

Contents

1 Foundations	1
1.1 PyMath	1
1.2 Java Integer	1
1.3 Java String	1
1.4 Java String builder	2
1.5 Java Math	2
2 Mathematics AND Number Theory	2
2.1 formula	2
2.2 extended gcd	2
3 Data Structure	2
3.1 MST	2
3.2 SegmentTree	2
3.3 HLD	3
3.4 PST	3
3.5 Trie	4
3.6 BIT ver1	4
3.7 BIT ver2	4
3.8 BIT ver3	4
4 Graph	5
4.1 cut vertex AND bridges	5
4.2 SCC - Tarjan	5
4.3 BCC - Tarjan	5
4.4 Convex	5
4.5 Max Flow	6
4.6 min cut max flow	6
5 String	6
5.1 KMP	6
5.2 ACAM	7
6 Techniques	7
6.1 二分搜	7
6.2 倍增 LCA	7
6.3 SG	7
7 DP	7
7.1 輪廓線 DP	7
7.2 數位 DP	7
7.3 樹 DP	7

1 Foundations

1.1 PyMath

```

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

```

1.2 Java Integer

```

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

```

```

41 static int remainderUnsigned(int dividend,
      int divisor)

```

1.3 Java String

```

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

```

1.4 Java String builder

```

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

```

1.5 Java Math

```

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

```

```

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

```

```

1 struct Edge{
2     int u, v, w ;
3     // 這是最大生成樹，最小生成樹要改成 w < o.w
4     bool operator>(const Edge &o) const
5     {return w > o.w ;}
6
7     int par[N] ;
8     int sz[N] ;
9     int sum ;
10
11    vector<Edge> edge ;
12
13    void init(){
14        edge.clear();
15        for ( int i=0 ; i<N ; i++ ){
16            par[i] = i ;
17            sz[i] = 1 ;
18        }
19        sum = 0 ;
20    }
21
22    int find(int x){
23        if(x == par[x]) return x ;
24        return par[x] = find(par[x]) ;
25    }
26
27    int merge(int x, int y){
28        x = find(x) ;
29        y = find(y) ;
30
31        if(x == y) return 0 ;
32        if(sz[x] > sz[y]) swap(x, y) ;
33        par[x] = y ;
34        sz[y] += sz[x] ;
35
36        return 1 ;
37    }
38
39    void MST(){
40        int cnt = 0 ;
41        for ( int i=0 ; i<edge.size() && cnt < n-1
   ; 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    }

```

2 Mathematics AND Number Theory

2.1 formula

1. Catalan Number

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

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

2. Euler's Formula

對於 v 個點, e 條邊, f 個面, c 個連通分量

$$V + F = E + 2$$

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

3. Pick's Theorem

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

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

2.2 extended gcd

給定 a, b, c , 求 $ax + by = c$ 的解

```

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

```

3 Data Structure

3.1 MST

3.2 SegmentTree

```

1 #define lc (id << 1)
2 #define rc ((id << 1) | 1)
3
4 struct LazyTag{
5     // type 0 : increase val
6     // type 1 : set to val
7     // type 1 can overwrite type 0
8     int type ;
9     ll val ;
10}
11
12 struct Node{

```

```

13     LazyTag tag ;
14     ll sum ;
15     int sz ;
16 }seg[Maxn << 2] ;
17
18 class SegmentTree{
19 private:
20     void pull(int id){
21         seg[id].sum = seg[lc].sum +
22             seg[rc].sum ;
23     }
24
25     void AddTag(int id, LazyTag &tag){
26         if(tag.type == 0){
27             seg[id].sum += tag.val *
28                 seg[id].sz ;
29             seg[id].tag.val += tag.val ;
30         }
31         else{
32             seg[id].sum = tag.val *
33                 seg[id].sz ;
34             seg[id].tag = {1, tag.val} ;
35         }
36
37     void push(int id){
38         AddTag(lc, seg[id].tag) ;
39         AddTag(rc, seg[id].tag) ;
40         seg[id].tag = {0, 0} ;
41     }
42
43 public:
44     void build(int L=1, int R=n, int id=1){
45         seg[id].sum = 0 ;
46         seg[id].tag = {0, 0} ;
47         seg[id].sz = 1 ;
48
49         if(L == R){
50             seg[id].sum = arr[L] ;
51             return ;
52         }
53
54         int M = (L + R) >> 1 ;
55         build(L, M, lc) ;
56         build(M+1, R, rc) ;
57
58         pull(id) ;
59         seg[id].sz = seg[lc].sz + seg[rc].sz ;
60     }
61
62     void modify(int l, int r, LazyTag &tag,
63         int L=1, int R=n, int id=1){
64         if(l <= L && R <= r){
65             AddTag(id, tag) ;
66             return ;
67         }
68
69         push(id) ;
70         int M = (L + R) >> 1 ;
71         if(r <= M) modify(l, r, tag, L, M,
72             lc) ;
73         else if(l > M) modify(l, r, tag, M+1,
74             R, rc) ;
75         else{
76             modify(l, r, tag, L, M, lc) ;
77             modify(l, r, tag, M+1, R, rc) ;
78         }
79         pull(id) ;
80
81     ll query(int l, int r, int L=1, int R=n,
82         int id=1){
83         if(l <= L && R <= r) return
84             seg[id].sum ;
85
86         push(id) ;
87         int M = (L + R) >> 1 ;

```

3.3 HLD

```

1 /* HLD */
2 int fa[Maxn], top[Maxn], son[Maxn],
3     sz[Maxn], dep[Maxn] = {0}, dfn[Maxn],
4     rk[Maxn], dfscnt = 0 ;
5
6 void dfs1(int u, int from){
7     fa[u] = from ;
8     dep[u] = dep[from] + 1 ;
9     sz[u] = 1 ;
10
11    for ( auto v : g[u] ) if(v != from){
12        dfs1(v, u) ;
13        sz[u] += sz[v] ;
14        if(son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v ;
15    }
16
17    void dfs2(int u, int t){
18        top[u] = t ;
19        dfn[u] = ++dfscnt ;
20        rk[dfscnt] = u ;
21
22        if(son[u] == -1) return ;
23
24        dfs2(son[u], t) ;
25
26        for ( auto v : g[u] ) if(v != fa[u] && v
27            != son[u]){
28            dfs2(v, v) ;
29        }
30
31     /* Segment Tree */
32     #define lc (id << 1)
33     #define rc (((id << 1) | 1)
34
35     struct ColorSeg{
36         int left, right, tot ;
37
38         ColorSeg operator+(const ColorSeg &o)
39             const {
40             if(tot == 0) return o ;
41             if(o.tot == 0) return *this ;
42
43             ColorSeg tmp ;
44             tmp.left = left ;
45             tmp.right = o.right ;
46             tmp.tot = tot + o.tot - (right ==
47                 o.left) ;
48
49             return tmp ;
50         }
51
52         struct Node{
53             ColorSeg color ;
54             int tag ;
55         }seg[Maxn << 2] ;
56
57         class SegmentTree{
58             private:
59                 void pull(int id){
60                     // normal pull
61                 }
62
63                 if(r <= M) return query(l, r, L, M,
64                     lc) ;
65                 else if(l > M) return query(l, r,
66                     M+1, R, rc) ;
67                 else return query(l, r, L, M, lc) +
68                     query(l, r, M+1, R, rc) ;
69             }
70         }
71     }
72     }
73
74     void AddTag(int id, int tag){
75         // normal AddTag
76     }
77
78     void push(int id){
79         // normal push
80     }
81
82     void modify(int l, int r, int tag, int
83         L=1, int R=n, int id=1){
84         // normal modify
85     }
86
87     ColorSeg query(int l, int r, int L=1, int
88         R=n, int id=1){
89         // normal query
90     }
91
92     public:
93         void build(int L=1, int R=n, int id=1){
94             // normal build
95         }
96
97         // update val from u to v (simple path)
98         void update(int u, int v, int val){
99             while(top[u] != top[v]){
100                 if(dep[top[u]] < dep[top[v]]) swap(u,
101                     v) ;
102                 modify(dfn[top[u]], dfn[u], val) ;
103                 u = fa[top[u]] ;
104             }
105
106             if(dep[u] < dep[v]) swap(u, v) ;
107             modify(dfn[v], dfn[u], val) ;
108         }
109
110         // get sum from u to v (simple path)
111         int get(int u, int v){
112             pair<int, ColorSeg> U, V ;
113             ColorSeg M ;
114             U = {u, {0, 0, 0}} ;
115             V = {v, {0, 0, 0}} ;
116
117             while(top[U.first] != top[V.first]){
118                 if(dep[top[U.first]] <
119                     dep[top[V.first]]) swap(U, V) ;
120                 U.second = query(dfn[top[U.first]],
121                     dfn[U.first]) + U.second ;
122                 U.first = fa[top[U.first]] ;
123             }
124
125             if(dep[U.first] < dep[V.first]) swap(U,
126                 V) ;
127
128             M = query(dfn[V.first], dfn[U.first]) ;
129
130             return (U.second.tot + V.second.tot +
131                 M.tot) - (U.second.left == M.right)
132                 - (V.second.left == M.left) ;
133         }
134
135         void init(){
136             memset(son, -1, sizeof(son)) ;
137         }
138
139     }
140
141     struct Node{
142         int sum, left, right ;
143     }seg[Maxn + 20 * Maxn] ;
144
145     class PersistentSegmentTree{
146         private:
147             int n ;
148             int cnt ;
149             vector<int> version ;
150     }
151
152     // Find range k-th largest number
153     struct Node{
154         int sum, left, right ;
155     }seg[Maxn + 20 * Maxn] ;
156
157     class PersistentSegmentTree{
158         private:
159             int n ;
160             int cnt ;
161             vector<int> version ;
162     }
163
164     void addTag(int id, int tag){
165         // normal addTag
166     }
167
168     void push(int id){
169         // normal push
170     }
171
172     void modify(int l, int r, int tag, int
173         L=1, int R=n, int id=1){
174         // normal modify
175     }
176
177     ColorSeg query(int l, int r, int L=1, int
178         R=n, int id=1){
179         // normal query
180     }
181
182     public:
183         void build(int L=1, int R=n, int id=1){
184             // normal build
185         }
186
187         // update val from u to v (simple path)
188         void update(int u, int v, int val){
189             while(top[u] != top[v]){
190                 if(dep[top[u]] < dep[top[v]]) swap(u,
191                     v) ;
192                 modify(dfn[top[u]], dfn[u], val) ;
193                 u = fa[top[u]] ;
194             }
195
196             if(dep[u] < dep[v]) swap(u, v) ;
197             modify(dfn[v], dfn[u], val) ;
198         }
199
200         // get sum from u to v (simple path)
201         int get(int u, int v){
202             pair<int, ColorSeg> U, V ;
203             ColorSeg M ;
204             U = {u, {0, 0, 0}} ;
205             V = {v, {0, 0, 0}} ;
206
207             while(top[U.first] != top[V.first]){
208                 if(dep[top[U.first]] <
209                     dep[top[V.first]]) swap(U, V) ;
210                 U.second = query(dfn[top[U.first]],
211                     dfn[U.first]) + U.second ;
212                 U.first = fa[top[U.first]] ;
213             }
214
215             if(dep[U.first] < dep[V.first]) swap(U,
216                 V) ;
217
218             M = query(dfn[V.first], dfn[U.first]) ;
219
220             return (U.second.tot + V.second.tot +
221                 M.tot) - (U.second.left == M.right)
222                 - (V.second.left == M.left) ;
223         }
224
225         void init(){
226             memset(son, -1, sizeof(son)) ;
227         }
228
229     }
230
231     struct Node{
232         int sum, left, right ;
233     }seg[Maxn + 20 * Maxn] ;
234
235     class PersistentSegmentTree{
236         private:
237             int n ;
238             int cnt ;
239             vector<int> version ;
240     }
241
242     // Find range k-th largest number
243     struct Node{
244         int sum, left, right ;
245     }seg[Maxn + 20 * Maxn] ;
246
247     class PersistentSegmentTree{
248         private:
249             int n ;
250             int cnt ;
251             vector<int> version ;
252     }
253
254     void addTag(int id, int tag){
255         // normal addTag
256     }
257
258     void push(int id){
259         // normal push
260     }
261
262     void modify(int l, int r, int tag, int
263         L=1, int R=n, int id=1){
264         // normal modify
265     }
266
267     ColorSeg query(int l, int r, int L=1, int
268         R=n, int id=1){
269         // normal query
270     }
271
272     public:
273         void build(int L=1, int R=n, int id=1){
274             // normal build
275         }
276
277         // update val from u to v (simple path)
278         void update(int u, int v, int val){
279             while(top[u] != top[v]){
280                 if(dep[top[u]] < dep[top[v]]) swap(u,
281                     v) ;
282                 modify(dfn[top[u]], dfn[u], val) ;
283                 u = fa[top[u]] ;
284             }
285
286             if(dep[u] < dep[v]) swap(u, v) ;
287             modify(dfn[v], dfn[u], val) ;
288         }
289
290         // get sum from u to v (simple path)
291         int get(int u, int v){
292             pair<int, ColorSeg> U, V ;
293             ColorSeg M ;
294             U = {u, {0, 0, 0}} ;
295             V = {v, {0, 0, 0}} ;
296
297             while(top[U.first] != top[V.first]){
298                 if(dep[top[U.first]] <
299                     dep[top[V.first]]) swap(U, V) ;
300                 U.second = query(dfn[top[U.first]],
301                     dfn[U.first]) + U.second ;
302                 U.first = fa[top[U.first]] ;
303             }
304
305             if(dep[U.first] < dep[V.first]) swap(U,
306                 V) ;
307
308             M = query(dfn[V.first], dfn[U.first]) ;
309
310             return (U.second.tot + V.second.tot +
311                 M.tot) - (U.second.left == M.right)
312                 - (V.second.left == M.left) ;
313         }
314
315         void init(){
316             memset(son, -1, sizeof(son)) ;
317         }
318
319     }
320
321     struct Node{
322         int sum, left, right ;
323     }seg[Maxn + 20 * Maxn] ;
324
325     class PersistentSegmentTree{
326         private:
327             int n ;
328             int cnt ;
329             vector<int> version ;
330     }
331
332     // Find range k-th largest number
333     struct Node{
334         int sum, left, right ;
335     }seg[Maxn + 20 * Maxn] ;
336
337     class PersistentSegmentTree{
338         private:
339             int n ;
340             int cnt ;
341             vector<int> version ;
342     }
343
344     void addTag(int id, int tag){
345         // normal addTag
346     }
347
348     void push(int id){
349         // normal push
350     }
351
352     void modify(int l, int r, int tag, int
353         L=1, int R=n, int id=1){
354         // normal modify
355     }
356
357     ColorSeg query(int l, int r, int L=1, int
358         R=n, int id=1){
359         // normal query
360     }
361
362     public:
363         void build(int L=1, int R=n, int id=1){
364             // normal build
365         }
366
367         // update val from u to v (simple path)
368         void update(int u, int v, int val){
369             while(top[u] != top[v]){
370                 if(dep[top[u]] < dep[top[v]]) swap(u,
371                     v) ;
372                 modify(dfn[top[u]], dfn[u], val) ;
373                 u = fa[top[u]] ;
374             }
375
376             if(dep[u] < dep[v]) swap(u, v) ;
377             modify(dfn[v], dfn[u], val) ;
378         }
379
380         // get sum from u to v (simple path)
381         int get(int u, int v){
382             pair<int, ColorSeg> U, V ;
383             ColorSeg M ;
384             U = {u, {0, 0, 0}} ;
385             V = {v, {0, 0, 0}} ;
386
387             while(top[U.first] != top[V.first]){
388                 if(dep[top[U.first]] <
389                     dep[top[V.first]]) swap(U, V) ;
390                 U.second = query(dfn[top[U.first]],
391                     dfn[U.first]) + U.second ;
392                 U.first = fa[top[U.first]] ;
393             }
394
395             if(dep[U.first] < dep[V.first]) swap(U,
396                 V) ;
397
398             M = query(dfn[V.first], dfn[U.first]) ;
399
400             return (U.second.tot + V.second.tot +
401                 M.tot) - (U.second.left == M.right)
402                 - (V.second.left == M.left) ;
403         }
404
405         void init(){
406             memset(son, -1, sizeof(son)) ;
407         }
408
409     }
410
411     struct Node{
412         int sum, left, right ;
413     }seg[Maxn + 20 * Maxn] ;
414
415     class PersistentSegmentTree{
416         private:
417             int n ;
418             int cnt ;
419             vector<int> version ;
420     }
421
422     // Find range k-th largest number
423     struct Node{
424         int sum, left, right ;
425     }seg[Maxn + 20 * Maxn] ;
426
427     class PersistentSegmentTree{
428         private:
429             int n ;
430             int cnt ;
431             vector<int> version ;
432     }
433
434     void addTag(int id, int tag){
435         // normal addTag
436     }
437
438     void push(int id){
439         // normal push
440     }
441
442     void modify(int l, int r, int tag, int
443         L=1, int R=n, int id=1){
444         // normal modify
445     }
446
447     ColorSeg query(int l, int r, int L=1, int
448         R=n, int id=1){
449         // normal query
450     }
451
452     public:
453         void build(int L=1, int R=n, int id=1){
454             // normal build
455         }
456
457         // update val from u to v (simple path)
458         void update(int u, int v, int val){
459             while(top[u] != top[v]){
460                 if(dep[top[u]] < dep[top[v]]) swap(u,
461                     v) ;
462                 modify(dfn[top[u]], dfn[u], val) ;
463                 u = fa[top[u]] ;
464             }
465
466             if(dep[u] < dep[v]) swap(u, v) ;
467             modify(dfn[v], dfn[u], val) ;
468         }
469
470         // get sum from u to v (simple path)
471         int get(int u, int v){
472             pair<int, ColorSeg> U, V ;
473             ColorSeg M ;
474             U = {u, {0, 0, 0}} ;
475             V = {v, {0, 0, 0}} ;
476
477             while(top[U.first] != top[V.first]){
478                 if(dep[top[U.first]] <
479                     dep[top[V.first]]) swap(U, V) ;
480                 U.second = query(dfn[top[U.first]],
481                     dfn[U.first]) + U.second ;
482                 U.first = fa[top[U.first]] ;
483             }
484
485             if(dep[U.first] < dep[V.first]) swap(U,
486                 V) ;
487
488             M = query(dfn[V.first], dfn[U.first]) ;
489
490             return (U.second.tot + V.second.tot +
491                 M.tot) - (U.second.left == M.right)
492                 - (V.second.left == M.left) ;
493         }
494
495         void init(){
496             memset(son, -1, sizeof(son)) ;
497         }
498
499     }
500
501     struct Node{
502         int sum, left, right ;
503     }seg[Maxn + 20 * Maxn] ;
504
505     class PersistentSegmentTree{
506         private:
507             int n ;
508             int cnt ;
509             vector<int> version ;
510     }
511
512     // Find range k-th largest number
513     struct Node{
514         int sum, left, right ;
515     }seg[Maxn + 20 * Maxn] ;
516
517     class PersistentSegmentTree{
518         private:
519             int n ;
520             int cnt ;
521             vector<int> version ;
522     }
523
524     void addTag(int id, int tag){
525         // normal addTag
526     }
527
528     void push(int id){
529         // normal push
530     }
531
532     void modify(int l, int r, int tag, int
533         L=1, int R=n, int id=1){
534         // normal modify
535     }
536
537     ColorSeg query(int l, int r, int L=1, int
538         R=n, int id=1){
539         // normal query
540     }
541
542     public:
543         void build(int L=1, int R=n, int id=1){
544             // normal build
545         }
546
547         // update val from u to v (simple path)
548         void update(int u, int v, int val){
549             while(top[u] != top[v]){
550                 if(dep[top[u]] < dep[top[v]]) swap(u,
551                     v) ;
552                 modify(dfn[top[u]], dfn[u], val) ;
553                 u = fa[top[u]] ;
554             }
555
556             if(dep[u] < dep[v]) swap(u, v) ;
557             modify(dfn[v], dfn[u], val) ;
558         }
559
560         // get sum from u to v (simple path)
561         int get(int u, int v){
562             pair<int, ColorSeg> U, V ;
563             ColorSeg M ;
564             U = {u, {0, 0, 0}} ;
565             V = {v, {0, 0, 0}} ;
566
567             while(top[U.first] != top[V.first]){
568                 if(dep[top[U.first]] <
569                     dep[top[V.first]]) swap(U, V) ;
570                 U.second = query(dfn[top[U.first]],
571                     dfn[U.first]) + U.second ;
572                 U.first = fa[top[U.first]] ;
573             }
574
575             if(dep[U.first] < dep[V.first]) swap(U,
576                 V) ;
577
578             M = query(dfn[V.first], dfn[U.first]) ;
579
580             return (U.second.tot + V.second.tot +
581                 M.tot) - (U.second.left == M.right)
582                 - (V.second.left == M.left) ;
583         }
584
585         void init(){
586             memset(son, -1, sizeof(son)) ;
587         }
588
589     }
590
591     struct Node{
592         int sum, left, right ;
593     }seg[Maxn + 20 * Maxn] ;
594
595     class PersistentSegmentTree{
596         private:
597             int n ;
598             int cnt ;
599             vector<int> version ;
600     }
601
602     // Find range k-th largest number
603     struct Node{
604         int sum, left, right ;
605     }seg[Maxn + 20 * Maxn] ;
606
607     class PersistentSegmentTree{
608         private:
609             int n ;
610             int cnt ;
611             vector<int> version ;
612     }
613
614     void addTag(int id, int tag){
615         // normal addTag
616     }
617
618     void push(int id){
619         // normal push
620     }
621
622     void modify(int l, int r, int tag, int
623         L=1, int R=n, int id=1){
624         // normal modify
625     }
626
627     ColorSeg query(int l, int r, int L=1, int
628         R=n, int id=1){
629         // normal query
630     }
631
632     public:
633         void build(int L=1, int R=n, int id=1){
634             // normal build
635         }
636
637         // update val from u to v (simple path)
638         void update(int u, int v, int val){
639             while(top[u] != top[v]){
640                 if(dep[top[u]] < dep[top[v]]) swap(u,
641                     v) ;
642                 modify(dfn[top[u]], dfn[u], val) ;
643                 u = fa[top[u]] ;
644             }
645
646             if(dep[u] < dep[v]) swap(u, v) ;
647             modify(dfn[v], dfn[u], val) ;
648         }
649
650         // get sum from u to v (simple path)
651         int get(int u, int v){
652             pair<int, ColorSeg> U, V ;
653             ColorSeg M ;
654             U = {u, {0, 0, 0}} ;
655             V = {v, {0, 0, 0}} ;
656
657             while(top[U.first] != top[V.first]){
658                 if(dep[top[U.first]] <
659                     dep[top[V.first]]) swap(U, V) ;
660                 U.second = query(dfn[top[U.first]],
661                     dfn[U.first]) + U.second ;
662                 U.first = fa[top[U.first]] ;
663             }
664
665             if(dep[U.first] < dep[V.first]) swap(U,
666                 V) ;
667
668             M = query(dfn[V.first], dfn[U.first]) ;
669
670             return (U.second.tot + V.second.tot +
671                 M.tot) - (U.second.left == M.right)
672                 - (V.second.left == M.left) ;
673         }
674
675         void init(){
676             memset(son, -1, sizeof(son)) ;
677         }
678
679     }
680
681     struct Node{
682         int sum, left, right ;
683     }seg[Maxn + 20 * Maxn] ;
684
685     class PersistentSegmentTree{
686         private:
687             int n ;
688             int cnt ;
689             vector<int> version ;
690     }
691
692     // Find range k-th largest number
693     struct Node{
694         int sum, left, right ;
695     }seg[Maxn + 20 * Maxn] ;
696
697     class PersistentSegmentTree{
698         private:
699             int n ;
700             int cnt ;
701             vector<int> version ;
702     }
703
704     void addTag(int id, int tag){
705         // normal addTag
706     }
707
708     void push(int id){
709         // normal push
710     }
711
712     void modify(int l, int r, int tag, int
713         L=1, int R=n, int id=1){
714         // normal modify
715     }
716
717     ColorSeg query(int l, int r, int L=1, int
718         R=n, int id=1){
719         // normal query
720     }
721
722     public:
723         void build(int L=1, int R=n, int id=1){
724             // normal build
725         }
726
727         // update val from u to v (simple path)
728         void update(int u, int v, int val){
729             while(top[u] != top[v]){
730                 if(dep[top[u]] < dep[top[v]]) swap(u,
731                     v) ;
732                 modify(dfn[top[u]], dfn[u], val) ;
733                 u = fa[top[u]] ;
734             }
735
736             if(dep[u] < dep[v]) swap(u, v) ;
737             modify(dfn[v], dfn[u], val) ;
738         }
739
740         // get sum from u to v (simple path)
741         int get(int u, int v){
742             pair<int, ColorSeg> U, V ;
743             ColorSeg M ;
744             U = {u, {0, 0, 0}} ;
745             V = {v, {0, 0, 0}} ;
746
747             while(top[U.first] != top[V.first]){
748                 if(dep[top[U.first]] <
749                     dep[top[V.first]]) swap(U, V) ;
750                 U.second = query(dfn[top[U.first]],
751                     dfn[U.first]) + U.second ;
752                 U.first = fa[top[U.first]] ;
753             }
754
755             if(dep[U.first] < dep[V.first]) swap(U,
756                 V) ;
757
758             M = query(dfn[V.first], dfn[U.first]) ;
759
760             return (U.second.tot + V.second.tot +
761                 M.tot) - (U.second.left == M.right)
762                 - (V.second.left == M.left) ;
763         }
764
765         void init(){
766             memset(son, -1, sizeof(son)) ;
767         }
768
769     }
770
771     struct Node{
772         int sum, left, right ;
773     }seg[Maxn + 20 * Maxn] ;
774
775     class PersistentSegmentTree{
776         private:
777             int n ;
778             int cnt ;
779             vector<int> version ;
780     }
781
782     // Find range k-th largest number
783     struct Node{
784         int sum, left, right ;
785     }seg[Maxn + 20 * Maxn] ;
786
787     class PersistentSegmentTree{
788         private:
789             int n ;
790             int cnt ;
791             vector<int> version ;
792     }
793
794     void addTag(int id, int tag){
795         // normal addTag
796     }
797
798     void push(int id){
799         // normal push
800     }
801
802     void modify(int l, int r, int tag, int
803         L=1, int R=n, int id=1){
804         // normal modify
805     }
806
807     ColorSeg query(int l, int r, int L=1, int
808         R=n, int id=1){
809         // normal query
810     }
811
812     public:
813         void build(int L=1, int R=n, int id=1){
814             // normal build
815         }
816
817         // update val from u to v (simple path)
818         void update(int u, int v, int val){
819             while(top[u] != top[v]){
820                 if(dep[top[u]] < dep[top[v]]) swap(u,
821                     v) ;
822                 modify(dfn[top[u]], dfn[u], val) ;
823                 u = fa[top[u]] ;
824             }
825
826             if(dep[u] < dep[v]) swap(u, v) ;
827             modify(dfn[v], dfn[u], val) ;
828         }
829
830         // get sum from u to v (simple path)
831         int get(int u, int v){
832             pair<int, ColorSeg> U, V ;
833             ColorSeg M ;
834             U = {u, {0, 0, 0}} ;
835             V = {v, {0, 0, 0}} ;
836
837             while(top[U.first] != top[V.first]){
838                 if(dep[top[U.first]] <
839                     dep[top[V.first]]) swap(U, V) ;
840                 U.second = query(dfn[top[U.first]],
841                     dfn[U.first]) + U.second ;
842                 U.first = fa[top[U.first]] ;
843             }
844
845             if(dep[U.first] < dep[V.first]) swap(U,
846                 V) ;
847
848             M = query(dfn[V.first], dfn[U.first]) ;
849
850             return (U.second.tot + V.second.tot +
851                 M.tot) - (U.second.left == M.right)
852                 - (V.second.left == M.left) ;
853         }
854
855         void init(){
856             memset(son, -1, sizeof(son)) ;
857         }
858
859     }
860
861     struct Node{
862         int sum, left, right ;
863     }seg[Maxn + 20 * Maxn] ;
864
865     class PersistentSegmentTree{
866         private:
867             int n ;
868             int cnt ;
869             vector<int> version ;
870     }
871
872     // Find range k-th largest number
873     struct Node{
874         int sum, left, right ;
875     }seg[Maxn + 20 * Maxn] ;
876
877     class PersistentSegmentTree{
878         private:
879             int n ;
880             int cnt ;
881             vector<int> version ;
882     }
883
884     void addTag(int id, int tag){
885         // normal addTag
886     }
887
888     void push(int id){
889         // normal push
890     }
891
892     void modify(int l, int r, int tag, int
893         L=1, int R=n, int id=1){
894         // normal modify
895     }
896
897     ColorSeg query(int l, int r, int L=1, int
898         R=n, int id=1){
899         // normal query
900     }
901
902     public:
903         void build(int L=1, int R=n, int id=1){
904             // normal build
905         }
906
907         // update val from u to v (simple path)
908         void update(int u, int v, int val){
909             while(top[u] != top[v]){
910                 if(dep[top[u]] < dep[top[v]]) swap(u,
911                     v) ;
912                 modify(dfn[top[u]], dfn[u], val) ;
913                 u = fa[top[u]] ;
914             }
915
916             if(dep[u] < dep[v]) swap(u, v) ;
917             modify(dfn[v], dfn[u], val) ;
918         }
919
920         // get sum from u to v (simple path)
921         int get(int u, int v){
922             pair<int, ColorSeg> U, V ;
923             ColorSeg M ;
924             U = {u, {0, 0, 0}} ;
925             V = {v, {0, 0, 0}} ;
926
927             while(top[U.first] != top[V.first]){
928                 if(dep[top[U.first]] <
929                     dep[top[V.first]]) swap(U, V) ;
930                 U.second = query(dfn[top[U.first]],
931                     dfn[U.first]) + U.second ;
932                 U.first = fa[top[U.first]] ;
933             }
934
935             if(dep[U.first] < dep[V.first]) swap(U,
936                 V) ;
937
938             M = query(dfn[V.first], dfn[U.first]) ;
939
940             return (U.second.tot + V.second.tot +
941                 M.tot) - (U.second.left == M.right)
942                 - (V.second.left == M.left) ;
943         }
944
945         void init(){
946             memset(son, -1, sizeof(son)) ;
947         }
948
949     }
950
951     struct Node{
952         int sum, left, right ;
953     }seg[Maxn + 20 * Maxn] ;
954
955     class PersistentSegmentTree{
956         private:
957             int n ;
958             int cnt ;
959             vector<int> version ;
960     }
961
962     // Find range k-th largest number
963     struct Node{
964         int sum, left, right ;
965     }seg[Maxn + 20 * Maxn] ;
966
967     class PersistentSegmentTree{
968         private:
969             int n ;
970             int cnt ;
971             vector<int> version ;
972     }
973
974     void addTag(int id, int tag){
975         // normal addTag
976     }
977
978     void push(int id){
979         // normal push
980     }
981
982     void modify(int l, int r, int tag, int
983         L=1, int R=n, int id=1){
984         // normal modify
985     }
986
987     ColorSeg query(int l, int r, int L=1, int
988         R=n, int id=1){
989         // normal query
990     }
991
992     public:
993         void build(int L=1, int R=n, int id=1){
994             // normal build
995         }
996
997         // update val from u to v (simple path)
998         void update(int u, int v, int val){
999             while(top[u] != top[v]){
1000                 if(dep[top[u]] < dep[top[v]]) swap(u,
1001                     v) ;
1002                 modify(dfn[top[u]], dfn[u], val) ;
1003                 u = fa[top[u]] ;
1004             }
1005
1006             if(dep[u] < dep[v]) swap(u, v) ;
1007             modify(dfn[v], dfn[u], val) ;
1008         }
1009
1010         // get sum from u to v (simple path)
1011         int get(int u, int v){
1012             pair<int, ColorSeg> U, V ;
1013             ColorSeg M ;
1014             U = {u, {0, 0, 0}} ;
1015             V = {v, {0, 0, 0}} ;
1016
1017             while(top[U.first] != top[V.first]){
1018                 if(dep[top[U.first]] <
1019                     dep[top[V.first]]) swap(U, V) ;
1020                 U.second = query(dfn[top[U.first]],
1021                     dfn[U.first]) + U.second ;
1022                 U.first = fa[top[U.first]] ;
1023             }
1024
1025             if(dep[U.first] < dep[V.first]) swap(U,
1026                 V) ;
1027
1028             M = query(dfn[V.first], dfn[U.first]) ;
1029
1030             return (U.second.tot + V.second.tot +
1031                 M.tot) - (U.second.left == M.right)
1032                 - (V.second.left == M.left) ;
1033         }
1034
1035         void init(){
1036             memset(son, -1, sizeof(son)) ;
1037         }
1038
1039     }
1040
1041     struct Node{
1042         int sum, left, right ;
1043     }seg[Maxn + 20 * Maxn] ;
1044
1045     class PersistentSegmentTree{
1046         private:
1047             int n ;
1048             int cnt ;
1049             vector<int> version ;
1050     }
1051
1052     // Find range k-th largest number
1053     struct Node{
1054         int sum, left, right ;
1055     }seg[Maxn + 20 * Maxn] ;
1056
1057     class PersistentSegmentTree{
1058         private:
1059             int n ;
1060             int cnt ;
1061             vector<int> version ;
1062     }
1063
1064     void addTag(int id, int tag){
1065         // normal addTag
1066     }
1067
1068     void push(int id){
1069         // normal push
1070     }
1071
1072     void modify(int l, int r, int tag, int
1073         L=1, int R=n, int id=1){
1074         // normal modify
1075     }
1076
1
```

```

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
27 public:
28 PersistentSegmentTree(int _n){
29     n = _n ;
30     cnt = 0 ;
31
32     int root = build(1, n) ;
33     version.push_back(root) ;
34 }
35
36 void update(int ver, int idx){
37     auto upd = [&](auto &self, const int
38                 cur, int L, int R){
39         int cur_cnt = cnt++ ;
40
41         if(L == R){
42             seg[cur_cnt] = {seg[cur].sum + 1, 0,
43                             0} ;
44             return cur_cnt ;
45         }
46
47         int M = (L + R) >> 1 ;
48         int lc = seg[cur].left ;
49         int rc = seg[cur].right ;
50
51         if(idx <= M) lc = self(self,
52             seg[cur].left, L, M) ;
53         else rc = self(self, seg[cur].right,
54                         M+1, R) ;
55
56         seg[cur_cnt] = {seg[lc].sum +
57                         seg[rc].sum, lc, rc} ;
58
59         return cur_cnt ;
60     };
61
62     int root = upd(upd, version[ver], 1, n) ;
63     version.push_back(root) ;
64 }
65
66 int query(int verL, int verR, int k){
67     auto qry = [&](auto &self, const int
68                 cur_old, const int cur_new, int L,
69                 int R){
70         if(L == R) return L ;
71
72         int old_l = seg[cur_old].left, old_r =
73             seg[cur_old].right ;
74         int new_l = seg[cur_new].left, new_r =
75             seg[cur_new].right ;
76
77         int dl = seg[new_l].sum -
78             seg[old_l].sum ;
79         int dr = seg[new_r].sum -
80             seg[old_r].sum ;
81
82         int M = (L + R) >> 1 ;
83
84         if(dl >= k) return self(self, old_l,
85             new_l, L, M) ;
86         k -= dl ;
87         return self(self, old_r, new_r, M+1,
88                     R) ;
89     };
90
91     int idx = qry(qry, version[verL-1],
92                   version[verR], 1, n) ;
93     return idx ;
94 }

```

3.5 Trie

```

1 class Trienode{
2 public:
3     set<int> end ;
4     Trienode *next[26] ;
5
6     Trienode(){
7         for ( int i=0 ; i<26 ; i++ ) next[i]
8             = nullptr ;
9     }
10
11 class Trie{
12 private:
13     int cnt ;
14     Trienode *root ;
15 public:
16     Trie() : cnt(0) {
17         root = new Trienode() ;
18     }
19
20     void insert(string &str, int n){
21         Trienode* node = root ;
22         for ( auto s : str ){
23             int path = s - 'a' ;
24
25             if(node->next[path] == nullptr)
26                 node->next[path] = new
27                     Trienode() ;
28             node = node->next[path] ;
29         }
30         node->end.insert(n) ;
31     }
32
33     void search(string &str){
34         Trienode* node = root ;
35         for ( auto s : str ){
36             int path = s - 'a' ;
37             if(node->next[path] == nullptr)
38                 return ;
39             node = node->next[path] ;
40         }
41         int flg = 0 ;
42         for ( auto n : node->end ){
43             if(flg) cout << " " ;
44             else flg = 1 ;
45
46             cout << n ;
47         }
48
49         void clear(Trienode* node) {
50             if (!node) return ;
51             for ( int i = 0; i < 26; i++ ) {
52                 if (node->next[i]) {
53                     clear(node->next[i]) ;
54                 }
55                 delete node ;
56             }
57         }
58
59         ~Trie(){
60             clear(root) ;
61         }
62     }
63
64     ~TrieNode(){
65         clear(root) ;
66     }
67 }

```

3.6 BIT ver1

```

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

```

3.7 BIT ver2

```

1 // 區間修改，單點查詢
2 #define lowbit(x) (x & -x)
3
4 int BIT[MAX_SIZE] ;
5 int n ;
6
7 void modify(int idx, int val){
8     for ( ; idx <= n ; idx += lowbit(idx) ) {
9         BIT[idx] += val ;
10    }
11 }
12
13 // ans: query(i)
14 int query(int idx){
15     int sum = 0 ;
16     for ( ; idx ; idx -= lowbit(idx) ){
17         sum += BIT[idx] ;
18     }
19
20     return sum ;
21 }
22
23 void init(){
24     memset(BIT, 0, sizeof(BIT)) ;
25 }
26
27 void build(){
28     arr[0] = 0 ;
29     for ( int i=1 ; i<=n ; i++ ) modify(i,
30                     arr[i] - arr[i-1]) ;
31 }
32
33 // usage
34 // add val
35 modify(L, x) ;
36 modify(R+1, -x) ;

```

3.8 BIT ver3

```

1 // 區間修改 區間查詢
2 #define lowbit(x) x & -x
3
4 ll BIT1[MAXN], BIT2[MAXN] ;
5
6 void update(int idx, ll val){
7     for ( int i=idx ; i<=n ; i+=lowbit(i) ) {

```

```

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

4 Graph

4.1 cut vertex AND bridges

```

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

4.2 SCC - Tarjan

```

1 vector<int> scc[Maxn] ;
2 int dfn[Maxn], low[Maxn], sccId[Maxn],
3     dfscnt = 0, cnt_scc = 0 ;
4 stack<int> st ;
5 bitset<Maxn> inSt, vis ;
6 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 // SCC to DAG (after dfs)
33 vector<int> dag[Maxn] ;
34
35 void scc_to_dag(){
36     vector<int> dag[Maxn] ;
37     for ( int u=1 ; u<=n ; u++ ){
38         for ( auto v : g[u] ){
39             if(sccId[u] != sccId[v]){
40                 dag[sccId[u]].push_back(sccId[v]) ;
41             }
42         }
43     }
44 }
45
46 void init(){
47     memset(dfn, -1, sizeof(dfn)) ;
48     memset(low, -1, sizeof(low)) ;
49 }
50
51 int main(){
52     init() ;
53     input() ;
54     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
55         == -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],
16     vis_bcc[Maxn], bcc_cnt = 0 ;
17
18 void dfs1(int u, int from){
19     dfn[u] = low[u] = ++dfn_cnt ;
```

```

20     for ( int i=head[u] ; i!= -1 ; i=e[i].next
21         ){
22         int v = e[i].v ;
23
24         if(dfn[v] == -1){
25             dfs1(v, i) ;
26             if(dfn[u] < low[v]) bz[i] = bz[i^1] =
27                 1 ;
28             low[u] = min(low[u], low[v]) ;
29     }
30 }
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         ){
38         int v = e[i].v ;
39
40         if(vis_bcc[v] != -1 || bz[i]) continue ;
41         dfs2(v, id) ;
42     }
43 }
44
45 void init(){
46     memset(dfn, -1, sizeof(dfn)) ;
47     memset(head, -1, sizeof(head)) ;
48     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
49 }
50
51 int main(){
52     init() ;
53     input() ;
54     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
55         == -1){
56         dfs1(i, 0) ;
57     }
58
59     for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i]
60         == -1){
61         bcc.push_back(vector<int>()) ;
62         dfs2(i, bcc_cnt++) ;
63     }
64 }
```

4.4 Convex

```

1 struct Coordinate{
2     long long x, y ;
3
4     friend bool operator<(const Coordinate&a,
5                             const Coordinate&b){
6         if(a.x == b.x) return a.y < b.y ;
7         return a.x < b.x ;
8     }
9
10    friend bool operator==(const Coordinate&
11                           a, const Coordinate& b){
12        return a.x == b.x && a.y == b.y ;
13    }
14
15    vector<Coordinate> nodes ;
16
17    long long cross(const Coordinate&o, const
18                     Coordinate&a, const Coordinate&b){
19        return (a.x - o.x) * (b.y - o.y) - (a.y -
20                                         o.y) * (b.x - o.x) ;
21    }
22
23    void input(){
24        nodes.clear() ;
25    }
26}
```

```

int n, x, y ;
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

```

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 ;
}

```

```

37 int dfs(int u, int flow){
38     if(u == T) return flow ;
39     int d, rest = 0 ;
40
41     for ( int &i=cur[u] ; i!=-1 ;
42         i=e[i].next ){
43         int v = e[i].v ;
44         if(dep[v] == dep[u] + 1 && e[i].cap >
45             0){
46             d = dfs(v, min(flow - rest,
47                         e[i].cap)) ;
48
49             if(d > 0){
50                 e[i].cap -= d ;
51                 e[i^1].cap += d ;
52                 rest += d ;
53
54                 if(rest == flow) break ;
55             }
56
57             if(rest != flow) dep[u] = -1 ;
58             return rest ;
59         }
60     public:
61     MaxFlow(int n, int s, int t){
62         N = n ; S = s ; T = t ;
63         e.reserve(n*n) ;
64         head.assign(n+1, -1) ;
65         cur.resize(n+1) ;
66         dep.resize(n+1) ;
67     }
68     void AddEdge(int u, int v, int cap){
69         e.push_back({v, cap, head[u]}) ;
70         head[u] = e.size() - 1 ;
71         e.push_back({u, 0, head[v]}) ;
72         head[v] = e.size() - 1 ;
73     }
74
75     int run(){
76         int ans = 0 ;
77         while(bfs()){
78             ans += dfs(S, 0x3f3f3f3f) ;
79         }
80         return ans ;
81     }
82 };

```

4.6 min cut max flow

```

1 struct Edge{
2     int v, cap, cost , next ;
3 };
4
5 using pii = pair<int, int> ;
6 class MCMF{
7 private:
8     int N, s, t, tot ;
9     vector<Edge> e ;
10    vector<int> head ;
11 public:
12    MCMF(int n, int _s, int _t){
13        N = n ;
14        s = _s ;
15        t = _t ;
16        e.resize(n*n + 5) ;
17        head.assign(n+5, -1) ;
18        tot = -1 ;
19    }
20
21    void AddEdge(int u, int v, int cap, int
22                  cost){
23        e[++tot] = {v, cap, cost, head[u]} ;
24        head[u] = tot ;
25        e[++tot] = {u, 0, -cost, head[v]} ;
26    }
27
28    int run(){
29        vector<int> dis(N+1), pot(N+1, 0),
30                  preE(N+1) ;
31        int flow = 0, cost = 0 ;
32
33        auto dijkstra = [&](){
34            fill(dis.begin(), dis.end(), INF) ;
35            priority_queue<pii, vector<pii>,
36                          greater<pii>> pq ;
37            dis[s] = 0 ;
38            pq.push({0, s}) ;
39
40            while(!pq.empty()){
41                auto [d, u] = pq.top() ; pq.pop() ;
42                if(d > dis[u]) continue ;
43                for ( int i=head[u] ; i!=-1 ;
44                      i=e[i].next ){
45                    int v = e[i].v, cap = e[i].cap, w =
46                        e[i].cost ;
47                    if(cap && dis[v] > d + w + pot[u] -
48                        pot[v]){
49                        dis[v] = d + w + pot[u] - pot[v] ;
50                        preE[v] = i ;
51                        pq.push({dis[v], v}) ;
52                    }
53                }
54            }
55
56            return dis[t] != INF ;
57        };
58
59        while(dijkstra()){
60            for ( int v=1 ; v<=N ; v++ ) if(dis[v]
61                < INF){
62                pot[v] += dis[v] ;
63            }
64
65            int aug = INT_MAX ;
66            for ( int v=t ; v!=s ;
67                  v=e[preE[v]^1].v ){
68                aug = min(aug, e[preE[v]].cap) ;
69            }
70
71            for ( int v=t ; v!=s ;
72                  v=e[preE[v]^1].v ){
73                e[preE[v]].cap -= aug ;
74                e[preE[v]^1].cap += aug ;
75                cost += aug * e[preE[v]].cost ;
76            }
77
78            return cost ;
79        };
80    };

```

5 String

5.1 KMP

```

1 int Next[N] ;
2 void kmp(string &str){
3     Next[0] = -1 ;
4     if(str.size() <= 1) return ;
5     Next[1] = 0 ;
6
7     int cur = 2, check = 0 ;
8
9     while(cur < str.size()){
10        if(str[cur - 1] == str[check])
11            Next[cur++] = ++check ;
12        else if(check > 0) check =
13            Next[check] ;
14        else Next[cur++] = 0 ;
15    }
16}

```

4.6 min cut max flow

```
1 struct Edge{
2     int v, cap, cost , next ;
3 };
4
5 using pii = pair<int, int> ;
6 class MCMF{
7 private:
8     int N, s, t, tot ;
9     vector<Edge> e ;
10    vector<int> head ;
11 public:
12    MCMF(int n, int _s, int _t){
13        N = n ;
14        s = _s ;
15        t = _t ;
16        e.resize(n*n + 5) ;
17        head.assign(n+5, -1) ;
18        tot = -1 ;
19    }
20
21    void AddEdge(int u, int v, int cap, int
22                  cost){
23        e[++tot] = {v, cap, cost, head[u]} ;
24        head[u] = tot ;
25        e[++tot] = {u, 0, -cost, head[v]} ;
```

5 String

5.1 KMF

```
1 int Next[N] ;
2 void kmp(string &str){
3     Next[0] = -1 ;
4     if(str.size() <= 1) return ;
5     Next[1] = 0 ;
6
7     int cur = 2, check = 0 ;
8
9     while(cur < str.size()){
10         if(str[cur - 1] == str[check])
11             Next[cur++] = ++check ;
12         else if(check > 0) check =
13             Next[check] ;
14         else Next[cur++] = 0 ;
15     }
16 }
```

```

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 ACAutomation{
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
15 public:
16     ACAutomation(int _base='a', int _alpha=26)
17     : base(_base), alpha(_alpha) {
18         clear() ;
19     }
20
21     void clear(){
22         fail.assign(1, 0) ;
23         order.clear() ;
24         end.clear() ;
25         tree.assign(1, vector<int>(alpha, 0)) ;
26     }
27
28     void add_pattern(const string &pattern){
29         int u = 0 ;
30         for ( auto ch : pattern ){
31             int v = ch - base ;
32
33             if(tree[u][v] == 0) tree[u][v] =
34                 new_node() ;
35             u = tree[u][v] ;
36
37         end.push_back(u) ;
38     }
39
40     void build(){
41         queue<int> q ;
42         order.clear() ;
43         order.push_back(0) ;

```

```

44     for ( int i=0 ; i<alpha ; i++ )
45         if(tree[0][i] > 0){
46             q.push(tree[0][i]) ;
47         }
48
49         while(!q.empty()){
50             int u = q.front() ; q.pop() ;
51             order.push_back(u) ;
52
53             for ( int i=0 ; i<alpha ; i++ ){
54                 if(tree[u][i] == 0) tree[u][i] =
55                     tree[fail[u]][i] ;
56                 else{
57                     fail[tree[u][i]] = tree[fail[u]][i]
58                     ;
59                     q.push(tree[u][i]) ;
60                 }
61             }
62
63             vector<int> count_per_pattern(const string
64                                         &text) const {
65                 int u = 0 ;
66                 vector<int> vis(tree.size(), 0) ;
67
68                 for ( char ch : text ){
69                     u = tree[u][ch - base] ;
70                     vis[u]++;
71                 }
72
73                 for ( int i=order.size()-1 ; i>=1 ; i-- )
74                     int x = order[i] ;
75                     vis[fail[x]] += vis[x] ;
76
77                 vector<int> ans(end.size(), 0) ;
78                 for ( int id=0 ; id<end.size() ; id++ ){
79                     ans[id] = vis[end[id]] ;
80                 }
81
82                 return ans ;
83             };

```

6 Techniques

6.1 二分搜

```

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

```

6.2 倍增 LCA

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 const int Maxn = 500005 ;
6
7 vector<int> e[Maxn] ;
8 int depth[Maxn] ;
9 int up[Maxn][40] ;
10 int MaxLog ;
11
12 void dfs(int u, int from, int d){
13     up[u][0] = from ;
14     depth[u] = d ;
15
16     for ( int i=1 ; i<=MaxLog ; i++ ){
17         up[u][i] = up[up[u][i-1]][i-1] ;
18     }
19
20     for ( auto v : e[u] ){
21         if(v == from) continue ;
22         dfs(v, u, d + 1) ;
23     }
24 }
25
26 int lca(int u, int v){
27     if(depth[u] < depth[v]) swap(u, v) ;
28
29     for ( int i=MaxLog ; i>=0 ; i-- )
30         if(depth[u] - (1 << i) >= depth[v]){
31             u = up[u][i] ;
32         }
33
34     if(u == v) return u ;
35
36     for ( int i=MaxLog ; i>=0 ; i-- )
37         if(up[u][i] != up[v][i]){
38             u = up[u][i] ;
39             v = up[v][i] ;
40         }
41
42     return up[u][0] ;
43 }
44
45 int main(){
46     int n, q, root ;
47     scanf("%d%d%d", &n, &q, &root) ;
48     MaxLog = __lg(n) ;
49
50     for ( int i=0 ; i<n-1 ; i++ ){
51         int u, v ;
52         scanf("%d%d", &u, &v) ;
53         e[u].push_back(v) ;
54         e[v].push_back(u) ;
55     }
56
57     dfs(root, root, 0) ;
58
59     while(q--){
60         int u, v ;
61         scanf("%d%d", &u, &v) ;
62         printf("%d\n", lca(u, v)) ;
63     }
64 }

```

6.3 SG

```

1 long long SG(long long k){
2
3     if(k % 2 == 0){
4         return k / 2;
5     }

```

```

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

7 DP

7.1 輪廓線 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ll = long long;
5
6 ll dp[2][(1 << 10) + 5];
7 int n, m;
8 int cur;
9
10 void update(int s1, int s2){
11     if(s2 & (1 << m)){
12 }
```

```

12     dp[cur][s2 ^ (1 << m)] += dp[cur] ^
13         1][s1] ;
14 }
15
16 int main(){
17     while(~scanf("%d%d", &n, &m)){
18         if(m > n) swap(n, m);
19         memset(dp, 0, sizeof(dp));
20         cur = 0;
21         dp[cur][(1 << m) - 1] = 1;
22         for ( int i=0 ; i<n ; i++ ) for ( int
23             j=0 ; j<m ; j++ ){
24             cur ^= 1;
25             memset(dp[cur], 0, sizeof(dp[cur]));
26
27             for ( int k=0 ; k<(1 << m) ; k++ ){
28                 update(k, k << 1); // not put
29                 if(i && !(k & (1 << (m - 1))) )
30                     update(k, (k << 1) | (1 << m) |
31                         1); // put up
32                 if(j && !(k & 1)) update(k, (k << 1)
33                     | 3); // put left
34             }
35             printf("%lld\n", dp[cur][(1 << m) - 1]);
36         }
37         return 0;
38     }
39 }
```

7.2 數位 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int K;
6 int dp[20][105][105][2];
7 vector<int> dig;
8
9 int solve(int pos, int sum, int dsum, bool
10 lim){
11     if(pos == -1){
12         if(sum == 0 && dsum == 0) return 1;
13         return 0;
14     }
15
16     int &d = dp[pos][sum][dsum][lim];
17     if(d != -1) return d;
18
19     int up = lim ? dig[pos] : 9;
20     int res = 0;
21     for ( int i=0 ; i<=up ; i++ ){
22         res += solve(pos-1, (sum * 10 + i) %
23             K, (dsum + i) % K, lim && i==up);
24     }
25
26     return d = res;
27 }
28
29 int count(int n){
30     memset(dp, -1, sizeof(dp));
31     dig.clear();
32
33     while(n > 0){
34         dig.push_back(n % 10);
35         n /= 10;
36     }
37
38     return solve(dig.size() - 1, 0, 0, 1);
39 }
40
41 int main(){
42     int t = 0;
43
44     while(~scanf("%d", &n) && n){
45         init();
46         for ( int i=0 ; i<n-1 ; i++ ){
47             int u, v, w;
48             scanf("%d%d%d", &v, &u, &w);
49             edge[u].push_back({v, w});
50         }
51
52         DFS(0);
53         printf("Case %d:\n", ++t);
54
55         int q, e;
56         scanf("%d", &q);
57     }
58 }
```

```

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

7.3 樹 DP

```

1 #include <bits/stdc++.h>
2
3 #define N 505
4 #define INF 0x3f3f3f3f
5
6 using namespace std;
7
8 struct Edge{
9     int v, w;
10 } ;
11
12 vector<Edge> edge[N];
13 int n;
14 int cnt[N];
15 int dp[N][N][2];
16
17 void init(){
18     for ( int i=0 ; i<N ; i++ )
19         edge[i].clear();
20     memset(cnt, 0, sizeof(cnt));
21     memset(dp, INF, sizeof(dp));
22 }
23
24 void DFS(int u){
25     cnt[u] = 1;
26     for ( auto [v, w] : edge[u] ){
27         DFS(v);
28         cnt[u] += cnt[v];
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 (
34             int j=1 ; j<i && j<=cnt[v] ; j++ ){
35             dp[u][i][1] = min(dp[u][i][1],
36                 dp[u][i-j][1] + dp[v][j][1] + 2 *
37                 w);
38             dp[u][i][0] = min(dp[u][i][0],
39                 dp[u][i-j][1] + dp[v][j][0] + w);
40             dp[u][i][0] = min(dp[u][i][0],
41                 dp[u][i-j][0] + dp[v][j][1] + 2 *
42                 w);
43         }
44     }
45
46     int main(){
47         int t = 0;
48
49         while(~scanf("%d", &n) && n){
50             init();
51             for ( int i=0 ; i<n-1 ; i++ ){
52                 int u, v, w;
53                 scanf("%d%d%d", &v, &u, &w);
54                 edge[u].push_back({v, w});
55             }
56             DFS(0);
57             printf("Case %d:\n", ++t);
58         }
59     }
60 }
```

```
58     while(q--){
59         scanf("%d", &e) ;
60
61         for ( int i=n ; i>=1 ; i-- )
62             if(dp[0][i][0] <= e){
63                 printf("%d\n", i) ;
64                 break ;
65             }
66     }
67
68     return 0 ;
69 }
```