# Contents



# 1 Foundations

## 1.1 PyMath

```python
import math

math.ceil(x) #上高斯
math.floor(x) #下高斯
math.factorial(x) #階乘
math.fabs(x) #絕對值
math.fsum(arr) #求和
math.gcd(x, y)
math.exp(x) # e^x
math.log(x, base)
math.log2(x)
math.log10(x)
math.sqrt(x)
math.pow(x, y, mod)
math.sin(x) # cos, tan, asin, acos, atan, atan2, sinh ...
math.hypot(x, y) #歐幾里德範數
math.degrees(x) #x從弧度轉角度
math.radians(x) #x從角度轉弧度
math.gamma(x) #x的gamma函數
math.pi #const
math.e #const
math.inf
```

## 1.2 Java Integer

```java
// 常量
MAX_VALUE, MIN_VALUE, BYTES, SIZE, TYPE

// 轉換/解析
static int   parseInt(String s)
static int   parseInt(String s, int radix)
static int   parseUnsignedInt(String s)
static int   parseUnsignedInt(String s, int radix)
static Integer valueOf(int i)
static Integer valueOf(String s)
static Integer valueOf(String s, int radix)
static String toString(int i)
static String toString(int i, int radix)
static String toUnsignedString(int i)
static String toUnsignedString(int i, int radix)
static long  toUnsignedLong(int x)
static Integer decode(String nm)                // 支援 0x/0/# 前綴
static Integer getInteger(String nm[, int val]) // 從系統屬性讀取整數

// 比較/雜湊/聚合
static int compare(int x, int y)
static int compareUnsigned(int x, int y)
static int hashCode(int value)
static int min(int a, int b)
static int max(int a, int b)
static int sum(int a, int b)

// 位元操作
static int bitCount(int i)   // 設定位數
static int highestOneBit(int i)
static int lowestOneBit(int i)
static int numberOfLeadingZeros(int i)
static int numberOfTrailingZeros(int i)
static int rotateLeft(int i, int distance)
static int rotateRight(int i, int distance)
static int reverse(int i)
static int reverseBytes(int i)

// 無號運算
static int divideUnsigned(int dividend, int divisor)
static int remainderUnsigned(int dividend, int divisor)
```

## 1.3 Java String

```
// 查詢
int     length()
boolean isEmpty()
boolean isBlank()     // (since 11)
char    charAt(int index)
int     codePointAt(int index)
int     codePointBefore(int index)
int     codePointCount(int beginIndex, int endIndex)
boolean contains(CharSequence s)
boolean startsWith(String prefix[, int toffset])
boolean endsWith(String suffix)
int     indexOf(String str[, int fromIndex])
int     lastIndexOf(String str[, int fromIndex])

// 取子字串/子序列
String      substring(int beginIndex)
String      substring(int beginIndex, int endIndex)
CharSequence subSequence(int beginIndex, int endIndex)

// 比較/等價
boolean equals(Object obj)
boolean equalsIgnoreCase(String anotherString)
int     compareTo(String anotherString)
int     compareToIgnoreCase(String str)
boolean matches(String regex)
boolean regionMatches(int toffset, String other, int ooffset, int len)
boolean regionMatches(boolean ignoreCase, int toffset, String other,
     int ooffset, int len)

// 建構/轉換/連接
String concat(String str)
String replace(char oldChar, char newChar)
String replace(CharSequence target, CharSequence replacement)
String replaceAll(String regex, String replacement)
String replaceFirst(String regex, String replacement)
String[] split(String regex[, int limit])
String toLowerCase()
String toUpperCase()
String trim()
String strip()           // (since 11)
String stripLeading()    // (since 11)
String stripTrailing()   // (since 11)
String repeat(int count) // (since 11)
IntStream chars()
Stream<String> lines()   // (since 11)
String intern()

// 靜態工具
static String format(String format, Object... args)
static String join(CharSequence delimiter, CharSequence... elements)
static String join(CharSequence delimiter, Iterable<? extends
     CharSequence> elements)
static String valueOf(primitive/char[]/Object)
static String copyValueOf(char[] data[, int offset, int count])
```

## 1.4 Java String builder

```
// 長度/容量
int  length()
int  capacity()
void ensureCapacity(int minimumCapacity)
void trimToSize()
void setLength(int newLength)

// 存取/修改
char         charAt(int index)
void         setCharAt(int index, char ch)
StringBuilder append(... 各種型別 ...)
StringBuilder insert(int offset, ... 各種型別 ...)
StringBuilder delete(int start, int end)
StringBuilder deleteCharAt(int index)
StringBuilder replace(int start, int end, String str)
StringBuilder reverse()

// 子字串/查找
String      substring(int start)
String      substring(int start, int end)
CharSequence subSequence(int start, int end)
int         indexOf(String str[, int fromIndex])
int         lastIndexOf(String str[, int fromIndex])

// 轉換
String toString()
```

## 1.5 Java Math

```
// 常量
static final double E, PI

// 絕對值/比較
static int/long/float/double abs(x)
static T max(a, b)
static T min(a, b)

// 取整/四捨五入
static double floor(double a)
static double ceil(double a)
static double rint(double a)              // 最接近整數(偶數優先)
static long  round(double a) / int round(float a)
static int   floorDiv(int x, int y)
static int   floorMod(int x, int y)

// 溢位保護 (exact 系列, Java 8+)
static int/long addExact(a, b)
static int/long subtractExact(a, b)
static int/long multiplyExact(a, b)
static int/long incrementExact(a)
static int/long decrementExact(a)
static int    toIntExact(long value)
static int/long negateExact(a)

// 指對數/冪根
static double pow(double a, double b)
static double sqrt(double a)
static double cbrt(double a)
static double exp(double a)
static double expm1(double x)
static double log(double a)
static double log10(double a)
static double log1p(double x)

// 三角/雙曲
static double sin/cos/tan(double a)
static double asin/acos/atan(double a)
static double atan2(double y, double x)
static double sinh/cosh/tanh(double a)

// 其他實用
static double hypot(double x, double y)
static double toDegrees(double angrad)
static double toRadians(double angdeg)
static double copySign(double magnitude, double sign)
static double nextUp/nextDown(double a)
static double nextAfter(double start, double direction)
static double ulp(double d)
static double random()
static double scalb(double d, int scaleFactor)
static double fma(double a, double b, double c) // (since 8)
static long  multiplyHigh(long x, long y)      // (since 9)
static long  multiplyFull(int x, int y)        // (since 9, 回傳 long)
```

# 2 Mathematics & Number Theory

## 2.1 Number Theory

**Fermat's Little Theorem:**

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

**Euler's Theorem:**

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

**Modular Inverse:**

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

**Euler Totient:**

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

**Fast Modular Exponentiation**

```cpp
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 $n$**

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

其中 $\mu$ 為莫比烏斯函數；常用於反演及計算互質對數量。

**Modulo Arithmetic Quick Facts**

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

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

**Extended Euclidean algorithm** 給定 $a$, $b$, $c$, 求 $ax+by=c$ 的解

```cpp
ll extgcd(ll a, ll b, ll &x, ll &y){
    if(b == 0){
        x = 1 ;
        y = 0 ;
        return a ;
    }
    ll d = extgcd(b, a%b, c, x, y), tmp = x ;
    x = y ;
    y = tmp - (a/b)*y ;
    return d ;
}

// x, y 為 ax + by = gcd(a, b) 的解
x = x * c / d ; // ax + by = c 的解
y = y * c / d ; // ax + by = c 的解
```

**Modular Inverse Basics** 對於 $\gcd(a,m)=1$，存在唯一 $a^{-1} \in [0,m)$ 使得 $a \cdot a^{-1} \equiv 1 \pmod{m}$。

- 用途：實作模除法、解線性同餘、組合數除法、CRT 等。
- 求法：擴展歐幾里得適用任意模數；若 $m$ 為質數可用 $a^{m-2} \bmod m$。
- 存在條件：僅當 $\gcd(a,m)=1$；不互質時可除以 $d=\gcd(a,m)$ 後再檢查。

```cpp
g = gcd(a, m, c, x, y) ;
if(g == 1){
    inv = (x % m + m) % m ;
}
```

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

- $a \equiv b \pmod{m} \iff m \mid (a-b)$，同餘類以差整除判斷。
- $a \equiv b \pmod{m} \Rightarrow f(a) \equiv f(b) \pmod{m}$ 對所有以整數係數的多項式 $f$ 成立。
- 若 $a \equiv b \pmod{m}$ 且 $c \equiv d \pmod{m}$，則 $a \pm c \equiv b \pm d \pmod{m}$，$ac \equiv bd \pmod{m}$，$a^n \equiv b^n \pmod{m}$。
- 若 $\gcd(k,m)=1$ 且 $ka \equiv kb \pmod{m}$，可約去 $k$：$a \equiv b \pmod{m}$。
- 若 $a \equiv b \pmod{m}$，則 $a \equiv b \pmod{d}$ 對任何 $d$ 整除 $m$ 亦成立。
- 若 $d \mid a, b, m$，則 $\frac{a}{d} \equiv \frac{b}{d} \pmod{\frac{m}{d}}$

**Bézout's Identity**

$$\text{若} a,b \in \mathbb{Z}, \quad \exists x,y \in \mathbb{Z} \text{ 使得} ax+by=\gcd(a,b).$$

任何方程 $ax+by=c$ 有整數解當且僅當 $\gcd(a,b) \mid c$。擴展歐幾里得演算法可同時求得 gcd 與一組 $(x,y)$，常用於計算模逆與結合 China Remainder。
根據貝祖定理，只有當 $d \mid c$ 時，$ax+by=c$ 才有整數解
對於 $ax+by=c$ 假設有整數解，且 $gcd(a,b)=1$ 使用 exgcd 求出 $(a_0,b_0)$ 的特解後，根據逆模元可得出通解：$a=a_0+\frac{b}{d}t,\ b=b_0-\frac{a}{d}t$ 其中 t 可以為任意整數

**Euler 定理與指數化簡**

$$\gcd(a,m)=1 \ \Rightarrow \ a^{\varphi(m)} \equiv 1 \pmod{m}.$$

設 $e \equiv b \pmod{\varphi(m)}$。若 $\gcd(a,m)=1$ 且 $b$ 很大，常用以下化簡：

$$a^b \equiv a^e \pmod{m} \quad (\text{若} e=0 \text{，可取} e := \varphi(m)).$$

**Carmichael 函數與更強化簡**

對所有互質$a,m$： $a^{\lambda(m)} \equiv 1 \pmod{m}$, $x \equiv y \pmod{\lambda(m)} \Rightarrow a^x \equiv a^y \pmod{m}$.

分解 $m=\prod p_i^{k_i}$ 後

$$\lambda(m) = \operatorname{lcm}\big(\lambda(p_1^{k_1}), \ldots, \lambda(p_r^{k_r})\big), \quad \lambda(p^k) = \begin{cases} 2^{k-2}, & p=2,\ k \geq 3, \\ \varphi(p^k), & \text{其餘情形.} \end{cases}$$

實務上，若 $\gcd(a,m)=1$，可把指數先對 $\lambda(m)$（或 $\varphi(m)$）取模，再用上式處理 $e=0$ 的情形。

**乘法階（Multiplicative Order）**

$$\operatorname{ord}_m(a) = \min\{t>0 : a^t \equiv 1 \pmod{m}\}, \quad \operatorname{ord}_m(a) \mid \lambda(m) \mid \varphi(m).$$

常用判別：$a^x \equiv a^y \pmod{m}$ 當且僅當 $x \equiv y \pmod{\operatorname{ord}_m(a)}$（假設 $\gcd(a,m)=1$）。

**模指數運算快速規則**

$$\begin{aligned}
(a^x \bmod m)(a^y \bmod m) &\equiv a^{x+y} \pmod{m}, \\
(a^x)^y &\equiv a^{xy} \pmod{m}, \\
\gcd(a,m)=1,\ x \equiv y \pmod{\varphi(m)} &\Rightarrow a^x \equiv a^y \pmod{m}, \\
p \text{ 為質數且 } \gcd(a,p)=1 &: a^{k(p-1)+r} \equiv a^r \pmod{p}.
\end{aligned}$$

若 $m=\prod p_i^{k_i}$，計算 $a^b \bmod m$ 可先各自求 $a^b \bmod p_i^{k_i}$（遇到不互質時可配合質因數分解與指數提取），再用中國剩餘定理合併。

### Chinese Remainder Theorem

設同餘系統

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

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）

```cpp
// solve x  a1 (mod m1), x  a2 (mod m2)
// return {x0, lcm}; if no solution, lcm = -1
pair<ll, ll> crt(ll a1, ll m1,
                 ll a2, ll m2) {
    ll g = std::gcd(m1, m2);
    if ((a2 - a1) % g != 0) return {0, -1}; // no solution

    ll lcm = m1 / g * m2;
    ll m1_reduced = m1 / g;
    ll m2_reduced = m2 / g;

    ll diff = (a2 - a1) / g % m2_reduced;
    if (diff < 0) diff += m2_reduced;

    ll inv = mod_pow(m1_reduced, m2_reduced - 1, m2_reduced);
    ll step = diff * inv % m2_reduced;
    ll x0 = (a1 + step * m1) % lcm;
    if (x0 < 0) x0 += lcm;
    return {x0, lcm};
}
```

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

```
ll extgcd(ll a, ll b, ll c, ll &x, ll &y){
    if(b == 0){
        x = c/a ;
        y = 0 ;
        return a ;
    }
    ll d = extgcd(b, a%b, c, x, y), tmp = x ;
    x = y ;
    y = tmp - (a/b)*y ;
    return d ;
}
```

## 2.2 Combinatorics

### Binomial Coefficient Identities

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$
$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$$
$$\sum_{k=0}^{n} \binom{n}{k} = 2^n, \qquad \sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}.$$

### Stars and Bars 非負整數解數量：

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

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 $A_1, \ldots, A_k$：

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

計算滿足限制的排列或整數解時廣泛使用。

### Catalan Numbers 基本定義：

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

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

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

```
struct Edge{
  int u, v, w ;
    // 這是最大生成樹，最小生成樹要改成 w < o.w
  bool operator>(const Edge &o) const {return w > o.w ;} ;
} ;

int par[N] ;
int sz[N] ;
int sum ;

vector<Edge> edge ;

void init(){
  edge.clear() ;
  for ( int i=0 ; i<N ; i++ ){
    par[i] = i ;
    sz[i] = 1 ;
  }
  sum = 0 ;
}

int find(int x){
  if(x == par[x]) return x ;
  return par[x] = find(par[x]) ;
}

int merge(int x, int y){
  x = find(x) ;
  y = find(y) ;

  if(x == y) return 0 ;
  if(sz[x] > sz[y]) swap(x, y) ;
  par[x] = y ;
  sz[y] += sz[x] ;

  return 1 ;
}

void MST(){
  int cnt = 0 ;
  for ( int i=0 ; i<edge.size() && cnt < n-1 ; i++ ){
    auto [u, v, w] = edge[i] ;
    if(merge(u, v)){
      cnt++ ;
      sum -= w ;
    }
  }
}

int main(){
    for ( int i=0 ; i<m ; i++ ){
        scanf("%d%d%d", &u, &v, &w) ;
        edge.push_back({u, v, w}) ;
        sum += w ;
    }

    sort(edge.begin(), edge.end(), greater<Edge>()) ;
    MST() ;
}
```

## 3.2 SegmentTree

```
#define lc (id << 1)
#define rc ((id << 1) | 1)

struct LazyTag{
    // type 0 : increase val
    // type 1 : set to val
    // type 1 can overwrite type 0
    int type ;
    ll val ;
} ;

struct Node{
    LazyTag tag ;
    ll sum ;
    int sz ;
```

```
}seg[Maxn << 2] ;

class SegmentTree{
private:
    void pull(int id){
        seg[id].sum = seg[lc].sum + seg[rc].sum ;
    }

    void AddTag(int id, LazyTag &tag){
        if(tag.type == 0){
            seg[id].sum += tag.val * seg[id].sz ;
            seg[id].tag.val += tag.val ;
        }
        else{
            seg[id].sum = tag.val * seg[id].sz ;
            seg[id].tag = {1, tag.val} ;
        }
    }

    void push(int id){
        AddTag(lc, seg[id].tag) ;
        AddTag(rc, seg[id].tag) ;
        seg[id].tag = {0, 0} ;
    }

public:
    void build(int L=1, int R=n, int id=1){
        seg[id].sum = 0 ;
        seg[id].tag = {0, 0} ;
        seg[id].sz = 1 ;

        if(L == R){
            seg[id].sum = arr[L] ;
            return ;
        }

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

        pull(id) ;
        seg[id].sz = seg[lc].sz + seg[rc].sz ;
    }

    void modify(int l, int r, LazyTag &tag, int L=1, int R=n, int
         id=1){
        if(l <= L && R <= r){
            AddTag(id, tag) ;
            return ;
        }

        push(id) ;
        int M = (L + R) >> 1 ;
        if(r <= M) modify(l, r, tag, L, M, lc) ;
        else if(l > M) modify(l, r, tag, M+1, R, rc) ;
        else{
            modify(l, r, tag, L, M, lc) ;
            modify(l, r, tag, M+1, R, rc) ;
        }
        pull(id) ;
    }

    ll query(int l, int r, int L=1, int R=n, int id=1){
        if(l <= L && R <= r) return seg[id].sum ;

        push(id) ;
        int M = (L + R) >> 1 ;
        if(r <= M) return query(l, r, L, M, lc) ;
        else if(l > M) return query(l, r, M+1, R, rc) ;
        else return query(l, r, L, M, lc) + query(l, r, M+1, R, rc) ;
    }
}tree ;
```

## 3.3 HLD

```
/* HLD */
int fa[Maxn], top[Maxn], son[Maxn], sz[Maxn], dep[Maxn] = {0},
     dfn[Maxn], rnk[Maxn], dfscnt = 0 ;

void dfs1(int u, int from){
  fa[u] = from ;
  dep[u] = dep[from] + 1 ;
  sz[u] = 1 ;

  for ( auto v : g[u] ) if(v != from){
    dfs1(v, u) ;
    sz[u] += sz[v] ;
    if(son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v ;
  }
}

void dfs2(int u, int t){
  top[u] = t ;
  dfn[u] = ++dfscnt ;
  rnk[dfscnt] = u ;

  if(son[u] == -1) return ;

  dfs2(son[u], t) ;

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

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

struct ColorSeg{
  int left, right, tot ;

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

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

    return tmp ;
  }
} ;

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

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

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

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

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

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

```
  }

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

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

    // get sum from u to v (simple path)
  int get(int u, int v){
    pair<int, ColorSeg> U, V ;
    ColorSeg M ;
    U = {u, {0, 0, 0}} ;
    V = {v, {0, 0, 0}} ;

    while(top[U.first] != top[V.first]){
      if(dep[top[U.first]] < dep[top[V.first]]) swap(U, V) ;
      U.second = query(dfn[top[U.first]], dfn[U.first]) + U.second ;
      U.first = fa[top[U.first]] ;
    }

    if(dep[U.first] < dep[V.first]) swap(U, V) ;

    M = query(dfn[V.first], dfn[U.first]) ;

    return (U.second.tot + V.second.tot + M.tot) - (U.second.left ==
         M.right) - (V.second.left == M.left) ;
  }
}tree ;

void init(){
  memset(son, -1, sizeof(son)) ;
}
```

## 3.4 PST

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

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

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

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

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

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

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

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

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

      if(idx <= M) lc = self(self, seg[cur].left, L, M) ;
      else rc = self(self, seg[cur].right, M+1, R) ;

      seg[cur_cnt] = {seg[lc].sum + seg[rc].sum, lc, rc} ;

      return cur_cnt ;
    };

    int root = upd(upd, version[ver], 1, n) ;
    version.push_back(root) ;
  }

  int query(int verL, int verR, int k){
    auto qry = [&](auto &&self, const int cur_old, const int cur_new,
         int L, int R){
      if(L == R) return L ;

      int old_l = seg[cur_old].left, old_r = seg[cur_old].right ;
      int new_l = seg[cur_new].left, new_r = seg[cur_new].right ;

      int dl = seg[new_l].sum - seg[old_l].sum ;
      int dr = seg[new_r].sum - seg[old_r].sum ;

      int M = (L + R) >> 1 ;

      if(dl >= k) return self(self, old_l, new_l, L, M) ;
      k -= dl ;
      return self(self, old_r, new_r, M+1, R) ;
    };

    int idx = qry(qry, version[verL-1], version[verR], 1, n) ;
    return idx ;
  }
};
```

## 3.5 Trie

```
class TrieNode{
public:
    set<int> end ;
    TrieNode *next[26] ;

    TrieNode(){
        for ( int i=0 ; i<26 ; i++ ) next[i] = nullptr ;
    }
};

class Trie{
private:
    int cnt ;
    TrieNode *root ;
public:
    Trie() : cnt(0) {
        root = new TrieNode() ;
    }

    void insert(string &str, int n){
        TrieNode* node = root ;
        for ( auto s : str ){
            int path = s - 'a' ;

            if(node->next[path] == nullptr) node->next[path] = new
                 TrieNode() ;
            node = node->next[path] ;
        }
```

```
        node->end.insert(n) ;
    }

    void search(string &str){
        TrieNode* node = root ;
        for ( auto s : str ){
            int path = s - 'a' ;
            if(node->next[path] == nullptr) return ;
            node = node->next[path] ;
        }

        int flg = 0 ;
        for ( auto n : node->end ){
            if(flg) cout << " " ;
            else flg = 1 ;

            cout << n ;
        }
    }

    void clear(TrieNode* node) {
        if (!node) return ;
        for (int i = 0; i < 26; i++) {
            if (node->next[i]) {
                clear(node->next[i]) ;
            }
        }
        delete node ;
    }

    ~Trie(){
        clear(root) ;
    }
};
```

## 3.6 BIT 單修區查

```
// 單點修改 區間查詢
#define lowbit(x) (x & -x)

int BIT[MAX_SIZE] ;
int n ;

void modify(int idx, int val){
    for ( ; idx <= n ; idx += lowbit(idx) ){
        BIT[idx] += val ;
    }
}

// ans: query(R) - query(L-1)
int query(int idx){
    int sum = 0 ;
    for ( ; idx ; idx -= lowbit(idx) ){
        sum += BIT[idx] ;
    }
}

void init(){
    memset(BIT, 0, sizeof(BIT)) ;
}
```

## 3.7 BIT 區修單查

```
// 區間修改, 單點查詢
#define lowbit(x) (x & -x)

int BIT[MAX_SIZE] ;
int n ;

void modify(int idx, int val){
    for ( ; idx <= n ; idx += lowbit(idx) ){
        BIT[idx] += val ;
    }
}

// ans: query(i)
int query(int idx){
    int sum = 0 ;
    for ( ; idx ; idx -= lowbit(idx) ){
        sum += BIT[idx] ;
    }

    return sum ;
}

void init(){
    memset(BIT, 0, sizeof(BIT)) ;
}

void build(){
    arr[0] = 0 ;
    for ( int i=1 ; i<=n ; i++ ) modify(i, arr[i] - arr[i-1]) ;
}

// usage
// add val
modify(L, x) ;
modify(R+1, -x) ;
```

## 3.8 BIT 區修區查

```
// 區間修改 區間查詢
#define lowbit(x) x & -x

ll BIT1[MAXN], BIT2[MAXN] ;

void update(int idx, ll val){
    for ( int i=idx ; i<=n ; i+=lowbit(i) ){
        BIT1[i] += val ;
        BIT2[i] += idx * val ;
    }
}

// range query: query(R) - query(X - 1)
ll query(int idx){
    ll sum1 = 0, sum 2 = 0 ;
    for ( int i=idx ; i ; i-=lowbit(i) ){
        sum1 += BIT1[i] ;
        sum2 += BIT2[i] ;
    }

    return (idx + 1) * sum1 - sum2 ;
}

void build(){
    for (int i=1 ; i<=n ; i++ ){
        update(i, arr[i] - arr[i-1]) ;
    }
}

void usage(){
    update(L, x) ;
    update(R+1, -x) ;
}

void init(){
    memset(BIT1, 0, sizeof(BIT1)) ;
    memset(BIT2, 0, sizeof(BIT2)) ;
}
```

# 4 Graph

## 4.1 cut vertex AND bridges

```
int dfn[Maxn] = {-1}, low[Maxn] = {-1}, dfscnt ;

void dfs(int u, int fa){
  dfn[u] = low[u] = ++dfscnt ;
```

```
  int child = 0 ;

  for ( auto v : g[u] ) if(v != fa){
    if(dfn[v] == -1){
      child++ ;
      dfs(v, u) ;
      low[u] = min(low[u], low[v]) ;

      if(low[v] >= dfn[u]){
        // this edge is a bridge
      }

      if(u != fa && low[v] >= dfn[u]){
        // this node v is a articulation point
      }
    }
    else low[u] = min(low[u], dfn[v]) ;
  }

  if(u == fa && child > 1){
    // this node u is a articulation point
  }
}
```

## 4.2 SCC - Tarjan

```
vector<int> scc[Maxn] ;
int dfn[Maxn], low[Maxn], sccId[Maxn], dfscnt = 0, cnt_scc = 0 ;
stack<int> st ;
bitset<Maxn> inSt, vis ;

void dfs(int u, int from){
  dfn[u] = low[u] = ++dfscnt ;
  st.push(u) ;
  inSt[u] = 1 ;

  for ( auto v : g[u] ){
    if(!inSt[v] && dfn[v] != -1) continue ;
    if(dfn[v] == -1) dfs(v, u) ;
    low[u] = min(low[u], low[v]) ;
  }

  if(dfn[u] == low[u]){
    cnt_scc++ ;
    int x ;

    do{
      x = st.top() ;
      st.pop() ;

      inSt[x] = 0 ;
      sccId[x] = cnt_scc ;
      scc[cnt_scc].push_back(x) ;
    }
    while(x != u) ;
  }
}

// SCC to DAG (after dfs)
vector<int> dag[Maxn] ;

void scc_to_dag(){
  vector<int> dag[Maxn] ;
  for ( int u=1 ; u<=n ; u++ ){
    for ( auto v : g[u] ){
      if(sccId[u] != sccId[v]){
        dag[sccId[u]].push_back(sccId[v]) ;
      }
    }
  }
}

void init(){
  memset(dfn, -1, sizeof(dfn)) ;
  memset(low, -1, sizeof(low)) ;
}

int main(){
    init() ;
    input() ;
    for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
    dfs(i, i) ;
  }
}
```

## 4.3 BCC - Tarjan

```
struct Edge{
  int v, next ;
}e[Maxm << 1] ;
int head[Maxm], tot = 1 ;

void add(int u, int v){
  e[++tot] = {v, head[u]} ;
  head[u] = tot ;
  e[++tot] = {u, head[v]} ;
  head[v] = tot ;
}

bitset<Maxm << 1> bz ;
vector<vector<int> > bcc ;
int dfn_cnt = 0, dfn[Maxn], low[Maxn], vis_bcc[Maxn], bcc_cnt = 0 ;

void dfs1(int u, int from){
  dfn[u] = low[u] = ++dfn_cnt ;

  for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
    int v = e[i].v ;

    if(dfn[v] == -1){
      dfs1(v, i) ;
      if(dfn[u] < low[v]) bz[i] = bz[i^1] = 1 ;
      low[u] = min(low[u], low[v]) ;
    }
    else if(i != (from ^ 1)) low[u] = min(low[u], dfn[v]) ;
  }
}

void dfs2(int u, int id){
  vis_bcc[u] = id ;
  bcc[id].push_back(u) ;

  for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
    int v = e[i].v ;

    if(vis_bcc[v] != -1 || bz[i]) continue ;
    dfs2(v, id) ;
  }
}

void init(){
  memset(dfn, -1, sizeof(dfn)) ;
  memset(head, -1, sizeof(head)) ;
  memset(vis_bcc, -1, sizeof(vis_bcc)) ;
}

int main(){
    init() ;
    input() ;
    for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
    dfs1(i, 0) ;
  }

  for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i] == -1){
    bcc.push_back(vector<int>()) ;
    dfs2(i, bcc_cnt++) ;
  }
}
```

## 4.4 Convex

```cpp
struct Coordinate{
  long long x, y ;

  friend bool operator<(const Coordinate&a, const Coordinate& b){
    if(a.x == b.x) return a.y < b.y ;
    return a.x < b.x ;
  }

  friend bool operator==(const Coordinate& a, const Coordinate& b){
    return a.x == b.x && a.y == b.y ;
  }
} ;

vector<Coordinate> nodes ;

long long cross(const Coordinate& o, const Coordinate& a, const
     Coordinate& b){
  return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x) ;
}

void input(){
  nodes.clear() ;

  int n, x, y ;
  char c ;
  cin >> n ;

  for ( int i=0 ; i<n ; i++ ){
    cin >> x >> y >> c ;
    if(c == 'Y') nodes.push_back({x, y}) ;
  }
}

void monotone(){
  sort(nodes.begin(), nodes.end()) ;

  int n = unique(nodes.begin(), nodes.end()) - nodes.begin() ;

  vector<Coordinate> ch(n+1) ;

  int m = 0 ;

  for ( int i=0 ; i<n ; i++ ){
    while(m > 1 && cross(ch[m-2], ch[m-1], nodes[i]) < 0) m-- ;
    ch[m++] = nodes[i] ;
  }
  for ( int i=n-2, t=m ; i>=0 ; i-- ){
    while(m > t && cross(ch[m-2], ch[m-1], nodes[i]) < 0) m-- ;
    ch[m++] = nodes[i] ;
  }

  if(n > 1) m-- ;
  cout << m << endl ;

  for ( int i=0 ; i<m ; i++ ) cout << ch[i].x << " " << ch[i].y <<
       endl ;
}
```

## 4.5 Max Flow

```cpp
struct Edge{
  int v, cap, next ;
};

class MaxFlow{
private:
  int N, S, T ;
  vector<Edge> e ;
  vector<int> head, cur, dep ;

  bool bfs(){
    queue<int> q ;
    for ( int i=0 ; i<=N ; i++ ){
      cur[i] = head[i] ;
      dep[i] = -1 ;
    }
    q.push(S) ;
    dep[S] = 0 ;

    while(!q.empty()){
      int u = q.front() ; q.pop() ;

      for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
        int v = e[i].v ;
        if(dep[v] == -1 && e[i].cap > 0){
          dep[v] = dep[u] + 1 ;
          if(v == T) return 1 ;
          q.push(v) ;
        }
      }
    }

    return 0 ;
  }

  int dfs(int u, int flow){
    if(u == T) return flow ;
    int d, rest = 0 ;

    for ( int &i=cur[u] ; i!=-1 ; i=e[i].next ){
      int v = e[i].v ;
      if(dep[v] == dep[u] + 1 && e[i].cap > 0){
        d = dfs(v, min(flow - rest, e[i].cap)) ;

        if(d > 0){
          e[i].cap -= d ;
          e[i^1].cap += d ;
          rest += d ;

          if(rest == flow) break ;
        }
      }
    }

    if(rest != flow) dep[u] = -1 ;
    return rest ;
  }
public:
  MaxFlow(int n, int s, int t){
    N = n ; S = s ; T = t ;
    e.reserve(n*n) ;
    head.assign(n+1, -1) ;
    cur.resize(n+1) ;
    dep.resize(n+1) ;
  }

  void AddEdge(int u, int v, int cap){
    e.push_back({v, cap, head[u]}) ;
    head[u] = e.size() - 1 ;
    e.push_back({u, 0, head[v]}) ;
    head[v] = e.size() - 1 ;
  }

  int run(){
    int ans = 0 ;
    while(bfs()){
      ans += dfs(S, 0x3f3f3f3f) ;
    }
    return ans ;
  }
};
```

## 4.6 KM（二分圖最大權匹配）

```cpp
// KM (Kuhn-Munkres) Algorithm — Maximum Weight Bipartite Matching
// Usage:
//   - Build: KM km(nLeft, nRight);          // 1-indexed
//   - Add edges: km.addEdge(u, v, w);       // 1 <= u <= nLeft, 1 <=
     v <= nRight
//   - Solve: long long ans = km.solve();    // maximum total weight
//   - Retrieve matching (right -> left): auto y2x =
     km.match_right(); // size nRight+1
// Notes:
```

```cpp
//   - Complexity: O(N^3), N = max(nLeft, nRight).
//   - If nLeft != nRight, internally pads the smaller side with 0 weights.
//   - If an edge (u, v) is not allowed, you may set a very negative weight
//     (e.g., -INF) to forbid it and effectively achieve "max weight among valid edges".
//   - For minimum weight matching, invert weights (e.g., add a constant or negate).

struct KM {
    int nL, nR, N; // N = max(nL, nR)
    vector<vector<long long>> w; // 1..N x 1..N (padded with 0)
    vector<long long> lx, ly, slack; // labels and slack for Y
    vector<int> slackx, prevy;    // slack argmin and previous X for Y in augmenting tree
    vector<int> xy, yx;           // matching: xy[x]=y, yx[y]=x
    static constexpr long long INF = (long long)4e18;

    KM(int _nL, int _nR){
        nL = _nL; nR = _nR; N = max(nL, nR);
        w.assign(N+1, vector<long long>(N+1, 0));
        lx.assign(N+1, 0);
        ly.assign(N+1, 0);
        slack.assign(N+1, 0);
        slackx.assign(N+1, 0);
        prevy.assign(N+1, 0);
        xy.assign(N+1, 0);
        yx.assign(N+1, 0);
    }

    void addEdge(int u, int v, long long weight){
        if(u<=0 || u>nL || v<=0 || v>nR) return;
        w[u][v] = max(w[u][v], weight); // keep the best weight if duplicated
    }

    // Core augment step for left node 'root'
    void augment(int root){
        vector<char> S(N+1, false), T(N+1, false);
        queue<int> q;
        q.push(root); S[root] = true;
        for(int y=1; y<=N; ++y){
            slack[y] = lx[root] + ly[y] - w[root][y];
            slackx[y] = root;
            prevy[y] = 0;
        }

        int y_free = 0; // free Y found to augment
        while(true){
            while(!q.empty()){
                int x = q.front(); q.pop();
                for(int y=1; y<=N; ++y){
                    if(T[y]) continue;
                    long long d = lx[x] + ly[y] - w[x][y];
                    if(d == 0){
                        // tight edge, add to tree
                        T[y] = true; prevy[y] = x;
                        if(yx[y] == 0){ y_free = y; goto AUG; }
                        // else grow the tree
                        int x2 = yx[y];
                        if(!S[x2]){ S[x2] = true; q.push(x2);
                            for(int yy=1; yy<=N; ++yy){
                                long long dd = lx[x2] + ly[yy] - w[x2][yy];
                                if(dd < slack[yy]){ slack[yy] = dd; slackx[yy] = x2; }
                            }
                        }
                    } else if(d < slack[y]){
                        slack[y] = d; slackx[y] = x;
                    }
                }
            }
            // relabel
            long long delta = INF;
            for(int y=1; y<=N; ++y) if(!T[y]) delta = min(delta, slack[y]);
            for(int x=1; x<=N; ++x) if(S[x]) lx[x] -= delta;
            for(int y=1; y<=N; ++y){
                if(T[y]) ly[y] += delta;
                else slack[y] -= delta;
            }
            for(int y=1; y<=N; ++y){
                if(!T[y] && slack[y] == 0){
                    T[y] = true; prevy[y] = slackx[y];
                    if(yx[y] == 0){ y_free = y; goto AUG; }
                    int x2 = yx[y];
                    if(!S[x2]){ S[x2] = true; q.push(x2);
                        for(int yy=1; yy<=N; ++yy){
                            long long dd = lx[x2] + ly[yy] - w[x2][yy];
                            if(dd < slack[yy]){ slack[yy] = dd; slackx[yy] = x2; }
                        }
                    }
                }
            }
        }
    AUG:
        // Augment along alternating path ending at y_free
        while(y_free){
            int x = prevy[y_free];
            int next_y = xy[x];
            yx[y_free] = x; xy[x] = y_free;
            y_free = next_y;
        }
    }

    long long solve(){
        // init labels
        for(int x=1; x<=N; ++x){
            lx[x] = w[x][1];
            for(int y=2; y<=N; ++y) lx[x] = max(lx[x], w[x][y]);
        }
        fill(ly.begin(), ly.end(), 0);
        fill(xy.begin(), xy.end(), 0);
        fill(yx.begin(), yx.end(), 0);

        for(int x=1; x<=N; ++x) augment(x);

        long long ans = 0;
        // Sum only over real nodes to avoid counting padded matches
        for(int x=1; x<=nL; ++x){
            int y = xy[x];
            if(1 <= y && y <= nR) ans += w[x][y];
        }
        return ans;
    }

    // Right side matching (1..nR), value is the matched left index (or 0 if unmatched)
    vector<int> match_right() const{
        vector<int> ret(nR+1, 0);
        for(int y=1; y<=nR; ++y){
            if(y<=N) ret[y] = (y<= (int)yx.size()-1 ? yx[y] : 0);
        }
        return ret;
    }

    // Left side matching (1..nL), value is the matched right index (or 0 if unmatched)
    vector<int> match_left() const{
        vector<int> ret(nL+1, 0);
        for(int x=1; x<=nL; ++x){
            if(x<=N) ret[x] = (x<= (int)xy.size()-1 ? xy[x] : 0);
        }
        return ret;
    }
};

/*
Example:
    int n = 3, m = 3;
    KM km(n, m);
    km.addEdge(1, 1, 5);
    km.addEdge(1, 2, 6);
```

```
    km.addEdge(2, 2, 4);
    km.addEdge(2, 3, 7);
    km.addEdge(3, 1, 3);
    long long ans = km.solve(); // maximum weight
    auto y2x = km.match_right(); // y2x[v] = matched u
*/
```

### 4.7 min cut max flow

```
struct Edge{
  int v, cap, cost , next ;
};

using pii = pair<int, int> ;
class MCMF{
private:
  int N, s, t, tot ;
  vector<Edge> e ;
  vector<int> head ;
public:
  MCMF(int n, int _s, int _t){
    N = n ;
    s = _s ;
    t = _t ;
    e.resize(n*n + 5) ;
    head.assign(n+5, -1) ;
    tot = -1 ;
  }

  void AddEdge(int u, int v, int cap, int cost){
    e[++tot] = {v, cap, cost, head[u]} ;
    head[u] = tot ;
    e[++tot] = {u, 0, -cost, head[v]} ;
    head[v] = tot ;
  }

  int run(){
    vector<int> dis(N+1), pot(N+1, 0), preE(N+1) ;
    int flow = 0, cost = 0 ;

    auto dijkestra = [&](){
      fill(dis.begin(), dis.end(), INF) ;
      priority_queue<pii, vector<pii>, greater<pii>> pq ;
      dis[s] = 0 ;
      pq.push({0, s}) ;

      while(!pq.empty()){
        auto [d, u] = pq.top() ; pq.pop() ;
        if(d > dis[u]) continue ;
        for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
          int v = e[i].v, cap = e[i].cap, w = e[i].cost ;
          if(cap && dis[v] > d + w + pot[u] - pot[v]){
            dis[v] = d + w + pot[u] - pot[v] ;
            preE[v] = i ;
            pq.push({dis[v], v}) ;
          }
        }
      }

      return dis[t] != INF ;
    };

    while(dijkestra()){
      for ( int v=1 ; v<=N ; v++ ) if(dis[v] < INF){
        pot[v] += dis[v] ;
      }

      int aug = INT_MAX ;
      for ( int v=t ; v!=s ; v=e[preE[v]^1].v ){
        aug = min(aug, e[preE[v]].cap) ;
      }

      for ( int v=t ; v!=s ; v=e[preE[v]^1].v ){
        e[preE[v]].cap -= aug ;
        e[preE[v]^1].cap += aug ;
        cost += aug * e[preE[v]].cost ;
      }
    }

    return cost ;
  }
};
```

## 5 String

### 5.1 KMP

```
int Next[N] ;
void kmp(string &str){
    Next[0] = -1 ;
    if(str.size() <= 1) return ;
    Next[1] = 0 ;

    int cur = 2, check = 0 ;

    while(cur < str.size()){
        if(str[cur - 1 ] == str[check]) Next[cur++] = ++check ;
        else if(check > 0) check = Next[check] ;
        else Next[cur++] = 0 ;
    }
}

int main(){
    ios::sync_with_stdio(false) ;
    cin.tie(nullptr) ;
    cout.tie(nullptr) ;

    string s1, s2 ;
    while(cin >> s1){
        s2 = s1 ;
        reverse(s2.begin(), s2.end()) ;
        kmp(s2) ;

        int x=0, y=0 ;
        while(x < s1.size() && y < s2.size()){
            if(s1[x] == s2[y]){
                x++ ;
                y++ ;
            }
            else if(y > 0) y = Next[y] ;
            else x++ ;
        }

        cout << s1 << s2.substr(y) << endl ;
    }

    return 0 ;
}
```

### 5.2 ACAM

```
class ACAutomation{
private:
  vector<int> fail, end, order ;
  vector<vector<int>> tree ;

  int base, alpha ;

  int new_node(){
    tree.emplace_back(alpha, 0) ;
    fail.push_back(0) ;

    return tree.size() - 1 ;
  }
public:
  ACAutomation(int _base='a', int _alpha=26)
    : base(_base), alpha(_alpha) {
      clear() ;
    }

  void clear(){
```

```
    fail.assign(1, 0) ;
    order.clear() ;
    end.clear() ;
    tree.assign(1, vector<int>(alpha, 0)) ;
  }

  void add_pattern(const string &pattern){
    int u = 0 ;
    for ( auto ch : pattern ){
      int v = ch - base ;

      if(tree[u][v] == 0) tree[u][v] = new_node() ;
      u = tree[u][v] ;
    }

    end.push_back(u) ;
  }

  void build(){
    queue<int> q ;
    order.clear() ;
    order.push_back(0) ;

    for ( int i=0 ; i<alpha ; i++ ) if(tree[0][i] > 0){
      q.push(tree[0][i]) ;
    }

    while(!q.empty()){
      int u = q.front() ; q.pop() ;
      order.push_back(u) ;

      for ( int i=0 ; i<alpha ; i++ ){
        if(tree[u][i] == 0) tree[u][i] = tree[fail[u]][i] ;
        else{
          fail[tree[u][i]] = tree[fail[u]][i] ;
          q.push(tree[u][i]) ;
        }
      }
    }
  }

  vector<int> count_per_pattern(const string &text) const {
    int u = 0 ;
    vector<int> vis(tree.size(), 0) ;

    for ( char ch : text ){
      u = tree[u][ch - base] ;
      vis[u]++ ;
    }

    for ( int i=order.size()-1 ; i>=1 ; i-- ){
      int x = order[i] ;
      vis[fail[x]] += vis[x] ;
    }

    vector<int> ans(end.size(), 0) ;
    for ( int id=0 ; id<end.size() ; id++ ){
      ans[id] = vis[end[id]] ;
    }

    return ans ;
  }
};
```

# 6 Techniques

## 6.1 二分搜

```
// xxxxxoooo 找最小解
bool binary_search(){
    while(l < r){
        int m = (l + r) >> 1 ;
        if(check(m)) r = m ;
        else l = m + 1 ;
    }

    return l ;
}

// ooooxxxx 找最大解
bool binary_search(){
    while(l < r){
        int m = (l + r) >> 1 ;
        if(check(m)) l = m ;
        else r = m - 1 ;
    }

    return l ;
}

// 如果l & r 太大, m = (l + (r - l)) >> 1 ;
```

## 6.2 倍增 LCA

```
#include <bits/stdc++.h>

using namespace std ;

const int Maxn = 500005 ;

vector<int> e[Maxn] ;
int depth[Maxn] ;
int up[Maxn][40] ;
int MaxLog ;

void dfs(int u, int from, int d){
  up[u][0] = from ;
  depth[u] = d ;

  for ( int i=1 ; i<=MaxLog ; i++ ){
    up[u][i] = up[up[u][i - 1]][i - 1] ;
  }

  for ( auto v : e[u] ){
    if(v == from) continue ;
    dfs(v, u, d + 1) ;
  }
}

int lca(int u, int v){
  if(depth[u] < depth[v]) swap(u, v) ;

  for ( int i=MaxLog ; i>=0 ; i-- ) if(depth[u] - (1 << i) >=
       depth[v]){
    u = up[u][i] ;
  }

  if(u == v) return u ;

  for ( int i=MaxLog ; i>=0 ; i-- ) if(up[u][i] != up[v][i]){
    u = up[u][i] ;
    v = up[v][i] ;
  }

  return up[u][0] ;
}

int main(){
  int n, q, root ;
  scanf("%d%d%d", &n, &q, &root) ;
  MaxLog = __lg(n) ;

  for ( int i=0 ; i<n-1 ; i++ ){
    int u, v ;
    scanf("%d%d", &u, &v) ;
    e[u].push_back(v) ;
    e[v].push_back(u) ;
  }

  dfs(root, root, 0) ;

  while(q--){
    int u, v ;
```

```
    scanf("%d%d", &u, &v) ;
    printf("%d\n", lca(u, v)) ;
  }

  return 0 ;
}
```

## 6.3 SG

```
long long SG(long long k){

  if(k % 2 == 0){
    return k / 2;
  }
  else{
    return SG(k / 2);
  }

}

int main(){
  int cas, n;

  scanf("%d", &cas);
  while(cas--){
    scanf("%d", &n);

    long long s, v = 0;

    for(int i = 0; i < n; i++){
      scanf("%lld", &s);
      v ^= SG(s); //XOR
    }

    if(v) printf("YES\n");
    else printf("NO\n");
  }
}

int SG[30] ;
int vis[Maxn], stone[Maxn] ;

void build(){
  SG[0] = 0 ;
  memset(vis, 0, sizeof(vis)) ;

  for ( int i=1 ; i<30 ; i++ ){
    int cur = 0 ;
    for ( int j=0 ; j<i ; j++ ) for ( int k=0 ; k<=j ; k++ ){
      vis[SG[j] ^ SG[k]] = i ;
    }
    while(vis[cur] == i) cur++ ;
    SG[i] = cur ;
  }
}

int main(){
  build() ;

  int T = 0 ;
  while(~scanf("%d", &n) && n){
    int ans = 0 ;

    for ( int i=1 ; i<=n ; i++ ) scanf("%d", &stone[i]) ;

    for ( int i=1 ; i<=n ; i++ ) if(stone[i] & 1){
      ans ^= SG[n-i] ;
    }
    }
}
```

# 7 DP

## 7.1 輪廓線 DP

```
#include <bits/stdc++.h>

using namespace std ;
using ll = long long ;

ll dp[2][(1 << 10) + 5] ;
int n, m ;
int cur ;

void update(int s1, int s2){
  if(s2 & (1 << m)){
    dp[cur][s2 ^ (1 << m)] += dp[cur ^ 1][s1] ;
  }
}

int main(){
  while(~scanf("%d%d", &n, &m)){
    if(m > n) swap(n, m) ;
    memset(dp, 0, sizeof(dp)) ;
    cur = 0 ;
    dp[cur][(1 << m) - 1] = 1 ;
    for ( int i=0 ; i<n ; i++ ) for ( int j=0 ; j<m ; j++ ){
      cur ^= 1 ;
      memset(dp[cur], 0, sizeof(dp[cur])) ;

      for ( int k=0 ; k<(1 << m) ; k++ ){
        update(k, k << 1) ; // not put
        if(i && !(k & (1 << (m - 1)))) update(k, (k << 1) | (1 << m) |
            1) ; // put up
        if(j && !(k & 1)) update(k, (k << 1) | 3) ; // put left
      }
    }
    printf("%lld\n", dp[cur][(1 << m) - 1]) ;
  }
  return 0 ;
}
```

## 7.2 數位 DP

```
#include <bits/stdc++.h>

using namespace std ;

int K ;
int dp[20][105][105][2] ;
vector<int> dig ;

int solve(int pos, int sum, int dsum, bool lim){
    if(pos == -1){
        if(sum == 0 && dsum == 0) return 1 ;
        return 0 ;
    }

    int &d = dp[pos][sum][dsum][lim] ;
    if(d != -1) return d ;

    int up = lim ? dig[pos] : 9 ;
    int res = 0 ;
    for ( int i=0 ; i<=up ; i++ ){
        res += solve(pos-1, (sum * 10 + i) % K, (dsum + i) % K, lim &&
            i==up) ;
    }

    return d = res ;
}

int count(int n){
    memset(dp, -1, sizeof(dp)) ;
    dig.clear() ;

    while(n > 0){
        dig.push_back(n % 10) ;
        n /= 10 ;
    }

    return solve(dig.size() - 1, 0, 0, 1) ;
}
```

```

int main(){
    int T ;
    scanf("%d", &T) ;

    int a, b ;
    while(T--){
        scanf("%d%d%d", &a, &b, &K) ;
        if(K > 90) printf("0\n") ;
        else printf("%d\n", count(b) - count(a-1)) ;
    }

    return 0 ;
}
```

## 7.3 樹 DP

```
#include <bits/stdc++.h>

#define N 505
#define INF 0x3f3f3f3f

using namespace std ;

struct Edge{
  int v, w ;
} ;

vector<Edge> edge[N] ;
int n ;
int cnt[N] ;
int dp[N][N][2] ;

void init(){
  for ( int i=0 ; i<N ; i++ ) edge[i].clear() ;
  memset(cnt, 0, sizeof(cnt)) ;
  memset(dp, INF, sizeof(dp)) ;
}

void DFS(int u){
  cnt[u] = 1 ;
  for ( auto [v, w] : edge[u] ){
    DFS(v) ;
    cnt[u] += cnt[v] ;
  }

  dp[u][1][0] = dp[u][1][1] = 0 ;

  for ( auto [v, w] : edge[u] ){
    for ( int i=cnt[u] ; i>1 ; i-- ) for ( int j=1 ; j<i && j<=cnt[v]
        ; j++ ){
      dp[u][i][1] = min(dp[u][i][1], dp[u][i-j][1] + dp[v][j][1] + 2
          * w) ;
      dp[u][i][0] = min(dp[u][i][0], dp[u][i-j][1] + dp[v][j][0] + w)
          ;
      dp[u][i][0] = min(dp[u][i][0], dp[u][i-j][0] + dp[v][j][1] + 2
          * w) ;
    }
  }
}

int main(){
  int t = 0 ;

  while(~scanf("%d", &n) && n){
    init() ;
    for ( int i=0 ; i<n-1 ; i++ ){
      int u, v, w ;
      scanf("%d%d%d", &v, &u, &w) ;
      edge[u].push_back({v, w}) ;
    }

    DFS(0) ;
    printf("Case %d:\n", ++t) ;

    int q, e ;
    scanf("%d", &q) ;

    while(q--){
      scanf("%d", &e) ;

      for ( int i=n ; i>=1 ; i-- ) if(dp[0][i][0] <= e){
        printf("%d\n", i) ;
        break ;
      }
    }
  }

  return 0 ;
}
```

## 7.4 背包模板

```
// Classic Knapsack Templates
// Usage:
//   - zeroOneKnapsack(w, v, W)    // 0/1 knapsack, each item once
//   - completeKnapsack(w, v, W)   // complete/unbounded knapsack
//   - multipleKnapsack(w, v, c, W) // multiple items, count c[i] via
    binary splitting
// When to use:
//   - 0/1: select subset with capacity limit
//   - complete: unlimited copies of each item
//   - multiple: each item has count c[i] (convert to 0/1 by
    splitting counts)

long long zeroOneKnapsack(const vector<int>& w, const vector<long
    long>& v, int W){
    vector<long long> dp(W+1, 0);
    int n = (int)w.size();
    for(int i=0;i<n;++i){
        for(int c=W;c>=w[i];--c)
            dp[c] = max(dp[c], dp[c-w[i]] + v[i]);
    }
    return *max_element(dp.begin(), dp.end());
}

long long completeKnapsack(const vector<int>& w, const vector<long
    long>& v, int W){
    vector<long long> dp(W+1, 0);
    int n = (int)w.size();
    for(int i=0;i<n;++i){
        for(int c=w[i]; c<=W; ++c)
            dp[c] = max(dp[c], dp[c-w[i]] + v[i]);
    }
    return *max_element(dp.begin(), dp.end());
}

long long multipleKnapsack(vector<int> w, vector<long long> v, const
    vector<int>& cnt, int W){
    // Binary splitting → 0/1 items
    vector<int> ww; vector<long long> vv;
    int n = (int)w.size();
    for(int i=0;i<n;++i){
        int c = cnt[i], k = 1;
        while(c>0){
            int take = min(k, c);
            ww.push_back(w[i]*take);
            vv.push_back(v[i]*take);
            c -= take; k <<= 1;
        }
    }
    return zeroOneKnapsack(ww, vv, W);
}
```

## 7.5 LIS 最長遞增子序列

```
// LIS (Longest Increasing Subsequence) — O(n log n)
// Usage:
//   int len = LISLen(a);
// Notes:
//   - Strictly increasing by default (use <). For non-decreasing,
    replace lower_bound with upper_bound.
```

```cpp

int LISLen(const vector<long long>& a){
    vector<long long> tail; // tail[k] = minimum possible tail of an LIS with length k+1
    for(long long x : a){
        auto it = lower_bound(tail.begin(), tail.end(), x);
        if(it == tail.end()) tail.push_back(x);
        else *it = x;
    }
    return (int)tail.size();
}
```

## 7.6 LCS 最長公共子序列

```cpp
// LCS (Longest Common Subsequence) — O(n*m)
// Usage:
//   int len = LCSLen(s, t);
// Notes: Uses O(min(n,m)) memory if desired; this is a simple O(n*m) DP.

int LCSLen(const string& s, const string& t){
    int n = (int)s.size(), m = (int)t.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            if(s[i-1]==t[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
            else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }
    return dp[n][m];
}
```

## 7.7 編輯距離

```cpp
// Edit Distance (Levenshtein) — O(n*m)
// Usage:
//   int d = editDistance(a, b, 1, 1, 1); // costs: ins, del, sub

int editDistance(const string& a, const string& b, int ins=1, int del=1, int sub=1){
    int n = (int)a.size(), m = (int)b.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
    for(int i=0;i<=n;++i) dp[i][0] = i*del;
    for(int j=0;j<=m;++j) dp[0][j] = j*ins;
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            if(a[i-1]==b[j-1]) dp[i][j] = dp[i-1][j-1];
            else dp[i][j] = min({dp[i-1][j] + del, dp[i][j-1] + ins, dp[i-1][j-1] + sub});
        }
    }
    return dp[n][m];
}
```

## 7.8 TSP 狀壓

```cpp
// TSP (Bitmask DP) — O(n^2 * 2^n)
// Finds min Hamiltonian cycle cost (start at 0) given distance matrix.
// Usage:
//   vector<vector<long long>> dist(n, vector<long long>(n, INF));
//   long long ans = tsp_bitmask(dist);

const long long INF = (long long)4e18;

long long tsp_bitmask(const vector<vector<long long>>& dist){
    int n = (int)dist.size();
    int S = 1<<n;
    vector<vector<long long>> dp(S, vector<long long>(n, INF));
    dp[1][0] = 0; // start at 0
    for(int mask=1; mask<S; ++mask){
        for(int u=0; u<n; ++u){
            if(!(mask & (1<<u)) || dp[mask][u] >= INF) continue;
            for(int v=0; v<n; ++v){
                if(mask & (1<<v)) continue;
                int nmask = mask | (1<<v);
                dp[nmask][v] = min(dp[nmask][v], dp[mask][u] + dist[u][v]);
            }
        }
    }
    long long ans = INF;
    for(int u=0; u<n; ++u){
        if(dp[S-1][u] < INF && dist[u][0] < INF)
            ans = min(ans, dp[S-1][u] + dist[u][0]);
    }
    return ans;
}
```

## 7.9 區間 DP（矩陣鏈乘）

```cpp
// Interval DP — Matrix Chain Multiplication
// Given dimensions p[0..n], matrices are (p[i-1] x p[i]).
// Minimize multiplication cost. Complexity: O(n^3).
// Usage:
//   long long cost = matrixChainMinCost(p);

const long long INF = (long long)4e18;

long long matrixChainMinCost(const vector<int>& p){
    int n = (int)p.size()-1;
    vector<vector<long long>> dp(n+1, vector<long long>(n+1, 0));
    for(int len=2; len<=n; ++len){
        for(int l=1; l+len-1<=n; ++l){
            int r = l+len-1;
            dp[l][r] = INF;
            for(int k=l; k<r; ++k){
                long long cost = dp[l][k] + dp[k+1][r] + 1LL*p[l-1]*p[k]*p[r];
                dp[l][r] = min(dp[l][r], cost);
            }
        }
    }
    return dp[1][n];
}
```

## 7.10 Li Chao Tree（斜率優化）

```cpp
// Li Chao Segment Tree (Min Query)
// Maintains lines y = m x + b over integer x in [X_MIN, X_MAX].
// add_line(m, b), query(x) returns min value at x. Complexity: O(log range).
// When to use:
//   - DP of form dp[i] = min_j (m_j * x_i + b_j) without monotonic slope/query assumptions.

struct LiChao {
    struct Line { long long m, b; long long get(long long x) const { return m*x + b; } };
    struct Node { Line ln; Node *l=nullptr, *r=nullptr; Node(Line v): ln(v) {} };
    long long X_MIN, X_MAX; Node* root;
    static constexpr long long INF = (long long)4e18;

    LiChao(long long xmin, long long xmax): X_MIN(xmin), X_MAX(xmax), root(nullptr) {}

    void add_line(Line nw){ add_line(root, X_MIN, X_MAX, nw); }
    void add_line(long long m, long long b){ add_line({m,b}); }

    void add_line(Node* &p, long long l, long long r, Line nw){
        if(!p){ p = new Node(nw); return; }
        long long mid = (l + r) >> 1;
        bool lef = nw.get(l) < p->ln.get(l);
        bool mdf = nw.get(mid) < p->ln.get(mid);
        if(mdf) swap(nw, p->ln);
```

```
        if(l == r) return;
        if(lef != mdf) add_line(p->l, l, mid, nw);
        else add_line(p->r, mid+1, r, nw);
    }

    long long query(long long x) const { return query(root, X_MIN, X_MAX, x); }
    long long query(Node* p, long long l, long long r, long long x) const{
        if(!p) return INF;
        long long res = p->ln.get(x);
        if(l == r) return res;
        long long mid = (l + r) >> 1;
        if(x <= mid) return min(res, query(p->l, l, mid, x));
        else return min(res, query(p->r, mid+1, r, x));
    }
};
```

## 7.11 單調隊列優化

```
// Monotonic Queue Optimization (Sliding window DP)
// Pattern:
//   dp[i] = cost[i] + min_{j in [i-W, i-1]} dp[j]
// Maintain deque of candidate indices with increasing dp[j], dropping out-of-window.
// Complexity: O(n)
// When to use:
//   - Transition takes min/max over a fixed-width window of previous states.

long long solve_with_window(const vector<long long>& cost, int W){
    int n = (int)cost.size();
    vector<long long> dp(n, (long long)4e18);
    deque<int> dq; // indices of candidates, dp[dq] increasing
    for(int i=0;i<n;++i){
        // remove out-of-window (keep j in [i-W, i-1])
        while(!dq.empty() && dq.front() < i - W) dq.pop_front();
        // base for i: if dq empty, no feasible j; else take min dp[j]
        if(!dq.empty()) dp[i] = cost[i] + dp[dq.front()];
        else if(i==0) dp[i] = cost[i]; // starting point if needed
        // push current i into deque
        while(!dq.empty() && dp[dq.back()] >= dp[i]) dq.pop_back();
        dq.push_back(i);
    }
    return dp.back();
}
```

## 7.12 分治優化模板

```
// Divide & Conquer DP Optimization
// Problems with transitions:
//   dp[k][i] = min_{j < i} { dp[k-1][j] + C(j, i) },
//   where the optimal j (argmin) is monotone: opt[k][i] <= opt[k][i+1].
// Complexity: O(K N log N) or O(K N) depending on cost.
// When to use:
//   - Cost C satisfies quadrangle inequality or structure leading to monotone opt.

const long long INF64 = (long long)4e18;

// Example scaffolding: define your cost function using precomputed prefix sums, etc.
inline long long C(int j, int i){
    // TODO: implement problem-specific cost here
    // e.g., return S[i]-S[j] + (i-j)*X;
    return 0;
}

void compute_layer(int k, int l, int r, int optL, int optR,
                   const vector<long long>& prev, vector<long long>& cur){
    if(l > r) return;
    int mid = (l + r) >> 1;
    pair<long long,int> best = {INF64, -1};
    for(int j=optL; j<=min(mid-1, optR); ++j){
        long long val = prev[j] + C(j, mid);
        if(val < best.first){ best = {val, j}; }
    }
    cur[mid] = best.first;
    int opt = max(best.second, optL);
    compute_layer(k, l, mid-1, optL, opt, prev, cur);
    compute_layer(k, mid+1, r, opt, optR, prev, cur);
}

// Driver:
//   dp[0][i] = base case
//   for k = 1..K: compute_layer(k, 1..N)
```

## 7.13 Knuth 優化模板

```
// Knuth Optimization — Interval DP O(n^2)
// Applies to DP of the form:
//   dp[l][r] = min_{l<=k<r} { dp[l][k] + dp[k+1][r] } + w(l,r)
// Conditions:
//   1) Quadrangle inequality on w: A[a][c] + A[b][d] <= A[a][d] + A[b][c] for a<=b<=c<=d
//   2) Monge array (or equivalent)   opt[l][r-1] <= opt[l][r] <= opt[l+1][r]
// Typical: merging stones, optimal BST, matrix chain variants.

const long long INF = (long long)4e18;

// Example: w(l,r) via prefix sums (replace with your cost)
inline long long w(int l, int r, const vector<long long>& pref){
    // cost of merging [l..r]
    return pref[r] - pref[l-1];
}

long long knuth_dp(const vector<long long>& a){
    int n = (int)a.size();
    vector<long long> pref(n+1,0);
    for(int i=1;i<=n;++i) pref[i]=pref[i-1]+a[i-1];
    vector<vector<long long>> dp(n+2, vector<long long>(n+2, 0));
    vector<vector<int>> opt(n+2, vector<int>(n+2, 0));
    for(int i=1;i<=n;++i) opt[i][i]=i;
    for(int len=2; len<=n; ++len){
        for(int l=1; l+len-1<=n; ++l){
            int r=l+len-1;
            dp[l][r] = INF;
            int st = opt[l][r-1], ed = opt[l+1][r];
            if(st==0) st=l; if(ed==0) ed=r-1;
            for(int k=st; k<=ed; ++k){
                long long val = dp[l][k] + dp[k+1][r] + w(l, r, pref);
                if(val < dp[l][r]){ dp[l][r]=val; opt[l][r]=k; }
            }
        }
    }
    return dp[1][n];
}
```