

## Contents

1 Basic	1
1.1 PyMath . . . . .	1
1.2 Java Integer . . . . .	1
1.3 Java String . . . . .	1
1.4 Java String builder . . . . .	2
1.5 Java Math . . . . .	2
2 Math	2
2.1 formula . . . . .	2
2.2 extended gcd . . . . .	2
3 Tree	2
3.1 SegmentTree . . . . .	2
3.2 HLD . . . . .	3
3.3 PST . . . . .	3
4 Graph	4
4.1 cut vertex AND bridges . . . . .	4
4.2 SCC - Tarjan . . . . .	4
4.3 BCC - Tarjan . . . . .	4
4.4 Convex . . . . .	4
5 String	5
5.1 KMP . . . . .	5
5.2 Trie . . . . .	5
5.3 ACAM . . . . .	5
6 Algorithm	6
6.1 LCA . . . . .	6
6.2 MST . . . . .	6
6.3 SG . . . . .	6
6.4 Max Flow . . . . .	6
6.5 min cut max flow . . . . .	7
7 DP	7
7.1 輪廓線 DP . . . . .	7
7.2 數位 DP . . . . .	7
7.3 樹 DP . . . . .	7

## 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
9     math.gcd(x, y)
10    math.exp(x) # e^x
11    math.log(x, base)
12    math.log2(x)
13    math.log10(x)
14    math.sqrt(x)
15    math.pow(x, y, mod)
16    math.sin(x) # cos, tan, asin, acos, atan,
17      atan2, sinh ...
18
19    math.hypot(x, y) #歐幾里德範數
20
21    math.degrees(x) #x從弧度轉角度
22    math.radians(x) #x從角度轉弧度
23
24    math.gamma(x) #x的gamma函數
25
26    math.pi #const
27    math.e #const
28    math.inf

```

### 1.2 Java Integer

```

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

```

```

41 static int remainderUnsigned(int dividend,
      int divisor)

```

### 1.3 Java String

```

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

```

```

51 static String
52     valueOf(primitive/char[]/Object)
53 static String copyValueOf(char[] data[, int
      offset, int count])

```

## 1.4 Java String builder

```

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

```

## 1.5 Java Math

```

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

```

```

33 static double log10(double a)
34 static double log1p(double x)
35
36 // 三角/雙曲
37 static double sin/cos/tan(double a)
38 static double asin/acos/atan(double a)
39 static double atan2(double y, double x)
40 static double sinh/cosh/tanh(double a)
41
42 // 其他實用
43 static double hypot(double x, double y)
44 static double toDegrees(double angrad)
45 static double toRadians(double angdeg)
46 static double copySign(double magnitude,
   double sign)
47 static double nextUp/nextDown(double a)
48 static double nextAfter(double start, double
   direction)
49 static double ulp(double d)
50 static double random()
51 static double scalb(double d, int
   scaleFactor)
52 static double fma(double a, double b, double
   c) // (since 8)
53 static long multiplyHigh(long x, long y)
   // (since 9)
54 static long multiplyFull(int x, int y)
   // (since 9, 回傳 long)

```

## 2 Math

### 2.1 formula

#### 1. Catalan Number

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

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

#### 2. Euler's Formula

對於  $v$  個點,  $e$  條邊,  $f$  個面,  $c$  個連通分量

$$V + F = E + 2$$

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

#### 3. Pick's Theorem

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

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

### 2.2 extended gcd

給定  $a, b, c$ , 求  $ax + by = c$  的解

```

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

```

## 3 Tree

### 3.1 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
38     void push(int id){
39         AddTag(lc, seg[id].tag) ;
40         AddTag(rc, seg[id].tag) ;
41         seg[id].tag = {0, 0} ;
42     }
43
44 public:
45     void build(int L=1, int R=n, int id=1){
46         seg[id].sum = 0 ;
47         seg[id].tag = {0, 0} ;
48         seg[id].sz = 1 ;
49
50         if(L == R){
51             seg[id].sum = arr[L] ;
52             return ;
53         }
54
55         int M = (L + R) >> 1 ;
56         build(L, M, lc) ;
57         build(M+1, R, rc) ;
58
59         pull(id) ;
60         seg[id].sz = seg[lc].sz + seg[rc].sz ;
61     }
62
63     void modify(int l, int r, LazyTag &tag,
64         int L=1, int R=n, int id=1){
65         if(l <= L && R <= r){
66             AddTag(id, tag) ;
67             return ;
68         }
69
70         push(id) ;
71         int M = (L + R) >> 1 ;
72         if(r <= M) modify(l, r, tag, L, M,
73             lc) ;
74     }
75
76     push(id) ;
77     int M = (L + R) >> 1 ;
78     if(r <= M) modify(l, r, tag, L, M,
79             lc) ;
80
81     pull(id) ;
82     seg[id].sum = seg[lc].sum + seg[rc].sum ;
83     seg[id].tag = {0, 0} ;
84 }
85
86     seg[id].tag = {0, 0} ;
87 }
88
89     seg[id].tag = {0, 0} ;
90 }
91
92     seg[id].tag = {0, 0} ;
93 }
94
95     seg[id].tag = {0, 0} ;
96 }
97
98     seg[id].tag = {0, 0} ;
99 }
100
101     seg[id].tag = {0, 0} ;
102 }
103
104     seg[id].tag = {0, 0} ;
105 }
106
107     seg[id].tag = {0, 0} ;
108 }
109
110     seg[id].tag = {0, 0} ;
111 }
112
113     seg[id].tag = {0, 0} ;
114 }
115
116     seg[id].tag = {0, 0} ;
117 }
118
119     seg[id].tag = {0, 0} ;
120 }
121
122     seg[id].tag = {0, 0} ;
123 }
124
125     seg[id].tag = {0, 0} ;
126 }
127
128     seg[id].tag = {0, 0} ;
129 }
130
131     seg[id].tag = {0, 0} ;
132 }
133
134     seg[id].tag = {0, 0} ;
135 }
136
137     seg[id].tag = {0, 0} ;
138 }
139
140     seg[id].tag = {0, 0} ;
141 }
142
143     seg[id].tag = {0, 0} ;
144 }
145
146     seg[id].tag = {0, 0} ;
147 }
148
149     seg[id].tag = {0, 0} ;
150 }
151
152     seg[id].tag = {0, 0} ;
153 }
154
155     seg[id].tag = {0, 0} ;
156 }
157
158     seg[id].tag = {0, 0} ;
159 }
160
161     seg[id].tag = {0, 0} ;
162 }
163
164     seg[id].tag = {0, 0} ;
165 }
166
167     seg[id].tag = {0, 0} ;
168 }
169
170     seg[id].tag = {0, 0} ;
171 }
172
173     seg[id].tag = {0, 0} ;
174 }
175
176     seg[id].tag = {0, 0} ;
177 }
178
179     seg[id].tag = {0, 0} ;
180 }
181
182     seg[id].tag = {0, 0} ;
183 }
184
185     seg[id].tag = {0, 0} ;
186 }
187
188     seg[id].tag = {0, 0} ;
189 }
190
191     seg[id].tag = {0, 0} ;
192 }
193
194     seg[id].tag = {0, 0} ;
195 }
196
197     seg[id].tag = {0, 0} ;
198 }
199
199     seg[id].tag = {0, 0} ;
200 }
201
201     seg[id].tag = {0, 0} ;
202 }
203
203     seg[id].tag = {0, 0} ;
204 }
205
205     seg[id].tag = {0, 0} ;
206 }
207
207     seg[id].tag = {0, 0} ;
208 }
209
209     seg[id].tag = {0, 0} ;
210 }
211
211     seg[id].tag = {0, 0} ;
212 }
213
213     seg[id].tag = {0, 0} ;
214 }
215
215     seg[id].tag = {0, 0} ;
216 }
217
217     seg[id].tag = {0, 0} ;
218 }
219
219     seg[id].tag = {0, 0} ;
220 }
221
221     seg[id].tag = {0, 0} ;
222 }
223
223     seg[id].tag = {0, 0} ;
224 }
225
225     seg[id].tag = {0, 0} ;
226 }
227
227     seg[id].tag = {0, 0} ;
228 }
229
229     seg[id].tag = {0, 0} ;
230 }
231
231     seg[id].tag = {0, 0} ;
232 }
233
233     seg[id].tag = {0, 0} ;
234 }
235
235     seg[id].tag = {0, 0} ;
236 }
237
237     seg[id].tag = {0, 0} ;
238 }
239
239     seg[id].tag = {0, 0} ;
240 }
241
241     seg[id].tag = {0, 0} ;
242 }
243
243     seg[id].tag = {0, 0} ;
244 }
245
245     seg[id].tag = {0, 0} ;
246 }
247
247     seg[id].tag = {0, 0} ;
248 }
249
249     seg[id].tag = {0, 0} ;
250 }
251
251     seg[id].tag = {0, 0} ;
252 }
253
253     seg[id].tag = {0, 0} ;
254 }
255
255     seg[id].tag = {0, 0} ;
256 }
257
257     seg[id].tag = {0, 0} ;
258 }
259
259     seg[id].tag = {0, 0} ;
260 }
261
261     seg[id].tag = {0, 0} ;
262 }
263
263     seg[id].tag = {0, 0} ;
264 }
265
265     seg[id].tag = {0, 0} ;
266 }
267
267     seg[id].tag = {0, 0} ;
268 }
269
269     seg[id].tag = {0, 0} ;
270 }
271
271     seg[id].tag = {0, 0} ;
272 }
273
273     seg[id].tag = {0, 0} ;
274 }
275
275     seg[id].tag = {0, 0} ;
276 }
277
277     seg[id].tag = {0, 0} ;
278 }
279
279     seg[id].tag = {0, 0} ;
280 }
281
281     seg[id].tag = {0, 0} ;
282 }
283
283     seg[id].tag = {0, 0} ;
284 }
285
285     seg[id].tag = {0, 0} ;
286 }
287
287     seg[id].tag = {0, 0} ;
288 }
289
289     seg[id].tag = {0, 0} ;
290 }
291
291     seg[id].tag = {0, 0} ;
292 }
293
293     seg[id].tag = {0, 0} ;
294 }
295
295     seg[id].tag = {0, 0} ;
296 }
297
297     seg[id].tag = {0, 0} ;
298 }
299
299     seg[id].tag = {0, 0} ;
300 }
301
301     seg[id].tag = {0, 0} ;
302 }
303
303     seg[id].tag = {0, 0} ;
304 }
305
305     seg[id].tag = {0, 0} ;
306 }
307
307     seg[id].tag = {0, 0} ;
308 }
309
309     seg[id].tag = {0, 0} ;
310 }
311
311     seg[id].tag = {0, 0} ;
312 }
313
313     seg[id].tag = {0, 0} ;
314 }
315
315     seg[id].tag = {0, 0} ;
316 }
317
317     seg[id].tag = {0, 0} ;
318 }
319
319     seg[id].tag = {0, 0} ;
320 }
321
321     seg[id].tag = {0, 0} ;
322 }
323
323     seg[id].tag = {0, 0} ;
324 }
325
325     seg[id].tag = {0, 0} ;
326 }
327
327     seg[id].tag = {0, 0} ;
328 }
329
329     seg[id].tag = {0, 0} ;
330 }
331
331     seg[id].tag = {0, 0} ;
332 }
333
333     seg[id].tag = {0, 0} ;
334 }
335
335     seg[id].tag = {0, 0} ;
336 }
337
337     seg[id].tag = {0, 0} ;
338 }
339
339     seg[id].tag = {0, 0} ;
340 }
341
341     seg[id].tag = {0, 0} ;
342 }
343
343     seg[id].tag = {0, 0} ;
344 }
345
345     seg[id].tag = {0, 0} ;
346 }
347
347     seg[id].tag = {0, 0} ;
348 }
349
349     seg[id].tag = {0, 0} ;
350 }
351
351     seg[id].tag = {0, 0} ;
352 }
353
353     seg[id].tag = {0, 0} ;
354 }
355
355     seg[id].tag = {0, 0} ;
356 }
357
357     seg[id].tag = {0, 0} ;
358 }
359
359     seg[id].tag = {0, 0} ;
360 }
361
361     seg[id].tag = {0, 0} ;
362 }
363
363     seg[id].tag = {0, 0} ;
364 }
365
365     seg[id].tag = {0, 0} ;
366 }
367
367     seg[id].tag = {0, 0} ;
368 }
369
369     seg[id].tag = {0, 0} ;
370 }
371
371     seg[id].tag = {0, 0} ;
372 }
373
373     seg[id].tag = {0, 0} ;
374 }
375
375     seg[id].tag = {0, 0} ;
376 }
377
377     seg[id].tag = {0, 0} ;
378 }
379
379     seg[id].tag = {0, 0} ;
380 }
381
381     seg[id].tag = {0, 0} ;
382 }
383
383     seg[id].tag = {0, 0} ;
384 }
385
385     seg[id].tag = {0, 0} ;
386 }
387
387     seg[id].tag = {0, 0} ;
388 }
389
389     seg[id].tag = {0, 0} ;
390 }
391
391     seg[id].tag = {0, 0} ;
392 }
393
393     seg[id].tag = {0, 0} ;
394 }
395
395     seg[id].tag = {0, 0} ;
396 }
397
397     seg[id].tag = {0, 0} ;
398 }
399
399     seg[id].tag = {0, 0} ;
400 }
401
401     seg[id].tag = {0, 0} ;
402 }
403
403     seg[id].tag = {0, 0} ;
404 }
405
405     seg[id].tag = {0, 0} ;
406 }
407
407     seg[id].tag = {0, 0} ;
408 }
409
409     seg[id].tag = {0, 0} ;
410 }
411
411     seg[id].tag = {0, 0} ;
412 }
413
413     seg[id].tag = {0, 0} ;
414 }
415
415     seg[id].tag = {0, 0} ;
416 }
417
417     seg[id].tag = {0, 0} ;
418 }
419
419     seg[id].tag = {0, 0} ;
420 }
421
421     seg[id].tag = {0, 0} ;
422 }
423
423     seg[id].tag = {0, 0} ;
424 }
425
425     seg[id].tag = {0, 0} ;
426 }
427
427     seg[id].tag = {0, 0} ;
428 }
429
429     seg[id].tag = {0, 0} ;
430 }
431
431     seg[id].tag = {0, 0} ;
432 }
433
433     seg[id].tag = {0, 0} ;
434 }
435
435     seg[id].tag = {0, 0} ;
436 }
437
437     seg[id].tag = {0, 0} ;
438 }
439
439     seg[id].tag = {0, 0} ;
440 }
441
441     seg[id].tag = {0, 0} ;
442 }
443
443     seg[id].tag = {0, 0} ;
444 }
445
445     seg[id].tag = {0, 0} ;
446 }
447
447     seg[id].tag = {0, 0} ;
448 }
449
449     seg[id].tag = {0, 0} ;
450 }
451
451     seg[id].tag = {0, 0} ;
452 }
453
453     seg[id].tag = {0, 0} ;
454 }
455
455     seg[id].tag = {0, 0} ;
456 }
457
457     seg[id].tag = {0, 0} ;
458 }
459
459     seg[id].tag = {0, 0} ;
460 }
461
461     seg[id].tag = {0, 0} ;
462 }
463
463     seg[id].tag = {0, 0} ;
464 }
465
465     seg[id].tag = {0, 0} ;
466 }
467
467     seg[id].tag = {0, 0} ;
468 }
469
469     seg[id].tag = {0, 0} ;
470 }
471
471     seg[id].tag = {0, 0} ;
472 }
473
473     seg[id].tag = {0, 0} ;
474 }
475
475     seg[id].tag = {0, 0} ;
476 }
477
477     seg[id].tag = {0, 0} ;
478 }
479
479     seg[id].tag = {0, 0} ;
480 }
481
481     seg[id].tag = {0, 0} ;
482 }
483
483     seg[id].tag = {0, 0} ;
484 }
485
485     seg[id].tag = {0, 0} ;
486 }
487
487     seg[id].tag = {0, 0} ;
488 }
489
489     seg[id].tag = {0, 0} ;
490 }
491
491     seg[id].tag = {0, 0} ;
492 }
493
493     seg[id].tag = {0, 0} ;
494 }
495
495     seg[id].tag = {0, 0} ;
496 }
497
497     seg[id].tag = {0, 0} ;
498 }
499
499     seg[id].tag = {0, 0} ;
500 }
501
501     seg[id].tag = {0, 0} ;
502 }
503
503     seg[id].tag = {0, 0} ;
504 }
505
505     seg[id].tag = {0, 0} ;
506 }
507
507     seg[id].tag = {0, 0} ;
508 }
509
509     seg[id].tag = {0, 0} ;
510 }
511
511     seg[id].tag = {0, 0} ;
512 }
513
513     seg[id].tag = {0, 0} ;
514 }
515
515     seg[id].tag = {0, 0} ;
516 }
517
517     seg[id].tag = {0, 0} ;
518 }
519
519     seg[id].tag = {0, 0} ;
520 }
521
521     seg[id].tag = {0, 0} ;
522 }
523
523     seg[id].tag = {0, 0} ;
524 }
525
525     seg[id].tag = {0, 0} ;
526 }
527
527     seg[id].tag = {0, 0} ;
528 }
529
529     seg[id].tag = {0, 0} ;
530 }
531
531     seg[id].tag = {0, 0} ;
532 }
533
533     seg[id].tag = {0, 0} ;
534 }
535
535     seg[id].tag = {0, 0} ;
536 }
537
537     seg[id].tag = {0, 0} ;
538 }
539
539     seg[id].tag = {0, 0} ;
540 }
541
541     seg[id].tag = {0, 0} ;
542 }
543
543     seg[id].tag = {0, 0} ;
544 }
545
545     seg[id].tag = {0, 0} ;
546 }
547
547     seg[id].tag = {0, 0} ;
548 }
549
549     seg[id].tag = {0, 0} ;
550 }
551
551     seg[id].tag = {0, 0} ;
552 }
553
553     seg[id].tag = {0, 0} ;
554 }
555
555     seg[id].tag = {0, 0} ;
556 }
557
557     seg[id].tag = {0, 0} ;
558 }
559
559     seg[id].tag = {0, 0} ;
560 }
561
561     seg[id].tag = {0, 0} ;
562 }
563
563     seg[id].tag = {0, 0} ;
564 }
565
565     seg[id].tag = {0, 0} ;
566 }
567
567     seg[id].tag = {0, 0} ;
568 }
569
569     seg[id].tag = {0, 0} ;
570 }
571
571     seg[id].tag = {0, 0} ;
572 }
573
573     seg[id].tag = {0, 0} ;
574 }
575
575     seg[id].tag = {0, 0} ;
576 }
577
577     seg[id].tag = {0, 0} ;
578 }
579
579     seg[id].tag = {0, 0} ;
580 }
581
581     seg[id].tag = {0, 0} ;
582 }
583
583     seg[id].tag = {0, 0} ;
584 }
585
585     seg[id].tag = {0, 0} ;
586 }
587
587     seg[id].tag = {0, 0} ;
588 }
589
589     seg[id].tag = {0, 0} ;
590 }
591
591     seg[id].tag = {0, 0} ;
592 }
593
593     seg[id].tag = {0, 0} ;
594 }
595
595     seg[id].tag = {0, 0} ;
596 }
597
597     seg[id].tag = {0, 0} ;
598 }
599
599     seg[id].tag = {0, 0} ;
600 }
601
601     seg[id].tag = {0, 0} ;
602 }
603
603     seg[id].tag = {0, 0} ;
604 }
605
605     seg[id].tag = {0, 0} ;
606 }
607
607     seg[id].tag = {0, 0} ;
608 }
609
609     seg[id].tag = {0, 0} ;
610 }
611
611     seg[id].tag = {0, 0} ;
612 }
613
613     seg[id].tag = {0, 0} ;
614 }
615
615     seg[id].tag = {0, 0} ;
616 }
617
617     seg[id].tag = {0, 0} ;
618 }
619
619     seg[id].tag = {0, 0} ;
620 }
621
621     seg[id].tag = {0, 0} ;
622 }
623
623     seg[id].tag = {0, 0} ;
624 }
625
625     seg[id].tag = {0, 0} ;
626 }
627
627     seg[id].tag = {0, 0} ;
628 }
629
629     seg[id].tag = {0, 0} ;
630 }
631
631     seg[id].tag = {0, 0} ;
632 }
633
633     seg[id].tag = {0, 0} ;
634 }
635
635     seg[id].tag = {0, 0} ;
636 }
637
637     seg[id].tag = {0, 0} ;
638 }
639
639     seg[id].tag = {0, 0} ;
640 }
641
641     seg[id].tag = {0, 0} ;
642 }
643
643     seg[id].tag = {0, 0} ;
644 }
645
645     seg[id].tag = {0, 0} ;
646 }
647
647     seg[id].tag = {0, 0} ;
648 }
649
649     seg[id].tag = {0, 0} ;
650 }
651
651     seg[id].tag = {0, 0} ;
652 }
653
653     seg[id].tag = {0, 0} ;
654 }
655
655     seg[id].tag = {0, 0} ;
656 }
657
657     seg[id].tag = {0, 0} ;
658 }
659
659     seg[id].tag = {0, 0} ;
660 }
661
661     seg[id].tag = {0, 0} ;
662 }
663
663     seg[id].tag = {0, 0} ;
664 }
665
665     seg[id].tag = {0, 0} ;
666 }
667
667     seg[id].tag = {0, 0} ;
668 }
669
669     seg[id].tag = {0, 0} ;
670 }
671
671     seg[id].tag = {0, 0} ;
672 }
673
673     seg[id].tag = {0, 0} ;
674 }
675
675     seg[id].tag = {0, 0} ;
676 }
677
677     seg[id].tag = {0, 0} ;
678 }
679
679     seg[id].tag = {0, 0} ;
680 }
681
681     seg[id].tag = {0, 0} ;
682 }
683
683     seg[id].tag = {0, 0} ;
684 }
685
685     seg[id].tag = {0, 0} ;
686 }
687
687     seg[id].tag = {0, 0} ;
688 }
689
689     seg[id].tag = {0, 0} ;
690 }
691
691     seg[id].tag = {0, 0} ;
692 }
693
693     seg[id].tag = {0, 0} ;
694 }
695
695     seg[id].tag = {0, 0} ;
696 }
697
697     seg[id].tag = {0, 0} ;
698 }
699
699     seg[id].tag = {0, 0} ;
700 }
701
701     seg[id].tag = {0, 0} ;
702 }
703
703     seg[id].tag = {0, 0} ;
704 }
705
705     seg[id].tag = {0, 0} ;
706 }
707
707     seg[id].tag = {0, 0} ;
708 }
709
709     seg[id].tag = {0, 0} ;
710 }
711
711     seg[id].tag = {0, 0} ;
712 }
713
713     seg[id].tag = {0, 0} ;
714 }
715
715     seg[id].tag = {0, 0} ;
716 }
717
717     seg[id].tag = {0, 0} ;
718 }
719
719     seg[id].tag = {0, 0} ;
720 }
721
721     seg[id].tag = {0, 0} ;
722 }
723
723     seg[id].tag = {0, 0} ;
724 }
725
725     seg[id].tag = {0, 0} ;
726 }
727
727     seg[id].tag = {0, 0} ;
728 }
729
729     seg[id].tag = {0, 0} ;
730 }
731
731     seg[id].tag = {0, 0} ;
732 }
733
733     seg[id].tag = {0, 0} ;
734 }
735
735     seg[id].tag = {0, 0} ;
736 }
737
737     seg[id].tag = {0, 0} ;
738 }
739
739     seg[id].tag = {0, 0} ;
740 }
741
741     seg[id].tag = {0, 0} ;
742 }
743
743     seg[id].tag = {0, 0} ;
744 }
745
745     seg[id].tag = {0, 0} ;
746 }
747
747     seg[id].tag = {0, 0} ;
748 }
749
749     seg[id].tag = {0, 0} ;
750 }
751
751     seg[id].tag = {0, 0} ;
752 }
753
753     seg[id].tag = {0, 0} ;
754 }
755
755     seg[id].tag = {0, 0} ;
756 }
757
757     seg[id].tag = {0, 0} ;
758 }
759
759     seg[id].tag = {0, 0} ;
760 }
761
761     seg[id].tag = {0, 0} ;
762 }
763
763     seg[id].tag = {0, 0} ;
764 }
765
765     seg[id].tag = {0, 0} ;
766 }
767
767     seg[id].tag = {0, 0} ;
768 }
769
769     seg[id].tag = {0, 0} ;
770 }
771
771     seg[id].tag = {0, 0} ;
772 }
773
773     seg[id].tag = {0, 0} ;
774 }
775
775     seg[id].tag = {0, 0} ;
776 }
777
777     seg[id].tag = {0, 0} ;
778 }
779
779     seg[id].tag = {0, 0} ;
780 }
781
781     seg[id].tag = {0, 0} ;
782 }
783
783     seg[id].tag = {0, 0} ;
784 }
785
785     seg[id].tag = {0, 0} ;
786 }
787
787     seg[id].tag = {0, 0} ;
788 }
789
789     seg[id].tag = {0, 0} ;
790 }
791
791     seg[id].tag = {0, 0} ;
792 }
793
793     seg[id].tag = {0, 0} ;
794 }
795
795     seg[id].tag = {0, 0} ;
796 }
797
797     seg[id].tag = {0, 0} ;
798 }
799
799     seg[id].tag = {0, 0} ;
800 }
801
801     seg[id].tag = {0, 0} ;
802 }
803
803     seg[id].tag = {0, 0} ;
804 }
805
805     seg[id].tag = {0, 0} ;
806 }
807
807     seg[id].tag = {0, 0} ;
808 }
809
809     seg[id].tag = {0, 0} ;
810 }
811
811     seg[id].tag = {0, 0} ;
812 }
813
813     seg[id].tag = {0, 0} ;
814 }
815
815     seg[id].tag = {0, 0} ;
816 }
817
817     seg[id].tag = {0, 0} ;
818 }
819
819     seg[id].tag = {0, 0} ;
820 }
821
821     seg[id].tag = {0, 0} ;
822 }
823
823     seg[id].tag = {0, 0} ;
824 }
825
825     seg[id].tag = {
```

```

69     else if(l > M) modify(l, r, tag, M+1,
70         R, rc) ;
71     else{
72         modify(l, r, tag, L, M, lc) ;
73         modify(l, r, tag, M+1, R, rc) ;
74     }
75     pull(id) ;
76 }
77 ll query(int l, int r, int L=1, int R=n,
78         int id=1){
79     if(l <= L && R <= r) return
80         seg[id].sum ;
81     push(id) ;
82     int M = (L + R) >> 1 ;
83     if(r <= M) return query(l, r, L, M,
84                             lc) ;
85     else if(l > M) return query(l, r,
86                                 M+1, R, rc) ;
87     else return query(l, r, L, M, lc) +
88             query(l, r, M+1, R, rc) ;
89 }
90 }tree ;

```

### 3.2 HLD

```

1 /* HLD */
2 int fa[Maxn], top[Maxn], son[Maxn],
3     sz[Maxn], dep[Maxn] = {0}, dfn[Maxn],
4     rnk[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]])
15            son[u] = v ;
16    }
17 }
18
19 void dfs2(int u, int t){
20     top[u] = t ;
21     dfn[u] = ++dfscnt ;
22     rnk[dfscnt] = u ;
23
24     if(son[u] == -1) return ;
25
26     for ( auto v : g[u] ) if(v != fa[u] && v
27         != son[u]){
28         dfs2(v, v) ;
29     }
30
31 /* Segment Tree */
32 #define lc (id << 1)
33 #define rc ((id << 1) | 1)
34
35 struct ColorSeg{
36     int left, right, tot ;
37
38     ColorSeg operator+(const ColorSeg &o)
39     const {
40         if(tot == 0) return o ;
41         if(o.tot == 0) return *this ;
42
43         ColorSeg tmp ;
44         tmp.left = left ;
45         tmp.right = o.right ;
46         tmp.tot = tot + o.tot - (right ==
47             o.left) ;

```

FJCUUUUUU

```

48     }
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
85         // update val from u to v (simple path)
86         void update(int u, int v, int val){
87             while(top[u] != top[v]){
88                 if(dep[top[u]] < dep[top[v]]) swap(u,
89                     v) ;
90                 modify(dfn[top[u]], dfn[u], val) ;
91                 u = fa[top[u]] ;
92             }
93
94             if(dep[u] < dep[v]) swap(u, v) ;
95             modify(dfn[v], dfn[u], val) ;
96
97             // get sum from u to v (simple path)
98             int get(int u, int v){
99                 pair<int, ColorSeg> U, V ;
100                ColorSeg M ;
101                U = {u, {0, 0, 0}} ;
102                V = {v, {0, 0, 0}} ;
103
104                while(top[U.first] != top[V.first]){
105                    if(dep[top[U.first]] <
106                        dep[top[V.first]]) swap(U, V) ;
107                    U.second = query(dfn[top[U.first]],
108                                     dfn[U.first]) + U.second ;
109                    U.first = fa[top[U.first]] ;
110
111                    if(dep[U.first] < dep[V.first]) swap(U,
112                        V) ;
113
114                    M = query(dfn[V.first], dfn[U.first]) ;
115
116                    return (U.second.tot + V.second.tot +
117                            M.tot) - (U.second.left == M.right)
118                            - (V.second.left == M.left) ;
119                }
120            }
121
122 }tree ;
123
124 void init(){
125
126     memset(son, -1, sizeof(son)) ;
127 }

```

### 3.3 PST

```

1 // Find range k-th largest number
2 struct Node{
3     int sum, left, right ;
4 }seg[Maxn + 20 * Maxn] ;
5
6 class PersistentSegmentTree{
7 private:
8     int n ;
9     int cnt ;
10    vector<int> version ;
11
12    int build(int L, int R){
13        int cur_cnt = cnt++ ;
14        if(L == R){
15            seg[cur_cnt] = {0, 0, 0} ;
16            return cur_cnt ;
17        }
18
19        int M = (L + R) >> 1 ;
20        int lc = build(L, M) ;
21        int rc = build(M+1, R) ;
22
23        seg[cur_cnt] = {0, lc, rc} ;
24        return cur_cnt ;
25    }
26
27 public:
28    PersistentSegmentTree(int _n){
29        n = _n ;
30        cnt = 0 ;
31
32        int root = build(1, n) ;
33        version.push_back(root) ;
34    }
35
36    void update(int ver, int idx){
37        auto upd = [&](auto &self, const int
38                      cur, int L, int R){
39            int cur_cnt = cnt++ ;
40
41            if(L == R){
42                seg[cur_cnt] = {seg[cur].sum + 1, 0,
43                               0} ;
44                return cur_cnt ;
45            }
46
47            int M = (L + R) >> 1 ;
48            int lc = seg[cur].left ;
49            int rc = seg[cur].right ;
50
51            if(idx <= M) lc = self(self,
52                                    seg[cur].left, L, M) ;
53            else rc = self(self, seg[cur].right,
54                           M+1, R) ;
55
56            seg[cur_cnt] = {seg[lc].sum +
57                           seg[rc].sum, lc, rc} ;
58
59            return cur_cnt ;
60        };
61
62        int root = upd(upd, version[ver], 1, n) ;
63        version.push_back(root) ;
64    }
65
66    int query(int verL, int verR, int k){
67        auto qry = [&](auto &self, const int
68                      cur_old, const int cur_new, int L,
69                      int R){
70            if(L == R) return L ;
71
72            int old_l = seg[cur_old].left, old_r =
73                         seg[cur_old].right ;

```

```

55     int new_l = seg[cur_new].left, new_r = 16
56         seg[cur_new].right ;
57
58     int dl = seg[new_l].sum - 17
59         seg[old_l].sum ;
60     int dr = seg[new_r].sum - 18
61         seg[old_r].sum ;
62
63     int M = (L + R) >> 1 ; 19
64
65     if(dl >= k) return self(self, old_l, 20
66         new_l, L, M) ;
67     k -= dl ;
68     return self(self, old_r, new_r, M+1, 21
69         R) ;
70 }
71
72     int idx = qry(qry, version[verL-1], 22
73         version[verR], 1, n) ;
74     return idx ;
75 }
76
77 } 23
78
79 }
```

## 4 Graph

### 4.1 cut vertex AND bridges

```

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

### 4.2 SCC - Tarjan

```

1 vector<int> scc[Maxn] ;
2 int dfn[Maxn], low[Maxn], sccId[Maxn], 16
3     dfscnt = 0, cnt_scc = 0 ;
4 stack<int> st ;
5 bitset<Maxn> inSt, vis ;
6
7 void dfs(int u, int from){ 17
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 }
```

```

18     if(dfn[u] == low[u]){
19         cnt_scc++ ;
20         int x ;
21         do{
22             x = st.top() ;
23             st.pop() ;
24
25             inSt[x] = 0 ;
26             sccId[x] = cnt_scc ;
27             scc[cnt_scc].push_back(x) ;
28         }
29         while(x != u) ;
30     }
31 }
32
33 // SCC to DAG (after dfs)
34 vector<int> dag[Maxn] ;
35
36 void scc_to_dag(){
37     vector<int> dag[Maxn] ;
38     for ( int u=1 ; u<=n ; u++ ){
39         for ( auto v : g[u] ){
40             if(sccId[u] != sccId[v]){
41                 dag[sccId[u]].push_back(sccId[v]) ;
42             }
43         }
44     }
45 }
46
47 void init(){
48     memset(dfn, -1, sizeof(dfn)) ;
49     memset(low, -1, sizeof(low)) ;
50 }
51
52 int main(){
53     init() ;
54     input() ;
55     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
56         == -1){
57         dfs(i, 0) ;
58     }
59 }
60
61 }
```

### 4.3 BCC - Tarjan

```

1 struct Edge{
2     int v, next ;
3 }e[Maxm << 1] ;
4 int head[Maxm], tot = 1 ;
5
6 void add(int u, int v){
7     e[++tot] = {v, head[u]} ;
8     head[u] = tot ;
9     e[++tot] = {u, head[v]} ;
10    head[v] = tot ;
11 }
12
13 bitset<Maxm << 1> bz ;
14 vector<vector<int> > bcc ;
15 int dfn_cnt = 0, dfn[Maxn], low[Maxn],
16     vis_bcc[Maxn], bcc_cnt = 0 ;
17
18 void dfs1(int u, int from){
19     dfn[u] = low[u] = ++dfn_cnt ;
20
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 }
```

```

28     else if(i != (from ^ 1)) low[u] =
29         min(low[u], dfn[v]) ;
30 }
31
32 void dfs2(int u, int id){
33     vis_bcc[u] = id ;
34     bcc[id].push_back(u) ;
35
36 for ( int i=head[u] ; i!= -1 ; i=e[i].next
37         ){
38     int v = e[i].v ;
39
40     if(vis_bcc[v] != -1 || bz[i]) continue ;
41     dfs2(v, id) ;
42 }
43
44 void init(){
45     memset(dfn, -1, sizeof(dfn)) ;
46     memset(head, -1, sizeof(head)) ;
47     memset(vis_bcc, -1, sizeof(vis_bcc)) ;
48 }
49
50 int main(){
51     init() ;
52     input() ;
53     for ( int i=1 ; i<=n ; i++ ) if(dfn[i]
54         == -1){
55         dfs1(i, 0) ;
56     }
57
58 for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i]
59         == -1){
60         bcc.push_back(vector<int>()) ;
61         dfs2(i, bcc_cnt++) ;
62     }
63 }
```

### 4.4 Convex

```

1 struct Coordinate{
2     long long x, y ;
3
4     friend bool operator<(const Coordinate&a,
5         const Coordinate& b){
6         if(a.x == b.x) return a.y < b.y ;
7         return a.x < b.x ;
8     }
9
10    friend bool operator==(const Coordinate&
11         a, const Coordinate& b){
12         return a.x == b.x && a.y == b.y ;
13     }
14
15    vector<Coordinate> nodes ;
16
17    long long cross(const Coordinate& o, const
18        Coordinate& a, const Coordinate& b){
19        return (a.x - o.x) * (b.y - o.y) - (a.y -
20            o.y) * (b.x - o.x) ;
21    }
22
23    void input(){
24        nodes.clear() ;
25
26        int n, x, y ;
27        char c ;
28        cin >> n ;
29
30        for ( int i=0 ; i<n ; i++ ){
31            cin >> x >> y >> c ;
32            if(c == 'Y') nodes.push_back({x, y}) ;
33        }
34    }
35 }
```

```
33 void monotone(){  
34     sort(nodes.begin(), nodes.end()) ;  
35  
36     int n = unique(nodes.begin(), nodes.end()) -  
37         nodes.begin() ;  
38  
39     vector<Coordinate> ch(n+1) ;  
40  
41     int m = 0 ;  
42  
43     for ( int i=0 ; i<n ; i++ ){  
44         while(m > 1 && cross(ch[m-2], ch[m-1],  
45             nodes[i]) < 0) m-- ;  
46         ch[m++] = nodes[i] ;  
47     }  
48     for ( int i=n-2, t=m ; i>=0 ; i-- ){  
49         while(m > t && cross(ch[m-2], ch[m-1],  
50             nodes[i]) < 0) m-- ;  
51         ch[m++] = nodes[i] ;  
52     }  
53  
54     if(n > 1) m-- ;  
55     cout << m << endl ;  
56  
57     for ( int i=0 ; i<m ; i++ ) cout <<  
58         ch[i].x << " " << ch[i].y << endl ;  
59 }
```

5 String

## 5.1 KMP

```

1 int Next[N] ;
2 void kmp(string &str){
3     Next[0] = -1 ;
4     if(str.size() <= 1) return ;
5     Next[1] = 0 ;
6
7     int cur = 2, check = 0 ;
8
9     while(cur < str.size()){
10        if(str[cur - 1] == str[check])
11            Next[cur++] = ++check ;
12        else if(check > 0) check = Next[check] ;
13        else Next[cur++] = 0 ;
14    }
15
16    int main(){
17        ios::sync_with_stdio(false) ;
18        cin.tie(nullptr) ;
19        cout.tie(nullptr) ;
20
21        string s1, s2 ;
22        while(cin >> s1){
23            s2 = s1 ;
24            reverse(s2.begin(), s2.end()) ;
25            kmp(s2) ;
26
27            int x=0, y=0 ;
28            while(x < s1.size() && y <
29                  if(s1[x] == s2[y]){
30                      x++ ;
31                      y++ ;
32                }
33                else if(y > 0) y = Next[y] ;
34                else x++ ;
35            }
36
37            cout << s1 << s2.substr(y)
38        }
39
40        return 0 ;
41    }

```

## 5.2 Trie

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

### 5.3 ACAM

```

7 int new_node(){
8     tree.emplace_back(alpha, 0) ;
9     fail.push_back(0) ;
10
11    return tree.size() - 1 ;
12}
13
14public:
15ACAutomation(int _base='a', int _alpha=26)
16: base(_base), alpha(_alpha) {
17    clear() ;
18}
19
20void clear(){
21    fail.assign(1, 0) ;
22    order.clear() ;
23    end.clear() ;
24    tree.assign(1, vector<int>(alpha, 0)) ;
25}
26
27void add_pattern(const string &pattern){
28    int u = 0 ;
29    for ( auto ch : pattern ){
30        int v = ch - base ;
31
32        if(tree[u][v] == 0) tree[u][v] =
33            new_node() ;
34        u = tree[u][v] ;
35    }
36
37    end.push_back(u) ;
38}
39
40void build(){
41    queue<int> q ;
42    order.clear() ;
43    order.push_back(0) ;
44
45    for ( int i=0 ; i<alpha ; i++ )
46        if(tree[0][i] > 0){
47            q.push(tree[0][i]) ;
48        }
49
50    while(!q.empty()){
51        int u = q.front() ; q.pop() ;
52        order.push_back(u) ;
53
54        for ( int i=0 ; i<alpha ; i++ ){
55            if(tree[u][i] == 0) tree[u][i] =
56                tree[fail[u]][i] ;
57            else{
58                fail[tree[u][i]] = tree[fail[u]][i] ;
59                ;
60                q.push(tree[u][i]) ;
61            }
62        }
63    }
64}
65
66vector<int> count_per_pattern(const string
67    &text) const {
68    int u = 0 ;
69    vector<int> vis(tree.size(), 0) ;
70
71    for ( char ch : text ){
72        u = tree[u][ch - base] ;
73        vis[u]++;
74    }
75
76    for ( int i=order.size()-1 ; i>=1 ; i-- )
77    ){
78        int x = order[i] ;
79        vis[fail[x]] += vis[x] ;
80    }
81
82    vector<int> ans(end.size(), 0) ;
83    for ( int id=0 ; id<end.size() ; id++ ){
84        ans[id] = vis[end[id]] ;
85    }
86
87    return ans ;
88}

```

```

79 }
80
81     return ans ;
82 }
83 }

```

## 6 Algorithm

### 6.1 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
34     if(u == v) return u ;
35
36     for ( int i=MaxLog ; i>=0 ; i-- )
37         if(up[u][i] != up[v][i]){
38             u = up[u][i] ;
39             v = up[v][i] ;
40         }
41
42     return up[u][0] ;
43 }
44
45 int main(){
46     int n, q, root ;
47     scanf("%d%d%d", &n, &q, &root) ;
48     MaxLog = __lg(n) ;
49
50     for ( int i=0 ; i<n-1 ; i++ ){
51         int u, v ;
52         scanf("%d%d", &u, &v) ;
53         e[u].push_back(v) ;
54         e[v].push_back(u) ;
55     }
56
57     dfs(root, root, 0) ;
58
59     while(q--){
60         int u, v ;
61         scanf("%d%d", &u, &v) ;
62         printf("%d\n", lca(u, v)) ;
63     }
64 }

```

### 6.2 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
11    vector<Edge> edge ;
12
13    void init(){
14        edge.clear() ;
15        for ( int i=0 ; i<N ; i++ ){
16            par[i] = i ;
17            sz[i] = 1 ;
18        }
19        sum = 0 ;
20    }
21
22    int find(int x){
23        if(x == par[x]) return x ;
24        return par[x] = find(par[x]) ;
25    }
26
27    int merge(int x, int y){
28        x = find(x) ;
29        y = find(y) ;
30
31        if(x == y) return 0 ;
32        if(sz[x] > sz[y]) swap(x, y) ;
33        par[x] = y ;
34        sz[y] += sz[x] ;
35
36        return 1 ;
37    }
38
39    void MST(){
40        int cnt = 0 ;
41        for ( int i=0 ; i<edge.size() && cnt < n-1
42                ; i++ ){
43            auto [u, v, w] = edge[i] ;
44            if(merge(u, v)){
45                cnt++ ;
46                sum -= w ;
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        }

```

### 6.3 SG

```

1 long long SG(long long k){
2
3     if(k % 2 == 0){
4         return k / 2;
5     }
6     else{
7         return SG(k / 2);
8     }
9

```

```

10 }
11
12 int main(){
13     int cas, n;
14
15     scanf("%d", &cas);
16     while(cas--){
17         scanf("%d", &n);
18
19         long long s, v = 0;
20
21         for(int i = 0; i < n; i++){
22             scanf("%lld", &s);
23             v ^= SG(s); //XOR
24         }
25
26         if(v) printf("YES\n");
27         else printf("NO\n");
28     }
29
30     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

```

### 6.4 Max Flow

```

1 struct Edge{
2     int v, cap, next ;
3 };
4
5 class MaxFlow{
6 private:
7     int N, S, T ;
8     vector<Edge> e ;
9     vector<int> head, cur, dep ;
10
11    bool bfs(){
12        queue<int> q ;
13        for ( int i=0 ; i<=N ; i++ ){
14            cur[i] = head[i] ;
15            dep[i] = -1 ;
16        }
17
18        q.push(S) ;
19        dep[S] = 0 ;

```

```

20
21     while(!q.empty()){
22         int u = q.front() ; q.pop() ;
23
24         for ( int i=head[u] ; i!=-1 ;
25             i=e[i].next ){
26             int v = e[i].v ;
27             if(dep[v] == -1 && e[i].cap > 0){
28                 dep[v] = dep[u] + 1 ;
29                 if(v == T) return 1 ;
30                 q.push(v) ;
31             }
32         }
33
34     return 0 ;
35 }
36
37 int dfs(int u, int flow){
38     if(u == T) return flow ;
39     int d, rest = 0 ;
40
41     for ( int &i=cur[u] ; i!=-1 ;
42         i=e[i].next ){
43         int v = e[i].v ;
44         if(dep[v] == dep[u] + 1 && e[i].cap >
45             0){
46             d = dfs(v, min(flow - rest,
47                         e[i].cap)) ;
48
49             if(d > 0){
50                 e[i].cap -= d ;
51                 e[i^1].cap += d ;
52                 rest += d ;
53
54                 if(rest == flow) break ;
55             }
56
57         if(rest != flow) dep[u] = -1 ;
58     return rest ;
59 }
59 public:
60 MaxFlow(int n, int s, int t){
61     N = n ; S = s ; T = t ;
62     e.reserve(n*n) ;
63     head.assign(n+1, -1) ;
64     cur.resize(n+1) ;
65     dep.resize(n+1) ;
66 }
67
68 void AddEdge(int u, int v, int cap){
69     e.push_back({v, cap, head[u]}) ;
70     head[u] = e.size() - 1 ;
71     e.push_back({u, 0, head[v]}) ;
72     head[v] = e.size() - 1 ;
73 }
74
75 int run(){
76     int ans = 0 ;
77     while(bfs()){
78         ans += dfs(S, 0x3f3f3f3f) ;
79     }
80     return ans ;
81 }
82 }
```

## 6.5 min cut max flow

```

1 struct Edge{
2     int v, cap, cost , next ;
3 };
4
5 using pii = pair<int, int> ;
6 class MCMF{
7 private:
```

```

8     int N, s, t, tot ;
9     vector<Edge> e ;
10    vector<int> head ;
11
12    public:
13        MCMF(int n, int _s, int _t){
14            N = n ;
15            s = _s ;
16            t = _t ;
17            e.resize(n*n + 5) ;
18            head.assign(n+5, -1) ;
19            tot = -1 ;
20        }
21
22        void AddEdge(int u, int v, int cap, int
23                      cost){
24            e[++tot] = {v, cap, cost, head[u]} ;
25            head[u] = tot ;
26            e[++tot] = {u, 0, -cost, head[v]} ;
27            head[v] = tot ;
28        }
29
30        int run(){
31            vector<int> dis(N+1), pot(N+1, 0),
32                        preE(N+1) ;
33            int flow = 0, cost = 0 ;
34
35            auto dijkstra = [&](){
36                fill(dis.begin(), dis.end(), INF) ;
37                priority_queue<pii, vector<pii>,
38                                greater<pii>> pq ;
39                dis[s] = 0 ;
40                pq.push({0, s}) ;
41
42                while(!pq.empty()){
43                    auto [d, u] = pq.top() ; pq.pop() ;
44                    if(d > dis[u]) continue ;
45                    for ( int i=head[u] ; i!=-1 ;
46                        i=e[i].next ){
47                        int v = e[i].v, cap = e[i].cap, w =
48                            e[i].cost ;
49                        if(cap && dis[v] > d + w + pot[u] -
50                            pot[v]){
51                            dis[v] = d + w + pot[u] - pot[v] ;
52                            preE[v] = i ;
53                            pq.push({dis[v], v}) ;
54                        }
55                    }
56
57                    return dis[t] != INF ;
58                };
59
60                while(dijkstra()){
61                    for ( int v=1 ; v<=N ; v++ ) if(dis[v]
62                        < INF){
63                        pot[v] += dis[v] ;
64
65                        int aug = INT_MAX ;
66                        for ( int v=t ; v!=s ;
67                            v=e[preE[v]^1].v ){
68                            aug = min(aug, e[preE[v]].cap) ;
69
70                            for ( int v=t ; v!=s ;
71                                v=e[preE[v]^1].v ){
72                                e[preE[v]].cap -= aug ;
73                                e[preE[v]^1].cap += aug ;
74                                cost += aug * e[preE[v]].cost ;
75                            }
76
77                            return cost ;
78                        }
79
80                    }
81                }
82            };
83
84        }
85
86    };
87
88    private:
89        void update(int s1, int s2){
90            if(s2 & (1 << m)){
91                dp[cur][s2 ^ (1 << m)] += dp[cur] ^
92                    1][s1] ;
93            }
94        }
95
96        int main(){
97            while(~scanf("%d%d", &n, &m)){
98                if(m > n) swap(n, m) ;
99                memset(dp, 0, sizeof(dp)) ;
100               cur = 0 ;
101               dp[cur][(1 << m) - 1] = 1 ;
102               for ( int i=0 ; i<n ; i++ ) for ( int
103                   j=0 ; j<m ; j++ ){
104                   cur ^= 1 ;
105                   memset(dp[cur], 0, sizeof(dp[cur])) ;
106
107                   for ( int k=0 ; k<(1 << m) ; k++ ){
108                       update(k, k << 1) ; // not put
109                       if(i && !(k & (1 << (m - 1))) )
110                           update(k, (k << 1) | (1 << m) |
111                               1) ; // put up
112                       if(j && !(k & 1)) update(k, (k << 1)
113                           | 3) ; // put left
114                   }
115                   printf("%lld\n", dp[cur][(1 << m) - 1]) ;
116               }
117               return 0 ;
118           }
119       }
```

## 7 DP

### 7.1 輪廓線 DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std ;
4 using ll = long long ;
5
6 ll dp[2][(1 << 10) + 5] ;
7 int n, m ;
8 int cur ;
9
10 void update(int s1, int s2){
11     if(s2 & (1 << m)){
12         dp[cur][s2 ^ (1 << m)] += dp[cur] ^
13             1][s1] ;
14    }
15
16 int main(){
17     while(~scanf("%d%d", &n, &m)){
18         if(m > n) swap(n, m) ;
19         memset(dp, 0, sizeof(dp)) ;
20         cur = 0 ;
21         dp[cur][(1 << m) - 1] = 1 ;
22         for ( int i=0 ; i<n ; i++ ) for ( int
23             j=0 ; j<m ; j++ ){
24             cur ^= 1 ;
25             memset(dp[cur], 0, sizeof(dp[cur])) ;
26
27             for ( int k=0 ; k<(1 << m) ; k++ ){
28                 update(k, k << 1) ; // not put
29                 if(i && !(k & (1 << (m - 1))) )
30                     update(k, (k << 1) | (1 << m) |
31                         1) ; // put up
32                 if(j && !(k & 1)) update(k, (k << 1)
33                     | 3) ; // put left
34             }
35             printf("%lld\n", dp[cur][(1 << m) - 1]) ;
36         }
37         return 0 ;
38     }
39 }
```

### 7.2 數位 DP

```

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

```

26
27 int count(int n){
28     memset(dp, -1, sizeof(dp)) ;
29     dig.clear() ;
30
31     while(n > 0){
32         dig.push_back(n % 10) ;
33         n /= 10 ;
34     }
35
36     return solve(dig.size() - 1, 0, 0, 1) ;
37 }
38
39 int main(){
40     int T ;
41     scanf("%d", &T) ;
42
43     int a, b ;
44     while(T--){
45         scanf("%d%d%d", &a, &b, &K) ;
46         if(K > 90) printf("0\n") ;
47         else printf("%d\n", count(b) -
48             count(a-1)) ;
49     }
50
51     return 0 ;
}

```

```

41 int main(){
42     int t = 0 ;
43
44     while(~scanf(" %d", &n) && n){
45         init() ;
46         for ( int i=0 ; i<n-1 ; i++ ){
47             int u, v, w ;
48             scanf("%d%d%d", &u, &v, &w) ;
49             edge[u].push_back({v, w}) ;
50         }
51
52         DFS(0) ;
53         printf("Case %d: \n", ++t) ;
54
55         int q, e ;
56         scanf(" %d", &q) ;
57
58         while(q--){
59             scanf(" %d", &e) ;
60
61             for ( int i=n ; i>=1 ; i-- )
62                 if(dp[0][i][0] <= e){
63                     printf("%d\n", i) ;
64                     break ;
65                 }
66         }
67
68     }
69 }

```

### 7.3 樹 DP

```

1 #include <bits/stdc++.h>
2
3 #define N 505
4 #define INF 0x3f3f3f3f
5
6 using namespace std ;
7
8 struct Edge{
9     int v, w ;
10 } ;
11
12 vector<Edge> edge[N] ;
13 int n ;
14 int cnt[N] ;
15 int dp[N][N][2] ;
16
17 void init(){
18     for ( int i=0 ; i<N ; i++ )
19         edge[i].clear() ;
20     memset(cnt, 0, sizeof(cnt)) ;
21     memset(dp, INF, sizeof(dp)) ;
22 }
23
24 void DFS(int u){
25     cnt[u] = 1 ;
26     for ( auto [v, w] : edge[u] ){
27         DFS(v) ;
28         cnt[u] += cnt[v] ;
29     }
30
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 }
47
48
49 }

```