

## Contents

1 Foundations	1
1.1 PyMath	1
1.2 Java Integer	1
1.3 Java String	1
1.4 Java String builder	2
1.5 Java Math	2
2 Mathematics & Number Theory	2
2.1 Number Theory	2
2.2 Combinatorics	2
2.3 Geometry	3
3 Data Structure	3
3.1 MST	3
3.2 SegmentTree	3
3.3 HLD	3
3.4 PST	4
3.5 Trie	4
3.6 BIT 單修區查	5
3.7 BIT 區修單查	5
3.8 BIT 區修區查	5
4 Graph	5
4.1 cut vertex AND bridges	5
4.2 SCC - Tarjan	5
4.3 BCC - Tarjan	6
4.4 Convex	6
4.5 Max Flow	6
4.6 min cut max flow	7
5 String	7
5.1 KMP	7
5.2 ACAM	7
6 Techniques	8
6.1 二分搜	8
6.2 倍增 LCA	8
6.3 SG	8
7 DP	8
7.1 輪廓線 DP	8
7.2 數位 DP	8
7.3 樹 DP	8

# 1 Foundations

## 1.1 PyMath

```

1 import math
2
3     math.ceil(x) #上高斯
4     math.floor(x) #下高斯
5     math.factorial(x) #階乘
6     math.fabs(x) #絕對值
7     math.fsum(arr) #求和
8     math.gcd(x, y)
9     math.exp(x) # e^x
10    math.log(x, base)
11    math.log2(x)
12    math.log10(x)
13    math.sqrt(x)
14    math.pow(x, y, mod)
15    math.sin(x) # cos, tan, asin, acos, atan,
16      atan2, sinh ...
17    math.hypot(x, y) #歐幾里得範數
18    math.degrees(x) #x從弧度轉角度
19    math.radians(x) #x從角度轉弧度
20    math.gamma(x) #x的gamma函數
21    math.pi #const
22    math.e #const
23    math.inf

```

## 1.2 Java Integer

```

1 // 常量
2 MAX_VALUE, MIN_VALUE, BYTES, SIZE, TYPE
3
4 // 轉換/解析
5 static int parseInt(String s)
6 static int parseInt(String s, int radix)
7 static int parseUnsignedInt(String s)
8 static int parseUnsignedInt(String s, int
9   radix)
10 static Integer valueOf(int i)
11 static Integer valueOf(String s)
12 static Integer valueOf(String s, int radix)
13 static String toString(int i)
14 static String toString(int i, int radix)
15 static String toUnsignedString(int i)
16 static String toUnsignedString(int i, int
17   radix)
18 static long toUnsignedLong(int x)
19 static Integer decode(String nm)
20   // 支援 0x/0/# 前綴
21 static Integer getInteger(String nm[, int
22   val]) // 從系統屬性讀取整數
23
24 // 比較/雜湊/聚合
25 static int compare(int x, int y)
26 static int compareUnsigned(int x, int y)
27 static int hashCode(int value)
28 static int min(int a, int b)
29 static int max(int a, int b)
30 static int sum(int a, int b)
31
32 // 位元操作
33 static int bitCount(int i) // 設定位數
34 static int highestOneBit(int i)
35 static int lowestOneBit(int i)
36 static int numberOfLeadingZeros(int i)
37 static int numberOfTrailingZeros(int i)
38 static int rotateLeft(int i, int distance)
39 static int rotateRight(int i, int distance)
40 static int reverse(int i)
41 static int reverseBytes(int i)
42
43 // 無號運算
44 static int divideUnsigned(int dividend, int
45   divisor)

```

```

41 static int remainderUnsigned(int dividend,
42   int divisor)

```

## 1.3 Java String

```

1 // 查詢
2 int length()
3 boolean isEmpty()
4 boolean isBlank() // (since 11)
5 char charAt(int index)
6 int codePointAt(int index)
7 int codePointBefore(int index)
8 int codePointCount(int beginIndex, int
9   endIndex)
10 boolean contains(CharSequence s)
11 boolean startsWith(String prefix[, int
12   toffset])
13 boolean endsWith(String suffix)
14 int indexOf(String str[, int fromIndex])
15 int lastIndexOf(String str[, int
16   fromIndex])
17 // 取子字串/子序列
18 String substring(int beginIndex)
19 String substring(int beginIndex, int
20   endIndex)
21 CharSequence subSequence(int beginIndex, int
22   endIndex)
23
24 // 比較/等價
25 boolean equals(Object obj)
26 boolean equalsIgnoreCase(String
27   anotherString)
28 int compareTo(String anotherString)
29 int compareToIgnoreCase(String str)
30 boolean matches(String regex)
31 boolean regionMatches(int toffset, String
32   other, int offset, int len)
33 boolean regionMatches(boolean ignoreCase,
34   int toffset, String other, int offset,
35   int len)
36
37 // 建構/轉換/連接
38 String concat(String str)
39 String replace(char oldChar, char newChar)
40 String replace(CharSequence target,
41   CharSequence replacement)
42 String replaceAll(String regex, String
43   replacement)
44 String replaceFirst(String regex, String
45   replacement)
46 String[] split(String regex[, int limit])
47 String toLowerCase()
48 String toUpperCase()
49 String trim()
50 String strip() // (since 11)
51 String stripLeading() // (since 11)
52 String stripTrailing() // (since 11)
53 String repeat(int count) // (since 11)
54 IntStream chars()
55 Stream<String> lines() // (since 11)
56 String intern()
57
58 // 靜態工具
59 static String format(String format,
60   Object... args)
61 static String join(CharSequence delimiter,
62   CharSequence... elements)
63 static String join(CharSequence delimiter,
64   Iterable<? extends CharSequence>
65   elements)
66 static String
67   valueOf(primitive/char[]/Object)
68 static String copyValueOf(char[] data[, int
69   offset, int count])

```

## 1.4 Java String builder

```

1 // 長度/容量
2 int length()
3 int capacity()
4 void ensureCapacity(int minimumCapacity)
5 void trimToSize()
6 void setLength(int newLength)
7
8 // 存取/修改
9 char      charAt(int index)
10 void     setCharAt(int index, char ch)
11 StringBuilder append(... 各種型別 ...)
12 StringBuilder insert(int offset, ... 各種型別
... )
13 StringBuilder delete(int start, int end)
14 StringBuilder deleteCharAt(int index)
15 StringBuilder replace(int start, int end,
    String str)
16 StringBuilder reverse()
17
18 // 子字串/查找
19 String      substring(int start)
20 String      substring(int start, int end)
21 CharSequence subSequence(int start, int end)
22 int         indexOf(String str[], int
    fromIndex])
23 int         lastIndexOf(String str[], int
    fromIndex])
24
25 // 轉換
26 String toString()

```

## 1.5 Java Math

```

1 // 常量
2 static final double E, PI
3
4 // 絶對值/比較
5 static int/long/float/double abs(x)
6 static T max(a, b)
7 static T min(a, b)
8
9 // 取整/四捨五入
10 static double floor(double a)
11 static double ceil(double a)
12 static double rint(double a)          // 最接近整數(偶數優先)
13 static long round(double a) / int
    round(float a)
14 static int   floorDiv(int x, int y)
15 static int   floorMod(int x, int y)
16
17 // 溢位保護(exact 系列, Java 8+)
18 static int/long addExact(a, b)
19 static int/long subtractExact(a, b)
20 static int/long multiplyExact(a, b)
21 static int/long incrementExact(a)
22 static int/long decrementExact(a)
23 static int   toIntExact(long value)
24 static int/long negateExact(a)
25
26 // 指對數/冪根
27 static double pow(double a, double b)
28 static double sqrt(double a)
29 static double cbrt(double a)
30 static double exp(double a)
31 static double expm1(double x)
32 static double log(double a)
33 static double log10(double a)
34 static double log1p(double x)
35
36 // 三角/雙曲
37 static double sin/cos/tan(double a)
38 static double asin/acos/atan(double a)
39 static double atan2(double y, double x)

```

```

40 static double sinh/cosh/tanh(double a)
41
42 // 其他實用
43 static double hypot(double x, double y)
44 static double toDegrees(double angrad)
45 static double toRadians(double angdeg)
46 static double copySign(double magnitude,
    double sign)
47 static double nextUp/nextDown(double a)
48 static double nextAfter(double start, double
    direction)
49 static double ulp(double d)
50 static double random()
51 static double scalb(double d, int
    scaleFactor)
52 static double fma(double a, double b, double
    c) // (since 8)
53 static long multiplyHigh(long x, long y)
    // (since 9)
54 static long multiplyFull(int x, int y)
    // (since 9, 回傳 long)

```

## 2 Mathematics & Number Theory

### 2.1 Number Theory

Fermat's Little Theorem:

$$a^{p-1} \equiv 1 \pmod{p} \quad (\gcd(a, p) = 1, p \text{ prime})$$

Euler's Theorem:

$$a^{\varphi(n)} \equiv 1 \pmod{n} \quad (\gcd(a, n) = 1)$$

Modular Inverse:

$$a^{-1} \equiv a^{p-2} \pmod{p} \quad (\gcd(a, p) = 1, p \text{ prime})$$

Euler Totient:

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

Fast Modular Exponentiation

```

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

```

Counting Coprimes Below  $n$

$$\forall n > 0, \sum_{d|n} \varphi(d) = n, \quad \sum_{d|n} \mu(d) \left\lfloor \frac{n}{d} \right\rfloor = 1,$$

其中  $\mu$  為莫比烏斯函數；常用於反演及計算互質對數量。

Modulo Arithmetic Quick Facts

$$\begin{aligned} (a \pm b) \bmod m &= ((a \bmod m) \pm (b \bmod m)) \bmod m, \\ (ab) \bmod m &= ((a \bmod m)(b \bmod m)) \bmod m, \\ (a^k) \bmod m &= ((a \bmod m)^k) \bmod m, \\ (-a) \bmod m &= (m - (a \bmod m)) \bmod m. \end{aligned}$$

若  $\gcd(a, m) = 1$ ，可計算乘法逆元  $a^{-1}$  並套用  $(a/b) \bmod m \equiv a \cdot b^{-1} \bmod m$ 。

Congruence ( $a \equiv b \pmod{m}$ ) Essentials

- $a \equiv b \pmod{m} \iff m | (a - b)$ ，同餘類以差整除判斷。
- $a \equiv b \pmod{m} \Rightarrow f(a) \equiv f(b) \pmod{m}$  對所有以整數係數的多項式  $f$  成立。
- 若  $a \equiv b \pmod{m}$  且  $c \equiv d \pmod{m}$ ，則  $a \pm c \equiv b \pm d \pmod{m}$ ,  $ac \equiv bd \pmod{m}$ ,  $a^n \equiv b^n \pmod{m}$ 。

- 若  $\gcd(k, m) = 1$  且  $ka \equiv kb \pmod{m}$ ，可約去  $k: a \equiv b \pmod{m}$ 。
- 若  $a \equiv b \pmod{m}$ ，則  $a \equiv b \pmod{d}$  對任何  $d$  整除  $m$  亦成立。
- 若  $d | a, b, m$ ，則  $\frac{a}{d} \equiv \frac{b}{d} \pmod{\frac{m}{d}}$

### Chinese Remainder Theorem

設同餘系統

$$x \equiv a_i \pmod{m_i} \quad (i = 1, \dots, k),$$

其中  $m_i$  兩兩互質，令  $M = \prod_{i=1}^k m_i$ ,  $M_i = M/m_i$ ，再取  $t_i \equiv M_i^{-1} \pmod{m_i}$ 。則唯一解 (模  $M$ ) 為

$$x \equiv \sum_{i=1}^k a_i M_i t_i \pmod{M}.$$

### 兩式合併 (允許非互質)

```

1 // solve x a1 (mod m1), x a2 (mod m2)
2 // return {x0, lcm}; if no solution, lcm = -1
3 pair<ll, ll> crt(ll a1, ll m1,
4                  ll a2, ll m2) {
5     ll g = std::gcd(m1, m2);
6     if ((a2 - a1) % g != 0) return {0, -1};
    // no solution
7
8     ll lcm = m1 / g * m2;
9     ll m1_reduced = m1 / g;
10    ll m2_reduced = m2 / g;
11
12    ll diff = (a2 - a1) / g % m2_reduced;
13    if (diff < 0) diff += m2_reduced;
14
15    ll inv = mod_pow(m1_reduced, m2_reduced
    - 1, m2_reduced);
16    ll step = diff * inv % m2_reduced;
17    ll x0 = (a1 + step * m1) % lcm;
18    if (x0 < 0) x0 += lcm;
19    return {x0, lcm};
20}

```

遞增地將每個同餘式與當前解做合併即可取得最終答案，也能偵測無解情況。給定  $a, b, c$ ，求  $ax + by = c$  的解

```

1 ll extgcd(ll a, ll b, ll c, ll &x, ll
    &y){
2     if(b == 0){
3         x = c/a ;
4         y = 0 ;
5         return a ;
6     }
7     ll d = extgcd(b, a%b, c, x, y), tmp =
    x ;
8     x = y ;
9     y = tmp - (a/b)*y ;
10    return d ;
11}

```

## 2.2 Combinatorics

### Binomial Coefficient Identities

$${n \choose k} = \frac{n!}{k!(n-k)!},$$

$${n \choose k} = {n-1 \choose k} + {n-1 \choose k-1},$$

$$\sum_{k=0}^n {n \choose k} = 2^n, \quad \sum_{k=0}^n k {n \choose k} = n2^{n-1}.$$

### Stars and Bars 非負整數解數量：

$$x_1 + x_2 + \dots + x_k = n \Rightarrow {n+k-1 \choose k-1}.$$

若各變數至少為 1，將  $x_i = y_i + 1$  轉為非負情況即可。

**Inclusion-Exclusion Principle**對集合  $A_1, \dots, A_k$  :

$$\left| \bigcup_{i=1}^k A_i \right| = \sum_{i=1}^k |A_i| - \sum_{1 \leq i < j \leq k} |A_i \cap A_j| \\ + \cdots + (-1)^{k-1} |A_1 \cap \cdots \cap A_k|.$$

計算滿足限制的排列或整數解時廣泛使用。

**Catalan Numbers** 基本定義：

$$C_n = \frac{1}{n+1} \binom{2n}{n}, \quad C_{n+1} = \frac{4n+2}{n+2} C_n.$$

$$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \dots$$

常見應用包含合法括號、凸多邊形三角剖分、二元樹結構計數等。

## 2.3 Geometry

對於  $V$  個點， $E$  條邊， $F$  個面， $C$  個連通分量

$$V + F = E + 2$$

$$V + F = E + C + 1$$

點座標均是整數或是正方形格子點的簡單多邊形，其面積  $A$  和內部點數量  $i$ ，邊上格點數量  $b$  的關係為

$$A = i + \frac{b}{2} - 1$$

## 3 Data Structure

### 3.1 MST

```

1 struct Edge{
2     int u, v, w ;
3     // 這是最大生成樹，最小生成樹要改成 w < o.w
4     bool operator>(const Edge &o) const
5     {return w > o.w ;};
6 }
7 int par[N] ;
8 int sz[N] ;
9 int sum ;
10
11 vector<Edge> edge ;
12
13 void init(){
14     edge.clear() ;
15     for ( int i=0 ; i<N ; i++ ){
16         par[i] = i ;
17         sz[i] = 1 ;
18     }
19     sum = 0 ;
20 }
21
22 int find(int x){
23     if(x == par[x]) return x ;
24     return par[x] = find(par[x]) ;
25 }
26
27 int merge(int x, int y){
28     x = find(x) ;
29     y = find(y) ;
30
31     if(x == y) return 0 ;
32     if(sz[x] > sz[y]) swap(x, y) ;
33     par[x] = y ;
34     sz[y] += sz[x] ;
35
36     return 1 ;
37 }
```

```

48     seg[id].sum = arr[L] ;
49     return ;
50 }
51
52     int M = (L + R) >> 1 ;
53     build(L, M, lc) ;
54     build(M+1, R, rc) ;
55
56     pull(id) ;
57     seg[id].sz = seg[lc].sz + seg[rc].sz ;
58 }
59
60 void modify(int l, int r, LazyTag &tag,
61             int L=1, int R=n, int id=1){
62     if(l <= L && R <= r){
63         AddTag(id, tag) ;
64         return ;
65     }
66
67     push(id) ;
68     int M = (L + R) >> 1 ;
69     if(r <= M) modify(l, r, tag, L, M,
70                         lc) ;
71     else if(l > M) modify(l, r, tag, M+1,
72                           R, rc) ;
73     else{
74         modify(l, r, tag, L, M, lc) ;
75         modify(l, r, tag, M+1, R, rc) ;
76     }
77     pull(id) ;
78 }
79
80 int query(int l, int r, int L=1, int R=n,
81            int id=1){
82     if(l <= L && R <= r) return
83         seg[id].sum ;
84
85     push(id) ;
86     int M = (L + R) >> 1 ;
87     if(r <= M) return query(l, r, L, M,
88                             lc) ;
89     else if(l > M) return query(l, r,
90                                 M+1, R, rc) ;
91     else return query(l, r, L, M, lc) +
92             query(l, r, M+1, R, rc) ;
93 }
```

### 3.2 SegmentTree

```

1 #define lc (id << 1)
2 #define rc ((id << 1) | 1)
3
4 struct LazyTag{
5     // type 0 : increase val
6     // type 1 : set to val
7     // type 1 can overwrite type 0
8     int type ;
9     ll val ;
10 }
11
12 struct Node{
13     LazyTag tag ;
14     ll sum ;
15     int sz ;
16 }seg[Maxn << 2] ;
17
18 class SegmentTree{
19 private:
20     void pull(int id){
21         seg[id].sum = seg[lc].sum +
22             seg[rc].sum ;
23     }
24
25     void AddTag(int id, LazyTag &tag){
26         if(tag.type == 0){
27             seg[id].sum += tag.val *
28                 seg[id].sz ;
29             seg[id].tag.val += tag.val ;
30         }
31         else{
32             seg[id].sum = tag.val *
33                 seg[id].sz ;
34             seg[id].tag = {1, tag.val} ;
35         }
36     }
37
38     void push(int id){
39         AddTag(lc, seg[id].tag) ;
40         AddTag(rc, seg[id].tag) ;
41         seg[id].tag = {0, 0} ;
42     }
43
44 public:
45     void build(int L=1, int R=n, int id=1){
46         seg[id].sum = 0 ;
47         seg[id].tag = {0, 0} ;
48         seg[id].sz = 1 ;
49
50         if(L == R){
```

### 3.3 HLD

```

1 /* HLD */
2 int fa[Maxn], top[Maxn], son[Maxn],
3      sz[Maxn], dep[Maxn] = {0}, dfn[Maxn],
4      rk[Maxn], dfscnt = 0 ;
5
6 void dfs1(int u, int from){
7     fa[u] = from ;
8     dep[u] = dep[from] + 1 ;
9     sz[u] = 1 ;
10
11     for ( auto v : g[u] ) if(v != from){
12         dfs1(v, u) ;
13         sz[u] += sz[v] ;
14         if(son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v ;
15     }
16
17     void dfs2(int u, int t){
18         top[u] = t ;
19         dfn[u] = ++dfscnt ;
20         rk[dfscnt] = u ;
21         if(son[u] == -1) return ;
22
23         dfs2(son[u], t) ;
24     }
```

```

        for ( auto v : g[u] ) if(v != fa[u] && v
            != son[u]){
            dfs2(v, v) ;
        }
    }

/* Segment Tree */
#define lc (id << 1)
#define rc ((id << 1) | 1)

struct ColorSeg{
    int left, right, tot ;

ColorSeg operator+(const ColorSeg &o)
{
    const {
        if(tot == 0) return o ;
        if(o.tot == 0) return *this ;

        ColorSeg tmp ;
        tmp.left = left ;
        tmp.right = o.right ;
        tmp.tot = tot + o.tot - (right ==
            o.left) ;

        return tmp ;
    }
}

struct Node{
    ColorSeg color ;
    int tag ;
}seg[Maxn << 2] ;

class SegmentTree{
private:
    void pull(int id){
        // normal pull
    }

    void AddTag(int id, int tag){
        // normal AddTag
    }

    void push(int id){
        // normal push
    }

    void modify(int l, int r, int tag, int
        L=1, int R=n, int id=1){
        // normal modify
    }

    ColorSeg query(int l, int r, int L=1, int
        R=n, int id=1){
        // normal query
    }
public:
    void build(int L=1, int R=n, int id=1){
        // normal build
    }

    // update val from u to v (simple path)
    void update(int u, int v, int val){
        while(top[u] != top[v]){
            if(dep[top[u]] < dep[top[v]]) swap(u,
                v) ;
            modify(dfn[top[u]], dfn[u], val) ;
            u = fa[top[u]] ;
        }

        if(dep[u] < dep[v]) swap(u, v) ;
        modify(dfn[v], dfn[u], val) ;
    }

    // get sum from u to v (simple path)
    int get(int u, int v){
        pair<int, ColorSeg> U, V ;
        ColorSeg M ;
    }
}

```

### 3.4 PST

```

// Find range k-th largest number
struct Node{
    int sum, left, right ;
}seg[Maxn + 20 * Maxn] ;

class PersistentSegmentTree{
private:
    int n ;
    int cnt ;
    vector<int> version ;

    int build(int L, int R){
        int cur_cnt = cnt++ ;
        if(L == R){
            seg[cur_cnt] = {0, 0, 0} ;
            return cur_cnt ;
        }

        int M = (L + R) >> 1 ;
        int lc = build(L, M) ;
        int rc = build(M+1, R) ;

        seg[cur_cnt] = {0, lc, rc} ;
        return cur_cnt ;
    }
public:
    PersistentSegmentTree(int _n){
        n = _n ;
        cnt = 0 ;

        int root = build(1, n) ;
        version.push_back(root) ;
    }

    void update(int ver, int idx){
        auto upd = [&](auto &&self, const int
                      cur, int L, int R){
            int cur_cnt = cnt++ ;

            if(L == R){
                seg[cur_cnt] = {seg[cur].sum + 1, 0,
                               0} ;
                return cur_cnt ;
            }

            int M = (L + R) >> 1 ;
            int lc = seg[cur].left ;
            int rc = seg[cur].right ;
        }
    }
}

```

```

48     if(idx <= M) lc = self(self,
49         seg[cur].left, L, M) ;
50     else rc = self(self, seg[cur].right,
51         M+1, R) ;
52
53     seg[cur_cnt] = {seg[lc].sum +
54         seg[rc].sum, lc, rc} ;
55
56     return cur_cnt ;
57 };
58
59     int root = upd(upd, version[ver], 1, n) ;
60     version.push_back(root) ;
61 }
62
63 int query(int verL, int verR, int k){
64     auto qry = [&](auto &self, const int
65         cur_old, const int cur_new, int L,
66         int R){
67         if(L == R) return L ;
68
69         int old_l = seg[cur_old].left, old_r =
70             seg[cur_old].right ;
71         int new_l = seg[cur_new].left, new_r =
72             seg[cur_new].right ;
73
74         int dl = seg[new_l].sum -
75             seg[old_l].sum ;
76         int dr = seg[new_r].sum -
77             seg[old_r].sum ;
78
79         int M = (L + R) >> 1 ;
80
81         if(dl >= k) return self(self, old_l,
82             new_l, L, M) ;
83         k -= dl ;
84         return self(self, old_r, new_r, M+1,
85             R) ;
86     };
87
88     int idx = qry(qry, version[verL-1],
89         version[verR], 1, n) ;
90     return idx ;
91 }
92 }

```

```

27     }
28     node->end.insert(n) ;
29 }
30
31 void search(string &str){
32     TrieNode* node = root ;
33     for ( auto s : str ){
34         int path = s - 'a' ;
35         if(node->next[path] == nullptr)
36             return ;
37         node = node->next[path] ;
38 }
39
40     int flg = 0 ;
41     for ( auto n : node->end ){
42         if(flg) cout << " " ;
43         else flg = 1 ;
44
45         cout << n ;
46 }
47
48 void clear(TrieNode* node) {
49     if (!node) return ;
50     for (int i = 0; i < 26; i++) {
51         if (node->next[i]) {
52             clear(node->next[i]) ;
53         }
54     }
55     delete node ;
56 }
57
58 ~Trie(){
59     clear(root) ;
60 }
61 };

```

## 3.6 BIT 單修區查

```

1 // 單點修改 區間查詢
2 #define lowbit(x) (x & -x)
3
4 int BIT[MAX_SIZE] ;
5 int n ;
6
7 void modify(int idx, int val){
8     for ( ; idx <= n ; idx += lowbit(idx) ){
9         BIT[idx] += val ;
10    }
11 }
12
13 // ans: query(R) - query(L-1)
14 int query(int idx){
15     int sum = 0 ;
16     for ( ; idx ; idx -= lowbit(idx) ){
17         sum += BIT[idx] ;
18    }
19 }
20
21 void init(){
22     memset(BIT, 0, sizeof(BIT)) ;
23 }

```

## 3.7 BIT 區修單查

```

1 // 區間修改， 單點查詢
2 #define lowbit(x) (x & -x)
3
4 int BIT[MAX_SIZE] ;
5 int n ;
6
7 void modify(int idx, int val){
8     for ( ; idx <= n ; idx += lowbit(idx) ){
9         BIT[idx] += val ;

```

```

10    }
11 }
12
13 // ans: query(i)
14 int query(int idx){
15     int sum = 0 ;
16     for ( ; idx ; idx -= lowbit(idx) ){
17         sum += BIT[idx] ;
18    }
19
20     return sum ;
21 }
22
23 void init(){
24     memset(BIT, 0, sizeof(BIT)) ;
25 }
26
27 void build(){
28     arr[0] = 0 ;
29     for ( int i=1 ; i<=n ; i++ ) modify(i,
30           arr[i] - arr[i-1] ) ;
31 }
32
33 // usage
34 // add val
35 modify(L, x) ;
36 modify(R+1, -x) ;

```

## 3.8 BIT 區修區查

```

1 // 區間修改 區間查詢
2 #define lowbit(x) x & -x
3
4 ll BIT1[MAXN], BIT2[MAXN] ;
5
6 void update(int idx, ll val){
7     for ( int i=idx ; i<=n ; i+=lowbit(i) ){
8         BIT1[i] += val ;
9         BIT2[i] += idx * val ;
10    }
11 }
12
13 // range query: query(R) - query(X - 1)
14 ll query(int idx){
15     ll sum1 = 0, sum2 = 0 ;
16     for ( int i=idx ; i ; i-=lowbit(i) ){
17         sum1 += BIT1[i] ;
18         sum2 += BIT2[i] ;
19    }
20
21     return (idx + 1) * sum1 - sum2 ;
22 }
23
24 void build(){
25     for ( int i=1 ; i<=n ; i++ ){
26         update(i, arr[i] - arr[i-1] ) ;
27    }
28 }
29
30 void usage(){
31     update(L, x) ;
32     update(R+1, -x) ;
33 }
34
35 void init(){
36     memset(BIT1, 0, sizeof(BIT1)) ;
37     memset(BIT2, 0, sizeof(BIT2)) ;
38 }

```

## 4 Graph

### 4.1 cut vertex AND bridges

```

1 int dfn[Maxn] = {-1}, low[Maxn] = {-1},
2     dfscnt ;
3
4 void dfs(int u, int fa){
5     dfn[u] = low[u] = ++dfscnt ;
6     int child = 0 ;
7
8     for ( auto v : g[u] ) if(v != fa){
9         if(dfn[v] == -1){
10             child++ ;
11             dfs(v, u) ;
12             low[u] = min(low[u], low[v]) ;
13
14             if(low[v] >= dfn[u]){
15                 // this edge is a bridge
16             }
17
18             if(u != fa && low[v] >= dfn[u]){
19                 // this node v is a articulation point
20             }
21
22             else low[u] = min(low[u], dfn[v]) ;
23
24         if(u == fa && child > 1){
25             // this node u is a articulation point
26         }
27     }
28 }

```

## 4.2 SCC - Tarjan

```

1 vector<int> scc[Maxn] ;
2 int dfn[Maxn], low[Maxn], sccId[Maxn],
3     dfscnt = 0, cnt_scc = 0 ;
4 stack<int> st ;
5 bitset<Maxn> inSt, vis ;
6
7 void dfs(int u, int from){
8     dfn[u] = low[u] = ++dfscnt ;
9     st.push(u) ;
10    inSt[u] = 1 ;
11
12    for ( auto v : g[u] ){
13        if(!inSt[v] && dfn[v] != -1) continue ;
14        if(dfn[v] == -1) dfs(v, u) ;
15        low[u] = min(low[u], low[v]) ;
16    }
17
18    if(dfn[u] == low[u]){
19        cnt_scc++ ;
20        int x ;
21
22        do{
23            x = st.top() ;
24            st.pop() ;
25
26            inSt[x] = 0 ;
27            sccId[x] = cnt_scc ;
28            scc[cnt_scc].push_back(x) ;
29
30        } while(x != u) ;
31    }
32
33 // SCC to DAG (after dfs)
34 vector<int> dag[Maxn] ;
35
36 void scc_to_dag(){
37     vector<int> dag[Maxn] ;
38     for ( int u=1 ; u<=n ; u++ ){
39         for ( auto v : g[u] ){
40             if(sccId[u] != sccId[v]){
41                 dag[sccId[u]].push_back(sccId[v]) ;
42             }
43         }
44     }
45 }

```

```

46     void init(){
47         memset(dfn, -1, sizeof(dfn)) ;
48         memset(low, -1, sizeof(low)) ;
49     }
50 }
51 int main(){
52     init() ;
53     input() ;
54     for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
55         dfs(i, i) ;
56     }
57 }
```

### 4.3 BCC - Tarjan

```

1 struct Edge{
2     int v, next ;
3 }e[Maxm << 1] ;
4 int head[Maxm], tot = 1 ;
5
6 void add(int u, int v){
7     e[++tot] = {v, head[u]} ;
8     head[u] = tot ;
9     e[++tot] = {u, head[v]} ;
10    head[v] = tot ;
11 }
12
13 bitset<Maxm << 1> bz ;
14 vector<vector<int>> bcc ;
15 int dfn_cnt = 0, dfn[Maxn], low[Maxn],
16     vis_bcc[Maxn], bcc_cnt = 0 ;
17
18 void dfs1(int u, int from){
19     dfn[u] = low[u] = ++dfn_cnt ;
20
21     for ( int i=head[u] ; i!= -1 ; i=e[i].next )
22         int v = e[i].v ;
23
24         if(dfn[v] == -1){
25             dfs1(v, i) ;
26             if(dfn[u] < low[v]) bz[i] = bz[i^1] =
27                 1 ;
28             low[u] = min(low[u], low[v]) ;
29         }
30         else if(i != (from ^ 1)) low[u] =
31             min(low[u], dfn[v]) ;
32     }
33
34 void dfs2(int u, int id){
35     vis_bcc[u] = id ;
36     bcc[id].push_back(u) ;
37
38     for ( int i=head[u] ; i!= -1 ; i=e[i].next )
39         int v = e[i].v ;
40
41         if(vis_bcc[v] != -1 || bz[i]) continue ;
42         dfs2(v, id) ;
43     }
44
45 void init(){
46     memset(dfn, -1, sizeof(dfn)) ;
47     memset(head, -1, sizeof(head)) ;
48     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
49 }
50
51 int main(){
52     init() ;
53     input() ;
54     for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
55         dfs1(i, 0) ;
56     }
57 }
```

### 4.4 Convex

```

1 struct Coordinate{
2     long long x, y ;
3
4     friend bool operator<(const Coordinate&a,
5         const Coordinate& b){
6         if(a.x == b.x) return a.y < b.y ;
7         return a.x < b.x ;
8     }
9
10    friend bool operator==(const Coordinate&
11        a, const Coordinate& b){
12        return a.x == b.x && a.y == b.y ;
13    }
14
15    vector<Coordinate> nodes ;
16
17    long long cross(const Coordinate& o, const
18        Coordinate& a, const Coordinate& b){
19        return (a.x - o.x) * (b.y - o.y) - (a.y -
20            o.y) * (b.x - o.x) ;
21    }
22
23    void input(){
24        nodes.clear() ;
25
26        int n, x, y ;
27        char c ;
28        cin >> n ;
29
30        for ( int i=0 ; i<n ; i++ ){
31            cin >> x >> y >> c ;
32            if(c == 'Y') nodes.push_back({x, y}) ;
33        }
34
35        void monotone(){
36            sort(nodes.begin(), nodes.end()) ;
37
38            int n = unique(nodes.begin(), nodes.end()) -
39                nodes.begin() ;
40
41            vector<Coordinate> ch(n+1) ;
42
43            int m = 0 ;
44
45            for ( int i=0 ; i<n ; i++ ){
46                while(m > 1 && cross(ch[m-2], ch[m-1],
47                    nodes[i]) < 0) m-- ;
48                ch[m++] = nodes[i] ;
49            }
50
51            for ( int i=n-2, t=m ; i>=0 ; i-- ){
52                while(m > t && cross(ch[m-2], ch[m-1],
53                    nodes[i]) < 0) m-- ;
54                ch[m++] = nodes[i] ;
55            }
56
57            if(n > 1) m-- ;
58            cout << m << endl ;
59
60            for ( int i=0 ; i<m ; i++ ) cout <<
61                ch[i].x << " " << ch[i].y << endl ;
62        }
63    }
64 }
```

### 4.5 Max Flow

```

1 struct Edge{
2     int v, cap, next ;
3 };
4
5 class MaxFlow{
6     private:
7         int N, S, T ;
8         vector<Edge> e ;
9         vector<int> head, cur, dep ;
10
11     bool bfs(){
12         queue<int> q ;
13         for ( int i=0 ; i<=N ; i++ ){
14             cur[i] = head[i] ;
15             dep[i] = -1 ;
16         }
17
18         q.push(S) ;
19         dep[S] = 0 ;
20
21         while(!q.empty()){
22             int u = q.front() ; q.pop() ;
23
24             for ( int i=head[u] ; i!= -1 ;
25                 i=e[i].next ){
26                 int v = e[i].v ;
27                 if(dep[v] == -1 && e[i].cap > 0){
28                     dep[v] = dep[u] + 1 ;
29                     if(v == T) return 1 ;
30                     q.push(v) ;
31                 }
32             }
33
34             return 0 ;
35         }
36
37         int dfs(int u, int flow){
38             if(u == T) return flow ;
39             int d, rest = 0 ;
40
41             for ( int &i=cur[u] ; i!= -1 ;
42                 i=e[i].next ){
43                 int v = e[i].v ;
44                 if(dep[v] == dep[u] + 1 && e[i].cap >
45                     0){
46                     d = dfs(v, min(flow - rest,
47                         e[i].cap)) ;
48
49                     if(d > 0){
50                         e[i].cap -= d ;
51                         e[i^1].cap += d ;
52                         rest += d ;
53
54                         if(rest == flow) break ;
55                     }
56
57                     if(rest != flow) dep[u] = -1 ;
58                     return rest ;
59                 }
60             }
61             MaxFlow(int n, int s, int t){
62                 N = n ; S = s ; T = t ;
63                 e.reserve(n*n) ;
64                 head.assign(n+1, -1) ;
65                 cur.resize(n+1) ;
66                 dep.resize(n+1) ;
67             }
68
69             void AddEdge(int u, int v, int cap){
70                 e.push_back({v, cap, head[u]}) ;
71                 head[u] = e.size() - 1 ;
72                 e.push_back({u, 0, head[v]}) ;
73                 head[v] = e.size() - 1 ;
74             }
75         }
76     }
77 }
```

```

73 }
74
75 int run(){
76     int ans = 0 ;
77     while(bfs()){
78         ans += dfs(S, 0x3f3f3f3f) ;
79     }
80     return ans ;
81 }
82 };

```

## 4.6 min cut max flow

```

1 struct Edge{
2     int v, cap, cost , next ;
3 };
4
5 using pii = pair<int, int> ;
6 class MCMF{
7 private:
8     int N, s, t, tot ;
9     vector<Edge> e ;
10    vector<int> head ;
11 public:
12    MCMF(int n, int _s, int _t){
13        N = n ;
14        s = _s ;
15        t = _t ;
16        e.resize(n*n + 5) ;
17        head.assign(n+5, -1) ;
18        tot = -1 ;
19    }
20
21    void AddEdge(int u, int v, int cap, int cost){
22        e[++tot] = {v, cap, cost, head[u]} ;
23        head[u] = tot ;
24        e[++tot] = {u, 0, -cost, head[v]} ;
25        head[v] = tot ;
26    }
27
28    int run(){
29        vector<int> dis(N+1), pot(N+1, 0),
30            preE(N+1) ;
31        int flow = 0, cost = 0 ;
32
33        auto dijkstra = [&](){
34            fill(dis.begin(), dis.end(), INF) ;
35            priority_queue<pii, vector<pii>,
36                greater<pii>> pq ;
37            dis[s] = 0 ;
38            pq.push({0, s}) ;
39
40            while(!pq.empty()){
41                auto [d, u] = pq.top() ; pq.pop() ;
42                if(d > dis[u]) continue ;
43                for ( int i=head[u] ; i!= -1 ;
44                    i=e[i].next ){
45                    int v = e[i].v, cap = e[i].cap, w =
46                        e[i].cost ;
47                    if(cap && dis[v] > d + w + pot[u] -
48                        pot[v]){
49                        dis[v] = d + w + pot[u] - pot[v] ;
50                        preE[v] = i ;
51                        pq.push({dis[v], v}) ;
52                    }
53                }
54
55                return dis[t] != INF ;
56            }
57
58            while(dijkistra()){
59                for ( int v=1 ; v<=N ; v++ ) if(dis[v]
60                    < INF){
61                        pot[v] += dis[v] ;
62                    }
63            }
64        }
65    }
66
67    int run(){
68        int ans = 0 ;
69        while(bfs()){
70            ans += dfs(S, 0x3f3f3f3f) ;
71        }
72        return ans ;
73    }
74
75    void add_pattern(const string &pattern){
76        int u = 0 ;
77        for ( auto ch : pattern ){
78            int v = ch - base ;
79
80            if(tree[u][v] == 0) tree[u][v] =
81                new_node() ;
82            u = tree[u][v] ;
83        }
84
85        end.push_back(u) ;
86    }
87
88    void build(){
89        queue<int> q ;
90        order.clear() ;
91        order.push_back(0) ;
92
93        for ( int i=0 ; i<alpha ; i++ )
94            if(tree[0][i] > 0){
95                q.push(tree[0][i]) ;
96            }
97
98        while(!q.empty()){
99            int u = q.front() ; q.pop() ;
100           order.push_back(u) ;
101
102           for ( int i=0 ; i<alpha ; i++ ){
103               if(tree[u][i] == 0) tree[u][i] =
104                   tree[fail[u]][i] ;
105               else{
106                   fail[tree[u][i]] = tree[fail[u]][i] ;
107                   ;
108                   q.push(tree[u][i]) ;
109               }
110           }
111       }
112
113       vector<int> count_per_pattern(const string
114                                     &text) const {
115           int u = 0 ;
116           vector<int> vis(tree.size(), 0) ;
117
118           for ( char ch : text ){
119               u = tree[u][ch - base] ;
120               vis[u]++;
121           }
122
123           for ( int i=order.size()-1 ; i>=1 ; i-- )
124               int x = order[i] ;
125               vis[fail[x]] += vis[x] ;
126           }
127
128           vector<int> ans(end.size(), 0) ;
129           for ( int id=0 ; id<end.size() ; id++ )
130               ans[id] = vis[order[id]] ;
131       }
132   }
133
134   void clear(){
135       fail.assign(1, 0) ;
136       order.clear() ;
137       end.clear() ;
138       tree.assign(1, vector<int>(alpha, 0)) ;
139   }
140
141   AAutomation(int _base='a', int _alpha=26)
142     : base(_base), alpha(_alpha) {
143         clear() ;
144     }
145
146   void clear(){
147       fail.assign(1, 0) ;
148       order.clear() ;
149       end.clear() ;
150       tree.assign(1, vector<int>(alpha, 0)) ;
151   }
152
153   void add_pattern(const string &pattern){
154       int u = 0 ;
155       for ( auto ch : pattern ){
156           int v = ch - base ;
157
158           if(tree[u][v] == 0) tree[u][v] =
159               new_node() ;
160           u = tree[u][v] ;
161       }
162
163       end.push_back(u) ;
164   }
165
166   void build(){
167       queue<int> q ;
168       order.clear() ;
169       order.push_back(0) ;
170
171       for ( int i=0 ; i<alpha ; i++ )
172           if(tree[0][i] > 0){
173               q.push(tree[0][i]) ;
174           }
175
176       while(!q.empty()){
177           int u = q.front() ; q.pop() ;
178           order.push_back(u) ;
179
180           for ( int i=0 ; i<alpha ; i++ ){
181               if(tree[u][i] == 0) tree[u][i] =
182                   tree[fail[u]][i] ;
183               else{
184                   fail[tree[u][i]] = tree[fail[u]][i] ;
185                   ;
186                   q.push(tree[u][i]) ;
187               }
188           }
189       }
190
191       vector<int> count_per_pattern(const string
192                                     &text) const {
193           int u = 0 ;
194           vector<int> vis(tree.size(), 0) ;
195
196           for ( char ch : text ){
197               u = tree[u][ch - base] ;
198               vis[u]++;
199           }
200
201           for ( int i=order.size()-1 ; i>=1 ; i-- )
202               int x = order[i] ;
203               vis[fail[x]] += vis[x] ;
204           }
205
206           vector<int> ans(end.size(), 0) ;
207           for ( int id=0 ; id<end.size() ; id++ )
208               ans[id] = vis[order[id]] ;
209       }
210   }
211
212   void clear(){
213       fail.assign(1, 0) ;
214       order.clear() ;
215       end.clear() ;
216       tree.assign(1, vector<int>(alpha, 0)) ;
217   }
218
219   AAutomation(int _base='a', int _alpha=26)
220     : base(_base), alpha(_alpha) {
221         clear() ;
222     }
223
224   void clear(){
225       fail.assign(1, 0) ;
226       order.clear() ;
227       end.clear() ;
228       tree.assign(1, vector<int>(alpha, 0)) ;
229   }
230
231   void add_pattern(const string &pattern){
232       int u = 0 ;
233       for ( auto ch : pattern ){
234           int v = ch - base ;
235
236           if(tree[u][v] == 0) tree[u][v] =
237               new_node() ;
238           u = tree[u][v] ;
239       }
240
241       end.push_back(u) ;
242   }
243
244   void build(){
245       queue<int> q ;
246       order.clear() ;
247       order.push_back(0) ;
248
249       for ( int i=0 ; i<alpha ; i++ )
250           if(tree[0][i] > 0){
251               q.push(tree[0][i]) ;
252           }
253
254       while(!q.empty()){
255           int u = q.front() ; q.pop() ;
256           order.push_back(u) ;
257
258           for ( int i=0 ; i<alpha ; i++ ){
259               if(tree[u][i] == 0) tree[u][i] =
260                   tree[fail[u]][i] ;
261               else{
262                   fail[tree[u][i]] = tree[fail[u]][i] ;
263                   ;
264                   q.push(tree[u][i]) ;
265               }
266           }
267       }
268
269       vector<int> count_per_pattern(const string
270                                     &text) const {
271           int u = 0 ;
272           vector<int> vis(tree.size(), 0) ;
273
274           for ( char ch : text ){
275               u = tree[u][ch - base] ;
276               vis[u]++;
277           }
278
279           for ( int i=order.size()-1 ; i>=1 ; i-- )
280               int x = order[i] ;
281               vis[fail[x]] += vis[x] ;
282           }
283
284           vector<int> ans(end.size(), 0) ;
285           for ( int id=0 ; id<end.size() ; id++ )
286               ans[id] = vis[order[id]] ;
287       }
288   }
289
290   void clear(){
291       fail.assign(1, 0) ;
292       order.clear() ;
293       end.clear() ;
294       tree.assign(1, vector<int>(alpha, 0)) ;
295   }
296
297   AAutomation(int _base='a', int _alpha=26)
298     : base(_base), alpha(_alpha) {
299         clear() ;
300     }
301
302   void clear(){
303       fail.assign(1, 0) ;
304       order.clear() ;
305       end.clear() ;
306       tree.assign(1, vector<int>(alpha, 0)) ;
307   }
308
309   void add_pattern(const string &pattern){
310       int u = 0 ;
311       for ( auto ch : pattern ){
312           int v = ch - base ;
313
314           if(tree[u][v] == 0) tree[u][v] =
315               new_node() ;
316           u = tree[u][v] ;
317       }
318
319       end.push_back(u) ;
320   }
321
322   void build(){
323       queue<int> q ;
324       order.clear() ;
325       order.push_back(0) ;
326
327       for ( int i=0 ; i<alpha ; i++ )
328           if(tree[0][i] > 0){
329               q.push(tree[0][i]) ;
330           }
331
332       while(!q.empty()){
333           int u = q.front() ; q.pop() ;
334           order.push_back(u) ;
335
336           for ( int i=0 ; i<alpha ; i++ ){
337               if(tree[u][i] == 0) tree[u][i] =
338                   tree[fail[u]][i] ;
339               else{
340                   fail[tree[u][i]] = tree[fail[u]][i] ;
341                   ;
342                   q.push(tree[u][i]) ;
343               }
344           }
345       }
346
347       vector<int> count_per_pattern(const string
348                                     &text) const {
349           int u = 0 ;
350           vector<int> vis(tree.size(), 0) ;
351
352           for ( char ch : text ){
353               u = tree[u][ch - base] ;
354               vis[u]++;
355           }
356
357           for ( int i=order.size()-1 ; i>=1 ; i-- )
358               int x = order[i] ;
359               vis[fail[x]] += vis[x] ;
360           }
361
362           vector<int> ans(end.size(), 0) ;
363           for ( int id=0 ; id<end.size() ; id++ )
364               ans[id] = vis[order[id]] ;
365       }
366   }
367
368   void clear(){
369       fail.assign(1, 0) ;
370       order.clear() ;
371       end.clear() ;
372       tree.assign(1, vector<int>(alpha, 0)) ;
373   }
374
375   AAutomation(int _base='a', int _alpha=26)
376     : base(_base), alpha(_alpha) {
377         clear() ;
378     }
379
380   void clear(){
381       fail.assign(1, 0) ;
382       order.clear() ;
383       end.clear() ;
384       tree.assign(1, vector<int>(alpha, 0)) ;
385   }
386
387   void add_pattern(const string &pattern){
388       int u = 0 ;
389       for ( auto ch : pattern ){
390           int v = ch - base ;
391
392           if(tree[u][v] == 0) tree[u][v] =
393               new_node() ;
394           u = tree[u][v] ;
395       }
396
397       end.push_back(u) ;
398   }
399
400   void build(){
401       queue<int> q ;
402       order.clear() ;
403       order.push_back(0) ;
404
405       for ( int i=0 ; i<alpha ; i++ )
406           if(tree[0][i] > 0){
407               q.push(tree[0][i]) ;
408           }
409
410       while(!q.empty()){
411           int u = q.front() ; q.pop() ;
412           order.push_back(u) ;
413
414           for ( int i=0 ; i<alpha ; i++ ){
415               if(tree[u][i] == 0) tree[u][i] =
416                   tree[fail[u]][i] ;
417               else{
418                   fail[tree[u][i]] = tree[fail[u]][i] ;
419                   ;
420                   q.push(tree[u][i]) ;
421               }
422           }
423       }
424
425       vector<int> count_per_pattern(const string
426                                     &text) const {
427           int u = 0 ;
428           vector<int> vis(tree.size(), 0) ;
429
430           for ( char ch : text ){
431               u = tree[u][ch - base] ;
432               vis[u]++;
433           }
434
435           for ( int i=order.size()-1 ; i>=1 ; i-- )
436               int x = order[i] ;
437               vis[fail[x]] += vis[x] ;
438           }
439
440           vector<int> ans(end.size(), 0) ;
441           for ( int id=0 ; id<end.size() ; id++ )
442               ans[id] = vis[order[id]] ;
443       }
444   }
445
446   void clear(){
447       fail.assign(1, 0) ;
448       order.clear() ;
449       end.clear() ;
450       tree.assign(1, vector<int>(alpha, 0)) ;
451   }
452
453   AAutomation(int _base='a', int _alpha=26)
454     : base(_base), alpha(_alpha) {
455         clear() ;
456     }
457
458   void clear(){
459       fail.assign(1, 0) ;
460       order.clear() ;
461       end.clear() ;
462       tree.assign(1, vector<int>(alpha, 0)) ;
463   }
464
465   void add_pattern(const string &pattern){
466       int u = 0 ;
467       for ( auto ch : pattern ){
468           int v = ch - base ;
469
470           if(tree[u][v] == 0) tree[u][v] =
471               new_node() ;
472           u = tree[u][v] ;
473       }
474
475       end.push_back(u) ;
476   }
477
478   void build(){
479       queue<int> q ;
480       order.clear() ;
481       order.push_back(0) ;
482
483       for ( int i=0 ; i<alpha ; i++ )
484           if(tree[0][i] > 0){
485               q.push(tree[0][i]) ;
486           }
487
488       while(!q.empty()){
489           int u = q.front() ; q.pop() ;
490           order.push_back(u) ;
491
492           for ( int i=0 ; i<alpha ; i++ ){
493               if(tree[u][i] == 0) tree[u][i] =
494                   tree[fail[u]][i] ;
495               else{
496                   fail[tree[u][i]] = tree[fail[u]][i] ;
497                   ;
498                   q.push(tree[u][i]) ;
499               }
500           }
501       }
502
503       vector<int> count_per_pattern(const string
504                                     &text) const {
505           int u = 0 ;
506           vector<int> vis(tree.size(), 0) ;
507
508           for ( char ch : text ){
509               u = tree[u][ch - base] ;
510               vis[u]++;
511           }
512
513           for ( int i=order.size()-1 ; i>=1 ; i-- )
514               int x = order[i] ;
515               vis[fail[x]] += vis[x] ;
516           }
517
518           vector<int> ans(end.size(), 0) ;
519           for ( int id=0 ; id<end.size() ; id++ )
520               ans[id] = vis[order[id]] ;
521       }
522   }
523
524   void clear(){
525       fail.assign(1, 0) ;
526       order.clear() ;
527       end.clear() ;
528       tree.assign(1, vector<int>(alpha, 0)) ;
529   }
530
531   AAutomation(int _base='a', int _alpha=26)
532     : base(_base), alpha(_alpha) {
533         clear() ;
534     }
535
536   void clear(){
537       fail.assign(1, 0) ;
538       order.clear() ;
539       end.clear() ;
540       tree.assign(1, vector<int>(alpha, 0)) ;
541   }
542
543   void add_pattern(const string &pattern){
544       int u = 0 ;
545       for ( auto ch : pattern ){
546           int v = ch - base ;
547
548           if(tree[u][v] == 0) tree[u][v] =
549               new_node() ;
550           u = tree[u][v] ;
551       }
552
553       end.push_back(u) ;
554   }
555
556   void build(){
557       queue<int> q ;
558       order.clear() ;
559       order.push_back(0) ;
560
561       for ( int i=0 ; i<alpha ; i++ )
562           if(tree[0][i] > 0){
563               q.push(tree[0][i]) ;
564           }
565
566       while(!q.empty()){
567           int u = q.front() ; q.pop() ;
568           order.push_back(u) ;
569
570           for ( int i=0 ; i<alpha ; i++ ){
571               if(tree[u][i] == 0) tree[u][i] =
572                   tree[fail[u]][i] ;
573               else{
574                   fail[tree[u][i]] = tree[fail[u]][i] ;
575                   ;
576                   q.push(tree[u][i]) ;
577               }
578           }
579       }
580
581       vector<int> count_per_pattern(const string
582                                     &text) const {
583           int u = 0 ;
584           vector<int> vis(tree.size(), 0) ;
585
586           for ( char ch : text ){
587               u = tree[u][ch - base] ;
588               vis[u]++;
589           }
590
591           for ( int i=order.size()-1 ; i>=1 ; i-- )
592               int x = order[i] ;
593               vis[fail[x]] += vis[x] ;
594           }
595
596           vector<int> ans(end.size(), 0) ;
597           for ( int id=0 ; id<end.size() ; id++ )
598               ans[id] = vis[order[id]] ;
599       }
600   }
601
602   void clear(){
603       fail.assign(1, 0) ;
604       order.clear() ;
605       end.clear() ;
606       tree.assign(1, vector<int>(alpha, 0)) ;
607   }
608
609   AAutomation(int _base='a', int _alpha=26)
610     : base(_base), alpha(_alpha) {
611         clear() ;
612     }
613
614   void clear(){
615       fail.assign(1, 0) ;
616       order.clear() ;
617       end.clear() ;
618       tree.assign(1, vector<int>(alpha, 0)) ;
619   }
620
621   void add_pattern(const string &pattern){
622       int u = 0 ;
623       for ( auto ch : pattern ){
624           int v = ch - base ;
625
626           if(tree[u][v] == 0) tree[u][v] =
627               new_node() ;
628           u = tree[u][v] ;
629       }
630
631       end.push_back(u) ;
632   }
633
634   void build(){
635       queue<int> q ;
636       order.clear() ;
637       order.push_back(0) ;
638
639       for ( int i=0 ; i<alpha ; i++ )
640           if(tree[0][i] > 0){
641               q.push(tree[0][i]) ;
642           }
643
644       while(!q.empty()){
645           int u = q.front() ; q.pop() ;
646           order.push_back(u) ;
647
648           for ( int i=0 ; i<alpha ; i++ ){
649               if(tree[u][i] == 0) tree[u][i] =
650                   tree[fail[u]][i] ;
651               else{
652                   fail[tree[u][i]] = tree[fail[u]][i] ;
653                   ;
654                   q.push(tree[u][i]) ;
655               }
656           }
657       }
658
659       vector<int> count_per_pattern(const string
660                                     &text) const {
661           int u = 0 ;
662           vector<int> vis(tree.size(), 0) ;
663
664           for ( char ch : text ){
665               u = tree[u][ch - base] ;
666               vis[u]++;
667           }
668
669           for ( int i=order.size()-1 ; i>=1 ; i-- )
670               int x = order[i] ;
671               vis[fail[x]] += vis[x] ;
672           }
673
674           vector<int> ans(end.size(), 0) ;
675           for ( int id=0 ; id<end.size() ; id++ )
676               ans[id] = vis[order[id]] ;
677       }
678   }
679
680   void clear(){
681       fail.assign(1, 0) ;
682       order.clear() ;
683       end.clear() ;
684       tree.assign(1, vector<int>(alpha, 0)) ;
685   }
686
687   AAutomation(int _base='a', int _alpha=26)
688     : base(_base), alpha(_alpha) {
689         clear() ;
690     }
691
692   void clear(){
693       fail.assign(1, 0) ;
694       order.clear() ;
695       end.clear() ;
696       tree.assign(1, vector<int>(alpha, 0)) ;
697   }
698
699   void add_pattern(const string &pattern){
700       int u = 0 ;
701       for ( auto ch : pattern ){
702           int v = ch - base ;
703
704           if(tree[u][v] == 0) tree[u][v] =
705               new_node() ;
706           u = tree[u][v] ;
707       }
708
709       end.push_back(u) ;
710   }
711
712   void build(){
713       queue<int> q ;
714       order.clear() ;
715       order.push_back(0) ;
716
717       for ( int i=0 ; i<alpha ; i++ )
718           if(tree[0][i] > 0){
719               q.push(tree[0][i]) ;
720           }
721
722       while(!q.empty()){
723           int u = q.front() ; q.pop() ;
724           order.push_back(u) ;
725
726           for ( int i=0 ; i<alpha ; i++ ){
727               if(tree[u][i] == 0) tree[u][i] =
728                   tree[fail[u]][i] ;
729               else{
730                   fail[tree[u][i]] = tree[fail[u]][i] ;
731                   ;
732                   q.push(tree[u][i]) ;
733               }
734           }
735       }
736
737       vector<int> count_per_pattern(const string
738                                     &text) const {
739           int u = 0 ;
740           vector<int> vis(tree.size(), 0) ;
741
742           for ( char ch : text ){
743               u = tree[u][ch - base] ;
744               vis[u]++;
745           }
746
747           for ( int i=order.size()-1 ; i>=1 ; i-- )
748               int x = order[i] ;
749               vis[fail[x]] += vis[x] ;
750           }
751
752           vector<int> ans(end.size(), 0) ;
753           for ( int id=0 ; id<end.size() ; id++ )
754               ans[id] = vis[order[id]] ;
755       }
756   }
757
758   void clear(){
759       fail.assign(1, 0) ;
760       order.clear() ;
761       end.clear() ;
762       tree.assign(1, vector<int>(alpha, 0)) ;
763   }
764
765   AAutomation(int _base='a', int _alpha=26)
766     : base(_base), alpha(_alpha) {
767         clear() ;
768     }
769
770   void clear(){
771       fail.assign(1, 0) ;
772       order.clear() ;
773       end.clear() ;
774       tree.assign(1, vector<int>(alpha, 0)) ;
775   }
776
777   void add_pattern(const string &pattern){
778       int u = 0 ;
779       for ( auto ch : pattern ){
780           int v = ch - base ;
781
782           if(tree[u][v] == 0) tree[u][v] =
783               new_node() ;
784           u = tree[u][v] ;
785       }
786
787       end.push_back(u) ;
788   }
789
790   void build(){
791       queue<int> q ;
792       order.clear() ;
793       order.push_back(0) ;
794
795       for ( int i=0 ; i<alpha ; i++ )
796           if(tree[0][i] > 0){
797               q.push(tree[0][i]) ;
798           }
799
800       while(!q.empty()){
801           int u = q.front() ; q.pop() ;
802           order.push_back(u) ;
803
804           for ( int i=0 ; i<alpha ; i++ ){
805               if(tree[u][i] == 0) tree[u][i] =
806                   tree[fail[u]][i] ;
807               else{
808                   fail[tree[u][i]] = tree[fail[u]][i] ;
809                   ;
810                   q.push(tree[u][i]) ;
811               }
812           }
813       }
814
815       vector<int> count_per_pattern(const string
816                                     &text) const {
817           int u = 0 ;
818           vector<int> vis(tree.size(), 0) ;
819
820           for ( char ch : text ){
821               u = tree[u][ch - base] ;
822               vis[u]++;
823           }
824
825           for ( int i=order.size()-1 ; i>=1 ; i-- )
826               int x = order[i] ;
827               vis[fail[x]] += vis[x] ;
828           }
829
830           vector<int> ans(end.size(), 0) ;
831           for ( int id=0 ; id<end.size() ; id++ )
832               ans[id] = vis[order[id]] ;
833       }
834   }
835
836   void clear(){
837       fail.assign(1, 0) ;
838       order.clear() ;
839       end.clear() ;
840       tree.assign(1, vector<int>(alpha, 0)) ;
841   }
842
843   AAutomation(int _base='a', int _alpha=26)
844     : base(_base), alpha(_alpha) {
845         clear() ;
846     }
847
848   void clear(){
849       fail.assign(1, 0) ;
850       order.clear() ;
851       end.clear() ;
852       tree.assign(1, vector<int>(alpha, 0)) ;
853   }
854
855   void add_pattern(const string &pattern){
856       int u = 0 ;
857       for ( auto ch : pattern ){
858           int v = ch - base ;
859
860           if(tree[u][v] == 0) tree[u][v] =
861               new_node() ;
862           u = tree[u][v] ;
863       }
864
865       end.push_back(u) ;
866   }
867
868   void build(){
869       queue<int> q ;
870       order.clear() ;
871       order.push_back(0) ;
872
873       for ( int i=0 ; i<alpha ; i++ )
874           if(tree[0][i] > 0){
875               q.push(tree[0][i]) ;
876           }
877
878       while(!q.empty()){
879           int u = q.front() ; q.pop() ;
880           order.push_back(u) ;
881
882           for ( int i=0 ; i<alpha ; i++ ){
883               if(tree[u][i] == 0) tree[u][i] =
884                   tree[fail[u]][i] ;
885               else{
886                   fail[tree[u][i]] = tree[fail[u]][i] ;
887                   ;
888                   q.push(tree[u][i]) ;
889               }
890           }
891       }
892
893       vector<int> count_per_pattern(const string
894                                     &text) const {
895           int u = 0 ;
896           vector<int> vis(tree.size(), 0) ;
897
898           for ( char ch : text ){
899               u = tree[u][ch - base] ;
900               vis[u]++;
901           }
902
903           for ( int i=order.size()-1 ; i>=1 ; i-- )
904               int x = order[i] ;
905               vis[fail[x]] += vis[x] ;
906           }
907
908           vector<int> ans(end.size(), 0) ;
909           for ( int id=0 ; id<end.size() ; id++ )
910               ans[id] = vis[order[id]] ;
911       }
912   }
913
914   void clear(){
915       fail.assign(1, 0) ;
916       order.clear() ;
917       end.clear() ;
918       tree.assign(1, vector<int>(alpha, 0)) ;
919   }
920
921   AAutomation(int _base='a', int _alpha=26)
922     : base(_base), alpha(_alpha) {
923         clear() ;
924     }
925
926   void clear(){
927       fail.assign(1, 0) ;
928       order.clear() ;
929       end.clear() ;
930       tree.assign(1, vector<int>(alpha, 0)) ;
931   }
932
933   void add_pattern(const string &pattern){
934       int u = 0 ;
935       for ( auto ch : pattern ){
936           int v = ch - base ;
937
938           if(tree[u][v] == 0) tree[u][v] =
939               new_node() ;
940           u = tree[u][v] ;
941       }
942
943       end.push_back(u) ;
944   }
945
946   void build(){
947       queue<int> q ;
948       order.clear() ;
949       order.push_back(0) ;
950
951       for ( int i=0 ; i<alpha ; i++ )
952           if(tree[0][i] > 0){
953               q.push(tree[0][i]) ;
954           }
955
956       while(!q.empty()){
957           int u = q.front() ; q.pop() ;
958           order.push_back(u) ;
959
960           for ( int i=0 ; i<alpha ; i++ ){
961               if(tree[u][i] == 0) tree[u][i] =
962                   tree[fail[u]][i] ;
963               else{
964                   fail[tree[u][i]] = tree[fail[u]][i] ;
965                   ;
966                   q.push(tree[u][i]) ;
967               }
968           }
969       }
970
971       vector<int> count_per_pattern(const string
972                                     &text) const {
973           int u = 0 ;
974           vector<int> vis(tree.size(), 0) ;
975
976           for ( char ch : text ){
977               u = tree[u][ch - base] ;
978               vis[u]++;
979           }
980
981           for ( int i=order.size()-1 ; i>=1 ; i-- )
982               int x = order[i] ;
983               vis[fail[x]] += vis[x] ;
984           }
985
986           vector<int> ans(end.size(), 0) ;
987           for ( int id=0 ; id<end.size() ; id++ )
988               ans[id] = vis[order[id]] ;
989       }
990   }
991
992   void clear(){
993       fail.assign(1, 0) ;
994       order.clear() ;
995       end.clear() ;
996       tree.assign(1, vector<int>(alpha, 0)) ;
997   }
998
999   AAutomation(int _base='a', int _alpha=26)
1000    : base(_base), alpha(_alpha) {
1001        clear() ;
1002    }
1003
1004   void clear(){
1005       fail.assign(1, 0) ;
1006       order.clear() ;
1007       end.clear() ;
1008       tree.assign(1, vector<int>(alpha, 0)) ;
1009   }
1010
1011   void add_pattern(const string &pattern){
1012       int u = 0 ;
1013       for ( auto ch : pattern ){
1014           int v = ch - base ;
1015
1016           if(tree[u][v] == 0) tree[u][v] =
1017               new_node() ;
1018           u = tree[u][v] ;
1019       }
1020
1021       end.push_back(u) ;
1022   }
1023
1024   void build(){
1025       queue<int> q ;
1026       order.clear() ;
1027       order.push_back(0) ;
1028
1029       for ( int i=0 ; i<alpha ; i++ )
1030           if(tree[0][i] > 0){
1031               q.push(tree[0][i]) ;
1032           }
1033
1034       while(!q.empty()){
1035           int u = q.front() ; q.pop() ;
1036           order.push_back(u) ;
1037
1038           for ( int i=0 ; i<alpha ; i++ ){
1039               if(tree[u][i] == 0) tree[u][i] =
1040                   tree[fail[u]][i] ;
1041               else{
1042                   fail[tree[u][i]] = tree[fail[u]][i] ;
1043                   ;
1044                   q.push(tree[u][i]) ;
1045               }
1046           }
1047       }
1048
1049       vector<int> count_per_pattern(const string
1050                                     &text) const {
1051           int u = 0 ;
1052           vector<int> vis(tree.size(), 0) ;
1053
1054           for ( char ch : text ){
1055               u = tree[u][ch - base] ;
1056               vis[u]++;
1057           }
1058
1059           for ( int i=order.size()-1 ; i>=1 ; i-- )
1060               int x = order[i] ;
1061               vis[fail[x]] += vis[x] ;
1062           }
1063
1064           vector<int> ans(end.size(), 0) ;
1065           for ( int id=0 ; id<end.size() ; id++ )
1066               ans[id] = vis[order[id]] ;
1067       }
1068   }
1069
1070   void clear(){
1071       fail.assign(1, 0) ;
1072       order.clear() ;
1073       end.clear() ;
1074       tree.assign(1, vector<int>(alpha, 0)) ;
1075   }
1076
1077   AAutomation(int _base='a', int _alpha=26)
1078     : base(_base), alpha(_alpha) {
1079         clear() ;
1080     }
1081
1082   void clear(){
1083       fail.assign(1, 0) ;
1084       order.clear() ;
1085       end.clear() ;
1086       tree.assign(1, vector<int>(alpha, 0)) ;
1087   }
1088
1089   void add_pattern(const string &pattern){
1090       int u = 0 ;
1091       for ( auto ch : pattern ){
1092           int v = ch - base ;
1093
1094           if(tree[u][v] == 0) tree[u][v] =
1095               new_node() ;
1096           u = tree[u][v] ;
1097       }
1098
1099       end.push_back(u) ;
1100   }
1101
1102   void build(){
1103       queue<int> q ;
1104       order.clear() ;
1105       order.push_back(0) ;
1106
1107       for ( int i=0 ; i<alpha ; i++ )
1108           if(tree[0][i] > 0){
1109               q.push(tree[0][i]) ;
1110           }
1111
1112       while(!q.empty()){
1113           int u = q.front() ; q.pop() ;
1114           order.push_back(u) ;
1115
1116           for ( int i=0 ; i<alpha ; i++ ){
1117               if(tree[u][i] == 0) tree[u][i
```

```

78     ans[id] = vis[end[id]] ;
79 }
80
81     return ans ;
82 }
83 };

```

## 6 Techniques

### 6.1 二分搜

```

1 // xxxxxxxx 找最小解
2 bool binary_search(){
3     while(l < r){
4         int m = (l + r) >> 1 ;
5         if(check(m)) r = m ;
6         else l = m + 1 ;
7     }
8
9     return l ;
10}
11
12 // oooooxxx 找最大解
13 bool binary_search(){
14     while(l < r){
15         int m = (l + r) >> 1 ;
16         if(check(m)) l = m ;
17         else r = m - 1 ;
18     }
19
20     return l ;
21 }
22
23 // 如果l & r 太大, m = (l + (r - 1)) >> 1 ;

```

### 6.2 倍增 LCA

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 const int Maxn = 500005 ;
6
7 vector<int> e[Maxn] ;
8 int depth[Maxn] ;
9 int up[Maxn][40] ;
10 int MaxLog ;
11
12 void dfs(int u, int from, int d){
13     up[u][0] = from ;
14     depth[u] = d ;
15
16     for ( int i=1 ; i<=MaxLog ; i++ ){
17         up[u][i] = up[up[u][i - 1]][i - 1] ;
18     }
19
20     for ( auto v : e[u] ){
21         if(v == from) continue ;
22         dfs(v, u, d + 1) ;
23     }
24 }
25
26 int lca(int u, int v){
27     if(depth[u] < depth[v]) swap(u, v) ;
28
29     for ( int i=MaxLog ; i>=0 ; i-- )
30         if(depth[u] - (1 << i) >= depth[v]){
31             u = up[u][i] ;
32         }
33
34     if(u == v) return u ;
35
36     for ( int i=MaxLog ; i>=0 ; i-- )
37         if(up[u][i] != up[v][i]){

```

```

36         u = up[u][i] ;
37         v = up[v][i] ;
38     }
39
40     return up[u][0] ;
41 }
42
43 int main(){
44     int n, q, root ;
45     scanf(" %d %d ", &n, &q, &root) ;
46     MaxLog = __lg(n) ;
47
48     for ( int i=0 ; i<n-1 ; i++ ){
49         int u, v ;
50         scanf(" %d %d ", &u, &v) ;
51         e[u].push_back(v) ;
52         e[v].push_back(u) ;
53     }
54
55     dfs(root, root, 0) ;
56
57     while(q--){
58         int u, v ;
59         scanf(" %d %d ", &u, &v) ;
60         printf("%d\n", lca(u, v)) ;
61     }
62
63     return 0 ;
64 }

```

### 6.3 SG

```

1 long long SG(long long k){
2
3     if(k % 2 == 0){
4         return k / 2;
5     }
6     else{
7         return SG(k / 2);
8     }
9 }
10
11 int main(){
12     int cas, n;
13
14     scanf(" %d ", &cas);
15     while(cas--){
16         scanf(" %d ", &n);
17
18         long long s, v = 0;
19
20         for(int i = 0; i < n; i++){
21             scanf(" %lld ", &s);
22             v ^= SG(s); //XOR
23         }
24
25         if(v) printf("YES\n");
26         else printf("NO\n");
27     }
28
29     int SG[30] ;
30     int vis[Maxn], stone[Maxn] ;
31
32     void build(){
33         SG[0] = 0 ;
34         memset(vis, 0, sizeof(vis)) ;
35
36         for ( int i=1 ; i<30 ; i++ ){
37             int cur = 0 ;
38             for ( int j=0 ; j<i ; j++ ) for ( int
39                 k=0 ; k<=j ; k++ ){
40                 vis[SG[j] ^ SG[k]] = i ;
41             }
42             while(vis[cur] == i) cur++ ;
43             SG[i] = cur ;
44         }
45     }
46 }
47
48 int main(){
49     build() ;
50
51     int T = 0 ;
52     while(~scanf(" %d ", &n) && n){
53         int ans = 0 ;
54
55         for ( int i=1 ; i<=n ; i++ ) scanf(" %d ",
56             &stone[i]) ;
57
58         for ( int i=1 ; i<=n ; i++ ) if(stone[i]
59             & 1){
60             ans ^= SG[n-i] ;
61         }
62     }
63 }

```

## 7 DP

### 7.1 輪廓線 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4 using ll = long long ;
5
6 ll dp[2][(1 << 10) + 5] ;
7 int n, m ;
8 int cur ;
9
10 void update(int s1, int s2){
11     if(s2 & (1 << m)){
12         dp[cur][s2 ^ (1 << m)] += dp[cur ^ 1][s1] ;
13     }
14 }
15
16 int main(){
17     while(~scanf(" %d %d ", &n, &m)){
18         if(m > n) swap(n, m) ;
19         memset(dp, 0, sizeof(dp)) ;
20         cur = 0 ;
21         dp[cur][(1 << m) - 1] = 1 ;
22         for ( int i=0 ; i<n ; i++ ) for ( int
23             j=0 ; j<m ; j++ ){
24             cur ^= 1 ;
25             memset(dp[cur], 0, sizeof(dp[cur])) ;
26
27             for ( int k=0 ; k<(1 << m) ; k++ ){
28                 update(k, k << 1) ; // not put
29                 if(i && !(k & (1 << (m - 1))) )
30                     update(k, (k << 1) | (1 << m) |
31                         1) ; // put up
32                 if(j && !(k & 1)) update(k, (k << 1)
33                         | 3) ; // put left
34             }
35         }
36         printf("%lld\n", dp[cur][(1 << m) - 1]) ;
37     }
38     return 0 ;
39 }

```

### 7.2 數位 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 int K ;
6 int dp[20][105][105][2] ;
7 vector<int> dig ;

```

```

8
9 int solve(int pos, int sum, int dsum, bool
10    lim){
11    if(pos == -1){
12        if(sum == 0 && dsum == 0) return 1 ;
13        return 0 ;
14    }
15    int &d = dp[pos][sum][dsum][lim] ;
16    if(d != -1) return d ;
17
18    int up = lim ? dig[pos] : 9 ;
19    int res = 0 ;
20    for ( int i=0 ; i<=up ; i++ ){
21        res += solve(pos-1, (sum * 10 + i) %
22                      K, (dsum + i) % K, lim && i==up)
23                      ;
24    }
25    return d = res ;
26 }
27 int count(int n){
28    memset(dp, -1, sizeof(dp)) ;
29    dig.clear() ;
30
31    while(n > 0){
32        dig.push_back(n % 10) ;
33        n /= 10 ;
34    }
35
36    return solve(dig.size() - 1, 0, 0, 1) ;
37 }
38
39 int main(){
40    int T ;
41    scanf("%d", &T) ;
42
43    int a, b ;
44    while(T--){
45        scanf("%d%d", &a, &b, &K) ;
46        if(K > 90) printf("0\n") ;
47        else printf("%d\n", count(b) -
48                    count(a-1)) ;
49    }
50
51    return 0 ;
52 }

53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69

```

### 7.3 樹 DP

```
1 #include <bits/stdc++.h>
2
3 #define N 505
4 #define INF 0x3f3f3f3f
5
6 using namespace std ;
7
8 struct Edge{
9     int v, w ;
10 } ;
11
12 vector<Edge> edge[N] ;
13 int n ;
14 int cnt[N] ;
15 int dp[N][N][2] ;
16
17 void init(){
18     for ( int i=0 ; i<N ; i++ )
19         edge[i].clear() ;
20     memset(cnt, 0, sizeof(cnt)) ;
21     memset(dp, INF, sizeof(dp)) ;
22 }
23
24 void DFS(int u){
25     cnt[u] = 1 ;
26     for ( auto [v, w] : edge[u] ) if ( !cnt[v] ) DFS(v) ;
27 }
```