

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)

```

- 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。

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 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 μ 為莫比烏斯函數; 常用於反演及計算互質對數量。

Modulo Arithmetic Quick Facts

$$(a \pm b) \pmod{m} = ((a \pmod{m}) \pm (b \pmod{m})) \pmod{m},$$

$$(ab) \pmod{m} = ((a \pmod{m})(b \pmod{m})) \pmod{m},$$

$$(a^k) \pmod{m} = ((a \pmod{m})^k) \pmod{m},$$

$$(-a) \pmod{m} = (m - (a \pmod{m})) \pmod{m}.$$

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

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 $a \pm c \equiv b \pm d \pmod{m}$, $ac \equiv bd \pmod{m}$, $a^n \equiv b^n \pmod{m}$ 。

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     }
```

```

25   for ( auto v : g[u] ) if(v != fa[u] && v
26     != son[u]){
27       dfs2(v, v) ;
28   }
29
30 /* Segment Tree */
31 #define lc (id << 1)
32 #define rc ((id << 1) | 1)
33
34 struct ColorSeg{
35   int left, right, tot ;
36
37 ColorSeg operator+(const ColorSeg &o)
38   const {
39     if(tot == 0) return o ;
40     if(o.tot == 0) return *this ;
41
42     ColorSeg tmp ;
43     tmp.left = left ;
44     tmp.right = o.right ;
45     tmp.tot = tot + o.tot - (right ==
46                               o.left) ;
47
48     return tmp ;
49   }
50
51 struct Node{
52   ColorSeg color ;
53   int tag ;
54 }seg[Maxn << 2] ;
55
56 class SegmentTree{
57 private:
58   void pull(int id){
59     // normal pull
60   }
61   void AddTag(int id, int tag){
62     // normal AddTag
63   }
64   void push(int id){
65     // normal push
66   }
67
68   void modify(int l, int r, int tag, int
69   L=1, int R=n, int id=1){
70     // normal modify
71   }
72
73   ColorSeg query(int l, int r, int L=1, int
74   R=n, int id=1){
75     // normal query
76   }
77 public:
78   void build(int L=1, int R=n, int id=1){
79     // normal build
80   }
81
82   // update val from u to v (simple path)
83   void update(int u, int v, int val){
84     while(top[u] != top[v]){
85       if(dep[top[u]] < dep[top[v]]) swap(u,
86                                               v) ;
87       modify(dfn[top[u]], dfn[u], val) ;
88       u = fa[top[u]] ;
89     }
90
91     if(dep[u] < dep[v]) swap(u, v) ;
92     modify(dfn[v], dfn[u], val) ;
93
94   // get sum from u to v (simple path)
95   int get(int u, int v){
96     pair<int, ColorSeg> U, V ;
97     ColorSeg M ;

```

```

97   U = {u, {0, 0, 0}} ;
98   V = {v, {0, 0, 0}} ;
99
100  while(top[U.first] != top[V.first]){
101    if(dep[top[U.first]] <
102        dep[top[V.first]]) swap(U, V) ;
103    U.second = query(dfn[top[U.first]],
104                      dfn[U.first]) + U.second ;
105    U.first = fa[top[U.first]] ;
106
107    if(dep[U.first] < dep[V.first]) swap(U,
108                                              V) ;
109
110    M = query(dfn[V.first], dfn[U.first]) ;
111
112    return (U.second.tot + V.second.tot +
113            M.tot) - (U.second.left == M.right)
114          - (V.second.left == M.left) ;
115  }
116}

```

3.4 PST

```

1 // Find range k-th largest number
2 struct Node{
3   int sum, left, right ;
4 }seg[Maxn + 20 * Maxn] ;
5
6 class PersistentSegmentTree{
7 private:
8   int n ;
9   int cnt ;
10  vector<int> version ;
11
12  int build(int L, int R){
13    int cur_cnt = cnt++ ;
14    if(L == R){
15      seg[cur_cnt] = {0, 0, 0} ;
16      return cur_cnt ;
17    }
18
19    int M = (L + R) >> 1 ;
20    int lc = build(L, M) ;
21    int rc = build(M+1, R) ;
22
23    seg[cur_cnt] = {0, lc, rc} ;
24    return cur_cnt ;
25  }
26 public:
27  PersistentSegmentTree(int _n){
28    n = _n ;
29    cnt = 0 ;
30
31    int root = build(1, n) ;
32    version.push_back(root) ;
33  }
34
35  void update(int ver, int idx){
36    auto upd = [&](auto &self, const int
37                  cur, int L, int R){
38      int cur_cnt = cnt++ ;
39
40      if(L == R){
41        seg[cur_cnt] = {seg[cur].sum + 1, 0,
42                        0} ;
43        return cur_cnt ;
44      }
45
46      int M = (L + R) >> 1 ;
47      int lc = seg[cur].left ;
48      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 int query(int verL, int verR, int k){
63  auto qry = [&](auto &self, const int
64                  cur_old, const int cur_new, int L,
65                  int R){
66    if(L == R) return L ;
67
68    int old_l = seg[cur_old].left, old_r =
69                seg[cur_old].right ;
70    int new_l = seg[cur_new].left, new_r =
71                seg[cur_new].right ;
72
73    int dl = seg[new_l].sum -
74            seg[old_l].sum ;
75    int dr = seg[new_r].sum -
76            seg[old_r].sum ;
77
78    int M = (L + R) >> 1 ;
79
80    if(dl >= k) return self(self, old_l,
81                             new_l, L, M) ;
82    k -= dl ;
83    return self(self, old_r, new_r, M+1,
84                           R) ;
85  };
86
87  int idx = qry(qry, version[verL-1],
88                version[verR], 1, n) ;
89  return idx ;
90}

```

3.5 Trie

```

1 class TrieNode{
2 public:
3   set<int> end ;
4   TrieNode *next[26] ;
5
6   TrieNode(){
7     for ( int i=0 ; i<26 ; i++ ) next[i]
8       = nullptr ;
9   };
10
11 class Trie{
12 private:
13   int cnt ;
14   TrieNode *root ;
15 public:
16   Trie() : cnt(0) {
17     root = new TrieNode() ;
18   }
19
20   void insert(string &str, int n){
21     TrieNode* node = root ;
22     for ( auto s : str ){
23       int path = s - 'a' ;
24
25       if(node->next[path] == nullptr)
26         node->next[path] = new
27                     TrieNode() ;
28       node = node->next[path] ;
29     }
30   }
31
32   string search(string str){
33     TrieNode* node = root ;
34
35     for ( auto s : str ){
36       int path = s - 'a' ;
37
38       if(node->next[path] == nullptr)
39         return "NO" ;
40       node = node->next[path] ;
41     }
42
43     if(node->end.size() > 0)
44       return "YES" ;
45     else
46       return "NO" ;
47   }
48
49   int count(string str){
50     TrieNode* node = root ;
51
52     for ( auto s : str ){
53       int path = s - 'a' ;
54
55       if(node->next[path] == nullptr)
56         return 0 ;
57       node = node->next[path] ;
58     }
59
60     return node->end.size() ;
61   }
62
63   void erase(string str){
64     TrieNode* node = root ;
65
66     for ( auto s : str ){
67       int path = s - 'a' ;
68
69       if(node->next[path] == nullptr)
70         return ;
71       node = node->next[path] ;
72     }
73
74     if(node->end.size() == 0)
75       delete node ;
76     else
77       node->end.erase(str) ;
78   }
79
80   void print(TrieNode* node, string str=""){
81     if(node == nullptr)
82       return ;
83
84     for ( auto s : node->end )
85       cout << str << s ;
86
87     for ( auto s : node->next )
88       print(s, str + s) ;
89   }
90
91   void print(Trie* trie, string str=""){
92     for ( auto s : trie->end )
93       cout << str << s ;
94
95     for ( auto s : trie->next )
96       print(s, str + s) ;
97   }
98
99   void print(string str, string str2=""){
100    for ( auto s : str )
101      cout << str2 << s ;
102
103    for ( auto s : str2 )
104      cout << str << s ;
105  }
106
107  void print(string str, string str2=""){
108    for ( auto s : str )
109      cout << str2 << s ;
110
111    for ( auto s : str2 )
112      cout << str << s ;
113  }
114
115  void print(string str, string str2=""){
116    for ( auto s : str )
117      cout << str2 << s ;
118
119    for ( auto s : str2 )
120      cout << str << s ;
121  }
122
123  void print(string str, string str2=""){
124    for ( auto s : str )
125      cout << str2 << s ;
126
127    for ( auto s : str2 )
128      cout << str << s ;
129  }
130
131  void print(string str, string str2=""){
132    for ( auto s : str )
133      cout << str2 << s ;
134
135    for ( auto s : str2 )
136      cout << str << s ;
137  }
138
139  void print(string str, string str2=""){
140    for ( auto s : str )
141      cout << str2 << s ;
142
143    for ( auto s : str2 )
144      cout << str << s ;
145  }
146
147  void print(string str, string str2=""){
148    for ( auto s : str )
149      cout << str2 << s ;
150
151    for ( auto s : str2 )
152      cout << str << s ;
153  }
154
155  void print(string str, string str2=""){
156    for ( auto s : str )
157      cout << str2 << s ;
158
159    for ( auto s : str2 )
160      cout << str << s ;
161  }
162
163  void print(string str, string str2=""){
164    for ( auto s : str )
165      cout << str2 << s ;
166
167    for ( auto s : str2 )
168      cout << str << s ;
169  }
170
171  void print(string str, string str2=""){
172    for ( auto s : str )
173      cout << str2 << s ;
174
175    for ( auto s : str2 )
176      cout << str << s ;
177  }
178
179  void print(string str, string str2=""){
180    for ( auto s : str )
181      cout << str2 << s ;
182
183    for ( auto s : str2 )
184      cout << str << s ;
185  }
186
187  void print(string str, string str2=""){
188    for ( auto s : str )
189      cout << str2 << s ;
190
191    for ( auto s : str2 )
192      cout << str << s ;
193  }
194
195  void print(string str, string str2=""){
196    for ( auto s : str )
197      cout << str2 << s ;
198
199    for ( auto s : str2 )
200      cout << str << s ;
201  }
202
203  void print(string str, string str2=""){
204    for ( auto s : str )
205      cout << str2 << s ;
206
207    for ( auto s : str2 )
208      cout << str << s ;
209  }
210
211  void print(string str, string str2=""){
212    for ( auto s : str )
213      cout << str2 << s ;
214
215    for ( auto s : str2 )
216      cout << str << s ;
217  }
218
219  void print(string str, string str2=""){
220    for ( auto s : str )
221      cout << str2 << s ;
222
223    for ( auto s : str2 )
224      cout << str << s ;
225  }
226
227  void print(string str, string str2=""){
228    for ( auto s : str )
229      cout << str2 << s ;
230
231    for ( auto s : str2 )
232      cout << str << s ;
233  }
234
235  void print(string str, string str2=""){
236    for ( auto s : str )
237      cout << str2 << s ;
238
239    for ( auto s : str2 )
240      cout << str << s ;
241  }
242
243  void print(string str, string str2=""){
244    for ( auto s : str )
245      cout << str2 << s ;
246
247    for ( auto s : str2 )
248      cout << str << s ;
249  }
250
251  void print(string str, string str2=""){
252    for ( auto s : str )
253      cout << str2 << s ;
254
255    for ( auto s : str2 )
256      cout << str << s ;
257  }
258
259  void print(string str, string str2=""){
260    for ( auto s : str )
261      cout << str2 << s ;
262
263    for ( auto s : str2 )
264      cout << str << s ;
265  }
266
267  void print(string str, string str2=""){
268    for ( auto s : str )
269      cout << str2 << s ;
270
271    for ( auto s : str2 )
272      cout << str << s ;
273  }
274
275  void print(string str, string str2=""){
276    for ( auto s : str )
277      cout << str2 << s ;
278
279    for ( auto s : str2 )
280      cout << str << s ;
281  }
282
283  void print(string str, string str2=""){
284    for ( auto s : str )
285      cout << str2 << s ;
286
287    for ( auto s : str2 )
288      cout << str << s ;
289  }
290
291  void print(string str, string str2=""){
292    for ( auto s : str )
293      cout << str2 << s ;
294
295    for ( auto s : str2 )
296      cout << str << s ;
297  }
298
299  void print(string str, string str2=""){
300    for ( auto s : str )
301      cout << str2 << s ;
302
303    for ( auto s : str2 )
304      cout << str << s ;
305  }
306
307  void print(string str, string str2=""){
308    for ( auto s : str )
309      cout << str2 << s ;
310
311    for ( auto s : str2 )
312      cout << str << s ;
313  }
314
315  void print(string str, string str2=""){
316    for ( auto s : str )
317      cout << str2 << s ;
318
319    for ( auto s : str2 )
320      cout << str << s ;
321  }
322
323  void print(string str, string str2=""){
324    for ( auto s : str )
325      cout << str2 << s ;
326
327    for ( auto s : str2 )
328      cout << str << s ;
329  }
330
331  void print(string str, string str2=""){
332    for ( auto s : str )
333      cout << str2 << s ;
334
335    for ( auto s : str2 )
336      cout << str << s ;
337  }
338
339  void print(string str, string str2=""){
340    for ( auto s : str )
341      cout << str2 << s ;
342
343    for ( auto s : str2 )
344      cout << str << s ;
345  }
346
347  void print(string str, string str2=""){
348    for ( auto s : str )
349      cout << str2 << s ;
350
351    for ( auto s : str2 )
352      cout << str << s ;
353  }
354
355  void print(string str, string str2=""){
356    for ( auto s : str )
357      cout << str2 << s ;
358
359    for ( auto s : str2 )
360      cout << str << s ;
361  }
362
363  void print(string str, string str2=""){
364    for ( auto s : str )
365      cout << str2 << s ;
366
367    for ( auto s : str2 )
368      cout << str << s ;
369  }
370
371  void print(string str, string str2=""){
372    for ( auto s : str )
373      cout << str2 << s ;
374
375    for ( auto s : str2 )
376      cout << str << s ;
377  }
378
379  void print(string str, string str2=""){
380    for ( auto s : str )
381      cout << str2 << s ;
382
383    for ( auto s : str2 )
384      cout << str << s ;
385  }
386
387  void print(string str, string str2=""){
388    for ( auto s : str )
389      cout << str2 << s ;
390
391    for ( auto s : str2 )
392      cout << str << s ;
393  }
394
395  void print(string str, string str2=""){
396    for ( auto s : str )
397      cout << str2 << s ;
398
399    for ( auto s : str2 )
400      cout << str << s ;
401  }
402
403  void print(string str, string str2=""){
404    for ( auto s : str )
405      cout << str2 << s ;
406
407    for ( auto s : str2 )
408      cout << str << s ;
409  }
410
411  void print(string str, string str2=""){
412    for ( auto s : str )
413      cout << str2 << s ;
414
415    for ( auto s : str2 )
416      cout << str << s ;
417  }
418
419  void print(string str, string str2=""){
420    for ( auto s : str )
421      cout << str2 << s ;
422
423    for ( auto s : str2 )
424      cout << str << s ;
425  }
426
427  void print(string str, string str2=""){
428    for ( auto s : str )
429      cout << str2 << s ;
430
431    for ( auto s : str2 )
432      cout << str << s ;
433  }
434
435  void print(string str, string str2=""){
436    for ( auto s : str )
437      cout << str2 << s ;
438
439    for ( auto s : str2 )
440      cout << str << s ;
441  }
442
443  void print(string str, string str2=""){
444    for ( auto s : str )
445      cout << str2 << s ;
446
447    for ( auto s : str2 )
448      cout << str << s ;
449  }
450
451  void print(string str, string str2=""){
452    for ( auto s : str )
453      cout << str2 << s ;
454
455    for ( auto s : str2 )
456      cout << str << s ;
457  }
458
459  void print(string str, string str2=""){
460    for ( auto s : str )
461      cout << str2 << s ;
462
463    for ( auto s : str2 )
464      cout << str << s ;
465  }
466
467  void print(string str, string str2=""){
468    for ( auto s : str )
469      cout << str2 << s ;
470
471    for ( auto s : str2 )
472      cout << str << s ;
473  }
474
475  void print(string str, string str2=""){
476    for ( auto s : str )
477      cout << str2 << s ;
478
479    for ( auto s : str2 )
480      cout << str << s ;
481  }
482
483  void print(string str, string str2=""){
484    for ( auto s : str )
485      cout << str2 << s ;
486
487    for ( auto s : str2 )
488      cout << str << s ;
489  }
490
491  void print(string str, string str2=""){
492    for ( auto s : str )
493      cout << str2 << s ;
494
495    for ( auto s : str2 )
496      cout << str << s ;
497  }
498
499  void print(string str, string str2=""){
500    for ( auto s : str )
501      cout << str2 << s ;
502
503    for ( auto s : str2 )
504      cout << str << s ;
505  }
506
507  void print(string str, string str2=""){
508    for ( auto s : str )
509      cout << str2 << s ;
510
511    for ( auto s : str2 )
512      cout << str << s ;
513  }
514
515  void print(string str, string str2=""){
516    for ( auto s : str )
517      cout << str2 << s ;
518
519    for ( auto s : str2 )
520      cout << str << s ;
521  }
522
523  void print(string str, string str2=""){
524    for ( auto s : str )
525      cout << str2 << s ;
526
527    for ( auto s : str2 )
528      cout << str << s ;
529  }
530
531  void print(string str, string str2=""){
532    for ( auto s : str )
533      cout << str2 << s ;
534
535    for ( auto s : str2 )
536      cout << str << s ;
537  }
538
539  void print(string str, string str2=""){
540    for ( auto s : str )
541      cout << str2 << s ;
542
543    for ( auto s : str2 )
544      cout << str << s ;
545  }
546
547  void print(string str, string str2=""){
548    for ( auto s : str )
549      cout << str2 << s ;
550
551    for ( auto s : str2 )
552      cout << str << s ;
553  }
554
555  void print(string str, string str2=""){
556    for ( auto s : str )
557      cout << str2 << s ;
558
559    for ( auto s : str2 )
560      cout << str << s ;
561  }
562
563  void print(string str, string str2=""){
564    for ( auto s : str )
565      cout << str2 << s ;
566
567    for ( auto s : str2 )
568      cout << str << s ;
569  }
570
571  void print(string str, string str2=""){
572    for ( auto s : str )
573      cout << str2 << s ;
574
575    for ( auto s : str2 )
576      cout << str << s ;
577  }
578
579  void print(string str, string str2=""){
580    for ( auto s : str )
581      cout << str2 << s ;
582
583    for ( auto s : str2 )
584      cout << str << s ;
585  }
586
587  void print(string str, string str2=""){
588    for ( auto s : str )
589      cout << str2 << s ;
590
591    for ( auto s : str2 )
592      cout << str << s ;
593  }
594
595  void print(string str, string str2=""){
596    for ( auto s : str )
597      cout << str2 << s ;
598
599    for ( auto s : str2 )
600      cout << str << s ;
601  }
602
603  void print(string str, string str2=""){
604    for ( auto s : str )
605      cout << str2 << s ;
606
607    for ( auto s : str2 )
608      cout << str << s ;
609  }
610
611  void print(string str, string str2=""){
612    for ( auto s : str )
613      cout << str2 << s ;
614
615    for ( auto s : str2 )
616      cout << str << s ;
617  }
618
619  void print(string str, string str2=""){
620    for ( auto s : str )
621      cout << str2 << s ;
622
623    for ( auto s : str2 )
624      cout << str << s ;
625  }
626
627  void print(string str, string str2=""){
628    for ( auto s : str )
629      cout << str2 << s ;
630
631    for ( auto s : str2 )
632      cout << str << s ;
633  }
634
635  void print(string str, string str2=""){
636    for ( auto s : str )
637      cout << str2 << s ;
638
639    for ( auto s : str2 )
640      cout << str << s ;
641  }
642
643  void print(string str, string str2=""){
644    for ( auto s : str )
645      cout << str2 << s ;
646
647    for ( auto s : str2 )
648      cout << str << s ;
649  }
650
651  void print(string str, string str2=""){
652    for ( auto s : str )
653      cout << str2 << s ;
654
655    for ( auto s : str2 )
656      cout << str << s ;
657  }
658
659  void print(string str, string str2=""){
660    for ( auto s : str )
661      cout << str2 << s ;
662
663    for ( auto s : str2 )
664      cout << str << s ;
665  }
666
667  void print(string str, string str2=""){
668    for ( auto s : str )
669      cout << str2 << s ;
670
671    for ( auto s : str2 )
672      cout << str << s ;
673  }
674
675  void print(string str, string str2=""){
676    for ( auto s : str )
677      cout << str2 << s ;
678
679    for ( auto s : str2 )
680      cout << str << s ;
681  }
682
683  void print(string str, string str2=""){
684    for ( auto s : str )
685      cout << str2 << s ;
686
687    for ( auto s : str2 )
688      cout << str << s ;
689  }
690
691  void print(string str, string str2=""){
692    for ( auto s : str )
693      cout << str2 << s ;
694
695    for ( auto s : str2 )
696      cout << str << s ;
697  }
698
699  void print(string str, string str2=""){
700    for ( auto s : str )
701      cout << str2 << s ;
702
703    for ( auto s : str2 )
704      cout << str << s ;
705  }
706
707  void print(string str, string str2=""){
708    for ( auto s : str )
709      cout << str2 << s ;
710
711    for ( auto s : str2 )
712      cout << str << s ;
713  }
714
715  void print(string str, string str2=""){
716    for ( auto s : str )
717      cout << str2 << s ;
718
719    for ( auto s : str2 )
720      cout << str << s ;
721  }
722
723  void print(string str, string str2=""){
724    for ( auto s : str )
725      cout << str2 << s ;
726
727    for ( auto s : str2 )
728      cout << str << s ;
729  }
730
731  void print(string str, string str2=""){
732    for ( auto s : str )
733      cout << str2 << s ;
734
735    for ( auto s : str2 )
736      cout << str << s ;
737  }
738
739  void print(string str, string str2=""){
740    for ( auto s : str )
741      cout << str2 << s ;
742
743    for ( auto s : str2 )
744      cout << str << s ;
745  }
746
747  void print(string str, string str2=""){
748    for ( auto s : str )
749      cout << str2 << s ;
750
751    for ( auto s : str2 )
752      cout << str << s ;
753  }
754
755  void print(string str, string str2=""){
756    for ( auto s : str )
757      cout << str2 << s ;
758
759    for ( auto s : str2 )
760      cout << str << s ;
761  }
762
763  void print(string str, string str2=""){
764    for ( auto s : str )
765      cout << str2 << s ;
766
767    for ( auto s : str2 )
768      cout << str << s ;
769  }
770
771  void print(string str, string str2=""){
772    for ( auto s : str )
773      cout << str2 << s ;
774
775    for ( auto s : str2 )
776      cout << str << s ;
777  }
778
779  void print(string str, string str2=""){
780    for ( auto s : str )
781      cout << str2 << s ;
782
783    for ( auto s : str2 )
784      cout << str << s ;
785  }
786
787  void print(string str, string str2=""){
788    for ( auto s : str )
789      cout << str2 << s ;
790
791    for ( auto s : str2 )
792      cout << str << s ;
793  }
794
795  void print(string str, string str2=""){
796    for ( auto s : str )
797      cout << str2 << s ;
798
799    for ( auto s : str2 )
800      cout << str << s ;
801  }
802
803  void print(string str, string str2=""){
804    for ( auto s : str )
805      cout << str2 << s ;
806
807    for ( auto s : str2 )
808      cout << str << s ;
809  }
810
811  void print(string str, string str2=""){
812    for ( auto s : str )
813      cout << str2 << s ;
814
815    for ( auto s : str2 )
816      cout << str << s ;
817  }
818
819  void print(string str, string str2=""){
820    for ( auto s : str )
821      cout << str2 << s ;
822
823    for ( auto s : str2 )
824      cout << str << s ;
825  }
826
827  void print(string str, string str2=""){
828    for ( auto s : str )
829      cout << str2 << s ;
830
831    for ( auto s : str2 )
832      cout << str << s ;
833  }
834
835  void print(string str, string str2=""){
836    for ( auto s : str )
837      cout << str2 << s ;
838
839    for ( auto s : str2 )
840      cout << str << s ;
841  }
842
843  void print(string str, string str2=""){
844    for ( auto s : str )
845      cout << str2 << s ;
846
847    for ( auto s : str2 )
848      cout << str << s ;
849  }
850
851  void print(string str, string str2=""){
852    for ( auto s : str )
853      cout << str2 << s ;
854
855    for ( auto s : str2 )
856      cout << str << s ;
857  }
858
859  void print(string str, string str2=""){
860    for ( auto s : str )
861      cout << str2 << s ;
862
863    for ( auto s : str2 )
864      cout << str << s ;
865  }
866
867  void print(string str, string str2=""){
868    for ( auto s : str )
869      cout << str2 << s ;
870
871    for ( auto s : str2 )
872      cout << str << s ;
873  }
874
875  void print(string str, string str2=""){
876    for ( auto s : str )
877      cout << str2 << s ;
878
879    for ( auto s : str2 )
880      cout << str << s ;
881  }
882
883  void print(string str, string str2=""){
884    for ( auto s : str )
885      cout << str2
```

```

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        if(dfn[u] == low[u]){
18            cnt_scc++ ;
19            int x ;
20
21            do{
22                x = st.top() ;
23                st.pop() ;
24
25                inSt[x] = 0 ;
26                sccId[x] = cnt_scc ;
27                scc[cnt_scc].push_back(x) ;
28            }
29            while(x != u) ;
30        }
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(dijkstra()){
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    }
73
74    int run(){
75        int ans = 0 ;
76        while(bfs()){
77            ans += dfs(S, 0x3f3f3f3f) ;
78        }
79    }
80
81    int run(){
82        int ans = 0 ;
83        while(bfs()){
84            ans += dfs(S, 0x3f3f3f3f) ;
85        }
86    }
87
88    int run(){
89        int ans = 0 ;
90        while(bfs()){
91            ans += dfs(S, 0x3f3f3f3f) ;
92        }
93    }
94
95    int run(){
96        int ans = 0 ;
97        while(bfs()){
98            ans += dfs(S, 0x3f3f3f3f) ;
99        }
100    }
101
102    int run(){
103        int ans = 0 ;
104        while(bfs()){
105            ans += dfs(S, 0x3f3f3f3f) ;
106        }
107    }
108
109    int run(){
110        int ans = 0 ;
111        while(bfs()){
112            ans += dfs(S, 0x3f3f3f3f) ;
113        }
114    }
115
116    int run(){
117        int ans = 0 ;
118        while(bfs()){
119            ans += dfs(S, 0x3f3f3f3f) ;
120        }
121    }
122
123    int run(){
124        int ans = 0 ;
125        while(bfs()){
126            ans += dfs(S, 0x3f3f3f3f) ;
127        }
128    }
129
130    int run(){
131        int ans = 0 ;
132        while(bfs()){
133            ans += dfs(S, 0x3f3f3f3f) ;
134        }
135    }
136
137    int run(){
138        int ans = 0 ;
139        while(bfs()){
140            ans += dfs(S, 0x3f3f3f3f) ;
141        }
142    }
143
144    int run(){
145        int ans = 0 ;
146        while(bfs()){
147            ans += dfs(S, 0x3f3f3f3f) ;
148        }
149    }
150
151    int run(){
152        int ans = 0 ;
153        while(bfs()){
154            ans += dfs(S, 0x3f3f3f3f) ;
155        }
156    }
157
158    int run(){
159        int ans = 0 ;
160        while(bfs()){
161            ans += dfs(S, 0x3f3f3f3f) ;
162        }
163    }
164
165    int run(){
166        int ans = 0 ;
167        while(bfs()){
168            ans += dfs(S, 0x3f3f3f3f) ;
169        }
170    }
171
172    int run(){
173        int ans = 0 ;
174        while(bfs()){
175            ans += dfs(S, 0x3f3f3f3f) ;
176        }
177    }
178
179    int run(){
180        int ans = 0 ;
181        while(bfs()){
182            ans += dfs(S, 0x3f3f3f3f) ;
183        }
184    }
185
186    int run(){
187        int ans = 0 ;
188        while(bfs()){
189            ans += dfs(S, 0x3f3f3f3f) ;
190        }
191    }
192
193    int run(){
194        int ans = 0 ;
195        while(bfs()){
196            ans += dfs(S, 0x3f3f3f3f) ;
197        }
198    }
199
200    int run(){
201        int ans = 0 ;
202        while(bfs()){
203            ans += dfs(S, 0x3f3f3f3f) ;
204        }
205    }
206
207    int run(){
208        int ans = 0 ;
209        while(bfs()){
210            ans += dfs(S, 0x3f3f3f3f) ;
211        }
212    }
213
214    int run(){
215        int ans = 0 ;
216        while(bfs()){
217            ans += dfs(S, 0x3f3f3f3f) ;
218        }
219    }
220
221    int run(){
222        int ans = 0 ;
223        while(bfs()){
224            ans += dfs(S, 0x3f3f3f3f) ;
225        }
226    }
227
228    int run(){
229        int ans = 0 ;
230        while(bfs()){
231            ans += dfs(S, 0x3f3f3f3f) ;
232        }
233    }
234
235    int run(){
236        int ans = 0 ;
237        while(bfs()){
238            ans += dfs(S, 0x3f3f3f3f) ;
239        }
240    }
241
242    int run(){
243        int ans = 0 ;
244        while(bfs()){
245            ans += dfs(S, 0x3f3f3f3f) ;
246        }
247    }
248
249    int run(){
250        int ans = 0 ;
251        while(bfs()){
252            ans += dfs(S, 0x3f3f3f3f) ;
253        }
254    }
255
256    int run(){
257        int ans = 0 ;
258        while(bfs()){
259            ans += dfs(S, 0x3f3f3f3f) ;
260        }
261    }
262
263    int run(){
264        int ans = 0 ;
265        while(bfs()){
266            ans += dfs(S, 0x3f3f3f3f) ;
267        }
268    }
269
270    int run(){
271        int ans = 0 ;
272        while(bfs()){
273            ans += dfs(S, 0x3f3f3f3f) ;
274        }
275    }
276
277    int run(){
278        int ans = 0 ;
279        while(bfs()){
280            ans += dfs(S, 0x3f3f3f3f) ;
281        }
282    }
283
284    int run(){
285        int ans = 0 ;
286        while(bfs()){
287            ans += dfs(S, 0x3f3f3f3f) ;
288        }
289    }
290
291    int run(){
292        int ans = 0 ;
293        while(bfs()){
294            ans += dfs(S, 0x3f3f3f3f) ;
295        }
296    }
297
298    int run(){
299        int ans = 0 ;
300        while(bfs()){
301            ans += dfs(S, 0x3f3f3f3f) ;
302        }
303    }
304
305    int run(){
306        int ans = 0 ;
307        while(bfs()){
308            ans += dfs(S, 0x3f3f3f3f) ;
309        }
310    }
311
312    int run(){
313        int ans = 0 ;
314        while(bfs()){
315            ans += dfs(S, 0x3f3f3f3f) ;
316        }
317    }
318
319    int run(){
320        int ans = 0 ;
321        while(bfs()){
322            ans += dfs(S, 0x3f3f3f3f) ;
323        }
324    }
325
326    int run(){
327        int ans = 0 ;
328        while(bfs()){
329            ans += dfs(S, 0x3f3f3f3f) ;
330        }
331    }
332
333    int run(){
334        int ans = 0 ;
335        while(bfs()){
336            ans += dfs(S, 0x3f3f3f3f) ;
337        }
338    }
339
340    int run(){
341        int ans = 0 ;
342        while(bfs()){
343            ans += dfs(S, 0x3f3f3f3f) ;
344        }
345    }
346
347    int run(){
348        int ans = 0 ;
349        while(bfs()){
350            ans += dfs(S, 0x3f3f3f3f) ;
351        }
352    }
353
354    int run(){
355        int ans = 0 ;
356        while(bfs()){
357            ans += dfs(S, 0x3f3f3f3f) ;
358        }
359    }
360
361    int run(){
362        int ans = 0 ;
363        while(bfs()){
364            ans += dfs(S, 0x3f3f3f3f) ;
365        }
366    }
367
368    int run(){
369        int ans = 0 ;
370        while(bfs()){
371            ans += dfs(S, 0x3f3f3f3f) ;
372        }
373    }
374
375    int run(){
376        int ans = 0 ;
377        while(bfs()){
378            ans += dfs(S, 0x3f3f3f3f) ;
379        }
380    }
381
382    int run(){
383        int ans = 0 ;
384        while(bfs()){
385            ans += dfs(S, 0x3f3f3f3f) ;
386        }
387    }
388
389    int run(){
390        int ans = 0 ;
391        while(bfs()){
392            ans += dfs(S, 0x3f3f3f3f) ;
393        }
394    }
395
396    int run(){
397        int ans = 0 ;
398        while(bfs()){
399            ans += dfs(S, 0x3f3f3f3f) ;
400        }
401    }
402
403    int run(){
404        int ans = 0 ;
405        while(bfs()){
406            ans += dfs(S, 0x3f3f3f3f) ;
407        }
408    }
409
410    int run(){
411        int ans = 0 ;
412        while(bfs()){
413            ans += dfs(S, 0x3f3f3f3f) ;
414        }
415    }
416
417    int run(){
418        int ans = 0 ;
419        while(bfs()){
420            ans += dfs(S, 0x3f3f3f3f) ;
421        }
422    }
423
424    int run(){
425        int ans = 0 ;
426        while(bfs()){
427            ans += dfs(S, 0x3f3f3f3f) ;
428        }
429    }
430
431    int run(){
432        int ans = 0 ;
433        while(bfs()){
434            ans += dfs(S, 0x3f3f3f3f) ;
435        }
436    }
437
438    int run(){
439        int ans = 0 ;
440        while(bfs()){
441            ans += dfs(S, 0x3f3f3f3f) ;
442        }
443    }
444
445    int run(){
446        int ans = 0 ;
447        while(bfs()){
448            ans += dfs(S, 0x3f3f3f3f) ;
449        }
450    }
451
452    int run(){
453        int ans = 0 ;
454        while(bfs()){
455            ans += dfs(S, 0x3f3f3f3f) ;
456        }
457    }
458
459    int run(){
460        int ans = 0 ;
461        while(bfs()){
462            ans += dfs(S, 0x3f3f3f3f) ;
463        }
464    }
465
466    int run(){
467        int ans = 0 ;
468        while(bfs()){
469            ans += dfs(S, 0x3f3f3f3f) ;
470        }
471    }
472
473    int run(){
474        int ans = 0 ;
475        while(bfs()){
476            ans += dfs(S, 0x3f3f3f3f) ;
477        }
478    }
479
480    int run(){
481        int ans = 0 ;
482        while(bfs()){
483            ans += dfs(S, 0x3f3f3f3f) ;
484        }
485    }
486
487    int run(){
488        int ans = 0 ;
489        while(bfs()){
490            ans += dfs(S, 0x3f3f3f3f) ;
491        }
492    }
493
494    int run(){
495        int ans = 0 ;
496        while(bfs()){
497            ans += dfs(S, 0x3f3f3f3f) ;
498        }
499    }
500
501    int run(){
502        int ans = 0 ;
503        while(bfs()){
504            ans += dfs(S, 0x3f3f3f3f) ;
505        }
506    }
507
508    int run(){
509        int ans = 0 ;
510        while(bfs()){
511            ans += dfs(S, 0x3f3f3f3f) ;
512        }
513    }
514
515    int run(){
516        int ans = 0 ;
517        while(bfs()){
518            ans += dfs(S, 0x3f3f3f3f) ;
519        }
520    }
521
522    int run(){
523        int ans = 0 ;
524        while(bfs()){
525            ans += dfs(S, 0x3f3f3f3f) ;
526        }
527    }
528
529    int run(){
530        int ans = 0 ;
531        while(bfs()){
532            ans += dfs(S, 0x3f3f3f3f) ;
533        }
534    }
535
536    int run(){
537        int ans = 0 ;
538        while(bfs()){
539            ans += dfs(S, 0x3f3f3f3f) ;
540        }
541    }
542
543    int run(){
544        int ans = 0 ;
545        while(bfs()){
546            ans += dfs(S, 0x3f3f3f3f) ;
547        }
548    }
549
550    int run(){
551        int ans = 0 ;
552        while(bfs()){
553            ans += dfs(S, 0x3f3f3f3f) ;
554        }
555    }
556
557    int run(){
558        int ans = 0 ;
559        while(bfs()){
560            ans += dfs(S, 0x3f3f3f3f) ;
561        }
562    }
563
564    int run(){
565        int ans = 0 ;
566        while(bfs()){
567            ans += dfs(S, 0x3f3f3f3f) ;
568        }
569    }
570
571    int run(){
572        int ans = 0 ;
573        while(bfs()){
574            ans += dfs(S, 0x3f3f3f3f) ;
575        }
576    }
577
578    int run(){
579        int ans = 0 ;
580        while(bfs()){
581            ans += dfs(S, 0x3f3f3f3f) ;
582        }
583    }
584
585    int run(){
586        int ans = 0 ;
587        while(bfs()){
588            ans += dfs(S, 0x3f3f3f3f) ;
589        }
590    }
591
592    int run(){
593        int ans = 0 ;
594        while(bfs()){
595            ans += dfs(S, 0x3f3f3f3f) ;
596        }
597    }
598
599    int run(){
600        int ans = 0 ;
601        while(bfs()){
602            ans += dfs(S, 0x3f3f3f3f) ;
603        }
604    }
605
606    int run(){
607        int ans = 0 ;
608        while(bfs()){
609            ans += dfs(S, 0x3f3f3f3f) ;
610        }
611    }
612
613    int run(){
614        int ans = 0 ;
615        while(bfs()){
616            ans += dfs(S, 0x3f3f3f3f) ;
617        }
618    }
619
620    int run(){
621        int ans = 0 ;
622        while(bfs()){
623            ans += dfs(S, 0x3f3f3f3f) ;
624        }
625    }
626
627    int run(){
628        int ans = 0 ;
629        while(bfs()){
630            ans += dfs(S, 0x3f3f3f3f) ;
631        }
632    }
633
634    int run(){
635        int ans = 0 ;
636        while(bfs()){
637            ans += dfs(S, 0x3f3f3f3f) ;
638        }
639    }
640
641    int run(){
642        int ans = 0 ;
643        while(bfs()){
644            ans += dfs(S, 0x3f3f3f3f) ;
645        }
646    }
647
648    int run(){
649        int ans = 0 ;
650        while(bfs()){
651            ans += dfs(S, 0x3f3f3f3f) ;
652        }
653    }
654
655    int run(){
656        int ans = 0 ;
657        while(bfs()){
658            ans += dfs(S, 0x3f3f3f3f) ;
659        }
660    }
661
662    int run(){
663        int ans = 0 ;
664        while(bfs()){
665            ans += dfs(S, 0x3f3f3f3f) ;
666        }
667    }
668
669    int run(){
670        int ans = 0 ;
671        while(bfs()){
672            ans += dfs(S, 0x3f3f3f3f) ;
673        }
674    }
675
676    int run(){
677        int ans = 0 ;
678        while(bfs()){
679            ans += dfs(S, 0x3f3f3f3f) ;
680        }
681    }
682
683    int run(){
684        int ans = 0 ;
685        while(bfs()){
686            ans += dfs(S, 0x3f3f3f3f) ;
687        }
688    }
689
690    int run(){
691        int ans = 0 ;
692        while(bfs()){
693            ans += dfs(S, 0x3f3f3f3f) ;
694        }
695    }
696
697    int run(){
698        int ans = 0 ;
699        while(bfs()){
700            ans += dfs(S, 0x3f3f3f3f) ;
701        }
702    }
703
704    int run(){
705        int ans = 0 ;
706        while(bfs()){
707            ans += dfs(S, 0x3f3f3f3f) ;
708        }
709    }
710
711    int run(){
712        int ans = 0 ;
713        while(bfs()){
714            ans += dfs(S, 0x3f3f3f3f) ;
715        }
716    }
717
718    int run(){
719        int ans = 0 ;
720        while(bfs()){
721            ans += dfs(S, 0x3f3f3f3f) ;
722        }
723    }
724
725    int run(){
726        int ans = 0 ;
727        while(bfs()){
728            ans += dfs(S, 0x3f3f3f3f) ;
729        }
730    }
731
732    int run(){
733        int ans = 0 ;
734        while(bfs()){
735            ans += dfs(S, 0x3f3f3f3f) ;
736        }
737    }
738
739    int run(){
740        int ans = 0 ;
741        while(bfs()){
742            ans += dfs(S, 0x3f3f3f3f) ;
743        }
744    }
745
746    int run(){
747        int ans = 0 ;
748        while(bfs()){
749            ans += dfs(S, 0x3f3f3f3f) ;
750        }
751    }
752
753    int run(){
754        int ans = 0 ;
755        while(bfs()){
756            ans += dfs(S, 0x3f3f3f3f) ;
757        }
758    }
759
760    int run(){
761        int ans = 0 ;
762        while(bfs()){
763            ans += dfs(S, 0x3f3f3f3f) ;
764        }
765    }
766
767    int run(){
768        int ans = 0 ;
769        while(bfs()){
770            ans += dfs(S, 0x3f3f3f3f) ;
771        }
772    }
773
774    int run(){
775        int ans = 0 ;
776        while(bfs()){
777            ans += dfs(S, 0x3f3f3f3f) ;
778        }
779    }
780
781    int run(){
782        int ans = 0 ;
783        while(bfs()){
784            ans += dfs(S, 0x3f3f3f3f) ;
785        }
786    }
787
788    int run(){
789        int ans = 0 ;
790        while(bfs()){
791            ans += dfs(S, 0x3f3f3f3f) ;
792        }
793    }
794
795    int run(){
796        int ans = 0 ;
797        while(bfs()){
798            ans += dfs(S, 0x3f3f3f3f) ;
799        }
800    }
801
802    int run(){
803        int ans = 0 ;
804        while(bfs()){
805            ans += dfs(S, 0x3f3f3f3f) ;
806        }
807    }
808
809    int run(){
810        int ans = 0 ;
811        while(bfs()){
812            ans += dfs(S, 0x3f3f3f3f) ;
813        }
814    }
815
816    int run(){
817        int ans = 0 ;
818        while(bfs()){
819            ans += dfs(S, 0x3f3f3f3f) ;
820        }
821    }
822
823    int run(){
824        int ans = 0 ;
825        while(bfs()){
826            ans += dfs(S, 0x3f3f3f3f) ;
827        }
828    }
829
830    int run(){
831        int ans = 0 ;
832        while(bfs()){
833            ans += dfs(S, 0x3f3f3f3f) ;
834        }
835    }
836
837    int run(){
838        int ans = 0 ;
839        while(bfs()){
840            ans += dfs(S, 0x3f3f3f3f) ;
841        }
842    }
843
844    int run(){
845        int ans = 0 ;
846        while(bfs()){
847            ans += dfs(S, 0x3f3f3f3f) ;
848        }
849    }
850
851    int run(){
852        int ans = 0 ;
853        while(bfs()){
854            ans += dfs(S, 0x3f3f3f3f) ;
855        }
856    }
857
858    int run(){
859        int ans = 0 ;
860        while(bfs()){
861            ans += dfs(S, 0x3f3f3f3f) ;
862        }
863    }
864
865    int run(){
866        int ans = 0 ;
867        while(bfs()){
868            ans += dfs(S, 0x3f3f3f3f) ;
869        }
870    }
871
872    int run(){
873        int ans = 0 ;
874        while(bfs()){
875            ans += dfs(S, 0x3f3f3f3f) ;
876        }
877    }
878
879    int run(){
880        int ans = 0 ;
881        while(bfs()){
882            ans += dfs(S, 0x3f3f3f3f) ;
883        }
884    }
885
886    int run(){
887        int ans = 0 ;
888        while(bfs()){
889            ans += dfs(S, 0x3f3f3f3f) ;
890        }
891    }
892
893    int run(){
894        int ans = 0 ;
895        while(bfs()){
896            ans += dfs(S, 0x3f3f3f3f) ;
897        }
898    }
899
900    int run(){
901        int ans = 0 ;
902        while(bfs()){
903            ans += dfs(S, 0x3f3f3f3f) ;
904        }
905    }
906
907    int run(){
908        int ans = 0 ;
909        while(bfs()){
910            ans += dfs(S, 0x3f3f3f3f) ;
911        }
912    }
913
914    int run(){
915        int ans = 0 ;
916        while(bfs()){
917            ans += dfs(S, 0x3f3f3f3f) ;
918        }
919    }
920
921    int run(){
922        int ans = 0 ;
923        while(bfs()){
924            ans += dfs(S, 0x3f3f3f3f) ;
925        }
926    }
927
928    int run(){
929        int ans = 0 ;
930        while(bfs()){
931            ans += dfs(S, 0x3f3f3f3f) ;
932        }
933    }
934
935    int run(){
936        int ans = 0 ;
937        while(bfs()){
938            ans += dfs(S, 0x3f3f3f3f) ;
939        }
940    }
941
942    int run(){
943        int ans = 0 ;
944        while(bfs()){
945            ans += dfs(S, 0x3f3f3f3f) ;
946        }
947    }
948
949    int run(){
950        int ans = 0 ;
951        while(bfs()){
952            ans += dfs(S, 0x3f3f3f3f) ;
953        }
954    }
955
956    int run(){
957        int ans = 0 ;
958        while(bfs()){
959            ans += dfs(S, 0x3f3f3f3f) ;
960        }
961    }
962
963    int run(){
964        int ans = 0 ;
965        while(bfs()){
966            ans += dfs(S, 0x3f3f3f3f) ;
967        }
968    }
969
970    int run(){
971        int ans = 0 ;
972        while(bfs()){
973            ans += dfs(S, 0x3f3f3f3f) ;
974        }
975    }
976
977    int run(){
978        int ans = 0 ;
979        while(bfs()){
980            ans += dfs(S, 0x3f3f3f3f) ;
981        }
982    }
983
984    int run(){
985        int ans = 0 ;
986        while(bfs()){
987            ans += dfs(S, 0x3f3f3f3f) ;
988        }
989    }
990
991    int run(){
992        int ans = 0 ;
993        while(bfs()){
994            ans += dfs(S, 0x3f3f3f3f) ;
995        }
996    }
997
998    int run(){
999        int ans = 0 ;
1000       while(bfs()){
1001            ans += dfs(S, 0x3f3f3f3f) ;
1002        }
1003
1004    int run(){
1005        int ans = 0 ;
1006        while(bfs()){
1007            ans += dfs(S, 0x3f3f3f3f) ;
1008        }
1009    }
1010
1011    int run(){
1012        int ans = 0 ;
1013        while(bfs()){
1014            ans += dfs(S, 0x3f3f3f3f) ;
1015        }
1016    }
1017
1018    int run(){
1019        int ans = 0 ;
1020        while(bfs()){
1021            ans += dfs(S, 0x3f3f3f3f) ;
1022        }
1023    }
1024
1025    int run(){
1026        int ans = 0 ;
1027        while(bfs()){
1028            ans += dfs(S, 0x3f3f3f3f) ;
1029        }
1030    }
1031
1032    int run(){
1033        int ans = 0 ;
1034        while(bfs()){
1035            ans += dfs(S, 0x3f3f3f3f) ;
1036        }
1037    }
1038
1039    int run(){
1040        int ans = 0 ;
1041        while(bfs()){
1042            ans += dfs(S, 0x3f3f3f3f) ;
1043        }
1044    }
1045
1046    int run(){
1047        int ans = 0 ;
1048        while(bfs()){
1049            ans += dfs(S, 0x3f3f3f3f) ;
1050        }
1051    }
1052
1053    int run(){
1054        int ans = 0 ;
1055        while(bfs()){
1056            ans += dfs(S, 0x3f3f3f3f) ;
1057        }
1058    }
1059
1060    int run(){
1061        int ans = 0 ;
1062        while(bfs()){
1063            ans += dfs(S, 0x3f3f3f3f) ;
1064        }
1065    }
1066
1067    int run(){
1068        int ans = 0 ;
1069        while(bfs()){
1070            ans += dfs(S, 0x3f3f3f3f) ;
1071        }
1072    }
1073
1074    int run(){
1075        int ans = 0 ;
1076        while(bfs()){
1077            ans += dfs(S, 0x3f3f3f3f) ;
1078        }
1079    }
1080
1081    int run(){
1082        int ans = 0 ;
1083        while(bfs()){
1084            ans += dfs(S, 0x3f3f3f3f) ;
1085        }
1086    }
1087
1088    int run(){
1089        int ans = 0 ;
1090        while(bfs()){
1091            ans += dfs(S, 0x3f3f3f3f) ;
1092        }
1093    }
1094
1095    int run(){
1096        int ans = 0 ;
1097        while(bfs()){
1098            ans += dfs(S, 0x3f3f3f3f) ;
1099        }
1100    }
1101
1102    int run(){
1103        int ans = 0 ;
1104        while(bfs()){
1105            ans += dfs(S, 0x3f3f3f3f) ;
1106        }
1107    }
1108
1109    int run(){
1110        int ans = 0 ;
1111        while(bfs()){
1112            ans += dfs(S, 0x3f3f3f3f) ;
1113        }
1114    }
1115
1116    int run(){
1117        int ans = 0 ;
1118        while(bfs()){
1119            ans += dfs(S, 0x3f3f3f3f) ;
1120        }
1121    }
1122
1123    int run(){
1124        int ans = 0 ;
1125        while(bfs()){
1126            ans += dfs(S, 0x3f3f3f3f) ;
1127        }
1128    }
1129
1130    int run(){
1131        int ans = 0 ;
1132        while(bfs()){
1133            ans += dfs(S, 0x3f3f3f3f) ;
1134        }
1135    }
1136
1137    int run(){
1138        int ans = 0 ;
1139        while(bfs()){
1140            ans += dfs(S, 0x3f3f3f3f) ;
1141        }
1142    }
1143
1144    int run(){
1145        int ans = 0 ;
1146        while(bfs()){
1147            ans += dfs(S, 0x3f3f3f3f) ;
1148        }
1149    }
1150
1151    int run(){
1152        int ans = 0 ;
1153        while(bfs()){
1154            ans += dfs(S, 0x3f3f3f3f) ;
1155        }
1156    }
1157
1158    int run(){
1159        int ans = 0 ;
1160        while(bfs()){
1161            ans += dfs(S, 0x3f3f3f3f) ;
1162        }
1163    }
1164
1165    int run(){
1166        int ans = 0 ;
1167        while(bfs()){
1168            ans += dfs(S, 0x3f3f3f3f) ;
1169        }
1170    }
1171
1172    int run(){
1173        int ans = 0 ;
1174        while(bfs()){
1175            ans += dfs(S, 0x3f3f3f3f) ;
1176        }
1177    }
1178
1179    int run(){
1180        int ans = 0 ;
1181        while(bfs()){
1182            ans += dfs(S, 0x3f3f3f3f) ;
1183        }
1184    }
1185
1186    int run(){
1187        int ans = 0 ;
1188        while(bfs()){
1189            ans += dfs(S, 0x3f3f3f3f) ;
1190        }
1191    }
1192
1193    int run(){
1194        int ans = 0 ;
1195        while(bfs()){
1196            ans += dfs(S, 0x3f3f3f3f) ;
1197        }
1198    }
1199
1200    int run(){
1201        int ans = 0 ;
1202        while(bfs()){
1203            ans += dfs(S, 0x3f3f3f3f) ;
1204        }
1205    }
1206
1207    int run(){
1208        int ans = 0 ;
1209        while(bfs()){
1210            ans += dfs(S, 0x3f3f3f3f) ;
1211        }
1212    }
1213
1214    int run(){
1215        int ans = 0 ;
1216        while(bfs()){
1217            ans += dfs(S, 0x3f3f3f3f) ;
1218        }
1219    }
1220
1221    int run(){
1222        int ans = 0 ;
1223        while(bfs()){
1224            ans += dfs(S, 0x3f3f3f3f) ;
1225        }
1226    }
1227
1228    int run(){
1229        int ans = 0 ;
1230        while(bfs()){
1231            ans += dfs(S, 0x3f3f3f3f) ;
1232        }
1233    }
1234
1235    int run(){
1236        int ans = 0 ;
1237        while(bfs()){
1238            ans += dfs(S, 0x3f3f3f3f) ;
1239        }
1240    }
1241
1242    int run(){
1243        int ans = 0 ;
1244        while(bfs()){
1245            ans += dfs(S, 0x3f3
```

```

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%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 ^
13             1][s1] ;
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%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 DFS(v) ;
55 cnt[u] += cnt[v] ;
56 }
57
58 dp[u][1][0] = dp[u][1][1] = 0 ;
59
60 for ( auto [v, w] : edge[u] ){
61     for ( int i=cnt[u] ; i>1 ; i-- ) for (
62         int j=1 ; j<i && j<=cnt[v] ; j++ ){
63         dp[u][i][1] = min(dp[u][i][1],
64                            dp[u][i-j][1] + dp[v][j][1] + 2 *
65                            w) ;
66         dp[u][i][0] = min(dp[u][i][0],
67                            dp[u][i-j][1] + dp[v][j][0] + w) ;
68         dp[u][i][0] = min(dp[u][i][0],
69                            dp[u][i-j][0] + dp[v][j][1] + 2 *
70                            w) ;
71     }
72 }
73
74 int main(){
75     int t = 0 ;
76
77     while(~scanf("%d", &n) && n){
78         init() ;
79         for ( int i=0 ; i<n-1 ; i++ ){
80             int u, v, w ;
81             scanf("%d%d%d", &v, &u, &w) ;
82             edge[u].push_back({v, w}) ;
83         }
84
85         DFS(0) ;
86         printf("Case %d:\n", ++t) ;
87
88         int q, e ;
89         scanf("%d", &q) ;
90
91         while(q--){
92             scanf("%d", &e) ;
93
94             for ( int i=n ; i>=1 ; i-- )
95                 if(dp[0][i][0] <= e){
96                     printf("%d\n", i) ;
97                     break ;
98                 }
99             }
100
101         return 0 ;
102     }
103 }
```

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 }
```