;
            if(now > d[x].first) d[x].second = d[x].first,
d[x].first = now;
            else if(now > d[x].second)  d[x].second = now;
```

```
        }
        if(d[x].first + d[x].second > d[mx].first + d[mx].second)
 mx = x;
    };
    dfs(1, 0);
    cout << mx <<"\n";
    cout << d[mx].first + d[mx].second <<"\n";
}
```

最长上升子序列问题(LIS)

策略：二分查找最大长度的最大值

C#

```
    int n;  cin >> n;
    vc<int>a(n + 2 ), g(n + 2, INF),dp(n + 2);
    // dp[i]代表以i结尾的最长上升子序列长度
    fr(i, 1, n){
        cin >> a[i];
        int k = lower_bound(g.begin() + 1, g.begin() + n + 1, a[i])
- g.begin();
        dp[i] = k;
        g[k] = a[i];
    }
```

最长公共子序列(LCS)

dp[i] [j] 代表a[1] - a[i] 和b[1] - b[j] 的LCS长度

dp[i] [j] = max(dp[i - 1] [j], dp[i] [j - 1]);

当a[i] = b[j] 时

dp[i] [j] = dp[i - 1] [j - 1] + 1

C#

```
fr(i, 1, n)
    fr(j, 1, m){
        dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        if(a[i]==b[j])
        dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
    }
```

变种

两个数组都是排列

A数组映射到B数组，然后转换为LIS 问题。

字符串

字符串hash

C#

```
#include <bits/stdc++.h>
using namespace std;
#define ull unsigned long long
const int N = 1e6 + 7;
ull ha[N], fac[N];
ull seed = 201326611;   //233 50331653
ull getStr(int l, int r){
    return ha[r] - ha[l - 1] * fac[r - l + 1];
}
int main(){
    // 预处理
    fac[0] = 1;
    for(int i = 1; i < N; i++)  fac[i] = fac[i - 1] * seed;
    string s;   cin >> s;
    int sz = s.size();
    s = " " + s;
    ha[1] = s[1];
    for(int i = 2; i <= sz; i++)    ha[i] = ha[i - 1] * seed + s[i];

}
```

字典树

1.该字典树必须确保之前有相同的字符串出现过，而不是找前缀。

C#

```
struct Trie {
    int nex[N][26], cnt;
    bool exist[N];  // 该结点结尾的字符串是否存在

    void insert(string s, int sz) {  // 插入字符串
        int p = 0;
        for (int i = 0; i < sz; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结
点
            p = nex[p][c];
        }
        exist[p] = 1;
    }

    bool find(string s, int sz) {  // 查找字符串
        int p = 0;
        for (int i = 0; i < sz; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) return 0;
            p = nex[p][c];
        }
        return exist[p];
```

```
    }
};
```

2 .查找字符串s在字典中属于多少个字符串的前缀。

C#

```
struct Trie {
    int nex[N][26], cnt;
    int exist[N];  // 该结点结尾的字符串是否存在

    void insert(string s, int sz) {  // 插入字符串
        int p = 0;
        for (int i = 0; i < sz; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结点
            exist[p]++;
            p = nex[p][c];
        }
        exist[p]++;
    }

    int find(string s, int sz) {  // 查找字符串
        int p = 0;
        for (int i = 0; i < sz; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) return 0;
            p = nex[p][c];
        }
        return exist[p];
    }
};
```