

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 AND Number Theory	2
2.1 formula	2
2.2 extended gcd	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 ver1	5
3.7 BIT ver2	5
3.8 BIT ver3	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) #歐幾里德範數
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
      radix)
9 static Integer valueOf(int i)
10 static Integer valueOf(String s)
11 static Integer valueOf(String s, int radix)
12 static String toString(int i)
13 static String toString(int i, int radix)
14 static String toUnsignedString(int i)
15 static String toUnsignedString(int i, int
      radix)
16 static long toUnsignedLong(int x)
17 static Integer decode(String nm)
      // 支援 0x/0/# 前綴
18 static Integer getInteger(String nm[, int
      val]) // 從系統屬性讀取整數
19
20 // 比較/雜湊/聚合
21 static int compare(int x, int y)
22 static int compareUnsigned(int x, int y)
23 static int hashCode(int value)
24 static int min(int a, int b)
25 static int max(int a, int b)
26 static int sum(int a, int b)
27
28 // 位元操作
29 static int bitCount(int i) // 設定位數
30 static int highestOneBit(int i)
31 static int lowestOneBit(int i)
32 static int numberOfLeadingZeros(int i)
33 static int numberOfTrailingZeros(int i)
34 static int rotateLeft(int i, int distance)
35 static int rotateRight(int i, int distance)
36 static int reverse(int i)
37 static int reverseBytes(int i)
38
39 // 無號運算
40 static int divideUnsigned(int dividend, int
      divisor)

```

```

41 static int remainderUnsigned(int dividend,
      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
      endIndex)
9 boolean contains(CharSequence s)
10 boolean startsWith(String prefix[, int
      toffset])
11 boolean endsWith(String suffix)
12 int indexOf(String str[, int fromIndex])
13 int lastIndexOf(String str[, int
      fromIndex])
14
15 // 取子字串/子序列
16 String substring(int beginIndex)
17 String substring(int beginIndex, int
      endIndex)
18 CharSequence subSequence(int beginIndex, int
      endIndex)
19
20 // 比較/等價
21 boolean equals(Object obj)
22 boolean equalsIgnoreCase(String
      anotherString)
23 int compareTo(String anotherString)
24 int compareToIgnoreCase(String str)
25 boolean matches(String regex)
26 boolean regionMatches(int toffset, String
      other, int offset, int len)
27 boolean regionMatches(boolean ignoreCase,
      int toffset, String other, int offset,
      int len)
28
29 // 建構/轉換/連接
30 String concat(String str)
31 String replace(char oldChar, char newChar)
32 String replace(CharSequence target,
      CharSequence replacement)
33 String replaceAll(String regex, String
      replacement)
34 String replaceFirst(String regex, String
      replacement)
35 String[] split(String regex[, int limit])
36 String toLowerCase()
37 String toUpperCase()
38 String trim()
39 String strip() // (since 11)
40 String stripLeading() // (since 11)
41 String stripTrailing() // (since 11)
42 String repeat(int count) // (since 11)
43 IntStream chars()
44 Stream<String> lines() // (since 11)
45 String intern()
46
47 // 靜態工具
48 static String format(String format,
      Object... args)
49 static String join(CharSequence delimiter,
      CharSequence... elements)
50 static String join(CharSequence delimiter,
      Iterable<? extends CharSequence>
      elements)
51 static String
      valueOf(primitive/char[]/Object)
52 static String copyValueOf(char[] data[, int
      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/幂根
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 AND Number Theory

2.1 formula

1. Catalan Number

$$C_n = \frac{1}{n} \binom{2n}{n}, C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

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

2. Euler's Formula

對於 V 個點, E 條邊, F 個面, C 個連通分量

$$V + F = E + 2$$

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

3. Pick's Theorem

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

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

4. 中國剩餘定理

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<long long, long long> crt(long
    long a1, long long m1,
    long long
    a2,
    long long
    m2) {

```

```

5 long long g = std::gcd(m1, m2);
6 if ((a2 - a1) % g != 0) return {0,
    -1}; // no solution
7
8 long long lcm = m1 / g * m2;
9 long long m1_reduced = m1 / g;
10 long long m2_reduced = m2 / g;
11
12 long long diff = (a2 - a1) / g %
    m2_reduced;
13 if (diff < 0) diff += m2_reduced;
14
15 long long inv =
    mod_pow(m1_reduced,
    m2_reduced - 1, m2_reduced);
16 long long step = diff * inv %
    m2_reduced;
17 long long 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。

5. 組合數學

Combinatorics Cheat Sheet

Binomial Coefficient Identities

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

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

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

Stars and Bars 非負整數解數量:

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

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 :

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

計算滿足限制的排列或整數解時廣泛使用。

Catalan Numbers 基本定義:

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

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。

6. 數論

Number Theory Essentials

Fermat's Little Theorem: $a^{p-1} \equiv 1 \pmod{p}$ ($\gcd(a, p) = 1$)

Euler's Theorem: $a^{\varphi(n)} \equiv 1 \pmod{n}$ ($\gcd(a, n) = 1$)

Modular Inverse: $a^{-1} \equiv a^{p-2} \pmod{p}$ ($\gcd(a, p) = 1$)

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

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

其中 μ 為莫比烏斯函數；常用於反演及計算互質對數量。

2.2 extended gcd

給定 a, b, c , 求 $ax + by = c$ 的解

```
1 ll extgcd(ll a, ll b, ll c, ll &x, ll
2     &y){
3     if(b == 0){
4         x = c/a ;
5         y = 0 ;
6         return a ;
7     }
8     ll d = extgcd(b, a%b, c, x, y), tmp =
9         x ;
10    x = y ;
11    y = tmp - (a/b)*y ;
12    return d ;
13 }
```

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 }
38
39 void MST(){
40     int cnt = 0 ;
41     for ( int i=0 ; i<edge.size() && cnt < n-1
42             ; i++ ){
43         auto [u, v, w] = edge[i] ;
44         if(merge(u, v)){
45             cnt++ ;
46             sum -= w ;
47         }
48     }
49
50 int main(){
51     for ( int i=0 ; i<m ; i++ ){
52         scanf("%d%d%d", &u, &v, &w) ;
53         edge.push_back({u, v, w}) ;
54         sum += w ;
55     }
56
57     sort(edge.begin(), edge.end(),
58          greater<Edge>()) ;
59     MST() ;
60 }
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86 }
```

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 }
```

```
35
36     void push(int id){
37         AddTag(lc, seg[id].tag) ;
38         AddTag(rc, seg[id].tag) ;
39         seg[id].tag = {0, 0} ;
40     }
41 public:
42     void build(int L=1, int R=n, int id=1){
43         seg[id].sum = 0 ;
44         seg[id].tag = {0, 0} ;
45         seg[id].sz = 1 ;
46
47         if(L == R){
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     }
94 }
```

3.3 HLD

```
1 /* HLD */
2 int fa[Maxn], top[Maxn], son[Maxn],
3     sz[Maxn], dep[Maxn] = {0}, dfn[Maxn],
4     rnk[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     }
15 }
```

```

12     if(son[u] == -1 || sz[v] > sz[son[u]])    84      if(dep[top[u]] < dep[top[v]]) swap(u,      36      auto upd = [&](auto &&self, const int
13         son[u] = v ;                           v) ;           cur, int L, int R){
14     }                                         modify(dfn[top[u]], dfn[u], val) ;          37      int cur_cnt = cnt++ ;
15
16 void dfs2(int u, int t){                   85      u = fa[top[u]] ;                      38
17     top[u] = t ;                         86      }                                         if(L == R){
18     dfn[u] = ++dfscnt ;                  87      seg[cur_cnt] = {seg[cur].sum + 1, 0,      39
19     rk[dfscnt] = u ;                    88      }                                         0} ;
20
21 if(son[u] == -1) return ;                 89      return cur_cnt ;
22
23 dfs2(son[u], t) ;                     90      // get sum from u to v (simple path)
24
25 for ( auto v : g[u] ) if(v != fa[u] && v 91      int get(int u, int v){
26     != son[u]) {                      92      pair<int, ColorSeg> U, V ;
27         dfs2(v, v) ;                  93      ColorSeg M ;
28     }                                         U = {u, {0, 0, 0}} ;
29 /* Segment Tree */                   94      V = {v, {0, 0, 0}} ;
30 #define lc (id << 1)                  95      while(top[U.first] != top[V.first]){
31 #define rc ((id << 1) | 1)            96          if(dep[top[U.first]] <
32
33 struct ColorSeg{                      97              dep[top[V.first]]) swap(U, V) ;
34     int left, right, tot ;             98          U.second = query(dfn[top[U.first]],
35
36     ColorSeg operator+(const ColorSeg &o) 99              dfn[U.first]) + U.second ;
37         const {                      100             U.first = fa[top[U.first]] ;
38             if(tot == 0) return o ;       101             if(dep[U.first] < dep[V.first]) swap(U,
39             if(o.tot == 0) return *this ;   102                 V) ;
40
41     ColorSeg tmp ;                  103             M = query(dfn[V.first], dfn[U.first]) ;
42     tmp.left = left ;                104             return (U.second.tot + V.second.tot +
43     tmp.right = o.right ;             105                 M.tot) - (U.second.left == M.right)
44     tmp.tot = tot + o.tot - (right == 106                 - (V.second.left == M.left) ;
45         o.left) ;                  107             }
46
47     return tmp ;                  108         }tree ;
48 } ;                                     109         void init(){
49
50 struct Node{                          110             memset(son, -1, sizeof(son)) ;
51     ColorSeg color ;                  111         }
52     int tag ;                        112     }
53 }seg[Maxn << 2] ;
54
55 class SegmentTree{
56 private:
57     void pull(int id){               1 // Find range k-th largest number
58         // normal pull
59     }
60
61     void AddTag(int id, int tag){   2 struct Node{
62         // normal AddTag
63     }
64
65     void push(int id){              3     int sum, left, right ;
66         // normal push
67     }
68
69     void modify(int l, int r, int tag, int 4 }seg[Maxn + 20 * Maxn] ;
70         L=1, int R=n, int id=1){      5
71         // normal modify
72     }
73
74     ColorSeg query(int l, int r, int L=1, int 6 class PersistentSegmentTree{
75         R=n, int id=1){              7 private:
76         // normal query
77     }
78
79     public:
80         void build(int L=1, int R=n, int id=1){ 8             int n ;
81             // normal build
82         }
83
84         // update val from u to v (simple path)
85         void update(int u, int v, int val){ 9             int cnt = 0 ;
86             while(top[u] != top[v]){
87                 if(dep[top[u]] < dep[top[v]]) swap(u, 10             for (int i=0 ; i<26 ; i++) next[i]
88                     v) ;                         = nullptr ;
89                 modify(dfn[top[u]], dfn[u], val) ; 11             class Trie{
90                 u = fa[top[u]] ;                  12             private:
91             }                                         int cnt ;
92             }                                         TrieNode *root ;
93             if(dep[u] < dep[v]) swap(u, v) ;
94             modify(dfn[v], dfn[u], val) ;
95         }
96
97         // get sum from u to v (simple path)
98         int get(int u, int v){
99             pair<int, ColorSeg> U, V ;
100             ColorSeg M ;
101             U = {u, {0, 0, 0}} ;
102             V = {v, {0, 0, 0}} ;
103             while(top[U.first] != top[V.first]){
104                 if(dep[top[U.first]] <
105                     dep[top[V.first]]) swap(U, V) ;
106                 U.second = query(dfn[top[U.first]],
107                     dfn[U.first]) + U.second ;
108                 U.first = fa[top[U.first]] ;
109             }
110             if(dep[U.first] < dep[V.first]) swap(U,
111                 V) ;
112             M = query(dfn[V.first], dfn[U.first]) ;
113             return (U.second.tot + V.second.tot +
114                 M.tot) - (U.second.left == M.right)
115                 - (V.second.left == M.left) ;
116         }
117
118         void init(){
119             memset(son, -1, sizeof(son)) ;
120         }
121
122         // Find range k-th largest number
123         struct Node{
124             int sum, left, right ;
125         }seg[Maxn + 20 * Maxn] ;
126
127         class PersistentSegmentTree{
128     private:
129         int n ;
130         int cnt ;
131         vector<int> version ;
132
133         int build(int L, int R){
134             int cur_cnt = cnt++ ;
135             if(L == R){
136                 seg[cur_cnt] = {0, 0, 0} ;
137                 return cur_cnt ;
138             }
139
140             int M = (L + R) >> 1 ;
141             int lc = build(L, M) ;
142             int rc = build(M+1, R) ;
143
144             seg[cur_cnt] = {0, lc, rc} ;
145             return cur_cnt ;
146         }
147
148         public:
149             PersistentSegmentTree(int _n){
150                 n = _n ;
151                 cnt = 0 ;
152
153                 int root = build(1, n) ;
154                 version.push_back(root) ;
155             }
156
157             void update(int ver, int idx){
158                 if(dep[top[u]] < dep[top[v]]) swap(u, 159                     v) ;
160                 modify(dfn[top[u]], dfn[u], val) ;
161                 u = fa[top[u]] ;
162             }
163
164             if(dep[u] < dep[v]) swap(u, v) ;
165             modify(dfn[v], dfn[u], val) ;
166         }
167
168         // get sum from u to v (simple path)
169         int get(int u, int v){
170             pair<int, ColorSeg> U, V ;
171             ColorSeg M ;
172             U = {u, {0, 0, 0}} ;
173             V = {v, {0, 0, 0}} ;
174             while(top[U.first] != top[V.first]){
175                 if(dep[top[U.first]] <
176                     dep[top[V.first]]) swap(U, V) ;
177                 U.second = query(dfn[top[U.first]],
178                     dfn[U.first]) + U.second ;
179                 U.first = fa[top[U.first]] ;
180             }
181             if(dep[U.first] < dep[V.first]) swap(U,
182                 V) ;
183             M = query(dfn[V.first], dfn[U.first]) ;
184             return (U.second.tot + V.second.tot +
185                 M.tot) - (U.second.left == M.right)
186                 - (V.second.left == M.left) ;
187         }
188
189         void init(){
190             memset(son, -1, sizeof(son)) ;
191         }
192
193         // Find range k-th largest number
194         struct Node{
195             int sum, left, right ;
196         }seg[Maxn + 20 * Maxn] ;
197
198         class PersistentSegmentTree{
199     private:
200         int n ;
201         int cnt ;
202         vector<int> version ;
203
204         int build(int L, int R){
205             int cur_cnt = cnt++ ;
206             if(L == R){
207                 seg[cur_cnt] = {0, 0, 0} ;
208                 return cur_cnt ;
209             }
210
211             int M = (L + R) >> 1 ;
212             int lc = build(L, M) ;
213             int rc = build(M+1, R) ;
214
215             seg[cur_cnt] = {0, lc, rc} ;
216             return cur_cnt ;
217         }
218
219         public:
220             PersistentSegmentTree(int _n){
221                 n = _n ;
222                 cnt = 0 ;
223
224                 int root = build(1, n) ;
225                 version.push_back(root) ;
226             }
227
228             void update(int ver, int idx){
229                 if(dep[top[u]] < dep[top[v]]) swap(u, 23                     v) ;
230                 modify(dfn[top[u]], dfn[u], val) ;
231                 u = fa[top[u]] ;
232             }
233
234             if(dep[u] < dep[v]) swap(u, v) ;
235             modify(dfn[v], dfn[u], val) ;
236         }
237
238         // get sum from u to v (simple path)
239         int get(int u, int v){
240             pair<int, ColorSeg> U, V ;
241             ColorSeg M ;
242             U = {u, {0, 0, 0}} ;
243             V = {v, {0, 0, 0}} ;
244             while(top[U.first] != top[V.first]){
245                 if(dep[top[U.first]] <
246                     dep[top[V.first]]) swap(U, V) ;
247                 U.second = query(dfn[top[U.first]],
248                     dfn[U.first]) + U.second ;
249                 U.first = fa[top[U.first]] ;
250             }
251             if(dep[U.first] < dep[V.first]) swap(U,
252                 V) ;
253             M = query(dfn[V.first], dfn[U.first]) ;
254             return (U.second.tot + V.second.tot +
255                 M.tot) - (U.second.left == M.right)
256                 - (V.second.left == M.left) ;
257         }
258
259         void init(){
260             memset(son, -1, sizeof(son)) ;
261         }
262
263         // Find range k-th largest number
264         struct Node{
265             int sum, left, right ;
266         }seg[Maxn + 20 * Maxn] ;
267
268         class PersistentSegmentTree{
269     private:
270         int n ;
271         int cnt ;
272         vector<int> version ;
273
274         int build(int L, int R){
275             int cur_cnt = cnt++ ;
276             if(L == R){
277                 seg[cur_cnt] = {0, 0, 0} ;
278                 return cur_cnt ;
279             }
280
281             int M = (L + R) >> 1 ;
282             int lc = build(L, M) ;
283             int rc = build(M+1, R) ;
284
285             seg[cur_cnt] = {0, lc, rc} ;
286             return cur_cnt ;
287         }
288
289         public:
290             PersistentSegmentTree(int _n){
291                 n = _n ;
292                 cnt = 0 ;
293
294                 int root = build(1, n) ;
295                 version.push_back(root) ;
296             }
297
298             void update(int ver, int idx){
299                 if(dep[top[u]] < dep[top[v]]) swap(u, 30                     v) ;
300                 modify(dfn[top[u]], dfn[u], val) ;
301                 u = fa[top[u]] ;
302             }
303
304             if(dep[u] < dep[v]) swap(u, v) ;
305             modify(dfn[v], dfn[u], val) ;
306         }
307
308         // get sum from u to v (simple path)
309         int get(int u, int v){
310             pair<int, ColorSeg> U, V ;
311             ColorSeg M ;
312             U = {u, {0, 0, 0}} ;
313             V = {v, {0, 0, 0}} ;
314             while(top[U.first] != top[V.first]){
315                 if(dep[top[U.first]] <
316                     dep[top[V.first]]) swap(U, V) ;
317                 U.second = query(dfn[top[U.first]],
318                     dfn[U.first]) + U.second ;
319                 U.first = fa[top[U.first]] ;
320             }
321             if(dep[U.first] < dep[V.first]) swap(U,
322                 V) ;
323             M = query(dfn[V.first], dfn[U.first]) ;
324             return (U.second.tot + V.second.tot +
325                 M.tot) - (U.second.left == M.right)
326                 - (V.second.left == M.left) ;
327         }
328
329         void init(){
330             memset(son, -1, sizeof(son)) ;
331         }
332
333         // Find range k-th largest number
334         struct Node{
335             int sum, left, right ;
336         }seg[Maxn + 20 * Maxn] ;
337
338         class PersistentSegmentTree{
339     private:
340         int n ;
341         int cnt ;
342         vector<int> version ;
343
344         int build(int L, int R){
345             int cur_cnt = cnt++ ;
346             if(L == R){
347                 seg[cur_cnt] = {0, 0, 0} ;
348                 return cur_cnt ;
349             }
350
351             int M = (L + R) >> 1 ;
352             int lc = build(L, M) ;
353             int rc = build(M+1, R) ;
354
355             seg[cur_cnt] = {0, lc, rc} ;
356             return cur_cnt ;
357         }
358
359         public:
360             PersistentSegmentTree(int _n){
361                 n = _n ;
362                 cnt = 0 ;
363
364                 int root = build(1, n) ;
365                 version.push_back(root) ;
366             }
367
368             void update(int ver, int idx){
369                 if(dep[top[u]] < dep[top[v]]) swap(u, 37                     v) ;
370                 modify(dfn[top[u]], dfn[u], val) ;
371                 u = fa[top[u]] ;
372             }
373
374             if(dep[u] < dep[v]) swap(u, v) ;
375             modify(dfn[v], dfn[u], val) ;
376         }
377
378         // get sum from u to v (simple path)
379         int get(int u, int v){
380             pair<int, ColorSeg> U, V ;
381             ColorSeg M ;
382             U = {u, {0, 0, 0}} ;
383             V = {v, {0, 0, 0}} ;
384             while(top[U.first] != top[V.first]){
385                 if(dep[top[U.first]] <
386                     dep[top[V.first]]) swap(U, V) ;
387                 U.second = query(dfn[top[U.first]],
388                     dfn[U.first]) + U.second ;
389                 U.first = fa[top[U.first]] ;
390             }
391             if(dep[U.first] < dep[V.first]) swap(U,
392                 V) ;
393             M = query(dfn[V.first], dfn[U.first]) ;
394             return (U.second.tot + V.second.tot +
395                 M.tot) - (U.second.left == M.right)
396                 - (V.second.left == M.left) ;
397         }
398
399         void init(){
400             memset(son, -1, sizeof(son)) ;
401         }
402
403         // Find range k-th largest number
404         struct Node{
405             int sum, left, right ;
406         }seg[Maxn + 20 * Maxn] ;
407
408         class PersistentSegmentTree{
409     private:
410         int n ;
411         int cnt ;
412         vector<int> version ;
413
414         int build(int L, int R){
415             int cur_cnt = cnt++ ;
416             if(L == R){
417                 seg[cur_cnt] = {0, 0, 0} ;
418                 return cur_cnt ;
419             }
420
421             int M = (L + R) >> 1 ;
422             int lc = build(L, M) ;
423             int rc = build(M+1, R) ;
424
425             seg[cur_cnt] = {0, lc, rc} ;
426             return cur_cnt ;
427         }
428
429         public:
430             PersistentSegmentTree(int _n){
431                 n = _n ;
432                 cnt = 0 ;
433
434                 int root = build(1, n) ;
435                 version.push_back(root) ;
436             }
437
438             void update(int ver, int idx){
439                 if(dep[top[u]] < dep[top[v]]) swap(u, 44                     v) ;
440                 modify(dfn[top[u]], dfn[u], val) ;
441                 u = fa[top[u]] ;
442             }
443
444             if(dep[u] < dep[v]) swap(u, v) ;
445             modify(dfn[v], dfn[u], val) ;
446         }
447
448         // get sum from u to v (simple path)
449         int get(int u, int v){
450             pair<int, ColorSeg> U, V ;
451             ColorSeg M ;
452             U = {u, {0, 0, 0}} ;
453             V = {v, {0, 0, 0}} ;
454             while(top[U.first] != top[V.first]){
455                 if(dep[top[U.first]] <
456                     dep[top[V.first]]) swap(U, V) ;
457                 U.second = query(dfn[top[U.first]],
458                     dfn[U.first]) + U.second ;
459                 U.first = fa[top[U.first]] ;
460             }
461             if(dep[U.first] < dep[V.first]) swap(U,
462                 V) ;
463             M = query(dfn[V.first], dfn[U.first]) ;
464             return (U.second.tot + V.second.tot +
465                 M.tot) - (U.second.left == M.right)
466                 - (V.second.left == M.left) ;
467         }
468
469         void init(){
470             memset(son, -1, sizeof(son)) ;
471         }
472
473         // Find range k-th largest number
474         struct Node{
475             int sum, left, right ;
476         }seg[Maxn + 20 * Maxn] ;
477
478         class PersistentSegmentTree{
479     private:
480         int n ;
481         int cnt ;
482         vector<int> version ;
483
484         int build(int L, int R){
485             int cur_cnt = cnt++ ;
486             if(L == R){
487                 seg[cur_cnt] = {0, 0, 0} ;
488                 return cur_cnt ;
489             }
490
491             int M = (L + R) >> 1 ;
492             int lc = build(L, M) ;
493             int rc = build(M+1, R) ;
494
495             seg[cur_cnt] = {0, lc, rc} ;
496             return cur_cnt ;
497         }
498
499         public:
500             PersistentSegmentTree(int _n){
501                 n = _n ;
502                 cnt = 0 ;
503
504                 int root = build(1, n) ;
505                 version.push_back(root) ;
506             }
507
508             void update(int ver, int idx){
509                 if(dep[top[u]] < dep[top[v]]) swap(u, 51                     v) ;
510                 modify(dfn[top[u]], dfn[u], val) ;
511                 u = fa[top[u]] ;
512             }
513
514             if(dep[u] < dep[v]) swap(u, v) ;
515             modify(dfn[v], dfn[u], val) ;
516         }
517
518         // get sum from u to v (simple path)
519         int get(int u, int v){
520             pair<int, ColorSeg> U, V ;
521             ColorSeg M ;
522             U = {u, {0, 0, 0}} ;
523             V = {v, {0, 0, 0}} ;
524             while(top[U.first] != top[V.first]){
525                 if(dep[top[U.first]] <
526                     dep[top[V.first]]) swap(U, V) ;
527                 U.second = query(dfn[top[U.first]],
528                     dfn[U.first]) + U.second ;
529                 U.first = fa[top[U.first]] ;
530             }
531             if(dep[U.first] < dep[V.first]) swap(U,
532                 V) ;
533             M = query(dfn[V.first], dfn[U.first]) ;
534             return (U.second.tot + V.second.tot +
535                 M.tot) - (U.second.left == M.right)
536                 - (V.second.left == M.left) ;
537         }
538
539         void init(){
540             memset(son, -1, sizeof(son)) ;
541         }
542
543         // Find range k-th largest number
544         struct Node{
545             int sum, left, right ;
546         }seg[Maxn + 20 * Maxn] ;
547
548         class PersistentSegmentTree{
549     private:
550         int n ;
551         int cnt ;
552         vector<int> version ;
553
554         int build(int L, int R){
555             int cur_cnt = cnt++ ;
556             if(L == R){
557                 seg[cur_cnt] = {0, 0, 0} ;
558                 return cur_cnt ;
559             }
560
561             int M = (L + R) >> 1 ;
562             int lc = build(L, M) ;
563             int rc = build(M+1, R) ;
564
565             seg[cur_cnt] = {0, lc, rc} ;
566             return cur_cnt ;
567         }
568
569         public:
570             PersistentSegmentTree(int _n){
571                 n = _n ;
572                 cnt = 0 ;
573
574                 int root = build(1, n) ;
575                 version.push_back(root) ;
576             }
577
578             void update(int ver, int idx){
579                 if(dep[top[u]] < dep[top[v]]) swap(u, 58                     v) ;
580                 modify(dfn[top[u]], dfn[u], val) ;
581                 u = fa[top[u]] ;
582             }
583
584             if(dep[u] < dep[v]) swap(u, v) ;
585             modify(dfn[v], dfn[u], val) ;
586         }
587
588         // get sum from u to v (simple path)
589         int get(int u, int v){
590             pair<int, ColorSeg> U, V ;
591             ColorSeg M ;
592             U = {u, {0, 0, 0}} ;
593             V = {v, {0, 0, 0}} ;
594             while(top[U.first] != top[V.first]){
595                 if(dep[top[U.first]] <
596                     dep[top[V.first]]) swap(U, V) ;
597                 U.second = query(dfn[top[U.first]],
598                     dfn[U.first]) + U.second ;
599                 U.first = fa[top[U.first]] ;
600             }
601             if(dep[U.first] < dep[V.first]) swap(U,
602                 V) ;
603             M = query(dfn[V.first], dfn[U.first]) ;
604             return (U.second.tot + V.second.tot +
605                 M.tot) - (U.second.left == M.right)
606                 - (V.second.left == M.left) ;
607         }
608
609         void init(){
610             memset(son, -1, sizeof(son)) ;
611         }
612
613         // Find range k-th largest number
614         struct Node{
615             int sum, left, right ;
616         }seg[Maxn + 20 * Maxn] ;
617
618         class PersistentSegmentTree{
619     private:
620         int n ;
621         int cnt ;
622         vector<int> version ;
623
624         int build(int L, int R){
625             int cur_cnt = cnt++ ;
626             if(L == R){
627                 seg[cur_cnt] = {0, 0, 0} ;
628                 return cur_cnt ;
629             }
630
631             int M = (L + R) >> 1 ;
632             int lc = build(L, M) ;
633             int rc = build(M+1, R) ;
634
635             seg[cur_cnt] = {0, lc, rc} ;
636             return cur_cnt ;
637         }
638
639         public:
640             PersistentSegmentTree(int _n){
641                 n = _n ;
642                 cnt = 0 ;
643
644                 int root = build(1, n) ;
645                 version.push_back(root) ;
646             }
647
648             void update(int ver, int idx){
649                 if(dep[top[u]] < dep[top[v]]) swap(u, 65                     v) ;
650                 modify(dfn[top[u]], dfn[u], val) ;
651                 u = fa[top[u]] ;
652             }
653
654             if(dep[u] < dep[v]) swap(u, v) ;
655             modify(dfn[v], dfn[u], val) ;
656         }
657
658         // get sum from u to v (simple path)
659         int get(int u, int v){
660             pair<int, ColorSeg> U, V ;
661             ColorSeg M ;
662             U = {u, {0, 0, 0}} ;
663             V = {v, {0, 0, 0}} ;
664             while(top[U.first] != top[V.first]){
665                 if(dep[top[U.first]] <
666                     dep[top[V.first]]) swap(U, V) ;
667                 U.second = query(dfn[top[U.first]],
668                     dfn[U.first]) + U.second ;
669                 U.first = fa[top[U.first]] ;
670             }
671             if(dep[U.first] < dep[V.first]) swap(U,
672                 V) ;
673             M = query(dfn[V.first], dfn[U.first]) ;
674             return (U.second.tot + V.second.tot +
675                 M.tot) - (U.second.left == M.right)
676                 - (V.second.left == M.left) ;
677         }
678
679         void init(){
680             memset(son, -1, sizeof(son)) ;
681         }
682
683         // Find range k-th largest number
684         struct Node{
685             int sum, left, right ;
686         }seg[Maxn + 20 * Maxn] ;
687
688         class PersistentSegmentTree{
689     private:
690         int n ;
691         int cnt ;
692         vector<int> version ;
693
694         int build(int L, int R){
695             int cur_cnt = cnt++ ;
696             if(L == R){
697                 seg[cur_cnt] = {0, 0, 0} ;
698                 return cur_cnt ;
699             }
700
701             int M = (L + R) >> 1 ;
702             int lc = build(L, M) ;
703             int rc = build(M+1, R) ;
704
705             seg[cur_cnt] = {0, lc, rc} ;
706             return cur_cnt ;
707         }
708
709         public:
710             PersistentSegmentTree(int _n){
711                 n = _n ;
712                 cnt = 0 ;
713
714                 int root = build(1, n) ;
715                 version.push_back(root) ;
716             }
717
718             void update(int ver, int idx){
719                 if(dep[top[u]] < dep[top[v]]) swap(u, 72                     v) ;
720                 modify(dfn[top[u]], dfn[u], val) ;
721                 u = fa[top[u]] ;
722             }
723
724             if(dep[u] < dep[v]) swap(u, v) ;
725             modify(dfn[v], dfn[u], val) ;
726         }
727
728         // get sum from u to v (simple path)
729         int get(int u, int v){
730             pair<int, ColorSeg> U, V ;
731             ColorSeg M ;
732             U = {u, {0, 0, 0}} ;
733             V = {v, {0, 0, 0}} ;
734             while(top[U.first] != top[V.first]){
735                 if(dep[top[U.first]] <
736                     dep[top[V.first]]) swap(U, V) ;
737                 U.second = query(dfn[top[U.first]],
738                     dfn[U.first]) + U.second ;
739                 U.first = fa[top[U.first]] ;
740             }
741             if(dep[U.first] < dep[V.first]) swap(U,
742                 V) ;
743             M = query(dfn[V.first], dfn[U.first]) ;
744             return (U.second.tot + V.second.tot +
745                 M.tot) - (U.second.left == M.right)
746                 - (V.second.left == M.left) ;
747         }
748
749         void init(){
750             memset(son, -1, sizeof(son)) ;
751         }
752
753         // Find range k-th largest number
754         struct Node{
755             int sum, left, right ;
756         }seg[Maxn + 20 * Maxn] ;
757
758         class PersistentSegmentTree{
759     private:
760         int n ;
761         int cnt ;
762         vector<int> version ;
763
764         int build(int L, int R){
765             int cur_cnt = cnt++ ;
766             if(L == R){
767                 seg[cur_cnt] = {0, 0, 0} ;
768                 return cur_cnt ;
769             }
770
771             int M = (L + R) >> 1 ;
772             int lc = build(L, M) ;
773             int rc = build(M+1, R) ;
774
775             seg[cur_cnt] = {0, lc, rc} ;
776             return cur_cnt ;
777         }
778
779         public:
780             PersistentSegmentTree(int _n){
781                 n = _n ;
782                 cnt = 0 ;
783
784                 int root = build(1, n) ;
785                 version.push_back(root) ;
786             }
787
788             void update(int ver, int idx){
789                 if(dep[top[u]] < dep[top[v]]) swap(u, 79                     v) ;
790                 modify(dfn[top[u]], dfn[u], val) ;
791                 u = fa[top[u]] ;
792             }
793
794             if(dep[u] < dep[v]) swap(u, v) ;
795             modify(dfn[v], dfn[u], val) ;
796         }
797
798         // get sum from u to v (simple path)
799         int get(int u, int v){
800             pair<int, ColorSeg> U, V ;
801             ColorSeg M ;
802             U = {u, {0, 0, 0}} ;
803             V = {v, {0, 0, 0}} ;
804             while(top[U.first] != top[V.first]){
805                 if(dep[top[U.first]] <
806                     dep[top[V.first]]) swap(U, V) ;
807                 U.second = query(dfn[top[U.first]],
808                     dfn[U.first]) + U.second ;
809                 U.first = fa[top[U.first]] ;
810             }
811             if(dep[U.first] < dep[V.first]) swap(U,
812                 V) ;
813             M = query(dfn[V.first], dfn[U.first]) ;
814             return (U.second.tot + V.second.tot +
815                 M.tot) - (U.second.left == M.right)
816                 - (V.second.left == M.left) ;
817         }
818
819         void init(){
820             memset(son, -1, sizeof(son)) ;
821         }
822
823         // Find range k-th largest number
824         struct Node{
825             int sum, left, right ;
826         }seg[Maxn + 20 * Maxn] ;
827
828         class PersistentSegmentTree{
829     private:
830         int n ;
831         int cnt ;
832         vector<int> version ;
833
834         int build(int L, int R){
835             int cur_cnt = cnt++ ;
836             if(L == R){
837                 seg[cur_cnt] = {0, 0, 0} ;
838                 return cur_cnt ;
839             }
840
841             int M = (L + R) >> 1 ;
842             int lc = build(L, M) ;
843             int rc = build(M+1, R) ;
844
845             seg[cur_cnt] = {0, lc, rc} ;
846             return cur_cnt ;
847         }
848
849         public:
850             PersistentSegmentTree(int _n){
851                 n = _n ;
852                 cnt = 0 ;
853
854                 int root = build(1, n) ;
855                 version.push_back(root) ;
856             }
857
858             void update(int ver, int idx){
859                 if(dep[top[u]] < dep[top[v]]) swap(u, 86                     v) ;
860                 modify(dfn[top[u]], dfn[u], val) ;
861                 u = fa[top[u]] ;
862             }
863
864             if(dep[u] < dep[v]) swap(u, v) ;
865             modify(dfn[v], dfn[u], val) ;
866         }
867
868         // get sum from u to v (simple path)
869         int get(int u, int v){
870             pair<int, ColorSeg> U, V ;
871             ColorSeg M ;
872             U = {u, {0, 0, 0}} ;
873             V = {v, {0, 0, 0}} ;
874             while(top[U.first] != top[V.first]){
875                 if(dep[top[U.first]] <
876                     dep[top[V.first]]) swap(U, V) ;
877                 U.second = query(dfn[top[U.first]],
878                     dfn[U.first]) + U.second ;
879                 U.first = fa[top[U.first]] ;
880             }
881             if(dep[U.first] < dep[V.first]) swap(U,
882                 V) ;
883             M = query(dfn[V.first], dfn[U.first]) ;
884             return (U.second.tot + V.second.tot +
885                 M.tot) - (U.second.left == M.right)
886                 - (V.second.left == M.left) ;
887         }
888
889         void init(){
890             memset(son, -1, sizeof(son)) ;
891         }
892
893         // Find range k-th largest number
894         struct Node{
895             int sum, left, right ;
896         }seg[Maxn + 20 * Maxn] ;
897
898         class PersistentSegmentTree{
899     private:
900         int n ;
901         int cnt ;
902         vector<int> version ;
903
904         int build(int L, int R){
905             int cur_cnt = cnt++ ;
906             if(L == R){
907                 seg[cur_cnt] = {0, 0, 0} ;
908                 return cur_cnt ;
909             }
910
911             int M = (L + R) >> 1 ;
912             int lc = build(L, M) ;
913             int rc = build(M+1, R) ;
914
915             seg[cur_cnt] = {0, lc, rc} ;
916             return cur_cnt ;
917         }
918
919         public:
920             PersistentSegmentTree(int _n){
921                 n = _n ;
922                 cnt = 0 ;
923
924                 int root = build(1, n) ;
925                 version.push_back(root) ;
926             }
927
928             void update(int ver, int idx){
929                 if(dep[top[u]] < dep[top[v]]) swap(u, 93                     v) ;
930                 modify(dfn[top[u]], dfn[u], val) ;
931                 u = fa[top[u]] ;
932             }
933
934             if(dep[u] < dep[v]) swap(u, v) ;
935             modify(dfn[v], dfn[u], val) ;
936         }
937
938         // get sum from u to v (simple path)
939         int get(int u, int v){
940             pair<int, ColorSeg> U, V ;
941             ColorSeg M ;
942             U = {u, {0, 0, 0}} ;
943             V = {v, {0, 0, 0}} ;
944             while(top[U.first] != top[V.first]){
945                 if(dep[top[U.first]] <
946                     dep[top[V.first]]) swap(U, V) ;
947                 U.second = query(dfn[top[U.first]],
948                     dfn[U.first]) + U.second ;
949                 U.first = fa[top[U.first]] ;
950             }
951             if(dep[U.first] < dep[V.first]) swap(U,
952                 V) ;
953             M = query(dfn[V.first], dfn[U.first]) ;
954             return (U.second.tot + V.second.tot +
955                 M.tot) - (U.second.left == M.right)
956                 - (V.second.left == M.left) ;
957         }
958
959         void init(){
960             memset(son, -1, sizeof(son)) ;
961         }
962
963         // Find range k-th largest number
964         struct Node{
965             int sum, left, right ;
966         }seg[Maxn + 20 * Maxn] ;
967
968         class PersistentSegmentTree{
969     private:
970         int n ;
971         int cnt ;
972         vector<int> version ;
973
974         int build(int L, int R){
975             int cur_cnt = cnt++ ;
976             if(L == R){
977                 seg[cur_cnt] = {0, 0, 0} ;
978                 return cur_cnt ;
979             }
980
981             int M = (L + R) >> 1 ;
982             int lc = build(L, M) ;
983             int rc = build(M+1, R) ;
984
985             seg[cur_cnt] = {0, lc, rc} ;
986             return cur_cnt ;
987         }
988
989         public:
990             PersistentSegmentTree(int _n){
991                 n = _n ;
992                 cnt = 0 ;
993
994                 int root = build(1, n) ;
995                 version.push_back(root) ;
996             }
997
998             void update(int ver, int idx){
999                 if(dep[top[u]] < dep[top[v]]) swap(u, 1000                    v) ;
1000                 modify(dfn[top[u]], dfn[u], val) ;
1001                 u = fa[top[u]] ;
1002             }
1003
1004             if(dep[u] < dep[v]) swap(u, v) ;
1005             modify(dfn[v], dfn[u], val) ;
1006         }
1007
1008         // get sum from u to v (simple path)
1009         int get(int u, int v){
1010             pair<int, ColorSeg> U, V ;
1011             ColorSeg M ;
1012             U = {u, {0, 0, 0}} ;
1013             V = {v, {0, 0, 0}} ;
1014             while(top[U.first] != top[V.first]){
1015                 if(dep[top[U.first]] <
1016                     dep[top[V.first]]) swap(U, V) ;
1017                 U.second = query(d
```

```

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

```

3.6 BIT ver1

```

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 ver2

```

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 ver3

```

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             else low[u] = min(low[u], dfn[v]) ;
22         }
23
24     if(u == fa && child > 1){
25         // this node u is a articulation point
26     }
27 }

```

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            } while(x != u) ;
29        }
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
47 void init(){
48     memset(dfn, -1, sizeof(dfn)) ;
49     memset(low, -1, sizeof(low)) ;
50 }
51
52 int main(){
53     init() ;
54     input() ;
55     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
56         == -1){
57         dfs(i, i) ;
58     }

```

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         ){
23         int v = e[i].v ;
24
25         if(dfn[v] == -1){
26             dfs1(v, i) ;
27             if(dfn[u] < low[v]) bz[i] = bz[i^1] =
28                 1 ;
29             low[u] = min(low[u], low[v]) ;
30         }
31         else if(i != (from ^ 1)) low[u] =
32             min(low[u], dfn[v]) ;
33     }
34 }
35
36 void dfs2(int u, int id){
37     vis_bcc[u] = id ;
38     bcc[id].push_back(u) ;
39
40     for ( int i=head[u] ; i!= -1 ; i=e[i].next
41         ){
42         int v = e[i].v ;
43
44         if(vis_bcc[v] != -1 || bz[i]) continue ;
45         dfs2(v, id) ;
46     }
47 }
48 void init(){

```

```

45     memset(dfn, -1, sizeof(dfn)) ;
46     memset(head, -1, sizeof(head)) ;
47     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
48 }
49
50 int main(){
51     init() ;
52     input() ;
53     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
54         == -1){
55         dfs1(i, 0) ;
56     }
57
58     for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i]
59         == -1){
60         bcc.push_back(vector<int>()) ;
61         dfs2(i, bcc_cnt++) ;
62     }

```

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

```

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     }
35
36     return 0 ;
37 }
38
39 int dfs(int u, int flow){
40     if(u == T) return flow ;
41     int d, rest = 0 ;
42
43     for ( int &i=cur[u] ; i!= -1 ;
44         i=e[i].next ){
45         int v = e[i].v ;
46         if(dep[v] == dep[u] + 1 && e[i].cap >
47             0){
48             d = dfs(v, min(flow - rest,
49                 e[i].cap)) ;
50
51             if(d > 0){
52                 e[i].cap -= d ;
53                 e[i^1].cap += d ;
54                 rest += d ;
55
56                 if(rest == flow) break ;
57             }
58         }
59     }
60
61     if(rest != flow) dep[u] = -1 ;
62     return rest ;
63 }
64
65 public:
66     MaxFlow(int n, int s, int t){
67         N = n ; S = s ; T = t ;
68         e.reserve(n*n) ;
69     }

```

```

63     head.assign(n+1, -1) ;
64     cur.resize(n+1) ;
65     dep.resize(n+1) ;
66 }
67
68 void AddEdge(int u, int v, int cap){
69     e.push_back({v, cap, head[u]}) ;
70     head[u] = e.size() - 1 ;
71     e.push_back({u, 0, head[v]}) ;
72     head[v] = e.size() - 1 ;
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
22                  cost){
23        e[+tot] = {v, cap, cost, head[u]} ;
24        head[u] = tot ;
25        e[+tot] = {u, 0, -cost, head[v]} ;
26        head[v] = tot ;
27    }
28
29    int run(){
30        vector<int> dis(N+1), pot(N+1, 0),
31                  preE(N+1) ;
32        int flow = 0, cost = 0 ;
33
34        auto dijkstra = [&](){
35            fill(dis.begin(), dis.end(), INF) ;
36            priority_queue<pii, vector<pii>,
37                        greater<pii>> pq ;
38            dis[s] = 0 ;
39            pq.push({0, s}) ;
40
41            while(!pq.empty()){
42                auto [d, u] = pq.top() ; pq.pop() ;
43                if(d > dis[u]) continue ;
44                for ( int i=head[u] ; i!=-1 ;
45                      i=e[i].next ){
46                    int v = e[i].v, cap = e[i].cap, w =
47                        e[i].cost ;
48                    if(cap && dis[v] > d + w + pot[u] -
49                        pot[v]){
50                        dis[v] = d + w + pot[u] - pot[v] ;
51                        preE[v] = i ;
52                        pq.push({dis[v], v}) ;
53                    }
54                }
55            }
56        }
57
58        int ans = 0 ;
59        while(bfs()){
60            ans += dfs(S, 0x3f3f3f3f) ;
61        }
62    }
63 }
```

```

49    }
50
51    return dis[t] != INF ;
52 }
53
54 while(dijkstra()){
55     for ( int v=1 ; v<=N ; v++ ) if(dis[v]
56         < INF){
57             pot[v] += dis[v] ;
58         }
59
60     int aug = INT_MAX ;
61     for ( int v=t ; v!=s ;
62           v=e[preE[v]^1].v ){
63         aug = min(aug, e[preE[v]].cap) ;
64     }
65
66     for ( int v=t ; v!=s ;
67           v=e[preE[v]^1].v ){
68         e[preE[v]].cap -= aug ;
69         e[preE[v]^1].cap += aug ;
70         cost += aug * e[preE[v]].cost ;
71     }
72 }
73
74 return cost ;
75 }
```

5 String

5.1 KMP

```

1 int Next[N] ;
2 void kmp(string &str){
3     Next[0] = -1 ;
4     if(str.size() <= 1) return ;
5     Next[1] = 0 ;
6
7     int cur = 2, check = 0 ;
8
9     while(cur < str.size()){
10        if(str[cur - 1] == str[check])
11            Next[cur++] = ++check ;
12        else if(check > 0) check =
13            Next[check] ;
14        else Next[cur++] = 0 ;
15    }
16
17    int main(){
18        ios::sync_with_stdio(false) ;
19        cin.tie(nullptr) ;
20        cout.tie(nullptr) ;
21
22        string s1, s2 ;
23        while(cin >> s1){
24            s2 = s1 ;
25            reverse(s2.begin(), s2.end()) ;
26            kmp(s2) ;
27
28            int x=0, y=0 ;
29            while(x < s1.size() && y < s2.size()){
30                if(s1[x] == s2[y]){
31                    x++ ;
32                    y++ ;
33                }
34                else if(y > 0) y = Next[y] ;
35                else x++ ;
36            }
37            cout << s1 << s2.substr(y) << endl ;
38        }
39
40        return 0 ;
41    }
42 }
```

5.2 ACAM

```

1 class ACACutomation{
2 private:
3     vector<int> fail, end, order ;
4     vector<vector<int>> tree ;
5
6     int base, alpha ;
7
8     int new_node(){
9         tree.emplace_back(alpha, 0) ;
10        fail.push_back(0) ;
11
12        return tree.size() - 1 ;
13    }
14 public:
15    ACACutomation(int _base='a', int _alpha=26)
16        : base(_base), alpha(_alpha) {
17            clear() ;
18        }
19
20    void clear(){
21        fail.assign(1, 0) ;
22        order.clear() ;
23        end.clear() ;
24        tree.assign(1, vector<int>(alpha, 0)) ;
25    }
26
27    void add_pattern(const string &pattern){
28        int u = 0 ;
29        for ( auto ch : pattern ){
30            int v = ch - base ;
31
32            if(tree[u][v] == 0) tree[u][v] =
33                new_node() ;
34            u = tree[u][v] ;
35
36        end.push_back(u) ;
37    }
38
39    void build(){
40        queue<int> q ;
41        order.clear() ;
42        order.push_back(0) ;
43
44        for ( int i=0 ; i<alpha ; i++ ){
45            if(tree[0][i] > 0){
46                q.push(tree[0][i]) ;
47            }
48
49        while(!q.empty()){
50            int u = q.front() ; q.pop() ;
51            order.push_back(u) ;
52
53            for ( int i=0 ; i<alpha ; i++ ){
54                if(tree[u][i] == 0) tree[u][i] =
55                    tree[fail[u]][i] ;
56                else{
57                    fail[tree[u][i]] = tree[fail[u]][i] ;
58                    q.push(tree[u][i]) ;
59                }
60            }
61
62        vector<int> count_per_pattern(const string
63                                     &text) const {
64            int u = 0 ;
65            vector<int> vis(tree.size(), 0) ;
66
67            for ( char ch : text ){
68                u = tree[u][ch - base] ;
69                vis[u]++;
70            }
71        }
72    }
73 }
```

```

71   for ( int i=order.size()-1 ; i>=1 ; i-- )
72     ){
73       int x = order[i] ;
74       vis[fail[x]] += vis[x] ;
75   }
76   vector<int> ans(end.size(), 0) ;
77   for ( int id=0 ; id<end.size() ; id++ ){
78     ans[id] = vis[end[id]] ;
79   }
80   return ans ;
81 }
82 }
```

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) ;
```

```

29   for ( int i=MaxLog ; i>=0 ; i-- )
30     if(depth[u] - (1 << i) >= depth[v]){
31       u = up[u][i] ;
32     }
33   if(u == v) return u ;
34
35   for ( int i=MaxLog ; i>=0 ; i-- )
36     if(up[u][i] != up[v][i]){
37       u = up[u][i] ;
38       v = up[v][i] ;
39     }
40
41   return up[u][0] ;
42 }
43
44 int main(){
45   int n, q, root ;
46   scanf("%d%d", &n, &q, &root) ;
47   MaxLog = __lg(n) ;
48
49   for ( int i=0 ; i<n-1 ; i++ ){
50     int u, v ;
51     scanf("%d%d", &u, &v) ;
52     e[u].push_back(v) ;
53     e[v].push_back(u) ;
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
12 int main(){
13   int cas, n;
14
15   scanf("%d", &cas);
16   while(cas--){
17     scanf("%d", &n);
18
19     long long s, v = 0;
20
21     for(int i = 0; i < n; i++){
22       scanf("%lld", &s);
23       v ^= SG(s); //XOR
24     }
25
26     if(v) printf("YES\n");
27     else printf("NO\n");
28   }
29 }
30
31 int SG[30] ;
32 int vis[Maxn], stone[Maxn] ;
33
34 void build(){
35   SG[0] = 0 ;
36   memset(vis, 0, sizeof(vis)) ;
```

```

37
38   for ( int i=1 ; i<30 ; i++ ){
39     int cur = 0 ;
40     for ( int j=0 ; j<i ; j++ ) for ( int
41       k=0 ; k<=j ; k++ ){
42       vis[SG[j] ^ SG[k]] = i ;
43     }
44     while(vis[cur] == i) cur++ ;
45     SG[i] = cur ;
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
16    int &d = dp[pos][sum][dsum][lim] ;
17    if(d != -1) return d ;
18
19    int up = lim ? dig[pos] : 9 ;
20    int res = 0 ;
21    for ( int i=0 ; i<=up ; i++ ){
22        res += solve(pos-1, (sum * 10 + i) %
23                      K, (dsum + i) % K, lim && i==up)
24            ;
25    }
26
27    return d = res ;
28 }
29
30 int count(int n){
31    memset(dp, -1, sizeof(dp)) ;
32    dig.clear() ;
33
34    while(n > 0){
35        dig.push_back(n % 10) ;
36        n /= 10 ;
37    }
38
39    return solve(dig.size() - 1, 0, 0, 1) ;
40 }
41
42 int main(){
43    int T ;
44    scanf("%d", &T) ;
45
46    int a, b ;
47    while(T--){
48        scanf("%d%d%d", &a, &b, &K) ;
49        if(K > 90) printf("0\n") ;
50        else printf("%d\n", count(b) -
51                    count(a-1)) ;
52    }
53
54    return 0 ;
55 }
```

7.3 樹 DP

```

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] ){
27         DFS(v) ;
28         cnt[u] += cnt[v] ;
29     }
30
31     dp[u][1][0] = dp[u][1][1] = 0 ;
32
33     for ( auto [v, w] : edge[u] ){
34         for ( int i=cnt[u] ; i>1 ; i-- ) for (
35             int j=1 ; j<i && j<=cnt[v] ; j++ ){
36             dp[u][i][1] = min(dp[u][i][1],
37                               dp[u][i-j][1] + dp[v][j][1] + 2 *
38                               w) ;
39             dp[u][i][0] = min(dp[u][i][0],
40                               dp[u][i-j][1] + dp[v][j][0] + w) ;
41             dp[u][i][0] = min(dp[u][i][0],
42                               dp[u][i-j][0] + dp[v][j][1] + 2 *
43                               w) ;
44         }
45     }
46 }
47
48 int main(){
49     int t = 0 ;
50
51     while(~scanf(" %d", &n) && n){
52         init() ;
53         for ( int i=0 ; i<n-1 ; i++ ){
54             int u, v, w ;
55             scanf("%d%d%d", &v, &u, &w) ;
56             edge[u].push_back({v, w}) ;
57         }
58
59         DFS(0) ;
60         printf("Case %d:\n", ++t) ;
61
62         int q, e ;
63         scanf("%d", &q) ;
64
65         while(q--){
66             scanf("%d", &e) ;
67
68             for ( int i=n ; i>=1 ; i-- )
69                 if(dp[0][i][0] <= e){
70                     printf("%d\n", i) ;
71                     break ;
72                 }
73         }
74     }
75
76     return 0 ;
77 }
```

```

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
```