Chapter 3 Data Structures

Keshuai

2014-11-16

Copyright : 1.0

新增例题目录

目录

一、STL+基本数据结构

（一）常用 STL 语法

1、排序检索：

sort(begin, end, cmp);	根据 cmp 函数排序序列
next_permutation(begin, end, cmp);	获取全排列中当前序列的下一个序列
reverse(begin, end, cmp);	翻转整个序列
unique(begin,end);	将这个数组去重，需要先排序
lower_bound(begin, end, x);	返回序列中大于等于 x 最近的元素的指针
upper_bound(begin, end, x);	返回序列中大于 x 最近的元素的指针
swap(a,b);	交换 a，b 的值

2、队列+栈：

进行删除或者取首元素时需要考虑是否为空。

（1）队列(queue)

push();	将元素入队
pop();	队首元素出队
front();	取队首元素
size();	队列元素个数
empty();	判断队列是否为空

（2）栈(stack)

push();	将元素入栈
pop();	栈顶元素出栈
top();	取栈顶元素
size();	栈元素的个数
empty();	判断栈是否为空

（3）优先队列(priority_queue)

需要运算符重载 bool operator < (const Typ& u) const {},重载的是优先级，优先级大的优先出队。

push();	将元素入队
pop();	队首元素出队
front();	取队首元素
size();	队列元素个数
empty();	判断队列是否为空

（4）双端队列（deque）

双端队列的一个应用就是维护单调序列，可以用在一些高效算法的题目降低复杂度。

push_front();	向队列首部添加元素
push_back();	向队列尾部添加元素
pop_front();	队首元素出队
pop_back();	队尾元素出队
size();	队列元素个数
empty();	判断队列是否为空

3、容器：

Type::iterator 迭代器，支持自加自减，通过*或者->取对应的元素。

（1）vector:可变长度数组。

clear ( ) ;	清空数组
push_back();	在数组末端添加元素
size();	数组元素个数
pop_back();	删除数组末端的元素
erase(iter);	删除迭代器指向位置元素
begin();	返回首元素的地址
end();	返回尾元素的地址+1 后的地址

（2）set/multiset:自动排序集合（去重/不去重）

需要重载 bool operator < (const Type& u) const {}

clear();	清空 set
insert();	插入元素 x
count(x);	统计 x 元素的个数
size();	统计整个 set 中的元素个数
find(x);	返回元素 x 的地址
erase(iter);	删除 iter 地址的元素
erase(begin, end);	删除地址 begin 到 end 之间的元素
begin();	返回首元素的地址
end();	返回尾元素的地址+1 后的地址

注：如果 find(x) = end()，或者 count(x) = 0 的话，说明 set 中没有这个 x 元素。

（3）map/multiset:映射容器（去重/不去重）

需要重载 bool operator < (const Type& u) const {}，本质上是一棵红黑树。

iter->fitst	迭代器 iter 对应的 key 值
iter->second	迭代器 iter 对应的 val 值
map[x] = y	直接通过 key 值对 map 元素赋值
clear();	清空 map
count(x);	统计 x 元素的个数
size();	统计 map 中总元素的个数
erase(iter);	删除 iter 地址的元素
erase(begin, end);	删除地址 begin 到 end 之间的元素
begin();	返回首元素的地址
end();	返回尾元素的地址+1 后的地址

（4）string:字符串

length();	放回 string 的长度
+	拼接两个 string，可以拼接 char
substr(begin = 0, end = n);	返回 begin 到 end 组成的字符串
>、=、<	比较两个 string 的字典序大小
cout	输出用 cout

（5）bitset:二进制数

bitset<int>	初始化二进制数位的长度
count();	返回位为 1 的个数

size();	返回二进制数位的长度
flip();	01 置换，0 变 1，1 变 0
set();	全置为 1
set(pos);	将 pos 位置置为 1
clear();	全置为 0
operator[]	重载了[]操作
&、|、^	支持二进制数的基本运算
cout	输出用 cout

（二）高精度

1、C++：

```
/*********************************************************************************/
/**********************************Bign.cpp***************************************/
const int maxn = 10005;

struct bign {
    int len, num[maxn];

    bign () {
        len = 0;
        memset(num, 0, sizeof(num));
    }
    bign (int number) {*this = number;}
    bign (const char* number) {*this = number;}

    void delzero ();
    void Put ();

    void operator = (int number);
    void operator = (char* number);

    bool operator <  (const bign& b) const;
    bool operator >  (const bign& b) const { return b < *this; }
    bool operator <= (const bign& b) const { return !(b < *this); }
    bool operator >= (const bign& b) const { return !(*this < b); }
    bool operator != (const bign& b) const { return b < *this || *this < b;}
    bool operator == (const bign& b) const { return !(b != *this); }

    void operator ++ ();
    void operator -- ();
    bign operator + (const int& b);
    bign operator + (const bign& b);
    bign operator - (const int& b);
    bign operator - (const bign& b);
    bign operator * (const int& b);
    bign operator * (const bign& b);
    bign operator / (const int& b);
    //bign operator / (const bign& b);
    int operator % (const int& b);
};

/*Main*/

void bign::delzero () {
    while (len && num[len-1] == 0)
        len--;
    if (len == 0) {
        num[len++] = 0;
    }
}
void bign::Put () {
    for (int i = len-1; i >= 0; i--)
        printf("%d", num[i]);
}
void bign::operator = (char* number) {
    len = strlen (number);
    for (int i = 0; i < len; i++)
        num[i] = number[len-i-1] - '0';
    delzero ();
}
void bign::operator = (int number) {
    len = 0;
    while (number) {
        num[len++] = number%10;
```

```
            number /= 10;
        }
        delzero ();
    }
    bool bign::operator < (const bign& b) const {
        if (len != b.len)
            return len < b.len;
        for (int i = len-1; i >= 0; i--)
            if (num[i] != b.num[i])
                return num[i] < b.num[i];
        return false;
    }
    void bign::operator ++ () {
        int s = 1;
        for (int i = 0; i < len; i++) {
            s = s + num[i];
            num[i] = s % 10;
            s /= 10;
            if (!s) break;
        }
        while (s) {
            num[len++] = s%10;
            s /= 10;
        }
    }
    void bign::operator -- () {
        if (num[0] == 0 && len == 1) return;
        int s = -1;
        for (int i = 0; i < len; i++) {
            s = s + num[i];
            num[i] = (s + 10) % 10;
            if (s >= 0) break;
        }
        delzero ();
    }
    bign bign::operator + (const int& b) {
        bign a = b;
        return *this + a;
    }
    bign bign::operator + (const bign& b) {
        int bignSum = 0;
        bign ans;
        for (int i = 0; i < len || i < b.len; i++) {
            if (i < len) bignSum += num[i];
            if (i < b.len) bignSum += b.num[i];
            ans.num[ans.len++] = bignSum % 10;
            bignSum /= 10;
        }
        while (bignSum) {
            ans.num[ans.len++] = bignSum % 10;
            bignSum /= 10;
        }
        return ans;
    }
    bign bign::operator - (const int& b) {
        bign a = b;
        return *this - a;
    }
    bign bign::operator - (const bign& b) {
        int bignSub = 0;
        bign ans;
        for (int i = 0; i < len || i < b.len; i++) {
            bignSub += num[i];
            bignSub -= b.num[i];
            ans.num[ans.len++] = (bignSub + 10) % 10;
            if (bignSub < 0) bignSub = -1;
            else bignSub = 0;
```

```
        }
        ans.delzero ();
        return ans;
    }
    bign bign::operator * (const int& b) {
        int bignSum = 0;
        bign ans;
        ans.len = len;
        for (int i = 0; i < len; i++) {
            bignSum += num[i] * b;
            ans.num[i] = bignSum % 10;
            bignSum /= 10;
        }
        while (bignSum) {
            ans.num[ans.len++] = bignSum % 10;
            bignSum /= 10;
        }
        return ans;
    }
    bign bign::operator * (const bign& b) {
        bign ans;
        ans.len = 0;
        for (int i = 0; i < len; i++){
            int bignSum = 0;

            for (int j = 0; j < b.len; j++){
                bignSum += num[i] * b.num[j] + ans.num[i+j];
                ans.num[i+j] = bignSum % 10;
                bignSum /= 10;
            }
            ans.len = i + b.len;

            while (bignSum){
                ans.num[ans.len++] = bignSum % 10;
                bignSum /= 10;
            }
        }
        return ans;
    }
    bign bign::operator / (const int& b) {
        bign ans;
        int s = 0;
        for (int i = len-1; i >= 0; i--) {
            s = s * 10 + num[i];
            ans.num[i] = s/b;
            s %= b;
        }
        ans.len = len;
        ans.delzero ();
        return ans;
    }
    int bign::operator % (const int& b) {
        bign ans;
        int s = 0;
        for (int i = len-1; i >= 0; i--) {
            s = s * 10 + num[i];
            ans.num[i] = s/b;
            s %= b;
        }
        return s;
    }
/*********************************************************************************/
```

注意:除法和取模中，如果遇到值较大，运算过程中可能爆 int，那么就要用 long long 类型去计算。

2、java

（1）头文件

```
import java.util.*;
import java.math.*;
import java.io.*;
```

（2）基本框架

```
public class Main {
     //函数格式为 public static 类型 函数名() {}
     //全局变量为 public static 类型 变量名
     public static void main(String args[]) {
          Scanner cin = new Scanner(System.in);
     }

}
```

（3）读入输出

```
while (cin.hasNext()) {}        // 多组数据的读入格式
cin.nextInt()                   //读入整型数
cin.next()                      //读入字符串
int[] a;  a = new int[size];    //定义数组
System.out.println(String);     //带换行输出
System.out.print(String);       //不带换行输出
```

（4）大数类

高精度数字类型为　　BigInteger

高精度浮点数类型为　BigDecimal

add();	加法
subtract();	减法
multiply();	乘法
divide();	除法
mod();	取模
equals();	判相等
compareTo();	比较大小，大于返回正，等于返回 0，小于返回负
toString();	转换成字符串
toPlainString();	BigDecimal 保持高精度输出
stripTrailingZeros().toPlainString()	BigDecimal 保持高精度输出

（5）初始化

```
a = new BigType(Type);
a = BigType.valueOf(Type);
a = cin.next BigType();
```

3、分数

```
/*********************************************************************************/
/*********************************Fraction.cpp*********************************/
typedef long long type;

struct Fraction {
     type mem; // 分子;
     type den; // 分母;

     Fraction (type mem = 0, type den = 1);
     void operator = (type x) { this->set(x, 1); }
     Fraction operator * (const Fraction& u);
     Fraction operator / (const Fraction& u);
     Fraction operator + (const Fraction& u);
     Fraction operator - (const Fraction& u);

     void set(type mem, type den);
};

inline type gcd (type a, type b) {
     return b == 0 ? (a > 0 ? a : -a) : gcd(b, a % b);
}

inline type lcm (type a, type b) {
     return a / gcd(a, b) * b;
}
/*Code*/

Fraction::Fraction (type mem, type den) {
     this->set(mem, den);
}
Fraction Fraction::operator * (const Fraction& u) {
     type tmp_p = gcd(mem, u.den);
     type tmp_q = gcd(u.mem, den);
     return Fraction( (mem / tmp_p) * (u.mem / tmp_q), (den / tmp_q) * (u.den / tmp_p) );
}

Fraction Fraction::operator / (const Fraction& u) {
     type tmp_p = gcd(mem, u.mem);
     type tmp_q = gcd(den, u.den);
     return Fraction( (mem / tmp_p) * (u.den / tmp_q), (den / tmp_q) * (u.mem / tmp_p));
}

Fraction Fraction::operator + (const Fraction& u) {
     type tmp_l = lcm (den, u.den);
     return Fraction(tmp_l / den * mem + tmp_l / u.den * u.mem, tmp_l);
}

Fraction Fraction::operator - (const Fraction& u) {
     type tmp_l = lcm (den, u.den);
     return Fraction(tmp_l / den * mem - tmp_l / u.den * u.mem, tmp_l);
}

void Fraction::set (type mem, type den) {
     if (den == 0) {
          den = 1;
          mem = 0;
     }
     type tmp_d = gcd(mem, den);
     this->mem = mem / tmp_d;
     this->den = den / tmp_d;
 }
/*********************************************************************************/
```

（三）辅助

1、输入外挂：

```
/*******************************************************************************/
/*********************************qscanf.cpp************************************/
void scanf_(int &num) {
    char in;
    bool neg=false;

    while(((in=getchar()) > '9' || in<'0') && in!='-') ;

    if(in=='-') {
        neg=true;
        while((in=getchar()) >'9' || in<'0');
    }

    num=in-'0';

    while(in=getchar(), in>='0'&&in<='9')
        num *= 10,num += in-'0';

     if (neg)
          num = -num;
}
/*******************************************************************************/
```

2、手动括栈：

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

二、树形数据结构

（一）并查集（union-find）

f[i]表示 i 节点父亲节点，初始化 f[i]=i。

find(x);	查询 x 节点的根节点
union(u,v);	合并 u 和 v 所在的两棵子树

```
/*******************************************************************************/
/*****************************Union-find.cpp*********************************/
const int maxn = 1e5 + 5;

/*************递归版****************/
int f[maxn];
void UF_init(int n) {
     for (int i = 0; i <= n; i++)
          f[i] = i;
}
int UF_find(int x) {
     return x == f[x] ? x : f[x] = UF_find(f[x]);
}
void UF_union(int u, int v) {
     int fu = UF_find(u);
     int fv = UF_find(v);
     if (fu != fv)
          f[fu] = fv;
}

/*************加权版****************/
int f[maxn], r[maxn], relat = 3;
void UF_init(int n) {
     for (int i = 0; i <= n; i++) {
          f[i] = 1;
          r[i] = 0;
     }
}
int UF_find(int x) {
     if (f[x] == x)
          return x;
     int fx = UF_find(x);
     // According Problem;
     r[x] = (r[x] + r[fx]) % relat;
     return f[x] = fx;
}
bool UF_union(int u, int v, int rel) {
     int fu = UF_find(u);
     int fv = UF_find(v);

     if (fu != fv) {
          // According Problem;
          r[fu] = ((r[v] + rel - r[u]) % relat + relat) % relat;
          f[fu] = fv;
     } else
          return ((f[u] - f[v]) % relat + relat) % relat == rel;
     return true;
}
/*************按秩合并****************/
**按秩合并**为并查集的一种优化，在合并两个集合时，将集合元素个数少的合并到集合个数多的。root 为 1 时，该节点为根节点，此时 parent 记录的为集合元素的个数了；root 为 0 时，该节点为非根节点，parent 记录的为它的父亲节点。
/***********************************/
```

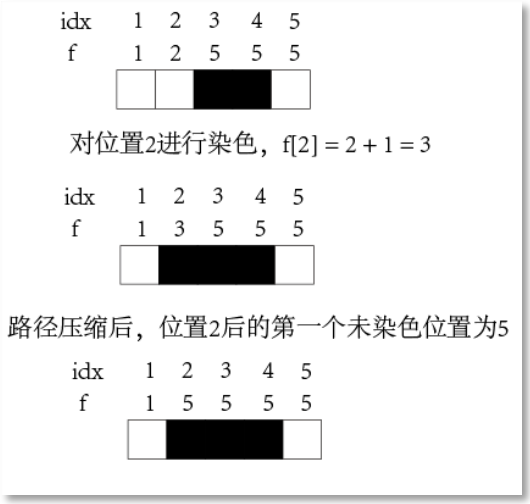
```
struct UFSet {
     int parent, root;
}f[maxn];

void UF_init(int n) {
     for (int i = 0; i <= n; i++) {
          f[i].parent = 1;
          f[i].root = 1;
     }
}

int UF_find(int x) {
     int p, q, tmp;
     p = q = x;
     while (!f[p].root)
          p = f[p].parent;
     while (q != p) {
          tmp = f[q].parent;
          f[q].parent = p;
          q = tmp;
     }
     return p;
}

void UF_union(int u, int v) {
     int fu = UF_find(u);
     int fv = UF_find(v);
     if  (f[fu].parent < f[fv].parent)
          swap(fu, fv);
     f[fu].parent += f[fv].parent;
     f[fv].parent = fu;
     f[fv].root = 0;
}
/*****************************************************************************/
```

应用：

（1）　判断图是否联通

（2）　确定联通分量个数（每进行一次 Union 联通分量减少 1）

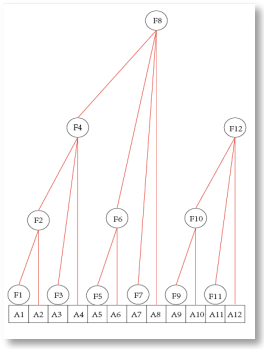
（3）　染色问题（可行域查找优化）



（二）树状数组(fenwick):动态连续和查询问题。

fenwick 数组下标从 1 开始，如果需要维护 0 位置的值，需将整体下标+1。

add(x,d);	对序列中第 x 个元素增加 d
sum(x);	计算从第一个元素到第 x 个元素的区间和
find(x);	解决第 K 大元素问题时查询第 x 大的元素值
lowbit(x);	取 x 对应二进制下的最后一位



lowbit 函数

38288 = 1001010110010000

& -38288 = 0110101001110000

0000000000010000

```
/*******************************************************************************/
/*****************************Fenwick-Tree.cpp*********************************/
#define lowbit(x) ((x)&(-x))
const int maxn = 1e5;

int fenw[maxn+5];

void fenwick_add (int x, int v) {
     while (x <= maxn) {
          fenw[x] += v;
          x += lowbit(x);
     }
}
int fenwick_sum (int x) {
     int ret = 0;
     while (x) {
          ret += fenw[x];
          x -= lowbit(x);
     }
     return ret;
}
int fenwick_find (int x) {
      int p = 0, ret = 0;
      for (int i = 20; i >= 0; i--) {
           p += (1<<i);
           if (p > maxn || ret + fenw[p] >= x)
                p -= (1<<i);
           else
                ret += fenw[p];
      }
      return p + 1;
}
/*******************************************************************************/
```

应用：

（1）　单点修改区间查询：

add(x, v); sum(r)-sum(l);

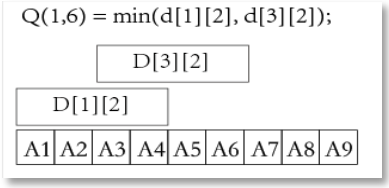
（2）　区间修改单点查询：

add(l,v),add(r+1,-v); sum(x);

（三）静态区间最值查询(RMQ)：范围最值问题。

预处理 d[i][j]数组，加速区间最值查询的速度，不支持动态修改。d[i][j]表示从 i 到 i+2^j 区间内最值。

init();	初始化 d 数组
query(L,R);	查询区间 L，R 的最值

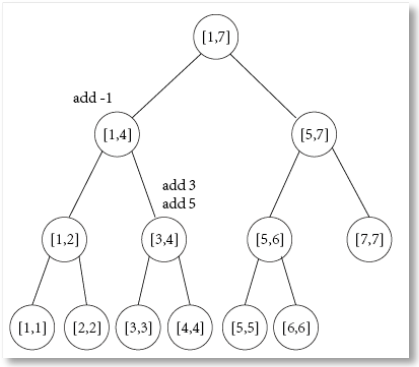



```
/*******************************************************************************/
/*****************************RMQ.cpp*******************************************/
const int maxn = 1e5;
const int maxr = 20;
/*************一维*****************/
int d[maxn][maxr];
void rmq_init (const vector<int>& arr) {
     int n = arr.size();
     for (int i = 0; i < n; i++)
          d[i][0] = arr[i];
     for (int j = 1; (1<<j) <= n; j++) {
          for (int i = 0; i + (1<<j) - 1 < n; i++)
               d[i][j] = min(d[i][j-1], d[i + (1<<(j-1))][j-1]);
     }
}
int rmq_query (int l, int r) {
     int k = 0;
     while ((1<<(k+1)) <= r - l + 1)
          k++;
     return min(d[l][k], d[r-(1<<k)+1][k]);
}
/*************二维*****************/
int N, M, Q, g[maxn][maxn], dp[maxn][maxn][9][9];
void rmq_init(int n, int m) {
     for (int i = 1; i <= n; i++) {
          for (int j = 1; j <= m; j++)
               dp[i][j][0][0] = g[i][j];
     }
     for (int x = 0; (1<<x) <= n; x++)
          for (int y = 0; (1<<y) <= m; y++)
               if (x + y)
                    for (int i = 1; i + (1<<x) - 1 <= n; i++)
                         for (int j = 1; j + (1<<y) - 1 <= m; j++) {
                              if (x)
                                   dp[i][j][x][y] = max(dp[i][j][x-1][y], dp[i+(1<<(x-1))][j][x-1][y]);
                              else
                                   dp[i][j][x][y] = max(dp[i][j][x][y-1], dp[i][j+(1<<(y-1))][x][y-1]);
                         }
}
int rmq_query(int x1, int y1, int x2, int y2) {
     int x = 0, y = 0;
     while ((1<<(x+1)) <= x2 - x1 + 1) x++;
     while ((1<<(y+1)) <= y2 - y1 + 1) y++;
     x2 = x2 - (1<<x) + 1;
     y2 = y2 - (1<<y) + 1;

     return max( max(dp[x1][y1][x][y], dp[x2][y1][x][y]), max(dp[x1][y2][x][y], dp[x2][y2][x][y]));
}
/*******************************************************************************/
```

（四）线段树：动态区间修改查询。

动态维护区间的修改和查询。节点的权值可以直接由左右孩子节点的权值计算得到，延迟操作时可以直接计算出节点的权值。线段树的节点个数需要多开 4 倍，防止数组越界。

build(u,l,r);	以 u 为根节点，对区间[l,r]建树
query(u,l,r);	查询区间[l，r]的值
insert(u,l,r,v);	对区间[l,r]上的每个元素增加 v
pushup(u);	更新 u 节点的权值
pushdown(u);	将 u 节点的延迟操作向下延迟一层




```
/*******************************************************************************/
/***************************Segment-Tree.cpp*******************************/
const int maxn = 1e5 + 5;
#define lson(x) ((x)<<1)
#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2], nd[maxn << 2], ad[maxn << 2];

inline void maintain (int u, int w) {
        // According Problem;
        nd[u] += ad[u] * (rc[u] - lc[u] + 1);
}
void pushup (int u) {
        // According Problem;
        nd[u] = nd[lson(u)] + nd[rson(u)];
}
void pushdown (int u) {
        // According Problem;
        if (ad[u]) {
              maintain(lson(u), ad[u]);
              maintain(rson(u), ad[u]);
              ad[u] = 0;
        }
}
void segtree_build (int u, int l, int r) {
        lc[u] = l;
        rc[u] = r;
        nd[u] = ad[u] = 0;
        if (l == r) {
         // According Problem;
         return;
        }
        int mid = (l + r) / 2;
        segtree_build(lson(u), l, mid);
        segtree_build(rson(u), mid + 1, r);
        pushup(u);
}
```

```
void segtree_modify (int u, int l, int r, int w) {
          if (l <= lc[u] && rc[u] <= r) {
                // According Problem;
                maintain(u, w);
                return ;
          }

          pushdown(u);
          int mid = (lc[u] + rc[u]) >> 1;
          if (l <= mid)
                segtree_modify(lson(u), l, r, w);
          if (r > mid)
                segtree_modify(rson(u), l, r, w);
          pushup(u);
}

int segtree_query (int u, int l, int r) {
          if (l <= lc[u] && rc[u] <= r)
           return nd[u];

          pushdown(u);
          int mid = (lc[u] + rc[u]) >> 1, ret = 0;
          if (l <= mid)
                ret += segtree_query(lson(u), l, r);
          if (r > mid)
                ret += segtree_query(rson(u), l, r);
          pushup(u);
          return ret;
}
/*****************************************************************************/
```

应用：

（1） 区间合并：一般解决类似一个区间有多少块连续 1，连续 1 的最大长度等问题。注：L 表示区间左端连续 1 的个数，R 表示区间右端连续 1 的个数，C 表示区间上连续有多少段连续的 1，S 表示区间上最长连续 1 的长度。

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

void pushup(int u) {
     C[u] = C[lson(u)] + C[rson(u)] + (R[lson(u)] && L[rson(u)] ? -1 : 0);
     S[u] = max(max(S[lson(u)], S[rson(u)]), R[lson(u)] + L[rson(u)]);
     L[u] = L[lson(u)] + (L[lson(u)] == rc[lson(u)] - lc[lson(u)] + 1 ? L[rson(u)] : 0);
     R[u] = R[rson(u)] + (R[rson(u)] == rc[rson(u)] - lc[rson(u)] + 1 ? R[lson(u)] : 0);
}

/**************区间上多少段连续的 1***************/
int query(int u, int l, int r) {
     if (l <= lc[u] && rc[u] <= r)
          return C[u];

     pushdown(u);
     int mid = (lc[u] + rc[u]) >> 1, ret;
     if (r <= mid)
          ret = query(lson(u), l, r);
     else if (l > mid)
          ret = query(rson(u), l, r);
     else
          ret = query(lson(u), l, r) + query(rson(u), l, r) - (R[lson(u)] && L[rson(u)] ? -1 : 0);
     pushup(u);
     return ret;
}
```

```
/**************区间上最长连续 1 的长度***************/
int query (int u, int l, int r) {
     if (l <= lc[u] && rc[u] <= r)
          return S[u];
     pushdown(u);
     int mid = (lc[u] + rc[u]) >> 1, ret;
     if (r <= mid)
          ret = query(lson(u), l, r);
     else if (l > mid)
          ret = query(rson(u), l, r);
     else {
          int ll = query(lson(u), l, r);
          int rr = query(rson(u), l, r);

          int a = min(L[rson(u)], r - mid);
          int b = min(R[lson(u)], mid - l + 1);

          ret = max(max(ll, rr), a + b);
     }
     pushup(u);
     return ret;
}

/**************第 K 段连续 1 的区间*****************/
pii query(int u, int k) {
     if (k > C[u])
          return make_pair(0, 0);

     pushdown(u);
     pii ret;
     int mid = (lc[u] + rc[u]) >> 1;

     if (C[lson(u)] == k && R[lson(u)])
          ret = make_pair(mid - R[lson(u)] + 1, mid + L[rson(u)]);
     else if (C[lson(u)] <= k)
          ret = query(lson(u), k);
     else
          ret = query(rson(u), k - C[lson(u)] + (R[lson(u)] && L[rson(u)] ? 1 : 0));
     pushup(u);
     return ret;
}

/**************满足长度 k 的最左区间*****************/
int query (int u, int k) {
    if (S[u] < k)
        return 0;

    pushdown(u);
    int ret;

    if (S[lson(u)] >= k)
        ret = query(lson(u), k);
    else if (R[lson(u)] + L[rson(u)] >= k) {
        int mid = (lc[u] + rc[u]) / 2;
        ret = mid - R[lson(u)] + 1;
    } else
        ret = query(rson(u), k);
    pushup(u);
    return ret;
}
/*******************************************************************************/
```

例题：Codeforces 484E（可持久化线段处理区间合并）

题目大意：给定给一个序列，每个位置有一个值，表示高度，现在有若干查询，每次查询 l，r，w，表示在区间 l，r 中，连续最长长度大于 w 的最大高度为多少。

解题思路：将高度按照从大到小的顺序排序，然后每次插入一个位置，线段树维护最长连续区间，因为插入是按照从大到小的顺序，所以每次的线段树中的连续最大长度都是满足高度大于等于当前新插入的 height 值。对于每次查询，二分高度，因为高度肯定是在已有的高度中，所以只接二分下标即可。

/*****************************CF484E.cpp*********************************/

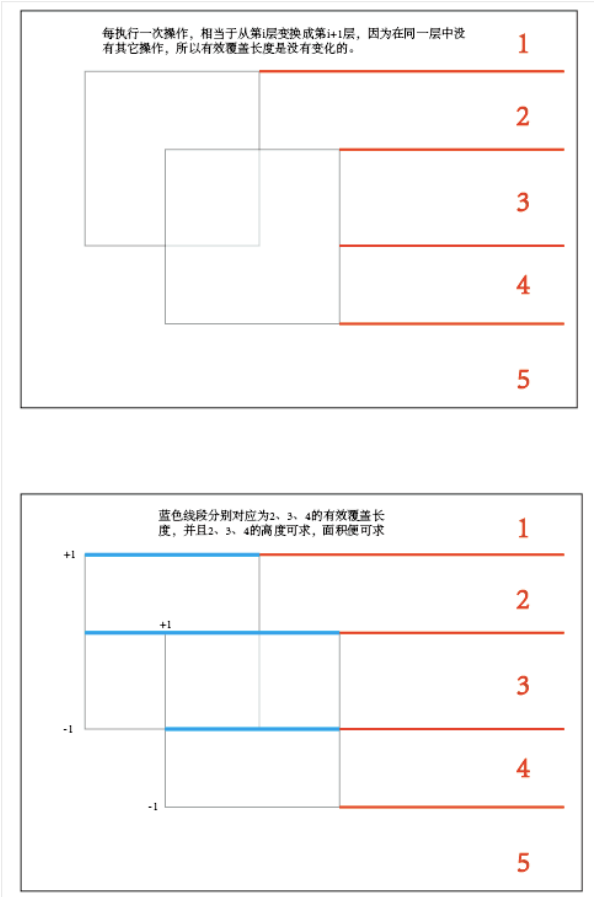
```
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>

using namespace std;

const int maxn = 1e6 + 5;
typedef pair<int,int> pii;
struct Node {
    int lc, rc, lp, rp, L, R, S;
    int length() {
        return rp - lp + 1;
    }
}nd[maxn << 2];

int N, sz, root[maxn];
pii blo[maxn];

inline int newNode() {
    return sz++;
}

inline void pushup(int u) {
    int lcid = nd[u].lc, rcid = nd[u].rc;
    nd[u].L = nd[lcid].L + (nd[lcid].L == nd[lcid].length() ? nd[rcid].L : 0);
    nd[u].R = nd[rcid].R + (nd[rcid].R == nd[rcid].length() ? nd[lcid].R : 0);
    nd[u].S = max(nd[lcid].R + nd[rcid].L, max(nd[lcid].S, nd[rcid].S));
}

inline Node merge(Node a, Node b) {
    Node u;
    u.lp = a.lp; u.rp = b.rp;
    u.L = a.L + (a.L == a.length() ? b.L : 0);
    u.R = b.R + (b.R == b.length() ? a.R : 0);
    u.S = max(a.R + b.L, max(a.S, b.S));
    return u;
}

void build(int& u, int l, int r) {
    if (u == 0) u = newNode();
    nd[u] = (Node){0, 0, l, r, 0, 0, 0};

    if (l == r)
        return;
    int mid = (l + r) >> 1;
    build(nd[u].lc, l, mid);
    build(nd[u].rc, mid +1, r);
    pushup(u);
}
int insert(int u, int x) {
    int k = newNode();
    nd[k] = nd[u];

    if (nd[k].lp == x && x == nd[k].rp) {
        nd[k].S = nd[k].L = nd[k].R = 1;
        return k;
    }
```

```
    int mid = (nd[k].lp + nd[k].rp) >> 1;
    if (x <= mid)
        nd[k].lc = insert(nd[k].lc, x);
    else
        nd[k].rc = insert(nd[k].rc, x);
    pushup(k);
    return k;
}

Node query(int u, int l, int r) {
    if (l <= nd[u].lp && nd[u].rp <= r)
        return nd[u];

    int mid = (nd[u].lp + nd[u].rp) >> 1;
    if (r <= mid)
        return query(nd[u].lc, l, r);
    else if (l > mid)
        return query(nd[u].rc, l, r);
    else {
        Node ll = query(nd[u].lc, l, r);
        Node rr = query(nd[u].rc, l, r);
        return merge(ll, rr);
    }
}
inline bool cmp (const pii& a, const pii& b) {
    return a.first > b.first;
}
void init () {
    sz = 1;
    scanf("%d", &N);

    for (int i = 1; i <= N; i++) {
        scanf("%d", &blo[i].first);
        blo[i].second = i;
    }
    sort(blo + 1, blo + 1 + N, cmp);
    build(root[0], 1, N);
    for (int i = 1; i <= N; i++)
        root[i] = insert(root[i-1], blo[i].second);
}

int main () {
    init();

    int q, l, r, w;
    scanf("%d", &q);
    while (q--) {
        scanf("%d%d%d", &l, &r, &w);
        int L = 1, R = N;
        while (L < R) {
            int mid = (L + R) >> 1;
            if (query(root[mid], l, r).S >= w)
                R = mid;
            else
                L = mid + 1;
        }
        printf("%d\n", blo[L].first);
    }
    return 0;
}
/**************
input
5
1 2 2 3 3
3
2 5 3
```

```
2 5 2
1 5 5
output
2
3
1

input
1
1000000000
1
1 1 1
output
1000000000
***************/
/*******************************************************************************/
```

（2）　扫描线：进行区间修改，但是没有向下更新。一般问题就是求矩形面积的并，周长的并，或者是覆盖 k 次的面积。

注：因为横坐标可能会很大，所以很多时候要离散化处理；线段树的维护中不能有 pushdown 操作，因为每一段区间的加都对应着减，如果将先前的加操作向下更新，那么后面的减操作将导致有负数的存在。



例题：Poj 2482（统计矩形内最大点个数问题转化）

题目大意：平面上有 N 个星星，问一个 W*H 的矩形最多能括进多少个星星。

解题思路：只要以每个点为左上角，建立矩形，这个矩形即为框框左下角放的位置可以括到该点，那么 N 个星星就有 N 个矩形，扫描线处理哪些位置覆盖次数最多。

```
/*****************************poj2482.cpp*********************************/
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>

using namespace std;

typedef long long ll;
const int maxn = 40000;

#define lson(x) ((x)<<1)
#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2];
ll v[maxn << 2], s[maxn << 2];

inline void pushup (int u) {
    s[u] = max(s[lson(u)], s[rson(u)]) + v[u];
}

inline void maintain (int u, int d) {
    v[u] += d;
    pushup(u);
}

void build (int u, int l, int r) {
    lc[u] = l;
    rc[u] = r;
    v[u] = s[u] = 0;

    if (l == r)
        return;

    int mid = (l + r) / 2;
    build(lson(u), l, mid);
    build(rson(u), mid + 1, r);
    pushup(u);
}

void modify (int u, int l, int r, int d) {
    if (l <= lc[u] && rc[u] <= r) {
        maintain(u, d);
        return;
    }

    int mid = (lc[u] + rc[u]) / 2;
    if (l <= mid)
        modify(lson(u), l, r, d);
    if (r > mid)
        modify(rson(u), l, r, d);
    pushup(u);
}

struct Seg {
    ll x, l, r, d;
    Seg (ll x = 0, ll l = 0, ll r = 0, ll d = 0) {
        this->x = x;
        this->l = l;
        this->r = r;
        this->d = d;
    }
    friend bool operator < (const Seg& a, const Seg& b) {
        return a.x < b.x;
    }
};
```

```
int N, W, H;
vector<ll> pos;
vector<Seg> vec;

inline int find (ll k) {
    return lower_bound(pos.begin(), pos.end(), k) - pos.begin();
}

void init () {
    ll x, y, d;

    pos.clear();
    vec.clear();
    for (int i = 0; i < N; i++) {
        scanf("%lld%lld%lld", &x, &y, &d);
        pos.push_back(y - H);
        pos.push_back(y);
        vec.push_back(Seg(x - W, y - H, y, d));
        vec.push_back(Seg(x, y - H, y, -d));
    }
    sort(pos.begin(), pos.end());
    sort(vec.begin(), vec.end());
}

ll solve () {
    ll ret = 0;
    build (1, 0, pos.size());

    for (int i = 0; i < vec.size(); i++) {
        modify(1, find(vec[i].l), find(vec[i].r) - 1, vec[i].d);
        ret = max(ret, s[1]);
    }
    return ret;
}
int main () {
    while (scanf("%d%d%d", &N, &W, &H) == 3) {
        init();
        printf("%lld\n", solve());
    }
    return 0;
}
/******************
input
3 5 4
1 2 3
2 3 2
6 3 1
3 5 4
1 2 3
2 3 2
5 3 1

output
5
6
******************/
/*******************************************************************************/
```

例题：Uva 11983（矩形覆盖 K 次面积问题）

题目大意：给定 n 个矩形，问说有多少区域的面积被覆盖 k 次以上。

解题思路：将每个矩形差分成两条线段，一段为添加覆盖值 1，一段为减少覆盖值 1，同时记录两段的高度（横坐标）。然后对纵坐标离散化建立线段树，然后对线段按照高度排序，维护整段区间中覆盖度大于 K 的长度，乘上高度上的范围即可。

```
/*****************************uva11983.cpp*********************************/
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>

using namespace std;
const int maxn = 30005;

#define lson(x) ((x)<<1)
#define rson(x) (((x)<<1)+1)
typedef long long ll;

int N, K, R;
vector<int> bins, arr;

struct Segment {
    int l, r, h, v;
    void set (int l, int r, int h, int v) {
        this->l = l;
        this->r = r;
        this->h = h;
        this->v = v;
    }
}seg[maxn*2];

struct Node {
    int l, r, add;
    int c[12];
    void set (int l, int r, int add) {
        this->l = l;
        this->r = r;
        this->add = add;
        memset(c, 0, sizeof(c));
    }
}node[maxn*24];

inline bool cmp (const Segment& a, const Segment& b) {
    return a.h < b.h;
}

inline int search (int v) {
    return lower_bound(bins.begin(), bins.end(), v) - bins.begin();
}

void pushup (int u) {
    memset(node[u].c, 0, sizeof(node[u].c));

    if (node[u].l == node[u].r) {
        int x = node[u].l;
        node[u].c[min(K, node[u].add)] = bins[x+1] - bins[x];
    } else {
        for (int i = 0; i <= K; i++) {
            int t = min(K, i + node[u].add);
            node[u].c[t] += node[lson(u)].c[i] + node[rson(u)].c[i];
        }
    }
}

void build_segTree (int u, int l, int r) {
    node[u].set(l, r, 0);

    if (l == r) {
        pushup(u);
        return;
```

```
    }

    int mid = (l + r) / 2;
    build_segTree(lson(u), l, mid);
    build_segTree(rson(u), mid + 1, r);
    pushup(u);
}

void insert_segTree (int u, int l, int r, int v) {
    if (l <= node[u].l && node[u].r <= r) {
        node[u].add += v;
        pushup(u);
        return;
    }

    int mid = (node[u].l + node[u].r) / 2;
    if (l <= mid)
        insert_segTree(lson(u), l, r, v);
    if (r > mid)
        insert_segTree(rson(u), l, r, v);
    pushup(u);
}

int query_segTree (int u, int l, int r) {
    if (l <= node[u].l && node[u].r <= r)
        return node[u].c[K];

    int mid = (node[u].l + node[u].r) / 2;
    int ret = 0;
    if (l <= mid)
        ret += query_segTree(lson(u), l, r);
    if (r > mid)
        ret += query_segTree(rson(u), l, r);
    return ret;
}
void add_hash (int x) {
    if (x)
        arr.push_back(x-1);
    arr.push_back(x);
    arr.push_back(x+1);
}

void init () {
    int x1, y1, x2, y2;
    arr.clear();
    scanf("%d%d", &N, &K);
    for (int i = 0; i < N; i++) {
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        seg[lson(i)].set(x1, x2, y1, 1);
        seg[rson(i)].set(x1, x2, y2+1, -1);

        add_hash(x1);
        add_hash(x2);
    }
    sort(arr.begin(), arr.end());

    bins.clear();
    bins.push_back(arr[0]);
    for (int i = 1; i < arr.size(); i++) {
        if (arr[i] != arr[i-1])
            bins.push_back(arr[i]);
    }

    R = bins.size() - 2;

    build_segTree (1, 0, R);
}
```

```
ll solve () {
    sort(seg, seg + 2 * N, cmp);

    ll ret = 0;
    for (int i = 0; i < 2 * N - 1; i++) {
        insert_segTree(1, search(seg[i].l), search(seg[i].r), seg[i].v);
        ll tmp = query_segTree(1, 0, R);
        ret += tmp * (seg[i+1].h - seg[i].h);
    }
    return ret;
}

int main () {
    int cas;
    scanf("%d", &cas);
    for (int kcas = 1; kcas <= cas; kcas++) {
        init();
        printf("Case %d: %lld\n", kcas, solve());
    }
    return 0;
}

/*******************
input
2
2 1
0 0 4 4
1 1 2 5
2 2
0 0 4 4
1 1 2 5

output
Case 1: 27
Case 2: 8
*******************/
/*******************************************************************************/
```

（3）　其它题型：

例题：Uva 12436（维护等差数列和问题）

题目大意：四种操作。

```
void A( int st, int nd ) {
    for( int i = st; i <= nd; i++ ) data[i] = data[i] + (i - st + 1);
}
void B( int st, int nd ) {
    for( int i = st; i <= nd; i++ ) data[i] = data[i] + (nd - i + 1);
}
void C( int st, int nd, int x ) {
    for( int i = st; i <= nd; i++ ) data[i] = x;
}
long long S( int st, int nd ) {
    long long res = 0;
    for( int i = st; i <= nd; i++ ) res += data[i];
    return res;
}
```

解题思路：即用线段树维护一个等差数列，因为一个等差加上一个等差还是一个等差数列，所以对于每个节点记录区间左端的值，也就是首项，以及公差即可。因为还有一个 S 操作，所以要开一个标记记录区间值是否相同。

```
/*******************************uva12436.cpp*********************************/
#include <cstdio>
#include <cstring>
#include <algorithm>
```

```
using namespace std;

typedef long long ll;
const int maxn = 250100;

#define lson(x) ((x)<<1)
#define rson(x) (((x)<<1)|1)
int lc[maxn << 2], rc[maxn << 2], v[maxn << 2];
ll nd[maxn << 2], ad[maxn << 2], s[maxn << 2];

void pushup(int u);
void pushdown (int u);

inline int length(int u) {
    return rc[u] - lc[u] + 1;
}

inline void change (int u, ll a) {
    v[u] = 1;
    ad[u] = 0;
    nd[u] = a;
    s[u] = a * length(u);
}

inline void maintain (int u, ll a, ll d) {
    if (v[u] && lc[u] != rc[u]) {
        pushdown(u);
        pushup(u);
    }

    v[u] = 0;
    nd[u] += a;
    ad[u] += d;
    ll n = length(u);
    s[u] += a * n + (((n-1) * n) / 2) * d;
}

inline void pushup (int u) {
    s[u] = s[lson(u)] + s[rson(u)];
}

inline void pushdown (int u) {
    if (v[u]) {
        change(lson(u), nd[u]);
        change(rson(u), nd[u]);
        v[u] = nd[u] = 0;
    } else if (nd[u] || ad[u]) {
        maintain(lson(u), nd[u], ad[u]);
        maintain(rson(u), nd[u] + length(lson(u)) * ad[u], ad[u]);
        nd[u] = ad[u] = 0;
    }
}

void build (int u, int l, int r) {
    lc[u] = l;
    rc[u] = r;
    nd[u] = ad[u] = s[u] = 0;

    if (l == r)
        return;
    int mid = (l + r) / 2;
    build(lson(u), l, mid);
    build(rson(u), mid + 1, r);
    pushup(u);
}
```

```
void modify(int u, int l, int r, ll a, ll d) {
    if (l <= lc[u] && rc[u] <= r) {
        maintain(u, a + d * (lc[u] - l), d);
        return;
    }

    pushdown(u);
    int mid = (lc[u] + rc[u]) / 2;
    if (l <= mid)
        modify(lson(u), l, r, a, d);
    if (r > mid)
        modify(rson(u), l, r, a, d);
    pushup(u);
}

void set(int u, int l, int r, ll a) {
    if (l <= lc[u] && rc[u] <= r) {
        change(u, a);
        return;
    }

    pushdown(u);
    int mid = (lc[u] + rc[u]) / 2;
    if (l <= mid)
        set(lson(u), l, r, a);
    if (r > mid)
        set(rson(u), l, r, a);
    pushup(u);
}

ll query (int u, int l, int r) {
    if (l <= lc[u] && rc[u] <= r)
        return s[u];

    pushdown(u);
    ll ret = 0;
    int mid = (lc[u] + rc[u]) / 2;
    if (l <= mid)
        ret += query(lson(u), l, r);
    if (r > mid)
        ret += query(rson(u), l, r);
    pushdown(u);
    return ret;
}

int N;
int main () {
    while (~scanf("%d", &N)) {
        char op[5];
        int l, r, x;
        build(1, 1, 250000);
        while (N--) {
            scanf("%s%d%d", op, &l, &r);
            if (op[0] == 'A')
                modify(1, l, r, 1, 1);
            else if (op[0] == 'B')
                modify(1, l, r, r - l + 1, -1);
            else if (op[0] == 'C') {
                scanf("%d", &x);
                set(1, l, r, x);
            } else
                printf("%lld\n", query(1, l, r));
        }
    }
    return 0;
}
```

```
/******************
input
7
A 1 4
B 2 3
S 1 3
C 3 4 -2
S 2 4
B 1 3
S 2 4
output
9
0
3
******************/
/*******************************************************************************/
```

例题：Hdu 3333（区间不重复求和）

题目大意：给定一个长度为 N 的序列，有 M 次查询，每次查询 l，r 之间元素的总和，相同元素只算一次。

解题思路：离线操作，将查询按照右区间排序，每次考虑一个询问，将 mv ~ r 的点全部标记为存在，并且对于每个位置 i，如果 A[i]在前面已经出现过了，那么将前面的那个位置减掉 A[i]，当前位置添加 A[i]，这样做维护了每个数尽量做，那么碰到查询，用 sum[r] - sum[l-1]即可。

/*****************************hdu3333.cpp*********************************/

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <map>
#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 30000;

int N, Q;
ll A[maxn+5], ans[100005];
map<ll, int> G;

#define lson(x) ((x)<<1)
#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2];
ll s[maxn << 2];

inline void pushup (int u) {
    s[u] = s[lson(u)] + s[rson(u)];
}

void build (int u, int l, int r) {
    lc[u] = l;
    rc[u] = r;
    s[u] = 0;

    if (l == r)
        return;

    int mid = (lc[u] + rc[u]) / 2;
    build(lson(u), l, mid);
    build(rson(u), mid + 1, r);
    pushup(u);
}
```

```
void modify(int u, int x, ll d) {
    if (x == lc[u] && rc[u] == x) {
        s[u] += d;
        return;
    }

    int mid = (lc[u] + rc[u]) / 2;
    if (x <= mid)
        modify(lson(u), x, d);
    else
        modify(rson(u), x, d);
    pushup(u);
}

ll query(int u, int l, int r) {
    if (l <= lc[u] && rc[u] <= r)
        return s[u];

    ll ret = 0;
    int mid = (lc[u] + rc[u]) / 2;
    if (l <= mid)
        ret += query(lson(u), l, r);
    if (r > mid)
        ret += query(rson(u), l, r);
    pushup(u);
    return ret;
}

struct Seg {
    int l, r, id;
    Seg (int l = 0, int r = 0, int id = 0) {
        this->l = l;
        this->r = r;
        this->id = id;
    }
    friend bool operator < (const Seg& a, const Seg& b) {
        return a.r < b.r;
    }
};
vector<Seg> vec;

void init () {
    int l, r;
    G.clear();
    vec.clear();

    scanf("%d", &N);
    for (int i = 1; i <= N; i++)
        scanf("%I64d", &A[i]);

    scanf("%d", &Q);
    for (int i = 1; i <= Q; i++) {
        scanf("%d%d", &l, &r);
        vec.push_back(Seg(l, r, i));
    }
    sort(vec.begin(), vec.end());
}

void solve () {
    build (1, 0, N);
    int k = 0;
    for (int i = 0; i < Q; i++) {
        for ( ; k <= vec[i].r; k++) {
            if (G[A[k]])
                modify(1, G[A[k]], -A[k]);
            G[A[k]] = k;
```

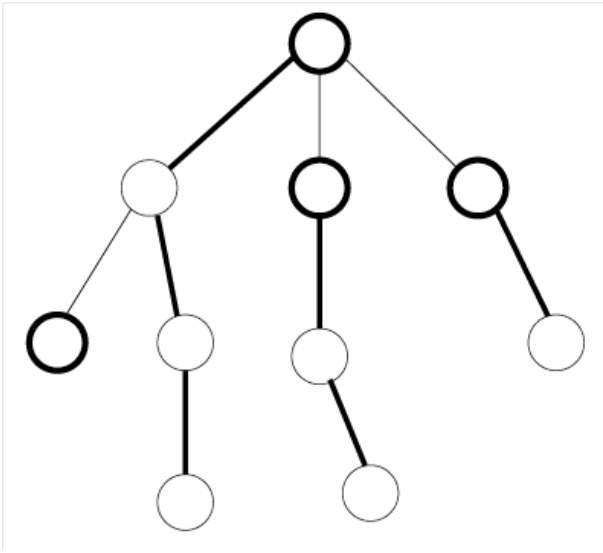
```
            modify(1, k, A[k]);
        }
        ans[vec[i].id] = query(1, vec[i].l, vec[i].r);
    }
    for (int i = 1; i <= Q; i++)
        printf("%I64d\n", ans[i]);
}

int main () {
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        init();
        solve();
    }
    return 0;
}

/********************
input
2
3
1 1 4
2
1 2
2 3
5
1 1 2 1 3
3
1 5
2 4
3 5
output
1
5
6
3
6
********************/
/*******************************************************************************/
```

（五）树链剖分：维护树的路劲上点权边权问题。

动态维护树上两节点间路径权值的修改和查询。将节点 1 默认为根节点，通过两次 dfs 处理出重链和轻链。维护则主要是在重链上操作。

init()	初始化操作，包括读入边
add_Edge(u, v);	填加一条边，以链表的形式储存
dfs(u,pre,d);	处理出每个节点的 far，dep，cnt，son
dfs(u, rot);	处理处每个节点的 top，idx
modify(u, v, w);	修改 u，v 节点之间的路径
query(u, v);	询问 u，v 节点之间的路径

idx：节点的映射值； dep:节点的深度； top：节点所在重链的根； far：节点的父亲节点；son：与节点在同一条重链的孩子节点； cnt：以节点为根节点的子树的节点数。



注：加粗变为重边，加粗节点为对应重边的根。

/***/
/*************************TreeChain-division.cpp*****************************/

```
const int maxn = 1e5+5;
int N, E, first[maxn], jump[maxn * 2], link[maxn * 2], val[maxn];
int id, idx[maxn], dep[maxn], top[maxn], far[maxn], son[maxn], cnt[maxn];

inline void add_Edge (int u, int v) {
    link[E] = v;
    jump[E] = first[u];
    first[u] = E++;
}
void dfs (int u, int pre, int d) {
    far[u] = pre;
    dep[u] = d;
    cnt[u] = 1;
    son[u] = 0;
    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == pre)
            continue;
        dfs(v, u, d + 1);
        cnt[u] += cnt[v];
        if (cnt[son[u]] < cnt[v])
            son[u] = v;
    }
}
```

```
void dfs(int u, int rot) {
    top[u] = rot;
    idx[u] = ++id;
    if (son[u])
        dfs(son[u], rot);
    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == far[u] || v == son[u])
            continue;
        dfs(v, v);
    }
}

void init () {
    int u, v;
    id = E = 0;
    memset(first, -1, sizeof(first));

     /*** input edge ***/

    dfs(1, 0, 0);
    dfs(1, 1);
}

void modify (int u, int v, int w) {
    int p = top[u], q = top[v];
    while (p != q) {
        if (dep[p] < dep[q]) {
            swap(p, q);
            swap(u, v);
        }
          /*** modify idx[p] ~ idx[u] ***/
        u = far[p];
        p = top[u];
    }

    if (dep[u] > dep[v])
          swap(u, v);
     /*** 点权 modify idx[u] ~ idx[v] ***/
     /*** 边权 modify idx[son[u]] ~ idx[v] ***/
}

int query (int u, int v) {
    int p = top[u], q = top[v], ret = 0;
    while (p != q) {
        if (dep[p] < dep[q]) {
            swap(p, q);
            swap(u, v);
        }
          /*** query idx[p] ~ idx[u] ***/
        u = far[p];
        p = top[u];
    }

    if (dep[u] > dep[v])
          swap(u, v);
     /*** 点权 query idx[u] ~ idx[v] ***/
     /*** 边权 query idx[son[u]] ~ idx[v] ***/
     return ret;
}
/*******************************************************************************/
```

例题：hdu 4897（树链剖分点边权变形）

题目大意：给定一棵树，每条边有黑白两种颜色，初始都是白色，现在有三种操作：

- 1 u v：u 到 v 路径上的边都取成相反的颜色
- 2 u v：u 到 v 路径上相邻的边都取成相反的颜色（相邻即仅有一个节点在路径上）
- 3 u v：查询 u 到 v 路径上有多少个黑色边

解题思路：树链剖分，用两个线段 W 和 L 维护，W 对应的是每条的黑白情况，L 表示的是每个节点的相邻边翻转情况

（对于轻链而言，重链直接在 W 上修改）

对于 1 操作，即为普通的树链剖分，直接在 W 上修改即可。

对于 2 操作，每次修改在 L 上，不过链的两端（即重链的部分）还需要在 W 上修改

对于 3 操作，每次查询，重链可以直接根据 W 中的值判断，轻链还要根据对应节点的 L 值来确定。

/**********hdu4897.cpp**********/

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;
const int maxn = 1e5+5;
#define lson(x) ((x)<<1)
#define rson(x) (((x)<<1)|1)
struct Tree {
    int lc[maxn << 2], rc[maxn << 2], fp[maxn << 2], s[maxn << 2];
    void splay(int u) {
        s[u] = rc[u] - lc[u] + 1 - s[u];
        fp[u] ^= 1;
    }
    void pushup(int u) {
        s[u] = s[lson(u)] + s[rson(u)];
    }
    void pushdown(int u) {
        if (fp[u]) {
            splay(lson(u));
            splay(rson(u));
            fp[u] = 0;
        }
    }
    void build (int u, int l, int r) {
        lc[u] = l;
        rc[u] = r;
        fp[u] = s[u] = 0;

        if (l == r)
            return;

        int mid = (l + r) / 2;
        build(lson(u), l, mid);
        build(rson(u), mid + 1, r);
        pushup(u);
    }
    void modify(int u, int l, int r) {
        if (l <= lc[u] && rc[u] <= r) {
            splay(u);
            return;
        }
        pushdown(u);
        int mid = (lc[u] + rc[u]) / 2;
        if (l <= mid)
            modify(lson(u), l, r);
        if (r > mid)
```

```
            modify(rson(u), l, r);
        pushup(u);
    }
    int query(int u, int l, int r) {
        if (l <= lc[u] && rc[u] <= r)
            return s[u];
        pushdown(u);
        int mid = (lc[u] + rc[u]) / 2, ret = 0;
        if (l <= mid)
            ret += query(lson(u), l, r);
        if (r > mid)
            ret += query(rson(u), l, r);
        pushup(u);
        return ret;
    }
}W, L;

int N, Q, E, first[maxn], jump[maxn * 2], link[maxn * 2];
int id, idx[maxn], top[maxn], far[maxn], son[maxn], dep[maxn], cnt[maxn];
inline void add_Edge(int u, int v) {
    link[E] = v;
    jump[E] = first[u];
    first[u] = E++;
}
void dfs (int u, int pre, int d) {
    far[u] = pre;
    son[u] = 0;
    cnt[u] = 1;
    dep[u] = d;

    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == pre)
            continue;
        dfs(v, u, d + 1);
        cnt[u] += cnt[v];
        if (cnt[son[u]] < cnt[v])
            son[u] = v;
    }
}
void dfs (int u, int rot) {
    top[u] = rot;
    idx[u] = ++id;
    if (son[u])
        dfs(son[u], rot);
    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == far[u] || v == son[u])
            continue;
        dfs(v, v);
    }
}
void init () {
    E = id = 0;
    memset(first, -1, sizeof(first));
    int u, v;
    scanf("%d", &N);
    for (int i = 1; i < N; i++) {
        scanf("%d%d", &u, &v);
        add_Edge(u, v);
        add_Edge(v, u);
    }
    dfs(1, 0, 0);
    dfs(1, 1);
    W.build(1, 1, N);
    L.build(1, 1, N);
}
```

```
void modify(int u, int v, int k) {
    int p = top[u], q = top[v];
    while (p != q) {
        if (dep[p] < dep[q]) {
            swap(p, q);
            swap(u, v);
        }
        if (k) {
            L.modify(1, idx[p], idx[u]);
            W.modify(1, idx[p], idx[p]);
            W.modify(1, idx[son[u]], idx[son[u]]);
        } else
            W.modify(1, idx[p], idx[u]);
        u = far[p];
        p = top[u];
    }
    if (dep[u] > dep[v])
        swap(u, v);
    if (k) {
        L.modify(1, idx[u], idx[v]);
        W.modify(1, idx[u], idx[u]);
        W.modify(1, idx[son[v]], idx[son[v]]);
    } else {
        if (u == v)
            return;
        W.modify(1, idx[son[u]], idx[v]);
    }
}
int query(int u, int v) {
    int p = top[u], q = top[v], ret = 0;
    while (p != q) {
        if (dep[p] < dep[q]) {
            swap(p, q);
            swap(u, v);
        }
        if (u != p)
            ret += W.query(1, idx[son[p]], idx[u]);
        ret += (W.query(1, idx[p], idx[p]) ^ L.query(1, idx[far[p]], idx[far[p]]));
        u = far[p];
        p = top[u];
    }
    if (u == v) return ret;
    if (dep[u] > dep[v])
        swap(u, v);
    ret += W.query(1, idx[son[u]], idx[v]);
    return ret;
}

int main () {
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        init();

        int k, u, v;
        scanf("%d", &Q);
        while (Q--) {
            scanf("%d%d%d", &k, &u, &v);
            if (k == 3)
                printf("%d\n", query(u, v));
            else
                modify(u, v, k - 1);
        }
    }
    return 0;
}
```

```
/*****************
input
1
10
2 1
3 1
4 1
5 1
6 5
7 4
8 3
9 5
10 6
10
2 1 6
1 3 8
3 8 10
2 3 4
2 10 8
2 4 10
1 7 6
2 7 3
2 1 4
2 10 10
output
3
*****************/
/*******************************************************************************/
```

三、字符串

（一）基本

1、最长回文判断（manacher）：

传入字符串,返回字符串中最长回文子串的长度;rad 和 str 需要开两倍;起始符和分隔符不能在原字符串中出现。

/*****************************Manacher.cpp*********************************/

```
const int maxn = 2 * 1e5 + 5;

int rad[maxn];
char str[maxn];

int manachar(char* tmpstr) {
     int len = strlen(tmpstr), cnt = 0;
     str[cnt++] = '$';
     for (int i = 0; i <= len; i++) {
          str[cnt++] = '#';
          str[cnt++] = tmpstr[i];
     }
     int ans = 0, mix = 0, id = 0;
     for (int i = 1; i <= cnt; i++) {
          if (mix > i)
               rad[i] = min(rad[2 * id - i], mix - i);
          else
               rad[i] = 1;

          for ( ; str[i - rad[i]] == str[i + rad[i]]; rad[i]++) {
               if (mix < i + rad[i]) {
                    mix = i + rad[i];
                    id = i;
               }
          }

          ans = max(ans, rad[i]);
     }
     return ans - 1;
}
```

/***/

2、最小表示法：

给定一个字符串，字符串首尾相连，求出以 x 下标起始的字符串字典序最小。

/*****************************Miniexpress.cpp*********************************/

```
int miniexpress(char* s) {
     int n = strlen(s), p = 0, q = 1;
     while (p < n && q < n) {
          int i;
          for (i = 0; i < n; i++) {
               if (s[(p+i)%n] != s[(q+i)%n])
                    break;
          }

          if (s[(p+i)%n] > s[(q+i)%n])
               p = p + i + 1;
          else
               q = q + i + 1;

          if (p == q)
               q++;
     }
     return min(p, q) + 1;;
}
```

/***/

（二）KMP

1、KMP：主要用于字符串匹配，fail[i]表示 T[i-fail[i]+1~i]和 T[1~fail[i]]是相同的。

getFail(str)	预处理匹配串的失配数组
match(str1,str2)	对一个文本或者是字符串进行匹配

循环节也就是前缀后缀串，p 从 fail[n]开始，所有的 p=fail[p]，1~p 均为循环节。（模板传入字符串从下标 1 开始）

```
/*********************************************************************************/
/**********************************KMP.cpp**********************************/
const int maxn = 1e5 + 5;
int fail[maxn];

void getFail (char* str) {
     int n = strlen(str+1);
    int p = fail[0] = fail[1] = 0;

    for (int i = 2; i <= n; i++) {
        while (p && str[p+1] != str[i])
            p = fail[p];

        if (str[p+1] == str[i])
            p++;
        fail[i] = p;
    }

     /*
     for (int i = 1; i <= n; i++)
          printf("%d%c", fail[i], i == n ? '\n' : ' ');
          */
}

int match (char* S, char* T) {
     getFail(T);

    int p = 0, ret = 0, n = strlen(S);
    for (int i = 1; i <= n; i++) {
        while (p && T[p+1] != S[i])
            p = fail[p];

        if (T[p+1] == S[i])
            p++;

        ret = max(ret, p);
    }
    return ret;
}
/******************
str:  abaabaaab
fail: 001123412
******************/
/*********************************************************************************/
```

2、e-KMP：给出两个字符串 A（称为模板串）和 B（称为子串），长度分别为 lenA 和 lenB，要求在线性时间内，对于每个 A[i]（0<i≤lenA），求出 A[i]往前和 B 的前缀匹配的最大匹配长度，即 T[i~i+cpfix[i]-1]和 T[1~cpfix[i]]相等。

getFail(str)	预处理匹配串的失配数组
getExtand(str1,str2)	对模板串求 extand 的数组

（模板传入字符串从下标 1 开始）

```
/*******************************************************************************/
/*********************************e-KMP.cpp*************************************/
const int maxn = 1e5 + 5;
int cpfix[maxn], extand[maxn];

void getCPfix(char* str) {
     int n = strlen(str + 1), tmp = 1;
     cpfix[1] = n;

     while (tmp < n && str[tmp] == str[tmp+1])
          tmp++;
     cpfix[2] = tmp-1;

     int p = 2;
     for (int k = 3; k <= n; k++) {
          int e = p + cpfix[p] - 1, l = cpfix[k-p+1]; // e 为目前匹配过的最末位置, l 为对应的偏移;

          if (k + l - 1 >= e) {
               int j = e - k > 0 ? e - k : 0;
               while (k + j <= n && str[k+j] == str[j+1])
                    j++;
               cpfix[k] = j, p = k;
          } else
               cpfix[k] = l;
     }
     /*
     for (int i = 1; i <= n; i++)
          printf("%d%c", cpfix[i], i == n ? '\n' : ' ');
          */
}

void getExtand(char* S, char* T) {
     getCPfix(T);

     int sn = strlen(S+1), tn = strlen(T+1);
     int n = min(sn, tn), tmp = 1;

     while (tmp <= n && S[tmp] == T[tmp])
          tmp++;
     extand[1] = tmp-1;

     int p = 1;
     for (int k = 2; k <= sn; k++) {
          int e = p + extand[p] - 1, l = cpfix[k-p+1];
          if (k + l - 1 >= e) {
               int j = e - k > 0 ? e - k : 0;
               while (k + j <= sn && j < tn && S[k+j] == T[j+1])
                    j++;
               extand[k] = j, p = k;

          } else
               extand[k] = l;
     }
     /*
     for (int i = 1; i <= sn; i++)
          printf("%d%c", extand[i], i == sn ? '\n' : ' ');
          */
}
/***************
S:     abaabaaab
T:     aaabaabab
Cpfix: 921021010
Extand:102104210
***************/
/*******************************************************************************/
```

（三）字典树(Tire)

对字符串集合建立的快速检索树。

init()	初始化，设置 sz
idx(ch)	映射字符（根据题意映射字符）
insert(str, id)	向字典树中插入一个字符串
match(str)	完全匹配一个字符串

```
/*****************************************************************************/
/*********************************Tire-Tree.cpp*******************************/
const int maxn = 3000005;
const int sigma_size = 26;

struct Tire {
    int sz;
    int g[maxn][sigma_size];
    int val[maxn];

    void init();
    int idx(char ch);
    void insert(char* s);
     int match(char* s);
};

/*Code*/

void Tire::init() {
    sz = 1;
    val[0] = 0;
    memset(g[0], 0, sizeof(g[0]));
}
int Tire::idx (char ch) {
    return ch - 'a';
}
int Tire::match(char* s) {
     int u = 0, n = strlen(s);

     for (int i = 0; i < n; i++) {
          int v = idx(s[i]);

          if (g[u][v] == 0)
               return 0;
          u = g[u][v];
     }
     return val[u];
}
void Tire::insert(char* s) {
    int u = 0, n = strlen(s);
    for (int i = 0; i < n; i++) {
        int v = idx(s[i]);

        if (g[u][v] == 0) {
            val[sz] = 0;
            memset(g[sz], 0, sizeof(g[sz]));
            g[u][v] = sz++;
        }

        u = g[u][v];
     }
     val[u]++;
}
/*****************************************************************************/
```

应用：

（1） 匹配去重：利用对所有的字符串建立字典树，两个字符串不相同的话，对应的终止节点肯定不同。

例题：hdu 4760（字典树去重高级应用）

题目大意：有一个防火墙，具有添加一个子网络，删除一个子网络，以及转发包的操作。

- 添加操作包含子网络的 id，以及子网络的子网掩码（计算出网络前缀，以及 ip 的下限），不会超过 15 个。
- 删除则是给定要删除的子网络 id。
- 转发操作，给定两个 ip，如果两个 ip 在同一个子网络中，则可以转发，否则丢弃。

解题思路：对子网掩码前缀建立字典树，每个前缀终止节点用一个 set 记录属于哪些子网络，ip 下限。那么增加和删除操作既可以解决了。对于查询操作，分别查询两个 ip，处理除两个 ip 可能属于的网络，判断有无共同即可。

/**********hdu4760.cpp**********/

```
#include <cstdio>
#include <cstring>
#include <set>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef pair<int, ll> pii;
typedef set<pii>::iterator iter;
const int maxn = 1024 * 15 + 10;
const int maxm = 105;
const int sigma_size = 2;

struct Tire {
    int sz, sv;
    int g[maxn * maxm][sigma_size];
    int idx[maxn * maxm], cnt[1030];
    set<pii> ans, vis[maxn];

    void init();
    int newSet();
    void addSet(int id, ll limt);
    void insert(char* str, pii x);
    void remove(char* str, pii x);
    void find(char* str);
    bool judge(char* a, char* b);
}T;

struct Network {
    int n;
    ll suf[20];
    char ip[20][maxm];
    Network() {
        n = 0;
        memset(suf, 0, sizeof(suf));
    }
}net[1030];
ll change(char* ans, bool flag) {
    char str[105];
    int n = strlen(str), a[4], b;
    if (flag)
        scanf("%d.%d.%d.%d/%d", &a[0], &a[1], &a[2], &a[3], &b);
    else {
        scanf("%d.%d.%d.%d", &a[0], &a[1], &a[2], &a[3]);
        b = 32;
    }

    int t = 0;
```

```
    ll ret = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 7; j >= 0; j--) {
            if (t < b)
                ans[t] = ((a[i]>>j)&1) + '0';
            else if (((a[i]>>j)&1))
                ret |= (1LL<<(31-t));
            t++;
        }
    }
    ans[t] = '\0';
    return ret;
}

int main () {
    int id;
    char op[5], a[maxm], b[maxm], c[maxm];

    T.init();
    while (scanf("%s", op) == 1) {
        if (op[0] == 'E') {
            scanf("%d", &id);
            scanf("%d", &net[id].n);

            for (int i = 0; i < net[id].n; i++) {
                net[id].suf[i] = change(net[id].ip[i], a);
                T.insert(net[id].ip[i], make_pair(id, net[id].suf[i]));
            }

        } else if (op[0] == 'D') {
            scanf("%d", &id);

            for (int i = 0; i < net[id].n; i++)
                T.remove(net[id].ip[i], make_pair(id, net[id].suf[i]));
            net[id].n = 0;

        } else {
            change(a, 0);
            change(b, 0);
            printf("%c\n", T.judge(a, b) ? 'F' : 'D');
        }
    }
    return 0;
}

bool Tire::judge(char* a, char* b) {
    memset(cnt, 0, sizeof(cnt));

    T.find(a);
    for (iter i = ans.begin(); i != ans.end(); i++)
        cnt[i->first] = 1;

    T.find(b);
    for (iter i = ans.begin(); i != ans.end(); i++)
        if (cnt[i->first])
            return true;
    return false;
}

void Tire::init() {
    sz = sv = 1;
    idx[0] = 0;
    vis[0].clear();
    memset(g[0], 0, sizeof(g[0]));
}

int Tire::newSet() {
```

```
    vis[sv].clear();
    return sv++;
}

void Tire::addSet(int id, ll limt) {
    for (iter i = vis[id].begin(); i != vis[id].end(); i++)
        if (i->second <= limt)
            ans.insert(*i);
}

void Tire::insert(char* str, pii x) {
    int u = 0, n = strlen(str);
    for (int i = 0; i < n; i++) {
        int v = str[i] - '0';
        if (g[u][v] == 0) {
            idx[sz] = 0;
            memset(g[sz], 0, sizeof(g[sz]));
            g[u][v] = sz++;
        }
        u = g[u][v];
    }
    if (idx[u] == 0)
        idx[u] = newSet();
    vis[idx[u]].insert(x);
}

void Tire::remove(char* str, pii x) {
    int u = 0, n = strlen(str);
    for (int i = 0; i < n; i++) {
        int v = str[i] - '0';
        if (g[u][v] == 0) {
            idx[sz] = 0;
            memset(g[sz], 0, sizeof(g[sz]));
            g[u][v] = sz++;
        }
        u = g[u][v];
    }
    vis[idx[u]].erase(x);
}

void Tire::find(char* str) {
    ans.clear();

    ll ret = 0;
    int u = 0, n = strlen(str);

    for (int i = 0; i < n; i++) {
        int v = str[i] - '0';

        if (g[u][v] == 0)
            return;

        u = g[u][v];
        if (idx[u]) {
            ll ret = 0;
            for (int j = i+1; j < n; j++)
                if (str[j] == '1')
                    ret |= (1LL<<(31-j));
            addSet(idx[u], ret);
        }
    }
}
/*****************************************************************************/
```

（2） 模糊匹配:即模拟搜索引擎的模拟搜索或者是正则表达式,一般要在字典树上做 DFS 进行匹配。

（3） 亦或和最大问题:对所有的数建立字典树,在有对于既定树,只要在字典树上移动,尽量向亦或和大的方向走。

例题：hdu 4757（可持久化字典树解决亦或和）

题目大意：给定一棵树，每个节点有一个值，现在有 Q 次询问，每次询问 u 到 v 路径上节点值与 w 亦或值的最大值。
解题思路： 可持久化字典树，在每次插入的同时，不修改原先的节点，而是对所有修改的节点复制一个新的节点，并且在新的节点上做操作，这样做的目的是能够获取某次修改前的状态。同过可持久化的操作，保留了修改前后的公共数据。

对给定树上的所有节点权值建立 01 字典树，然后每个节点都保存着一棵可持久化字典树，表示的是从根节点到该节点路径节点所形成的字典树。对每个节点建树的过程通过修改其父亲节点而得到。

查询时，对根据 u，v，lca(u,v)三棵字典树的情况确定亦或的最大值，注意 lca(u,v)这个节点要单独计算。

/*******************************hdu4757.cpp*********************************/

```
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

int N, Q, E, V[maxn], first[maxn], jump[maxn * 2], link[maxn * 2];
int id, idx[maxn], top[maxn], far[maxn], son[maxn], dep[maxn], cnt[maxn];

inline void add_Edge (int u, int v) {
    link[E] = v;
    jump[E] = first[u];
    first[u] = E++;
}

inline void dfs (int u, int pre, int d) {
    far[u] = pre;
    son[u] = 0;
    dep[u] = d;
    cnt[u] = 1;

    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == pre)
            continue;
        dfs(v, u, d + 1);
        cnt[u] += cnt[v];
        if (cnt[son[u]] < cnt[v])
            son[u] = v;
    }
}

inline void dfs (int u, int rot) {
    idx[u] = ++id;
    top[u] = rot;

    if(son[u])
        dfs(son[u], rot);

    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == far[u] || v == son[u])
            continue;
        dfs(v, v);
    }
}
```

```
inline int LCA (int u, int v) {
    int p = top[u], q = top[v];
    while (p != q) {
        if (dep[p] < dep[q]) {
            swap(p, q);
            swap(u, v);
        }

        u = far[p];
        p = top[u];
    }
    return dep[u] > dep[v] ? v : u;
}

void init() {
    E = id = 0;
    memset(first, -1, sizeof(first));
    for (int i = 1; i <= N; i++)
        scanf("%d", &V[i]);

    int u, v;
    for (int i = 1; i < N; i++) {
        scanf("%d%d", &u, &v);
        add_Edge(u, v);
        add_Edge(v, u);
    }
    dfs(1, 0, 0);
    dfs(1, 1);
}

struct node {
    int g[2], c;
}nd[maxn * 20];
int sz, root[maxn];

int insert (int r, int w) {
    int ret, x;
    ret = x = sz++;
    nd[x] = nd[r];

    for (int i = 15; i >= 0; i--) {
        int v = (w>>i)&1;
        int t = sz++;
        nd[t] = nd[nd[x].g[v]];
        nd[t].c++;
        nd[x].g[v] = t;
        x = t;
    }
    return ret;
}

void dfs(int u) {
    root[u] = insert(root[far[u]], V[u]);

    for (int i = first[u]; i + 1; i = jump[i]) {
        int v = link[i];
        if (v == far[u])
            continue;
        dfs(v);
    }
}
void Tire_init() {
    sz = 1;
    root[0] = nd[0].c = 0;
    memset(nd[0].g, 0, sizeof(nd[0].g));
    dfs(1);
}
```

```
int query(int x, int y, int z, int w) {
    int ans = V[z] ^ w, ret = 0;
    z = root[z];
    for (int i = 15; i >= 0; i--) {
        int v = ((w>>i)&1) ^ 1;
        int cnt = nd[nd[x].g[v]].c + nd[nd[y].g[v]].c - 2 * nd[nd[z].g[v]].c;

        if (cnt)
            ret |= (1<<i);
        else
            v = v^1;

        x = nd[x].g[v], y = nd[y].g[v], z = nd[z].g[v];
    }
    return max(ans, ret);
}

int main () {
    while (scanf("%d%d", &N, &Q) == 2) {
        init();
        Tire_init();

        int u, v, w;
        while (Q--) {
            scanf("%d%d%d", &u, &v, &w);
            printf("%d\n", query(root[u], root[v], LCA(u, v), w));
        }
    }
    return 0;
}
/*************************************************************************************/
```

（四）AC 自动机(Aho-Croasick)

结合 KMP 和字典树的性质，在字典树上构建失配函数。

init()	初始化，设置 sz
idx(ch)	映射字符（根据题意映射字符）
insert(str, id)	向 AC 自动机中插入一个字符串
getFail()	对 AC 自动机处理失配函数
match(str)	匹配一个字符串
mark()	标记匹配到的位置

```
/*****************************************************************************/
/*****************************Aho-Croasick.cpp*****************************/
const int maxn = 205;
const int sigma_size = 26;

struct Aho_Corasick {
    int sz, g[maxn][sigma_size];
    int tag[maxn], fail[maxn], last[maxn];

    void init();
    int idx(char ch);
    void insert(char* str, int k);
     void getFail();
     void match(char* str);
     void mark(int x, int y);
};

/* Code */

void Aho_Corasick::init() {
     sz = 1;
     tag[0] = 0;
     memset(g[0], 0, sizeof(g[0]));
}
int Aho_Corasick::idx(char ch) {
     return ch - 'a';
}
void Aho_Corasick::mark(int x, int y) {}
void Aho_Corasick::insert(char* str, int k) {
     int u = 0, n = strlen(str);
     for (int i = 0; i < n; i++) {
          int v = idx(str[i]);
          if (g[u][v] == 0) {
               tag[sz] = 0;
               memset(g[sz], 0, sizeof(g[sz]));
               g[u][v] = sz++;
          }
          u = g[u][v];
     }
     tag[u] = k;
}
void Aho_Corasick::match(char* str) {
     int n = strlen(str), u = 0;
     for (int i = 0; i < n; i++) {
          int v = idx(str[i]);
          while (u && g[u][v] == 0)
               u = fail[u];
          u = g[u][v];
          if (tag[u])     mark(i, u);
          else if (last[u])    mark(i, last[u]);
     }
}
```

```
void Aho_Corasick::getFail() {
     queue<int> que;
     for (int i  = 0; i < sigma_size; i++) {
          int u = g[0][i];
          if (u) {
               fail[u] = last[u] = 0;
               que.push(u);
          }
     }
     while (!que.empty()) {
          int r = que.front();
          que.pop();
          for (int i = 0; i < sigma_size; i++) {
               int u = g[r][i];
               if (u == 0) {
                    g[r][i] = g[fail[r]][i];
                    continue;
               }
               que.push(u);
               int v = fail[r];
               while (v && g[v][i] == 0)
                    v = fail[v];
               fail[u] = g[v][i];
               last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
          }
     }
}
/*****************************************************************************/
```

应用：

（1） 匹配问题：不同于字典树的是 AC 自动机是处理子串匹配。

（2） 结合 DP：一般有矩阵快速幂（根据自动机构建矩阵），状压 DP。

例题：hdu 3341（AC 自动机+变进制状压 DP）

题目大意：给定一些需要匹配的串，然后在给定一个目标串，现在可以通过交换目标串中任意两个位置的字符，要求最后生成的串匹配尽量多的匹配串，可以重复匹配。

解题思路：这题很明显是 AC 自动机+DP，但是 dp 的状态需要开 40*40*40*40（记录每种字符的个数），空间承受不了，但是其实因为目标串的长度有限，为 40；所以状态更本不需要那么多，最多只有 10*10*10*10，但是通过 40 进制的 hash 转换肯定是不行，可以根据目标串中 4 种字符的个数，来调整每个位的进制。

```
/*********************************hdu3341.cpp*********************************/
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
typedef pair<int,int> pii;
const int maxn = 505;
const int maxs = 11 * 11 * 11 * 11;
const int sigma_size = 4;
struct Aho_Corasick {
    int sz, g[maxn][sigma_size];
    int tag[maxn], fail[maxn], last[maxn];
    int c[4], bit[4], dp[maxs][maxn];
    void init();
    int idx(char ch);
    void insert(char* str, int k);
    void getFail();
    void match(char* str);
    void put(int x, int y);
    int solve(char* w);
    int hash(int a, int b, int c, int d);
}AC;
```

```
int N;
char w[50];
int main () {
    int cas = 1;
    while (scanf("%d", &N) == 1 && N) {
        AC.init();
        for (int i = 1; i <= N; i++) {
            scanf("%s", w);
            AC.insert(w, i);
        }
        scanf("%s", w);
        printf("Case %d: %d\n", cas++, AC.solve(w));
    }
    return 0;
}
int Aho_Corasick::hash(int a, int b, int c, int d) {
    return a * bit[0] + b * bit[1] + c * bit[2] + d;
}
int Aho_Corasick::solve(char* w) {
    getFail();
    int n = strlen(w);
    memset(c, 0, sizeof(c));
    for (int i = 0; i < n; i++)
        c[idx(w[i])]++;
    for (int i = 0; i < 4; i++) {
        bit[i] = 1;
        for (int j = i + 1; j < 4; j++)
            bit[i] *= (c[j]+1);
    }
    int ans = 0, t[4];
    memset(dp, -1, sizeof(dp));
    dp[hash(c[0], c[1], c[2], c[3])][0] = 0;

    for (t[0] = c[0]; t[0] >= 0; t[0]--)
        for (t[1] = c[1]; t[1] >= 0; t[1]--)
            for (t[2] = c[2]; t[2] >= 0; t[2]--)
                for (t[3] = c[3]; t[3] >= 0; t[3]--) {
                    int s = hash(t[0], t[1], t[2], t[3]);
                    for (int i = 0; i < 4; i++) {
                        if (t[i] == 0)
                            continue;
                        int ss = s - bit[i];
                        for (int k = 0; k < sz; k++) {
                            if (dp[s][k] < 0)
                                continue;
                            int u = k;
                            while (u && g[u][i] == 0)
                                u = fail[u];
                            u = g[u][i];

                            if (dp[ss][u] < dp[s][k] + tag[u]) {
                                dp[ss][u] = dp[s][k] + tag[u];
                                ans = max(ans, dp[ss][u]);
                            }
                        }
                    }
                }

    return ans;
}

void Aho_Corasick::init() {
    sz = 1;
    tag[0] = 0;
    memset(g[0], 0, sizeof(g[0]));
}
```

```
int Aho_Corasick::idx(char ch) {
    if (ch == 'A')
        return 0;
    if (ch == 'C')
        return 1;
    if (ch == 'G')
        return 2;
    return 3;
}
void Aho_Corasick::put(int x, int y) {}
void Aho_Corasick::insert(char* str, int k) {
    int u = 0, n = strlen(str);
    for (int i = 0; i < n; i++) {
        int v = idx(str[i]);
        if (g[u][v] == 0) {
            tag[sz] = 0;
            memset(g[sz], 0, sizeof(g[sz]));
            g[u][v] = sz++;
        }
        u = g[u][v];
    }
    tag[u]++;
}
void Aho_Corasick::match(char* str) {
    int n = strlen(str), u = 0;
    for (int i = 0; i < n; i++) {
        int v = idx(str[i]);
        while (u && g[u][v] == 0)
            u = fail[u];
        u = g[u][v];
        if (tag[u])
            put(i, u);
        else if (last[u])
            put(i, last[u]);
    }
}
void Aho_Corasick::getFail() {
    queue<int> que;
    for (int i  = 0; i < sigma_size; i++) {
        int u = g[0][i];
        if (u) {
            fail[u] = last[u] = 0;
            que.push(u);
        }
    }
    while (!que.empty()) {
        int r = que.front();
        que.pop();
        for (int i = 0; i < sigma_size; i++) {
            int u = g[r][i];
            if (u == 0) {
                g[r][i] = g[fail[r]][i];
                continue;
            }
            que.push(u);
            int v = fail[r];
            while (v && g[v][i] == 0)
                v = fail[v];
            fail[u] = g[v][i];
            tag[u] += tag[fail[u]];
            //last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
        }
    }
}
/*****************************************************************************/
```

（五）后缀数组

init()	初始化，读入字符串
build()	构建字符串的后缀数组
geteight	构建字符串的 height 和 rank 数组

```
/*****************************************************************************/
/*******************************Suffix-Arr.cpp********************************/
const int maxn = 100005;

struct Suffix_Arr {
    int n, s[maxn];
    int SA[maxn], rank[maxn], height[maxn];
    int tmp_one[maxn], tmp_two[maxn], c[305];

     int d[maxn][20];

    void init(char* str);
    void build(int m);
     void getHeight();
     void rmq_init();
     int rmq_query(int x, int y);
};

void Suffix_Arr::init(char* str) {
     n = 0;
     int len = strlen(str);
     for (int i = 0; i < len; i++)
          s[n++] = str[i] - 'a' + 1;
     s[n++] = 0;
}
void Suffix_Arr::rmq_init() {
     for (int i = 0; i < n; i++) d[i][0] = height[i];

     for (int k = 1; (1<<k) <= n; k++) {
          for (int i = 0; i + (1<<k) - 1 < n; i++)
               d[i][k] = min(d[i][k-1], d[i+(1<<(k-1))][k-1]);
     }
}

int Suffix_Arr::rmq_query(int x, int y) {
     if (x == y)
          return d[x][0];
     if (x > y)
          swap(x, y);
     x++;
     int k = 0;
     while ((1<<(k+1) <= y - x + 1)) k++;
     return min(d[x][k], d[y - (1<<k) + 1][k]);
}
void Suffix_Arr::getHeight() {
     for (int i = 0; i < n; i++)
          rank[SA[i]] = i;
     int mv = height[n-1] = 0;
     for (int i = 0; i < n - 1; i++) {
          if (mv) mv--;

          int j = SA[rank[i] - 1];
          while (s[i+mv] == s[j+mv])
               mv++;
          height[rank[i]] = mv;
     }
}
```

```
void Suffix_Arr::build (int m) {
     int *x = tmp_one, *y = tmp_two;

     for (int i = 0; i < m; i++) c[i] = 0;
     for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
     for (int i = 1; i < m; i++) c[i] += c[i-1];
     for (int i = n - 1; i >= 0; i--) SA[--c[x[i]]] = i;

     for (int k = 1; k <= n; k <<= 1) {

          int mv = 0;
          for (int i = n - k; i < n; i++) y[mv++] = i;
          for (int i = 0; i < n; i++) if (SA[i] >= k)
               y[mv++] = SA[i] - k;

          for (int i = 0; i < m; i++) c[i] = 0;
          for (int i = 0; i < n; i++) c[x[y[i]]]++;
          for (int i = 1; i < m; i++) c[i] += c[i-1];
          for (int i = n - 1; i >= 0; i--) SA[--c[x[y[i]]]] = y[i];

          swap(x, y);
          mv = 1;
          x[SA[0]] = 0;

          for (int i = 1; i < n; i++)
               x[SA[i]] = (y[SA[i-1]] == y[SA[i]] && y[SA[i-1] + k] == y[SA[i] + k] ? mv - 1 : mv++);

          if (mv >= n)
               break;
          m = mv;
     }
}
/*****************************************************************************/
```