

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 | 1 |
| 2 Mathematics & Number Theory | 1 |
| 2.1 Number Theory | 1 |
| 2.2 Combinatorics | 1 |
| 2.3 Geometry | 1 |
| 3 Data Structure | 1 |
| 3.1 MST | 1 |
| 3.2 SegmentTree | 1 |
| 3.3 HLD | 1 |
| 3.4 PST | 1 |
| 3.5 Trie | 1 |
| 3.6 BIT 單修區查 | 1 |
| 3.7 BIT 區修單查 | 1 |
| 3.8 BIT 區修區查 | 1 |
| 4 Graph | 1 |
| 4.1 cut vertex AND bridges | 1 |
| 4.2 SCC - Tarjan | 1 |
| 4.3 BCC - Tarjan | 1 |
| 4.4 Convex | 1 |
| 4.5 Max Flow | 1 |
| 4.6 KM (二分圖最大權匹配) | 1 |
| 4.7 min cut max flow | 1 |
| 5 String | 1 |
| 5.1 KMP | 1 |
| 5.2 ACAM | 1 |
| 6 Techniques | 1 |
| 6.1 二分搜 | 1 |
| 6.2 倍增 LCA | 1 |
| 6.3 SG | 1 |
| 7 DP | 1 |
| 7.1 輪廓線 DP | 1 |
| 7.2 數位 DP | 1 |
| 7.3 樹 DP | 1 |
| 7.4 背包模板 | 1 |
| 7.5 LIS 最長遞增子序列 | 1 |
| 7.6 LCS 最長公共子序列 | 1 |
| 7.7 編輯距離 | 1 |
| 7.8 TSP 狀壓 | 1 |
| 7.9 區間 DP (矩陣鏈乘) | 1 |
| 7.10 Li Chao Tree (斜率優化) | 1 |
| 7.11 單調隊列優化 | 1 |
| 7.12 分治優化模板 | 1 |
| 7.13 Knuth 優化模板 | 1 |

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, atan2, sinh ...
16 math.hypot(x, y) #歐幾里德範數
17 math.degrees(x) #x從弧度轉角度
18 math.radians(x) #x從角度轉弧度
19 math.gamma(x) #x的gamma函數
20 math.pi #const
21 math.e #const
22 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 number_of_leading_Zeros(int i)
33 static int number_of_trailing_Zeros(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 // 檢查
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 ooffset, int len)
27 boolean regionMatches(boolean ignoreCase, int toffset, String other,
   int ooffset, 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(CharSequence str[, int fromIndex])
23 int lastIndexOf(CharSequence 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/幕根
27 static double pow(double a, double b)
28 static double sqrt(double a)
29 static double cbrt(double a)
30 static double exp(double a)
31 static double expm1(double x)
32 static double log(double a)
33 static double log10(double a)
34 static double log1p(double x)
35
36 // 三角/雙曲
37 static double sin/cos/tan(double a)
38 static double asin/acos/atan(double a)
39 static double atan2(double y, double x)
40 static double sinh/cosh/tanh(double a)
41
42 // 其他實用
43 static double hypot(double x, double y)
44 static double toDegrees(double angrad)
45 static double toRadians(double angdeg)
46 static double copySign(double magnitude, double sign)
47 static double nextUp/nextDown(double a)
48 static double nextAfter(double start, double direction)
49 static double ulp(double d)
50 static double random()
51 static double scalb(double d, int scaleFactor)
52 static double fma(double a, double b, double c) // (since 8)
53 static long multiplyHigh(long x, long y) // (since 9)
54 static long multiplyFull(int x, int y) // (since 9, 回傳 long)

```

2 Mathematics & Number Theory

2.1 Number Theory

Fermat's Little Theorem:

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

Euler's Theorem:

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

Modular Inverse:

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

Euler Totient:

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

Fast Modular Exponentiation

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

Counting Coprimes Below n

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

其中 μ 為莫比烏斯函數；常用於反演及計算互質對數量。

Modulo Arithmetic Quick Facts

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

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

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

```
1  ll extgcd(ll a, ll b, ll &x, ll &y){
2      if(b == 0){
3          x = 1 ;
4          y = 0 ;
5          return a ;
6      }
7      ll d = extgcd(b, a%b, c, x, y), tmp = x ;
8      x = y ;
9      y = tmp - (a/b)*y ;
10     return d ;
11 }
12
13 // x, y 為 ax + by = gcd(a, b) 的解
14 x = x * c / d ; // ax + by = c 的解
15 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。

```
1  g = gcd(a, m, c, x, y) ;
2  if(g == 1){
3      inv = (x % m + m) % m ;
4  }
```

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

- $a \equiv b \pmod{m} \iff m | (a - b)$ ，同餘類以差整除判斷。
- $a \equiv b \pmod{m} \Rightarrow f(a) \equiv f(b) \pmod{m}$ 對所有以整數係數的多項式 f 成立。

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

Bézout's Identity

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

任何方程 $ax + by = c$ 有整數解當且僅當 $\gcd(a, b) | c$ 。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 $d | c$ 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

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

分解 $m = \prod p_i^{k_i}$ 後

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

實際上，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)

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

常用判別： $a^x \equiv a^y \pmod{m}$ 當且僅當 $x \equiv y \pmod{\text{ord}_m(a)}$ (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

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

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

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

兩式合併 (允許非互質)

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

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

```

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

2.2 Combinatorics

Binomial Coefficient Identities

$$\begin{aligned} \binom{n}{k} &= \frac{n!}{k!(n-k)!}, \\ \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1}, \\ \sum_{k=0}^n \binom{n}{k} &= 2^n, \quad \sum_{k=0}^n k \binom{n}{k} = n2^{n-1}. \end{aligned}$$

Stars and Bars 非負整數解數量：

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

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 ：

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

計算滿足限制的排列或整數解時廣泛使用。

Catalan Numbers 基本定義：

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

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

常見應用包含合法括號、凸多邊形三角剖分、二元樹結構計數等。

2.3 Geometry

對於 V 個點， E 條邊， F 個面， C 個連通分量

$$V + F = E + 2$$

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

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

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

3 Data Structure

3.1 MST

```

1 struct Edge{
2     int u, v, w ;
3     // 這是最大生成樹，最小生成樹要改成 w < o.w
4     bool operator>(const Edge &o) const {return w > o.w ;} ;
5 } ;
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 ; i++){
42         auto [u, v, w] = edge[i] ;
43         if(merge(u, v)){
44             cnt++ ;
45             sum -= w ;
46         }
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(), greater<Edge>()) ;
58     MST() ;
59 }
```

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 + seg[rc].sum ;
22     }
23
24     void AddTag(int id, LazyTag &tag){
25         if(tag.type == 0){
26             seg[id].sum += tag.val * seg[id].sz ;
27             seg[id].tag.val += tag.val ;
28         }
29         else{
30             seg[id].sum = tag.val * seg[id].sz ;
31             seg[id].tag = {1, tag.val} ;
32         }
33     }
34
35     void push(int id){
36         AddTag(lc, seg[id].tag) ;
37         AddTag(rc, seg[id].tag) ;
38         seg[id].tag = {0, 0} ;
39     }
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, int L=1, int R=n, int
61     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, lc) ;
70         else if(l > M) modify(l, r, tag, M+1, R, rc) ;
71         else{
72             modify(l, r, tag, L, M, lc) ;
73             modify(l, r, tag, M+1, R, rc) ;
74         }
75         pull(id) ;
76     }
77
78     ll query(int l, int r, int L=1, int R=n, int id=1){
79         if(l <= L && R <= r) return seg[id].sum ;
80
81         push(id) ;
82         int M = (L + R) >> 1 ;
83         if(r <= M) return query(l, r, L, M, lc) ;
84         else if(l > M) return query(l, r, M+1, R, rc) ;
85     }
86 }tree ;

```

3.3 HLD

/* HLD */

```

2     int fa[Maxn], top[Maxn], son[Maxn], sz[Maxn], dep[Maxn] = {0},
3         dfn[Maxn], rnk[Maxn], dfscnt = 0 ;
4
5     void dfs1(int u, int from){
6         fa[u] = from ;
7         dep[u] = dep[from] + 1 ;
8         sz[u] = 1 ;
9
10        for ( auto v : g[u] ) if(v != from){
11            dfs1(v, u) ;
12            sz[u] += sz[v] ;
13            if(son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v ;
14        }
15    }
16
17    void dfs2(int u, int t){
18        top[u] = t ;
19        dfn[u] = ++dfscnt ;
20        rnk[dfscnt] = u ;
21
22        if(son[u] == -1) return ;
23
24        dfs2(son[u], t) ;
25
26        for ( auto v : g[u] ) if(v != fa[u] && v != son[u]){
27            dfs2(v, v) ;
28        }
29
30        /* Segment Tree */
31        #define lc (id << 1)
32        #define rc ((id << 1) | 1)
33
34        struct ColorSeg{
35            int left, right, tot ;
36
37            ColorSeg operator+(const ColorSeg &o) const {
38                if(tot == 0) return o ;
39                if(o.tot == 0) return *this ;
40
41                ColorSeg tmp ;
42                tmp.left = left ;
43                tmp.right = o.right ;
44                tmp.tot = tot + o.tot - (right == o.left) ;
45
46                return tmp ;
47            }
48        } ;
49
50        struct Node{
51            ColorSeg color ;
52            int tag ;
53        }seg[Maxn << 2] ;
54
55        class SegmentTree{
56        private:
57            void pull(int id){
58                // normal pull
59            }
60
61            void AddTag(int id, int tag){
62                // normal AddTag
63            }
64
65            void push(int id){
66                // normal push
67            }
68
69            void modify(int l, int r, int tag, int L=1, int R=n, int
70            id=1){
71                // normal modify
72            }
73
74            ColorSeg query(int l, int r, int L=1, int R=n, int
75            id=1){
76                // normal query
77            }
78
79            public:
80                void build(int L=1, int R=n, int id=1){
81                    // normal build
82                }
83            }
84        }
85    }
86

```

```

79 }
80
81 // update val from u to v (simple path)
82 void update(int u, int v, int val){
83     while(top[u] != top[v]){
84         if(dep[top[u]] < dep[top[v]]) swap(u, v) ;
85         modify(dfn[top[u]], dfn[u], val) ;
86         u = fa[top[u]] ;
87     }
88
89     if(dep[u] < dep[v]) swap(u, v) ;
90     modify(dfn[v], dfn[u], val) ;
91 }
92
93 // get sum from u to v (simple path)
94 int get(int u, int v){
95     pair<int, ColorSeg> U, V ;
96     ColorSeg M ;
97     U = {u, {0, 0, 0}} ;
98     V = {v, {0, 0, 0}} ;
99
100    while(top[U.first] != top[V.first]){
101        if(dep[top[U.first]] < dep[top[V.first]]) swap(U, V) ;
102        U.second = query(dfn[top[U.first]], dfn[U.first]) + U.second ;
103        U.first = fa[top[U.first]] ;
104    }
105
106    if(dep[U.first] < dep[V.first]) swap(U, V) ;
107
108    M = query(dfn[V.first], dfn[U.first]) ;
109
110    return (U.second.tot + V.second.tot + M.tot) - (U.second.left ==
111        M.right) - (V.second.left == M.left) ;
112 }
113 }tree ;
114 void init(){
115     memset(son, -1, sizeof(son)) ;
116 }
```

3.4 PST

```

1 // Find range k-th largest number
2 struct Node{
3     int sum, left, right ;
4 }seg[Maxn + 20 * Maxn] ;
5
6 class PersistentSegmentTree{
7 private:
8     int n ;
9     int cnt ;
10    vector<int> version ;
11
12    int build(int L, int R){
13        int cur_cnt = cnt++ ;
14        if(L == R){
15            seg[cur_cnt] = {0, 0, 0} ;
16            return cur_cnt ;
17        }
18
19        int M = (L + R) >> 1 ;
20        int lc = build(L, M) ;
21        int rc = build(M+1, R) ;
22
23        seg[cur_cnt] = {0, lc, rc} ;
24        return cur_cnt ;
25    }
26 public:
27    PersistentSegmentTree(int _n){
28        n = _n ;
29        cnt = 0 ;
30
31        int root = build(1, n) ;
32        version.push_back(root) ;
33    }
34
35    void update(int ver, int idx){
```

```

36     auto upd = [&](auto &&self, const int cur, int L, int R){
37         int cur_cnt = cnt++ ;
38
39         if(L == R){
40             seg[cur_cnt] = {seg[cur].sum + 1, 0, 0} ;
41             return cur_cnt ;
42         }
43
44         int M = (L + R) >> 1 ;
45         int lc = seg[cur].left ;
46         int rc = seg[cur].right ;
47
48         if(idx <= M) lc = self(self, seg[cur].left, L, M) ;
49         else rc = self(self, seg[cur].right, M+1, R) ;
50
51         seg[cur_cnt] = {seg[lc].sum + seg[rc].sum, lc, rc} ;
52
53         return cur_cnt ;
54    };
55
56    int root = upd(upd, version[ver], 1, n) ;
57    version.push_back(root) ;
58 }
59
60    int query(int verL, int verR, int k){
61        auto qry = [&](auto &&self, const int cur_old, const int cur_new,
62                        int L, int R){
63            if(L == R) return L ;
64
65            int old_l = seg[cur_old].left, old_r = seg[cur_old].right ;
66            int new_l = seg[cur_new].left, new_r = seg[cur_new].right ;
67
68            int dl = seg[new_l].sum - seg[old_l].sum ;
69            int dr = seg[new_r].sum - seg[old_r].sum ;
70
71            int M = (L + R) >> 1 ;
72
73            if(dl >= k) return self(self, old_l, new_l, L, M) ;
74            k -= dl ;
75            return self(self, old_r, new_r, M+1, R) ;
76        };
77
78        int idx = qry(qry, version[verL-1], version[verR], 1, n) ;
79        return idx ;
80    };
81 }
```

3.5 Trie

```

1 class TrieNode{
2 public:
3     set<int> end ;
4     TrieNode *next[26] ;
5
6     TrieNode(){
7         for ( int i=0 ; i<26 ; i++ ) next[i] = nullptr ;
8     }
9 };
10
11 class Trie{
12 private:
13     int cnt ;
14     TrieNode *root ;
15 public:
16     Trie() : cnt(0) {
17         root = new TrieNode() ;
18     }
19
20     void insert(string &str, int n){
21         TrieNode* node = root ;
22         for ( auto s : str ){
23             int path = s - 'a' ;
24
25             if(node->next[path] == nullptr) node->next[path] = new
26                 TrieNode() ;
27             node = node->next[path] ;
28         }
29     }
30 }
```

```

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

```

3.6 BIT 單修區查

```

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

```

3.7 BIT 區修單查

```

1 // 區間修改，單點查詢
2 #define lowbit(x) (x & -x)
3
4 int BIT[MAX_SIZE] ;
5 int n ;
6
7 void modify(int idx, int val){
8     for ( ; idx <= n ; idx += lowbit(idx) ){
9         BIT[idx] += val ;
10    }
11 }

```

```

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

```

3.8 BIT 區修區查

```

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

```

4 Graph

4.1 cut vertex AND bridges

```

1 int dfn[Maxn] = {-1}, low[Maxn] = {-1}, dfscnt ;
2
3 void dfs(int u, int fa){
4     dfn[u] = low[u] = ++dfscnt ;

```

```

5 int child = 0 ;
6
7 for ( auto v : g[u] ) if(v != fa){
8     if(dfn[v] == -1){
9         child++ ;
10        dfs(v, u) ;
11        low[u] = min(low[u], low[v]) ;
12
13        if(low[v] >= dfn[u]){
14            // this edge is a bridge
15        }
16
17        if(u != fa && low[v] >= dfn[u]){
18            // this node v is a articulation point
19        }
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], dfscnt = 0, cnt_scc = 0 ;
3 stack<int> st ;
4 bitset<Maxn> inSt, vis ;
5
6 void dfs(int u, int from){
7     dfn[u] = low[u] = ++dfscnt ;
8     st.push(u) ;
9     inSt[u] = 1 ;
10
11    for ( auto v : g[u] ){
12        if(!inSt[v] && dfn[v] != -1) continue ;
13        if(dfn[v] == -1) dfs(v, u) ;
14        low[u] = min(low[u], low[v]) ;
15    }
16
17    if(dfn[u] == low[u]){
18        cnt_scc++ ;
19        int x ;
20
21        do{
22            x = st.top() ;
23            st.pop() ;
24
25            inSt[x] = 0 ;
26            sccId[x] = cnt_scc ;
27            scc[cnt_scc].push_back(x) ;
28        }
29        while(x != u) ;
30    }
31 }
32
33 // SCC to DAG (after dfs)
34 vector<int> dag[Maxn] ;
35
36 void scc_to_dag(){
37     vector<int> dag[Maxn] ;
38     for ( int u=1 ; u<=n ; u++ ){
39         for ( auto v : g[u] ){
40             if(sccId[u] != sccId[v]){
41                 dag[sccId[u]].push_back(sccId[v]) ;
42             }
43         }
44     }
45 }
46
47 void init(){
48     memset(dfn, -1, sizeof(dfn)) ;
49     memset(low, -1, sizeof(low)) ;
50 }
```

```

52 int main(){
53     init() ;
54     input() ;
55     for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
56         dfs(i, i) ;
57     }
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], vis_bcc[Maxn], bcc_cnt = 0 ;
16
17 void dfs1(int u, int from){
18     dfn[u] = low[u] = ++dfn_cnt ;
19
20     for ( int i=head[u] ; i!= -1 ; i=e[i].next ){
21         int v = e[i].v ;
22
23         if(dfn[v] == -1){
24             dfs1(v, i) ;
25             if(dfn[u] < low[v]) bz[i] = bz[i^1] = 1 ;
26             low[u] = min(low[u], low[v]) ;
27         }
28         else if(i != (from ^ 1)) low[u] = min(low[u], dfn[v]) ;
29     }
30 }
31
32 void dfs2(int u, int id){
33     vis_bcc[u] = id ;
34     bcc[id].push_back(u) ;
35
36     for ( int i=head[u] ; i!= -1 ; i=e[i].next ){
37         int v = e[i].v ;
38
39         if(vis_bcc[v] != -1 || bz[i]) continue ;
40         dfs2(v, id) ;
41     }
42 }
43
44 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] == -1){
54         dfs1(i, 0) ;
55     }
56
57     for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i] == -1){
58         bcc.push_back(vector<int>()) ;
59         dfs2(i, bcc_cnt++) ;
60     }
61 }
```

4.4 Convex

```

1 struct Coordinate{
2     long long x, y ;
3
4     friend bool operator<(const Coordinate&a, const Coordinate& b){
5         if(a.x == b.x) return a.y < b.y ;
6         return a.x < b.x ;
7     }
8
9     friend bool operator==(const Coordinate& a, const Coordinate& b){
10        return a.x == b.x && a.y == b.y ;
11    }
12}
13
14 vector<Coordinate> nodes ;
15
16 long long cross(const Coordinate& o, const Coordinate& a, const
17     Coordinate& b){
18     return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x) ;
19 }
20
21 void input(){
22     nodes.clear() ;
23
24     int n, x, y ;
25     char c ;
26     cin >> n ;
27
28     for ( int i=0 ; i<n ; i++ ){
29         cin >> x >> y >> c ;
30         if(c == 'Y') nodes.push_back({x, y}) ;
31     }
32
33 void monotone(){
34     sort(nodes.begin(), nodes.end()) ;
35
36     int n = unique(nodes.begin(), nodes.end()) - nodes.begin() ;
37
38     vector<Coordinate> ch(n+1) ;
39
40     int m = 0 ;
41
42     for ( int i=0 ; i<n ; i++ ){
43         while(m > 1 && cross(ch[m-2], ch[m-1], nodes[i]) < 0) m-- ;
44         ch[m++] = nodes[i] ;
45     }
46     for ( int i=n-2, t=m ; i>=0 ; i-- ){
47         while(m > t && cross(ch[m-2], ch[m-1], nodes[i]) < 0) m-- ;
48         ch[m++] = nodes[i] ;
49     }
50
51     if(n > 1) m-- ;
52     cout << m << endl ;
53
54     for ( int i=0 ; i<m ; i++ ) cout << ch[i].x << " " << ch[i].y <<
55         endl ;
56 }
57
58 q.push(S) ;
59 dep[S] = 0 ;
60
61 while(!q.empty()){
62     int u = q.front() ; q.pop() ;
63
64     for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
65         int v = e[i].v ;
66         if(dep[v] == -1 && e[i].cap > 0){
67             dep[v] = dep[u] + 1 ;
68             if(v == T) return 1 ;
69             q.push(v) ;
70     }
71 }
72 }
73
74 return 0 ;
75 }
76
77 int dfs(int u, int flow){
78     if(u == T) return flow ;
79     int d, rest = 0 ;
80
81     for ( int &i=cur[u] ; i!= -1 ; i=e[i].next ){
82         int v = e[i].v ;
83         if(dep[v] == dep[u] + 1 && e[i].cap > 0){
84             d = dfs(v, min(flow - rest, e[i].cap)) ;
85
86             if(d > 0){
87                 e[i].cap -= d ;
88                 e[i^1].cap += d ;
89                 rest += d ;
90
91                 if(rest == flow) break ;
92             }
93         }
94     }
95
96     if(rest != flow) dep[u] = -1 ;
97     return rest ;
98 }
99
100 public:
101 MaxFlow(int n, int s, int t){
102     N = n ; S = s ; T = t ;
103     e.reserve(n*n) ;
104     head.assign(n+1, -1) ;
105     cur.resize(n+1) ;
106     dep.resize(n+1) ;
107 }
108
109 void AddEdge(int u, int v, int cap){
110     e.push_back({v, cap, head[u]}) ;
111     head[u] = e.size() - 1 ;
112     e.push_back({u, 0, head[v]}) ;
113     head[v] = e.size() - 1 ;
114 }
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
```

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     }
```

4.6 KM (二分圖最大權匹配)

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

```

8 // - Complexity: O(N^3), N = max(nLeft, nRight).
9 // - If nLeft != nRight, internally pads the smaller side with 0
10 // weights.
11 // - If an edge (u, v) is not allowed, you may set a very negative
12 // weight (e.g., -INF) to forbid it and effectively achieve "max weight
13 // among valid edges".
14 // - For minimum weight matching, invert weights (e.g., add a
15 // constant or negate).
16
17 struct KM {
18     int nL, nR, N; // N = max(nL, nR)
19     vector<vector<long long>> w; // 1..N x 1..N (padded with 0)
20     vector<long long> lx, ly, slack; // labels and slack for Y
21     vector<int> slackx, prevy; // slack argmin and previous X for
22     // Y in augmenting tree
23     vector<int> xy, yx; // matching: xy[x]=y, yx[y]=x
24     static constexpr long long INF = (long long)4e18;
25
26     KM(int _nL, int _nR){
27         nL = _nL; nR = _nR; N = max(nL, nR);
28         w.assign(N+1, vector<long long>(N+1, 0));
29         lx.assign(N+1, 0);
30         ly.assign(N+1, 0);
31         slack.assign(N+1, 0);
32         slackx.assign(N+1, 0);
33         prevy.assign(N+1, 0);
34         xy.assign(N+1, 0);
35         yx.assign(N+1, 0);
36     }
37
38     void addEdge(int u, int v, long long weight){
39         if(u<=0 || u>nL || v<=0 || v>nR) return;
40         w[u][v] = max(w[u][v], weight); // keep the best weight if
41         // duplicated
42     }
43
44     // Core augment step for left node 'root'
45     void augment(int root){
46         vector<char> S(N+1, false), T(N+1, false);
47         queue<int> q;
48         q.push(root); S[root] = true;
49         for(int y=1; y<=N; ++y){
50             slack[y] = lx[root] + ly[y] - w[root][y];
51             slackx[y] = root;
52             prevy[y] = 0;
53         }
54
55         int y_free = 0; // free Y found to augment
56         while(true){
57             while(!q.empty()){
58                 int x = q.front(); q.pop();
59                 for(int y=1; y<=N; ++y){
60                     if(T[y]) continue;
61                     long long d = lx[x] + ly[y] - w[x][y];
62                     if(d == 0){
63                         // tight edge, add to tree
64                         T[y] = true; prevy[y] = x;
65                         if(yx[y] == 0){ y_free = y; goto AUG; }
66                         // else grow the tree
67                         int x2 = yx[y];
68                         if(!S[x2]){ S[x2] = true; q.push(x2);
69                             for(int yy=1; yy<=N; ++yy){
70                                 long long dd = lx[x2] + ly[yy] -
71                                     w[x2][yy];
72                                 if(dd < slack[yy]){ slack[yy] = dd;
73                                     slackx[yy] = x2; }
74                             }
75                         } else if(d < slack[y]){
76                             slack[y] = d; slackx[y] = x;
77                         }
78                     }
79                     // relabel
80                     long long delta = INF;
81                     for(int y=1; y<=N; ++y) if(!T[y]) delta = min(delta,
82                         slack[y]);
83                 }
84             }
85         }
86     }
87
88     // Complexity: O(N^3), N = max(nLeft, nRight).
89     // - If nLeft != nRight, internally pads the smaller side with 0
90     // weights.
91     // - If an edge (u, v) is not allowed, you may set a very negative
92     // weight (e.g., -INF) to forbid it and effectively achieve "max weight
93     // among valid edges".
94     // - For minimum weight matching, invert weights (e.g., add a
95     // constant or negate).
96
97     KM(int _nL, int _nR){
98         nL = _nL; nR = _nR; N = max(nL, nR);
99         w.assign(N+1, vector<long long>(N+1, 0));
100        lx.assign(N+1, 0);
101        ly.assign(N+1, 0);
102        slack.assign(N+1, 0);
103        slackx.assign(N+1, 0);
104        prevy.assign(N+1, 0);
105    }
106
107    void augment(int x){
108        int y = prevy[x];
109        lx[x] = w[x][y];
110        for(int y=2; y<=N; ++y) lx[x] = max(lx[x], w[x][y]);
111
112        fill(ly.begin(), ly.end(), 0);
113        fill(xy.begin(), xy.end(), 0);
114        fill(yx.begin(), yx.end(), 0);
115
116        for(int x=1; x<=N; ++x) augment(x);
117
118        long long ans = 0;
119        // Sum only over real nodes to avoid counting padded matches
120        for(int x=1; x<=nL; ++x){
121            int y = xy[x];
122            if(1 <= y && y <= nR) ans += w[x][y];
123        }
124        return ans;
125    }
126
127    // Right side matching (1..nR), value is the matched left index
128    // (or 0 if unmatched)
129    vector<int> match_right() const{
130        vector<int> ret(nR+1, 0);
131        for(int y=1; y<=nR; ++y){
132            if(y<=N) ret[y] = (y<= (int)yx.size()-1 ? yx[y] : 0);
133        }
134        return ret;
135    }
136
137    // Left side matching (1..nL), value is the matched right index
138    // (or 0 if unmatched)
139    vector<int> match_left() const{
140        vector<int> ret(nL+1, 0);
141        for(int x=1; x<=nL; ++x){
142            if(x<=N) ret[x] = (x<= (int)xy.size()-1 ? xy[x] : 0);
143        }
144    };
145
146    /*
147     * Example:
148     * int n = 3, m = 3;
149     * KM km(n, m);
150     * km.addEdge(1, 1, 5);
151     * km.addEdge(1, 2, 6);
152     */

```

```

77     for(int x=1; x<=N; ++x) if(S[x]) lx[x] -= delta;
78     for(int y=1; y<=N; ++y){
79         if(T[y]) ly[y] += delta;
80         else slack[y] -= delta;
81     }
82     for(int y=1; y<=N; ++y){
83         if(!T[y] && slack[y] == 0){
84             T[y] = true; prevy[y] = slackx[y];
85             if(yx[y] == 0){ y_free = y; goto AUG; }
86             int x2 = yx[y];
87             if(!S[x2]){ S[x2] = true; q.push(x2);
88                 for(int yy=1; yy<=N; ++yy){
89                     long long dd = lx[x2] + ly[yy] - w[x2][yy];
90                     if(dd < slack[yy]){ slack[yy] = dd;
91                         slackx[yy] = x2; }
92                 }
93             } else if(x2 == y_free){
94                 slack[y] = dd; slackx[y] = x2;
95             }
96         }
97     }
98 }
99
100 AUG:
101     // Augment along alternating path ending at y_free
102     while(y_free){
103         int x = prevy[y_free];
104         int next_y = xy[x];
105         yx[y_free] = x; xy[x] = y_free;
106         y_free = next_y;
107     }
108
109     long long solve(){
110         // init labels
111         for(int x=1; x<=N; ++x){
112             lx[x] = w[x][1];
113             for(int y=2; y<=N; ++y) lx[x] = max(lx[x], w[x][y]);
114         }
115         fill(ly.begin(), ly.end(), 0);
116         fill(xy.begin(), xy.end(), 0);
117         fill(yx.begin(), yx.end(), 0);
118
119         for(int x=1; x<=N; ++x) augment(x);
120
121         long long ans = 0;
122         // Sum only over real nodes to avoid counting padded matches
123         for(int x=1; x<=nL; ++x){
124             int y = xy[x];
125             if(1 <= y && y <= nR) ans += w[x][y];
126         }
127         return ans;
128     }
129
130     // Right side matching (1..nR), value is the matched left index
131     // (or 0 if unmatched)
132     vector<int> match_right() const{
133         vector<int> ret(nR+1, 0);
134         for(int y=1; y<=nR; ++y){
135             if(y<=N) ret[y] = (y<= (int)yx.size()-1 ? yx[y] : 0);
136         }
137         return ret;
138     }
139
140     // Left side matching (1..nL), value is the matched right index
141     // (or 0 if unmatched)
142     vector<int> match_left() const{
143         vector<int> ret(nL+1, 0);
144         for(int x=1; x<=nL; ++x){
145             if(x<=N) ret[x] = (x<= (int)xy.size()-1 ? xy[x] : 0);
146         }
147     };
148
149     /*
150      * Example:
151      * int n = 3, m = 3;
152      * KM km(n, m);
153      * km.addEdge(1, 1, 5);
154      * km.addEdge(1, 2, 6);
155      */

```

```

152 km.addEdge(2, 2, 4);
153 km.addEdge(2, 3, 7);
154 km.addEdge(3, 1, 3);
155 long long ans = km.solve(); // maximum weight
156 auto y2x = km.match_right(); // y2x[v] = matched u
157 */

```

4.7 min cut max flow

```

1 struct Edge{
2     int v, cap, cost , next ;
3 };
4
5 using pii = pair<int, int> ;
6 class MCMF{
7 private:
8     int N, s, t, tot ;
9     vector<Edge> e ;
10    vector<int> head ;
11 public:
12    MCMF(int n, int _s, int _t){
13        N = n ;
14        s = _s ;
15        t = _t ;
16        e.resize(n*n + 5) ;
17        head.assign(n+5, -1) ;
18        tot = -1 ;
19    }
20
21    void AddEdge(int u, int v, int cap, int cost){
22        e[++tot] = {v, cap, cost, head[u]} ;
23        head[u] = tot ;
24        e[++tot] = {u, 0, -cost, head[v]} ;
25        head[v] = tot ;
26    }
27
28    int run(){
29        vector<int> dis(N+1), pot(N+1, 0), preE(N+1) ;
30        int flow = 0, cost = 0 ;
31
32        auto dijkstra = [&](){
33            fill(dis.begin(), dis.end(), INF) ;
34            priority_queue<pii, vector<pii>, greater<pii>> pq ;
35            dis[s] = 0 ;
36            pq.push({0, s}) ;
37
38            while(!pq.empty()){
39                auto [d, u] = pq.top() ; pq.pop() ;
40                if(d > dis[u]) continue ;
41                for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
42                    int v = e[i].v, cap = e[i].cap, w = e[i].cost ;
43                    if(cap && dis[v] > d + w + pot[u] - pot[v]){
44                        dis[v] = d + w + pot[u] - pot[v] ;
45                        preE[v] = i ;
46                        pq.push({dis[v], v}) ;
47                    }
48                }
49            }
50
51            return dis[t] != INF ;
52        };
53
54        while(dijkstra()){
55            for ( int v=1 ; v<=N ; v++ ) if(dis[v] < INF){
56                pot[v] += dis[v] ;
57            }
58
59            int aug = INT_MAX ;
60            for ( int v=t ; v!=s ; v=e[preE[v]^1].v ){
61                aug = min(aug, e[preE[v]].cap) ;
62            }
63
64            for ( int v=t ; v!=s ; v=e[preE[v]^1].v ){
65                e[preE[v]].cap -= aug ;
66                e[preE[v]^1].cap += aug ;
67                cost += aug * e[preE[v]].cost ;
68            }

```

```

69    }
70
71    return cost ;
72 }
73 */

```

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

```

5.2 ACAM

```

1 class ACAutomaton{
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    ACAutomaton(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] = new_node() ;
33         u = tree[u][v] ;
34     }
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++ ) if(tree[0][i] > 0){
45         q.push(tree[0][i]) ;
46     }
47
48     while(!q.empty()){
49         int u = q.front() ; q.pop() ;
50         order.push_back(u) ;
51
52         for ( int i=0 ; i<alpha ; i++ ){
53             if(tree[u][i] == 0) tree[u][i] = tree[fail[u]][i] ;
54             else{
55                 fail[tree[u][i]] = tree[fail[u]][i] ;
56                 q.push(tree[u][i]) ;
57             }
58         }
59     }
60 }
61
62 vector<int> count_per_pattern(const string &text) const {
63     int u = 0 ;
64     vector<int> vis(tree.size(), 0) ;
65
66     for ( char ch : text ){
67         u = tree[u][ch - base] ;
68         vis[u]++;
69     }
70
71     for ( int i=order.size()-1 ; i>=1 ; i-- ){
72         int x = order[i] ;
73         vis[fail[x]] += vis[x] ;
74     }
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
81     return ans ;
82 }
83 };

```

6 Techniques

6.1 二分搜

```

1 // xxxxxxxx 找最小解
2 bool binary_search(){
3     while(l < r){
4         int m = (l + r) >> 1 ;
5         if(check(m)) r = m ;
6         else l = m + 1 ;
7     }
8 }

```

```

9     return 1 ;
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 1 ;
21 }
22
23 // 如果l & r 太大, m = (l + (r - 1)) >> 1 ;

```

6.2 倍增 LCA

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 const int Maxn = 500005 ;
6
7 vector<int> e[Maxn] ;
8 int depth[Maxn] ;
9 int up[Maxn][40] ;
10 int MaxLog ;
11
12 void dfs(int u, int from, int d){
13     up[u][0] = from ;
14     depth[u] = d ;
15
16     for ( int i=1 ; i<=MaxLog ; i++ ){
17         up[u][i] = up[up[u][i - 1]][i - 1] ;
18     }
19
20     for ( auto v : e[u] ){
21         if(v == from) continue ;
22         dfs(v, u, d + 1) ;
23     }
24 }
25
26 int lca(int u, int v){
27     if(depth[u] < depth[v]) swap(u, v) ;
28
29     for ( int i=MaxLog ; i>=0 ; i-- ) if(depth[u] - (1 << i) >=
30         depth[v]){
31         u = up[u][i] ;
32     }
33
34     if(u == v) return u ;
35
36     for ( int i=MaxLog ; i>=0 ; i-- ) 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
56     dfs(root, root, 0) ;
57
58     while(q--){
59         int u, v ;
60
61         if(depth[u] < depth[v]) swap(u, v) ;
62
63         for ( int i=MaxLog ; i>=0 ; i-- ) if(depth[u] - (1 << i) >=
64             depth[v]){
65             u = up[u][i] ;
66         }
67
68         if(u == v) cout << u << endl ;
69         else cout << up[u][0] << endl ;
70     }
71 }

```

```

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 k=0 ; k<=j ; k++ ){
41             vis[SG[j] ^ SG[k]] = i ;
42         }
43         while(vis[cur] == i) cur++ ;
44         SG[i] = cur ;
45     }
46 }
47
48 int main(){
49     build() ;
50
51     T = 0 ;
52     while(~scanf("%d", &n) && n){
53         int ans = 0 ;
54
55         for ( int i=1 ; i<=n ; i++ ) scanf("%d", &stone[i]) ;
56
57         for ( int i=1 ; i<=n ; i++ ) if(stone[i] & 1){
58             ans ^= SG[n-i] ;
59         }
60     }
61 }
```

7 DP

7.1 輪廓線 DP

```

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

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 lim){
10     if(pos == -1){
11         if(sum == 0 && dsum == 0) return 1 ;
12         return 0 ;
13     }
14
15     int &d = dp[pos][sum][dsum][lim] ;
16     if(d != -1) return d ;
17
18     int up = lim ? dig[pos] : 9 ;
19     int res = 0 ;
20     for ( int i=0 ; i<=up ; i++ ){
21         res += solve(pos-1, (sum * 10 + i) % K, (dsum + i) % K, lim & i==up) ;
22     }
23
24     return d = res ;
25 }
26
27 int count(int n){
28     memset(dp, -1, sizeof(dp)) ;
29     dig.clear() ;
30
31     while(n > 0){
32         dig.push_back(n % 10) ;
33         n /= 10 ;
34     }
35
36     return solve(dig.size() - 1, 0, 0, 1) ;
37 }
```

```

38
39 int main(){
40     int T ;
41     scanf("%d", &T) ;
42
43     int a, b ;
44     while(T--){
45         scanf("%d%d%d", &a, &b, &K) ;
46         if(K > 90) printf("0\n") ;
47         else printf("%d\n", count(b) - count(a-1)) ;
48     }
49
50     return 0 ;
51 }
```

7.3 樹 DP

```

1 #include <bits/stdc++.h>
2
3 #define N 505
4 #define INF 0x3f3f3f3f
5
6 using namespace std ;
7
8 struct Edge{
9     int v, w ;
10 } ;
11
12 vector<Edge> edge[N] ;
13 int n ;
14 int cnt[N] ;
15 int dp[N][N][2] ;
16
17 void init(){
18     for ( int i=0 ; i<N ; i++ ) edge[i].clear() ;
19     memset(cnt, 0, sizeof(cnt)) ;
20     memset(dp, INF, sizeof(dp)) ;
21 }
22
23 void DFS(int u){
24     cnt[u] = 1 ;
25     for ( auto [v, w] : edge[u] ){
26         DFS(v) ;
27         cnt[u] += cnt[v] ;
28     }
29
30     dp[u][1][0] = dp[u][1][1] = 0 ;
31
32     for ( auto [v, w] : edge[u] ){
33         for ( int i=cnt[u] ; i>1 ; i-- ) for ( int j=1 ; j<i && j<=cnt[v]
34             ; j++ ){
35             dp[u][i][1] = min(dp[u][i][1], dp[u][i-j][1] + dp[v][j][1] + 2
36             * w) ;
37             dp[u][i][0] = min(dp[u][i][0], dp[u][i-j][1] + dp[v][j][0] + w)
38             ;
39             dp[u][i][0] = min(dp[u][i][0], dp[u][i-j][0] + dp[v][j][1] + 2
40             * w) ;
41     }
42 }
43
44 int main(){
45     int t = 0 ;
46
47     while(~scanf("%d", &n) && n){
48         init() ;
49         for ( int i=0 ; i<n-1 ; i++ ){
50             int u, v, w ;
51             scanf("%d%d%d", &v, &u, &w) ;
52             edge[u].push_back({v, w}) ;
53         }
54
55         DFS(0) ;
56         printf("Case %d:\n", ++t) ;
57     }
58 }
```

```

57     while(q--){
58         scanf("%d", &e) ;
59
60         for ( int i=n ; i>=1 ; i-- ) if(dp[0][i][0] <= e){
61             printf("%d\n", i) ;
62             break ;
63         }
64     }
65 }
66
67 return 0 ;
68 }
```

7.4 背包模板

```

1 // Classic Knapsack Templates
2 // Usage:
3 //   - zeroOneKnapsack(w, v, W)    // 0/1 knapsack, each item once
4 //   - completeKnapsack(w, v, W)   // complete/unbounded knapsack
5 //   - multipleKnapsack(w, v, c, W) // multiple items, count c[i] via
6 //       binary splitting
7 // When to use:
8 //   - 0/1: select subset with capacity limit
9 //   - complete: unlimited copies of each item
10 //   - multiple: each item has count c[i] (convert to 0/1 by
11 //       splitting counts)
12
13 long long zeroOneKnapsack(const vector<int>& w, const vector<long
14 long>& v, int W){
15     vector<long long> dp(W+1, 0);
16     int n = (int)w.size();
17     for(int i=0;i<n;++i){
18         for(int c=W;c>=w[i];--c)
19             dp[c] = max(dp[c], dp[c-w[i]] + v[i]);
20     }
21     return *max_element(dp.begin(), dp.end());
22 }
23
24 long long completeKnapsack(const vector<int>& w, const vector<long
25 long>& v, int W){
26     vector<long long> dp(W+1, 0);
27     int n = (int)w.size();
28     for(int i=0;i<n;++i){
29         for(int c=w[i]; c<=W; ++c)
30             dp[c] = max(dp[c], dp[c-w[i]] + v[i]);
31     }
32     return *max_element(dp.begin(), dp.end());
33 }
34
35 long long multipleKnapsack(vector<int> w, vector<long long> v, const
36 vector<int>& cnt, int W){
37     // Binary splitting > 0/1 items
38     vector<int> ww; vector<long long> vv;
39     int n = (int)w.size();
40     for(int i=0;i<n;++i){
41         int c = cnt[i], k = 1;
42         while(c>0){
43             int take = min(k, c);
44             ww.push_back(w[i]*take);
45             vv.push_back(v[i]*take);
46             c -= take; k <= 1;
47         }
48     }
49     return zeroOneKnapsack(ww, vv, W);
50 }
```

7.5 LIS 最長遞增子序列

```

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

```

6
7 int LISLen(const vector<long long>& a){
8     vector<long long> tail; // tail[k] = minimum possible tail of an
9     LIS with length k+1
10    for(long long x : a){
11        auto it = lower_bound(tail.begin(), tail.end(), x);
12        if(it == tail.end()) tail.push_back(x);
13        else *it = x;
14    }
15    return (int)tail.size();
}

```

7.6 LCS 最長公共子序列

```

1 // LCS (Longest Common Subsequence) – O(n*m)
2 // Usage:
3 //   int len = LCSLen(s, t);
4 // Notes: Uses O(min(n,m)) memory if desired; this is a simple O(n*m)
5 // DP.
6
7 int LCSLen(const string& s, const string& t){
8     int n = (int)s.size(), m = (int)t.size();
9     vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
10    for(int i=1; i<=n; ++i){
11        for(int j=1; j<=m; ++j){
12            if(s[i-1]==t[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
13            else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
14        }
15    }
16    return dp[n][m];
}

```

7.7 編輯距離

```

1 // Edit Distance (Levenshtein) – O(n*m)
2 // Usage:
3 //   int d = editDistance(a, b, 1, 1, 1); // costs: ins, del, sub
4
5 int editDistance(const string& a, const string& b, int ins=1, int
6     del=1, int sub=1){
7     int n = (int)a.size(), m = (int)b.size();
8     vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
9     for(int i=0; i<=n; ++i) dp[i][0] = i*del;
10    for(int j=0; j<=m; ++j) dp[0][j] = j*ins;
11    for(int i=1; i<=n; ++i){
12        for(int j=1; j<=m; ++j){
13            if(a[i-1]==b[j-1]) dp[i][j] = dp[i-1][j-1];
14            else dp[i][j] = min({dp[i-1][j] + del, dp[i][j-1] + ins,
15                dp[i-1][j-1] + sub});
16        }
17    }
18    return dp[n][m];
}

```

7.8 TSP 狀壓

```

1 // TSP (Bitmask DP) – O(n^2 * 2^n)
2 // Finds min Hamiltonian cycle cost (start at 0) given distance
3 // matrix.
4 // Usage:
5 //   vector<vector<long long>> dist(n, vector<long long>(n, INF));
6 //   long long ans = tsp_bitmask(dist);
7
8 const long long INF = (long long)4e18;
9
10 long long tsp_bitmask(const vector<vector<long long>>& dist){
11     int n = (int)dist.size();
12     int S = 1<<n;
13     vector<vector<long long>> dp(S, vector<long long>(n, INF));
14     dp[1][0] = 0; // start at 0
15     for(int mask=1; mask<S; ++mask){
16         for(int u=0; u<n; ++u){

```

```

16         if(!(mask & (1<<u))) || dp[mask][u] >= INF) continue;
17         for(int v=0; v<n; ++v){
18             if(mask & (1<<v)) continue;
19             int nmask = mask | (1<<v);
20             dp[nmask][v] = min(dp[nmask][v], dp[mask][u] +
21                 dist[u][v]);
22         }
23     }
24     long long ans = INF;
25     for(int u=0; u<n; ++u){
26         if(dp[S-1][u] < INF && dist[u][0] < INF)
27             ans = min(ans, dp[S-1][u] + dist[u][0]);
28     }
29     return ans;
}

```

7.9 區間 DP (矩陣鏈乘)

```

1 // Interval DP – Matrix Chain Multiplication
2 // Given dimensions p[0..n], matrices are (p[i-1] x p[i]).
3 // Minimize multiplication cost. Complexity: O(n^3).
4 // Usage:
5 //   long long cost = matrixChainMinCost(p);
6
7 const long long INF = (long long)4e18;
8
9 long long matrixChainMinCost(const vector<int>& p){
10    int n = (int)p.size()-1;
11    vector<vector<long long>> dp(n+1, vector<long long>(n+1, 0));
12    for(int len=2; len<=n; ++len){
13        for(int l=1; l+l+len-1<=n; ++l){
14            int r = l+len-1;
15            dp[l][r] = INF;
16            for(int k=l; k<r; ++k){
17                long long cost = dp[l][k] + dp[k+1][r] +
18                    1LL*p[l-1]*p[k]*p[r];
19                dp[l][r] = min(dp[l][r], cost);
20            }
21        }
22    }
23    return dp[1][n];
}

```

7.10 Li Chao Tree (斜率優化)

```

1 // Li Chao Segment Tree (Min Query)
2 // Maintains lines y = m * x + b over integer x in [X_MIN, X_MAX].
3 // add_line(m, b), query(x) returns min value at x. Complexity: O(log
4 // range).
5 // When to use:
6 //   - DP of form dp[i] = min_j (m_j * x_i + b_j) without monotonic
7 //     slope/query assumptions.
8
9 struct LiChao {
10     struct Line { long long m, b; long long get(long long x) const {
11         return m*x + b; } };
12     struct Node { Line ln; Node *l=nullptr, *r=nullptr; Node(Line v):
13         ln(v) };
14     long long X_MIN, X_MAX; Node* root;
15     static constexpr long long INF = (long long)4e18;
16
17     LiChao(long long xmin, long long xmax): X_MIN(xmin), X_MAX(xmax),
18           root(nullptr) {}
19
20     void add_line(Line nw){ add_line(root, X_MIN, X_MAX, nw); }
21     void add_line(long long m, long long b){ add_line({m,b}); }
22
23     void add_line(Node*& p, long long l, long long r, Line nw){
24         if(!p){ p = new Node(nw); return; }
25         long long mid = (l + r) >> 1;
26         bool lef = nw.get(l) < p->ln.get(l);
27         bool mdf = nw.get(mid) < p->ln.get(mid);
28         if(mdf) swap(nw, p->ln);

```

```

24     if(l == r) return;
25     if(lef != mdf) add_line(p->l, l, mid, nw);
26     else add_line(p->r, mid+1, r, nw);
27 }
28
29 long long query(long long x) const { return query(root, X_MIN,
30   X_MAX, x); }
31 long long query(Node* p, long long l, long long r, long long x)
32   const{
33     if(!p) return INF;
34     long long res = p->ln.get(x);
35     if(l == r) return res;
36     long long mid = (l + r) >> 1;
37     if(x <= mid) return min(res, query(p->l, l, mid, x));
38     else return min(res, query(p->r, mid+1, r, x));
39 }
40

```

7.11 單調隊列優化

```

1 // Monotonic Queue Optimization (Sliding window DP)
2 // Pattern:
3 // dp[i] = cost[i] + min_{j in [i-W, i-1]} dp[j]
4 // Maintain deque of candidate indices with increasing dp[j],
5 // dropping out-of-window.
6 // Complexity: O(n)
7 // When to use:
8 // - Transition takes min/max over a fixed-width window of previous
9 //   states.
10
11 long long solve_with_window(const vector<long long>& cost, int W){
12   int n = (int)cost.size();
13   vector<long long> dp(n, (long long)4e18);
14   deque<int> dq; // indices of candidates, dp[dq] increasing
15   for(int i=0;i<n;++i){
16     // remove out-of-window (keep j in [i-W, i-1])
17     while(!dq.empty() && dq.front() < i - W) dq.pop_front();
18     // base for i: if dq empty, no feasible j; else take min dp[j]
19     if(!dq.empty()) dp[i] = cost[i] + dp[dq.front()];
20     else if(i==0) dp[i] = cost[i]; // starting point if needed
21     // push current i into deque
22     while(!dq.empty() && dp[dq.back()] >= dp[i]) dq.pop_back();
23     dq.push_back(i);
24   }
25   return dp.back();
26 }

```

7.12 分治優化模板

```

1 // Divide & Conquer DP Optimization
2 // Problems with transitions:
3 // dp[k][i] = min_{j < i} { dp[k-1][j] + C(j, i) },
4 // where the optimal j (argmin) is monotone: opt[k][i] <=
5 // opt[k][i+1].
6 // Complexity: O(K N log N) or O(K N) depending on cost.
7 // When to use:
8 // - Cost C satisfies quadrangle inequality or structure leading to
9 //   monotone opt.
10
11 const long long INF64 = (long long)4e18;
12
13 // Example scaffolding: define your cost function using precomputed
14 // prefix sums, etc.
15 inline long long C(int j, int i){
16   // TODO: implement problem-specific cost here
17   // e.g., return S[i]-S[j] + (i-j)*X;
18   return 0;
19 }
20
21 void compute_layer(int k, int l, int r, int optL, int optR,
22   const vector<long long>& prev, vector<long long>&
23   cur){
24   if(l > r) return;
25   int mid = (l + r) >> 1;
26
27   long long val = prev[j] + C(j, mid);
28   if(val < best.first){ best = {val, j}; }
29
30   cur[mid] = best.first;
31   int opt = max(best.second, optL);
32   compute_layer(k, l, mid-1, optL, opt, prev, cur);
33   compute_layer(k, mid+1, r, opt, optR, prev, cur);
34 }
35
36 // Driver:
37 // dp[0][i] = base case
38 // for k = 1..K: compute_layer(k, 1..N)

```

```

22 pair<long long,int> best = {INF64, -1};
23 for(int j=optL; j<=min(mid-1, optR); ++j){
24   long long val = prev[j] + C(j, mid);
25   if(val < best.first){ best = {val, j}; }
26
27 cur[mid] = best.first;
28 int opt = max(best.second, optL);
29 compute_layer(k, l, mid-1, optL, opt, prev, cur);
30 compute_layer(k, mid+1, r, opt, optR, prev, cur);
31 }
32
33 // Driver:
34 // dp[0][i] = base case
35 // for k = 1..K: compute_layer(k, 1..N)

```

7.13 Knuth 優化模板

```

1 // Knuth Optimization – Interval DP O(n^2)
2 // Applies to DP of the form:
3 // dp[l][r] = min_{l <= k < r} { dp[l][k] + dp[k+1][r] } + w(l,r)
4 // Conditions:
5 // 1) Quadrangle inequality on w: A[a][c] + A[b][d] <= A[a][d] +
6 //    A[b][c] for a <= b <= c <= d
7 // 2) Monge array (or equivalent) opt[l][r-1] <= opt[l][r] <=
8 //    opt[l+1][r]
9 // Typical: merging stones, optimal BST, matrix chain variants.
10
11 const long long INF = (long long)4e18;
12
13 // Example: w(l,r) via prefix sums (replace with your cost)
14 inline long long w(int l, int r, const vector<long long>& pref){
15   // cost of merging [l..r]
16   return pref[r] - pref[l-1];
17 }
18
19 long long knuth_dp(const vector<long long>& a){
20   int n = (int)a.size();
21   vector<long long> pref(n+1, 0);
22   for(int i=1;i<=n;++i) pref[i]=pref[i-1]+a[i-1];
23   vector<vector<long long>> dp(n+2, vector<long long>(n+2, 0));
24   vector<vector<int>> opt(n+2, vector<int>(n+2, 0));
25   for(int i=1;i<=n;++i) opt[i][i]=i;
26   for(int len=2; len<=n; ++len){
27     for(int l=1; l+len-1<=n; ++l){
28       int r=l+len-1;
29       dp[l][r] = INF;
30       int st = opt[l][r-1], ed = opt[l+1][r];
31       if(st==0) st=l; if(ed==0) ed=r-1;
32       for(int k=st; k<=ed; ++k){
33         long long val = dp[l][k] + dp[k+1][r] + w(l, r, pref);
34         if(val < dp[l][r]){ dp[l][r]=val; opt[l][r]=k; }
35     }
36   }
37 }
38
39 return dp[1][n];
40

```