

## Contents

1 Basic	1
1.1 PyMath	1
2 Tree	1
2.1 SegmentTree	1
2.2 HLD	1
2.3 Trie	1
3 String	2
3.1 KMP	2
4 Algorithm	3
4.1 LCA	3
4.2 MST	3
4.3 SG	3
4.4 Convex	3
4.5 Find Cut Vertex	3
4.6 SCC	4
4.7 BCC	4
5 DP	4
5.1 輪廓線 DP	4
5.2 數位 DP	5
5.3 樹 DP	5

## 1 Basic

### 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

```

## 2 Tree

### 2.1 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         AddTag(lc, seg[id].tag) ;
38         AddTag(rc, seg[id].tag) ;
39         seg[id].tag = {0, 0} ;
40     }

```

```

41 public:
42     void build(int L=1, int R=n, int id=1){
43         seg[id].sum = 0 ;
44         seg[id].tag = {0, 0} ;
45         seg[id].sz = 1 ;
46
47         if(L == R){
48             seg[id].sum = arr[L] ;
49             return ;
50         }
51
52         int M = (L + R) >> 1 ;
53         build(L, M, lc) ;
54         build(M+1, R, rc) ;
55
56         pull(id) ;
57         seg[id].sz = seg[lc].sz + seg[rc].sz ;
58     }
59
60     void modify(int l, int r, LazyTag &tag,
61                int L=1, int R=n, int id=1){
62         if(l <= L && R <= r){
63             AddTag(id, tag) ;
64             return ;
65         }
66
67         push(id) ;
68         int M = (L + R) >> 1 ;
69         if(r <= M) modify(l, r, tag, L, M,
70                           lc) ;
71         else if(l > M) modify(l, r, tag, M+1,
72                               R, rc) ;
73         else{
74             modify(l, r, tag, L, M, lc) ;
75             modify(l, r, tag, M+1, R, rc) ;
76         }
77         pull(id) ;
78     }
79
80     ll query(int l, int r, int L=1, int R=n,
81              int id=1){
82         if(l <= L && R <= r) return
83             seg[id].sum ;
84
85         push(id) ;
86         int M = (L + R) >> 1 ;
87         if(r <= M) return query(l, r, L, M,
88                               lc) ;
89         else if(l > M) return query(l, r,
90                                     M+1, R, rc) ;
91         else return query(l, r, L, M, lc) +
92             query(l, r, M+1, R, rc) ;
93     }
94 }tree ;

```

### 2.2 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    }

```

```

18 dfn[u] = ++dfscnt ;
19 rnk[dfscnt] = u ;
20
21 if(son[u] == -1) return ;
22
23 dfs2(son[u], t) ;
24
25 for ( auto v : g[u] ) if(v != fa[u] && v
26     != son[u]){
27     dfs2(v, v) ;
28 }
29
30 /* Segment Tree */
31 #define lc (id << 1)
32 #define rc ((id << 1) | 1)
33
34 struct ColorSeg{
35     int left, right, tot ;
36
37     ColorSeg operator+(const ColorSeg &o)
38         const {
39         if(tot == 0) return o ;
40         if(o.tot == 0) return *this ;
41
42         ColorSeg tmp ;
43         tmp.left = left ;
44         tmp.right = o.right ;
45         tmp.tot = tot + o.tot - (right ==
46             o.left) ;
47
48         return tmp ;
49     }
50
51     struct Node{
52         ColorSeg color ;
53         int tag ;
54     }seg[Maxn << 2] ;
55
56 class SegmentTree{
57 private:
58     void pull(int id){
59         // normal pull
60     }
61
62     void AddTag(int id, int tag){
63         // normal AddTag
64     }
65
66     void push(int id){
67         // normal push
68     }
69
70     void modify(int l, int r, int tag, int
71                 L=1, int R=n, int id=1){
72         // normal modify
73     }
74
75     ColorSeg query(int l, int r, int L=1, int
76                    R=n, int id=1){
77         // normal query
78     }
79
80     public:
81     void build(int L=1, int R=n, int id=1){
82         // normal build
83
84         // update val from u to v (simple path)
85         void update(int u, int v, int val){
86             while(top[u] != top[v]){
87                 if(dep[top[u]] < dep[top[v]]) swap(u,
88                     v) ;
89                 modify(dfn[top[u]], dfn[u], val) ;
90                 u = fa[top[u]] ;
91             }
92
93             if(dep[u] < dep[v]) swap(u, v) ;
94
95             // get sum from u to v (simple path)
96             pair<int, ColorSeg> U, V ;
97             ColorSeg M ;
98             U = {u, {0, 0, 0}} ;
99             V = {v, {0, 0, 0}} ;
100
101            while(top[U.first] != top[V.first]){
102                if(dep[top[U.first]] <
103                    dep[top[V.first]]) swap(U, V) ;
104                U.second = query(dfn[top[U.first]],
105                                  dfn[U.first]) + U.second ;
106                U.first = fa[top[U.first]] ;
107
108                if(dep[U.first] < dep[V.first]) swap(U,
109                    V) ;
110
111                M = query(dfn[V.first], dfn[U.first]) ;
112            }
113
114            void init(){
115                memset(son, -1, sizeof(son)) ;
116            }
117
118        }
119
120    }
121
122    ColorSeg operator+(const ColorSeg &o)
123        const {
124        if(tot == 0) return o ;
125        if(o.tot == 0) return *this ;
126
127        ColorSeg tmp ;
128        tmp.left = left ;
129        tmp.right = o.right ;
130        tmp.tot = tot + o.tot - (right ==
131            o.left) ;
132
133        return tmp ;
134    }
135
136    struct Node{
137        ColorSeg color ;
138        int tag ;
139    }seg[Maxn << 2] ;
140
141    class SegmentTree{
142    private:
143        void pull(int id){
144            // normal pull
145        }
146
147        void AddTag(int id, int tag){
148            // normal AddTag
149        }
150
151        void push(int id){
152            // normal push
153        }
154
155        void modify(int l, int r, int tag, int
156                    L=1, int R=n, int id=1){
157            // normal modify
158        }
159
160        ColorSeg query(int l, int r, int L=1, int
161                       R=n, int id=1){
162            // normal query
163        }
164
165        public:
166        void build(int L=1, int R=n, int id=1){
167            // normal build
168
169
170            // update val from u to v (simple path)
171            void update(int u, int v, int val){
172                while(top[u] != top[v]){
173                    if(dep[top[u]] < dep[top[v]]) swap(u,
174                        v) ;
175                    modify(dfn[top[u]], dfn[u], val) ;
176                    u = fa[top[u]] ;
177
178                    if(dep[u] < dep[v]) swap(u, v) ;
179
180                    // get sum from u to v (simple path)
181                    pair<int, ColorSeg> U, V ;
182                    ColorSeg M ;
183                    U = {u, {0, 0, 0}} ;
184                    V = {v, {0, 0, 0}} ;
185
186                    while(top[U.first] != top[V.first]){
187                        if(dep[top[U.first]] <
188                            dep[top[V.first]]) swap(U, V) ;
189                        U.second = query(dfn[top[U.first]],
190                                          dfn[U.first]) + U.second ;
191                        U.first = fa[top[U.first]] ;
192
193                        if(dep[U.first] < dep[V.first]) swap(U,
194                            V) ;
195
196                        M = query(dfn[V.first], dfn[U.first]) ;
197
198                    }
199
200                }
201
202            }
203
204        }
205
206    }
207
208    ColorSeg operator+(const ColorSeg &o)
209        const {
210        if(tot == 0) return o ;
211        if(o.tot == 0) return *this ;
212
213        ColorSeg tmp ;
214        tmp.left = left ;
215        tmp.right = o.right ;
216        tmp.tot = tot + o.tot - (right ==
217            o.left) ;
218
219        return tmp ;
220    }
221
222    struct Node{
223        ColorSeg color ;
224        int tag ;
225    }seg[Maxn << 2] ;
226
227    class SegmentTree{
228    private:
229        void pull(int id){
230            // normal pull
231        }
232
233        void AddTag(int id, int tag){
234            // normal AddTag
235        }
236
237        void push(int id){
238            // normal push
239        }
240
241        void modify(int l, int r, int tag, int
242                    L=1, int R=n, int id=1){
243            // normal modify
244        }
245
246        ColorSeg query(int l, int r, int L=1, int
247                       R=n, int id=1){
248            // normal query
249        }
250
251        public:
252        void build(int L=1, int R=n, int id=1){
253            // normal build
254
255
256            // update val from u to v (simple path)
257            void update(int u, int v, int val){
258                while(top[u] != top[v]){
259                    if(dep[top[u]] < dep[top[v]]) swap(u,
260                        v) ;
261                    modify(dfn[top[u]], dfn[u], val) ;
262                    u = fa[top[u]] ;
263
264                    if(dep[u] < dep[v]) swap(u, v) ;
265
266                    // get sum from u to v (simple path)
267                    pair<int, ColorSeg> U, V ;
268                    ColorSeg M ;
269                    U = {u, {0, 0, 0}} ;
270                    V = {v, {0, 0, 0}} ;
271
272                    while(top[U.first] != top[V.first]){
273                        if(dep[top[U.first]] <
274                            dep[top[V.first]]) swap(U, V) ;
275                        U.second = query(dfn[top[U.first]],
276                                          dfn[U.first]) + U.second ;
277                        U.first = fa[top[U.first]] ;
278
279                        if(dep[U.first] < dep[V.first]) swap(U,
280                            V) ;
281
282                        M = query(dfn[V.first], dfn[U.first]) ;
283
284                    }
285
286                }
287
288            }
289
290        }
291
292    }
293
294    ColorSeg operator+(const ColorSeg &o)
295        const {
296        if(tot == 0) return o ;
297        if(o.tot == 0) return *this ;
298
299        ColorSeg tmp ;
300        tmp.left = left ;
301        tmp.right = o.right ;
302        tmp.tot = tot + o.tot - (right ==
303            o.left) ;
304
305        return tmp ;
306    }
307
308    struct Node{
309        ColorSeg color ;
310        int tag ;
311    }seg[Maxn << 2] ;
312
313    class SegmentTree{
314    private:
315        void pull(int id){
316            // normal pull
317        }
318
319        void AddTag(int id, int tag){
320            // normal AddTag
321        }
322
323        void push(int id){
324            // normal push
325        }
326
327        void modify(int l, int r, int tag, int
328                    L=1, int R=n, int id=1){
329            // normal modify
330        }
331
332        ColorSeg query(int l, int r, int L=1, int
333                       R=n, int id=1){
334            // normal query
335        }
336
337        public:
338        void build(int L=1, int R=n, int id=1){
339            // normal build
340
341
342            // update val from u to v (simple path)
343            void update(int u, int v, int val){
344                while(top[u] != top[v]){
345                    if(dep[top[u]] < dep[top[v]]) swap(u,
346                        v) ;
347                    modify(dfn[top[u]], dfn[u], val) ;
348                    u = fa[top[u]] ;
349
350                    if(dep[u] < dep[v]) swap(u, v) ;
351
352                    // get sum from u to v (simple path)
353                    pair<int, ColorSeg> U, V ;
354                    ColorSeg M ;
355                    U = {u, {0, 0, 0}} ;
356                    V = {v, {0, 0, 0}} ;
357
358                    while(top[U.first] != top[V.first]){
359                        if(dep[top[U.first]] <
360                            dep[top[V.first]]) swap(U, V) ;
361                        U.second = query(dfn[top[U.first]],
362                                          dfn[U.first]) + U.second ;
363                        U.first = fa[top[U.first]] ;
364
365                        if(dep[U.first] < dep[V.first]) swap(U,
366                            V) ;
367
368                        M = query(dfn[V.first], dfn[U.first]) ;
369
370                    }
371
372                }
373
374            }
375
376        }
377
378    }
379
380    ColorSeg operator+(const ColorSeg &o)
381        const {
382        if(tot == 0) return o ;
383        if(o.tot == 0) return *this ;
384
385        ColorSeg tmp ;
386        tmp.left = left ;
387        tmp.right = o.right ;
388        tmp.tot = tot + o.tot - (right ==
389            o.left) ;
390
391        return tmp ;
392    }
393
394    struct Node{
395        ColorSeg color ;
396        int tag ;
397    }seg[Maxn << 2] ;
398
399    class SegmentTree{
400    private:
401        void pull(int id){
402            // normal pull
403        }
404
405        void AddTag(int id, int tag){
406            // normal AddTag
407        }
408
409        void push(int id){
410            // normal push
411        }
412
413        void modify(int l, int r, int tag, int
414                    L=1, int R=n, int id=1){
415            // normal modify
416        }
417
418        ColorSeg query(int l, int r, int L=1, int
419                       R=n, int id=1){
420            // normal query
421        }
422
423        public:
424        void build(int L=1, int R=n, int id=1){
425            // normal build
426
427
428            // update val from u to v (simple path)
429            void update(int u, int v, int val){
430                while(top[u] != top[v]){
431                    if(dep[top[u]] < dep[top[v]]) swap(u,
432                        v) ;
433                    modify(dfn[top[u]], dfn[u], val) ;
434                    u = fa[top[u]] ;
435
436                    if(dep[u] < dep[v]) swap(u, v) ;
437
438                    // get sum from u to v (simple path)
439                    pair<int, ColorSeg> U, V ;
440                    ColorSeg M ;
441                    U = {u, {0, 0, 0}} ;
442                    V = {v, {0, 0, 0}} ;
443
444                    while(top[U.first] != top[V.first]){
445                        if(dep[top[U.first]] <
446                            dep[top[V.first]]) swap(U, V) ;
447                        U.second = query(dfn[top[U.first]],
448                                          dfn[U.first]) + U.second ;
449                        U.first = fa[top[U.first]] ;
450
451                        if(dep[U.first] < dep[V.first]) swap(U,
452                            V) ;
453
454                        M = query(dfn[V.first], dfn[U.first]) ;
455
456                    }
457
458                }
459
460            }
461
462        }
463
464    }
465
466    ColorSeg operator+(const ColorSeg &o)
467        const {
468        if(tot == 0) return o ;
469        if(o.tot == 0) return *this ;
470
471        ColorSeg tmp ;
472        tmp.left = left ;
473        tmp.right = o.right ;
474        tmp.tot = tot + o.tot - (right ==
475            o.left) ;
476
477        return tmp ;
478    }
479
480    struct Node{
481        ColorSeg color ;
482        int tag ;
483    }seg[Maxn << 2] ;
484
485    class SegmentTree{
486    private:
487        void pull(int id){
488            // normal pull
489        }
490
491        void AddTag(int id, int tag){
492            // normal AddTag
493        }
494
495        void push(int id){
496            // normal push
497        }
498
499        void modify(int l, int r, int tag, int
500                    L=1, int R=n, int id=1){
501            // normal modify
502        }
503
504        ColorSeg query(int l, int r, int L=1, int
505                       R=n, int id=1){
506            // normal query
507        }
508
509        public:
510        void build(int L=1, int R=n, int id=1){
511            // normal build
512
513
514            // update val from u to v (simple path)
515            void update(int u, int v, int val){
516                while(top[u] != top[v]){
517                    if(dep[top[u]] < dep[top[v]]) swap(u,
518                        v) ;
519                    modify(dfn[top[u]], dfn[u], val) ;
520                    u = fa[top[u]] ;
521
522                    if(dep[u] < dep[v]) swap(u, v) ;
523
524                    // get sum from u to v (simple path)
525                    pair<int, ColorSeg> U, V ;
526                    ColorSeg M ;
527                    U = {u, {0, 0, 0}} ;
528                    V = {v, {0, 0, 0}} ;
529
530                    while(top[U.first] != top[V.first]){
531                        if(dep[top[U.first]] <
532                            dep[top[V.first]]) swap(U, V) ;
533                        U.second = query(dfn[top[U.first]],
534                                          dfn[U.first]) + U.second ;
535                        U.first = fa[top[U.first]] ;
536
537                        if(dep[U.first] < dep[V.first]) swap(U,
538                            V) ;
539
540                        M = query(dfn[V.first], dfn[U.first]) ;
541
542                    }
543
544                }
545
546            }
547
548        }
549
550    }
551
552    ColorSeg operator+(const ColorSeg &o)
553        const {
554        if(tot == 0) return o ;
555        if(o.tot == 0) return *this ;
556
557        ColorSeg tmp ;
558        tmp.left = left ;
559        tmp.right = o.right ;
560        tmp.tot = tot + o.tot - (right ==
561            o.left) ;
562
563        return tmp ;
564    }
565
566    struct Node{
567        ColorSeg color ;
568        int tag ;
569    }seg[Maxn << 2] ;
570
571    class SegmentTree{
572    private:
573        void pull(int id){
574            // normal pull
575        }
576
577        void AddTag(int id, int tag){
578            // normal AddTag
579        }
580
581        void push(int id){
582            // normal push
583        }
584
585        void modify(int l, int r, int tag, int
586                    L=1, int R=n, int id=1){
587            // normal modify
588        }
589
590        ColorSeg query(int l, int r, int L=1, int
591                       R=n, int id=1){
592            // normal query
593        }
594
595        public:
596        void build(int L=1, int R=n, int id=1){
597            // normal build
598
599
600            // update val from u to v (simple path)
601            void update(int u, int v, int val){
602                while(top[u] != top[v]){
603                    if(dep[top[u]] < dep[top[v]]) swap(u,
604                        v) ;
605                    modify(dfn[top[u]], dfn[u], val) ;
606                    u = fa[top[u]] ;
607
608                    if(dep[u] < dep[v]) swap(u, v) ;
609
610                    // get sum from u to v (simple path)
611                    pair<int, ColorSeg> U, V ;
612                    ColorSeg M ;
613                    U = {u, {0, 0, 0}} ;
614                    V = {v, {0, 0, 0}} ;
615
616                    while(top[U.first] != top[V.first]){
617                        if(dep[top[U.first]] <
618                            dep[top[V.first]]) swap(U, V) ;
619                        U.second = query(dfn[top[U.first]],
620                                          dfn[U.first]) + U.second ;
621                        U.first = fa[top[U.first]] ;
622
623                        if(dep[U.first] < dep[V.first]) swap(U,
624                            V) ;
625
626                        M = query(dfn[V.first], dfn[U.first]) ;
627
628                    }
629
630                }
631
632            }
633
634        }
635
636    }
637
638    ColorSeg operator+(const ColorSeg &o)
639        const {
640        if(tot == 0) return o ;
641        if(o.tot == 0) return *this ;
642
643        ColorSeg tmp ;
644        tmp.left = left ;
645        tmp.right = o.right ;
646        tmp.tot = tot + o.tot - (right ==
647            o.left) ;
648
649        return tmp ;
650    }
651
652    struct Node{
653        ColorSeg color ;
654        int tag ;
655    }seg[Maxn << 2] ;
656
657    class SegmentTree{
658    private:
659        void pull(int id){
660            // normal pull
661        }
662
663        void AddTag(int id, int tag){
664            // normal AddTag
665        }
666
667        void push(int id){
668            // normal push
669        }
670
671        void modify(int l, int r, int tag, int
672                    L=1, int R=n, int id=1){
673            // normal modify
674        }
675
676        ColorSeg query(int l, int r, int L=1, int
677                       R=n, int id=1){
678            // normal query
679        }
680
681        public:
682        void build(int L=1, int R=n, int id=1){
683            // normal build
684
685
686            // update val from u to v (simple path)
687            void update(int u, int v, int val){
688                while(top[u] != top[v]){
689                    if(dep[top[u]] < dep[top[v]]) swap(u,
690                        v) ;
691                    modify(dfn[top[u]], dfn[u], val) ;
692                    u = fa[top[u]] ;
693
694                    if(dep[u] < dep[v]) swap(u, v) ;
695
696                    // get sum from u to v (simple path)
697                    pair<int, ColorSeg> U, V ;
698                    ColorSeg M ;
699                    U = {u, {0, 0, 0}} ;
700                    V = {v, {0, 0, 0}} ;
701
702                    while(top[U.first] != top[V.first]){
703                        if(dep[top[U.first]] <
704                            dep[top[V.first]]) swap(U, V) ;
705                        U.second = query(dfn[top[U.first]],
706                                          dfn[U.first]) + U.second ;
707                        U.first = fa[top[U.first]] ;
708
709                        if(dep[U.first] < dep[V.first]) swap(U,
710                            V) ;
711
712                        M = query(dfn[V.first], dfn[U.first]) ;
713
714                    }
715
716                }
717
718            }
719
720        }
721
722    }
723
724    ColorSeg operator+(const ColorSeg &o)
725        const {
726        if(tot == 0) return o ;
727        if(o.tot == 0) return *this ;
728
729        ColorSeg tmp ;
730        tmp.left = left ;
731        tmp.right = o.right ;
732        tmp.tot = tot + o.tot - (right ==
733            o.left) ;
734
735        return tmp ;
736    }
737
738    struct Node{
739        ColorSeg color ;
740        int tag ;
741    }seg[Maxn << 2] ;
742
743    class SegmentTree{
744    private:
745        void pull(int id){
746            // normal pull
747        }
748
749        void AddTag(int id, int tag){
750            // normal AddTag
751        }
752
753        void push(int id){
754            // normal push
755        }
756
757        void modify(int l, int r, int tag, int
758                    L=1, int R=n, int id=1){
759            // normal modify
760        }
761
762        ColorSeg query(int l, int r, int L=1, int
763                       R=n, int id=1){
764            // normal query
765        }
766
767        public:
768        void build(int L=1, int R=n, int id=1){
769            // normal build
770
771
772            // update val from u to v (simple path)
773            void update(int u, int v, int val){
774                while(top[u] != top[v]){
775                    if(dep[top[u]] < dep[top[v]]) swap(u,
776                        v) ;
777                    modify(dfn[top[u]], dfn[u], val) ;
778                    u = fa[top[u]] ;
779
780                    if(dep[u] < dep[v]) swap(u, v) ;
781
782                    // get sum from u to v (simple path)
783                    pair<int, ColorSeg> U, V ;
784                    ColorSeg M ;
785                    U = {u, {0, 0, 0}} ;
786                    V = {v, {0, 0, 0}} ;
787
788                    while(top[U.first] != top[V.first]){
789                        if(dep[top[U.first]] <
790                            dep[top[V.first]]) swap(U, V) ;
791                        U.second = query(dfn[top[U.first]],
792                                          dfn[U.first]) + U.second ;
793                        U.first = fa[top[U.first]] ;
794
795                        if(dep[U.first] < dep[V.first]) swap(U,
796                            V) ;
797
798                        M = query(dfn[V.first], dfn[U.first]) ;
799
800                    }
801
802                }
803
804            }
805
806        }
807
808    }
809
810    ColorSeg operator+(const ColorSeg &o)
811        const {
812        if(tot == 0) return o ;
813        if(o.tot == 0) return *this ;
814
815        ColorSeg tmp ;
816        tmp.left = left ;
817        tmp.right = o.right ;
818        tmp.tot = tot + o.tot - (right ==
819            o.left) ;
820
821        return tmp ;
822    }
823
824    struct Node{
825        ColorSeg color ;
826        int tag ;
827    }seg[Maxn << 2] ;
828
829    class SegmentTree{
830    private:
831        void pull(int id){
832            // normal pull
833        }
834
835        void AddTag(int id, int tag){
836            // normal AddTag
837        }
838
839        void push(int id){
840            // normal push
841        }
842
843        void modify(int l, int r, int tag, int
844                    L=1, int R=n, int id=1){
845            // normal modify
846        }
847
848        ColorSeg query(int l, int r, int L=1, int
849                       R=n, int id=1){
850            // normal query
851        }
852
853        public:
854        void build(int L=1, int R=n, int id=1){
855            // normal build
856
857
858            // update val from u to v (simple path)
859            void update(int u, int v, int val){
860                while(top[u] != top[v]){
861                    if(dep[top[u]] < dep[top[v]]) swap(u,
862                        v) ;
863                    modify(dfn[top[u]], dfn[u], val) ;
864                    u = fa[top[u]] ;
865
866                    if(dep[u] < dep[v]) swap(u, v) ;
867
868                    // get sum from u to v (simple path)
869                    pair<int, ColorSeg> U, V ;
870                    ColorSeg M ;
871                    U = {u, {0, 0, 0}} ;
872                    V = {v, {0, 0, 0}} ;
873
874                    while(top[U.first] != top[V.first]){
875                        if(dep[top[U.first]] <
876                            dep[top[V.first]]) swap(U, V) ;
877                        U.second = query(dfn[top[U.first]],
878                                          dfn[U.first]) + U.second ;
879                        U.first = fa[top[U.first]] ;
880
881                        if(dep[U.first] < dep[V.first]) swap(U,
882                            V) ;
883
884                        M = query(dfn[V.first], dfn[U.first]) ;
885
886                    }
887
888                }
889
890            }
891
892        }
893
894    }
895
896    ColorSeg operator+(const ColorSeg &o)
897        const {
898        if(tot == 0) return o ;
899        if(o.tot == 0) return *this ;
900
901        ColorSeg tmp ;
902        tmp.left = left ;
903        tmp.right = o.right ;
904        tmp.tot = tot + o.tot - (right ==
905            o.left) ;
906
907        return tmp ;
908    }
909
910    struct Node{
911        ColorSeg color ;
912        int tag ;
913    }seg[Maxn << 2] ;
914
915    class SegmentTree{
916    private:
917        void pull(int id){
918            // normal pull
919        }
920
921        void AddTag(int id, int tag){
922            // normal AddTag
923        }
924
925        void push(int id){
926            // normal push
927        }
928
929        void modify(int l, int r, int tag, int
930                    L=1, int R=n, int id=1){
931            // normal modify
932        }
933
934        ColorSeg query(int l, int r, int L=1, int
935                       R=n, int id=1){
936            // normal query
937        }
938
939        public:
940        void build(int L=1, int R=n, int id=1){
941            // normal build
942
943
944            // update val from u to v (simple path)
945            void update(int u, int v, int val){
946                while(top[u] != top[v]){
947                    if(dep[top[u]] < dep[top[v]]) swap(u,
948                        v) ;
949                    modify(dfn[top[u]], dfn[u], val) ;
950                    u = fa[top[u]] ;
951
952                    if(dep[u] < dep[v]) swap(u, v) ;
953
954                    // get sum from u to v (simple path)
955                    pair<int, ColorSeg> U, V ;
956                    ColorSeg M ;
957                    U = {u, {0, 0, 0}} ;
958                    V = {v, {0, 0, 0}} ;
959
960                    while(top[U.first] != top[V.first]){
961                        if(dep[top[U.first]] <
962                            dep[top[V.first]]) swap(U, V) ;
963                        U.second = query(dfn[top[U.first]],
964                                          dfn[U.first]) + U.second ;
965                        U.first = fa[top[U.first]] ;
966
967                        if(dep[U.first] < dep[V.first]) swap(U,
968                            V) ;
969
970                        M = query(dfn[V.first], dfn[U.first]) ;
971
972                    }
973
974                }
975
976            }
977
978        }
979
980    }
981
982    ColorSeg operator+(const ColorSeg &o)
983        const {
984        if(tot == 0) return o ;
985        if(o.tot == 0) return *this ;
986
987        ColorSeg tmp ;
988        tmp.left = left ;
989        tmp.right = o.right ;
990        tmp.tot = tot + o.tot - (right ==
991            o.left) ;
992
993        return tmp ;
994    }
995
996    struct Node{
997        ColorSeg color ;
998        int tag ;
999    }seg[Maxn << 2] ;
1000
1001    class SegmentTree{
1002    private:
1003        void pull(int id){
1004            // normal pull
1005        }
1006
1007        void AddTag(int id, int tag){
1008            // normal AddTag
1009        }
1010
1011        void push(int id){
1012            // normal push
1013        }
1014
1015        void modify(int l, int r, int tag, int
1016                    L=1, int R=n, int id=1){
1017            // normal modify
1018        }
1019
1020        ColorSeg query(int l, int r, int L=1, int
1021                       R=n, int id=1){
1022            // normal query
1023        }
1024
1025        public:
1026        void build(int L=1, int R=n, int id=1){
1027            // normal build
1028
1029
1030            // update val from u to v (simple path)
1031            void update(int u, int v, int val){
1032                while(top[u] != top[v]){
1033                    if(dep[top[u]] < dep[top[v]]) swap(u,
1034                        v) ;
1035                    modify(dfn[top[u]], dfn[u], val) ;
1036                    u = fa[top[u]] ;
1037
1038                    if(dep[u] < dep[v]) swap(u, v) ;
1039
1040                    // get sum from u to v (simple path)
1041                    pair<int, ColorSeg> U, V ;
1042                    ColorSeg M ;
1043                    U = {u, {0, 0, 0}} ;
1044                    V = {v, {0, 0, 0}} ;
1045
1046                    while(top[U.first] != top[V.first]){
1047                        if(dep[top[U.first]] <
1048                            dep[top[V.first]]) swap(U, V) ;
1049                        U.second = query(dfn[top[U.first]],
1050                                          dfn[U.first]) + U.second ;
1051                        U.first = fa[top[U.first]] ;
1052
1053                        if(dep[U.first] < dep[V.first]) swap(U,
1054                            V) ;
1055
1056                        M = query(dfn[V.first], dfn[U.first]) ;
1057
1058                    }
1059
1060                }
1061
1062            }
1063
1064        }
1065
1066    }
1067
1068    ColorSeg operator+(const ColorSeg &o)
1069        const {
1070        if(tot == 0) return o ;
1071        if(o.tot == 0) return *this ;
1072
1073        ColorSeg tmp ;
1074        tmp.left = left ;
1075        tmp.right = o.right ;
1076        tmp.tot = tot + o.tot - (right ==
1077            o.left) ;
1078
1079        return tmp ;
1080    }
1081
1082    struct Node{
1083        ColorSeg color ;
1084        int tag ;
1085    }seg[Maxn << 2] ;
1086
1087    class SegmentTree{
1088    private:
1089        void pull(int id){
1090            // normal pull
1091        }
1092
1093        void AddTag(int id, int tag){
1094            // normal AddTag
1095        }
1096
1097        void push(int id){
1098            // normal push
1099        }
1100
1101        void modify(int l, int r, int tag, int
1102                    L=1, int R=n, int id=1){
1103            // normal modify
1104        }
1105
1106        ColorSeg query(int l, int r, int L=1, int
1107                       R=n, int id=1){
1108            // normal query
1109        }
1110
1111        public:
1112        void build(int L=1, int R=n, int id=1){
1113            // normal build
1114
1115
1116            // update val from u to v (simple path)
1117            void update(int u, int v, int val){
1118                while(top[u] != top[v]){
1119                    if(dep[top[u]] < dep[top[v]]) swap(u,
1120                        v) ;
1121                    modify(dfn[top[u]], dfn[u], val) ;
1122                    u = fa[top[u]] ;
1123
1124                    if(dep[u] < dep[v]) swap(u, v) ;
1125
1126                    // get sum from u to v (simple path)
1127                    pair<int, ColorSeg> U, V ;
1128                    ColorSeg M ;
1129                    U = {u, {0, 0, 0}} ;
1130                    V = {v, {0, 0, 0}} ;
1131
1132                    while(top[U.first] != top[V.first]){
1133                        if(dep[top[U.first]] <
1134                            dep[top[V.first]]) swap(U, V) ;
1135                        U.second = query(dfn[top[U.first]],
1136                                          dfn[U.first]) + U.second ;
1137                        U.first = fa[top[U.first]] ;
1138
1139                        if(dep[U.first] < dep[V.first]) swap(U,
1140                            V) ;
1141
1142                        M = query(dfn[V.first], dfn[U.first]) ;
1143
1144                    }
1145
1146                }
1147
1148            }
1149
1150        }
1151
1152    }
1153
1154    ColorSeg operator+(const ColorSeg &o)
1155        const {
1156        if(tot == 0) return o ;
1157        if(o.tot == 0) return *this ;
1158
1159        ColorSeg tmp ;
1160        tmp.left = left ;
1161        tmp.right = o.right ;
1162        tmp.tot = tot + o.tot - (right ==
1163            o.left) ;
1164
1165        return tmp ;
1166    }
1167
1168    struct Node{
1169        ColorSeg color ;
1170        int tag ;
1171    }seg[Maxn << 2] ;
1172
1173    class SegmentTree{
1174    private:
1175        void pull(int id){
1176            // normal pull
1177        }
1178
1179        void AddTag(int id, int tag){
1180            // normal AddTag
1181        }
1182
1183        void push(int id){
1184            // normal push
1185        }
1186
1187        void modify(int l, int r, int tag, int
1188                    L=1, int R=n, int id=1){
1189            // normal modify
1190        }
1191
1192        ColorSeg query(int l, int r, int L=1, int
1193                       R=n, int id=1){
1194            // normal query
1195        }
1196
1197        public:
1198        void build(int L=1, int R=n, int id=1){
1199            // normal build
1200
1201
1202            // update val from u to v (simple path)
1203            void update(int u, int v, int val){
1204                while(top[u] != top[v]){
1205                    if(dep[top[u]] < dep[top[v]]) swap(u,
1206                        v) ;
1207                    modify(dfn[top[u]], dfn[u], val) ;
1208                    u = fa[top[u]] ;
1209
1210                    if(dep[u] < dep[v]) swap(u, v) ;
1211
1212                    // get sum from u to v (simple path)
1213                    pair<int,
```

## 4 Algorithm

### 4.1 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         if(u == v) return u ;
34
35         for ( int i=MaxLog ; i>=0 ; i-- )
36             if(up[u][i] != up[v][i]){
37                 u = up[u][i] ;
38                 v = up[v][i] ;
39             }
40
41     return up[u][0] ;
42 }
43
44 int main(){
45     int n, q, root ;
46     scanf( "%d%d%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         scanf( "%d%d" , &u, &v ) ;
61         printf( "%d\n" , lca(u, v)) ;
62     }
63
64 }
```

### 4.2 MST

```

1 struct Edge{
2     int u, v, w ;
3     // 這是最大生成樹，最小生成樹要改成 w < o.w
```

```

4     bool operator>(const Edge &o) const
5     {return w > o.w ;} ;
6
7     int par[N] ;
8     int sz[N] ;
9     int sum ;
10    vector<Edge> edge ;
11
12    void init(){
13        edge.clear() ;
14        for ( int i=0 ; i<N ; i++ ){
15            par[i] = i ;
16            sz[i] = 1 ;
17        }
18        sum = 0 ;
19    }
20
21    int find(int x){
22        if(x == par[x]) return x ;
23        return par[x] = find(par[x]) ;
24    }
25
26    int merge(int x, int y){
27        x = find(x) ;
28        y = find(y) ;
29
30        if(x == y) return 0 ;
31        if(sz[x] > sz[y]) swap(x, y) ;
32        par[x] = y ;
33        sz[y] += sz[x] ;
34
35        return 1 ;
36    }
37
38    void MST(){
39        int cnt = 0 ;
40        for ( int i=0 ; i<edge.size() && cnt < n-1
41               ; 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(),
58              greater<Edge>()) ;
59        MST() ;
60    }
61 }
```

### 4.3 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    int main(){
11        int cas, n;
12        scanf( "%d" , &cas);
13 }
```

```

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    int SG[30] ;
31    int vis[Maxn], stone[Maxn] ;
32
33    void build(){
34        SG[0] = 0 ;
35        memset(vis, 0, sizeof(vis)) ;
36
37        for ( int i=1 ; i<30 ; i++ ){
38            int cur = 0 ;
39            for ( int j=0 ; j<i ; j++ ) for ( int
40                  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        int main(){
48            build();
49
50            int T = 0 ;
51            while(~scanf( "%d" , &n) && n){
52                int ans = 0 ;
53
54                for ( int i=1 ; i<=n ; i++ ) scanf( "%d" ,
55                                              &stone[i] );
56
57                for ( int i=1 ; i<=n ; i++ ) if(stone[i]
58                                              & 1){
59                    ans ^= SG[n-i] ;
60                }
61            }
62        }
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 }
```

```

22
23 int n, x, y ;
24 char c ;
25 cin >> n ;
26
27 for ( int i=0 ; i<n ; i++ ){
28     cin >> x >> y >> c ;
29     if(c == 'Y') nodes.push_back({x, y}) ;
30 }
31
32 void monotone(){
33     sort(nodes.begin(), nodes.end()) ;
34
35     int n = unique(nodes.begin(), nodes.end())
36         - 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],
44             nodes[i]) < 0) m-- ;
45         ch[m++] = nodes[i] ;
46
47         for ( int i=n-2, t=m ; i>=0 ; i-- ){
48             while(m > t && cross(ch[m-2], ch[m-1],
49                 nodes[i]) < 0) m-- ;
50             ch[m++] = nodes[i] ;
51
52         if(n > 1) m-- ;
53         cout << m << endl ;
54
55         for ( int i=0 ; i<m ; i++ ) cout <<
56             ch[i].x << " " << ch[i].y << endl ;
57     }
58
59 }

```

## 4.5 Find Cut Vertex

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 const int Maxn = 2e4 + 5 ;
6
7 int n, m ;
8 vector<int> g[Maxn] ;
9
10 int dfn[Maxn], low[Maxn], fa[Maxn], dfscnt,
11     cnt ;
12 set<int> ans ;
13
14 void init(){
15     memset(dfn, -1, sizeof(dfn)) ;
16     memset(low, -1, sizeof(low)) ;
17     memset(fa, -1, sizeof(fa)) ;
18     dfscnt = 0 ;
19 }
20
21 void dfs(int u){
22     dfn[u] = low[u] = ++dfscnt ;
23
24     for ( auto v : g[u] ) if(v != fa[u]){
25         if(dfn[v] == -1){
26             fa[v] = u ;
27             dfs(v) ;
28             low[u] = min(low[u], low[v]) ;
29             if(fa[u] == -1) cnt++ ;
30             else if(low[v] >= dfn[u]){
31                 ans.insert(u) ;
32             }
33             else low[u] = min(low[u], dfn[v]) ;
34     }
35 }

```

```

36
37 int main(){
38     init() ;
39
40     scanf("%d%d", &n, &m) ;
41
42     while(m--){
43         int u, v ;
44         scanf("%d%d", &u, &v) ;
45         g[u].push_back(v) ;
46         g[v].push_back(u) ;
47     }
48
49     for ( int i=1 ; i<=n ; i++ ) if(dfn[i] ==
50         -1){
51         cnt = 0 ;
52         dfs(i) ;
53         if(cnt > 1) ans.insert(i) ;
54
55         printf("%d\n", ans.size()) ;
56         for ( auto node : ans ) printf("%d ", node) ;
57         printf("\n") ;
58     }
59
60 }

```

## 4.6 SCC

```

1 vector<int> scc[Maxn] ;
2 int dfn[Maxn], low[Maxn], sccId[Maxn],
3     dfscnt = 0, cnt_scc = 0 ;
4 stack<int> st ;
5 bitset<Maxn> inSt, vis ;
6
7 void dfs(int u, int from){
8     dfn[u] = low[u] = ++dfscnt ;
9     st.push(u) ;
10    inSt[u] = 1 ;
11
12    for ( auto v : g[u] ){
13        if(!inSt[v] && dfn[v] != -1) continue ;
14        if(dfn[v] == -1) dfs(v, u) ;
15        low[u] = min(low[u], low[v]) ;
16
17        if(dfn[u] == low[u]){
18            cnt_scc++ ;
19            int x ;
20
21            do{
22                x = st.top() ;
23                st.pop() ;
24
25                inSt[x] = 0 ;
26                sccId[x] = cnt_scc ;
27                scc[cnt_scc].push_back(x) ;
28            }
29            while(x != u) ;
30        }
31    }
32
33 void init(){
34     memset(dfn, -1, sizeof(dfn)) ;
35     memset(low, -1, sizeof(low)) ;
36 }
37
38 int main(){
39     init() ;
40     input() ;
41     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
42         == -1){
43         dfs(i, 0) ;
44     }
45
46     for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i]
47         == -1){
48         bcc.push_back(vector<int>()) ;
49         dfs2(i, bcc_cnt++) ;
50     }
51 }

```

## 4.7 BCC

```

1 struct Edge{
2     int v, next ;
3 }e[Maxm << 1] ;
4 int head[Maxm], tot = 1 ;
5
6 void add(int u, int v){
7     e[+tot] = {v, head[u]} ;
8     head[u] = tot ;
9     e[+tot] = {u, head[v]} ;
10    head[v] = tot ;
11 }
12
13 bitset<Maxm << 1> bz ;
14 vector<vector<int>> bcc ;
15 int dfn_cnt = 0, dfn[Maxn], low[Maxn],
16     vis_bcc[Maxn], bcc_cnt = 0 ;
17
18 void dfs1(int u, int from){
19     dfn[u] = low[u] = ++dfn_cnt ;
20
21     for ( int i=head[u] ; i!= -1 ; i=e[i].next
22         ){
23         int v = e[i].v ;
24
25         if(dfn[v] == -1){
26             dfs1(v, i) ;
27             if(dfn[u] < low[v]) bz[i] = bz[i^1] =
28                 1 ;
29             low[u] = min(low[u], low[v]) ;
30         }
31     }
32
33 void dfs2(int u, int id){
34     vis_bcc[u] = id ;
35     bcc[id].push_back(u) ;
36
37     for ( int i=head[u] ; i!= -1 ; i=e[i].next
38         ){
39         int v = e[i].v ;
40
41         if(vis_bcc[v] != -1 || bz[i]) continue ;
42         dfs2(v, id) ;
43     }
44
45 void init(){
46     memset(dfn, -1, sizeof(dfn)) ;
47     memset(head, -1, sizeof(head)) ;
48     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
49 }
50
51 int main(){
52     init() ;
53     input() ;
54     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
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 }

```

## 5 DP

### 5.1 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
23             j=0 ; j<m ; j++ ){
24             cur ^= 1 ;
25             memset(dp[cur], 0, sizeof(dp[cur])) ;
26
27             for ( int k=0 ; k<(1 << m) ; k++ ){
28                 update(k, k << 1) ; // not put
29                 if(i && !(k & (1 << (m - 1)))) {
30                     update(k, (k << 1) | (1 << m
31                         - 1)) ; // put up
32                     if(j && !(k & 1)) update(k, (k << 1
33                         - 3)) ; // put left
34                 }
35             }
36             printf( "%lld\n", dp[cur][(1 << m) - 1]
37         }
38         return 0 ;
39     }
40 }
```

## 5.2 數位 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4
5 int K ;
6 int dp[20][105][105][2] ;
7 vector<int> dig ;
8
9 int solve(int pos, int sum, int dsum, bool
10    lim){
11    if(pos == -1){
12        if(sum == 0 && dsum == 0) return 1 ;
13        return 0 ;
14    }
15
16    int &d = dp[pos][sum][dsum][lim] ;
17    if(d != -1) return d ;
18
19    int up = lim ? dig[pos] : 9 ;
20    int res = 0 ;
21    for ( int i=0 ; i<=up ; i++ ){
22        res += solve(pos-1, (sum * 10 + i) %
23                      K, (dsum + i) % K, lim && i==up)
24                      ;
25    }
26
27    return d = res ;
28 }
29
30 int count(int n){
31    memset(dp, -1, sizeof(dp)) ;
32    dig.clear() ;
33
34    while(n > 0){
35

```

```

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("%d\n") ;
47         else printf("%d\n", count(b) -
48                     count(a-1)) ;
49     }
50
51     return 0 ;
52 }

53
54     int u, v, w ;
55     scanf("%d%d%d", &v, &u, &w) ;
56     edge[u].push_back({v, w}) ;
57 }
58
59     DFS(0) ;
60     printf("Case %d:\n", ++t) ;
61
62     int q, e ;
63     scanf("%d", &q) ;
64
65     while(q--){
66         scanf(" %d", &e) ;
67
68         for ( int i=n ; i>=1 ; i-- )
69             if(dp[0][i][0] <= e){
70                 printf("%d\n", i) ;
71                 break ;
72             }
73     }
74
75     return 0 ;
76 }
```

5.3 樹 DF