

## 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    }
20
21    void dfs3(int u, int t){
22        if(son[u] == -1) top[u] = t ;
23        else dfs3(son[u], t) ;
24    }
25
26    void dfs4(int u, int t){
27        if(son[u] == -1) top[u] = t ;
28        else dfs4(son[u], t) ;
29    }
30
31    void dfs5(int u, int t){
32        if(son[u] == -1) top[u] = t ;
33        else dfs5(son[u], t) ;
34    }
35
36    void dfs6(int u, int t){
37        if(son[u] == -1) top[u] = t ;
38        else dfs6(son[u], t) ;
39    }
40
41    void dfs7(int u, int t){
42        if(son[u] == -1) top[u] = t ;
43        else dfs7(son[u], t) ;
44    }
45
46    void dfs8(int u, int t){
47        if(son[u] == -1) top[u] = t ;
48        else dfs8(son[u], t) ;
49    }
50
51    void dfs9(int u, int t){
52        if(son[u] == -1) top[u] = t ;
53        else dfs9(son[u], t) ;
54    }
55
56    void dfs10(int u, int t){
57        if(son[u] == -1) top[u] = t ;
58        else dfs10(son[u], t) ;
59    }
60
61    void dfs11(int u, int t){
62        if(son[u] == -1) top[u] = t ;
63        else dfs11(son[u], t) ;
64    }
65
66    void dfs12(int u, int t){
67        if(son[u] == -1) top[u] = t ;
68        else dfs12(son[u], t) ;
69    }
70
71    void dfs13(int u, int t){
72        if(son[u] == -1) top[u] = t ;
73        else dfs13(son[u], t) ;
74    }
75
76    void dfs14(int u, int t){
77        if(son[u] == -1) top[u] = t ;
78        else dfs14(son[u], t) ;
79    }
80
81    void dfs15(int u, int t){
82        if(son[u] == -1) top[u] = t ;
83        else dfs15(son[u], t) ;
84    }
85
86    void dfs16(int u, int t){
87        if(son[u] == -1) top[u] = t ;
88        else dfs16(son[u], t) ;
89    }
90
91    void dfs17(int u, int t){
92        if(son[u] == -1) top[u] = t ;
93        else dfs17(son[u], t) ;
94    }
95
96    void dfs18(int u, int t){
97        if(son[u] == -1) top[u] = t ;
98        else dfs18(son[u], t) ;
99    }
100
101    void dfs19(int u, int t){
102        if(son[u] == -1) top[u] = t ;
103        else dfs19(son[u], t) ;
104    }
105
106    void dfs20(int u, int t){
107        if(son[u] == -1) top[u] = t ;
108        else dfs20(son[u], t) ;
109    }
110
111    void dfs21(int u, int t){
112        if(son[u] == -1) top[u] = t ;
113        else dfs21(son[u], t) ;
114    }
115
116    void dfs22(int u, int t){
117        if(son[u] == -1) top[u] = t ;
118        else dfs22(son[u], t) ;
119    }
120
121    void dfs23(int u, int t){
122        if(son[u] == -1) top[u] = t ;
123        else dfs23(son[u], t) ;
124    }
125
126    void dfs24(int u, int t){
127        if(son[u] == -1) top[u] = t ;
128        else dfs24(son[u], t) ;
129    }
130
131    void dfs25(int u, int t){
132        if(son[u] == -1) top[u] = t ;
133        else dfs25(son[u], t) ;
134    }
135
136    void dfs26(int u, int t){
137        if(son[u] == -1) top[u] = t ;
138        else dfs26(son[u], t) ;
139    }
140
141    void dfs27(int u, int t){
142        if(son[u] == -1) top[u] = t ;
143        else dfs27(son[u], t) ;
144    }
145
146    void dfs28(int u, int t){
147        if(son[u] == -1) top[u] = t ;
148        else dfs28(son[u], t) ;
149    }
150
151    void dfs29(int u, int t){
152        if(son[u] == -1) top[u] = t ;
153        else dfs29(son[u], t) ;
154    }
155
156    void dfs30(int u, int t){
157        if(son[u] == -1) top[u] = t ;
158        else dfs30(son[u], t) ;
159    }
160
161    void dfs31(int u, int t){
162        if(son[u] == -1) top[u] = t ;
163        else dfs31(son[u], t) ;
164    }
165
166    void dfs32(int u, int t){
167        if(son[u] == -1) top[u] = t ;
168        else dfs32(son[u], t) ;
169    }
170
171    void dfs33(int u, int t){
172        if(son[u] == -1) top[u] = t ;
173        else dfs33(son[u], t) ;
174    }
175
176    void dfs34(int u, int t){
177        if(son[u] == -1) top[u] = t ;
178        else dfs34(son[u], t) ;
179    }
180
181    void dfs35(int u, int t){
182        if(son[u] == -1) top[u] = t ;
183        else dfs35(son[u], t) ;
184    }
185
186    void dfs36(int u, int t){
187        if(son[u] == -1) top[u] = t ;
188        else dfs36(son[u], t) ;
189    }
190
191    void dfs37(int u, int t){
192        if(son[u] == -1) top[u] = t ;
193        else dfs37(son[u], t) ;
194    }
195
196    void dfs38(int u, int t){
197        if(son[u] == -1) top[u] = t ;
198        else dfs38(son[u], t) ;
199    }
200
201    void dfs39(int u, int t){
202        if(son[u] == -1) top[u] = t ;
203        else dfs39(son[u], t) ;
204    }
205
206    void dfs40(int u, int t){
207        if(son[u] == -1) top[u] = t ;
208        else dfs40(son[u], t) ;
209    }
210
211    void dfs41(int u, int t){
212        if(son[u] == -1) top[u] = t ;
213        else dfs41(son[u], t) ;
214    }
215
216    void dfs42(int u, int t){
217        if(son[u] == -1) top[u] = t ;
218        else dfs42(son[u], t) ;
219    }
220
221    void dfs43(int u, int t){
222        if(son[u] == -1) top[u] = t ;
223        else dfs43(son[u], t) ;
224    }
225
226    void dfs44(int u, int t){
227        if(son[u] == -1) top[u] = t ;
228        else dfs44(son[u], t) ;
229    }
230
231    void dfs45(int u, int t){
232        if(son[u] == -1) top[u] = t ;
233        else dfs45(son[u], t) ;
234    }
235
236    void dfs46(int u, int t){
237        if(son[u] == -1) top[u] = t ;
238        else dfs46(son[u], t) ;
239    }
240
241    void dfs47(int u, int t){
242        if(son[u] == -1) top[u] = t ;
243        else dfs47(son[u], t) ;
244    }
245
246    void dfs48(int u, int t){
247        if(son[u] == -1) top[u] = t ;
248        else dfs48(son[u], t) ;
249    }
250
251    void dfs49(int u, int t){
252        if(son[u] == -1) top[u] = t ;
253        else dfs49(son[u], t) ;
254    }
255
256    void dfs50(int u, int t){
257        if(son[u] == -1) top[u] = t ;
258        else dfs50(son[u], t) ;
259    }
260
261    void dfs51(int u, int t){
262        if(son[u] == -1) top[u] = t ;
263        else dfs51(son[u], t) ;
264    }
265
266    void dfs52(int u, int t){
267        if(son[u] == -1) top[u] = t ;
268        else dfs52(son[u], t) ;
269    }
270
271    void dfs53(int u, int t){
272        if(son[u] == -1) top[u] = t ;
273        else dfs53(son[u], t) ;
274    }
275
276    void dfs54(int u, int t){
277        if(son[u] == -1) top[u] = t ;
278        else dfs54(son[u], t) ;
279    }
280
281    void dfs55(int u, int t){
282        if(son[u] == -1) top[u] = t ;
283        else dfs55(son[u], t) ;
284    }
285
286    void dfs56(int u, int t){
287        if(son[u] == -1) top[u] = t ;
288        else dfs56(son[u], t) ;
289    }
290
291    void dfs57(int u, int t){
292        if(son[u] == -1) top[u] = t ;
293        else dfs57(son[u], t) ;
294    }
295
296    void dfs58(int u, int t){
297        if(son[u] == -1) top[u] = t ;
298        else dfs58(son[u], t) ;
299    }
299
300    void dfs60(int u, int t){
301        if(son[u] == -1) top[u] = t ;
302        else dfs60(son[u], t) ;
303    }
304
305    void dfs61(int u, int t){
306        if(son[u] == -1) top[u] = t ;
307        else dfs61(son[u], t) ;
308    }
309
310    void dfs62(int u, int t){
311        if(son[u] == -1) top[u] = t ;
312        else dfs62(son[u], t) ;
313    }
314
315    void dfs63(int u, int t){
316        if(son[u] == -1) top[u] = t ;
317        else dfs63(son[u], t) ;
318    }
319
320    void dfs64(int u, int t){
321        if(son[u] == -1) top[u] = t ;
322        else dfs64(son[u], t) ;
323    }
324
325    void dfs65(int u, int t){
326        if(son[u] == -1) top[u] = t ;
327        else dfs65(son[u], t) ;
328    }
329
330    void dfs66(int u, int t){
331        if(son[u] == -1) top[u] = t ;
332        else dfs66(son[u], t) ;
333    }
334
335    void dfs67(int u, int t){
336        if(son[u] == -1) top[u] = t ;
337        else dfs67(son[u], t) ;
338    }
339
340    void dfs68(int u, int t){
341        if(son[u] == -1) top[u] = t ;
342        else dfs68(son[u], t) ;
343    }
344
345    void dfs69(int u, int t){
346        if(son[u] == -1) top[u] = t ;
347        else dfs69(son[u], t) ;
348    }
349
350    void dfs70(int u, int t){
351        if(son[u] == -1) top[u] = t ;
352        else dfs70(son[u], t) ;
353    }
354
355    void dfs71(int u, int t){
356        if(son[u] == -1) top[u] = t ;
357        else dfs71(son[u], t) ;
358    }
359
360    void dfs72(int u, int t){
361        if(son[u] == -1) top[u] = t ;
362        else dfs72(son[u], t) ;
363    }
364
365    void dfs73(int u, int t){
366        if(son[u] == -1) top[u] = t ;
367        else dfs73(son[u], t) ;
368    }
369
370    void dfs74(int u, int t){
371        if(son[u] == -1) top[u] = t ;
372        else dfs74(son[u], t) ;
373    }
374
375    void dfs75(int u, int t){
376        if(son[u] == -1) top[u] = t ;
377        else dfs75(son[u], t) ;
378    }
379
380    void dfs76(int u, int t){
381        if(son[u] == -1) top[u] = t ;
382        else dfs76(son[u], t) ;
383    }
384
385    void dfs77(int u, int t){
386        if(son[u] == -1) top[u] = t ;
387        else dfs77(son[u], t) ;
388    }
389
390    void dfs78(int u, int t){
391        if(son[u] == -1) top[u] = t ;
392        else dfs78(son[u], t) ;
393    }
394
395    void dfs79(int u, int t){
396        if(son[u] == -1) top[u] = t ;
397        else dfs79(son[u], t) ;
398    }
399
399
400    void dfs81(int u, int t){
401        if(son[u] == -1) top[u] = t ;
402        else dfs81(son[u], t) ;
403    }
404
405    void dfs82(int u, int t){
406        if(son[u] == -1) top[u] = t ;
407        else dfs82(son[u], t) ;
408    }
409
410    void dfs83(int u, int t){
411        if(son[u] == -1) top[u] = t ;
412        else dfs83(son[u], t) ;
413    }
414
415    void dfs84(int u, int t){
416        if(son[u] == -1) top[u] = t ;
417        else dfs84(son[u], t) ;
418    }
419
420    void dfs85(int u, int t){
421        if(son[u] == -1) top[u] = t ;
422        else dfs85(son[u], t) ;
423    }
424
425    void dfs86(int u, int t){
426        if(son[u] == -1) top[u] = t ;
427        else dfs86(son[u], t) ;
428    }
429
430    void dfs87(int u, int t){
431        if(son[u] == -1) top[u] = t ;
432        else dfs87(son[u], t) ;
433    }
434
435    void dfs88(int u, int t){
436        if(son[u] == -1) top[u] = t ;
437        else dfs88(son[u], t) ;
438    }
439
440    void dfs89(int u, int t){
441        if(son[u] == -1) top[u] = t ;
442        else dfs89(son[u], t) ;
443    }
444
445    void dfs90(int u, int t){
446        if(son[u] == -1) top[u] = t ;
447        else dfs90(son[u], t) ;
448    }
449
450    void dfs91(int u, int t){
451        if(son[u] == -1) top[u] = t ;
452        else dfs91(son[u], t) ;
453    }
454
455    void dfs92(int u, int t){
456        if(son[u] == -1) top[u] = t ;
457        else dfs92(son[u], t) ;
458    }
459
460    void dfs93(int u, int t){
461        if(son[u] == -1) top[u] = t ;
462        else dfs93(son[u], t) ;
463    }
464
465    void dfs94(int u, int t){
466        if(son[u] == -1) top[u] = t ;
467        else dfs94(son[u], t) ;
468    }
469
470    void dfs95(int u, int t){
471        if(son[u] == -1) top[u] = t ;
472        else dfs95(son[u], t) ;
473    }
474
475    void dfs96(int u, int t){
476        if(son[u] == -1) top[u] = t ;
477        else dfs96(son[u], t) ;
478    }
479
480    void dfs97(int u, int t){
481        if(son[u] == -1) top[u] = t ;
482        else dfs97(son[u], t) ;
483    }
484
485    void dfs98(int u, int t){
486        if(son[u] == -1) top[u] = t ;
487        else dfs98(son[u], t) ;
488    }
489
490    void dfs99(int u, int t){
491        if(son[u] == -1) top[u] = t ;
492        else dfs99(son[u], t) ;
493    }
494
495    void dfs100(int u, int t){
496        if(son[u] == -1) top[u] = t ;
497        else dfs100(son[u], t) ;
498    }
499
500    void dfs101(int u, int t){
501        if(son[u] == -1) top[u] = t ;
502        else dfs101(son[u], t) ;
503    }
504
505    void dfs102(int u, int t){
506        if(son[u] == -1) top[u] = t ;
507        else dfs102(son[u], t) ;
508    }
509
510    void dfs103(int u, int t){
511        if(son[u] == -1) top[u] = t ;
512        else dfs103(son[u], t) ;
513    }
514
515    void dfs104(int u, int t){
516        if(son[u] == -1) top[u] = t ;
517        else dfs104(son[u], t) ;
518    }
519
520    void dfs105(int u, int t){
521        if(son[u] == -1) top[u] = t ;
522        else dfs105(son[u], t) ;
523    }
524
525    void dfs106(int u, int t){
526        if(son[u] == -1) top[u] = t ;
527        else dfs106(son[u], t) ;
528    }
529
530    void dfs107(int u, int t){
531        if(son[u] == -1) top[u] = t ;
532        else dfs107(son[u], t) ;
533    }
534
535    void dfs108(int u, int t){
536        if(son[u] == -1) top[u] = t ;
537        else dfs108(son[u], t) ;
538    }
539
540    void dfs109(int u, int t){
541        if(son[u] == -1) top[u] = t ;
542        else dfs109(son[u], t) ;
543    }
544
545    void dfs110(int u, int t){
546        if(son[u] == -1) top[u] = t ;
547        else dfs110(son[u], t) ;
548    }
549
550    void dfs111(int u, int t){
551        if(son[u] == -1) top[u] = t ;
552        else dfs111(son[u], t) ;
553    }
554
555    void dfs112(int u, int t){
556        if(son[u] == -1) top[u] = t ;
557        else dfs112(son[u], t) ;
558    }
559
560    void dfs113(int u, int t){
561        if(son[u] == -1) top[u] = t ;
562        else dfs113(son[u], t) ;
563    }
564
565    void dfs114(int u, int t){
566        if(son[u] == -1) top[u] = t ;
567        else dfs114(son[u], t) ;
568    }
569
570    void dfs115(int u, int t){
571        if(son[u] == -1) top[u] = t ;
572        else dfs115(son[u], t) ;
573    }
574
575    void dfs116(int u, int t){
576        if(son[u] == -1) top[u] = t ;
577        else dfs116(son[u], t) ;
578    }
579
580    void dfs117(int u, int t){
581        if(son[u] == -1) top[u] = t ;
582        else dfs117(son[u], t) ;
583    }
584
585    void dfs118(int u, int t){
586        if(son[u] == -1) top[u] = t ;
587        else dfs118(son[u], t) ;
588    }
589
590    void dfs119(int u, int t){
591        if(son[u] == -1) top[u] = t ;
592        else dfs119(son[u], t) ;
593    }
594
595    void dfs120(int u, int t){
596        if(son[u] == -1) top[u] = t ;
597        else dfs120(son[u], t) ;
598    }
599
599
600    void dfs122(int u, int t){
601        if(son[u] == -1) top[u] = t ;
602        else dfs122(son[u], t) ;
603    }
604
605    void dfs123(int u, int t){
606        if(son[u] == -1) top[u] = t ;
607        else dfs123(son[u], t) ;
608    }
609
610    void dfs124(int u, int t){
611        if(son[u] == -1) top[u] = t ;
612        else dfs124(son[u], t) ;
613    }
614
615    void dfs125(int u, int t){
616        if(son[u] == -1) top[u] = t ;
617        else dfs125(son[u], t) ;
618    }
619
620    void dfs126(int u, int t){
621        if(son[u] == -1) top[u] = t ;
622        else dfs126(son[u], t) ;
623    }
624
625    void dfs127(int u, int t){
626        if(son[u] == -1) top[u] = t ;
627        else dfs127(son[u], t) ;
628    }
629
630    void dfs128(int u, int t){
631        if(son[u] == -1) top[u] = t ;
632        else dfs128(son[u], t) ;
633    }
634
635    void dfs129(int u, int t){
636        if(son[u] == -1) top[u] = t ;
637        else dfs129(son[u], t) ;
638    }
639
640    void dfs130(int u, int t){
641        if(son[u] == -1) top[u] = t ;
642        else dfs130(son[u], t) ;
643    }
644
645    void dfs131(int u, int t){
646        if(son[u] == -1) top[u] = t ;
647        else dfs131(son[u], t) ;
648    }
649
650    void dfs132(int u, int t){
651        if(son[u] == -1) top[u] = t ;
652        else dfs132(son[u], t) ;
653    }
654
655    void dfs133(int u, int t){
656        if(son[u] == -1) top[u] = t ;
657        else dfs133(son[u], t) ;
658    }
659
660    void dfs134(int u, int t){
661        if(son[u] == -1) top[u] = t ;
662        else dfs134(son[u], t) ;
663    }
664
665    void dfs135(int u, int t){
666        if(son[u] == -1) top[u] = t ;
667        else dfs135(son[u], t) ;
668    }
669
670    void dfs136(int u, int t){
671        if(son[u] == -1) top[u] = t ;
672        else dfs136(son[u], t) ;
673    }
674
675    void dfs137(int u, int t){
676        if(son[u] == -1) top[u] = t ;
677        else dfs137(son[u], t) ;
678    }
679
680    void dfs138(int u, int t){
681        if(son[u] == -1) top[u] = t ;
682        else dfs138(son[u], t) ;
683    }
684
685    void dfs139(int u, int t){
686        if(son[u] == -1) top[u] = t ;
687        else dfs139(son[u], t) ;
688    }
689
690    void dfs140(int u, int t){
691        if(son[u] == -1) top[u] = t ;
692        else dfs140(son[u], t) ;
693    }
694
695    void dfs141(int u, int t){
696        if(son[u] == -1) top[u] = t ;
697        else dfs141(son[u], t) ;
698    }
699
699
700    void dfs142(int u, int t){
701        if(son[u] == -1) top[u] = t ;
702        else dfs142(son[u], t) ;
703    }
704
705    void dfs143(int u, int t){
706        if(son[u] == -1) top[u] = t ;
707        else dfs143(son[u], t) ;
708    }
709
710    void dfs144(int u, int t){
711        if(son[u] == -1) top[u] = t ;
712        else dfs144(son[u], t) ;
713    }
714
715    void dfs145(int u, int t){
716        if(son[u] == -1) top[u] = t ;
717        else dfs145(son[u], t) ;
718    }
719
720    void dfs146(int u, int t){
721        if(son[u] == -1) top[u] = t ;
722        else dfs146(son[u], t) ;
723    }
724
725    void dfs147(int u, int t){
726        if(son[u] == -1) top[u] = t ;
727        else dfs147(son[u], t) ;
728    }
729
730    void dfs148(int u, int t){
731        if(son[u] == -1) top[u] = t ;
732        else dfs148(son[u], t) ;
733    }
734
735    void dfs149(int u, int t){
736        if(son[u] == -1) top[u] = t ;
737        else dfs149(son[u], t) ;
738    }
739
740    void dfs150(int u, int t){
741        if(son[u] == -1) top[u] = t ;
742        else dfs150(son[u], t) ;
743    }
744
745    void dfs151(int u, int t){
746        if(son[u] == -1) top[u] = t ;
747        else dfs151(son[u], t) ;
748    }
749
750    void dfs152(int u, int t){
751        if(son[u] == -1) top[u] = t ;
752        else dfs152(son[u], t) ;
753    }
754
755    void dfs153(int u, int t){
756        if(son[u] == -1) top[u] = t ;
757        else dfs153(son[u], t) ;
758    }
759
760    void dfs154(int u, int t){
761        if(son[u] == -1) top[u] = t ;
762        else dfs154(son[u], t) ;
763    }
764
765    void dfs155(int u, int t){
766        if(son[u] == -1) top[u] = t ;
767        else dfs155(son[u], t) ;
768    }
769
770    void dfs156(int u, int t){
771        if(son[u] == -1) top[u] = t ;
772        else dfs156(son[u], t) ;
773    }
774
775    void dfs157(int u, int t){
776        if(son[u] == -1) top[u] = t ;
777        else dfs157(son[u], t) ;
778    }
779
780    void dfs158(int u, int t){
781        if(son[u] == -1) top[u] = t ;
782        else dfs158(son[u], t) ;
783    }
784
785    void dfs159(int u, int t){
786        if(son[u] == -1) top[u] = t ;
787        else dfs159(son[u], t) ;
788    }
789
790    void dfs160(int u, int t){
791        if(son[u] == -1) top[u] = t ;
792        else dfs160(son[u], t) ;
793    }
794
795    void dfs161(int u, int t){
796        if(son[u] == -1) top[u] = t ;
797        else dfs161(son[u], t) ;
798    }
799
799
800    void dfs162(int u, int t){
801        if(son[u] == -1) top[u] = t ;
802        else dfs162(son[u], t) ;
803    }
804
805    void dfs163(int u, int t){
806        if(son[u] == -1) top[u] = t ;
807        else dfs163(son[u], t) ;
808    }
809
810    void dfs164(int u, int t){
811        if(son[u] == -1) top[u] = t ;
812        else dfs164(son[u], t) ;
813    }
814
815    void dfs165(int u, int t){

```

```

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                    void init(){
200                        memset(son, -1, sizeof(son)) ;
201                    }
202
203                }
204
205            }
206
207        }
208
209    }
210
211    ColorSeg operator+(const ColorSeg &o)
212        const {
213        if(tot == 0) return o ;
214        if(o.tot == 0) return *this ;
215
216        ColorSeg tmp ;
217        tmp.left = left ;
218        tmp.right = o.right ;
219        tmp.tot = tot + o.tot - (right ==
220            o.left) ;
221
222        return tmp ;
223    }
224
225    struct Node{
226        ColorSeg color ;
227        int tag ;
228    }seg[Maxn << 2] ;
229
230    class SegmentTree{
231    private:
232        void pull(int id){
233            // normal pull
234        }
235
236        void AddTag(int id, int tag){
237            // normal AddTag
238        }
239
240        void push(int id){
241            // normal push
242        }
243
244        void modify(int l, int r, int tag, int
245                    L=1, int R=n, int id=1){
246            // normal modify
247        }
248
249        ColorSeg query(int l, int r, int L=1, int
250                       R=n, int id=1){
251            // normal query
252        }
253
254        public:
255        void build(int L=1, int R=n, int id=1){
256            // normal build
257
258
259            // update val from u to v (simple path)
260            void update(int u, int v, int val){
261                while(top[u] != top[v]){
262                    if(dep[top[u]] < dep[top[v]]) swap(u,
263                        v) ;
264                    modify(dfn[top[u]], dfn[u], val) ;
265                    u = fa[top[u]] ;
266
267                    if(dep[u] < dep[v]) swap(u, v) ;
268
269                    // get sum from u to v (simple path)
270                    pair<int, ColorSeg> U, V ;
271                    ColorSeg M ;
272                    U = {u, {0, 0, 0}} ;
273                    V = {v, {0, 0, 0}} ;
274
275                    while(top[U.first] != top[V.first]){
276                        if(dep[top[U.first]] <
277                            dep[top[V.first]]) swap(U, V) ;
278                        U.second = query(dfn[top[U.first]],
279                                          dfn[U.first]) + U.second ;
280                        U.first = fa[top[U.first]] ;
281
282                        if(dep[U.first] < dep[V.first]) swap(U,
283                            V) ;
284
285                        M = query(dfn[V.first], dfn[U.first]) ;
286                    }
287
288                    void init(){
289                        memset(son, -1, sizeof(son)) ;
290                    }
291
292                }
293
294            }
295
296        }
297
298    }
299
300    ColorSeg operator+(const ColorSeg &o)
301        const {
302        if(tot == 0) return o ;
303        if(o.tot == 0) return *this ;
304
305        ColorSeg tmp ;
306        tmp.left = left ;
307        tmp.right = o.right ;
308        tmp.tot = tot + o.tot - (right ==
309            o.left) ;
310
311        return tmp ;
312    }
313
314    struct Node{
315        ColorSeg color ;
316        int tag ;
317    }seg[Maxn << 2] ;
318
319    class SegmentTree{
320    private:
321        void pull(int id){
322            // normal pull
323        }
324
325        void AddTag(int id, int tag){
326            // normal AddTag
327        }
328
329        void push(int id){
330            // normal push
331        }
332
333        void modify(int l, int r, int tag, int
334                    L=1, int R=n, int id=1){
335            // normal modify
336        }
337
338        ColorSeg query(int l, int r, int L=1, int
339                       R=n, int id=1){
340            // normal query
341        }
342
343        public:
344        void build(int L=1, int R=n, int id=1){
345            // normal build
346
347
348            // update val from u to v (simple path)
349            void update(int u, int v, int val){
350                while(top[u] != top[v]){
351                    if(dep[top[u]] < dep[top[v]]) swap(u,
352                        v) ;
353                    modify(dfn[top[u]], dfn[u], val) ;
354                    u = fa[top[u]] ;
355
356                    if(dep[u] < dep[v]) swap(u, v) ;
357
358                    // get sum from u to v (simple path)
359                    pair<int, ColorSeg> U, V ;
360                    ColorSeg M ;
361                    U = {u, {0, 0, 0}} ;
362                    V = {v, {0, 0, 0}} ;
363
364                    while(top[U.first] != top[V.first]){
365                        if(dep[top[U.first]] <
366                            dep[top[V.first]]) swap(U, V) ;
367                        U.second = query(dfn[top[U.first]],
368                                          dfn[U.first]) + U.second ;
369                        U.first = fa[top[U.first]] ;
370
371                        if(dep[U.first] < dep[V.first]) swap(U,
372                            V) ;
373
374                        M = query(dfn[V.first], dfn[U.first]) ;
375                    }
376
377                    void init(){
378                        memset(son, -1, sizeof(son)) ;
379                    }
380
381                }
382
383            }
384
385        }
386
387    }
388
389    ColorSeg operator+(const ColorSeg &o)
390        const {
391        if(tot == 0) return o ;
392        if(o.tot == 0) return *this ;
393
394        ColorSeg tmp ;
395        tmp.left = left ;
396        tmp.right = o.right ;
397        tmp.tot = tot + o.tot - (right ==
398            o.left) ;
399
400        return tmp ;
401    }
402
403    struct Node{
404        ColorSeg color ;
405        int tag ;
406    }seg[Maxn << 2] ;
407
408    class SegmentTree{
409    private:
410        void pull(int id){
411            // normal pull
412        }
413
414        void AddTag(int id, int tag){
415            // normal AddTag
416        }
417
418        void push(int id){
419            // normal push
420        }
421
422        void modify(int l, int r, int tag, int
423                    L=1, int R=n, int id=1){
424            // normal modify
425        }
426
427        ColorSeg query(int l, int r, int L=1, int
428                       R=n, int id=1){
429            // normal query
430        }
431
432        public:
433        void build(int L=1, int R=n, int id=1){
434            // normal build
435
436
437            // update val from u to v (simple path)
438            void update(int u, int v, int val){
439                while(top[u] != top[v]){
440                    if(dep[top[u]] < dep[top[v]]) swap(u,
441                        v) ;
442                    modify(dfn[top[u]], dfn[u], val) ;
443                    u = fa[top[u]] ;
444
445                    if(dep[u] < dep[v]) swap(u, v) ;
446
447                    // get sum from u to v (simple path)
448                    pair<int, ColorSeg> U, V ;
449                    ColorSeg M ;
450                    U = {u, {0, 0, 0}} ;
451                    V = {v, {0, 0, 0}} ;
452
453                    while(top[U.first] != top[V.first]){
454                        if(dep[top[U.first]] <
455                            dep[top[V.first]]) swap(U, V) ;
456                        U.second = query(dfn[top[U.first]],
457                                          dfn[U.first]) + U.second ;
458                        U.first = fa[top[U.first]] ;
459
460                        if(dep[U.first] < dep[V.first]) swap(U,
461                            V) ;
462
463                        M = query(dfn[V.first], dfn[U.first]) ;
464                    }
465
466                    void init(){
467                        memset(son, -1, sizeof(son)) ;
468                    }
469
470                }
471
472            }
473
474        }
475
476    }
477
478    ColorSeg operator+(const ColorSeg &o)
479        const {
480        if(tot == 0) return o ;
481        if(o.tot == 0) return *this ;
482
483        ColorSeg tmp ;
484        tmp.left = left ;
485        tmp.right = o.right ;
486        tmp.tot = tot + o.tot - (right ==
487            o.left) ;
488
489        return tmp ;
490    }
491
492    struct Node{
493        ColorSeg color ;
494        int tag ;
495    }seg[Maxn << 2] ;
496
497    class SegmentTree{
498    private:
499        void pull(int id){
500            // normal pull
501        }
502
503        void AddTag(int id, int tag){
504            // normal AddTag
505        }
506
507        void push(int id){
508            // normal push
509        }
510
511        void modify(int l, int r, int tag, int
512                    L=1, int R=n, int id=1){
513            // normal modify
514        }
515
516        ColorSeg query(int l, int r, int L=1, int
517                       R=n, int id=1){
518            // normal query
519        }
520
521        public:
522        void build(int L=1, int R=n, int id=1){
523            // normal build
524
525
526            // update val from u to v (simple path)
527            void update(int u, int v, int val){
528                while(top[u] != top[v]){
529                    if(dep[top[u]] < dep[top[v]]) swap(u,
530                        v) ;
531                    modify(dfn[top[u]], dfn[u], val) ;
532                    u = fa[top[u]] ;
533
534                    if(dep[u] < dep[v]) swap(u, v) ;
535
536                    // get sum from u to v (simple path)
537                    pair<int, ColorSeg> U, V ;
538                    ColorSeg M ;
539                    U = {u, {0, 0, 0}} ;
540                    V = {v, {0, 0, 0}} ;
541
542                    while(top[U.first] != top[V.first]){
543                        if(dep[top[U.first]] <
544                            dep[top[V.first]]) swap(U, V) ;
545                        U.second = query(dfn[top[U.first]],
546                                          dfn[U.first]) + U.second ;
547                        U.first = fa[top[U.first]] ;
548
549                        if(dep[U.first] < dep[V.first]) swap(U,
550                            V) ;
551
552                        M = query(dfn[V.first], dfn[U.first]) ;
553                    }
554
555                    void init(){
556                        memset(son, -1, sizeof(son)) ;
557                    }
558
559                }
560
561            }
562
563        }
564
565    }
566
567    ColorSeg operator+(const ColorSeg &o)
568        const {
569        if(tot == 0) return o ;
570        if(o.tot == 0) return *this ;
571
572        ColorSeg tmp ;
573        tmp.left = left ;
574        tmp.right = o.right ;
575        tmp.tot = tot + o.tot - (right ==
576            o.left) ;
577
578        return tmp ;
579    }
580
581    struct Node{
582        ColorSeg color ;
583        int tag ;
584    }seg[Maxn << 2] ;
585
586    class SegmentTree{
587    private:
588        void pull(int id){
589            // normal pull
590        }
591
592        void AddTag(int id, int tag){
593            // normal AddTag
594        }
595
596        void push(int id){
597            // normal push
598        }
599
600        void modify(int l, int r, int tag, int
601                    L=1, int R=n, int id=1){
602            // normal modify
603        }
604
605        ColorSeg query(int l, int r, int L=1, int
606                       R=n, int id=1){
607            // normal query
608        }
609
610        public:
611        void build(int L=1, int R=n, int id=1){
612            // normal build
613
614
615            // update val from u to v (simple path)
616            void update(int u, int v, int val){
617                while(top[u] != top[v]){
618                    if(dep[top[u]] < dep[top[v]]) swap(u,
619                        v) ;
620                    modify(dfn[top[u]], dfn[u], val) ;
621                    u = fa[top[u]] ;
622
623                    if(dep[u] < dep[v]) swap(u, v) ;
624
625                    // get sum from u to v (simple path)
626                    pair<int, ColorSeg> U, V ;
627                    ColorSeg M ;
628                    U = {u, {0, 0, 0}} ;
629                    V = {v, {0, 0, 0}} ;
630
631                    while(top[U.first] != top[V.first]){
632                        if(dep[top[U.first]] <
633                            dep[top[V.first]]) swap(U, V) ;
634                        U.second = query(dfn[top[U.first]],
635                                          dfn[U.first]) + U.second ;
636                        U.first = fa[top[U.first]] ;
637
638                        if(dep[U.first] < dep[V.first]) swap(U,
639                            V) ;
640
641                        M = query(dfn[V.first], dfn[U.first]) ;
642                    }
643
644                    void init(){
645                        memset(son, -1, sizeof(son)) ;
646                    }
647
648                }
649
650            }
651
652        }
653
654    }
655
656    ColorSeg operator+(const ColorSeg &o)
657        const {
658        if(tot == 0) return o ;
659        if(o.tot == 0) return *this ;
660
661        ColorSeg tmp ;
662        tmp.left = left ;
663        tmp.right = o.right ;
664        tmp.tot = tot + o.tot - (right ==
665            o.left) ;
666
667        return tmp ;
668    }
669
670    struct Node{
671        ColorSeg color ;
672        int tag ;
673    }seg[Maxn << 2] ;
674
675    class SegmentTree{
676    private:
677        void pull(int id){
678            // normal pull
679        }
680
681        void AddTag(int id, int tag){
682            // normal AddTag
683        }
684
685        void push(int id){
686            // normal push
687        }
688
689        void modify(int l, int r, int tag, int
690                    L=1, int R=n, int id=1){
691            // normal modify
692        }
693
694        ColorSeg query(int l, int r, int L=1, int
695                       R=n, int id=1){
696            // normal query
697        }
698
699        public:
700        void build(int L=1, int R=n, int id=1){
701            // normal build
702
703
704            // update val from u to v (simple path)
705            void update(int u, int v, int val){
706                while(top[u] != top[v]){
707                    if(dep[top[u]] < dep[top[v]]) swap(u,
708                        v) ;
709                    modify(dfn[top[u]], dfn[u], val) ;
710                    u = fa[top[u]] ;
711
712                    if(dep[u] < dep[v]) swap(u, v) ;
713
714                    // get sum from u to v (simple path)
715                    pair<int, ColorSeg> U, V ;
716                    ColorSeg M ;
717                    U = {u, {0, 0, 0}} ;
718                    V = {v, {0, 0, 0}} ;
719
720                    while(top[U.first] != top[V.first]){
721                        if(dep[top[U.first]] <
722                            dep[top[V.first]]) swap(U, V) ;
723                        U.second = query(dfn[top[U.first]],
724                                          dfn[U.first]) + U.second ;
725                        U.first = fa[top[U.first]] ;
726
727                        if(dep[U.first] < dep[V.first]) swap(U,
728                            V) ;
729
730                        M = query(dfn[V.first], dfn[U.first]) ;
731                    }
732
733                    void init(){
734                        memset(son, -1, sizeof(son)) ;
735                    }
736
737                }
738
739            }
740
741        }
742
743    }
744
745    ColorSeg operator+(const ColorSeg &o)
746        const {
747        if(tot == 0) return o ;
748        if(o.tot == 0) return *this ;
749
750        ColorSeg tmp ;
751        tmp.left = left ;
752        tmp.right = o.right ;
753        tmp.tot = tot + o.tot - (right ==
754            o.left) ;
755
756        return tmp ;
757    }
758
759    struct Node{
760        ColorSeg color ;
761        int tag ;
762    }seg[Maxn << 2] ;
763
764    class SegmentTree{
765    private:
766        void pull(int id){
767            // normal pull
768        }
769
770        void AddTag(int id, int tag){
771            // normal AddTag
772        }
773
774        void push(int id){
775            // normal push
776        }
777
778        void modify(int l, int r, int tag, int
779                    L=1, int R=n, int id=1){
780            // normal modify
781        }
782
783        ColorSeg query(int l, int r, int L=1, int
784                       R=n, int id=1){
785            // normal query
786        }
787
788        public:
789        void build(int L=1, int R=n, int id=1){
790            // normal build
791
792
793            // update val from u to v (simple path)
794            void update(int u, int v, int val){
795                while(top[u] != top[v]){
796                    if(dep[top[u]] < dep[top[v]]) swap(u,
797                        v) ;
798                    modify(dfn[top[u]], dfn[u], val) ;
799                    u = fa[top[u]] ;
800
801                    if(dep[u] < dep[v]) swap(u, v) ;
802
803                    // get sum from u to v (simple path)
804                    pair<int, ColorSeg> U, V ;
805                    ColorSeg M ;
806                    U = {u, {0, 0, 0}} ;
807                    V = {v, {0, 0, 0}} ;
808
809                    while(top[U.first] != top[V.first]){
810                        if(dep[top[U.first]] <
811                            dep[top[V.first]]) swap(U, V) ;
812                        U.second = query(dfn[top[U.first]],
813                                          dfn[U.first]) + U.second ;
814                        U.first = fa[top[U.first]] ;
815
816                        if(dep[U.first] < dep[V.first]) swap(U,
817                            V) ;
818
819                        M = query(dfn[V.first], dfn[U.first]) ;
820                    }
821
822                    void init(){
823                        memset(son, -1, sizeof(son)) ;
824                    }
825
826                }
827
828            }
829
830        }
831
832    }
833
834    ColorSeg operator+(const ColorSeg &o)
835        const {
836        if(tot == 0) return o ;
837        if(o.tot == 0) return *this ;
838
839        ColorSeg tmp ;
840        tmp.left = left ;
841        tmp.right = o.right ;
842        tmp.tot = tot + o.tot - (right ==
843            o.left) ;
844
845        return tmp ;
846    }
847
848    struct Node{
849        ColorSeg color ;
850        int tag ;
851    }seg[Maxn << 2] ;
852
853    class SegmentTree{
854    private:
855        void pull(int id){
856            // normal pull
857        }
858
859        void AddTag(int id, int tag){
860            // normal AddTag
861        }
862
863        void push(int id){
864            // normal push
865        }
866
867        void modify(int l, int r, int tag, int
868                    L=1, int R=n, int id=1){
869            // normal modify
870        }
871
872        ColorSeg query(int l, int r, int L=1, int
873                       R=n, int id=1){
874            // normal query
875        }
876
877        public:
878        void build(int L=1, int R=n, int id=1){
879            // normal build
880
881
882            // update val from u to v (simple path)
883            void update(int u, int v, int val){
884                while(top[u] != top[v]){
885                    if(dep[top[u]] < dep[top[v]]) swap(u,
886                        v) ;
887                    modify(dfn[top[u]], dfn[u], val) ;
888                    u = fa[top[u]] ;
889
890                    if(dep[u] < dep[v]) swap(u, v) ;
891
892                    // get sum from u to v (simple path)
893                    pair<int, ColorSeg> U, V ;
894                    ColorSeg M ;
895                    U = {u, {0, 0, 0}} ;
896                    V = {v, {0, 0, 0}} ;
897
898                    while(top[U.first] != top[V.first]){
899                        if(dep[top[U.first]] <
900                            dep[top[V.first]]) swap(U, V) ;
901                        U.second = query(dfn[top[U.first]],
902                                          dfn[U.first]) + U.second ;
903                        U.first = fa[top[U.first]] ;
904
905                        if(dep[U.first] < dep[V.first]) swap(U,
906                            V) ;
907
908                        M = query(dfn[V.first], dfn[U.first]) ;
909                    }
910
911                    void init(){
912                        memset(son, -1, sizeof(son)) ;
913                    }
914
915                }
916
917            }
918
919        }
920
921    }
922
923    ColorSeg operator+(const ColorSeg &o)
924        const {
925        if(tot == 0) return o ;
926        if(o.tot == 0) return *this ;
927
928        ColorSeg tmp ;
929        tmp.left = left ;
930        tmp.right = o.right ;
931        tmp.tot = tot + o.tot - (right ==
932            o.left) ;
933
934        return tmp ;
935    }
936
937    struct Node{
938        ColorSeg color ;
939        int tag ;
940    }seg[Maxn << 2] ;
941
942    class SegmentTree{
943    private:
944        void pull(int id){
945            // normal pull
946        }
947
948        void AddTag(int id, int tag){
949            // normal AddTag
950        }
951
952        void push(int id){
953            // normal push
954        }
955
956        void modify(int l, int r, int tag, int
957                    L=1, int R=n, int id=1){
958            // normal modify
959        }
960
961        ColorSeg query(int l, int r, int L=1, int
962                       R=n, int id=1){
963            // normal query
964        }
965
966        public:
967        void build(int L=1, int R=n, int id=1){
968            // normal build
969
970
971            // update val from u to v (simple path)
972            void update(int u, int v, int val){
973                while(top[u] != top[v]){
974                    if(dep[top[u]] < dep[top[v]]) swap(u,
975                        v) ;
976                    modify(dfn[top[u]], dfn[u], val) ;
977                    u = fa[top[u]] ;
978
979                    if(dep[u] < dep[v]) swap(u, v) ;
980
981                    // get sum from u to v (simple path)
982                    pair<int, ColorSeg> U, V ;
983                    ColorSeg M ;
984                    U = {u, {0, 0, 0}} ;
985                    V = {v, {0, 0, 0}} ;
986
987                    while(top[U.first] != top[V.first]){
988                        if(dep[top[U.first]] <
989                            dep[top[V.first]]) swap(U, V) ;
990                        U.second = query(dfn[top[U.first]],
991                                          dfn[U.first]) + U.second ;
992                        U.first = fa[top[U.first]] ;
993
994                        if(dep[U.first] < dep[V.first]) swap(U,
995                            V) ;
996
997                        M = query(dfn[V.first], dfn[U.first]) ;
998                    }
999
1000                    void init(){
1001                        memset(son, -1, sizeof(son)) ;
1002                    }
1003
1004                }
1005
1006            }
1007
1008        }
1009
1010    }
1011
1012    ColorSeg operator+(const ColorSeg &o)
1013        const {
1014        if(tot == 0) return o ;
1015        if(o.tot == 0) return *this ;
1016
1017        ColorSeg tmp ;
1018        tmp.left = left ;
1019        tmp.right = o.right ;
1020        tmp.tot = tot + o.tot - (right ==
1021            o.left) ;
1022
1023        return tmp ;
1024    }
1025
1026    struct Node{
1027        ColorSeg color ;
1028        int tag ;
1029    }seg[Maxn << 2] ;
1030
1031    class SegmentTree{
1032    private:
1033        void pull(int id){
1034            // normal pull
1035        }
1036
1037        void AddTag(int id, int tag){
1038            // normal AddTag
1039        }
1040
1041        void push(int id){
1042            // normal push
1043        }
1044
1045        void modify(int l, int r, int tag, int
1046                    L=1, int R=n, int id=1){
1047            // normal modify
1048        }
1049
1050        ColorSeg query(int l, int r, int L=1, int
1051                       R=n, int id=1){
1052            // normal query
1053        }
1054
1055        public:
1056        void build(int L=1, int R=n, int id=1){
1057            // normal build
1058
1059
1060            // update val from u to v (simple path)
1061            void update(int u, int v, int val){
1062                while(top[u] != top[v]){
1063                    if(dep[top[u]] < dep[top[v]]) swap(u,
1064                        v) ;
1065                    modify(dfn[top[u]], dfn[u], val) ;
1066                    u = fa[top[u]] ;
1067
1068                    if(dep[u] < dep[v]) swap(u, v) ;
1069
1070                    // get sum from u to v (simple path)
1071                    pair<int, ColorSeg> U, V ;
1072                    ColorSeg M ;
1073                    U = {u, {0, 0, 0}} ;
1074                    V = {v, {0, 0, 0}} ;
1075
1076                    while(top[U.first] != top[V.first]){
1077                        if(dep[top[U.first]] <
1078                            dep[top[V.first]]) swap(U, V) ;
1079                        U.second = query(dfn[top[U.first]],
1080                                          dfn[U.first]) + U.second ;
1081                        U.first = fa[top[U.first]] ;
1082
1083                        if(dep[U.first] < dep[V.first]) swap(U,
1084                            V) ;
1085
1086                        M = query(dfn[V.first], dfn[U.first]) ;
1087                    }
1088
1089                    void init(){
1090                        memset(son, -1, sizeof(son)) ;
1091                    }
1092
1093                }
1094
1095            }
1096
1097        }
1098
1099    }
1100
1101    ColorSeg operator+(const ColorSeg &o)
1102        const {
1103        if(tot == 0) return o ;
1104        if(o.tot == 0) return *this ;
1105
1106        ColorSeg tmp ;
1107        tmp.left = left ;
1108        tmp.right = o.right ;
1109        tmp.tot = tot + o.tot - (right ==
1110            o.left) ;
1111
1112        return tmp ;
1113    }
1114
1115    struct Node{
1116        ColorSeg color ;
1117        int tag ;
1118    }seg[Maxn << 2] ;
1119
1120    class SegmentTree{
1121    private:
1122        void pull(int id){
1123            // normal pull
1124        }
1125
1126        void AddTag(int id, int tag){
1127            // normal AddTag
1128        }
1129
1130        void push(int id){
1131            // normal push
1132        }
1133
1134        void modify(int l, int r, int tag, int
1135                    L=1, int R=n, int id=1){
1136            // normal modify
1137        }
1138
1139        ColorSeg query(int l, int r, int L=1, int
1140                       R=n, int id=1){
1141            // normal query
1142        }
1143
1144        public:
1145        void build(int L=1, int R=n, int id=1){
1146            // normal build
1147
1148
1149            // update val from u to v (simple path)
1150            void update(int u, int v, int val){
1151                while(top[u] != top[v]){
1152                    if(dep[top[u]] < dep[top[v]]) swap(u,
1153                        v) ;
1154                    modify(dfn[top[u]], dfn[u], val) ;
1155                    u = fa[top[u]] ;
1156
1157                    if(dep[u] < dep[v]) swap(u, v) ;
1158
1159                    // get sum from u to v (simple path)
1160                    pair<int, ColorSeg> U, V ;
1161                    ColorSeg M ;
1162                    U = {u, {0, 0, 0}} ;
1163                    V = {v, {0, 0, 0}} ;
1164
1165                    while(top[U.first] != top[V.first]){
1166                        if(dep[top[U.first]] <
1167                            dep[top[V.first]]) swap(U, V) ;
1168                        U.second = query(dfn[top[U.first]],
1169                                          dfn[U.first]) + U.second ;
1170                        U.first = fa[top[U.first]] ;
1171
1172                        if(dep[U.first] < dep[V.first]) swap(U,
1173                            V) ;
1174
1175                        M = query(dfn[V.first], dfn[U.first]) ;
1176                    }
1177
1178                    void init(){
1179                        memset(son, -1, sizeof(son)) ;
1180                    }
1181
1182                }
1183
1184            }
1185
1186        }
1187
1188    }
1189
1190    ColorSeg operator+(const ColorSeg &o)
1191        const {
1192        if(tot == 0) return o ;
1193        if(o.tot == 0) return *this ;
1194
1195        ColorSeg tmp ;
1196        tmp.left = left ;
1197        tmp.right = o.right ;
1198        tmp.tot = tot + o.tot - (right ==
1199            o.left) ;
1200
1201        return tmp ;
1202    }
1203
1204    struct Node{
1205        ColorSeg color ;
1206        int tag ;
1207    }seg[Maxn << 2] ;
1208
1209    class SegmentTree{
1210    private:
1211        void pull(int id){
1212            // normal pull
1213        }
1214
1215        void AddTag(int id
```

## 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 }

```

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

## 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         if(n < 10) dig.push_back(n) ;
36         else{
37             int t = n / 10 ;
38             n = n % 10 ;
39             if(t > 9) continue ;
40             if(dig[t] == 0) dig.push_back(t) ;
41         }
42     }
43
44     sort(dig.begin(), dig.end()) ;
45 }
```

```

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", &a, &b, &K) ;
46         if(K > 90) printf("0\n") ;
47         else printf("%d\n", count(b) -
48             count(a-1)) ;
49     }
50
51     return 0 ;
52 }
```

## 5.3 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
31     dp[u][1][0] = dp[u][1][1] = 0 ;
32
33     for ( auto [v, w] : edge[u] ){
34         for ( int i=cnt[u] ; i>1 ; i-- ) for (
35             int j=1 ; j<i && j<=cnt[v] ; j++ ){
36             dp[u][i][1] = min(dp[u][i][1],
37                 dp[u][i-j][1] + dp[v][j][1] + 2 *
38                     w) ;
39             dp[u][i][0] = min(dp[u][i][0],
40                 dp[u][i-j][1] + dp[v][j][0] + w) ;
41             dp[u][i][0] = min(dp[u][i][0],
42                 dp[u][i-j][0] + dp[v][j][1] + 2 *
43                     w) ;
44         }
45     }
46 }
```