

ACM-ICPC Template

tokitsukaze

2024 年 11 月 2 日



目录	
1 字符串	1
1.1 KMP	1
1.1.1 KMP	1
1.1.2 exKMP	1
1.2 hash	2
1.2.1 hash	2
1.2.2 good_hash_prime	2
1.2.3 hash_map	2
1.2.4 BKDRHash	3
1.3 Manacher	3
1.3.1 插字符	3
1.3.2 不插字符	3
1.4 后缀数组	4
1.4.1 倍增 sa	4
1.4.2 SA-IS	5
1.5 自动机	5
1.5.1 AC 自动机	5
1.5.2 大字符集 AC 自动机	6
1.5.3 回文自动机	7
1.5.4 序列自动机	8
1.5.5 KMP 自动机	8
1.5.6 后缀自动机	9
1.5.7 广义后缀自动机	10
1.6 最小表示法	11
1.6.1 最小表示法	11
1.6.2 最大表示法	11
1.7 shift_and	11
2 数据结构	12
2.1 离散化	12
2.2 RMQ	12
2.2.1 一维 RMQ	12
2.2.2 二维 RMQ	12
2.3 单调队列	13
2.4 并查集	13
2.4.1 并查集	13
2.4.2 map 实现并查集	13
2.4.3 可撤销并查集	14
2.5 树状数组	14
2.5.1 一维单点 BIT	14
2.5.2 一维区间 BIT	15
2.5.3 二维单点 BIT	15
2.6 线段树	16
2.6.1 线段树	16
2.6.2 动态开点线段树	16
2.6.3 区间查询最大子段和	18
2.6.4 矩形面积并	18
2.6.5 线段树维护 hash	20
2.7 平衡树	20
2.7.1 Treap	20
2.7.2 Splay 维护序列	22
2.7.3 FHQ-Treap 维护序列	23
2.7.4 pbds	25
2.8 字典树	25
2.8.1 trie	25
2.8.2 01trie	25
2.9 可持久化	26
2.9.1 可持久化线段树	26
2.9.2 可持久化 01trie	27
2.9.3 可持久化数组	28
2.9.4 可持久化并查集	29
2.10 树套树	30
2.10.1 线段树套线段树	30
2.10.2 线段树套 treap	31
2.10.3 树状数组套 treap	34
2.11 李超树	36
2.12 kd-tree	37
2.13 可并堆	40
2.13.1 pbds 可并堆	40
2.13.2 左偏树 (支持打 tag, 不支持删除)	40
2.13.3 左偏树 (支持删除, 不支持打 tag)	41
2.14 k 叉哈夫曼树	42
2.15 笛卡尔树	42
2.16 析合树	43
2.17 莫队算法	45
2.18 ODT	45
2.19 分块	46
3 树	46
3.1 LCA	46
3.1.1 倍增 LCA	46
3.1.2 RMQ 维护欧拉序求 LCA	47
3.2 树链剖分	48
3.2.1 轻重链剖分	48
3.3 树的重心	50
3.4 树上倍增	50
3.5 树 hash	50
3.6 虚树	51
3.7 树分治	51
3.7.1 点分治	51
3.8 Link-Cut-Tree	52
4 图论	54
4.1 链式前向星	54
4.2 最短路	54
4.2.1 dijkstra	54
4.2.2 spfa	55
4.2.3 floyd 求最小环	55

4.2.4	Johnson	56	5.8	exBSGS	79
4.2.5	同余最短路	57	5.9	Miller_Rabin+Pollard_rho	79
4.3	最小生成树	58	5.10	第二类 Stirling 数	81
4.3.1	kruskal	58	5.11	原根	81
4.3.2	kruskal 重构树	58	5.12	二次剩余	82
4.3.3	prim	59	6	组合数学	82
4.4	二分图匹配	60	6.1	组合数	82
4.4.1	二分图最大匹配	60	6.1.1	公式预处理	82
4.4.2	二分图最大权完美匹配	60	6.1.2	杨辉三角递推	82
4.5	最大流	62	6.2	卢卡斯定理	83
4.5.1	dinic	62	6.2.1	Lucas	83
4.5.2	有源汇上下界网络流	63	6.2.2	exLucas	83
4.6	费用流	64	6.3	卡特兰数公式	84
4.6.1	spfa 费用流	64	7	多项式	84
4.6.2	dijkstra 费用流 (dij 求 h)	65	7.1	FFT	84
4.6.3	dijkstra 费用流 (spfa 求 h)	66	7.2	NTT	85
4.7	连通性	67	7.3	FWT	86
4.7.1	强连通分量	67	7.4	拉格朗日插值	87
4.7.2	边双连通分量	68	8	矩阵	87
4.7.3	点双连通分量	69	8.1	矩阵类	87
4.7.4	圆方树	69	8.2	高斯消元	88
4.8	团	69	8.2.1	浮点数方程组	88
4.8.1	最大团	69	8.2.2	异或方程组	89
4.8.2	极大团计数	70	8.2.3	同余方程组	89
4.9	拓扑排序	71	8.3	线性基	90
4.10	2-sat	71	8.3.1	线性基	90
4.10.1	2-sat 输出任意解	71	8.3.2	带删除线性基	91
4.10.2	2-sat 字典序最小解	72	9	博弈	92
4.11	支配树	73	9.1	sg 函数	92
4.12	最小斯坦纳树	74	9.2	结论	92
5	数论	75	10	dp	92
5.1	素数筛	75	10.1	LIS	92
5.1.1	埃筛	75	10.2	LPS	93
5.1.2	线性筛	75	10.3	数位 dp	93
5.1.3	区间筛	75	11	杂项	93
5.2	逆元	75	11.1	FastIO	93
5.2.1	exgcd 求逆元	75	11.2	O(1) 快速乘	94
5.2.2	线性预处理	75	11.3	快速模	94
5.3	扩展欧几里得	76	11.4	xor_sum(1,n)	95
5.3.1	exgcd	76	11.5	约瑟夫环 kth	95
5.3.2	ax+by=c	76	11.6	判断星期几	96
5.4	中国剩余定理	76	11.7	整数三分	96
5.4.1	CRT	76	11.8	有根树与 prufer 序列的转换	96
5.4.2	exCRT	77	11.9	网格整数点正方形个数	96
5.5	欧拉函数	77	11.10	模拟退火	96
5.5.1	直接求	77	11.11	斐波那契 01 串的第 k 个字符	97
5.5.2	线性筛	78			
5.6	莫比乌斯函数	78			
5.7	Berlekamp-Massey	78			

11.12光速幂	98
11.13倍增求等比数列和	98
11.14矩阵旋转 90 度	98
12 附录	98
12.1 NTT 常用模数	98
12.2 线性基求交	99

1 字符串

1.1 KMP

1.1.1 KMP

```

1 struct KMP
2 {
3     #define type char
4     int nex[MAX],len;
5     type t[MAX];
6     void get_next(type *s,int n)
7     {
8         int i,j;
9         len=n;
10        for(i=1;i<=len;i++) t[i]=s[i];
11        t[len+1]=0;
12        nex[0]=nex[1]=0;
13        j=0;
14        for(i=2;i<=len;i++)
15        {
16            while(j&&t[j+1]!=s[i]) j=nex[j];
17            if(t[j+1]==s[i]) j++;
18            nex[i]=j;
19        }
20    }
21    vector<int> match(type *s,int n)
22    {
23        int i,j;
24        vector<int> res;
25        for(i=1,j=0;i<=n;i++)
26        {
27            while(j&&t[j+1]!=s[i]) j=nex[j];
28            if(t[j+1]==s[i]) j++;
29            if(j==len)
30            {
31                res.push_back(i-len+1);
32                j=nex[j];
33            }
34        }
35        return res;
36    }
37    #undef type
38 }kmp;
39 /*
40 kmp.get_next(t,len); // t[1..len]
41 kmp.match(s,n); // s[1..n] return all pos t in s
42 */

```

1.1.2 exKMP

```

1 struct Z_Algorithm
2 {
3     char s[MAX];

```

```

4     int n,z[MAX],ex[MAX];
5     void get_z_func(char *_s,int _n) // s[0..n-1]
6     {
7         int i,j,l,r;
8         n=_n;
9         memcpy(s,_s,n);
10        z[0]=l=r=0;
11        for(i=1;i<n;i++)
12        {
13            if(i+z[i-1]-1<r) z[i]=z[i-1];
14            else
15            {
16                j=max(0,r-i+1);
17                while(i+j<n&&s[i+j]==s[j]) j++;
18                z[i]=j;
19                if(i+z[i]-1>r)
20                {
21                    l=i;
22                    r=i+z[i]-1;
23                }
24            }
25        }
26        z[0]=n;
27    }
28    void get_ex(char *t,int m) // t[0..m-1]
29    {
30        int i,j,l,r;
31        j=l=0;
32        while(j<n&&j<m&&t[j]==s[j]) j++;
33        ex[0]=j;
34        r=l+ex[0]-1;
35        for(i=1;i<m;i++)
36        {
37            if(i+z[i-1]-1<r) ex[i]=z[i-1];
38            else
39            {
40                j=max(0,r-i+1);
41                while(i+j<m&&t[i+j]==s[j]) j++;
42                ex[i]=j;
43                if(i+ex[i]-1>r)
44                {
45                    l=i;
46                    r=i+ex[i]-1;
47                }
48            }
49        }
50    }
51 }z;
52 /*
53 z[i]: lcp(s,s[i..n-1]) i=0..n-1
54 ex[i]: lcp(s,t[i..m-1]) i=0..m-1
55
56 z.get_z_func(s,n) s[0..n-1]
57 z.get_ex(t,m) t[0..m-1]

```

```
58 */
```

1.2 hash

1.2.1 hash

```
1 namespace HASH
2 {
3     #define type int
4     int m,k;
5     vector<type> sd,p;
6     vector<vector<type>> tmp;
7     void set(vector<type> _sd,vector<type> _p)
8     {
9         sd=_sd;
10        p=_p;
11        assert(sd.size()==p.size());
12        k=sd.size();
13    }
14    void init(int _m)
15    {
16        int i,j;
17        m=_m;
18        tmp.resize(k);
19        for(j=0;j<k;j++)
20        {
21            tmp[j].resize(m+1);
22            tmp[j][0]=1;
23            for(i=1;i<=m;i++) tmp[j][i]=111*tmp[j][i-1]*sd[j]%p[j];
24        }
25    }
26    struct hash_table
27    {
28        vector<vector<type>> ha;
29        void work(char *s,int n)
30        {
31            int i,j;
32            assert(n<=m);
33            ha.resize(k);
34            for(j=0;j<k;j++)
35            {
36                ha[j].resize(n+1);
37                ha[j][0]=0;
38                for(i=1;i<=n;i++) ha[j][i]=(111*ha[j][i-1]*sd[j]+s[i])%p[j];
39            }
40        }
41        vector<type> get(int l,int r)
42        {
43            vector<type> res(k);
44            for(int j=0;j<k;j++)
45            {
```

```
46            res[j]=(ha[j][r]-111*ha[j][l-1]*tmp[j][r-l+1])%p[j];
47            if(res[j]<0) res[j]+=p[j];
48        }
49        return res;
50    }
51 };
52 #undef type
53 }
54 HASH::hash_table ha;
55 /*
56 HASH::set({131,233},{402653189,805306457});
57 HASH::init(m);
58 ha.work(s,n); // n<=m
59 */
```

1.2.2 good_hash_prime

```
1 vector<ll> good_hash_prime={
2 50331653,100663319,201326611,
3 402653189,805306457,1610612741
4 };
5 vector<ll> get_hash_prime(int cnt=2)
6 {
7     assert(cnt<=good_hash_prime.size());
8     srand(time(0));
9     random_shuffle(good_hash_prime.begin(),
10                    good_hash_prime.end());
11     return vector<ll>(good_hash_prime.begin(),
12                       good_hash_prime.begin()+cnt);
```

1.2.3 hash_map

```
1 struct hash_map
2 {
3     #define type int
4     #define TA pair<type,type>
5     static const int p=999917;
6     TA val[MAX];
7     type w[MAX];
8     int tot,head[p],nex[MAX];
9     int top,st[MAX];
10    hash_map(){top=tot=0;}
11    void clear(){tot=0;while(top) head[st[top]--]=0;}
12    void add(int x,TA y){val[++tot]=y;nex[tot]=head[x];head[x]=tot;w[tot]=0;}
13    bool count(TA y)
14    {
15        int x=y.first%p;
16        for(int i=head[x];i;i=nex[i])
17        {
```

```

18         if(y==val[i]) return 1;
19     }
20     return 0;
21 }
22 type& operator [] (TA y)
23 {
24     int x=y.first%p;
25     for(int i=head[x];i;i=nex[i])
26     {
27         if(y==val[i]) return w[i];
28     }
29     add(x,y);
30     st[++top]=x;
31     return w[tot];
32 }
33 #undef TA
34 #undef type
35 }flag;

```

1.2.4 BKDRHash

```

1 struct BKDRHash
2 {
3     static const ull seed=1313131;
4     static const int p=2000007;
5     ull Hash[MAX],tmp[MAX];
6     ull val[MAX];
7     int last[p+10],nex[MAX],cnt;
8     void init()//clear hash table
9     {
10         mem(last,0);
11         cnt=0;
12     }
13     bool insert(ull x)
14     {
15         int u=x%p;
16         for(int i=last[u];i;i=nex[i])
17         {
18             if(val[i]==x) return 1;
19         }
20         nex[++cnt]=last[u];
21         last[u]=cnt;
22         val[cnt]=x;
23         return 0;
24     }
25     void work(char *s,int n)
26     {
27         tmp[0]=1;
28         Hash[0]=0;
29         for(int i=1;i<=n;i++)
30         {
31             tmp[i]=tmp[i-1]*seed;
32             Hash[i]=Hash[i-1]*seed+s[i];

```

```

33     }
34 }
35 ull get(int l,int r)
36 {
37     return Hash[r]-Hash[l-1]*tmp[r-l+1];
38 }
39 }bkdr; //bkdr.init();

```

1.3 Manacher

1.3.1 插字符

```

1 struct Manacher
2 {
3     int p[MAX<<1];
4     char s[MAX<<1];
5     int work(char *a,int n) // a[1..n]
6     {
7         int i,mid,r,res=0;
8         for(i=1;i<=n;i++)
9         {
10             p[i]=0;
11             s[2*i-1]='%';
12             s[2*i]=a[i];
13         }
14         s[n=n*2+1]='%';
15         mid=r=0;
16         for(i=1;i<=n;i++)
17         {
18             if(i<r) p[i]=min(p[2*mid-i],r-i);
19             else p[i]=1;
20             while(i-p[i]>=1&&i+p[i]<=n&&s[i-p[i]]==s[
21                 i+p[i]]) p[i]++;
22             if(i+p[i]>r)
23             {
24                 r=i+p[i];
25                 mid=i;
26             }
27             res=max(res,p[i]-1);
28         }
29         return res;
30 }la;

```

1.3.2 不插字符

```

1 struct Manacher
2 {
3     int p[MAX];
4     int work(char *s,int n) //s[1..n]
5     {
6         int r,mid,i,res=0;
7         //odd

```

```

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

```

1.4 后缀数组

1.4.1 倍增 sa

```

1 struct Suffix_Array
2 {
3     int s[MAX],n,SZ;
4     int c[MAX],rk[MAX],tmp[MAX],sa[MAX],h[MAX];

```

```

5 void get_sa()
6 {
7     int m,i,j,k,tot;
8     m=SZ;
9     for(i=1;i<=m;i++) c[i]=0;
10    for(i=1;i<=n;i++) c[rk[i]=s[i]]++;
11    for(i=2;i<=m;i++) c[i]+=c[i-1];
12    for(i=n;i;i--) sa[c[rk[i]]--]=i;
13    for(k=1;k<=n;k<=1)
14    {
15        tot=0;
16        for(i=n-k+1;i<=n;i++) tmp[++tot]=i;
17        for(i=1;i<=n;i++)
18        {
19            if(sa[i]>k) tmp[++tot]=sa[i]-k;
20        }
21        for(i=1;i<=m;i++) c[i]=0;
22        for(i=1;i<=n;i++) c[rk[i]]++;
23        for(i=2;i<=m;i++) c[i]+=c[i-1];
24        for(i=n;i;i--)
25        {
26            sa[c[rk[tmp[i]]]--]=tmp[i];
27            tmp[i]=0;
28        }
29        swap(rk,tmp);
30        rk[sa[1]]=1;
31        tot=1;
32        for(i=2;i<=n;i++)
33        {
34            if(sa[i]+k>n||sa[i-1]+k>n||
35                (!(tmp[sa[i]]==tmp[sa[i-1]]&&
36                    tmp[sa[i]+k]==tmp[sa[i-1]+k])))
37                tot++;
38            rk[sa[i]]=tot;
39        }
40        if(tot==n) break;
41        m=tot;
42    }
43    h[1]=0;
44    k=0;
45    for(i=1;i<=n;i++)
46    {
47        if(rk[i]==1) continue;
48        if(k>0) k--;
49        j=sa[rk[i]-1];
50        while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) k++;
51        h[rk[i]]=k;
52    }
53 }
54 void work(char *_s,int _n) //s[1..n]
55 {
56     SZ=0; // char size
57     n=_n;

```



```

57     for(int i=1;i<=n;i++)
58     {
59         s[i]=_s[i];
60         SZ=max(SZ,s[i]);
61     }
62     get_sa();
63 }
64 }sa;
65 /*
66 sa[i]: s[sa[i]..n] rank is i
67 rk[i]: s[i..n] rank is rk[i]
68 h[i]: lcp(s[sa[i]..n],s[sa[i-1]..n])
69 lcp(s[i..n],s[j..n]) (i<j): min{h[i+1..j]}
70
71 sa.work(s,n) s[1..n]
72 */

```

1.4.2 SA-IS

```

1 struct SA
2 {
3     char S[MAX]; int n,m;
4     int s[MAX<<1],t[MAX<<1],H[MAX],sa[MAX],r[MAX],p[
        MAX],c[MAX],w[MAX];
5     inline int trans(int n,const char* S){
6         int m=*max_element(S+1,S+1+n);
7         for(int i=1;i<=n;++i) r[S[i]]=1;
8         for(int i=1;i<=m;++i) r[i]+=r[i-1];
9         for(int i=1;i<=n;++i) s[i]=r[S[i]];
10        return r[m];
11    }
12    #define ps(x) sa[w[s[x]]--]=x
13    #define pl(x) sa[w[s[x]]++]=x
14    inline void radix(int* v,int* s,int* t,int n,int
        m,int n1){
15        memset(sa,0,n+1<<2); memset(c,0,m+1<<2);
16        for(int i=1;i<=n;++i) ++c[s[i]];
17        for(int i=1;i<=m;++i) w[i]=c[i]+c[i-1];
18        for(int i=n1;i--i) ps(v[i]);
19        for(int i=1;i<=m;++i) w[i]=c[i-1]+1;
20        for(int i=1;i<=n;++i) if(sa[i]>1 && t[sa[i
            ]-1]) pl(sa[i]-1);
21        for(int i=1;i<=m;++i) w[i]=c[i];
22        for(int i=n;i--i) if(sa[i]>1 && !t[sa[i]-1])
            ps(sa[i]-1);
23    }
24    inline void SAIS(int n,int m,int* s,int* t,int*
        p){
25        int n1=0,ch=r[1]=0,*s1=s+n; t[n]=0;
26        for(int i=n-1;i--i) t[i]=s[i]==s[i+1]?t[i
            +1]:s[i]>s[i+1];
27        for(int i=2;i<=n;++i) r[i]=t[i-1]&&!t[i]?(p
            [++n1]=i,n1):0;
28        radix(p,s,t,n,m,n1);

```

```

29     for(int i=1,x,y;i<=n;++i) if(x=r[sa[i]]){
30         if(ch<=1 || p[x+1]-p[x]!=p[y+1]-p[y]) ++
            ch;
31         else for(int j=p[x],k=p[y];j<=p[x+1];++j
            ,++k)
32             if((s[j]<<1|t[j])^(s[k]<<1|t[k])){ ++
                ch; break; }
33         s1[y=x]=ch;
34     }
35     if(ch<n1) SAIS(n1,ch,s1,t+n,p+n1);
36     else for(int i=1;i<=n1;++i) sa[s1[i]]=i;
37     for(int i=1;i<=n1;++i) s1[i]=p[sa[i]];
38     radix(s1,s,t,n,m,n1);
39 }
40 inline void get_sa(int n,const char* S,int *ssa,
    int *h){
41     int m=trans(++n,S); SAIS(n,m,s,t,p);
42     for(int i=1;i<=n;++i) r[sa[i]=sa[i+1]]=i;
43     for(int i=1,j,k=0;i<=n;++i) if(r[i]>1){
44         for(j=sa[r[i]-1];S[i+k]==S[j+k];++k);
45         if(H[r[i]]=k) --k;
46     }
47     for(int i=1;i<=n;i++)
48     {
49         ssa[i]=sa[i];
50         h[i]=H[i];
51     }
52 }
53 }sais;

```

1.5 自动机

1.5.1 AC 自动机

```

1 struct AC_Automaton
2 {
3     static const int K=;
4     int nex[MAX][K],fail[MAX],cnt[MAX],lst[MAX];
5     int root,tot,pos[MAX];
6     int getid(char c){return c-;} //may need change
7     int newnode()
8     {
9         memset(nex[tot],0,sizeof nex[tot]);
10        fail[tot]=0;
11        cnt[tot]=0;
12        return tot++;
13    }
14    void init()
15    {
16        tot=0;
17        root=newnode();
18    }
19    void insert(char *s,int n,int x) // s[0..n-1]
20    {

```

```

21     int now,i,t;
22     now=root;
23     for(i=0;i<n;i++)
24     {
25         t=getid(s[i]);
26         if(!nex[now][t]) nex[now][t]=newnode();
27         now=nex[now][t];
28     }
29     cnt[now]++;
30     pos[x]=now;
31 }
32 void work()
33 {
34     int i,now;
35     queue<int> q;
36     for(i=0;i<K;i++)
37     {
38         if(nex[root][i]) q.push(nex[root][i]);
39     }
40     while(!q.empty())
41     {
42         now=q.front();
43         q.pop();
44
45         //suffix link
46         if(cnt[fail[now]]) lst[now]=fail[now];
47         else lst[now]=lst[fail[now]];
48
49         for(i=0;i<K;i++)
50         {
51             if(nex[now][i])
52             {
53                 fail[nex[now][i]]=nex[fail[now]][i];
54                 q.push(nex[now][i]);
55             }
56             else nex[now][i]=nex[fail[now]][i];
57         }
58     }
59 }
60 int out[MAX];
61 void topsort()
62 {
63     int i,t;
64     queue<int> q;
65     for(i=1;i<tot;i++) out[fail[i]]++;
66     for(i=1;i<tot;i++)
67     {
68         if(!out[i]) q.push(i);
69     }
70     while(!q.empty())
71     {
72         t=q.front();
73         q.pop();

```

```

74
75         // do something
76
77         out[fail[t]]--;
78         if(out[fail[t]]==0) q.push(fail[t]);
79     }
80 }
81 int query(char *s,int n)
82 {
83     int len,now,i,res,t,tmp;
84     now=root;
85     res=0;
86     vector<pair<int,int>> del;
87     for(i=0;i<n;i++)
88     {
89         t=getid(s[i]);
90         now=nex[now][t];
91         tmp=now;
92         while(tmp!=root&&cnt[tmp]!=-1)
93         {
94             res+=cnt[tmp];
95             del.push_back({tmp,cnt[tmp]});
96             cnt[tmp]=-1;
97             tmp=lst[tmp];
98         }
99     }
100     for(auto &it:del) cnt[it.first]=it.second;
101     return res;
102 }
103 void build_fail_tree(vector<int> mp[])
104 {
105     for(int i=0;i<=tot;i++) mp[i].clear();
106     for(int i=1;i<tot;i++) mp[fail[i]].push_back(i);
107 }
108 }ac;
109 /*
110 i is the suffix for each node in the subtree(i) of
111 the fail tree
112
113 ac.init();
114 ac.insert(s,len,id); s[0..len-1], id:1..n
115 ac.work();
116 ac.query(s,len); s[0..len-1]
117 ac.build_fail_tree(mp);
118 */

```

1.5.2 大字符集 AC 自动机

```

1 struct AC_Automaton
2 {
3     map<int,int> nex[MAX];
4     VI toplist;
5     int fail[MAX],last[MAX],cnt[MAX];

```

```

6   int root,tot;
7   int newnode()
8   {
9       tot++;
10      nex[tot].clear();
11      return tot;
12  }
13  void init()
14  {
15      toplist.clear();
16      tot=0;
17      root=newnode();
18  }
19  void insert(VI &s)
20  {
21      int len,now,i;
22      len=sz(s);
23      now=root;
24      for(i=0;i<len;i++)
25      {
26          int t=s[i];
27          if(!nex[now].count(t)) nex[now][t]=
28              newnode();
29          now=nex[now][t];
30      }
31      cnt[now]=1;
32  }
33  void work()
34  {
35      int i,now;
36      queue<int>q;
37      for(auto it:nex[root])
38      {
39          fail[it.se]=root;
40          q.push(it.se);
41      }
42      fail[root]=1;
43      while(!q.empty())
44      {
45          now=q.front();
46          q.pop();
47          toplist.pb(now);
48          //suffix link
49          /* if(cnt[fail[now]]) last[now]=fail[now];
50             else last[now]=last[fail[now]];*/
51          cnt[now]+=cnt[fail[now]];
52          for(auto it:nex[now])
53          {
54              int fail_now=fail[now];
55              while(fail_now>1&&!nex[fail_now].count
56                  (it.fi)) fail_now=fail[fail_now];
57              if(nex[fail_now].count(it.fi)) fail[it
58                  .se]=nex[fail_now][it.fi];
59              else fail[it.se]=root;

```

```

57          q.push(it.se);
58      }
59  }
60  }
61  int query(VI& s,int x)
62  {
63      int len,now,i,res;
64      len=sz(s);
65      now=root;
66      res=0;
67      for(i=0;i<len;i++)
68      {
69          int t=s[i];
70          while(now>1&&!nex[now].count(t)) now=fail
71              [now];
72          if(nex[now].count(t)) now=nex[now][t];
73          else now=root;
74          //do something
75      }
76      return res;
77  }
78  void toptrans()
79  {
80      for(int i=sz(toplist)-1;~i;i--)/*do something
81          */;
82  }
83 }ac;

```

1.5.3 回文自动机

```

1  struct Palindrome_Tree
2  {
3      static const int N=MAX;
4      static const int LOGN=log2(N)+3;
5      static const int K=26;// char size: [0,25]
6      int len[N],nex[N][K],fail[N],last,pos[N],s[N],
7          tot,n;
8      int cnt[N],dep[N];
9      int newnode(int l)
10     {
11         memset(nex[tot],0,sizeof nex[tot]);
12         fail[tot]=0;
13         dep[tot]=cnt[tot]=0;
14         len[tot]=l;
15         return tot++;
16     }
17     void init()
18     {
19         tot=n=last=0;
20         newnode(0);
21         newnode(-1);
22         s[0]=-1;
23         fail[0]=1;

```

```

24 int get_fail(int x)
25 {
26     while(s[n-len[x]-1]!=s[n]) x=fail[x];
27     return x;
28 }
29 void add(int t,int p)
30 {
31     int id,now;
32     s[++n]=t;
33     now=get_fail(last);
34     if(!nex[now][t])
35     {
36         id=newnode(len[now]+2);
37         fail[id]=nex[get_fail(fail[now])][t];
38         dep[id]=dep[fail[id]]+1;
39         nex[now][t]=id;
40     }
41     last=nex[now][t];
42     cnt[last]++;
43     pos[p]=last;
44 }
45 void topsort()
46 {
47     for(int i=tot-1;i;i--) cnt[fail[i]]+=cnt[i];
48 }
49 int st[N][LOGN];
50 void init_ST()
51 {
52     int i,j;
53     for(i=2;i<tot;i++)
54     {
55         st[i][0]=fail[i];
56         for(j=1;j<LOGN;j++)
57         {
58             st[i][j]=st[st[i][j-1]][j-1];
59         }
60     }
61 }
62 int get_substr(int l,int r)//init_ST
63 {
64     int now,tmp,i;
65     now=pos[r];
66     for(i=LOGN-1;~i;i--)
67     {
68         if(len[st[now][i]]>=r-l+1) now=st[now][i];
69     }
70     //maybe need judge if len[now]==r-l+1
71     return now;
72 }
73 void build_tree(vector<int> mp[])// root is 0
74 {
75     for(int i=0;i<=tot+1;i++) mp[i].clear();

```

```

76         for(int i=1;i<tot;i++) mp[fail[i]].push_back(
77             i);
78     }
79 }pam;
80 /*
81 pam.init();
82 pam.add(t,id); t is int
83 */

```

1.5.4 序列自动机

```

1 int nex[MAX][26];
2 void work(char *s,int len)
3 {
4     mem(nex[len],0);
5     for(int i=len;i;i--)
6     {
7         for(int j=0;j<26;j++)
8         {
9             nex[i-1][j]=nex[i][j];
10        }
11        nex[i-1][s[i]-'a']=i;
12    }
13 }

```

1.5.5 KMP 自动机

```

1 struct KMP_Automaton
2 {
3     #define type char
4     static const int K=;
5     int nex[MAX],len,lst[MAX][K];
6     type t[MAX];
7     int get_id(char c){return c-;} //may need change
8     void get_next(type *s,int n)
9     {
10        int i,j,k;
11        len=n;
12        for(i=1;i<=len;i++) t[i]=s[i];
13        t[len+1]=0;
14        nex[0]=nex[1]=0;
15        j=0;
16        for(i=2;i<=len;i++)
17        {
18            while(j&&t[j+1]!=s[i]) j=nex[j];
19            if(t[j+1]==s[i]) j++;
20            nex[i]=j;
21        }
22        memset(lst[0],0,sizeof lst[0]);
23        for(i=1;i<=len;i++)
24        {

```

```

25         for(k=0;k<K;k++) lst[i][k]=lst[nex[i]][k];
26         if(i+1<=len) lst[i][get_id(s[i+1])]=i;
27     }
28 }
29 vector<int> match(type *s,int n)
30 {
31     int i,j;
32     vector<int> res;
33     for(i=1,j=0;i<=n;i++)
34     {
35         j=lst[j][get_id(s[i])];
36         if(t[j+1]==s[i]) j++;
37         if(j==len)
38         {
39             res.push_back(i-len+1);
40             j=nex[j];
41         }
42     }
43     return res;
44 }
45 #undef type
46 }kmp;
47 /*
48 kmp.get_next(t,len); // t[1..len]
49 kmp.match(s,n); // s[1..n] return all pos t in s
50 */

```

1.5.6 后缀自动机

```

1 struct Suffix_Automaton
2 {
3     static const int N=MAX<<1;
4     static const int K=26;// char size: [0,25]
5     int tot,lst,nex[N][K],fa[N],len[N],cnt[N],maxlen,
6         ,root;
7     int newnode()
8     {
9         tot++;
10        fa[tot]=len[tot]=cnt[tot]=0;
11        memset(nex[tot],0,sizeof nex[tot]);
12        return tot;
13    }
14    void init()
15    {
16        fa[0]=len[0]=cnt[0]=0;
17        memset(nex[0],0,sizeof nex[0]);
18        tot=0;
19        maxlen=0;
20        root=lst=newnode();
21    }
22    void add(int x)
23    {
24        int p,q,np,nq;

```

```

24        p=lst;
25        np=lst=newnode();
26        len[np]=len[p]+1;
27        maxlen=max(maxlen,len[np]);
28        cnt[lst]=1;
29        while(p&&!nex[p][x])
30        {
31            nex[p][x]=np;
32            p=fa[p];
33        }
34        if(p==0) fa[np]=root;
35        else
36        {
37            q=nex[p][x];
38            if(len[q]==len[p]+1) fa[np]=q;
39            else
40            {
41                nq=newnode();
42                memcpy(nex[nq],nex[q],sizeof(nex[q]));
43                len[nq]=len[p]+1;
44                maxlen=max(maxlen,len[nq]);
45                fa[nq]=fa[q];
46                fa[q]=fa[np]=nq;
47                while(p&&nex[p][x]==q)
48                {
49                    nex[p][x]=nq;
50                    p=fa[p];
51                }
52            }
53        }
54    }
55    int sum[N],tp[N];
56    void topsort()
57    {
58        int i;
59        for(i=1;i<=maxlen;i++) sum[i]=0;
60        for(i=1;i<=tot;i++) sum[len[i]]++;
61        for(i=1;i<=maxlen;i++) sum[i]+=sum[i-1];
62        for(i=1;i<=tot;i++) tp[sum[len[i]]--]=i;
63        for(i=tot;i-->0) cnt[fa[tp[i]]]+=cnt[tp[i]];
64    }
65    void build_tree(VI mp[])
66    {
67        for(int i=1;i<=tot;i++) mp[i].clear();
68        for(int i=1;i<=tot;i++) mp[fa[i]].pb(i);
69    }
70    int pos[N],id[N];
71    void init_pos(char *s,int n)//s[1..n]
72    {
73        int now=1;
74        for(int i=1;i<=tot;i++) id[i]=-1;
75        for(int i=1;i<=n;i++)
76        {
77            now=nex[now][s[i]-'a'];

```

```

78         pos[i]=now;
79         id[now]=i;
80     }
81 }
82 int st[N][21];
83 void init_ST()
84 {
85     int i,j,x;
86     for(i=1;i<=tot;i++)
87     {
88         x=tp[i];
89         st[x][0]=fa[x];
90         for(j=1;j<20;j++)
91         {
92             st[x][j]=st[st[x][j-1]][j-1];
93         }
94     }
95 }
96 int get_substr(int l,int r)//init_pos init_ST
97 {
98     int now,tmp,i;
99     now=pos[r];
100    for(i=19;~i;i--)
101    {
102        tmp=st[now][i];
103        if(tmp&&len[tmp]>=r-l+1) now=tmp;
104    }
105    return now;
106 }
107 }sam;
108 /*
109 sam.init();
110 sam.add(int x);
111 sam.topsort();
112 */
113 */

```

1.5.7 广义后缀自动机

```

1 struct Suffix_Automaton
2 {
3     static const int N=MAX<<1;
4     static const int K=26;// char size: [0,25]
5     int tot,lst,nex[N][K],fa[N],len[N],cnt[N],mxlen,
        root;
6     int newnode()
7     {
8         tot++;
9         fa[tot]=len[tot]=cnt[tot]=0;
10        memset(nex[tot],0,sizeof nex[tot]);
11        return tot;
12    }
13    void init()
14    {

```

```

15        fa[0]=len[0]=cnt[0]=0;
16        memset(nex[0],0,sizeof nex[0]);
17        tot=0;
18        mxlen=0;
19        root=lst=newnode();
20    }
21    void add(int x)
22    {
23        int p,q,np,nq;
24        if(nex[lst][x])
25        {
26            p=lst;
27            q=nex[lst][x];
28            if(len[q]==len[p]+1) lst=q;
29            else
30            {
31                nq=newnode();
32                memcpy(nex[nq],nex[q],sizeof(nex[q]));
33                len[nq]=len[p]+1;
34                mxlen=max(mxlen,len[nq]);
35                fa[nq]=fa[q];
36                fa[q]=nq;
37                while(p&&nex[p][x]==q)
38                {
39                    nex[p][x]=nq;
40                    p=fa[p];
41                }
42            }
43        }
44        else
45        {
46            p=lst;
47            np=lst=newnode();
48            len[np]=len[p]+1;
49            mxlen=max(mxlen,len[np]);
50            cnt[lst]=1;
51            while(p&&!nex[p][x])
52            {
53                nex[p][x]=np;
54                p=fa[p];
55            }
56            if(p==0) fa[np]=root;
57            else
58            {
59                q=nex[p][x];
60                if(len[q]==len[p]+1) fa[np]=q;
61                else
62                {
63                    nq=newnode();
64                    memcpy(nex[nq],nex[q],sizeof(nex[q]));
65                    len[nq]=len[p]+1;
66                    mxlen=max(mxlen,len[nq]);
67                    fa[nq]=fa[q];

```

```

68         fa[q]=fa[np]=nq;
69         while(p&&nex[p][x]==q)
70         {
71             nex[p][x]=nq;
72             p=fa[p];
73         }
74     }
75 }
76 }
77 }
78 void insert(char *s,int n)
79 {
80     int i;
81     lst=root;
82     for(i=1;i<=n;i++) add(s[i]-'a');
83 }
84 int sum[N],tp[N];
85 void topsort()
86 {
87     int i;
88     for(i=1;i<=mxlen;i++) sum[i]=0;
89     for(i=1;i<=tot;i++) sum[len[i]]++;
90     for(i=1;i<=mxlen;i++) sum[i]+=sum[i-1];
91     for(i=1;i<=tot;i++) tp[sum[len[i]]--]=i;
92     for(i=tot;i;i--) cnt[fa[tp[i]]]+=cnt[tp[i]];
93 }
94 }sam;
95 /*
96 sam.init();
97 sam.add(int x);
98 */

```

1.6 最小表示法

1.6.1 最小表示法

```

1 int min_representation(char *s,int n) // s[0..n-1]
2 {
3     int i,j,k,tmp;
4     i=k=0;
5     j=1;
6     while(i<n&&j<n&&k<n)
7     {
8         tmp=s[(i+k)%n]-s[(j+k)%n];
9         if(!tmp) k++;
10        else
11        {
12            if(tmp>0) i=i+k+1;
13            else j=j+k+1;
14            if(i==j) j++;
15            k=0;
16        }
17    }
18    return i<j?i:j;

```

```

19 }

```

1.6.2 最大表示法

```

1 int max_representation(char *s,int n) // s[0..n-1]
2 {
3     int i,j,k,tmp;
4     i=k=0;
5     j=1;
6     while(i<n&&j<n&&k<n)
7     {
8         tmp=s[(i+k)%n]-s[(j+k)%n];
9         if(!tmp) k++;
10        else
11        {
12            if(tmp<0) i=i+k+1;
13            else j=j+k+1;
14            if(i==j) j++;
15            k=0;
16        }
17    }
18    return i<j?i:j;
19 }

```

1.7 shift_and

```

1 void shift_and(char *s,char *t)//s[1..n],t[1..m] (n
   <=m)
2 {
3     static const int SZ=26;
4     int n,m,i;
5     bitset<MAX> b[SZ],d;
6     for(i=0;i<SZ;i++) b[i].reset();
7     d.reset();
8     n=strlen(s+1);
9     m=strlen(t+1);
10    for(i=1;i<=n;i++)
11    {
12        b[s[i]-'a'].set(i-1);//change
13        //other matching character sets
14    }
15    for(i=1;i<=m;i++)
16    {
17        d<<=1;
18        d.set(0);
19        d&=(b[t[i]-'a']);//change
20        if(d[n-1]==1)//successful match
21        {
22
23        }
24    }
25 }

```

2 数据结构

2.1 离散化

```

1 struct Discretization
2 {
3     #define type ll
4     #define all(x) x.begin(),x.end()
5     vector<type> a;
6     void init(){a.clear();}
7     void add(type x){a.push_back(x);}
8     void work(){sort(all(a));a.resize(unique(all(a))
9         -a.begin());}
10    int get_pos(type x){return lower_bound(all(a),x)
11        -a.begin()+1;}
12    type get_val(int pos){return a[pos-1];}
13    int size(){return a.size();}
14    #undef type
15    #undef all
16 }d;

```

```

25         pmn[j][i]=pmin(pmn[j-1][i],pmn[j-1][i+
26             bin[j-1]]);
27     }
28 }
29 int ask_pmax(int l,int r)
30 {
31     int t=lg[r-l+1];
32     return pmax(pmx[t][l],pmx[t][r-bin[t]+1]);
33 }
34 int ask_pmin(int l,int r)
35 {
36     int t=lg[r-l+1];
37     return pmin(pmn[t][l],pmn[t][r-bin[t]+1]);
38 }
39 type ask_max(int l,int r){return v[ask_pmax(l,r)
40     ];}
41 type ask_min(int l,int r){return v[ask_pmin(l,r)
42     ];}
43 #undef type
44 }rmq;

```

2.2 RMQ

2.2.1 一维 RMQ

```

1 struct ST_table
2 {
3     #define type int
4     static const int N=MAX;
5     static const int LOG=21;
6     type v[N];
7     int lg[N],bin[LOG],pmx[LOG][N],pmn[LOG][N];
8     int pmax(const int &a,const int &b){return v[a]>
9         v[b]?a:b;}
10    int pmin(const int &a,const int &b){return v[a]<
11        v[b]?a:b;}
12    void work(int n,type *a)
13    {
14        int i,j;
15        for(i=bin[0]=1;1<=(i-1)<=n;i++) bin[i]=(bin[i-1]<<1);
16        for(i=2,lg[1]=0;i<=n;i++) lg[i]=lg[i>>1]+1;
17        for(i=1;i<=n;i++)
18        {
19            v[i]=a[i];
20            pmx[0][i]=pmn[0][i]=i;
21        }
22        for(j=1;1<=(j-1)<=n;j++)
23        {
24            for(i=1;i+bin[j]-1<=n;i++)
25            {
26                pmx[j][i]=pmax(pmx[j-1][i],pmx[j-1][i+
27                    bin[j-1]]);
28                pmn[j][i]=pmin(pmn[j-1][i],pmn[j-1][i+
29                    bin[j-1]]);
30            }
31        }
32    }
33 }

```

2.2.2 二维 RMQ

```

1 int v[302][302];
2 int maxx[302][302][9][9],minn[302][302][9][9];
3 void RMQ(int n,int m)
4 {
5     int i,j,ii,jj;
6     for(i=1;i<=n;i++)
7     {
8         for(j=1;j<=m;j++)
9         {
10            maxx[i][j][0][0]=minn[i][j][0][0]=v[i][j];
11        }
12    }
13    for(ii=0;(1<=ii)<=n;ii++)
14    {
15        for(jj=0;(1<=jj)<=m;jj++)
16        {
17            if(!(ii+jj)) continue;
18            for(i=1;i+(1<=ii)-1<=n;i++)
19            {
20                for(j=1;j+(1<=jj)-1<=m;j++)
21                {
22                    if(ii)
23                    {
24                        minn[i][j][ii][jj]=min(minn[i][j][ii-1][jj],minn[i+(1<=ii)-1][j][ii-1][jj]);
25                        maxx[i][j][ii][jj]=max(maxx[i][j][ii-1][jj],maxx[i+(1<=ii)-1][j][ii-1][jj]);
26                    }
27                }
28            }
29        }
30    }
31 }

```



```

26         }
27         else
28         {
29             minn[i][j][ii][jj]=min(minn[i][j][ii][jj-1],minn[i][j][ii-1][jj]);
30             maxx[i][j][ii][jj]=max(maxx[i][j][ii][jj-1],maxx[i][j][ii-1][jj]);
31         }
32     }
33 }
34 }
35 }
36 }
37 int ask_max(int x1,int y1,int x2,int y2)
38 {
39     int k1=0;
40     while((1<<(k1+1))<=x2-x1+1) k1++;
41     int k2=0;
42     while((1<<(k2+1))<=y2-y1+1) k2++;
43     x2=x2-(1<<k1)+1;
44     y2=y2-(1<<k2)+1;
45     return max(max(maxx[x1][y1][k1][k2],maxx[x1][y2][k1][k2]),max(maxx[x2][y1][k1][k2],maxx[x2][y2][k1][k2]));
46 }
47 int ask_min(int x1,int y1,int x2,int y2)
48 {
49     int k1=0;
50     while((1<<(k1+1))<=x2-x1+1) k1++;
51     int k2=0;
52     while((1<<(k2+1))<=y2-y1+1) k2++;
53     x2=x2-(1<<k1)+1;
54     y2=y2-(1<<k2)+1;
55     return min(min(minn[x1][y1][k1][k2],minn[x1][y2][k1][k2]),min(minn[x2][y1][k1][k2],minn[x2][y2][k1][k2]));
56 }

```

2.3 单调队列

```

1 struct Monotone_queue
2 {
3     #define type int
4     type v[MAX][2]; // 0 is min, 1 is max
5     int p[MAX][2];
6     int l[2],r[2];
7     void clear()
8     {
9         l[0]=r[0]=0;
10        l[1]=r[1]=0;
11    }

```

```

12 void insert(type x,int pos)
13 {
14     while(r[0]-l[0]&&v[r[0]-1][0]>=x) r[0]--;
15     v[r[0]][0]=x;
16     p[r[0]][0]=pos;
17     while(r[1]-l[1]&&v[r[1]-1][1]<=x) r[1]--;
18     v[r[1]][1]=x;
19     p[r[1]][1]=pos;
20 }
21 void erase(int pos)
22 {
23     while(r[0]-l[0]&&p[l[0]][0]<=pos) l[0]++;
24     while(r[1]-l[1]&&p[l[1]][1]<=pos) l[1]++;
25 }
26 type get_min(){return v[l[0]][0];}
27 type get_max(){return v[l[1]][1];}
28 #undef type
29 }dq;

```

2.4 并查集

2.4.1 并查集

```

1 struct Disjoint_Set_Union
2 {
3     int pre[MAX],sz[MAX];
4     void init(int n)
5     {
6         int i;
7         for(i=1;i<=n;i++)
8         {
9             pre[i]=i;
10            sz[i]=1;
11        }
12    }
13    int find(int x)
14    {
15        if(pre[x]!=x) pre[x]=find(pre[x]);
16        return pre[x];
17    }
18    bool merge(int x,int y,bool dir=false)
19    {
20        x=find(x);
21        y=find(y);
22        if(x==y) return 0;
23        if(!dir && sz[x]>sz[y]) swap(x,y);
24        pre[x]=y; // x -> y
25        sz[y]+=sz[x];
26        return 1;
27    }
28 }dsu;

```

2.4.2 map 实现并查集

```

1 struct dsu
2 {
3     #define type int
4     unordered_map<type,type> pre;
5     void init(){pre.clear();}
6     type find(type x)
7     {
8         if(pre.count(x)) pre[x]=find(pre[x]);
9         else return x;
10        return pre[x];
11    }
12    bool merge(type a,type b)
13    {
14        type ra,rb;
15        ra=find(a);
16        rb=find(b);
17        if(ra!=rb)
18        {
19            pre[ra]=rb;
20            return 1;
21        }
22        return 0;
23    }
24    #undef type
25 }dsu;

```

2.4.3 可撤销并查集

```

1 struct Disjoint_Set_Union
2 {
3     pair<int,int> st[MAX];
4     int pre[MAX],top,sz[MAX];
5     void init(int n)
6     {
7         int i;
8         for(i=1;i<=n;i++)
9         {
10            pre[i]=i;
11            sz[i]=1;
12        }
13        top=0;
14    }
15    int find(int x)
16    {
17        while(x!=pre[x]) x=pre[x];
18        return x;
19    }
20    bool merge(int x,int y)
21    {
22        x=find(x);
23        y=find(y);
24        if(x==y) return 0;
25        if(sz[x]>sz[y]) swap(x,y);

```

```

26        pre[x]=y;
27        sz[y]+=sz[x];
28        st[top++]={x,y};
29        return 1;
30    }
31    void roll_back()
32    {
33        pair<int,int> now=st[--top];
34        pre[now.first]=now.first;
35        sz[now.second]-=sz[now.first];
36    }
37 }dsu;

```

2.5 树状数组

2.5.1 一维单点 BIT

```

1 struct Fenwick_Tree
2 {
3     #define type int
4     type bit[MAX],tmp[MAX];
5     int n;
6     void init(int _n)
7     {
8         n=_n;
9         for(int i=0;i<=n;i++) bit[i]=0;
10    }
11    void init(int _n,type *a)
12    {
13        n=_n;
14        tmp[0]=0;
15        for(int i=1;i<=n;i++)
16        {
17            tmp[i]=a[i];
18            tmp[i]+=tmp[i-1];
19            bit[i]=tmp[i]-tmp[i-lowbit(i)];
20        }
21    }
22    int lowbit(int x){return x&(-x);}
23    type get(int x)
24    {
25        type res=0;
26        while(x)
27        {
28            res+=bit[x];
29            x-=lowbit(x);
30        }
31        return res;
32    }
33    void upd(int x,type v)
34    {
35        while(x<=n)
36        {
37            bit[x]+=v;

```

```

38         x+=lowbit(x);
39     }
40 }
41 type ask(int l,int r)
42 {
43     if(l>r) return 0;
44     if(l-1<=0) return get(r);
45     return get(r)-get(l-1);
46 }
47 #undef type
48 }tr;

```

2.5.2 一维区间 BIT

```

1 struct Fenwick_Tree
2 {
3     #define type int
4     type bit[MAX][2];
5     int n;
6     void init(int _n)
7     {
8         n=_n;
9         for(int i=0;i<=n;i++)
10         {
11             bit[i][0]=bit[i][1]=0;
12         }
13     }
14     int lowbit(int x){return x&(-x);}
15     void _insert(int x,type v)
16     {
17         for(int i=x;i<=n;i+=lowbit(i))
18         {
19             bit[i][0]+=v;
20             bit[i][1]+=v*(x-1);
21         }
22     }
23     type get(int x)
24     {
25         type res=0;
26         for(int i=x;i>0;i-=lowbit(i))
27         {
28             res+=x*bit[i][0]-bit[i][1];
29         }
30         return res;
31     }
32     void upd(int l,int r,type v)
33     {
34         _insert(l,v);
35         _insert(r+1,-v);
36     }
37     type ask(int l,int r)
38     {
39         if(l-1<=0) return get(r);
40         return get(r)-get(l-1);

```

```

41     }
42     #undef type
43 }tr;

```

2.5.3 二维单点 BIT

```

1 struct Fenwick_Tree
2 {
3     #define type int
4     type bit[MAX][MAX];
5     int n,m;
6     void init(int _n,int _m)
7     {
8         int i,j;
9         n=_n;
10        m=_m;
11        for(i=0;i<=n;i++)
12        {
13            for(j=0;j<=m;j++) bit[i][j]=0;
14        }
15    }
16    int lowbit(int x){return x&(-x);}
17    type get(int x,int y)
18    {
19        type i,j,res=0;
20        for(i=x;i>0;i-=lowbit(i))
21        {
22            for(j=y;j>0;j-=lowbit(j))
23            {
24                res+=bit[i][j];
25            }
26        }
27        return res;
28    }
29    void upd(int x,int y,type v)
30    {
31        int i,j;
32        for(i=x;i<=n;i+=lowbit(i))
33        {
34            for(j=y;j<=m;j+=lowbit(j))
35            {
36                bit[i][j]+=v;
37            }
38        }
39    }
40    type ask(int x1,int y1,int x2,int y2)
41    {
42        if(x1>x2||y1>y2) return 0;
43        x1--;
44        y1--;
45        return get(x2,y2)-get(x1,y2)-get(x2,y1)+get(
46            x1,y1);
47    }
48    #undef type

```

```
48 }tr;
```

2.6 线段树

2.6.1 线段树

```
1 struct Segment_Tree
2 {
3     #define type int
4     static const type inf=INF;
5     #define ls (id<<1)
6     #define rs (id<<1|1)
7     int n,ql,qv,qop;
8     type a[MAX],tag[MAX<<2],qv;
9     struct node
10    {
11        type v;
12        void init(){
13            }t[MAX<<2],null_node;
14        node merge_node(node x,node y)
15        {
16            node res;
17
18            return res;
19        }
20        void pushup(int id){t[id]=merge_node(t[ls],t[rs]);}
21        void pushdown(int l,int r,int id)
22        {
23            if(!tag[id]) return;
24            int mid=(l+r)>>1;
25
26        }
27        void build(int l,int r,int id)
28        {
29            tag[id]=0;
30            t[id].init();
31            if(l==r)
32            {
33                //init
34                return;
35            }
36            int mid=(l+r)>>1;
37            build(l,mid,ls);
38            build(mid+1,r,rs);
39            pushup(id);
40        }
41        void update(int l,int r,int id)
42        {
43            if(l>=ql&&r<=qr)
44            {
45
46                return;
47            }
```

```
48        pushdown(l,r,id);
49        int mid=(l+r)>>1;
50        if(ql<=mid) update(l,mid,ls);
51        if(qr>mid) update(mid+1,r,rs);
52        pushup(id);
53    }
54    node query(int l,int r,int id)
55    {
56        if(l>=ql&&r<=qr) return t[id];
57        pushdown(l,r,id);
58        int mid=(l+r)>>1;
59        if(qr<=mid) return query(l,mid,ls);
60        if(ql>mid) return query(mid+1,r,rs);
61        return merge_node(query(l,mid,ls),query(mid
62            +1,r,rs));
63    }
64    void build(int _n)
65    {
66        n=_n;
67        build(1,n,1);
68        null_node.init();
69    }
70    void upd(int l,int r,type v)
71    {
72        if(l>r) return;
73        ql=l;
74        qr=r;
75        qv=v;
76        update(1,n,1);
77    }
78    type ask(int l,int r)
79    {
80        if(l>r) return null_node.v;
81        ql=l;
82        qr=r;
83        return query(1,n,1).v;
84    }
85    #undef type
86    #undef ls
87    #undef rs
88 }tr;
89 /*
90 tr.build(n);
91 tr.upd(1,r,v);
92 tr.ask(1,r);
93 Segment_Tree::node res=tr.merge_node(nodex,nodey);
94 */
```

2.6.2 动态开点线段树

```
1 struct Segment_Tree
2 {
3     #define type int
4     static const int LOG=31;
```

```

5  struct node
6  {
7      type v;
8      void init(){v=0;}
9  }t[MAX*LOG],null_node;
10 int root[MAX],ls[MAX*LOG],rs[MAX*LOG],rt,tot,qop
    ;
11 int st[MAX*LOG],top;
12 type ql,qv,qv,n,tag[MAX*LOG];
13 Segment_Tree &operator[](const int _rt){this->rt
    =_rt;return *this;}
14 void init(type _n)
15 {
16     n=_n;
17     rt=1;
18     tot=top=0;
19     t[0].init();
20     ls[0]=rs[0]=0;
21     null_node.init();
22 }
23 void build(){root[rt]=0;}
24 void delnode(int id){st[top++]=id;}
25 int newnode()
26 {
27     int id;
28     id=top?st[--top]:++tot;
29     t[id].init();
30     ls[id]=rs[id]=0;
31     tag[id]=0;
32     return id;
33 }
34 node merge_node(node x,node y)
35 {
36     node res;
37     //
38     return res;
39 }
40 void pushup(int id){t[id]=merge_node(t[ls[id]],t
    [rs[id]]);}
41 void pushdown(type l,type r,int id)
42 {
43     if(!tag[id]) return;
44     if(!ls[id]) ls[id]=newnode();
45     if(!rs[id]) rs[id]=newnode();
46
47     tag[id]=0;
48 }
49 void update(type l,type r,int &id)
50 {
51     if(!id) id=newnode();
52     if(l>=ql&&r<=qr)
53     {
54         //
55 
```

```

56         return;
57     }
58     pushdown(l,r,id);
59     type mid=(l+r)>>1;
60     if(ql<=mid) update(l,mid,ls[id]);
61     if(qr>mid) update(mid+1,r,rs[id]);
62     pushup(id);
63 }
64 node query(type l,type r,int id)
65 {
66     if(!id) return null_node;
67     if(l>=ql&&r<=qr) return t[id];
68     pushdown(l,r,id);
69     type mid=(l+r)>>1;
70     if(qr<=mid) return query(l,mid,ls[id]);
71     if(ql>mid) return query(mid+1,r,rs[id]);
72     return merge_node(query(l,mid,ls[id]),query(
        mid+1,r,rs[id]));
73 }
74 int split(type l,type r,int &id)
75 {
76     int x;
77     if(!id) return 0;
78     if(l>=ql&&r<=qr)
79     {
80         x=id;
81         id=0;
82         return x;
83     }
84     x=newnode();
85     int mid=(l+r)>>1;
86     if(ql<=mid) ls[x]=split(l,mid,ls[id]);
87     if(qr>mid) rs[x]=split(mid+1,r,rs[id]);
88     pushup(x);
89     pushup(id);
90     return x;
91 }
92 int merge(type l,type r,int x,int y)
93 {
94     if(!x||!y) return x+y;
95     if(l==r)
96     {
97         //merge info x <- y
98         delnode(y);
99         return x;
100    }
101    type mid=(l+r)>>1;
102    ls[x]=merge(l,mid,ls[x],ls[y]);
103    rs[x]=merge(mid+1,r,rs[x],rs[y]);
104    pushup(x);
105    delnode(y);
106    return x;
107 }
108 void split_segtree(type l,type r,int new_tree)

```

```

109 {
110     ql=l;
111     qr=r;
112     root[new_tree]=split(1,n,root[rt]);
113 }
114 void merge_segtree(int y)
115 {
116     root[rt]=merge(1,n,root[rt],root[y]);
117     root[y]=0;
118 }
119 void upd(type l,type r,type v)
120 {
121     ql=l;
122     qr=r;
123     qv=v;
124     update(1,n,root[rt]);
125 }
126 type ask(type l,type r)
127 {
128     ql=l;
129     qr=r;
130     return query(1,n,root[rt]).v;
131 }
132 #undef type
133 }tr;
134 /*
135 tr.init(n);
136 tr.build();
137 tr.upd(1,r,v);
138 tr.ask(1,r);
139 Segment_Tree::node res=tr.merge_node(nodex,nodey);
140 -----
141 tr[x].build();
142 tr[x].merge_segtree(y);
143 tr[x].split_segtree(1,r,new_tree);
144 */

```

2.6.3 区间查询最大子段和

```

1 struct node
2 {
3     type sum,lmx,rmx,mx;
4     void init(){sum=0;lmx=rmx=mx=-inf;}
5 }t[MAX<<2],null_node;
6 node merge_node(node x,node y)
7 {
8     node res;
9     res.sum=x.sum+y.sum;
10    res.lmx=max(x.lmx,x.sum+y.lmx);
11    res.rmx=max(y.rmx,y.sum+x.rmx);
12    res.mx=max({x.mx,y.mx,x.rmx+y.lmx});
13    return res;
14 }

```

2.6.4 矩形面积并

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int INF=0x3f3f3f3f;
5 const int MAX=4e5+10;
6 struct Discretization
7 {
8     #define type ll
9     #define pb push_back
10    #define all(x) x.begin(),x.end()
11    vector<type> a;
12    void init(){a.clear();}
13    void add(type x){a.pb(x);}
14    void work(){sort(all(a));a.resize(unique(all(a))
15        -a.begin());}
16    int get_pos(type x){return lower_bound(all(a),x)
17        -a.begin()+1;}
18    type get_val(int pos){return a[pos-1];}
19    int size(){return a.size();}
20    #undef type
21    #undef pb
22    #undef all
23 }dz;
24 struct Segment_Tree
25 {
26     #define type ll
27     #define ls (id<<1)
28     #define rs (id<<1|1)
29     int n,ql,qr;
30     type a[MAX],v[MAX<<2],mn[MAX<<2],tag[MAX<<2],
31         cntmn[MAX<<2],qv;
32     void pushup(int id)
33     {
34         mn[id]=min(mn[ls],mn[rs]);
35         if(mn[ls]==mn[rs]) cntmn[id]=cntmn[ls]+cntmn[rs];
36         else cntmn[id]=(mn[ls]<mn[rs]?cntmn[ls]:cntmn[rs]);
37         v[id]=v[ls]+v[rs];
38     }
39     void maintain(int pre,int now,int id)
40     {
41         if(pre==0&&now>0) v[id]+=cntmn[id];
42         if(pre>0&&now==0) v[id]-=cntmn[id];
43     }
44     void pushdown(int l,int r,int id)
45     {
46         if(!tag[id]) return;
47         int mid=(l+r)>>1;
48         maintain(mn[ls],mn[ls]+tag[id],ls);
49         maintain(mn[rs],mn[rs]+tag[id],rs);
50         mn[ls]+=tag[id];

```

```

48     mn[rs]+=tag[id];
49     tag[ls]+=tag[id];
50     tag[rs]+=tag[id];
51     tag[id]=0;
52 }
53 void build(int l,int r,int id)
54 {
55     tag[id]=0;
56     if(l==r)
57     {
58         mn[id]=0;
59         cntmn[id]=dz.get_val(l)-dz.get_val(l-1);
60         v[id]=0;
61         return;
62     }
63     int mid=(l+r)>>1;
64     build(l,mid,ls);
65     build(mid+1,r,rs);
66     pushup(id);
67 }
68 void update(int l,int r,int id)
69 {
70     if(l>=ql&&r<=qr)
71     {
72         maintain(mn[id],mn[id]+qv,id);
73         mn[id]+=qv;
74         tag[id]+=qv;
75         return;
76     }
77     pushdown(l,r,id);
78     int mid=(l+r)>>1;
79     if(ql<=mid) update(l,mid,ls);
80     if(qr>mid) update(mid+1,r,rs);
81     pushup(id);
82 }
83 type res;
84 void query(int l,int r,int id)
85 {
86     if(l>=ql&&r<=qr)
87     {
88         res+=v[id];
89         return;
90     }
91     pushdown(l,r,id);
92     int mid=(l+r)>>1;
93     if(ql<=mid) query(l,mid,ls);
94     if(qr>mid) query(mid+1,r,rs);
95 }
96 void build(int _n){n=_n;build(1,n,1);}
97 void upd(int l,int r,type v)
98 {
99     if(l>r) return;
100     ql=l;
101     qr=r;

```

```

102     qv=v;
103     update(1,n,1);
104 }
105 type ask(int l,int r)//init res
106 {
107     ql=l;
108     qr=r;
109     res=0;
110     query(1,n,1);
111     return res;
112 }
113 #undef type
114 #undef ls
115 #undef rs
116 }tr;
117 struct node{int pos,l,r,v;};
118 ll work(vector<node> &res)
119 {
120     int i,j,pos;
121     ll ans;
122     sort(res.begin(),res.end(),[&](node x,node y){
123         if(x.pos==y.pos) return x.v>y.v;
124         return x.pos<y.pos;
125     });
126     ans=0;
127     pos=1;
128     tr.build(dz.size());
129     for(i=0;i<res.size();i++)
130     {
131         if(i) ans+=tr.ask(1,dz.size())*(res[i].pos-
132             pos);
133         tr.upd(dz.get_pos(res[i].l),dz.get_pos(res[i]
134             ].r),res[i].v);
135         pos=res[i].pos;
136     }
137     ans+=tr.ask(2,dz.size())*(res[i].pos-pos);
138     return ans;
139 }
140 int main()
141 {
142     int n,i,a,b,c,d;
143     scanf("%d",&n);
144     vector<node> x;
145     dz.init();
146     for(i=1;i<=n;i++)
147     {
148         scanf("%d%d%d%d",&a,&b,&c,&d);
149         dz.add(b);
150         dz.add(b-1);
151         dz.add(d-2);
152         dz.add(d-1);
153         x.push_back({a,b,d-1,1});
154         x.push_back({c,b,d-1,-1});
155     }

```

```

154     dz.work();
155     printf("%lld\n",work(x));
156     return 0;
157 }

```

2.6.5 线段树维护 hash

```

1 struct node
2 {
3     type len,ha;
4     void init(){len=ha=0;}
5 }t[MAX<<2],null_node;
6 node merge_node(node x,node y)
7 {
8     node res;
9     res.len=x.len+y.len;
10    res.ha=x.ha*qpow(bs,y.len)+y.ha;
11    return res;
12 }

```

2.7 平衡树

2.7.1 Treap

```

1 struct Treap
2 {
3     #define type int
4     static const type inf=INF;
5     struct node
6     {
7         int ch[2],fix,sz,cnt;
8         type v;
9         node(){}
10        node(type x,int _sz)
11        {
12            v=x;
13            fix=rand();
14            sz=cnt=_sz;
15            ch[0]=ch[1]=0;
16        }
17    }t[MAX];
18    int tot,root[MAX],rt;
19    void init(int n=1)
20    {
21        for(int i=0;i<=n;i++) root[i]=0;
22        rt=1;
23        srand(time(0));
24        tot=0;
25        t[0].sz=t[0].cnt=0;
26        memset(t[0].ch,0,sizeof t[0].ch);
27    }
28    void pushup(int id)
29    {

```

```

30        t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
31        t[id].cnt;
32    }
33    void rotate(int &id,int k)
34    {
35        int y=t[id].ch[k^1];
36        t[id].ch[k^1]=t[y].ch[k];
37        t[y].ch[k]=id;
38        pushup(id);
39        pushup(y);
40        id=y;
41    }
42    void _insert(int &id,type v,int cnt)
43    {
44        if(!id)
45        {
46            id=++tot;
47            t[id]=node(v,cnt);
48            return;
49        }
50        if(t[id].v==v) t[id].cnt+=cnt;
51        else
52        {
53            int tmp=(v>t[id].v);
54            _insert(t[id].ch[tmp],v,cnt);
55            if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
56            (id,tmp^1);
57        }
58        pushup(id);
59    }
60    void _erase(int &id,type v,int cnt)
61    {
62        if(!id) return;
63        if(t[id].v==v)
64        {
65            cnt=min(t[id].cnt,cnt);
66            if(t[id].cnt>cnt)
67            {
68                t[id].cnt-=cnt;
69                pushup(id);
70                return;
71            }
72            if(!(t[id].ch[0]&& t[id].ch[1]))
73            {
74                id=t[id].ch[0]+t[id].ch[1];
75                return;
76            }
77            else
78            {
79                int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
80                [1]].fix);

```



```

81     }
82 }
83 else
84 {
85     _erase(t[id].ch[v>t[id].v],v,cnt);
86     pushup(id);
87 }
88 }
89 int _find(type key,int f)
90 {
91     int id=root[rt],res=0;
92     while(id)
93     {
94         if(t[id].v<=key)
95         {
96             res+=t[t[id].ch[0]].sz+t[id].cnt;
97             if(f&&key==t[id].v) res-=t[id].cnt;
98             id=t[id].ch[1];
99         }
100        else id=t[id].ch[0];
101    }
102    return res;
103 }
104 type find_by_order(int k)//k small
105 {
106     int id=root[rt];
107     if(id==0) return 0;
108     while(id)
109     {
110         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
111         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
112             return t[id].v;
113         else
114         {
115             k-=t[t[id].ch[0]].sz+t[id].cnt;
116             id=t[id].ch[1];
117         }
118     }
119     return 0;
120 }
121 int count(type key)
122 {
123     int id=root[rt];
124     while(id)
125     {
126         if(t[id].v<key)
127         {
128             if(key==t[id].v) return t[id].cnt;
129             id=t[id].ch[1];
130         }
131         else id=t[id].ch[0];
132     }
133     return 0;

```

```

134 type find_pre(type key)
135 {
136     type res=-inf;
137     int id=root[rt];
138     while(id)
139     {
140         if(t[id].v<key)
141         {
142             res=t[id].v;
143             id=t[id].ch[1];
144         }
145         else id=t[id].ch[0];
146     }
147     return res;
148 }
149 type find_nex(type key)
150 {
151     type res=inf;
152     int id=root[rt];
153     while(id)
154     {
155         if(t[id].v>key)
156         {
157             res=t[id].v;
158             id=t[id].ch[0];
159         }
160         else id=t[id].ch[1];
161     }
162     return res;
163 }
164 Treap &operator()(const int _rt){this->rt=_rt;
165     return *this;}
166 void insert(type v,int sz=1){_insert(root[rt],v,
167     sz);}
168 void erase(type v,int sz=1){_erase(root[rt],v,sz
169     );}
170 int upper_bound_count(type key){return _find(key
171     ,0);}//the count <=key
172 int lower_bound_count(type key){return _find(key
173     ,1);}//the count <key
174 int order_of_key(type key){return
175     lower_bound_count(key)+1;}
176 int size(){return t[root[rt]].sz;}
177 #undef type
178 }tr;
179 /*
180 1 treap
181 tr.init();
182 tr.insert(x);
183 tr.erase(x);
184 tr.count(x);
185 tr.order_of_key(x); // rank
186 tr.find_by_order(k); // kth
187 tr.find_pre(x);

```

```

182 tr.find_nex(x);
183 tr.upper_bound_count(x); //the count <=key
184 tr.lower_bound_count(x); //the count <key
185
186 n treap
187 tr.init(n);
188 tr[i].insert(x);
189 */

```

2.7.2 Splay 维护序列

```

1 struct Splay
2 {
3     #define type int
4     const type inf=INF;
5     const type zero=0;
6     struct node
7     {
8         int ch[2],fa,sz,cnt,rev,tag;
9         type v;
10    }t[MAX];
11    int tot,root;
12    type a[MAX];
13    queue<int> pool;
14    void init_null_node()
15    {
16        memset(t[0].ch,0,sizeof t[0].ch);
17        t[0].sz=t[0].cnt=t[0].fa=0;
18        t[0].v=zero;
19        t[0].mnid=0;
20    }
21    void init()
22    {
23        root=tot=0;
24        while(!pool.empty()) pool.pop();
25        init_null_node();
26        a[0]=a[1]=zero;
27        root=build(0,1,0);
28    }
29    int newnode(type v,int fa)
30    {
31        int id;
32        if(pool.size()>0)
33        {
34            id=pool.front();
35            pool.pop();
36        }
37        else id=++tot;
38        memset(t[id].ch,0,sizeof t[id].ch);
39        t[id].fa=fa;
40        t[id].sz=t[id].cnt=1;
41        t[id].tag=t[id].rev=0;
42        t[id].v=v;
43        return id;

```

```

44    }
45    void pushup(int id)
46    {
47        int ls=t[id].ch[0];
48        int rs=t[id].ch[1];
49        t[id].sz=t[ls].sz+t[rs].sz+t[id].cnt;
50
51
52    }
53    void pushdown(int id)
54    {
55        int ls=t[id].ch[0];
56        int rs=t[id].ch[1];
57        if(t[id].tag)
58        {
59            if(ls)
60            {
61
62                t[ls].tag=1;
63            }
64            if(rs)
65            {
66
67                t[rs].tag=1;
68            }
69            t[id].tag=0;
70        }
71        if(t[id].rev)
72        {
73            t[ls].rev^=1;
74            t[rs].rev^=1;
75            swap(t[ls].ch[0],t[ls].ch[1]);
76            swap(t[rs].ch[0],t[rs].ch[1]);
77            t[id].rev=0;
78        }
79    }
80    void insert(int pos,vector<int> nums)
81    {
82        int x,y,z,i;
83        for(i=0;i<nums.size();i++) a[i+1]=nums[i];
84        z=build(1,nums.size(),0);
85        x=find(pos);
86        y=find(pos+1);
87        splay(x,0);
88        splay(y,x);
89        t[y].ch[0]=z;
90        t[z].fa=y;
91        pushup(y);
92        pushup(x);
93    }
94    void rotate(int x)
95    {
96        int y,z,k;
97        y=t[x].fa;

```

```

98     z=t[y].fa;
99     k=(x==t[y].ch[1]);
100    t[y].ch[k]=t[x].ch[k^1];
101    if(t[x].ch[k^1]) t[t[x].ch[k^1]].fa=y;
102    t[x].ch[k^1]=y;
103    t[y].fa=x;
104    t[x].fa=z;
105    if(z) t[z].ch[y==t[z].ch[1]]=x;
106    pushup(y);
107    pushup(x);
108 }
109 void splay(int x,int goal)
110 {
111     int y,z;
112     while(t[x].fa!=goal)
113     {
114         y=t[x].fa;
115         z=t[y].fa;
116         if(z!=goal)
117         {
118             if((t[z].ch[0]==y)^(t[y].ch[0]==x))
119                 rotate(x);
120             else rotate(y);
121         }
122         rotate(x);
123     }
124     if(goal==0) root=x;
125 int kth(int k)//k small
126 {
127     int id=root;
128     while(id)
129     {
130         pushdown(id);
131         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
132         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
133             break;
134         else
135         {
136             k-=t[t[id].ch[0]].sz+t[id].cnt;
137             id=t[id].ch[1];
138         }
139     }
140     splay(id,0);
141     return id;
142 }
143 int find(int x){return kth(x+1);}
144 int build(int l,int r,int fa)
145 {
146     if(l>r) return 0;
147     int id,mid;
148     mid=(l+r)>>1;
149     id=newnode(a[mid],fa);
150     t[id].ch[0]=build(l,mid-1,id);

```

```

150     t[id].ch[1]=build(mid+1,r,id);
151     pushup(id);
152     return id;
153 }
154 void _del(int id)
155 {
156     if(!id) return;
157     pool.push(id);
158     _del(t[id].ch[0]);
159     _del(t[id].ch[1]);
160 }
161 void erase(int l,int r)
162 {
163     int x,fa;
164     x=split(l,r);
165     fa=t[x].fa;
166     t[fa].ch[0]=0;
167     _del(x);
168     pushup(fa);
169     pushup(t[fa].fa);
170 }
171 int split(int l,int r)
172 {
173     int x,y;
174     x=find(l-1);
175     y=find(r+1);
176     splay(x,0);
177     splay(y,x);
178     return t[y].ch[0];
179 }
180 void rev(int l,int r)
181 {
182     int x,fa;
183     x=split(l,r);
184     fa=t[x].fa;
185     t[x].rev^=1;
186     swap(t[x].ch[0],t[x].ch[1]);
187     pushup(fa);
188     pushup(t[fa].fa);
189 }
190 int ask(int l,int r)
191 {
192     int x=split(l,r);
193     return ;
194 }
195 int size(){return t[root].sz-2;}
196 #undef type
197 }tr; //tr.init();

```

2.7.3 FHQ-Treap 维护序列

```

1 struct FHQ_Treap
2 {
3     #define type int

```

```

4  static const type inf=INF;
5  struct node
6  {
7      int ls,rs,fix,sz;
8      type v;
9      node(){}
10     node(type x,int _sz)
11     {
12         v=x;
13         fix=rand();
14         sz=_sz;
15         ls=rs=0;
16     }
17 }t[MAX];
18 int st[MAX];
19 int revtag[MAX];
20 int rt,tot;
21 void init()
22 {
23     rt=0;
24     srand(time(0));
25     tot=0;
26     t[0].sz=0;
27     t[0].ls=t[0].rs=0;
28 }
29 int newnode(type v)
30 {
31     t[++tot]=node(v,1);
32     revtag[tot]=0;
33     return tot;
34 }
35 void pushup(int id)
36 {
37     t[id].sz=t[t[id].ls].sz+t[t[id].rs].sz+1;
38 }
39 void pushdown(int id)
40 {
41     if(revtag[id])
42     {
43         int ls=t[id].ls;
44         int rs=t[id].rs;
45         swap(t[ls].ls,t[ls].rs);
46         swap(t[rs].ls,t[rs].rs);
47         revtag[ls]^=revtag[id];
48         revtag[rs]^=revtag[id];
49         revtag[id]=0;
50     }
51 }
52 int build(vector<type> &a)
53 {
54     int id,k,top=0;
55     for(auto &it:a)
56     {
57         id=newnode(it);

```

```

58         k=top;
59         while(k>0&&t[st[k-1]].fix>t[id].fix)
60             pushup(st[--k]);
61         if(k) t[st[k-1]].rs=id;
62         if(k<top) t[id].ls=st[k];
63         st[k++]=id;
64         top=k;
65     }
66     while(top>0) pushup(st[--top]);
67     return st[0];
68 }
69 void split(int id,int pos,int &x,int &y)
70 {
71     if(!id)
72     {
73         x=y=0;
74         return;
75     }
76     pushdown(id);
77     int tmp=t[t[id].ls].sz+1;
78     if(tmp<=pos)
79     {
80         x=id;
81         split(t[id].rs,pos-tmp,t[id].rs,y);
82     }
83     else
84     {
85         y=id;
86         split(t[id].ls,pos,x,t[id].ls);
87     }
88     pushup(id);
89 }
90 int merge(int x,int y)
91 {
92     if(!x||!y) return x|y;
93     int id;
94     if(t[x].fix<t[y].fix)
95     {
96         pushdown(x);
97         id=x;
98         t[x].rs=merge(t[x].rs,y);
99     }
100    else
101    {
102        pushdown(y);
103        id=y;
104        t[y].ls=merge(x,t[y].ls);
105    }
106    pushup(id);
107    return id;
108 }
109 void insert(int pos,type v)
110 {
111     int ra,rb;

```

```

111     split(rt,pos-1,ra,rb);
112     rt=merge(merge(ra,newnode(v)),rb);
113 }
114 void insert(int pos,vector<type> a)
115 {
116     int ra,rb;
117     split(rt,pos-1,ra,rb);
118     rt=merge(merge(ra,build(a)),rb);
119 }
120 void erase(int pos,type v)
121 {
122     int ra,rb,rc;
123     split(rt,pos,ra,rc);
124     split(ra,pos-1,ra,rb);
125     rt=merge(ra,rc);
126 }
127 void rev(int l,int r)
128 {
129     int ra,rb,rc;
130     split(rt,r,ra,rc);
131     split(ra,l-1,ra,rb);
132     revtag[rb]^=1;
133     swap(t[rb].ls,t[rb].rs);
134     rt=merge(merge(ra,rb),rc);
135 }
136 vector<int> res;
137 void dfs(int id)
138 {
139     if(!id) return;
140     pushdown(id);
141     dfs(t[id].ls);
142     res.push_back(t[id].v);
143     dfs(t[id].rs);
144 }
145 int size(){return t[rt].sz;}
146 #undef type
147 }tr;

```

2.7.4 pbds

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 tree<int,
5     null_type,
6     less<int>,
7     rb_tree_tag,
8     tree_order_statistics_node_update> tr;

```

2.8 字典树

2.8.1 trie

```

1 struct Trie
2 {
3     #define type int
4     static const int K=26;
5     int nex[MAX][K],cnt[MAX];
6     int root,tot;
7     int getid(char c){return c-'a';}
8     int newnode()
9     {
10         tot++;
11         memset(nex[tot],0,sizeof nex[tot]);
12         cnt[tot]=0;
13         return tot;
14     }
15     void init()
16     {
17         tot=0;
18         memset(nex[0],0,sizeof nex[0]);
19         cnt[0]=0;
20         root=newnode();
21     }
22     void insert(char *s,int n) // s[0..n-1]
23     {
24         int now,i,t;
25         now=root;
26         for(i=0;i<n;i++)
27         {
28             t=getid(s[i]);
29             if(!nex[now][t]) nex[now][t]=newnode();
30             now=nex[now][t];
31         }
32         cnt[now]++;
33     }
34     #undef type
35 }tr;
36 /*
37 tr.init();
38 tr.insert(s,len); s[0..len-1]
39 */

```

2.8.2 01trie

```

1 struct Trie
2 {
3     #define type int
4     static const int LOG=30;
5     static const int K=LOG+2;
6     static const type inf=INF;
7     int rt,root[MAX],tot,ch[MAX*K][2],cnt[MAX*K];
8     Trie &operator()(const int _rt){this->rt=_rt;
9         return *this;}
10     int newnode()
11     {

```

```

11     tot++;
12     memset(ch[tot],0,sizeof ch[tot]);
13     cnt[tot]=0;
14     return tot;
15 }
16 void init(int n=1)
17 {
18     ch[0][0]=ch[0][1]=cnt[0]=0;
19     tot=1;
20     for(int i=0;i<=n;i++) root[i]=0;
21     rt=1;
22 }
23 void insert(type x)
24 {
25     int id,t,i;
26     if(!root[rt]) root[rt]=newnode();
27     id=root[rt];
28     for(i=LOG;~i;i--)
29     {
30         cnt[id]++;
31         t=(x>>i)&1;
32         if(!ch[id][t]) ch[id][t]=newnode();
33         id=ch[id][t];
34     }
35     cnt[id]++;
36 }
37 type ask_max(type x)
38 {
39     int id,i;
40     type res,t;
41     if(!root[rt]) return -inf;
42     id=root[rt];
43     res=0;
44     for(i=LOG;~i;i--)
45     {
46         t=(x>>i)&1;
47         if(ch[id][t^1]&&cnt[ch[id][t^1]]) t^=1;
48         res|=(t<<i);
49         id=ch[id][t];
50     }
51     return res^x;
52 }
53 type ask_min(type x)
54 {
55     int id,i;
56     type res,t;
57     if(!root[rt]) return inf;
58     id=root[rt];
59     res=0;
60     for(i=LOG;~i;i--)
61     {
62         t=(x>>i)&1;
63         if(!ch[id][t]||!cnt[ch[id][t]]) t^=1;
64         res|=(t<<i);

```

```

65         id=ch[id][t];
66     }
67     return res^x;
68 }
69 #undef type
70 }tr;

```

2.9 可持久化

2.9.1 可持久化线段树

```

1 struct Persistent_Segment_Tree
2 {
3     #define type int
4     int root[MAX],ls[MAX<<5],rs[MAX<<5],tot,ql,qr,n;
5     type v[MAX<<5],qv,res;
6     void init(int _n)
7     {
8         n=_n;
9         root[0]=0;
10        ls[0]=rs[0]=v[0]=tot=0;
11    }
12    int copy_node(int x)
13    {
14        tot++;
15        ls[tot]=ls[x];
16        rs[tot]=rs[x];
17        v[tot]=v[x];
18        return tot;
19    }
20    void update(int l,int r,int &id,int pre)
21    {
22        if(!id) id=copy_node(pre);
23        v[id]+=qv;
24        if(l==r) return;
25        int mid=(l+r)>>1;
26        if(ql<=mid) update(l,mid,ls[id]=0,ls[pre]);
27        else update(mid+1,r,rs[id]=0,rs[pre]);
28    }
29    void query_valsum(int l,int r,int id,int pre)
30    {
31        if(!id) return;
32        if(l>=ql&&r<=qr)
33        {
34            res+=v[id]-v[pre];
35            return;
36        }
37        int mid=(l+r)>>1;
38        if(ql<=mid) query_valsum(l,mid,ls[id],ls[pre]);
39        if(qr>mid) query_valsum(mid+1,r,rs[id],rs[pre]);
40    }
41    int kth_small(int l,int r,int id,int pre,int k)

```

```

42 {
43     if(l==r) return 1;
44     int mid=(l+r)>>1;
45     int tmp=v[ls[id]]-v[ls[pre]];
46     if(tmp>=k) return kth_small(1,mid,ls[id],ls[
47         pre],k);
48     else return kth_small(mid+1,r,rs[id],rs[pre],
49         k-tmp);
50 }
51 void update_ver(int now_ver,int pre_ver,int pos,
52     type v)
53 {
54     ql=qr=pos;
55     qv=v;
56     update(1,n,root[now_ver],root[pre_ver]);
57 }
58 void copy_ver(int now_ver,int pre_ver)
59 {
60     root[now_ver]=root[pre_ver];
61 }
62 void create_ver(int now_ver,int pre_ver,int pos,
63     type v)
64 {
65     root[now_ver]=0;
66     update_ver(now_ver,pre_ver,pos,v);
67 }
68 int ask_kth_small(int l,int r,int k)
69 {
70     return kth_small(1,n,root[r],root[l-1],k);
71 }
72 type ask_valsum(int pre_ver,int now_ver,int
73     val_l,int val_r)
74 {
75     ql=val_l;
76     qr=val_r;
77     res=0;
78     query_valsum(1,n,root[now_ver],root[pre_ver
79         -1]);
80     return res;
81 }
82 #undef type
83 }tr;
84 /*
85 tr.init(n);
86 tr.create_ver(now_ver,pre_ver,pos,v);
87 tr.copy_ver(now_ver,pre_ver);
88 tr.ask_kth_small(1,r,k);
89 tr.ask_valsum(1,r,val_l,val_r);
90 */

```

2.9.2 可持久化 01trie

```

1 struct Persistent_01Trie
2 {

```

```

3 #define type int
4 static const int LOG=30;
5 static const int K=LOG+2;
6 int root[MAX],tot,nex[MAX*K][2],cnt[MAX*K];
7 void init()
8 {
9     root[0]=0;
10    nex[0][0]=nex[0][1]=cnt[0]=tot=0;
11 }
12 int newnode()
13 {
14     tot++;
15     memset(nex[tot],0,sizeof nex[tot]);
16     cnt[tot]=0;
17     return tot;
18 }
19 void update(type x,int &rt,int pre)
20 {
21     int id,t,i;
22     if(!rt) rt=newnode();
23     id=rt;
24     for(i=LOG;~i;i--)
25     {
26         cnt[id]=cnt[pre]+1;
27         t=(x>>i)&1;
28         nex[id][t^1]=nex[pre][t^1];
29         nex[id][t]=newnode();
30         id=nex[id][t];
31         pre=nex[pre][t];
32     }
33     cnt[id]=cnt[pre]+1;
34 }
35 type query_max(type x,int rt,int pre)
36 {
37     int id,i;
38     type res,t;
39     id=rt;
40     res=0;
41     for(i=LOG;~i;i--)
42     {
43         t=(x>>i)&1;
44         if(cnt[nex[id][t^1]]-cnt[nex[pre][t
45             ^1]]>0) t^=1;
46         res|=(t<<i);
47         id=nex[id][t];
48         pre=nex[pre][t];
49     }
50     return res^x;
51 }
52 void copy_ver(int now_ver,int pre_ver)
53 {
54     root[now_ver]=root[pre_ver];
55 }
56 void create_ver(int now_ver,int pre_ver,type x)

```

```

56     {
57         root[now_ver]=0;
58         update(x,root[now_ver],root[pre_ver]);
59     }
60     type ask_max(int pre_ver,int now_ver,type x)
61     {
62         assert(pre_ver>0);
63         return query_max(x,root[now_ver],root[pre_ver]-1]);
64     }
65     #undef type
66 }tr;
67 /*
68 tr.init();
69 tr.create_ver(now_ver,pre_ver,v);
70 tr.copy_ver(now_ver,pre_ver);
71 tr.ask_max(l,r,v);
72 */

```

2.9.3 可持久化数组

```

1 struct Persistent_Array
2 {
3     #define type int
4     struct node{int ls,rs;type v;}t[MAX<<5];
5     int root[MAX],tot,qx,n;
6     type a[MAX],qv;
7     void init(int _n)
8     {
9         n=_n;
10        tot=root[0]=0;
11        t[0]={0,0,0};
12    }
13    void build(int l,int r,int &id)
14    {
15        if(!id) id=++tot;
16        t[id]={0,0,0};
17        if(l==r)
18        {
19            t[id]={0,0,a[l]};
20            return;
21        }
22        int mid=(l+r)>>1;
23        build(l,mid,t[id].ls);
24        build(mid+1,r,t[id].rs);
25    }
26    int copy_node(int x)
27    {
28        t[++tot]=t[x];
29        return tot;
30    }
31    void update(int l,int r,int &id,int pre)
32    {
33        if(!id) id=copy_node(pre);

```

```

34        if(l==r)
35        {
36            t[id].v=qv;
37            return;
38        }
39        int mid=(l+r)>>1;
40        if(qx<=mid) update(l,mid,t[id].ls=0,t[pre].ls);
41        else update(mid+1,r,t[id].rs=0,t[pre].rs);
42    }
43    type query(int l,int r,int id)
44    {
45        if(!id) return 0;
46        if(l==r) return t[id].v;
47        int mid=(l+r)>>1;
48        if(qx<=mid) return query(l,mid,t[id].ls);
49        else return query(mid+1,r,t[id].rs);
50    }
51    void build_ver(int now_ver){build(1,n,root[
52        now_ver]=0);}
53    void clear_ver(int now_ver){root[now_ver]=0;}
54    void copy_ver(int now_ver,int pre_ver){root[
55        now_ver]=root[pre_ver];}
56    void create_ver(int now_ver,int pre_ver,int pos,
57        type v)
58    {
59        root[now_ver]=0;
60        qx=pos;
61        qv=v;
62        update(1,n,root[now_ver],root[pre_ver]);
63    }
64    void update_ver(int now_ver,int pos,type v)
65    {
66        qx=pos;
67        qv=v;
68        update(1,n,root[now_ver],root[now_ver]);
69    }
70    type ask(int now_ver,int pos)
71    {
72        qx=pos;
73        return query(1,n,root[now_ver]);
74    }
75    #undef type
76 }pa;
77 /*
78 pa.init(n);
79 pa.build_ver(now_ver);
80 pa.clear_ver(now_ver);
81 pa.copy_ver(now_ver,pre_ver);
82 pa.create_ver(now_ver,pre_ver,pos,v);
83 pa.update_ver(now_ver,pos,v);
84 pa.ask(now_ver,pos);
85 */

```


2.9.4 可持久化并查集

```

1 struct Persistent_DSU
2 {
3     struct node{int ls,rs,fa,dep;}t[MAX<<5];
4     int root[MAX],tot,n,qx,qop,qv;
5     void build(int l,int r,int &id)
6     {
7         if(!id) id=++tot;
8         t[id]={0,0,0,0};
9         if(l==r)
10            {
11                t[id]={0,0,l,1};
12                return;
13            }
14         int mid=(l+r)>>1;
15         build(l,mid,t[id].ls);
16         build(mid+1,r,t[id].rs);
17     }
18     void init(int _n,int init_ver)
19     {
20         n=_n;
21         tot=root[0]=0;
22         t[0]={0,0,0,0};
23         build(1,n,root[init_ver]);
24     }
25     int copy_node(int x)
26     {
27         t[++tot]=t[x];
28         return tot;
29     }
30     void update(int l,int r,int &id,int pre)
31     {
32         if(!id) id=copy_node(pre);
33         if(l==r)
34            {
35                if(qop==1) t[id].fa=qv;
36                else t[id].dep+=qv;
37                return;
38            }
39         int mid=(l+r)>>1;
40         if(qx<=mid) update(l,mid,t[id].ls=0,t[pre].ls);
41         else update(mid+1,r,t[id].rs=0,t[pre].rs);
42     }
43     int query(int l,int r,int id)
44     {
45         if(!id) return 0;
46         if(l==r) return id;
47         int mid=(l+r)>>1;
48         if(qx<=mid) return query(l,mid,t[id].ls);
49         else return query(mid+1,r,t[id].rs);
50     }

```

```

51 void create_ver(int now_ver,int pre_ver,int pos,
52               int v,int op)
53 {
54     root[now_ver]=0;
55     qx=pos;
56     qv=v;
57     qop=op;
58     update(1,n,root[now_ver],root[pre_ver]);
59 }
60 void update_ver(int now_ver,int pos,int v,int op)
61 {
62     qx=pos;
63     qv=v;
64     qop=op;
65     update(1,n,root[now_ver],root[now_ver]);
66 }
67 int ask_id(int now_ver,int pos)
68 {
69     qx=pos;
70     return query(1,n,root[now_ver]);
71 }
72 void clear_ver(int now_ver){root[now_ver]=0;}
73 void copy_ver(int now_ver,int pre_ver){root[
74     now_ver]=root[pre_ver];}
75 int find_id(int now_ver,int x)
76 {
77     int id;
78     while(1)
79     {
80         id=ask_id(now_ver,x);
81         if(x==t[id].fa) return id;
82         x=t[id].fa;
83     }
84     return 0;
85 }
86 int find(int now_ver,int x){return t[find_id(
87     now_ver,x)].fa;}
88 bool merge(int now_ver,int pre_ver,int a,int b)
89 {
90     int raid,rbid;
91     raid=find_id(pre_ver,a);
92     rbid=find_id(pre_ver,b);
93     if(t[raid].fa==t[rbid].fa)
94     {
95         copy_ver(now_ver,pre_ver);
96         return 0;
97     }
98     if(t[raid].dep>t[rbid].dep) swap(raid,rbid);
99     if(now_ver==pre_ver) update_ver(now_ver,t[
100         raid].fa,t[rbid].fa,1);
101     else create_ver(now_ver,pre_ver,t[raid].fa,t[
102         rbid].fa,1);

```

```

98         if(t[raid].dep==t[rbid].dep) update_ver(
          now_ver,t[rbid].fa,1,2);
99         return 1;
100     }
101 }dsu;
102 /*
103 dsu.init(n,init_ver);
104 dsu.find(now_ver,x);
105 dsu.merge(now_ver,pre_ver,a,b);
106 dsu.clear_ver(now_ver);
107 dsu.copy_ver(now_ver,pre_ver);
108 */

```

2.10 树套树

2.10.1 线段树套线段树

```

1 struct Segment_Tree_2D
2 {
3     #define type int
4     static const int insert_num=;
5     static const int N=insert_num*20*20; //
        insert_num*20*log(m)
6     int root[MAX<<2],tot,ls[N],rs[N],n,m;
7     int ql_in,qr_in,ql_out,qr_out;
8     type v[N],qv,tag[N];
9     void init(int _n,int _m)
10    {
11        n=_n;
12        m=_m;
13        mem(root,0);
14        ls[0]=rs[0]=0;
15        tag[0]=0;
16        v[0]=0;
17        tot=1;
18    }
19    int newnode()
20    {
21        ls[tot]=rs[tot]=0;
22        v[tot]=0;
23        tag[tot]=0;
24        return tot++;
25    }
26    void pushup(int id)
27    {
28    }
29    void pushdown(int id)
30    {
31    }
32    if(!tag[id]) return;
33    if(!ls[id]) ls[id]=newnode();
34    if(!rs[id]) rs[id]=newnode();
35
36

```

```

37     tag[id]=0;
38 }
39 void update_in(int l,int r,int &id)
40 {
41     if(!id) id=newnode();
42     if(l>=ql_in&&r<=qr_in)
43     {
44         v[id]+=qv; //must update not =
45         tag[id]+=qv;
46         return;
47     }
48     pushdown(id);
49     int mid=(l+r)>>1;
50     if(ql_in<=mid) update_in(l,mid,ls[id]);
51     if(qr_in>mid) update_in(mid+1,r,rs[id]);
52     pushup(id);
53 }
54 type res_in;
55 void query_in(int l,int r,int &id)
56 {
57     if(!id) return;
58     if(l>=ql_in&&r<=qr_in)
59     {
60         res_in+=v[id];
61         return;
62     }
63     pushdown(id);
64     int mid=(l+r)>>1;
65     type res=0;
66     if(ql_in<=mid) query_in(l,mid,ls[id]);
67     if(qr_in>mid) query_in(mid+1,r,rs[id]);
68 }
69 /*
        *****
        */
70 #define ls (id<<1)
71 #define rs (id<<1|1)
72 void update_out(int l,int r,int id)
73 {
74     update_in(1,m,root[id]);
75     if(l>=ql_out&&r<=qr_out) return;
76     int mid=(l+r)>>1;
77     if(ql_out<=mid) update_out(l,mid,ls);
78     if(qr_out>mid) update_out(mid+1,r,rs);
79 }
80 type res_out;
81 void query_out(int l,int r,int id)
82 {
83     if(l>=ql_out&&r<=qr_out)
84     {
85         res_in=0;
86         query_in(1,m,root[id]);
87         res_out+=res_in;
88         return;

```

```

89     }
90     int mid=(l+r)>>1;
91     if(q1_out<=mid) query_out(l,mid,ls);
92     if(qr_out>mid) query_out(mid+1,r,rs);
93 }
94 #undef ls
95 #undef rs
96 void upd(int x1,int y1,int x2,int y2,type val)
97 {
98     q1_out=x1;
99     qr_out=x2;
100    q1_in=y1;
101    qr_in=y2;
102    qv=val;
103    update_out(1,n,1);
104 }
105 type ask(int x1,int y1,int x2,int y2)
106 {
107     q1_out=x1;
108     qr_out=x2;
109     q1_in=y1;
110     qr_in=y2;
111     res_out=0;
112     query_out(1,n,1);
113     return res_out;
114 }
115 #undef type
116 }tr2d;

```

2.10.2 线段树套 treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int INF=0x3f3f3f3f;
5 const int MAX=5e4+10;
6 struct Treap
7 {
8     #define type int
9     #define inf INF
10    struct node
11    {
12        int ch[2],fix,sz,cnt;
13        type v;
14        node(){}
15        node(type x,int _sz)
16        {
17            v=x;
18            fix=rand();
19            sz=cnt=_sz;
20            ch[0]=ch[1]=0;
21        }
22    }t[MAX*40];
23    int tot,root[MAX<<2],rt;

```

```

24 void init(int n=1)
25 {
26     for(int i=0;i<=n;i++) root[i]=0;
27     rt=1;
28     srand(time(0));
29     tot=0;
30     t[0].sz=t[0].cnt=0;
31     memset(t[0].ch,0,sizeof t[0].ch);
32 }
33 void pushup(int id)
34 {
35     t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
36         t[id].cnt;
37 }
38 void rotate(int &id,int k)
39 {
40     int y=t[id].ch[k^1];
41     t[id].ch[k^1]=t[y].ch[k];
42     t[y].ch[k]=id;
43     pushup(id);
44     pushup(y);
45     id=y;
46 }
47 void _insert(int &id,type v,int cnt)
48 {
49     if(!id)
50     {
51         id=++tot;
52         t[id]=node(v,cnt);
53         return;
54     }
55     if(t[id].v==v) t[id].cnt+=cnt;
56     else
57     {
58         int tmp=(v>t[id].v);
59         _insert(t[id].ch[tmp],v,cnt);
60         if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
61             (id,tmp^1);
62     }
63     pushup(id);
64 }
65 void _erase(int &id,type v,int cnt)
66 {
67     if(!id) return;
68     if(t[id].v==v)
69     {
70         cnt=min(t[id].cnt,cnt);
71         if(t[id].cnt>cnt)
72         {
73             t[id].cnt-=cnt;
74             pushup(id);
75             return;
76         }
77         if(!(t[id].ch[0]&&t[id].ch[1]))

```

```

76     {
77         id=t[id].ch[0]+t[id].ch[1];
78         return;
79     }
80     else
81     {
82         int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
83             [1]].fix);
84         rotate(id,tmp);
85         _erase(t[id].ch[tmp],v,cnt);
86         pushup(id);
87     }
88     else
89     {
90         _erase(t[id].ch[v>t[id].v],v,cnt);
91         pushup(id);
92     }
93 }
94 int _find(type key,int f)
95 {
96     int id=root[rt],res=0;
97     while(id)
98     {
99         if(t[id].v<key)
100         {
101             res+=t[t[id].ch[0]].sz+t[id].cnt;
102             if(f&&key==t[id].v) res-=t[id].cnt;
103             id=t[id].ch[1];
104         }
105         else id=t[id].ch[0];
106     }
107     return res;
108 }
109 type find_by_order(int k)//k small
110 {
111     int id=root[rt];
112     if(id==0) return 0;
113     while(id)
114     {
115         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
116         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
117             return t[id].v;
118         else
119         {
120             k-=t[t[id].ch[0]].sz+t[id].cnt;
121             id=t[id].ch[1];
122         }
123     }
124 }
125 int count(type key)
126 {
127     int id=root[rt];
128     while(id)

```

```

128     {
129         if(t[id].v<key)
130         {
131             if(key==t[id].v) return t[id].cnt;
132             id=t[id].ch[1];
133         }
134         else id=t[id].ch[0];
135     }
136     return 0;
137 }
138 type find_pre(type key)
139 {
140     type res=-inf;
141     int id=root[rt];
142     while(id)
143     {
144         if(t[id].v<key)
145         {
146             res=t[id].v;
147             id=t[id].ch[1];
148         }
149         else id=t[id].ch[0];
150     }
151     return res;
152 }
153 type find_nex(type key)
154 {
155     type res=inf;
156     int id=root[rt];
157     while(id)
158     {
159         if(t[id].v>key)
160         {
161             res=t[id].v;
162             id=t[id].ch[0];
163         }
164         else id=t[id].ch[1];
165     }
166     return res;
167 }
168 Treap &operator[(const int _rt)]{this->rt=_rt;
169     return *this;}
170 void insert(type v,int sz=1){_insert(root[rt],v,
171     sz);}
172 void erase(type v,int sz=1){_erase(root[rt],v,sz
173     );}
174 int upper_bound_count(type key){return _find(key
175     ,0);}//the count <=key
176 int lower_bound_count(type key){return _find(key
177     ,1);}//the count <key
178 int order_of_key(type key){return
179     lower_bound_count(key)+1;}
180 int size(){return t[root[rt]].sz;}
181 }treap;

```

```

176 struct Segment_Tree
177 {
178     #define ls (id<<1)
179     #define rs (id<<1|1)
180     int n,ql,qr,qop;
181     type qv;
182     void update(int l,int r,int id)
183     {
184         if(qop==1) treap[id].insert(qv);
185         else treap[id].erase(qv);
186         if(l>=ql&&r<=qr) return;
187         int mid=(l+r)>>1;
188         if(ql<=mid) update(l,mid,ls);
189         if(qr>mid) update(mid+1,r,rs);
190     }
191     vector<int> treap_id;
192     void dfs(int l,int r,int id)
193     {
194         if(l>=ql&&r<=qr)
195         {
196             treap_id.push_back(id);
197             return;
198         }
199         int mid=(l+r)>>1;
200         if(ql<=mid) dfs(l,mid,ls);
201         if(qr>mid) dfs(mid+1,r,rs);
202     }
203     void get_treap_id(int l,int r)
204     {
205         ql=l;
206         qr=r;
207         treap_id.clear();
208         dfs(1,n,1);
209     }
210     void build(int _n){n=_n;treap.init(n<<2);}
211     void insert(int pos,type v)
212     {
213         ql=qr=pos;
214         qop=1;
215         qv=v;
216         update(1,n,1);
217     }
218     void erase(int pos,type v)
219     {
220         ql=qr=pos;
221         qop=2;
222         qv=v;
223         update(1,n,1);
224     }
225     int ask_rank(int l,int r,type v)
226     {
227         get_treap_id(l,r);
228         int res=1;
229         for(auto &rt:treap_id) res+=treap[rt].
230             order_of_key(v)-1;
231         return res;
232     }
233     int ask_kth(int l,int r,int k)
234     {
235         get_treap_id(l,r);
236         l=0;
237         r=1e8;
238         while(l<r)
239         {
240             int mid=(l+r)>>1,now=1;
241             for(auto &rt:treap_id) now+=treap[rt].
242                 order_of_key(mid+1)-1;
243             if(now<=k) l=mid+1;
244             else r=mid;
245         }
246         return l;
247     }
248     type find_pre(int l,int r,type v)
249     {
250         get_treap_id(l,r);
251         type res=-inf;
252         for(auto &rt:treap_id) res=max(res,treap[rt].
253             find_pre(v));
254         if(res===-inf) return -2147483647;
255         return res;
256     }
257     type find_nex(int l,int r,type v)
258     {
259         get_treap_id(l,r);
260         type res=inf;
261         for(auto &rt:treap_id) res=min(res,treap[rt].
262             find_nex(v));
263         if(res==inf) return 2147483647;
264         return res;
265     }
266     #undef type
267     #undef ls
268     #undef rs
269 }tr;
270 int a[MAX];
271 int main()
272 {
273     int n,m,i,op,l,r,k;
274     scanf("%d%d",&n,&m);
275     tr.build(n);
276     for(i=1;i<=n;i++)
277     {
278         scanf("%d",&a[i]);
279         tr.insert(i,a[i]);
280     }
281     while(m--)
282     {

```

```

279     scanf("%d%d",&op,&l);
280     if(op==3) scanf("%d",&k);
281     else scanf("%d%d",&r,&k);
282     if(op==1) printf("%d\n",tr.ask_rank(l,r,k));
283     else if(op==2) printf("%d\n",tr.ask_kth(l,r,k));
284     else if(op==3)
285     {
286         tr.erase(l,a[l]);
287         a[l]=k;
288         tr.insert(l,a[l]);
289     }
290     else if(op==4) printf("%d\n",tr.find_pre(l,r,
291         k));
292     else if(op==5) printf("%d\n",tr.find_nex(l,r,
293         k));
294 }
return 0;
}

```

2.10.3 树状数组套 treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int INF=0x3f3f3f3f;
5 const int MAX=1e5+10;
6 struct Treap
7 {
8     #define type int
9     #define inf INF
10    struct node
11    {
12        int ch[2],fix,sz,cnt;
13        type v;
14        node(){
15            node(type x,int _sz)
16            {
17                v=x;
18                fix=rand();
19                sz=cnt=_sz;
20                ch[0]=ch[1]=0;
21            }
22        }t[MAX*20];
23        int tot,root[MAX],rt;
24        void init(int n=1)
25        {
26            for(int i=0;i<=n;i++) root[i]=0;
27            rt=1;
28            srand(time(0));
29            tot=0;
30            t[0].sz=t[0].cnt=0;
31            memset(t[0].ch,0,sizeof t[0].ch);
32        }

```

```

33    void pushup(int id)
34    {
35        t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
36            t[id].cnt;
37    }
38    void rotate(int &id,int k)
39    {
40        int y=t[id].ch[k^1];
41        t[id].ch[k^1]=t[y].ch[k];
42        t[y].ch[k]=id;
43        pushup(id);
44        pushup(y);
45        id=y;
46    }
47    void _insert(int &id,type v,int cnt)
48    {
49        if(!id)
50        {
51            id=++tot;
52            t[id]=node(v,cnt);
53            return;
54        }
55        if(t[id].v==v) t[id].cnt+=cnt;
56        else
57        {
58            int tmp=(v>t[id].v);
59            _insert(t[id].ch[tmp],v,cnt);
60            if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
61                (id,tmp^1);
62        }
63        pushup(id);
64    }
65    void _erase(int &id,type v,int cnt)
66    {
67        if(!id) return;
68        if(t[id].v==v)
69        {
70            cnt=min(t[id].cnt,cnt);
71            if(t[id].cnt>cnt)
72            {
73                t[id].cnt-=cnt;
74                pushup(id);
75                return;
76            }
77            if(!(t[id].ch[0]&& t[id].ch[1]))
78            {
79                id=t[id].ch[0]+t[id].ch[1];
80                return;
81            }
82            else
83            {
84                int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
85                    [1]].fix);
86                rotate(id,tmp);

```

```

84         _erase(t[id].ch[tmp],v,cnt);
85         pushup(id);
86     }
87 }
88 else
89 {
90     _erase(t[id].ch[v>t[id].v],v,cnt);
91     pushup(id);
92 }
93 }
94 int _find(type key,int f)
95 {
96     int id=root[rt],res=0;
97     while(id)
98     {
99         if(t[id].v<key)
100         {
101             res+=t[t[id].ch[0]].sz+t[id].cnt;
102             if(f&&key==t[id].v) res-=t[id].cnt;
103             id=t[id].ch[1];
104         }
105         else id=t[id].ch[0];
106     }
107     return res;
108 }
109 type find_by_order(int k)//k small
110 {
111     int id=root[rt];
112     if(id==0) return 0;
113     while(id)
114     {
115         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
116         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
117             return t[id].v;
118         else
119         {
120             k-=t[t[id].ch[0]].sz+t[id].cnt;
121             id=t[id].ch[1];
122         }
123     }
124 }
125 int count(type key)
126 {
127     int id=root[rt];
128     while(id)
129     {
130         if(t[id].v<key)
131         {
132             if(key==t[id].v) return t[id].cnt;
133             id=t[id].ch[1];
134         }
135         else id=t[id].ch[0];
136     }
137     return 0;

```

```

137 }
138 type find_pre(type key)
139 {
140     type res=-inf;
141     int id=root[rt];
142     while(id)
143     {
144         if(t[id].v<key)
145         {
146             res=t[id].v;
147             id=t[id].ch[1];
148         }
149         else id=t[id].ch[0];
150     }
151     return res;
152 }
153 type find_nex(type key)
154 {
155     type res=inf;
156     int id=root[rt];
157     while(id)
158     {
159         if(t[id].v>key)
160         {
161             res=t[id].v;
162             id=t[id].ch[0];
163         }
164         else id=t[id].ch[1];
165     }
166     return res;
167 }
168 Treap &operator[](const int _rt){this->rt=_rt;
169     return *this;}
170 void insert(type v,int sz=1){_insert(root[rt],v,
171     sz);}
172 void erase(type v,int sz=1){_erase(root[rt],v,sz
173     );}
174 int upper_bound_count(type key){return _find(key
175     ,0);};//the count <=key
176 int lower_bound_count(type key){return _find(key
177     ,1);};//the count <key
178 int order_of_key(type key){return
179     lower_bound_count(key)+1;}
180 int size(){return t[root[rt]].sz;}
181 }treap;
182 struct Fenwick_Tree
183 {
184     int n;
185     void init(int _n)
186     {
187         n=_n;
188         treap.init(n);
189     }
190     int lowbit(int x){return x&(-x);}

```

```

185 type get(int x,type v)
186 {
187     type res=0;
188     while(x)
189     {
190         res+=treap[x].lower_bound_count(v);
191         x-=lowbit(x);
192     }
193     return res;
194 }
195 void insert(int x,type v)
196 {
197     while(x<=n)
198     {
199         treap[x].insert(v);
200         x+=lowbit(x);
201     }
202 }
203 void erase(int x,type v)
204 {
205     while(x<=n)
206     {
207         treap[x].erase(v);
208         x+=lowbit(x);
209     }
210 }
211 int ask_kth(int ql,int qr,int k)
212 {
213     int l,r,mid;
214     l=0;
215     r=1e9;
216     while(l<r)
217     {
218         mid=(l+r)>>1;
219         if(get(qr,mid+1)-get(ql-1,mid+1)+1<=k) l=
                mid+1;
220         else r=mid;
221     }
222     return l;
223 }
224 #undef type
225 }tr;
226 int a[MAX];
227 int main()
228 {
229     int n,m,i,l,r,k;
230     char op[3];
231     scanf("%d%d",&n,&m);
232     tr.init(n);
233     for(i=1;i<=n;i++)
234     {
235         scanf("%d",&a[i]);
236         tr.insert(i,a[i]);
237     }

```

```

238 while(m--)
239 {
240     scanf("%s%d",op,&l);
241     if(op[0]=='Q')
242     {
243         scanf("%d%d",&r,&k);
244         printf("%d\n",tr.ask_kth(l,r,k));
245     }
246     else
247     {
248         scanf("%d",&k);
249         tr.erase(l,a[l]);
250         a[l]=k;
251         tr.insert(l,a[l]);
252     }
253 }
254 return 0;
255 }

```

2.11 李超树

```

1 struct LiChao_Segment_Tree
2 {
3     #define type ll
4     static const type inf=LLINF;
5     #define ls (id<<1)
6     #define rs (id<<1|1)
7     static const int MIN_TAG=0;
8     static const int MAX_TAG=1;
9     int TAG;
10    #define init_val (TAG==MIN_TAG?inf:-inf)
11    struct line{type k,b;}sg[MAX<<2];
12    type v[MAX<<2];
13    bool ext[MAX<<2];
14    int ql,qr,n;
15    type cal(const line &l,const int &x){return l.k*
            x+l.b;}
16    int sgn(const type &x){return x==0?0:(x>0?1:-1)};
17    int cmp_min(const type &x,const type &y){return
            sgn(y-x);}
18    int cmp_max(const type &x,const type &y){return
            sgn(x-y);}
19    int cmp(const type &x,const type &y){return TAG
            ==MIN_TAG?cmp_min(x,y):cmp_max(x,y);}
20    type ckres(const type &x,const type &y){return
            TAG==MIN_TAG?min(x,y):max(x,y);}
21    void build(int l,int r,int id)
22    {
23        ext[id]=0;
24        v[id]=init_val;
25        sg[id]={0,init_val};
26        if(l==r) return;

```



```

27     int mid=(l+r)>>1;
28     build(l,mid,ls);
29     build(mid+1,r,rs);
30 }
31 void pushup(int id)
32 {
33     v[id]=ckres(v[id],v[ls]);
34     v[id]=ckres(v[id],v[rs]);
35 }
36 void pushdown(int l,int r,int id,line qv)
37 {
38     int cl,cr,mid;
39     if(!ext[id])
40     {
41         ext[id]=1;
42         sg[id]=qv;
43         goto pushdown_end;
44     }
45     mid=(l+r)>>1;
46     if(cmp(cal(qv,mid),cal(sg[id],mid))==1) swap(
47         qv,sg[id]);
48     cl=cmp(cal(qv,l),cal(sg[id],l));
49     cr=cmp(cal(qv,r),cal(sg[id],r));
50     if(cl>=0&&cr>=0)
51     {
52         sg[id]=qv;
53         goto pushdown_end;
54     }
55     if(cl==1) pushdown(l,mid,ls,qv);
56     else if(cr==1) pushdown(mid+1,r,rs,qv);
57 pushdown_end:
58     v[id]=ckres(cal(sg[id],l),cal(sg[id],r));
59     if(l!=r) pushup(id);
60 }
61 void update(int l,int r,int id,line qv)
62 {
63     if(l>=ql&&r<=qr)
64     {
65         pushdown(l,r,id,qv);
66         return;
67     }
68     int mid=(l+r)>>1;
69     if(ql<=mid) update(l,mid,ls,qv);
70     if(qr>mid) update(mid+1,r,rs,qv);
71     v[id]=ckres(cal(sg[id],l),cal(sg[id],r));
72     pushup(id);
73 }
74 type res;
75 void query(int l,int r,int id)
76 {
77     if(l>=ql&&r<=qr)
78     {
79         res=ckres(v[id],res);
80         return;

```

```

80     }
81     res=ckres(res,cal(sg[id],max(l,ql)));
82     res=ckres(res,cal(sg[id],min(r,qr)));
83     int mid=(l+r)>>1;
84     if(ql<=mid) query(l,mid,ls);
85     if(qr>mid) query(mid+1,r,rs);
86 }
87 void build(int _n,int _TAG){n=_n;TAG=_TAG;build
88     (1,n,1);}
89 void upd(int l,int r,type k,type b)
90 {
91     ql=l;
92     qr=r;
93     update(1,n,1,{k,b});
94 }
95 type ask(int l,int r)
96 {
97     ql=l;
98     qr=r;
99     res=init_val;
100     query(1,n,1);
101     return res;
102 }
103 void dfs(int l,int r,int id)
104 {
105     cout<<l<<" "<<r<<" "<<sg[id].k<<" "<<sg[id].b
106         <<endl;
107     if(l==r) return;
108     int mid=(l+r)>>1;
109     dfs(l,mid,ls);
110     dfs(mid+1,r,rs);
111 }
112 #undef type
113 #undef init_val
114 #undef ls
115 #undef rs
116 }tr;
117 /*
118 upd:O(log^2), ask:O(log)
119 tr.build(n,LiChao_Segment_Tree::MIN_TAG);
120 tr.build(n,LiChao_Segment_Tree::MAX_TAG);
121 tr.upd(1,r,k,b);
122 tr.ask(1,r);
123 */

```

2.12 kd-tree

```

1 namespace kd_tree
2 {
3     const double alpha=0.75;
4     const int dim=2;
5     #define type int

```

```

6  const type NONE=INF; //初始值
7  struct kdtnode
8  {
9      bool exist;
10     int l,r,sz,fa,dep,x[dim],mx[dim],mn[dim];
11     type v,tag;
12     kdtnode(){
13         void initval()
14         {
15             sz=exist;tag=v;
16             if(exist) for(int i=0;i<dim;i++) mn[i]=mx
17                 [i]=x[i];
18         }
19         void null()
20         {
21             exist=sz=0;
22             v=tag=NONE;
23             for(int i=0;i<dim;i++)
24             {
25                 mx[i]=-INF;
26                 mn[i]=INF;
27             }
28         }
29         void newnode(int x0,int x1,type val=NONE)
30         {
31             x[0]=x0;
32             x[1]=x1;
33             l=r=fa=0;
34             exist=1;
35             v=val;
36             initval();
37         }
38         kdtnode(int a,int b,type d=NONE){newnode(a,b,
39             d);}
40     };
41     struct KDT
42     {
43         #define ls t[id].l
44         #define rs t[id].r
45         kdtnode t[MAX];
46         int tot,idx,root;
47         inline void pushup(int id)
48         {
49             t[id].initval();
50             t[id].sz+=t[ls].sz+t[rs].sz;
51             t[id].tag=min({t[ls].tag,t[rs].tag,t[id].
52                 tag});
53             for(int i=0;i<dim;i++)
54             {

```

```

55                 t[id].mn[i]=min(t[id].mn[i],t[ls].
56                     mn[i]);
57             }
58             if(rs)
59             {
60                 t[id].mx[i]=max(t[id].mx[i],t[rs].
61                     mx[i]);
62                 t[id].mn[i]=min(t[id].mn[i],t[rs].
63                     mn[i]);
64             }
65         }
66     }
67     bool isbad(int id){return t[id].sz*alpha+3<
68         max(t[ls].sz,t[rs].sz);}
69     int st[MAX],top;
70     void build(int &id,int l,int r,int fa,int dep
71         =0)
72     {
73         id=0;if(l>r) return;
74         int m=(l+r)>>1; idx=dep;
75         nth_element(st+l,st+m,st+r+1,[&](int x,
76             int y){return t[x].x[idx]<t[y].x[idx
77                 ];});
78         id=st[m];
79         build(ls,l,m-1,id,(dep+1)%dim);
80         build(rs,m+1,r,id,(dep+1)%dim);
81         pushup(id);
82         t[id].dep=dep;
83         t[id].fa=fa;
84     }
85     inline void init(int n=0)
86     {
87         root=0;
88         t[0].null();
89         for(int i=1;i<=n;i++) st[i]=i;
90         if(n) build(root,1,n,0);
91         tot=n;
92     }
93     void travel(int id)
94     {
95         if(!id) return;
96         if(t[id].exist) st[++top]=id;
97         travel(ls);
98         travel(rs);
99     }
100     void rebuild(int &id,int dep)
101     {

```

```

102         id=now;
103         t[id].dep=dep;
104         t[id].fa=fa;
105         return;
106     }
107     if(t[now].x[dep]<t[id].x[dep]) insert(ls,
108         now,id,(dep+1)%dim);
109     else insert(rs,now,id,(dep+1)%dim);
110     pushup(id);
111     if(isbad(id)) rebuild(id,t[id].dep);
112     t[id].dep=dep;
113     t[id].fa=fa;
114 }
115 inline void insert(kdtnode x){t[++tot]=x;
116     insert(root,tot,0,0);}
117 inline void del(int id)
118 {
119     if(!id) return;
120     t[id].null();
121     int x=id;
122     while(x)
123     {
124         pushup(x);
125         x=t[x].fa;
126     }
127     if(isbad(id))
128     {
129         x=t[id].fa;
130         rebuild(root==id?root:(t[x].l==id?t[x]
131             ].l:t[x].r),t[id].dep);
132     }
133 }
134 kdtnode q;
135 ll dist(ll x,ll y){return x*x+y*y;}
136 ll getdist(int id)//点离区域qt[id]最短距离
137 {
138     if(!id) return LLINF;
139     ll res=0;
140     if(q.x[0]<t[id].mn[0]) res+=dist(q.x[0]-t
141         [id].mn[0],0);
142     if(q.x[1]<t[id].mn[1]) res+=dist(q.x[1]-t
143         [id].mn[1],0);
144     if(q.x[0]>t[id].mx[0]) res+=dist(q.x[0]-t
145         [id].mx[0],0);
146     if(q.x[1]>t[id].mx[1]) res+=dist(q.x[1]-t
147         [id].mx[1],0);
148     return res;
149 }
150 kdtnode a,b;
151 inline int check(kdtnode &x)//在矩形x(a,b)内
152 {
153     int ok=1;
154     for(int i=0;i<dim;i++)
155     {

```

```

149         ok&=(x.x[i]>=a.x[i]);
150         ok&=(x.x[i]<=b.x[i]);
151     }
152     return ok;
153 }
154 inline int allin(kdtnode &x)//的子树全在矩
155     形x(a,b)内
156 {
157     int ok=1;
158     for(int i=0;i<dim;i++)
159     {
160         ok&=(x.mn[i]>=a.x[i]);
161         ok&=(x.mx[i]<=b.x[i]);
162     }
163     return ok;
164 }
165 inline int allout(kdtnode &x)//的子树全不在矩
166     形x(a,b)内
167 {
168     int ok=0;
169     for(int i=0;i<dim;i++)
170     {
171         ok|=(x.mx[i]<a.x[i]);
172         ok|=(x.mn[i]>b.x[i]);
173     }
174     return ok;
175 }
176 type res;
177 void query(int id)
178 {
179     if(!id) return;
180     if(allout(t[id])||t[id].sz==0) return;
181     if(allin(t[id]))
182     {
183         res=min(res,t[id].tag);
184         return;
185     }
186     if(check(t[id])&&t[id].exist) res=min(res
187         ,t[id].v);
188     int l,r;
189     l=ls;
190     r=rs;
191     if(t[l].tag>t[r].tag) swap(l,r);
192     if(t[l].tag<res) query(l);
193     if(t[r].tag<res) query(r);
194 }
195 inline type query(kdtnode _a,kdtnode _b)
196 {
197     a=_a;b=_b;
198     res=INF;
199     query(root);
200     return res;
201 }
202 }kd;
203 #undef type

```

```

201     #undef ls
202     #undef rs
203 }
204 using namespace kd_tree;

```

2.13 可并堆

2.13.1 pbds 可并堆

```

1 #include <ext/pb_ds/priority_queue.hpp>
2 using namespace __gnu_pbds;
3 __gnu_pbds::priority_queue<int, greater<int>,
4     pairing_heap_tag> q[MAX];
5 //q[i].join(q[j]) -> q[j]c^q[i],q[j]

```

2.13.2 左偏树 (支持打 tag, 不支持删除)

```

1 struct Leftist_Tree
2 {
3     #define type ll
4     struct node
5     {
6         int ls,rs,d,id;
7         type v;
8         void init(){ls=rs=d=id=v=0;}
9     }t[MAX];
10    int root[MAX],rt,tot,sz[MAX];
11    type tagadd[MAX],tagmul[MAX];
12    leftist_tree &operator[](const int _rt){this->rt
13        =_rt;return *this;}
14    void init(int n=1)
15    {
16        rt=1;
17        tot=0;
18        t[0].init();
19        sz[0]=0;
20        for(int i=0;i<=n;i++) root[i]=0;
21    }
22    int newNode(type v,int p)
23    {
24        int id=++tot;
25        t[id].init();
26        t[id].id=p;
27        t[id].v=v;
28        sz[id]=1;
29        tagadd[id]=0;
30        tagmul[id]=1;
31        return id;
32    }
33    void pushdown(int id)
34    {
35        int ls=t[id].ls;
36        int rs=t[id].rs;

```

```

36    if(tagmul[id]!=1)
37    {
38        if(ls)
39        {
40            t[ls].v*=tagmul[id];
41            tagmul[ls]*=tagmul[id];
42            tagadd[ls]*=tagmul[id];
43        }
44        if(rs)
45        {
46            t[rs].v*=tagmul[id];
47            tagmul[rs]*=tagmul[id];
48            tagadd[rs]*=tagmul[id];
49        }
50        tagmul[id]=1;
51    }
52    if(tagadd[id])
53    {
54        if(ls)
55        {
56            t[ls].v+=tagadd[id];
57            tagadd[ls]+=tagadd[id];
58        }
59        if(rs)
60        {
61            t[rs].v+=tagadd[id];
62            tagadd[rs]+=tagadd[id];
63        }
64        tagadd[id]=0;
65    }
66 }
67 int merge(int x,int y)
68 {
69     if(x==y) return x;
70     if(!x||!y) return x+y;
71     if(t[x].v>t[y].v) swap(x,y); // > small, <
72     pushdown(x);
73     t[x].rs=merge(t[x].rs,y);
74     if(t[t[x].rs].d>t[t[x].ls].d) swap(t[x].ls,t[
75         x].rs);
76     t[x].d=t[t[x].rs].d+1;
77     sz[x]=sz[t[x].ls]+sz[t[x].rs]+1;
78     return x;
79 }
80 void clear(){root[rt]=0;}
81 void join(int y)
82 {
83     if(rt==y) return;
84     root[rt]=merge(root[rt],root[y]);
85     root[y]=0;
86 }
87 void push(type v,int p=0){root[rt]=merge(root[rt
88     ],newNode(v,p));}

```

```

87 void pop()
88 {
89     int x=root[rt];
90     pushdown(x);
91     root[rt]=merge(t[x].ls,t[x].rs);
92 }
93 node top(){return t[root[rt]];}
94 bool empty(){return !root[rt];}
95 int size(){return sz[root[rt]];}
96 void upd(int op,type v)
97 {
98     int x=root[rt];
99     if(op==0)
100     {
101         tagadd[x]+=v;
102         t[x].v+=v;
103     }
104     else
105     {
106         tagadd[x]*=v;
107         tagmul[x]*=v;
108         t[x].v*=v;
109     }
110 }
111 #undef type
112 }q;
113 /*
114 q.init();
115 q.init(n);
116 q[x].push(x);
117 q[x].push(x,id);
118 q[x].top();
119 q[x].pop();
120 q[x].join(y);
121 q[x].clear();
122 */

```

2.13.3 左偏树 (支持删除, 不支持打 tag)

```

1 struct leftist_tree
2 {
3     #define type int
4     struct node
5     {
6         int ls,rs,d,fa,id;
7         type v;
8         void init(){ls=rs=d=fa=id=v=0;}
9     }t[MAX];
10    int root[MAX],rt,tot,pos[MAX];
11    int st[MAX],sttop;
12    leftist_tree &operator[](const int _rt){this->rt
        =_rt;return *this;}
13    void init(int n=1)
14    {

```

```

15        rt=1;
16        tot=0;
17        t[0].init();
18        for(int i=0;i<=n;i++) root[i]=0;
19    }
20    int newnode(type v,int p)
21    {
22        int id=sttop?st[--sttop]:++tot;
23        t[id].init();
24        t[id].v=v;
25        t[id].id=p;
26        pos[p]=id;
27        return id;
28    }
29    void delnode(int id){st[sttop++]=id;}
30    int merge(int x,int y)
31    {
32        if(x==y) return x;
33        if(!x||!y) return x+y;
34        if(t[x].v<t[y].v) swap(x,y); // > small, <
            big
35        t[x].rs=merge(t[x].rs,y);
36        t[t[x].rs].fa=x;
37        if(t[t[x].rs].d>t[t[x].ls].d) swap(t[x].ls,t[
            x].rs);
38        t[x].d=t[t[x].rs].d+1;
39        return x;
40    }
41    void erase_by_id(int id)
42    {
43        int x,y;
44        x=pos[id];
45        if(root[rt]==x)
46        {
47            pop();
48            return;
49        }
50        delnode(x);
51        pos[id]=0;
52        y=merge(t[x].ls,t[x].rs);
53        t[y].fa=t[x].fa;
54        if(t[t[x].fa].ls==x) t[t[x].fa].ls=y;
55        else if(t[t[x].fa].rs==x) t[t[x].fa].rs=y;
56        x=t[y].fa;
57        while(x)
58        {
59            if(t[x].d==t[t[x].rs].d+1) break;
60            t[x].d=t[t[x].rs].d+1;
61            x=t[x].fa;
62        }
63    }
64    void del(int x)
65    {
66        if(!x) return;

```

```

67     del(t[x].ls);
68     del(t[x].rs);
69     delnode(x);
70 }
71 void clear(){del(root[rt]);root[rt]=0;}
72 void join(int y)
73 {
74     if(rt==y) return;
75     root[rt]=merge(root[rt],root[y]);
76     root[y]=0;
77 }
78 void push(type v,int p=0){root[rt]=merge(root[rt]
    ],newnode(v,p));}
79 void pop()
80 {
81     int x=root[rt];
82     pos[t[x].id]=0;
83     root[rt]=merge(t[x].ls,t[x].rs);
84     delnode(x);
85 }
86 node top(){return t[root[rt]];}
87 bool empty(){return !root[rt];}
88 void dfs(int x,type v)
89 {
90     if(!x) return;
91     t[x].v+=v;
92     dfs(t[x].ls,v);
93     dfs(t[x].rs,v);
94 }
95 void updall(type v){dfs(root[rt],v);}
96 node get_by_id(int id){return t[pos[id]];}
97 #undef type
98 }q;
99 /*
100 q.init();
101 q.init(n);
102 q[x].push(x);
103 q[x].push(x,id);
104 q[x].top();
105 q[x].pop();
106 q[x].join(y);
107 q[x].clear();
108 q[x].erase_by_id(id);
109 */

```

2.14 k 叉哈夫曼树

用两个队列代替优先队列复杂度 $O(n)$

注意：小的先进原数组有序

```

1 struct k_Huffman
2 {

```

```

3 #define type ll
4 type work(int n,int k,type *a)// a[1..n], sorted
5 {
6     int i;
7     type res,s;
8     queue<type> q,d;
9     s=((n-1)%(k-1)?k-1-(n-1)%(k-1):0);//
10     while(s-->0) q.push(0);
11     for(i=1;i<=n;i++) q.push(a[i]);
12     res=0;
13     while(q.size()+d.size()>1)
14     {
15         s=0;
16         for(i=0;i<k;i++)
17         {
18             if(q.size()&&d.size())
19             {
20                 if(q.front()<=d.front())
21                 {
22                     s+=q.front();
23                     q.pop();
24                 }
25                 else
26                 {
27                     s+=d.front();
28                     d.pop();
29                 }
30             }
31             else if(q.size())
32             {
33                 s+=q.front();
34                 q.pop();
35             }
36             else if(d.size())
37             {
38                 s+=d.front();
39                 d.pop();
40             }
41         }
42         res+=s;
43         d.push(s);
44     }
45     return res;
46 }
47 #undef type
48 }hfm;

```

2.15 笛卡尔树

$O(n)$ 构造笛卡尔树返回根

性质:

1. 树中的元素满足二叉搜索树性质，要求按照中序遍历得到的序列为原数组序列
2. 树中节点满足堆性质，节点的 key 值要大于其左右子节点的 key 值

```

1 struct Cartesian_Tree
2 {
3     int l[MAX],r[MAX],vis[MAX],stk[MAX];
4     int build(int *a,int n)
5     {
6         int i,top=0;
7         for(i=1;i<=n;i++) l[i]=r[i]=vis[i]=0;
8         for(i=1;i<=n;i++)
9         {
10             int k=top;
11             while(k>0&&a[stk[k-1]]>a[i]) k--;
12             if(k) r[stk[k-1]]=i;
13             if(k<top) l[i]=stk[k];
14             stk[k++] = i;
15             top=k;
16         }
17         for(i=1;i<=n;i++) vis[l[i]]=vis[r[i]]=1;
18         for(i=1;i<=n;i++)
19         {
20             if(!vis[i]) return i;
21         }
22     }
23 }ct;

```

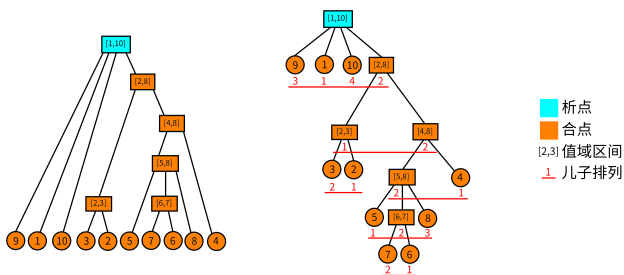
2.16 析合树

id[x]: 序列中第 x 个数在析合树上的编号。

l[x],r[x]: 节点 x 的作用域。

type[x]: 节点 x 的类型，0 表示析点，1 表示合点，默认叶子节点为析点。

注意：若一个节点为合点，这个节点的儿子序列有序。析合树举例：



```

1 struct Permutation_Tree
2 {
3     struct RMQ
4     {

```

```

5     #define type int
6     int lg[MAX],bin[22];
7     type mx[MAX][22],mn[MAX][22];
8     void work(int n,type *v)
9     {
10         int i,j;
11         for(i=bin[0]=1;1<=(i-1)<=n;i++) bin[i]=(
12             bin[i-1]<<1);
13         for(i=2,lg[1]=0;i<=n;i++) lg[i]=lg[i
14             >>1]+1;
15         for(i=1;i<=n;i++) mx[i][0]=mn[i][0]=v[i];
16         for(j=1;1<=(j-1)<=n;j++)
17         {
18             for(i=1;i+bin[j]-1<=n;i++)
19             {
20                 mx[i][j]=max(mx[i][j-1],mx[i+bin[j
21                     -1]][j-1]);
22                 mn[i][j]=min(mn[i][j-1],mn[i+bin[j
23                     -1]][j-1]);
24             }
25         }
26         type ask_max(int l,int r)
27         {
28             int t=lg[r-l+1];
29             return max(mx[l][t],mx[r-bin[t]+1][t]);
30         }
31         type ask_min(int l,int r)
32         {
33             int t=lg[r-l+1];
34             return min(mn[l][t],mn[r-bin[t]+1][t]);
35         }
36         #undef type
37     }rmq;
38     struct Segment_Tree
39     {
40         #define type int
41         #define ls (id<<1)
42         #define rs (id<<1|1)
43         int n,ql,qr;
44         type mn[MAX<<2],tag[MAX<<2],qv;
45         void mdf(int id,type v){mn[id]+=v,tag[id]+=v
46             ;}
47         void pushup(int id){mn[id]=min(mn[ls],mn[rs])
48             ;}
49         void pushdown(int id)
50         {
51             if(!tag[id]) return;
52             mdf(ls,tag[id]);
53             mdf(rs,tag[id]);
54             tag[id]=0;
55         }
56         void build(int l,int r,int id)
57         {

```

```

53     tag[id]=mn[id]=0;
54     if(l==r) return;
55     int mid=(l+r)>>1;
56     build(l,mid,ls);
57     build(mid+1,r,rs);
58     pushup(id);
59 }
60 void update(int l,int r,int id)
61 {
62     if(l>=ql&&r<=qr){mdf(id,qv);return;}
63     pushdown(id);
64     int mid=(l+r)>>1;
65     if(ql<=mid) update(l,mid,ls);
66     if(qr>mid) update(mid+1,r,rs);
67     pushup(id);
68 }
69 int query(int l,int r,int id)
70 {
71     if(l==r) return l;
72     pushdown(id);
73     int mid=(l+r)>>1;
74     if(!mn[ls]) return query(l,mid,ls);
75     else query(mid+1,r,rs);
76 }
77 void build(int _n){n=_n;build(1,n,1);}
78 void upd(int l,int r,type v){ql=l;qr=r;qv=v;
79     update(1,n,1);}
80 type ask(int l,int r){ql=l;qr=r;return query
81     (1,n,1);}
82 #undef type
83 #undef ls
84 #undef rs
85 }tr;
86 bool check(int l,int r){return rmq.ask_max(l,r)-
87     rmq.ask_min(l,r)==r-l;}
88 int st[MAX],st1[MAX],st2[MAX],top,top1,top2,m[
89     MAX];
90 int tot,id[MAX],l[MAX],r[MAX],type[MAX];
91 VI mp[MAX];
92 void add_edge(int a,int b){mp[a].pb(b);}
93 int build(int n,int *a)
94 {
95     int now,i,tmp;
96     tr.build(n);
97     rmq.work(n,a);
98     for(i=0;i<=2*n;i++)
99     {
100         mp[i].clear();
101         type[i]=0;
102     }
103     top=top1=top2=0;
104     tot=0;
105     for(i=1;i<=n;i++)
106     {

```

```

103     while(top1&&a[i]<=a[st1[top1]])
104     {
105         tr.upd(st1[top1-1]+1,st1[top1],a[st1[
106             top1]]);
107         top1--;
108     }
109     while(top2&&a[i]>=a[st2[top2]])
110     {
111         tr.upd(st2[top2-1]+1,st2[top2],-a[st2[
112             top2]]);
113         top2--;
114     }
115     tr.upd(st1[top1]+1,i,-a[i]);
116     st1[++top1]=i;
117     tr.upd(st2[top2]+1,i,a[i]);
118     st2[++top2]=i;
119     id[i]=++tot;
120     l[tot]=r[tot]=i;
121     tmp=tr.ask(1,n);
122     now=tot;
123     while(top&&l[st[top]]>=tmp)
124     {
125         if(type[st[top]]&&check(m[st[top]],i))
126         {
127             r[st[top]]=i;
128             add_edge(st[top],now);
129             now=st[top--];
130         }
131         else if(check(l[st[top]],i))
132         {
133             type[++tot]=1;
134             l[tot]=l[st[top]];
135             r[tot]=i;
136             m[tot]=l[now];
137             add_edge(tot,st[top--]);
138             add_edge(tot,now);
139             now=tot;
140         }
141     }
142     else
143     {
144         add_edge(++tot,now);
145         do
146         {
147             add_edge(tot,st[top--]);
148             }while(top&&!check(l[st[top]],i));
149         l[tot]=l[st[top]];
150         r[tot]=i;
151         add_edge(tot,st[top--]);
152         now=tot;
153     }
154 }
155 st[++top]=now;
156 tr.upd(1,i,-1);
157 }

```



```

155     return st[1];
156 }
157 void work(int n,int *a)// a[1..n]
158 {
159     int rt=build(n,a);
160
161 }
162 }pt;// MAX must *2

```

2.17 莫队算法

```

1 struct MO_Algorithm
2 {
3     #define type int
4     struct query_info{int l,r,id;type v;};
5     vector<query_info> qst;
6     int n,q,a[MAX];
7     type ans[MAX],res;
8     void init(int _n,int _q)
9     {
10         qst.clear();
11         n=_n;
12         q=_q;
13     }
14     void add_qst(int l,int r,int id,type v=0){qst.
15         push_back({l,r,id,v});}
16     void add(int x)
17     {
18     }
19     void del(int x)
20     {
21     }
22 }
23 void work()
24 {
25     int i,l,r,sq;
26     sq=sqrt(q);
27     sort(qst.begin(),qst.end(),[&](query_info a,
28         query_info b){
29         if(a.l/sq!=b.l/sq) return a.l/sq<b.l/sq;
30         if((a.l/sq)&1) return a.r>b.r;
31         else return a.r<b.r;
32     });
33     for(i=1;i<=q;i++) ans[i]=0;
34     l=1;
35     r=0;
36     res=0;
37     for(auto &q:qst)
38     {
39         while(l<q.l) del(l++);
40         while(l>q.l) add(--l);

```

```

41         while(r<q.r) add(++r);
42         while(r>q.r) del(r--);
43         ans[q.id]=res;
44     }
45 }
46 #undef type
47 }mo;
48 /*
49 O(n*sqrt(q))
50 mo.init(n,q);
51 mo.add_qst(l,r,id);
52 mo.work();
53 */

```

2.18 ODT

```

1 struct ODT
2 {
3     #define type int
4     #define init_val 0
5     struct ODT_node
6     {
7         int l,r;
8         mutable type v;
9         inline bool operator<(const ODT_node &o)const
10         {return l<o.l;}
11 };
12 int n;
13 set<ODT_node> odt;
14 typedef set<ODT_node>::iterator odt_iter;
15 void init(int _n)
16 {
17     n=_n;
18     odt.clear();
19     odt.insert({1,n,init_val});
20 }
21 odt_iter find(int x){return --odt.upper_bound({x
22     ,0,init_val});}
23 odt_iter split(int x)
24 {
25     if(x>n) return odt.end();
26     odt_iter it=find(x);
27     if(it->l==x) return it;
28     int l=it->l,r=it->r;
29     type v=it->v;
30     odt.erase(it);
31     odt.insert({l,x-1,v});
32     return odt.insert({x,r,v}).first;
33 }
34 void assign(int l,int r,type v)
35 {
36     odt_iter itr=split(r+1),itl=split(l);
37     odt.erase(itl,itr);

```

```

36     odt.insert({l,r,v});
37 }
38 #undef init_val
39 #undef type
40 }odt;

```

2.19 分块

```

1 struct Block
2 {
3     #define type int
4     static const int N=MAX;
5     static const int SZ=sqrt(N);
6     static const int B=N/SZ+10;
7     int n,bl[B],br[B],bid[N];
8     type a[N],sum[B],tag[B];
9     void build(int _n)
10    {
11        int i,j,id;
12        n=_n;
13        id=0;
14        for(i=1;i<=n;i+=SZ)
15        {
16            bl[++id]=i;
17            br[id]=min(n,i+SZ-1);
18            tag[id]=0;
19            for(j=bl[id];j<=br[id];j++)
20            {
21                bid[j]=id;
22            }
23        }
24    }
25    void pushdown(int id)
26    {
27        int i;
28        if(tag[id])
29        {
30
31        }
32    }
33    void upd_point(int x,type v)
34    {
35
36    }
37    void upd_block(int id,type v)
38    {
39
40    }
41    void upd(int l,int r,type v)
42    {
43        int i;
44        if(bid[l]==bid[r])
45

```

```

46    {
47        pushdown(bid[l]);
48        for(i=l;i<=r;i++) upd_point(i,v);
49        return;
50    }
51    pushdown(bid[l]);
52    pushdown(bid[r]);
53    for(i=l;i<=br[bid[l]];i++) upd_point(i,v);
54    for(i=bl[bid[r]];i<=r;i++) upd_point(i,v);
55    for(i=bid[l]+1;i<=bid[r]-1;i++) upd_block(i,v);
56    }
57    type res;
58    void ask_point(int x)
59    {
60
61    }
62    void ask_block(int id)
63    {
64
65    }
66    type ask(int l,int r)
67    {
68        int i;
69        res=0;
70        if(bid[l]==bid[r])
71        {
72            pushdown(bid[l]);
73            for(i=l;i<=r;i++) ask_point(i);
74            return res;
75        }
76        pushdown(bid[l]);
77        pushdown(bid[r]);
78        for(i=l;i<=br[bid[l]];i++) ask_point(i);
79        for(i=bl[bid[r]];i<=r;i++) ask_point(i);
80        for(i=bid[l]+1;i<=bid[r]-1;i++) ask_block(i);
81        return res;
82    }
83    #undef type
84 }blk;

```

3 树

3.1 LCA

3.1.1 倍增 LCA

```

1 struct LCA
2 {
3     static const int N=MAX;
4     static const int LOG=log2(N)+2;
5     int fa[LOG][N],dep[N];
6     vector<int> *mp;

```

```

7 void dfs(int x,int pre)
8 {
9     int i;
10    fa[0][x]=pre;
11    for(i=1;i<LOG;i++) fa[i][x]=fa[i-1][fa[i-1][x]
12        ]];
13    for(auto &to:mp[x])
14    {
15        if(to==pre) continue;
16        dep[to]=dep[x]+1;
17        dfs(to,x);
18    }
19    void work(int root,vector<int> *_mp)
20    {
21        mp=_mp;
22        for(int j=0;j<LOG;j++) fa[j][0]=0;
23        dep[root]=0;
24        dfs(root,0);
25    }
26    int go(int x,int d)
27    {
28        for(int i=0;i<LOG;i++)
29        {
30            if((d>>i)&1) x=fa[i][x];
31        }
32        return x;
33    }
34    int lca(int x,int y)
35    {
36        if(dep[x]<dep[y]) swap(x,y);
37        x=go(x,dep[x]-dep[y]);
38        if(x==y) return x;
39        for(int i=LOG-1;~i;i--)
40        {
41            if(fa[i][x]!=fa[i][y])
42            {
43                x=fa[i][x];
44                y=fa[i][y];
45            }
46        }
47        return fa[0][x];
48    }
49 }lca;
50 /*
51 O(nlogn)-O(logn)
52 lca.work(n,root,mp);
53 */

```

3.1.2 RMQ 维护欧拉序求 LCA

```

1 struct LCA
2 {
3     static const int N=MAX;

```

```

4     static const int LOG2N=log2(2*N)+3;
5     #define type int
6     struct node{int to;type w;};
7     type dis[N];
8     int path[2*N],deep[2*N],first[N],len[N],tot,n;
9     int dp[2*N][LOG2N];
10    vector<node> mp[N];
11    void init(int _n)
12    {
13        n=_n;
14        for(int i=0;i<=n;i++)
15        {
16            dis[i]=len[i]=0;
17            mp[i].clear();
18        }
19    }
20    void add_edge(int a,int b,type w=1){mp[a].
21        push_back({b,w});}
22    void dfs(int x,int pre,int h)
23    {
24        int i;
25        path[++tot]=x;
26        first[x]=tot;
27        deep[tot]=h;
28        for(i=0;i<mp[x].size();i++)
29        {
30            int to=mp[x][i].to;
31            if(to==pre) continue;
32            dis[to]=dis[x]+mp[x][i].w;
33            len[to]=len[x]+1;
34            dfs(to,x,h+1);
35            path[++tot]=x;
36            deep[tot]=h;
37        }
38    }
39    void ST(int n)
40    {
41        int i,j,x,y;
42        for(i=1;i<=n;i++) dp[i][0]=i;
43        for(j=1;(1<<j)<=n;j++)
44        {
45            for(i=1;i+(1<<j)-1<=n;i++)
46            {
47                x=dp[i][j-1];
48                y=dp[i+(1<<(j-1))][j-1];
49                dp[i][j]=deep[x]<deep[y]?x:y;
50            }
51        }
52    }
53    int query(int l,int r)
54    {
55        int len,x,y;
56        len=__lg(r-l+1);
57        x=dp[l][len];
58        y=dp[r-(1<<len)+1][len];
59        return deep[x]<deep[y]?x:y;
60    }

```

```

57     y=dp[r-(1<<len)+1][len];
58     return deep[x]<deep[y]?x:y;
59 }
60 int lca(int x,int y)
61 {
62     int l,r,pos;
63     l=first[x];
64     r=first[y];
65     if(l>r) swap(l,r);
66     pos=query(l,r);
67     return path[pos];
68 }
69 type get_dis(int a,int b){return dis[a]+dis[b
70 ]-2*dis[lca(a,b)];}
71 int get_len(int a,int b){return len[a]+len[b]-2*
72 len[lca(a,b)];}
73 void work(int rt)
74 {
75     tot=0;
76     dfs(rt,0,0);
77     ST(2*n-1);
78 }
79 int lca_root(int rt,int x,int y)
80 {
81     int fx,fy;
82     fx=lca(x,rt);
83     fy=lca(y,rt);
84     if(fx==fy) return lca(x,y);
85     else
86     {
87         if(get_len(fx,rt)<get_len(fy,rt)) return
88         fx;
89         else return fy;
90     }
91 }
92 #undef type
93 }lca;
94 /*
95 O(n*log(n))-O(1)
96 lca.init(n);
97 lca.add_edge(a,b,w);
98 lca.work(rt);
99 */

```

3.2 树链剖分

3.2.1 轻重链剖分

```

1 struct Heavy_Light_Decomposition
2 {
3     #define type int
4     struct edge{int a,b;type v;};
5     vector<int> mp[MAX];

```

```

6     vector<edge> e;
7     int dep[MAX],fa[MAX],sz[MAX],son[MAX];
8     int id[MAX],top[MAX],dfn[MAX],tot;
9     int n,rt;
10    void init(int _n)
11    {
12        n=_n;
13        for(int i=0;i<=n;i++) mp[i].clear();
14        e.clear();
15        e.push_back({0,0,0});
16    }
17    void add_edge(int a,int b,type v=0)
18    {
19        e.push_back({a,b,v});
20        mp[a].push_back(b);
21        mp[b].push_back(a);
22    }
23    void dfs1(int x,int pre,int h)
24    {
25        int i,to;
26        dep[x]=h;
27        fa[x]=pre;
28        sz[x]=1;
29        for(i=0;i<mp[x].size();i++)
30        {
31            to=mp[x][i];
32            if(to==pre) continue;
33            dfs1(to,x,h+1);
34            sz[x]+=sz[to];
35            if(son[x]==-1||sz[to]>sz[son[x]]) son[x]=
36                to;
37        }
38    }
39    void dfs2(int x,int tp)
40    {
41        int i,to;
42        dfn[x]=++tot;
43        id[dfn[x]]=x;
44        top[x]=tp;
45        if(son[x]==-1) return;
46        dfs2(son[x],tp);
47        for(i=0;i<mp[x].size();i++)
48        {
49            to=mp[x][i];
50            if(to!=son[x]&&to!=fa[x]) dfs2(to,to);
51        }
52    }
53    void work(int _rt)
54    {
55        rt=_rt;
56        for(int i=0;i<=n;i++) son[i]=-1;
57        tot=0;
58        dfs1(rt,0,0);
59        dfs2(rt,rt);

```

```

59     }
60     int lca(int x,int y)
61     {
62         while(top[x]!=top[y])
63         {
64             if(dep[top[x]]<dep[top[y]]) swap(x,y);
65             x=fa[top[x]];
66         }
67         if(dep[x]>dep[y]) swap(x,y);
68         return x;
69     }
70     int find_yson_toward_x(int x,int y)
71     {
72         while(top[x]!=top[y])
73         {
74             if(fa[top[x]]==y) return top[x];
75             x=fa[top[x]];
76         }
77         return son[y];
78     }
79     //node
80     void init_node(type *v)
81     {
82         for(int i=1;i<=n;i++) tr.a[dfn[i]]=v[i];
83         tr.build(n);
84     }
85     void upd_node(int x,int y,type v)
86     {
87         while(top[x]!=top[y])
88         {
89             if(dep[top[x]]<dep[top[y]]) swap(x,y);
90             tr.upd(dfn[top[x]],dfn[x],v);
91             x=fa[top[x]];
92         }
93         if(dep[x]>dep[y]) swap(x,y);
94         tr.upd(dfn[x],dfn[y],v);
95     }
96     type ask_node(int x,int y)
97     {
98         type res=0;
99         while(top[x]!=top[y])
100         {
101             if(dep[top[x]]<dep[top[y]]) swap(x,y);
102             res+=tr.ask(dfn[top[x]],dfn[x]);
103             x=fa[top[x]];
104         }
105         if(dep[x]>dep[y]) swap(x,y);
106         res+=tr.ask(dfn[x],dfn[y]);
107         return res;
108     }
109     //path
110     void init_path()
111     {
112         tr.a[dfn[rt]]=0;

```

```

113         for(int i=1;i<n;i++)
114         {
115             if(dep[e[i].a]<dep[e[i].b]) swap(e[i].a,e
116                 [i].b);
117             tr.a[dfn[e[i].a]]=e[i].v;
118         }
119         tr.build(n);
120     }
121     void upd_edge(int id,type v)
122     {
123         if(dep[e[id].a]>dep[e[id].b]) tr.upd(dfn[e[id
124             ].a],dfn[e[id].a],v);
125         else tr.upd(dfn[e[id].b],dfn[e[id].b],v);
126     }
127     void upd_path(int x,int y,type v)
128     {
129         while(top[x]!=top[y])
130         {
131             if(dep[top[x]]<dep[top[y]]) swap(x,y);
132             tr.upd(dfn[top[x]],dfn[x],v);
133             x=fa[top[x]];
134         }
135         if(dep[x]>dep[y]) swap(x,y);
136         if(x!=y) tr.upd(dfn[x]+1,dfn[y],v);
137     }
138     type ask_path(int x,int y)
139     {
140         type res=0;
141         while(top[x]!=top[y])
142         {
143             if(dep[top[x]]<dep[top[y]]) swap(x,y);
144             res+=tr.ask(dfn[top[x]],dfn[x]);
145             x=fa[top[x]];
146         }
147         if(dep[x]>dep[y]) swap(x,y);
148         if(x!=y) res+=tr.ask(dfn[x]+1,dfn[y]);
149         return res;
150     }
151     // sub tree
152     void change_root(int x){rt=x;}
153     void upd_subtree(int x,type v)
154     {
155         if(x==rt) tr.upd(1,n,v);
156         if(dfn[rt]>=dfn[x]&&dfn[rt]<=dfn[x]+sz[x]-1)
157         {
158             x=find_yson_toward_x(rt,x);
159             tr.upd(1,dfn[x]-1,v);
160             tr.upd(dfn[x]+sz[x],n,v);
161         }
162         tr.upd(dfn[x],dfn[x]+sz[x]-1,v);
163     }
164     type ask_subtree(int x)
165     {
166         if(x==rt) return tr.ask(1,n).v;

```

```

165         if(dfn[rt]>=dfn[x]&&dfn[rt]<=dfn[x]+sz[x]-1)
166         {
167             x=find_yson_toward_x(rt,x);
168             return tr.merge(tr.ask(1,dfn[x]-1),
169                             tr.ask(dfn[x]+sz[x],n)).v;
170         }
171         return tr.ask(dfn[x],dfn[x]+sz[x]-1).v;
172     }
173     #undef type
174 }hld;
175 /*
176 hld.init(n)
177 hld.add_edge(a,b,v=0); a <-> b
178 hld.work(rt);
179 hld.lca(a,b);
180 ---- node ----
181 hld.init_node(type *v);
182 hld.upd_node(a,b,v);
183 hld.ask_node(a,b);
184 ---- path ----
185 hld.init_path();
186 hld.upd_edge(id,v); id:1..n-1
187 hld.upd_path(a,b,v);
188 hld.ask_path(a,b);
189 ---- subtree ----
190 hld.change_root(rt);
191 hld.upd_subtree(x,v);
192 hld.ask_subtree(x);
193 */

```

3.3 树的重心

```

1 struct Tree_Centroid
2 {
3     vector<int> *mp;
4     int sz[MAX],mx[MAX],n;
5     void dfs(int x,int fa)
6     {
7         sz[x]=1;
8         mx[x]=0;
9         for(auto &to:mp[x])
10        {
11            if(to==fa) continue;
12            dfs(to,x);
13            sz[x]+=sz[to];
14            mx[x]=max(mx[x],sz[to]);
15        }
16        mx[x]=max(mx[x],n-sz[x]);
17    }
18    vector<int> get_tree_centroid(int _n,vector<int>
19                                *_mp,int root)
20    {
21        int i,mn;

```

```

21        n=_n;
22        mp=_mp;
23        dfs(root,-1);
24        vector<int> res;
25        mn=n+1;
26        for(i=1;i<=n;i++) mn=min(mn,mx[i]);
27        for(i=1;i<=n;i++)
28        {
29            if(mx[i]==mn) res.push_back(i);
30        }
31        return res;
32    }
33 }trct;

```

3.4 树上倍增

```

1 const int LOG=log2(MAX);
2 int fa[MAX][LOG+1];
3 void dfs(int x,int pre)
4 {
5     fa[x][0]=pre;
6     for(int i=1;i<=LOG;i++) fa[x][i]=fa[fa[x][i-1]][
7         i-1];
8     for(auto &to:mp[x])
9     {
10        if(to==pre) continue;
11        dfs(to,x);
12    }
13 }
14 int jump(int x,int target)
15 {
16     for(int i=LOG;~i;i--)
17     {
18        if(fa[x][i]&&fa[x][i]<=target) x=fa[x][i];
19    }
20    return x;
21 }
22 int main()
23 {
24     memset(fa[0],0,sizeof fa[0]);
25     return 0;

```

3.5 树 hash

```

1 struct Tree_Hash
2 {
3     const ull mask=std::chrono::steady_clock::now().
4         time_since_epoch().count();
5     ull shift(ull x)
6     {
7         x^=mask;
8         x^=x<<13;

```

```

8      x^=x>>7;
9      x^=x<<17;
10     x^=mask;
11     return x;
12 }
13 ull hash[MAX];
14 void dfs(VI *mp,int x,int fa)
15 {
16     hash[x]=1;
17     for(auto to:mp[x])
18     {
19         if(to==fa) continue;
20         dfs(mp,to,x);
21         hash[x]+=shift(hash[to]);
22     }
23 }
24 void get_tree_hash(VI *mp,int root)
25 {
26     dfs(mp,root,0);
27 }
28 }trha;

```

```

27     {
28         vtree_mp[now_lca].pb(st[top-1]);
29         st[top-1]=now_lca;
30         tmp.push_back(now_lca);
31     }
32     st[top++]=a[i];
33 }
34 while(top>1)
35 {
36     vtree_mp[st[top-2]].pb(st[top-1]);
37     top--;
38 }
39 for(auto it:tmp) a.push_back(it);
40 return st[0];
41 }
42 void clear_vtree(VI &a,VI vtree_mp[])
43 {
44     for(auto it:a) vtree_mp[it].clear();
45 }
46 }vt; // need lca and dfn

```

3.6 虚树

```

1 struct Virtual_Tree
2 {
3     int st[MAX],top;
4     int build_vtree(VI &a,VI vtree_mp[])// return
       root
5     {
6         int now_lca;
7         sort(all(a),[&](int x,int y){return lca.dfn[x]
           <lca.dfn[y];});
8         a.erase(unique(all(a)),a.end());
9         assert(sz(a)>0);
10        top=0;
11        st[top++]=a[0];
12        VI tmp;
13        for(int i=1;i<sz(a);i++)
14        {
15            if(top==0)
16            {
17                st[top++]=a[i];
18                continue;
19            }
20            now_lca=lca.lca(a[i],st[top-1]);
21            while(top>1&& lca.dfn[st[top-2]]>=lca.dfn[
                now_lca])
22            {
23                vtree_mp[st[top-2]].pb(st[top-1]);
24                top--;
25            }
26            if(now_lca!=st[top-1])

```

3.7 树分治

3.7.1 点分治

```

1 struct tree_divide
2 {
3     #define type int
4     struct edge{int to;type w;};
5     vector<edge> mp[MAX];
6     bool vis[MAX];
7     int sz[MAX],mx[MAX],sum,rt,n;
8     void init(int _n)
9     {
10         int i;
11         n=_n;
12         for(i=0;i<=n;i++) mp[i].clear();
13     }
14     void add_edge(int x,int y,type w){mp[x].
        push_back({y,w});}
15     void get_centroid(int x,int fa)
16     {
17         sz[x]=1;
18         mx[x]=0;
19         for(auto &it:mp[x])
20         {
21             int to=it.to;
22             if(to==fa||vis[to]) continue;
23             get_centroid(to,x);
24             sz[x]+=sz[to];
25             mx[x]=max(mx[x],sz[to]);
26         }
27         mx[x]=max(mx[x],sum-sz[x]);
28         if(mx[x]<mx[rt]) rt=x;

```

```

29     }
30     void get_sz(int x,int fa)
31     {
32         sz[x]=1;
33         for(auto &it:mp[x])
34         {
35             int to=it.to;
36             if(to==fa||vis[to]) continue;
37             get_sz(to,x);
38             sz[x]+=sz[to];
39         }
40     }
41     type d[MAX],del[MAX];
42     int totd,totdel;
43     void get_dis(int x,int fa,type h)
44     {
45         d[++totd]=h;
46         for(auto &it:mp[x])
47         {
48             int to=it.to;
49             if(to==fa||vis[to]) continue;
50             get_dis(to,x,h+it.w);
51         }
52     }
53     void cal(int x)
54     {
55         int i,j;
56         totdel=0;
57         for(auto &it:mp[x])
58         {
59             int to=it.to;
60             if(vis[to]) continue;
61             totd=0;
62             get_dis(to,x,it.w);
63             // do something
64             for(i=1;i<=totd;i++) del[++totdel]=d[i];
65         }
66         for(i=1;i<=totdel;i++)
67         {
68             // del
69         }
70     }
71     void divide(int x)
72     {
73         vis[x]=1;
74         cal(x);
75         for(auto &it:mp[x])
76         {
77             int to=it.to;
78             if(vis[to]) continue;
79             get_sz(to,-1);
80             rt=0;
81             mx[rt]=sum=sz[to];
82             get_centroid(to,-1);

```

```

83             divide(rt);
84         }
85     }
86     void work()
87     {
88         int i;
89         for(i=0;i<=n;i++) vis[i]=0;
90         rt=0;
91         mx[rt]=sum=n;
92         get_centroid(1,-1);
93         // init ans
94         divide(rt);
95     }
96     #undef type
97 }tdv;
98 /*
99 tdv.init(n);
100 tdv.add_edge(x,y,w) // x -> y
101 tdv.work();
102 */

```

3.8 Link-Cut-Tree

```

1 struct Link_Cut_Tree
2 {
3     #define type int
4     const type inf=INF;
5     struct node
6     {
7         int ch[2],fa,sz,rev,tag;
8         type v,sum;
9     }t[MAX];
10     int tot,root,st[MAX];
11     void maintain(int id,type v)
12     {
13         t[id].v=v;
14         t[id].sum=v;
15     }
16     void pushup(int id)
17     {
18         int ls=t[id].ch[0];
19         int rs=t[id].ch[1];
20         t[id].sz=t[ls].sz+t[rs].sz+1;
21         t[id].sum=t[ls].sum+t[rs].sum+t[id].v;
22     }
23     void pushdown(int id)
24     {
25         int ls=t[id].ch[0];
26         int rs=t[id].ch[1];
27         if(t[id].tag)
28         {
29             if(ls)
30                 {

```



```

31         maintain(ls,t[id].v);
32         t[ls].tag=1;
33     }
34     if(rs)
35     {
36         maintain(rs,t[id].v);
37         t[rs].tag=1;
38     }
39     t[id].tag=0;
40 }
41 if(t[id].rev)
42 {
43     t[ls].rev^=1;
44     t[rs].rev^=1;
45     swap(t[ls].ch[0],t[ls].ch[1]);
46     swap(t[rs].ch[0],t[rs].ch[1]);
47     t[id].rev=0;
48 }
49 }
50 int newnode(type v,int fa)
51 {
52     int id=++tot;
53     memset(t[id].ch,0,sizeof t[id].ch);
54     t[id].fa=fa;
55     t[id].sz=1;
56     t[id].tag=t[id].rev=0;
57     maintain(id,v);
58     return id;
59 }
60 int is_splay_root(int x)
61 {
62     int fa=t[x].fa;
63     return t[fa].ch[0]!=x&& t[fa].ch[1]!=x;
64 }
65 void rotate(int x)
66 {
67     int y,z,k;
68     y=t[x].fa;
69     z=t[y].fa;
70     k=(x==t[y].ch[1]);
71     if(!is_splay_root(y)) t[z].ch[y==t[z].ch[1]]=
72         x;
73     t[y].ch[k]=t[x].ch[k^1];
74     if(t[x].ch[k^1]) t[t[x].ch[k^1]].fa=y;
75     t[x].ch[k^1]=y;
76     t[y].fa=x;
77     t[x].fa=z;
78     pushup(y);
79 }
80 void splay(int x)
81 {
82     int y,z,top;
83     top=0;
84     y=x;

```

```

84     st[++top]=y;
85     while(!is_splay_root(y))
86     {
87         y=t[y].fa;
88         st[++top]=y;
89     }
90     while(top>0) pushdown(st[top--]);
91     while(!is_splay_root(x))
92     {
93         y=t[x].fa;
94         z=t[y].fa;
95         if(!is_splay_root(y))
96         {
97             if((t[z].ch[0]==y)^(t[y].ch[0]==x))
98                 rotate(x);
99             else rotate(y);
100         }
101         rotate(x);
102     }
103     pushup(x);
104 }
105 void init_null_node()
106 {
107     memset(t[0].ch,0,sizeof t[0].ch);
108     t[0].sz=t[0].fa=0;
109     t[0].v=t[0].sum=0;
110 }
111 void init(int n,type *v)
112 {
113     int i;
114     tot=0;
115     init_null_node();
116     for(i=1;i<=n;i++) newnode(v[i],0);
117 }
118 void init(int n)
119 {
120     int i;
121     tot=0;
122     init_null_node();
123     for(i=1;i<=n;i++) newnode(0,0);
124 }
125 int access(int x)
126 {
127     int fa=0;
128     while(x)
129     {
130         splay(x);
131         t[x].ch[1]=fa;
132         pushup(x);
133         fa=x;
134         x=t[x].fa;
135     }
136     return fa;

```

```

137 int findroot(int x)
138 {
139     access(x);
140     splay(x);
141     pushdown(x);
142     while(t[x].ch[0])
143     {
144         x=t[x].ch[0];
145         pushdown(x);
146     }
147     splay(x);
148     return x;
149 }
150 void makeroot(int x)
151 {
152     x=access(x);
153     swap(t[x].ch[0],t[x].ch[1]);
154     t[x].rev^=1;
155 }
156 int split(int x,int y)
157 {
158     makeroot(x);
159     access(y);
160     splay(y);
161     return y;
162 }
163 void link(int x,int y)
164 {
165     makeroot(x);
166     splay(x);
167     if(findroot(y)!=x) t[x].fa=y;
168 }
169 void cut(int x,int y)
170 {
171     makeroot(x);
172     if(findroot(y)==x&&t[y].fa==x&&t[y].ch[0]==0)
173     {
174         t[y].fa=t[x].ch[1]=0;
175         pushup(x);
176     }
177 }
178 int is_connect(int x,int y)
179 {
180     makeroot(x);
181     return findroot(y)==x;
182 }
183 void upd_node(int x,type v)
184 {
185     splay(x);
186     maintain(x,v);
187 }
188 #undef type
189 }lct;
190 /*

```

```

191 lct.init(n);
192 lct.init(n,*v); v[1..n]
193 */

```

4 图论

4.1 链式前向星

```

1 struct Fast_Vector
2 {
3     static const int N=;
4     static const int M=;
5     Fast_Vector &operator()(const int _x){this->x=_x
6         ;return *this;}
7     int head[N],tot,x;
8     struct node{int to,nex;} e[M];
9     void clear(int n)
10    {
11        for(int i=0;i<=n;i++) head[i]=0;
12        tot=0;
13    }
14    void push_back(const int &to)
15    {
16        tot++;
17        e[tot].to=to;
18        e[tot].nex=head[x];
19        head[x]=tot;
20    }
21    //for(i=mp.head[x];i;i=mp.e[i].nex) to=mp.e[i].to;

```

4.2 最短路

4.2.1 dijkstra

```

1 struct Dijkstra
2 {
3     #define type int
4     const type inf=INF;
5     struct node
6     {
7         int id;
8         type v;
9         friend bool operator <(node a,node b){return
10             a.v>b.v;}
11 };
12 static const int N=MAX;
13 vector<node> mp[N];
14 type dis[N];
15 int n,vis[N];
16 void init(int _n)
17 {
18     n=_n;

```

```

18     for(int i=0;i<=n;i++) mp[i].clear();
19 }
20 void add_edge(int x,int y,type v){mp[x].
    push_back({y,v});}
21 void work(int s)
22 {
23     int i,to;
24     type w;
25     priority_queue<node> q;
26     for(i=0;i<=n;i++)
27     {
28         dis[i]=inf;
29         vis[i]=0;
30     }
31     dis[s]=0;
32     q.push({s,type(0)});
33     while(!q.empty())
34     {
35         node t=q.top();
36         q.pop();
37         if(vis[t.id]) continue;
38         vis[t.id]=1;// this node has already been
            extended
39         for(auto &it:mp[t.id])
40         {
41             to=it.id;
42             w=it.v;
43             if(dis[to]>dis[t.id]+w)
44             {
45                 dis[to]=dis[t.id]+w;
46                 if(!vis[to]) q.push({to,dis[to]});
47             }
48         }
49     }
50 }
51 #undef type
52 }dij;

```

4.2.2 spfa

```

1 struct SPFA
2 {
3     #define type int
4     static const type inf=INF;
5     static const int N=;
6     vector<pair<int,type> > mp[N];
7     type dis[N];
8     int n,vis[N],cnt[N];
9     void init(int _n)
10    {
11        n=_n;
12        for(int i=0;i<=n;i++) mp[i].clear();
13    }

```

```

14 void add_edge(int x,int y,type v){mp[x].
    push_back({y,v});}
15 bool work(int s)
16 {
17     int i,x,to;
18     type w;
19     queue<int> q;
20     for(i=0;i<=n;i++)
21     {
22         dis[i]=inf;
23         vis[i]=cnt[i]=0;
24     }
25     dis[s]=0;
26     vis[s]=1;
27     q.push(s);
28     while(!q.empty())
29     {
30         x=q.front();
31         q.pop();
32         vis[x]=0;
33         for(auto &it:mp[x])
34         {
35             to=it.first;
36             w=it.second;
37             if(dis[to]>dis[x]+w)
38             {
39                 dis[to]=dis[x]+w;
40                 cnt[to]=cnt[x]+1;
41                 if(cnt[to]>n)
42                 {
43                     // cnt is edge counts of
                        current short path
44                     // if cnt >= (sum of node), the
                        graph exists negative ring
45                     return false;
46                 }
47                 if(!vis[to])
48                 {
49                     q.push(to);
50                     vis[to]=1;
51                 }
52             }
53         }
54     }
55     return true;
56 }
57 #undef type
58 }spfa;

```

4.2.3 floyd 求最小环

```

1 struct Floyd
2 {
3     #define type int

```

```

4  const type inf=INF;
5  static const int N=;
6  int n;
7  type mp[N][N],dis[N][N];
8  type min_circle_3;// len(circle)>=3
9  type min_circle; // len(circle)>=1
10 void init(int _n)
11 {
12     int i,j;
13     n=_n;
14     for(i=1;i<=n;i++)
15     {
16         for(j=1;j<=n;j++)
17         {
18             mp[i][j]=dis[i][j]=inf;
19         }
20     }
21 }
22 void add_edge(int x,int y,type w)
23 {
24     w=min(mp[x][y],w);
25     mp[x][y]=dis[x][y]=w;
26 }
27 void work()
28 {
29     int i,j,k;
30     min_circle_3=inf;
31     for(k=1;k<=n;k++)
32     {
33         for(i=1;i<k;i++)
34         {
35             if(mp[i][k]==inf) continue;
36             for(j=1;j<k;j++)
37             {
38                 if(i==j||mp[k][j]==inf||dis[j][i]==
39                     inf) continue;
40                 min_circle_3=min(min_circle_3,mp[i
41                     ][k]+mp[k][j]+dis[j][i]);
42             }
43         }
44         for(i=1;i<=n;i++)
45         {
46             if(dis[i][k]==inf) continue;
47             for(j=1;j<=n;j++)
48             {
49                 if(dis[k][j]==inf) continue;
50                 dis[i][j]=min(dis[i][j],dis[i][k]+
51                     dis[k][j]);
52             }
53         }
54     }
55     min_circle=inf;
56     for(i=1;i<=n;i++) min_circle=min(min_circle,
57         dis[i][i]);

```

```

54     }
55     #undef type
56 }flyd;
57 /*
58 flyd.init(n);
59 flyd.add_edge(x,y,w); x,y [1..n] x->y
60 flyd.work();
61 */

```

4.2.4 Johnson

```

1 struct Johnson
2 {
3     #define type int
4     #define inf INF
5     #define PTI pair<type,int>
6     static const int N=3000+10;
7     vector<pair<int,type> > mp[N];
8     type dis[N],h[N];
9     int n,vis[N],cnt[N];
10    bool spfa_flag;
11    void init(int _n)
12    {
13        n=_n;
14        spfa_flag=false;
15        for(int i=0;i<=n;i++) mp[i].clear();
16    }
17    void add_edge(int x,int y,type v){mp[x].pb(MP(y,
18        v));}
19    bool spfa(int s)
20    {
21        int i,x,to;
22        type w;
23        queue<int> q;
24        for(i=0;i<=n;i++)
25        {
26            h[i]=inf;
27            vis[i]=cnt[i]=0;
28        }
29        h[s]=0;
30        vis[s]=1;
31        q.push(s);
32        while(!q.empty())
33        {
34            x=q.front();
35            q.pop();
36            vis[x]=0;
37            for(auto it:mp[x])
38            {
39                to=it.fi;
40                w=it.se;
41                if(h[to]>h[x]+w)
42                {

```

```

43         cnt[to]=cnt[x]+1;
44         if(cnt[to]>n)
45         {
46             // cnt is edge counts of
47             // current short path
48             // if cnt >= (sum of node), the
49             // graph exists negative ring
50             return false;
51         }
52         if(!vis[to])
53         {
54             q.push(to);
55             vis[to]=1;
56         }
57     }
58     }
59     spfa_flag=true;
60     return true;
61 }
62 void dij(int s)
63 {
64     assert(spfa_flag);
65     int i,to;
66     type w;
67     priority_queue<PTI ,vector<PTI>,greater<PTI>
68         > q;
69     for(i=0;i<=n;i++)
70     {
71         dis[i]=inf;
72         vis[i]=0;
73     }
74     dis[s]=0;
75     q.push(MP(type(0),s));
76     while(!q.empty())
77     {
78         PTI t=q.top();
79         q.pop();
80         if(vis[t.se]) continue;
81         vis[t.se]=1;
82         for(auto it:mp[t.se])
83         {
84             to=it.fi;
85             w=it.se+h[t.se]-h[to];
86             if(dis[to]>dis[t.se]+w)
87             {
88                 dis[to]=dis[t.se]+w;
89                 if(!vis[to]) q.push(MP(dis[to],to))
90                 ;
91             }
92         }
93     }
94     for(i=0;i<=n;i++)
95     {

```

```

93         if(dis[i]==inf) continue;
94         dis[i]-=h[s]-h[i];
95     }
96 }
97 #undef type
98 #undef inf
99 #undef PTI
100 }js;

```

4.2.5 同余最短路

```

1 struct Dijkstra
2 {
3     #define type ll
4     const type inf=LLINF;
5     struct node
6     {
7         int id;
8         type v;
9         friend bool operator <(node a,node b){return
10             a.v>b.v;}
11 };
12 static const int N=MAX;
13 vector<node> mp[N];
14 type dis[N];
15 int n,vis[N];
16 void init(int _n)
17 {
18     n=_n;
19     for(int i=0;i<=n;i++) mp[i].clear();
20 }
21 void add_edge(int x,int y,type v){ mp[x].
22     push_back({y,v});}
23 void work(int s)
24 {
25     int i,to;
26     type w;
27     priority_queue<node> q;
28     for(i=0;i<=n;i++)
29     {
30         dis[i]=inf;
31         vis[i]=0;
32     }
33     dis[s]=0;
34     q.push({s,type(0)});
35     while(!q.empty())
36     {
37         node t=q.top();
38         q.pop();
39         if(vis[t.id]) continue;
40         vis[t.id]=1;
41         // this node has already been
42         // extended
43         for(auto &it:mp[t.id])
44         {

```

```

41         to=it.id;
42         w=it.v;
43         if(dis[to]>dis[t.id]+w)
44         {
45             dis[to]=dis[t.id]+w;
46             if(!vis[to]) q.push({to,dis[to]});
47         }
48     }
49 }
50 }
51 #undef type
52 }dij;
53 // [l,r] b sum{ai*xi}=b, xi>=0
54 ll congruent_short_path(vector<int> &a,ll l,ll r)
55 {
56     int n,i,j;
57     ll res;
58     n=a.size();
59     sort(a.begin(),a.end());
60     a.resize(unique(a.begin(),a.end())-a.begin());
61     dij.init(a[0]);
62     for(i=0;i<a[0];i++)
63     {
64         for(j=1;j<n;j++)
65         {
66             dij.add_edge(i,(i+a[j])%a[0],a[j]);
67         }
68     }
69     dij.work(0);
70     res=0;
71     for(i=0;i<a[0];i++)
72     {
73         if(dij.dis[i]<=r) res+=(r-dij.dis[i])/a[0]+1;
74         if(dij.dis[i]<=l-1) res-=((l-1)-dij.dis[i])/a[0]+1;
75     }
76     return res;
77 }

```

4.3 最小生成树

4.3.1 kruskal

```

1 struct Disjoint_Set_Union
2 {
3     int pre[MAX],sz[MAX];
4     void init(int n)
5     {
6         int i;
7         for(i=1;i<=n;i++)
8         {
9             pre[i]=i;
10            sz[i]=1;
11        }

```

```

12    }
13    int find(int x)
14    {
15        if(pre[x]!=x) pre[x]=find(pre[x]);
16        return pre[x];
17    }
18    bool merge(int x,int y,bool dir=false)
19    {
20        x=find(x);
21        y=find(y);
22        if(x==y) return 0;
23        if(!dir && sz[x]>sz[y]) swap(x,y);
24        pre[x]=y; // x -> y
25        sz[y]+=sz[x];
26        return 1;
27    }
28 }dsu;
29 struct Kruskal
30 {
31     #define type int
32     #define inf INF
33     struct edge{int x,y;type w;};
34     vector<edge> e;
35     void init(){e.clear();}
36     void add_edge(int a,int b,type w){e.push_back({a,b,w});}
37     type work(int n)
38     {
39         int i,cnt;
40         type res=0;
41         dsu.init(n);
42         sort(e.begin(),e.end(),[&](edge x,edge y){
43             return x.w<y.w;
44         });
45         for(auto &it:e)
46         {
47             if(dsu.merge(it.x,it.y)) res+=it.w;
48         }
49         cnt=0;
50         for(i=1;i<=n;i++) cnt+=dsu.find(i)==i;
51         if(cnt!=1) return inf; // no connect
52         return res;
53     }
54     #undef type
55     #undef inf
56 }krsk;

```

4.3.2 kruskal 重构树

```

1 struct Disjoint_Set_Union
2 {
3     int pre[MAX],sz[MAX];
4     void init(int n)
5     {

```

```

6         int i;
7         for(i=1;i<=n;i++)
8         {
9             pre[i]=i;
10            sz[i]=1;
11        }
12    }
13    int find(int x)
14    {
15        if(pre[x]!=x) pre[x]=find(pre[x]);
16        return pre[x];
17    }
18    bool merge(int x,int y,bool dir=false)
19    {
20        x=find(x);
21        y=find(y);
22        if(x==y) return 0;
23        if(!dir && sz[x]>sz[y]) swap(x,y);
24        pre[x]=y; // x -> y
25        sz[y]+=sz[x];
26        return 1;
27    }
28    }dsu;
29    struct Kruskal_Tree
30    {
31        #define type int
32        struct edge{int x,y;type w;};
33        vector<edge> e;
34        void init(){e.clear();}
35        void add_edge(int x,int y,type w){e.push_back({x
36            ,y,w});}
37        int build_kruskal_tree(int n,vector<int> *mp,
38            type *w)
39        {
40            int rt,x,y,i;
41            for(i=1;i<=2*n-1;i++)
42            {
43                mp[i].clear();
44                w[i]=0;
45            }
46            dsu.init(2*n-1);
47            sort(e.begin(),e.end(),[&](edge x,edge y){
48                return x.w<y.w;
49            });
50            rt=n;
51            for(auto &it:e)
52            {
53                x=dsu.find(it.x);
54                y=dsu.find(it.y);
55                if(x==y) continue;
56                rt++;
57                w[rt]=it.w;
58                mp[rt].push_back(x);
59                mp[rt].push_back(y);

```

```

58        dsu.merge(x,rt);
59        dsu.merge(y,rt);
60    }
61    return rt;
62    }
63    #undef type
64    }krsk;

```

4.3.3 prim

```

1    struct Prim
2    {
3        #define type int
4        const type inf=INF;
5        struct node{int id;type w;};
6        static const int N=MAX;
7        vector<node> mp[N];
8        type dis[N];
9        int n,vis[N];
10       void init(int _n)
11       {
12           n=_n;
13           for(int i=0;i<=n;i++) mp[i].clear();
14       }
15       void add_edge(int x,int y,type w)
16       {
17           mp[x].push_back({y,w});
18           mp[y].push_back({x,w});
19       }
20       type work()
21       {
22           int i,x,cnt;
23           type res,mn;
24           for(i=0;i<=n;i++)
25           {
26               dis[i]=inf;
27               vis[i]=0;
28           }
29           x=1;
30           for(auto &to:mp[x]) dis[to.id]=min(dis[to.id
31               ],to.w);
32           res=0;
33           cnt=0;
34           while(1)
35           {
36               mn=inf;
37               vis[x]=1;
38               cnt++;
39               for(i=1;i<=n;i++)
40               {
41                   if(!vis[i]&&dis[i]<mn)
42                   {
43                       mn=dis[i];

```

```

44         }
45     }
46     if(mn==inf) break;
47     res+=mn;
48     for(auto &to:mp[x]) dis[to.id]=min(dis[to
        .id],to.w);
49     }
50     if(cnt<n) return inf; // no connect
51     return res;
52 }
53 #undef type
54 }prim;
55 /*
56 O(n^2+m)
57
58 prim.init(n);
59 prim.add_edge(x,y,w); x<->y
60 prim.work();
61 */

```

4.4 二分图匹配

4.4.1 二分图最大匹配

```

1  /*C 作 gj
2  μ. 匹配问题=CCg1
3  • %g, 2, =CCgμn-
4  b% CC= • %g, 2, =CCgμn-
5  */
6  struct Bipartite_Matching
7  {
8      static const int N=;
9      static const int M=;
10     int n,m;
11     vector<int> mp[N];
12     int flag[N],s[N],link[M],used[M];
13     void init(int _n,int _m)
14     {
15         n=_n;
16         m=_m;
17         for(int i=0;i<=n;i++) mp[i].clear();
18     }
19     void add_edge(int a,int b){mp[a].push_back(b);}
20     bool dfs(int x,int timetag)
21     {
22         int i,to;
23         flag[x]=1;
24         for(i=0;i<mp[x].size();i++)
25         {
26             to=mp[x][i];
27             if(used[to]==timetag) continue;
28             used[to]=timetag;
29             if(link[to]==-1||dfs(link[to],timetag))
30             {

```

```

31         link[to]=x;
32         s[x]=to;
33         return 1;
34     }
35     }
36     return 0;
37 }
38 int max_match()
39 {
40     int i,res;
41     memset(link,-1,sizeof link);
42     memset(s,-1,sizeof s);
43     memset(used,0,sizeof used);
44     res=0;
45     for(i=1;i<=n;i++)
46     {
47         if(mp[i].size()==0) continue;
48         if(dfs(i,i)) res++;
49     }
50     return res;
51 }
52 int min_cover(vector<int> &x,vector<int> &y)
53 {
54     int i,res;
55     res=max_match();
56     memset(flag,0,sizeof flag);
57     memset(used,0,sizeof used);
58     x.clear();
59     y.clear();
60     for(i=1;i<=n;i++)
61     {
62         if(s[i]==-1) dfs(i);
63     }
64     for(i=1;i<=n;i++)
65     {
66         if(!flag[i]) x.push_back(i);
67     }
68     for(i=1;i<=m;i++)
69     {
70         if(used[i]) y.push_back(i);
71     }
72     return res;
73 }
74 }bpm;
75 /*
76 O(n*m) : n node,m edge
77 bpm.init(n,m);
78 bpm.add_edge(a,b); a:left,1-n b:right,1-m
79 */

```

4.4.2 二分图最大权完美匹配

```

1 struct Kuhn_Munkres
2 {

```



```

3  #define type int
4  const type inf=INF;
5  static const int N=;
6  int n,mx[N],my[N],pre[N];
7  type slk[N],lx[N],ly[N],w[N][N];
8  bool vx[N],vy[N];
9  void init(int _n)
10 {
11     n=_n;
12     for(int i=1;i<=n;i++)
13     {
14         for(int j=1;j<=n;j++)
15         {
16             /*
17              - 000000000000, 000000000000
18              - 000000000000^2», 000000000000
19              400000'00000000μ,000000000000
20              • φ»μ0»-¼
21             */
22             w[i][j]=0;
23         }
24     }
25 }
26 void add_edge(int x,int y,type val)
27 {
28     // 0L0000000000000000-°±,°
29     w[x][y]=val;
30 }
31 void match(int y){while(y) swap(y,mx[my[y]=pre[y]]);}
32 void bfs(int x)
33 {
34     int i,y;
35     type d;
36     for(i=1;i<=n;i++)
37     {
38         vx[i]=vy[i]=pre[i]=0;
39         slk[i]=inf;
40     }
41     queue<int> q;
42     q.push(x);
43     vx[x]=1;
44     while(1)
45     {
46         while(!q.empty())
47         {
48             x=q.front();
49             q.pop();
50             for(y=1;y<=n;y++)
51             {
52                 d=lx[x]+ly[y]-w[x][y];
53                 if(!vy[y]&&d<=slk[y])
54                 {
55                     pre[y]=x;

```

```

56                 if(!d)
57                 {
58                     if(!my[y]) return match(y);
59                     q.push(my[y]);
60                     vx[my[y]]=1;
61                     vy[y]=1;
62                 }
63                 else slk[y]=d;
64             }
65         }
66     }
67     d=inf+1;
68     for(i=1;i<=n;i++)
69     {
70         if(!vy[i]&&slk[i]<d)
71         {
72             d=slk[i];
73             y=i;
74         }
75     }
76     for(i=1;i<=n;i++)
77     {
78         if(vx[i]) lx[i]-=d;
79         if(vy[i]) ly[i]+=d;
80         else slk[i]-=d;
81     }
82     if(!my[y]) return match(y);
83     q.push(my[y]);
84     vx[my[y]]=1;
85     vy[y]=1;
86 }
87 }
88 type max_match()
89 {
90     int i,j;
91     type res;
92     for(i=1;i<=n;i++)
93     {
94         mx[i]=my[i]=ly[i]=0;
95         lx[i]=*max_element(w[i]+1,w[i]+n+1);
96     }
97     for(i=1;i<=n;i++) bfs(i);
98     res=0;
99     for(i=1;i<=n;i++)
100     {
101         // ^2»
102         res+=w[i][mx[i]];
103     }
104     return res;
105 }
106 #undef type
107 }km;
108 /*
109 O(n^3)

```

```

110 km.init(n);
111 km.add_edge(a,b,val); a,b: 1~n
112 */

```

4.5 最大流

4.5.1 dinic

```

1 struct Dinic
2 {
3     #define type int
4     const type inf=INF;
5     static const int N=;
6     struct node
7     {
8         int from,to;
9         type cap,flow;
10        node(int u,int v,type c,type f):from(u),to(v)
11            ,cap(c),flow(f){}
12    };
13    int n,s,t;
14    vector<node> edge;
15    vector<int> mp[N];
16    int vis[N],dist[N],id[N];
17    void init(int _n)
18    {
19        n=_n;
20        edge.clear();
21        for(int i=0;i<=n;i++)
22        {
23            mp[i].clear();
24            id[i]=dist[i]=vis[i]=0;
25        }
26    }
27    void add_edge(int from,int to,type cap)
28    {
29        edge.push_back(node(from,to,cap,0));
30        edge.push_back(node(to,from,0,0));
31        int m=edge.size();
32        mp[from].push_back(m-2);
33        mp[to].push_back(m-1);
34    }
35    bool bfs()
36    {
37        int i,x;
38        memset(vis,0,sizeof vis);
39        queue<int>q;
40        q.push(s);
41        dist[s]=0;
42        vis[s]=1;
43        while(!q.empty())
44        {
45            x=q.front();
46            q.pop();

```

```

46        for(i=0;i<mp[x].size();i++)
47        {
48            node &e=edge[mp[x][i]];
49            if(!vis[e.to]&&e.cap>e.flow)
50            {
51                vis[e.to]=1;
52                dist[e.to]=dist[x]+1;
53                q.push(e.to);
54            }
55        }
56    }
57    return vis[t];
58 }
59 type dfs(int x,type a)
60 {
61     if(x==t||!a) return a;
62     type flow=0,f;
63     for(int &i=id[x];i<mp[x].size();i++)
64     {
65         node &e=edge[mp[x][i]];
66         if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min
67             (a,e.cap-e.flow)))>0)
68         {
69             e.flow+=f;
70             edge[mp[x][i]^1].flow-=f;
71             flow+=f;
72             a-=f;
73             if(!a) break;
74         }
75     }
76     return flow;
77 }
78 type max_flow(int _s,int _t)
79 {
80     s=_s;
81     t=_t;
82     type res=0;
83     while(bfs())
84     {
85         for(int i=0;i<=n;i++) id[i]=0;
86         res+=dfs(s,inf);
87     }
88     return res;
89 }
90 #undef type
91 }dc;
92 /*
93 O(n^2*m)
94 bipartite graph: O(m*sqrt(n))
95
96 dc.init(n);
97 dc.add_edge(a,b,cap); a,b: 1~n
98 */

```

4.5.2 有源汇上下界网络流

```

1 struct Dinic
2 {
3     #define type int
4     const type inf=INF;
5     static const int N=;
6     struct node
7     {
8         int from,to;
9         type cap,flow;
10        node(int u,int v,type c,type f):from(u),to(v)
11            ,cap(c),flow(f){}
12    };
13    int n,s,t,s1,t1;
14    vector<node> edge;
15    vector<int> mp[N];
16    int vis[N],dist[N],id[N];
17    type in[N],out[N];
18    void init(int _n)
19    {
20        s1=_n+1;
21        t1=s1+1;
22        n=t1;
23        assert(n<N);
24        edge.clear();
25        for(int i=0;i<=n;i++)
26        {
27            mp[i].clear();
28            id[i]=dist[i]=0;
29            in[i]=out[i]=0;
30        }
31    }
32    void add_edge(int from,int to,type lcap,type
33        rcap)
34    {
35        edge.push_back(node(from,to,rcap-lcap,0));
36        edge.push_back(node(to,from,0,0));
37        in[to]+=lcap;
38        out[from]+=lcap;
39        int m=edge.size();
40        mp[from].push_back(m-2);
41        mp[to].push_back(m-1);
42    }
43    bool bfs()
44    {
45        int i,x;
46        for(int i=0;i<=n;i++) vis[i]=0;
47        queue<int>q;
48        q.push(s);
49        dist[s]=0;
50        vis[s]=1;
51        while(!q.empty())
52        {

```

```

51            x=q.front();
52            q.pop();
53            for(i=0;i<mp[x].size();i++)
54            {
55                node &e=edge[mp[x][i]];
56                if(!vis[e.to]&&e.cap>e.flow)
57                {
58                    vis[e.to]=1;
59                    dist[e.to]=dist[x]+1;
60                    q.push(e.to);
61                }
62            }
63        }
64        return vis[t];
65    }
66    type dfs(int x,type a)
67    {
68        if(x==t||!a) return a;
69        type flow=0,f;
70        for(int &i=id[x];i<mp[x].size();i++)
71        {
72            node &e=edge[mp[x][i]];
73            if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min
74                (a,e.cap-e.flow)))>0)
75            {
76                e.flow+=f;
77                edge[mp[x][i]^1].flow-=f;
78                flow+=f;
79                a-=f;
80                if(!a) break;
81            }
82        }
83        return flow;
84    }
85    type dinic(int _s,int _t)
86    {
87        int i;
88        s=_s;
89        t=_t;
90        type res=0;
91        while(bfs())
92        {
93            for(i=0;i<=n;i++) id[i]=0;
94            res+=dfs(s,inf);
95        }
96        return res;
97    }
98    type min_flow(int _s,int _t)
99    {
100        int i;
101        s=s1;
102        t=t1;
103        for(i=0;i<=s1;i++)
104        {

```

```

104         if(in[i]>out[i]) add_edge(s,i,0,in[i]-out
105             [i]);
106         else if(in[i]<out[i]) add_edge(i,t,0,out[
107             i]-in[i]);
108     }
109     add_edge(_t,_s,0,inf);
110     dinic(s,t);
111     for(i=0;i<mp[s].size();i++)
112     {
113         node &e=edge[mp[s][i]];
114         if(e.cap-e.flow!=0) return -1;
115     }
116     s=_s;
117     t=_t;
118     type res;
119     for(i=0;i<mp[t].size();i++)
120     {
121         node &e=edge[mp[t][i]];
122         if(e.to==s)
123         {
124             res=e.flow;
125             e.flow=edge[mp[t][i]^1].flow=0;
126             e.cap=edge[mp[t][i]^1].cap=0;
127         }
128     }
129     res-=dinic(t,s);
130     return res;
131 }
132 type max_flow(int _s,int _t)
133 {
134     int i;
135     s=s1;
136     t=t1;
137     for(i=0;i<s1;i++)
138     {
139         if(in[i]>out[i]) add_edge(s,i,0,in[i]-out
140             [i]);
141         else if(in[i]<out[i]) add_edge(i,t,0,out[
142             i]-in[i]);
143     }
144     add_edge(_t,_s,0,inf);
145     dinic(s,t);
146     for(i=0;i<mp[s].size();i++)
147     {
148         node &e=edge[mp[s][i]];
149         if(e.cap-e.flow!=0) return -1;
150     }
151     s=_s;
152     t=_t;
153     type res;
154     for(i=0;i<mp[t].size();i++)
155     {
156         node &e=edge[mp[t][i]];
157         if(e.to==s)

```

```

154     {
155         res=e.flow;
156         e.flow=edge[mp[t][i]^1].flow=0;
157         e.cap=edge[mp[t][i]^1].cap=0;
158     }
159     }
160     res+=dinic(s,t);
161     return res;
162 }
163 #undef type
164 }dc;
165 /*
166 dc.init(n);
167 dc.add_edge(a,b,lcap,rcap); a,b: 1~n
168 dc.min_flow(s,t);
169 dc.max_flow(s,t);
170 */

```

4.6 费用流

4.6.1 spfa 费用流

```

1 struct MCMF
2 {
3     #define type int
4     #define inf INF
5     static const int N=;
6     struct node
7     {
8         int from,to;
9         type cap,flow,cost;
10        node(){}
11        node(int u,int v,type c,type f,type co):from(
12            u),to(v),cap(c),flow(f),cost(co){}
13    };
14    int n,s,t;
15    vector<node> edge;
16    vector<int> mp[N];
17    int vis[N],id[N];
18    type d[N],a[N];
19    void init(int _n)
20    {
21        n=_n;
22        for(int i=0;i<n;i++) mp[i].clear();
23        edge.clear();
24    }
25    void add_edge(int from,int to,type cap,type cost
26        =0)
27    {
28        edge.pb(node(from,to,cap,0,cost));
29        edge.pb(node(to,from,0,0,-cost));
30        int m=edge.size();
31        mp[from].pb(m-2);
32        mp[to].pb(m-1);

```

```

31 }
32 bool spfa(type& flow,type& cost)
33 {
34     for(int i=0;i<=n;i++)
35     {
36         d[i]=inf;
37         vis[i]=0;
38     }
39     d[s]=0;vis[s]=1;id[s]=0;a[s]=inf;
40     queue<int> q;
41     q.push(s);
42     while(!q.empty())
43     {
44         int x=q.front();
45         q.pop();
46         vis[x]=0;
47         for(int i=0;i<mp[x].size();i++)
48         {
49             node& e=edge[mp[x][i]];
50             int to=e.to;
51             if(e.cap>e.flow&&d[to]>d[x]+e.cost)
52             {
53                 d[to]=d[x]+e.cost;
54                 a[to]=min(a[x],e.cap-e.flow);
55                 id[to]=mp[x][i];
56                 if(!vis[to])
57                 {
58                     vis[to]=1;
59                     q.push(to);
60                 }
61             }
62         }
63     }
64     if(d[t]==inf) return false;
65     flow+=a[t];
66     cost+=a[t]*d[t];
67     int x=t;
68     while(x!=s)
69     {
70         edge[id[x]].flow+=a[t];
71         edge[id[x]^1].flow-=a[t];
72         x=edge[id[x]].from;
73     }
74     return true;
75 }
76 pair<type,type> mincost_maxflow(int _s,int _t)
77 {
78     type flow=0,cost=0;
79     s=_s;
80     t=_t;
81     while(spfa(flow,cost));
82     return MP(cost,flow);
83 }
84 #undef type

```

```

85 #undef inf
86 }mcmf;
87 /*
88 mcmf.init(n);
89 mcmf.add_edge(a,b,cap,cost); a,b: 1~n
90 */

```

4.6.2 dijkstra 费用流 (dij 求 h)

```

1 struct MCMF_dij
2 {
3     #define type int
4     #define inf INF
5     #define PTI pair<type,int>
6     static const int N=;
7     struct node
8     {
9         int from,to;
10        type flow,cost;
11        node(){}
12        node(int u,int v,type f,type co):from(u),to(v),flow(f),cost(co){}
13    };
14    int n,s,t,id[N];
15    vector<node> edge;
16    vector<int> mp[N];
17    type dis[N],h[N];
18    void init(int _n)
19    {
20        n=_n;
21        for(int i=0;i<=n;i++) mp[i].clear();
22        edge.clear();
23    }
24    void add_edge(int from,int to,type cap,type cost=0)
25    {
26        edge.push_back(node(from,to,cap,cost));
27        edge.push_back(node(to,from,0,-cost));
28        int m=edge.size();
29        mp[from].push_back(m-2);
30        mp[to].push_back(m-1);
31    }
32    bool dij()
33    {
34        int i,x,to;
35        type cost,now_cost;
36        for(i=0;i<=n;i++) dis[i]=inf;
37        dis[s]=0;id[s]=0;
38        priority_queue<PTI ,vector<PTI>,greater<PTI> > q;
39        q.push({type(0),s});
40        while(!q.empty())
41        {
42            PTI tmp=q.top();

```

```

43     q.pop();
44     cost=tmp.first;
45     x=tmp.second;
46     if(cost>dis[x]) continue;
47     for(i=0;i<mp[x].size();i++)
48     {
49         node& e=edge[mp[x][i]];
50         to=e.to;
51         type now_cost=e.cost+h[x]-h[to];
52         if(e.flow>0&&dis[to]>dis[x]+now_cost)
53         {
54             dis[to]=dis[x]+now_cost;
55             q.push({dis[to],to});
56             e.from=x;
57             id[to]=mp[x][i];
58         }
59     }
60 }
61 return dis[t]!=inf;
62 }
63 pair<type,type> mincost_maxflow(int _s,int _t)
64 {
65     int i;
66     type flow=0,cost=0;
67     for(int i=0;i<=n;i++) h[i]=0;
68     s=_s;
69     t=_t;
70     while(dij())
71     {
72         for(i=0;i<=n;i++) h[i]+=dis[i];
73         type new_flow=inf;
74         for(i=t;i!=s;i=edge[id[i]].from)
75         {
76             new_flow=min(new_flow,edge[id[i]].flow);
77         }
78         for(i=t;i!=s;i=edge[id[i]].from)
79         {
80             edge[id[i]].flow-=new_flow;
81             edge[id[i]^1].flow+=new_flow;
82         }
83         flow+=new_flow;
84         cost+=new_flow*h[t];
85     }
86     return {cost,flow};
87 }
88 #undef type
89 #undef inf
90 #undef PTI
91 }mcmf; //upper: O(nmlog(nm) + max_flow*mlogm)

```

4.6.3 dijkstra 费用流 (spfa 求 h)

```

1 struct MCMF_dij
2 {
3     #define type int
4     #define inf INF
5     #define PTI pair<type,int>
6     static const int N=;
7     struct node
8     {
9         int from,to;
10        type flow,cost;
11        node(){}
12        node(int u,int v,type f,type co):from(u),to(v),flow(f),cost(co){}
13    };
14    int n,s,t,id[N],vis[N];
15    vector<node> edge;
16    vector<int> mp[N];
17    type dis[N],h[N];
18    void init(int _n)
19    {
20        n=_n;
21        for(int i=0;i<=n;i++) mp[i].clear();
22        edge.clear();
23    }
24    void add_edge(int from,int to,type cap,type cost=0)
25    {
26        edge.push_back(node(from,to,cap,cost));
27        edge.push_back(node(to,from,0,-cost));
28        int m=edge.size();
29        mp[from].push_back(m-2);
30        mp[to].push_back(m-1);
31    }
32    void spfa()
33    {
34        int i,x,to;
35        for(i=0;i<=n;i++)
36        {
37            h[i]=inf;
38            vis[i]=0;
39        }
40        queue<int> q;
41        q.push(s);
42        h[s]=0;
43        vis[s]=1;
44        while(!q.empty())
45        {
46            x=q.front();
47            q.pop();
48            vis[x]=0;
49            for(i=0;i<mp[x].size();i++)
50            {
51                node &e=edge[mp[x][i]];
52                to=e.to;

```

```

53         if(e.flow>0&&h[to]>h[x]+e.cost)
54         {
55             h[to]=h[x]+e.cost;
56             if(!vis[to])
57             {
58                 vis[to]=1;
59                 q.push(to);
60             }
61         }
62     }
63 }
64 }
65 bool dij()
66 {
67     int i,x,to;
68     type cost,now_cost;
69     for(i=0;i<n;i++) dis[i]=inf;
70     dis[s]=0;id[s]=0;
71     priority_queue<PTI ,vector<PTI>,greater<PTI>
72         > q;
73     q.push({type(0),s});
74     while(!q.empty())
75     {
76         PTI tmp=q.top();
77         q.pop();
78         cost=tmp.first;
79         x=tmp.second;
80         if(cost>dis[x]) continue;
81         for(i=0;i<mp[x].size();i++)
82         {
83             node& e=edge[mp[x][i]];
84             to=e.to;
85             type now_cost=e.cost+h[x]-h[to];
86             if(e.flow>0&&dis[to]>dis[x]+now_cost)
87             {
88                 dis[to]=dis[x]+now_cost;
89                 q.push({dis[to],to});
90                 e.from=x;
91                 id[to]=mp[x][i];
92             }
93         }
94         return dis[t]!=inf;
95     }
96     pair<type,type> mincost_maxflow(int _s,int _t)
97     {
98         int i;
99         type flow=0,cost=0;
100         s=_s;
101         t=_t;
102         spfa();
103         while(dij())
104         {
105             for(i=0;i<n;i++) h[i]+=dis[i];

```

```

106         type new_flow=inf;
107         for(i=t;i!=s;i=edge[id[i]].from)
108         {
109             new_flow=min(new_flow,edge[id[i]].flow
110                 );
111         }
112         for(i=t;i!=s;i=edge[id[i]].from)
113         {
114             edge[id[i]].flow-=new_flow;
115             edge[id[i]^1].flow+=new_flow;
116         }
117         flow+=new_flow;
118         cost+=new_flow*h[t];
119     }
120     return {cost,flow};
121 }
122 #undef type
123 #undef inf
124 #undef PTI
125 }mcmf; // upper: O(nm + max_flow*mlogm)

```

4.7 连通性

4.7.1 强连通分量

```

1 struct Strongly_Connected_Components
2 {
3     int scc_cnt,tot;
4     int low[MAX],dfn[MAX],col[MAX],sz[MAX];
5     int st[MAX],top,flag[MAX];
6     vector<int> *mp;
7     void dfs(int x)
8     {
9         int tmp;
10        st[top++]=x;
11        flag[x]=1;
12        low[x]=dfn[x]=++tot;
13        for(auto &to:mp[x])
14        {
15            if(!dfn[to])
16            {
17                dfs(to);
18                low[x]=min(low[x],low[to]);
19            }
20            else if(flag[to]) low[x]=min(low[x],dfn[to]);
21        }
22        if(low[x]==dfn[x])
23        {
24            scc_cnt++;
25            do
26            {
27                tmp=st[--top];
28                flag[tmp]=0;

```

```

29         col[tmp]=scc_cnt;
30         sz[scc_cnt]++;
31     }while(tmp!=x);
32 }
33 }
34 void work(int n,vector<int> *_mp)
35 {
36     int i;
37     mp=_mp;
38     for(i=1;i<=n;i++) col[i]=sz[i]=flag[i]=0;
39     scc_cnt=top=tot=0;
40     for(i=1;i<=n;i++)
41     {
42         if(col[i]) continue;
43         dfs(i);
44     }
45 }
46 void rebuild(int n,vector<int> *g)
47 {
48     int i;
49     for(i=1;i<=n;i++) g[i].clear();
50     for(i=1;i<=n;i++)
51     {
52         for(auto &to:mp[i])
53         {
54             if(col[i]==col[to]) continue;
55             g[col[i]].push_back(col[to]);
56         }
57     }
58 }
59 }scc;
60 /*
61 scc.work(n,mp);
62 */

```

4.7.2 边双连通分量

```

1 struct Edge_Biconnected_Component
2 {
3     int bcc_cnt,tot;
4     int low[MAX],dfn[MAX],col[MAX];
5     int st[MAX],top;
6     vector<int> mp[MAX];
7     vector<pair<int,int>> bridge;
8     void dfs(int x,int fa)
9     {
10         int i,tmp,k;
11         st[top++]=x;
12         low[x]=dfn[x]=++tot;
13         k=0;
14         for(auto &to:mp[x])
15         {
16             if(to==fa&&!k)
17             {

```

```

18                 k++;
19                 continue;
20             }
21             if(!dfn[to])
22             {
23                 dfs(to,x);
24                 low[x]=min(low[x],low[to]);
25                 if(low[to]>dfn[x]) bridge.push_back({x
26                     ,to});
27             }
28             else low[x]=min(low[x],dfn[to]);
29         }
30         if(low[x]==dfn[x])
31         {
32             bcc_cnt++;
33             do
34             {
35                 tmp=st[--top];
36                 col[tmp]=bcc_cnt;
37             }while(tmp!=x);
38         }
39     }
40     void work(int n,vector<int> *_mp)
41     {
42         int i;
43         for(i=1;i<=n;i++)
44         {
45             low[i]=dfn[i]=0;
46             mp[i]=_mp[i];
47         }
48         bcc_cnt=top=tot=0;
49         bridge.clear();
50         for(i=1;i<=n;i++)
51         {
52             if(!dfn[i]) dfs(i,i);
53         }
54     }
55     void rebuild(int n,vector<int> *g)
56     {
57         int i;
58         for(i=1;i<=n;i++) g[i].clear();
59         for(i=1;i<=n;i++)
60         {
61             for(auto &to:mp[i])
62             {
63                 if(col[i]==col[to]) continue;
64                 g[col[i]].push_back(col[to]);
65             }
66         }
67     }
68     }bcc;
69     /*
70     bcc.work(n,mp);
71     */

```


4.7.3 点双连通分量

```

1 struct Node_Biconnected_Component
2 {
3     int low[MAX],dfn[MAX],tot;
4     vector<int> *mp,cut_node;
5     bool cut_vis[MAX];
6     void dfs(int x,int fa)
7     {
8         int i,cnt;
9         low[x]=dfn[x]=++tot;
10        cnt=0;
11        for(auto &to:mp[x])
12        {
13            if(!dfn[to])
14            {
15                cnt++;
16                dfs(to,x);
17                low[x]=min(low[x],low[to]);
18                if(x!=fa && low[to]>=dfn[x]) cut_vis[x]
19                    =1;
20            }
21            else if(to!=fa) low[x]=min(low[x],dfn[to]);
22        }
23        if(x==fa && cnt>1) cut_vis[x]=1;
24        if(cut_vis[x]) cut_node.push_back(x);
25    }
26    void work(int n,vector<int> *_mp)
27    {
28        int i;
29        mp=_mp;
30        cut_node.clear();
31        for(i=0;i<n;i++) low[i]=dfn[i]=cut_vis[i]=0;
32        tot=0;
33        for(i=1;i<n;i++)
34        {
35            if(!dfn[i]) dfs(i,i);
36        }
37    }bcc;
38    /*
39    bcc.work(n,mp);
40    */

```

4.7.4 圆方树

```

1 struct Round_Square_Tree
2 {
3     int tot,id,bcc_cnt;
4     int low[MAX],dfn[MAX];
5     int st[MAX],top;
6     vector<int> *mp,*g;
7     void dfs(int x,int fa)

```

```

8     {
9         int i,tmp,k;
10        st[top++]=x;
11        low[x]=dfn[x]=++tot;
12        for(auto &to:mp[x])
13        {
14            if(!dfn[to])
15            {
16                dfs(to,x);
17                low[x]=min(low[x],low[to]);
18                if(low[to]==dfn[x])
19                {
20                    id++;
21                    while(top)
22                    {
23                        tmp=st[--top];
24                        g[tmp].push_back(id);
25                        g[id].push_back(tmp);
26                        if(tmp==to) break;
27                    }
28                    g[x].push_back(id);
29                    g[id].push_back(x);
30                }
31            }
32            else low[x]=min(low[x],dfn[to]);
33        }
34    }
35    int build(int n,vector<int> *_mp,vector<int> *_g)
36    {
37        int i;
38        mp=_mp;
39        g=_g;
40        for(i=0;i<n;i++) low[i]=dfn[i]=0;
41        for(i=0;i<=(n<<1);i++) g[i].clear();
42        top=tot=0;
43        id=n;
44        for(i=1;i<n;i++)
45        {
46            if(!dfn[i]) dfs(i,i);
47        }
48        bcc_cnt=id-n;
49        return id;
50    }
51 }rst;
52 /*
53 vector<int> mp[MAX],g[MAX<<1];
54 tot=rst.build(n,mp,g);
55 */

```

4.8 团

4.8.1 最大团

```

1 struct Maximum_Clique
2 {
3     static const int N=;
4     vector<int> sol; // vertex of maximum clique
5     int mp[N][N/30+1],s[N][N/30+1];
6     int n,ans,dp[N];
7     void init(int _n)
8     {
9         n=_n;
10        for(int i=0;i<=n;i++)
11        {
12            dp[i]=0;
13            mem(mp[i],0);
14        }
15    }
16    void add_edge(int a,int b) //0~n-1
17    {
18        if(a>b) swap(a,b);
19        if(a==b) return;
20        mp[a][b/32]|=(1<<(b%32));
21    }
22    bool dfs(int x,int k)
23    {
24        int c=0,d=0;
25        for(int i=0;i<(n+31)/32;i++)
26        {
27            s[k][i]=mp[x][i];
28            if(k!=1) s[k][i]&=s[k-1][i];
29            c+=__builtin_popcount(s[k][i]);
30        }
31        if(c==0)
32        {
33            if(k>ans)
34            {
35                ans=k;
36                sol.clear();
37                sol.pb(x);
38                return 1;
39            }
40            return 0;
41        }
42        for(int i=0;i<(n+31)/32;i++)
43        {
44            for(int a=s[k][i];a;d++)
45            {
46                if(k+(c-d)<=ans) return 0;
47                int lb=a&(-a),lg=0;
48                a^=lb;
49                while(lb!=1)
50                {
51                    lb=(unsigned int)(lb)>>1;
52                    lg++;
53                }

```

```

54                int u=i*32+lg;
55                if(k+dp[u]<=ans) return 0;
56                if(dfs(u,k+1))
57                {
58                    sol.pb(x);
59                    return 1;
60                }
61            }
62        }
63        return 0;
64    }
65    int maximum_clique()
66    {
67        ans=0;
68        for(int i=n-1;i>=0;i--)
69        {
70            dfs(i,1);
71            dp[i]=ans;
72        }
73        return ans;
74    }
75 }mcp;
76 /*
77 undirected graph
78 mcp.init(n);
79 mcp.add_edge(a,b); a,b: 0~n-1
80 */

```

4.8.2 极大团计数

```

1 struct Bron_Kerbosch
2 {
3     static const int N=;
4     bitset<N> MASK,ZERO,mp[N];
5     int n,cnt_clique;
6     void init(int _n)
7     {
8         n=_n;
9         for(int i=0;i<=n;i++) mp[i].reset();
10        ZERO.reset();
11        MASK=ZERO;
12        MASK.flip();
13    }
14    void add_edge(int a,int b) //0~n-1 , undir
15    {
16        if(a==b) return;
17        mp[a][b]=mp[b][a]=1;
18    }
19    void dfs(bitset<N> now,bitset<N> some,bitset<N>
20            none)
21    {
22        if(some.none()&&none.none())//one maximal
23            clique

```

```

23         cnt_clique++;
24         return;
25     }
26     bitset<N> r=some;
27     bool fi=1;
28     for(int i=0;i<n;i++)
29     {
30         if(!r[i]) continue;
31         if(fi)
32         {
33             fi=0;
34             r&=mp[i]^MASK;
35         }
36         now[i]=1;
37         dfs(now,some&mp[i],none&mp[i]);
38         now[i]=0;
39         some[i]=0;
40         none[i]=1;
41     }
42 }
43 int count_maximal_clique()
44 {
45     cnt_clique=0;
46     bitset<N> now;
47     dfs(now,MASK,ZERO);
48     return cnt_clique;
49 }
50 }bkb;
51 /*
52 undirected graph
53 bk.init(n);
54 bk.add_edge(a,b); a,b: 0~n-1
55 */

```

4.9 拓扑排序

```

1 vector<int> topsort(vector<int> &node,vector<int>
   mp[],int *in)
2 {
3     queue<int> q;
4     for(auto &it:node)
5     {
6         if(!in[it]) q.push(it);
7     }
8     vector<int> toplist;
9     while(!q.empty())
10    {
11        int x=q.front();
12        q.pop();
13        toplist.push_back(x);
14        for(auto &to:mp[x])
15        {
16            in[to]--;

```

```

17            if(!in[to]) q.push(to);
18        }
19    }
20    return toplist;
21 }

```

4.10 2-sat

4.10.1 2-sat 输出任意解

```

1 //判断是否有解 输出任意一组解O(n+m)
2 int scc,top,tot;
3 vector<int> mp[MAX];
4 int low[MAX],dfn[MAX],belong[MAX];
5 int stk[MAX],flag[MAX];
6 int pos[MAX],degree[MAX],ans[MAX],outflag[MAX],cnt;
7 vector<int> dag[MAX];
8 void init(int n)
9 {
10     int i;
11     for(i=0;i<2*n;i++)
12     {
13         mp[i].clear();
14         dag[i].clear();
15         low[i]=0;
16         dfn[i]=0;
17         stk[i]=0;
18         flag[i]=0;
19         degree[i]=0;
20         outflag[i]=0;
21     }
22     scc=top=tot=0;
23 }
24 void tarjan(int x)
25 {
26     int to,i,temp;
27     stk[top++]=x;
28     flag[x]=1;
29     low[x]=dfn[x]=++tot;
30     for(i=0;i<mp[x].size();i++)
31     {
32         to=mp[x][i];
33         if(!dfn[to])
34         {
35             tarjan(to);
36             low[x]=min(low[x],low[to]);
37         }
38         else if(flag[to]) low[x]=min(low[x],dfn[to]);
39     }
40     if(low[x]==dfn[x])
41     {
42         scc++;
43         do
44         {

```

```

45     temp=stk[--top];
46     flag[temp]=0;
47     belong[temp]=scc;
48     }while(temp!=x);
49 }
50 }
51 void add(int x,int y)
52 {
53     mp[x].pb(y);
54 }
55 void topsort(int n)
56 {
57     int i,t;
58     queue<int> q;
59     cnt=0;
60     for(i=1;i<=scc;i++)
61     {
62         if(degree[i]==0) q.push(i);
63         outflag[i]=0;
64     }
65     while(!q.empty())
66     {
67         t=q.front();
68         q.pop();
69         if(outflag[t]==0)
70         {
71             outflag[t]=1;
72             outflag[pos[t]]=2;
73         }
74         for(i=0;i<sz(dag[t]);i++)
75         {
76             int to=dag[t][i];
77             degree[to]--;
78             if(degree[to]==0) q.push(to);
79         }
80     }
81 }
82 void bulddag(int n)
83 {
84     int i,j,to;
85     for(i=0;i<2*n;i++)
86     {
87         for(j=0;j<sz(mp[i]);j++)
88         {
89             to=mp[i][j];
90             if(belong[i]!=belong[to])
91             {
92                 degree[belong[i]]++;
93                 dag[belong[to]].pb(belong[i]);
94             }
95         }
96     }
97 }
98 void twosat(int n)

```

```

99 {
100     int i;
101     for(i=0;i<2*n;i++)
102     {
103         if(!dfn[i]) tarjan(i);
104     }
105     for(i=0;i<n;i++)
106     {
107         if(belong[2*i]==belong[2*i+1])//无解
108         {
109             puts("NO");
110             return;
111         }
112         pos[belong[2*i]]=belong[2*i+1];
113         pos[belong[2*i+1]]=belong[2*i];
114     }
115     bulddag(n);
116     topsort(n);
117     cnt=0;
118     for(i=0;i<2*n;i++)
119     {
120         if(outflag[belong[i]]==1) ans[cnt++]=i+1;
121     }
122     for(i=0;i<cnt;i++)
123     {
124         printf("%d\n",ans[i]);
125     }
126 }

```

4.10.2 2-sat 字典序最小解

```

1 //判断是否有解 输出字典序最小的解O(n*m)
2 vector<int> mp[MAX];
3 bool flag[MAX];
4 int cnt,s[MAX];
5 void init(int n)
6 {
7     int i;
8     for(i=0;i<2*n;i++)
9     {
10         mp[i].clear();
11     }
12     mem(flag,0);
13 }
14 bool dfs(int x)
15 {
16     int i;
17     if(flag[x^1]) return 0;
18     if(flag[x]) return 1;
19     s[cnt++]=x;
20     flag[x]=1;
21     for(i=0;i<sz(mp[x]);i++)
22     {
23         if(!dfs(mp[x][i])) return 0;

```

```

24     }
25     return 1;
26 }
27 void twosat(int n)
28 {
29     int i;
30     for(i=0;i<2*n;i++)
31     {
32         if(!flag[i]&&!flag[i^1])
33         {
34             cnt=0;
35             if(!dfs(i))
36             {
37                 while(cnt) flag[s[--cnt]]=0;
38                 if(!dfs(i^1))//无解
39                 {
40                     puts("NO");
41                     return;
42                 }
43             }
44         }
45     }
46     for(i=0;i<2*n;i+=2)
47     {
48         if(flag[i]) printf("%d\n",i+1);
49         else printf("%d\n",i+2);
50     }
51 }

```

4.11 支配树

```

1 struct Dominator_Tree
2 {
3     int n,tot,dfn[MAX],best[MAX],semi[MAX],idom[MAX],id[MAX],fa[MAX];
4     vector<int> nex[MAX],pre[MAX],tmp[MAX],son[MAX];
5     void init(int _n)
6     {
7         n=_n;
8         for(int i=0;i<=n;i++)
9         {
10             nex[i].clear();
11             pre[i].clear();
12             tmp[i].clear();
13             son[i].clear();
14             dfn[i]=0;
15             idom[i]=semi[i]=best[i]=fa[i]=i;
16         }
17     }
18     void add_edge(int x,int y)
19     {
20         nex[x].pb(y);
21         pre[y].pb(x);

```

```

22     }
23     int ckmin(int x,int y){return dfn[semi[x]]<dfn[semi[y]]?x:y;}
24     int getfa(int k)
25     {
26         if(k==fa[k]) return k;
27         int ret=getfa(fa[k]);
28         best[k]=ckmin(best[fa[k]],best[k]);
29         return fa[k]=ret;
30     }
31     void dfs(int x)
32     {
33         dfn[x]=++tot;
34         id[tot]=x;
35         for(auto &to:nex[x])
36         {
37             if(dfn[to]) continue;
38             dfs(to);
39             son[x].pb(to);
40         }
41     }
42     void tarjan(vector<int> *mp)
43     {
44         int i,j,k;
45         for(i=tot;i-->0)
46         {
47             k=id[i];
48             for(auto &to:pre[k])
49             {
50                 if(!dfn[to]) continue;
51                 if(dfn[to]<dfn[k])
52                 {
53                     if(dfn[to]<dfn[semi[k]]) semi[k]=to;
54                     ;
55                 }
56                 else
57                 {
58                     getfa(to);
59                     semi[k]=semi[ckmin(best[to],k)];
60                 }
61             }
62             if(k!=semi[k]) tmp[semi[k]].pb(k);
63             for(auto &to:tmp[k])
64             {
65                 getfa(to);
66                 if(semi[best[to]]==k) idom[to]=k;
67                 else idom[to]=best[to];
68             }
69             for(auto &to:son[k]) fa[to]=k;
70         }
71         for(i=2;i<=tot;i++)
72         {
73             k=id[i];

```

```

73         if(idom[k]!=semi[k]) idom[k]=idom[idom[k]
74             []];
75         if (k!=idom[k])
76         {
77             mp[idom[k]].push_back(k); //add edge
78         }
79     }
80 }
81 void work(int rt,vector<int> *mp)
82 {
83     for(int i=0;i<=n;i++) mp[i].clear();
84     tot=0;
85     dfs(rt);
86     tarjan(mp);
87 }
88 }dt;
89 /*
90 dt.init(n);
91 dt.add_edge(a,b); // DAG
92 dt.work(rt,mp);
93 */

```

4.12 最小斯坦纳树

```

1 struct Minimum_Steiner_Tree
2 {
3     #define type int
4     const type inf=INF;
5     static const int N=;
6     static const int K=;
7     struct node
8     {
9         int id;
10        type v;
11        friend bool operator <(node a,node b){return
12            a.v>b.v;}
13    };
14    vector<node> mp[N];
15    type dp[(1<<K)+3][N];
16    int n,vis[N];
17    void init(int _n)
18    {
19        n=_n;
20        for(int i=1;i<=n;i++) mp[i].clear();
21    }
22    void add_edge(int x,int y,type v){ mp[x].
23        push_back({y,v});}
24    void dijkstra(int s)
25    {
26        int i,to;
27        type w;
28        priority_queue<node> q;

```

```

27        for(i=1;i<=n;i++)
28        {
29            vis[i]=0;
30            if(dp[s][i]!=inf) q.push({i,dp[s][i]});
31        }
32        while(!q.empty())
33        {
34            node t=q.top();
35            q.pop();
36            if(vis[t.id]) continue;
37            vis[t.id]=1;
38            for(auto &it:mp[t.id])
39            {
40                to=it.id;
41                w=it.v;
42                if(dp[s][to]>dp[s][t.id]+w)
43                {
44                    dp[s][to]=dp[s][t.id]+w;
45                    if(!vis[to]) q.push({to,dp[s][to]});
46                }
47            }
48        }
49    }
50    type work(vector<int> key_node)
51    {
52        int s,t,i,k;
53        type res;
54        k=key_node.size();
55        for(s=0;s<(1<<k);s++)
56        {
57            for(i=1;i<=n;i++) dp[s][i]=inf;
58        }
59        for(i=0;i<k;i++) dp[(1<<i)][key_node[i]]=0;
60        for(s=0;s<(1<<k);s++)
61        {
62            for(t=s&(s-1);t;t=s&(t-1))
63            {
64                if(t<(s^t)) break;
65                for(i=1;i<=n;i++)
66                {
67                    dp[s][i]=min(dp[s][i],dp[t][i]+dp[s
68                        ^t][i]);
69                }
70                dijkstra(s);
71            }
72            res=inf;
73            for(i=1;i<=n;i++) res=min(res,dp[(1<<k)-1][i
74                ]);
75            return res;
76        }
77        #undef type
78    }stn;

```

```

78  /*
79  minimum spanning tree including all k key_node
80  O(n*3^k + m*log(m)*2^k)
81
82  stn.init(n);
83  stn.add_edge(a,b,w);
84  stn.work(key_node);
85  */

```

5 数论

5.1 素数筛

5.1.1 埃筛

```

1  //x is a prime if prime[x]==x(x>=2)
2  int p[MAX],tot,prime[MAX];
3  void init(int n)
4  {
5      int i,j;
6      tot=0;
7      mem(prime,0);
8      prime[1]=1;
9      for(i=2;i<=n;i++)
10     {
11         if(prime[i]) continue;
12         p[tot++]=i;
13         for(j=i;j<=n;j+=i)
14         {
15             if(!prime[j]) prime[j]=i;
16         }
17     }
18 }

```

5.1.2 线性筛

```

1  //x is a prime if prime[x]==x(x>=2)
2  int p[MAX],tot,prime[MAX];
3  void init_prime(int n)
4  {
5      int i,j;
6      tot=0;
7      memset(prime,0,sizeof prime);
8      prime[1]=1;
9      for(i=2;i<=n;i++)
10     {
11         if(!prime[i]) prime[i]=p[tot++]=i;
12         for(j=0;j<tot&& p[j]*i<=n;j++)
13         {
14             prime[i*p[j]]=p[j];
15             if(i%p[j]==0) break;
16         }
17     }

```

```

18 }

```

5.1.3 区间筛

```

1  //O(r-l+1)
2  const int N=2e7+10;
3  ll p[N],tot;
4  bool vis[N],prime[N];
5  void init(ll l,ll r)
6  {
7      ll i,j,sq=sqrt(r+0.5);
8      tot=0;
9      for(i=0;i<=sq;i++) vis[i]=1;
10     for(i=l;i<=r;i++) prime[i-l]=1;
11     if(l==0) prime[0]=prime[1]=0;
12     if(l==1) prime[0]=0;
13     for(i=2;i<=sq;i++)
14     {
15         if(!vis[i]) continue;
16         for(j=i+i;j<=sq;j+=i) vis[j]=0;
17         for(j=max(2LL,(l+i-1)/i)*i;j<=r;j+=i) prime[j-l]=0;
18     }
19     for(i=l;i<=r;i++)
20     {
21         if(prime[i-l]) p[tot++]=i;
22     }
23 }

```

5.2 逆元

5.2.1 exgcd 求逆元

```

1  ll inv_exgcd(ll a,ll p)
2  {
3      ll g,x,y;
4      g=exgcd(a,p,x,y);
5      return g==1?(x%p+p)%p:-1;
6  }
7  /*
8  gcd(a,p)!=1, no solution, return -1
9  */

```

5.2.2 线性预处理

```

1  ll inv[MAX];
2  void init_inv(int n,ll p)
3  {
4      inv[1]=1;
5      for(int i=2;i<=n;i++) inv[i]=(p-p/i)*inv[p%i]%p;
6  }

```

5.3 扩展欧几里得

5.3.1 exgcd

```

1 ll exgcd(ll a,ll b,ll &x,ll &y)
2 {
3     if(b==0)
4     {
5         x=1;
6         y=0;
7         return a;
8     }
9     ll g,tmp;
10    g=exgcd(b,a%b,x,y);
11    tmp=x;
12    x=y;
13    y=tmp-a/b*y;
14    return g;
15 }
16 /*
17 xa+yb=gcd(a,b)
18
19 get x,y
20 x1=x+b/gcd(a,b)*t
21 y1=y-a/gcd(a,b)*t
22 t is any integer
23 */

```

5.3.2 ax+by=c

```

1 ll exgcd(ll a,ll b,ll &x,ll &y)
2 {
3     if(b==0)
4     {
5         x=1;
6         y=0;
7         return a;
8     }
9     ll g,tmp;
10    g=exgcd(b,a%b,x,y);
11    tmp=x;
12    x=y;
13    y=tmp-a/b*y;
14    return g;
15 }
16 ll linear_equation(ll a,ll b,ll c,ll &x,ll &y)
17 {
18
19     ll g,dx,dy,t;
20     g=exgcd(a,b,x,y);
21     if(a==0&&b==0)
22     {
23         if(c) return -1;
24         x=y=0;

```

```

25         return g;
26     }
27     if(c%g) return -1; //no solution
28     dx=b/g,dy=-a/g;
29     x*=c/g,y*=c/g;
30     t=(x%dx-x)/dx;
31     if(x+dx*t<=0) t++;
32     x+=dx*t,y+=dy*t;
33     return g;
34 }
35 /*
36 xa+yb=c
37
38 get x,y and x>0 is min
39 have solution: c%gcd(a,b)==0
40 x=x0+dx*t
41 y=y0-dy*t
42 */

```

5.4 中国剩余定理

5.4.1 CRT

```

1 ll qmul(ll a,ll b,ll p)
2 {
3     ll res=0;
4     while(b>0)
5     {
6         if(b&1) res=(res+a)%p;
7         a=(a+a)%p;
8         b>>=1;
9     }
10    return res;
11 }
12 ll exgcd(ll a,ll b,ll &x,ll &y)
13 {
14     if(b==0)
15     {
16         x=1;
17         y=0;
18         return a;
19     }
20     ll g,tmp;
21     g=exgcd(b,a%b,x,y);
22     tmp=x;
23     x=y;
24     y=tmp-a/b*y;
25     return g;
26 }
27 ll inv_exgcd(ll a,ll p)
28 {
29     ll g,x,y;
30     g=exgcd(a,p,x,y);
31     return g==1?(x%p+p)%p:-1;

```



```

32 }
33 ll CRT(int n,ll *a,ll *m)
34 {
35     int i;
36     ll p,mi,res,invmi;
37     p=1;
38     res=0;
39     for(i=1;i<=n;i++) a[i]=(a[i]%m[i]+m[i])%m[i];
40     for(i=1;i<=n;i++) p*=m[i];
41     for(i=1;i<=n;i++)
42     {
43         mi=p/m[i];
44         invmi=inv_exgcd(mi,m[i]);
45         res=(res+qmul(a[i]*mi,invmi,p))%p;
46     }
47     return res;
48 }
49 /*
50 x = a_i (mod m_i) a:1..n, m:1..n
51 m_i must coprime
52
53 CRT return: x = res + k*p, k is interge
54 */

```

5.4.2 exCRT

```

1 ll qmul(ll a,ll b,ll p)
2 {
3     ll res=0;
4     while(b>0)
5     {
6         if(b&1) res=(res+a)%p;
7         a=(a+a)%p;
8         b>>=1;
9     }
10    return res;
11 }
12 ll exgcd(ll a,ll b,ll &x,ll &y)
13 {
14     if(b==0)
15     {
16         x=1;
17         y=0;
18         return a;
19     }
20     ll g,tmp;
21     g=exgcd(b,a%b,x,y);
22     tmp=x;
23     x=y;
24     y=tmp-a/b*y;
25     return g;
26 }
27 ll linear_equation(ll a,ll b,ll c,ll &x,ll &y)
28 {

```

```

29     ll g,dx,dy,t;
30     g=exgcd(a,b,x,y);
31     if(a==0&&b==0)
32     {
33         if(c) return -1;
34         x=y=0;
35         return g;
36     }
37     if(c%g) return -1; //no solution
38     dx=b/g;
39     x=qmul(x,c/g,dx);
40     if(x<0) x+=dx;
41     return g;
42 }
43 ll exCRT(int n,ll *a,ll *m)
44 {
45     {
46         int i;
47         ll prep,p,res,g,x,y;
48         for(i=1;i<=n;i++) a[i]=(a[i]%m[i]+m[i])%m[i];
49         p=m[1];
50         res=a[1];
51         for(i=2;i<=n;i++)
52         {
53             /*
54             m[i-1]*x-m[i]*y=a[i]-a[i-1]
55             m[i-1]=p, a[i-1]=res
56             -> p*x-m[i]*y=a[i]-res
57             */
58             g=linear_equation(p,m[i],(a[i]-res)%m[i]+m[i],x,y);
59             if(g==-1) return -1;
60             prep=p;
61             p=p/g*m[i];
62             res=(res+qmul(prepare,x,p))%p;
63         }
64         return res;
65     }
66     /*
67     x = a_i (mod m_i) a:1..n, m:1..n
68
69     exCRT return: x = res + k*p, k is interge
70     */

```

5.5 欧拉函数

$\leq n$ 且与 n 互质的数的和: $n \cdot \phi(n)/2$

5.5.1 直接求

```

1 //O(sqrt(n))
2 int get_phi(int n)

```

```

3 {
4     int ans,i;
5     ans=n;
6     for(i=2;i*i<=n;i++)
7     {
8         if(n%i==0)
9         {
10             ans=ans-ans/i;
11             while(n%i==0) n/=i;
12         }
13     }
14     if(n>1) ans=ans-ans/n;
15     return ans;
16 }

```

5.5.2 线性筛

```

1 int p[MAX],phi[MAX],tot,prime[MAX];
2 void init_phi(int n)
3 {
4     int i,j;
5     tot=0;
6     memset(prime,0,sizeof prime);
7     prime[1]=phi[1]=1;
8     for(i=2;i<=n;i++)
9     {
10         if(!prime[i])
11         {
12             prime[i]=p[tot++]=i;
13             phi[i]=i-1;
14         }
15         for(j=0;i*p[j]<=n;j++)
16         {
17             prime[i*p[j]]=p[j];
18             if(i%p[j]==0)
19             {
20                 phi[i*p[j]]=phi[i]*p[j];
21                 break;
22             }
23             else phi[i*p[j]]=phi[i]*(p[j]-1);
24         }
25     }
26 }

```

5.6 莫比乌斯函数

```

1 int mo[MAX],prime[MAX],tot;
2 bool flag[MAX];
3 void initmo(int n)
4 {
5     int i,j;
6     mem(flag,0);
7     mem(mo,0);

```

```

8     tot=0;
9     mo[1]=1;
10    for(i=2;i<=n;i++)
11    {
12        if(!flag[i])
13        {
14            prime[tot++]=i;
15            mo[i]=-1;
16        }
17        for(j=0;j<tot&&prime[j]*i<=n;j++)
18        {
19            flag[i*prime[j]]=1;
20            if(i%prime[j]==0)
21            {
22                mo[prime[j]*i]=0;
23                break;
24            }
25            mo[prime[j]*i]=-mo[i];
26        }
27    }
28 }

```

5.7 Berlekamp-Massey

```

1 //Berlekamp-Massey
2 typedef vector<int> VI;
3 namespace linear_seq
4 {
5     #define rep(i,a,n) for (int i=a;i<n;i++)
6     #define SZ(x) ((int)(x).size())
7     const ll mod=1e9+7;
8     ll powmod(ll a,ll b){ll res=1;a%=mod; assert(b
9         >=0); for(;b>=>1){if(b&1)res=res*a%mod;a=
10         *a%mod;}return res;}
11     const int N=10010;
12     ll res[N],base[N],_c[N],_md[N];
13     vector<int> Md;
14     void mul(ll *a,ll *b,int k)
15     {
16         rep(i,0,k+k) _c[i]=0;
17         rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i
18             +j]+a[i]*b[j])%mod;
19         for (int i=k+k-1;i>=k;i--) if (_c[i])
20             rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[
21                 j]]-_c[i]*_md[Md[j]])%mod;
22         rep(i,0,k) a[i]=_c[i];
23     }
24     int solve(ll n,VI a,VI b){
25         ll ans=0,pnt=0;
26         int k=SZ(a);
27         assert(SZ(a)==SZ(b));
28         rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
29         Md.clear();

```

```

26 rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
27 rep(i,0,k) res[i]=base[i]=0;
28 res[0]=1;
29 while ((1ll<<pnt)<=n) pnt++;
30 for (int p=pnt;p>=0;p--) {
31     mul(res,res,k);
32     if ((n>>p)&1) {
33         for (int i=k-1;i>=0;i--) res[i+1]=res[
34             i];res[0]=0;
35         rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]
36             ]-res[k]*_md[Md[j]])%mod;
37     }
38     rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
39     if (ans<0) ans+=mod;
40     return ans;
41 }
42 VI BM(VI s){
43     VI C(1,1),B(1,1);
44     int L=0,m=1,b=1;
45     rep(n,0,SZ(s)){
46         ll d=0;
47         rep(i,0,L+1) d=(d+(1ll)C[i]*s[n-i])%mod;
48         if(d==0) ++m;
49         else if(2*L<=n){
50             VI T=C;
51             ll c=mod-d*powmod(b,mod-2)%mod;
52             while (SZ(C)<SZ(B)+m) C.pb(0);
53             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%
54                 mod;
55             L=n+1-L; B=T; b=d; m=1;
56         } else {
57             ll c=mod-d*powmod(b,mod-2)%mod;
58             while (SZ(C)<SZ(B)+m) C.pb(0);
59             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%
60                 mod;
61             ++m;
62         }
63     }
64     return C;
65 }
66 int gao(VI a,ll n)
67 {
68     VI c=BM(a);
69     c.erase(c.begin());
70     rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
71     return solve(n,c,VI(a.begin(),a.begin()+SZ(c)
72         ));
73 }
74 };//linear_seq::gao(VI{},n-1)

```

5.8 exBSGS

```

1 //a^x=b (mod c)
2 ll exBSGS(ll a,ll b,ll c)
3 {
4     ll i,g,d,num,now,sq,t,x,y;
5     if(c==1) return b?-1:(a!=1);
6     if(b==1) return a?0:-1;
7     if(a%c==0) return b?-1:1;
8     num=0;
9     d=1;
10    while((__gcd(a,c))>1)
11    {
12        if(b%g) return -1;
13        num++;
14        b/=g;
15        c/=g;
16        d=(d*a/g)%c;
17        if(d==b) return num;
18    }
19    mp.clear();
20    sq=ceil(sqrt(c));
21    t=1;
22    for(i=0;i<sq;i++)
23    {
24        if(!mp.count(t)) mp[t]=i;
25        else mp[t]=min(mp[t],i);
26        t=t*a%c;
27    }
28    for(i=0;i<sq;i++)
29    {
30        exgcd(d,c,x,y);
31        x=(x*b%c+c)%c;
32        if(mp.count(x)) return i*sq+mp[x]+num;
33        d=d*t%c;
34    }
35    return -1;
36 }

```

5.9 Miller_Rabin+Pollard_rho

```

1 struct Montgomery
2 {
3     #define i64 int64_t
4     #define u64 uint64_t
5     #define u128 __uint128_t
6     u64 mod,R;
7     void set_mod(u64 p)
8     {
9         mod=R=p;
10        for(int i=0;i<5;i++) R*=2-mod*R;
11    }
12    u64 mul_add(u64 a,u64 b,u64 c)
13    {
14        u128 d=u128(a)*b;

```

```

15     u64 e=c+mod+(d>>64);
16     u64 f=u64(d)*R;
17     u64 g=(u128(f)*mod)>>64;
18     return e-g;
19 }
20 u64 mul(u64 a,u64 b) {return mul_add(a,b,0);}
21 #undef i64
22 #undef u64
23 #undef u128
24 };
25 // m.set_mod(p);
26 ll qpow(ll a,ll b,ll p)
27 {
28     ll res=1;
29     while(b>0)
30     {
31         if(b&1) res=(__int128)res*a%p;
32         a=(__int128)a*a%p;
33         b>>=1;
34     }
35     return res;
36 }
37 bool miller_rabin(ll n,const initializer_list<ll>&
    as)
38 {
39     int e=__builtin_ctzll(n-1);
40     return all_of(as.begin(),as.end(),[&](ll it)
41     {
42         if(n<=it) return true;
43         ll z=qpow(it,(n-1)>>e,n); // it^(t*2^e)
44         if(z==1||z==n-1) return true;
45         for(int i=0;i<e-1;i++)
46         {
47             z=__int128(z)*z%n;
48             if(z==1) return false;
49             if(z==n-1) return true;
50         }
51         return false;
52     });
53 }
54 bool is_prime(ll n)
55 {
56     if(n<=2) return n==2;
57     if(!(n&1)) return false;
58     if(n<4759123141LL) return miller_rabin(
        {2,7,61});
59     return miller_rabin(n
        {2,325,9375,28178,450775,9780504,1795265022});
60 }
61 ll pollard_rho(ll n)
62 {
63     Montgomery m;
64     m.set_mod(n);

```

```

65     if(!(n&1)) return 2;
66     int i,j,k;
67     ll q1,q2,g1,g2,w1,w2,x1,x2,y1,y2,z1,z2,g,c,x,z;
68     constexpr ll c1=1;
69     constexpr ll c2=2;
70     constexpr ll M=512;
71     w1=1,w2=2;
72     retry:
73     z1=w1,z2=w2;
74     for(i=M;;i<=1)
75     {
76         x1=z1+n,x2=z2+n;
77         for(j=0;j<i;j+=M)
78         {
79             y1=z1,y2=z2;
80             q1=1,q2=2;
81             for(k=0;k<M;k++)
82             {
83                 z1=m.mul_add(z1,z1,c1);
84                 z2=m.mul_add(z2,z2,c2);
85                 q1=m.mul(q1,(x1-z1));
86                 q2=m.mul(q2,(x2-z2));
87             }
88             g=__gcd((ll)m.mul(q1,q2),n);
89             if(g==1) continue;
90             if(g!=n) return g;
91             g1=__gcd(q1,n);
92             g2=__gcd(q2,n);
93             if(g1!=1) c=c1,x=x1,z=y1,g=g1;
94             else c=c2,x=x2,z=y2,g=g2;
95             if(g==n)
96             {
97                 do{
98                     z=m.mul_add(z,z,c);
99                     g=__gcd(n,x-z);
100                 }while(g==1);
101             }
102             if(g!=n) return g;
103             w1+=2,w2+=2;
104             goto retry;
105         }
106     }
107 }
108 void find_fac(ll n,vector<ll>& res)
109 {
110     if(n<=1) return;
111     if(is_prime(n)) {res.push_back(n); return;}
112     ll p=pollard_rho(n);
113     find_fac(p,res);
114     find_fac(n/p,res);
115 }
116 vector<ll> factorize(ll n)
117 {
118     vector<ll> res;

```

```

119     find_fac(n,res);
120     return res;
121 }

```

5.10 第二类 Stirling 数

```

1 //dp[i][j]表示j个元素划分到i个不可区分的非空盒子里的方案
  数。ik
2 ll dp[MAX][MAX];
3 void init()
4 {
5     ll i,j;
6     mem(dp,0);
7     dp[1][1]=1;
8     for(i=2;i<MAX;i++)
9     {
10         for(j=1;j<=i;j++)
11         {
12             dp[i][j]=(dp[i-1][j-1]+j*dp[i-1][j])%mod;
13         }
14     }
15 }

```

5.11 原根

原根性质

1. 一个数 m 如果有原根，则其原根个数为 $\phi[\phi[m]]$ 。
若 m 为素数，则其原根个数为 $\phi[\phi[m]] = \phi[m-1]$ 。
2. 有原根的数只有 $2, 4, p^n, 2 * p^n$ (p 为质数, n 为正整数)
3. 一个数的最小原根的大小是 $O(n^{0.25})$ 的
4. 如果 g 为 n 的原根，则 g^d 为 n 的原根的充要条件是 $\gcd(d, \phi[n]) = 1$

指标法则

1. $I(a * b) = I(a) + I(b) \pmod{p-1}$
2. $I(a^k) = k * I(a) \pmod{p-1}$

```

1 int p[MAX],tot,prime[MAX];
2 void init(int n)
3 {
4     int i,j;
5     tot=0;
6     mem(prime,0);
7     prime[1]=1;
8     for(i=2;i<=n;i++)
9     {
10         if(!prime[i]) prime[i]=p[tot++]=i;
11         for(j=0;j<tot&&p[j]*i<=n;j++)
12         {
13             prime[i*p[j]]=p[j];

```

```

14             if(i%p[j]==0) break;
15         }
16     }
17 }
18 ll pow2(ll a,ll b,ll p)
19 {
20     ll res=1;
21     while(b)
22     {
23         if(b&1) res=res*a%p;
24         a=a*a%p;
25         b>>=1;
26     }
27     return res;
28 }
29 int tp[MAX];
30 int find_root(int x)//求素数原根
31 {
32     if(x==2) return 1;
33     int f,phi=x-1;
34     tp[0]=0;
35     for(int i=0;phi&&i<tot;i++)
36     {
37         if(phi%p[i]==0)
38         {
39             tp[++tp[0]]=p[i];
40             while(phi%p[i]==0) phi/=p[i];
41         }
42     }
43     if(phi!=1) tp[++tp[0]]=phi;
44     phi=x-1;
45     for(int g=2;g<=x-1;g++)
46     {
47         f=1;
48         for(int i=1;i<=tp[0];i++)
49         {
50             if(pow2(g,phi/tp[i],x)==1)
51             {
52                 f=0;
53                 break;
54             }
55         }
56         if(f) return g;
57     }
58     return 0;
59 }
60 int I[MAX];
61 void get_I(int p)//求指标表
62 {
63     int g,now;
64     g=find_root(p);
65     now=1;
66     for(int i=1;i<p;i++)
67     {

```

```

68     now=now*g%p;
69     I[now]=i;
70 }
71 }

```

5.12 二次剩余

```

1 //O(log^2)
2 struct Tonelli_Shanks
3 {
4     ll mul2(ll a,ll b,ll p)
5     {
6         ll res=0;
7         a%=p;
8         while(b)
9         {
10             if(b&1)
11             {
12                 res+=a;
13                 if(res>=p) res-=p;
14             }
15             a+=a;
16             if(a>=p) a-=p;
17             b>>=1;
18         }
19         return res;
20     }
21     ll pow2(ll a,ll b,ll p)
22     {
23         ll res=1;
24         while(b)
25         {
26             if(b&1) res=mul2(res,a,p);
27             a=mul2(a,a,p);
28             b>>=1;
29         }
30         return res;
31     }
32     ll sqrt(ll n,ll p)
33     {
34         if(p==2) return (n&1)?1:-1;
35         if(pow2(n,p>>1,p)!=1) return -1;
36         if(p&2) return pow2(n,(p+1)>>2,p);
37         ll q,z,c,r,t,tmp,s,i,m;
38         s=__builtin_ctzll(p^1);
39         q=p>>s;
40         z=2;
41         for(;pow2(z,p>>1,p)==1;z++);
42         c=pow2(z,q,p);
43         r=pow2(n,(q+1)>>1,p);
44         t=pow2(n,q,p);
45         for(m=s;t!=1;)
46         {

```

```

47             for(i=0,tmp=t;tmp!=1;i++) tmp=tmp*tmp%p;
48             for(;i<--m;) c=c*c%p;
49             r=r*c%p;
50             c=c*c%p;
51             t=t*c%p;
52         }
53         return r;
54     }
55 }ts;

```

6 组合数学

6.1 组合数

6.1.1 公式预处理

```

1 ll qpow(ll a,ll b)
2 {
3     ll res=1;
4     while(b>0)
5     {
6         if(b&1) res=res*a%mod;
7         a=a*a%mod;
8         b>>=1;
9     }
10    return res;
11 }
12 ll inv(ll x){return qpow(x,mod-2);}
13 int fac[MAX],invfac[MAX];
14 void init_comb(int n)
15 {
16     assert(n<MAX);
17     fac[0]=1;
18     for(int i=1;i<=n;i++) fac[i]=1ll*fac[i-1]*i%mod;
19     invfac[n]=inv(fac[n]);
20     for(int i=n-1;~i;i--) invfac[i]=1ll*invfac[i+1]*(i+1)%mod;
21 }
22 ll C(int n,int m)
23 {
24     if(m>n||m<0||n<0) return 0;
25     return 1ll*fac[n]*invfac[m]%mod*invfac[n-m]%mod;
26 }
27 ll A(int n,int m)
28 {
29     if(m>n||m<0||n<0) return 0;
30     return 1ll*fac[n]*invfac[n-m]%mod;
31 }

```

6.1.2 杨辉三角递推

```

1 int comb[MAX][105];
2 void init_comb(int n,int m)

```

```

3 {
4     int i,j;
5     comb[0][0]=1;
6     for(i=1;i<=n;i++)
7     {
8         comb[i][0]=1;
9         for(j=1;j<=i;j++)
10        {
11            comb[i][j]=comb[i-1][j]+comb[i-1][j-1];
12            if(comb[i][j]>=mod) comb[i][j]-=mod;
13        }
14    }
15 }
16 ll C(int n,int m)
17 {
18     if(m>n||m<0||n<0) return 0;
19     return comb[n][m];
20 }

```

6.2 卢卡斯定理

6.2.1 Lucas

```

1 ll Lucas(ll n,ll m,int p)
2 {
3     if(m==0) return 1;
4     return C(n%p,m%p)*Lucas(n/p,m/p,p)%p;
5 }
6 /*
7 p must be a prime number
8 init(p-1);
9 */

```

6.2.2 exLucas

```

1 namespace exLucas
2 {
3     ll qpow(ll a,ll b,ll p)
4     {
5         ll res=1;
6         while(b>0)
7         {
8             if(b&1) res=res*a%p;
9             a=a*a%p;
10            b>>=1;
11        }
12        return res;
13    }
14    ll qmul(ll a,ll b,ll p)
15    {
16        ll res=0;
17        while(b>0)
18        {

```

```

19            if(b&1) res=(res+a)%p;
20            a=(a+a)%p;
21            b>>=1;
22        }
23        return res;
24    }
25    ll exgcd(ll a,ll b,ll &x,ll &y)
26    {
27        if(b==0)
28        {
29            x=1;
30            y=0;
31            return a;
32        }
33        ll g,tmp;
34        g=exgcd(b,a%p,x,y);
35        tmp=x;
36        x=y;
37        y=tmp-a/b*y;
38        return g;
39    }
40    ll inv(ll a,ll p)
41    {
42        ll g,x,y,res;
43        g=exgcd(a,p,x,y);
44        res=(g==1?(x+p)%p:-1);
45        assert(res!=-1);
46        return res;
47    }
48    ll CRT(vector<ll> a,vector<ll> m)
49    {
50        int i,n;
51        ll p,mi,res,invmi;
52        n=a.size();
53        p=1;
54        res=0;
55        for(i=0;i<n;i++) a[i]=(a[i]%m[i]+m[i])%m[i];
56        for(i=0;i<n;i++) p*=m[i];
57        for(i=0;i<n;i++)
58        {
59            mi=p/m[i];
60            invmi=inv(mi,m[i]);
61            res=(res+qmul(a[i]*mi,invmi,p))%p;
62        }
63        return res;
64    }
65    map<ll,pair<vector<ll>,vector<ll>>> mp;
66    map<pair<ll,ll>,vector<ll>> fac;
67    void init(vector<ll> mod_list)
68    {
69        ll i,j,p;
70        mp.clear();
71        fac.clear();
72        for(auto &mod_i:mod_list)

```

```

73     {
74         p=mod_i;
75         vector<ll> a,b;
76         for(i=2;i*i<=p;i++)
77         {
78             if(p%i) continue;
79             b.push_back(1LL);
80             while(p%i==0) b[b.size()-1]*=i,p/=i;
81             a.push_back(i);
82         }
83         if(p>1) a.push_back(p),b.push_back(p);
84         mp[mod_i]={a,b};
85         for(i=0;i<a.size();i++)
86         {
87             if(fac.count({a[i],b[i]})) continue;
88             vector<ll> fac_tmp(b[i]+1);
89             fac_tmp[0]=1;
90             for(j=1;j<=b[i];j++)
91             {
92                 if(j%a[i]) fac_tmp[j]=fac_tmp[j-1]*
93                     j%b[i];
94                 else fac_tmp[j]=fac_tmp[j-1];
95             }
96             fac[{a[i],b[i]}]=fac_tmp;
97         }
98     }
99 ll cal_fac(ll n,ll x,ll p)
100 {
101     if(!n) return 1LL;
102     ll res=1;
103     assert(fac.count({x,p}));
104     res=res*fac[{x,p}][p-1]%p;
105     res=qpow(res,n/p,p);
106     res=res*fac[{x,p}][n%p]%p;
107     return res*cal_fac(n/x,x,p)%p;
108 }
109 ll comb(ll n,ll m,ll x,ll p)
110 {
111     if(m>n) return 0;
112     ll i,cnt;
113     cnt=0;
114     for(i=n;i/=x) cnt+=i/x;
115     for(i=m;i/=x) cnt-=i/x;
116     for(i=n-m;i/=x) cnt-=i/x;
117     return qpow(x,cnt,p)* \
118         cal_fac(n,x,p)%p* \
119         inv(cal_fac(m,x,p),p)%p* \
120         inv(cal_fac(n-m,x,p),p)%p;
121 }
122 ll C(ll n,ll m,ll p)
123 {
124     if(m>n||m<0||n<0) return 0;
125     ll i,res;

```

```

126     vector<ll> a,b,resa;
127     assert(mp.count(p));
128     a=mp[p].first;
129     b=mp[p].second;
130     for(i=0;i<a.size();i++) resa.push_back(comb(n
131         ,m,a[i],b[i]));
132     res=CRT(resa,b);
133     assert(res!=-1);
134     return res;
135 }
136 /*
137 mod=p1^k1 * p2^k2 * ...
138 init: O(sqrt(mod)+sum{p^k})
139 C(n,m,mod): O(log)
140
141 exLucas::init({mod}); //init mod list
142 exLucas::C(n,m,mod);
143 */

```

6.3 卡特兰数公式

```
1 C(2*n,n)-C(2*n,n-1)
```

7 多项式

7.1 FFT

```

1 namespace FFT
2 {
3     #define rep(i,a,b) for(int i=(a);i<=(b);i++)
4     const double pi=acos(-1);
5     const int maxn=(1<<19)+10;
6     struct cp
7     {
8         double a,b;
9         cp(){}
10        cp(double _x,double _y){a=_x,b=_y;}
11        cp operator +(const cp &o)const{return (cp){a
12            +o.a,b+o.b};}
13        cp operator -(const cp &o)const{return (cp){a
14            -o.a,b-o.b};}
15        cp operator *(const cp &o)const{return (cp){a
16            *o.a-b*o.b,b*o.a+a*o.b};}
17        cp operator *(const double &o)const{return (
18            cp){a*o,b*o};}
19        cp operator !(const cp &o){return (cp){a,-b};}
20    }x[maxn],y[maxn],z[maxn],w[maxn];
21    void fft(cp x[],int k,int v)
22    {
23        int i,j,l;
24        for(i=0,j=0;i<k;i++)

```



```

21 {
22     if(i>j)swap(x[i],x[j]);
23     for(l=k>>1;(j^=1)<l;l>>=1);
24 }
25 w[0]=(cp){1,0};
26 for(i=2;i<=k;i<=1)
27 {
28     cp g=(cp){cos(2*pi/i),(v?-1:1)*sin(2*pi/i
29         )});
30     for(j=(i>>1);j>=0;j-=2)w[j]=w[j>>1];
31     for(j=1;j<i>>1;j+=2)w[j]=w[j-1]*g;
32     for(j=0;j<k;j+=i)
33     {
34         cp *a=x+j,*b=a+(i>>1);
35         for(l=0;l<i>>1;l++)
36         {
37             cp o=b[l]*w[l];
38             b[l]=a[l]-o;
39             a[l]=a[l]+o;
40         }
41     }
42     if(v)for(i=0;i<k;i++)x[i]=(cp){x[i].a/k,x[i].
43         b/k};
44 }
45 // a=b*c
46 // b[0..11]
47 // c[0..12]
48 void mul(int *a,int *b,int *c,int l1,int l2)
49 {
50     if(l1<128&&l2<128)
51     {
52         rep(i,0,l1+l2)a[i]=0;
53         rep(i,0,l1)rep(j,0,l2)a[i+j]+=b[i]*c[j];
54         return;
55     }
56     int K;
57     for(K=1;K<=l1+l2;K<=1);
58     rep(i,0,l1)x[i]=cp(b[i],0);
59     rep(i,0,l2)y[i]=cp(c[i],0);
60     rep(i,l1+1,K)x[i]=cp(0,0);
61     rep(i,l2+1,K)y[i]=cp(0,0);
62     fft(x,K,0);fft(y,K,0);
63     rep(i,0,K)z[i]=x[i]*y[i];
64     fft(z,K,1);
65     rep(i,0,l1+l2)a[i]=(l1)(z[i].a+0.5);
66 }
67 };

```

7.2 NTT

```

2 {
3     const int g=3;
4     const int p=998244353;
5     int wn[35];
6     int qpow(int a,int b)
7     {
8         int res=1;
9         while(b>0)
10         {
11             if(b&1) res=1ll*res*a%p;
12             a=1ll*a*a%p;
13             b>>=1;
14         }
15         return res;
16     }
17     void getwn()
18     {
19         assert(p==mod);
20         for(int i=0;i<25;i++) wn[i]=qpow(g,(p-1)/(1ll
21             <<i));
22     }
23     void ntt(vector<int> &a,int len,int f)
24     {
25         int i,j=0,t,k,w,id;
26         for(i=1;i<len-1;i++)
27         {
28             for(t=len;j^=t>>=1,~j&t);
29             if(i<j) swap(a[i],a[j]);
30         }
31         for(i=1,id=1;i<len;i<=1,id++)
32         {
33             t=i<<1;
34             for(j=0;j<len;j+=t)
35             {
36                 for(k=0,w=1;k<i;k++,w=1ll*w*wn[id]%p)
37                 {
38                     int x=a[j+k],y=1ll*w*a[j+k+i]%p;
39                     a[j+k]=x+y;
40                     if(a[j+k]>=p) a[j+k]-=p;
41                     a[j+k+i]=x-y;
42                     if(a[j+k+i]<0) a[j+k+i]+=p;
43                 }
44             }
45         }
46         if(f)
47         {
48             for(i=1,j=len-1;i<j;i++,j--) swap(a[i],a[
49                 j]);
50             int inv=qpow(len,p-2);
51             for(i=0;i<len;i++) a[i]=1ll*a[i]*inv%p;
52         }
53     }
54     vector<int> qpow(vector<int> a,int b)//limt: sz(
55         a)*b is small

```

```
namespace NTT
```

```

53 {
54     int len,i,l1;
55     l1=a.size();
56     for(len=1;len<((l1+1)*b-1);len<=1);
57     a.resize(len,0);
58     ntt(a,len,0);
59     vector<int> res(len);
60     for(i=0;i<len;i++) res[i]=pow2(a[i],b);
61     ntt(res,len,1);
62     res.resize((l1+1)*b-1);
63     return res;
64 }
65 vector<int> mul(vector<int> a,vector<int> b)
66 {
67     int len,i,l1,l2;
68     l1=a.size();
69     l2=b.size();
70     for(len=1;len<l1+l2;len<=1);
71     a.resize(len,0);
72     b.resize(len,0);
73     ntt(a,len,0);ntt(b,len,0);
74     vector<int> res(len);
75     for(i=0;i<len;i++) res[i]=l1l*a[i]*b[i]%p;
76     ntt(res,len,1);
77     res.resize(l1+l2-1);
78     return res;
79 }
80
81 //get kth
82 vector<int> merge_generating_functions(vector<
83     vector<int>> &dp,int k)
84 {
85     int i,j;
86     priority_queue<pair<int,int>> q;
87     for(i=0;i<dp.size();i++) q.push({-dp[i].size
88         (),i});
89     while(q.size()>1)
90     {
91         i=q.top().second;
92         q.pop();
93         j=q.top().second;
94         q.pop();
95         dp[i]=mul(dp[i],dp[j]);
96         if(dp[i].size()>k) dp[i].resize(k+1);
97         q.push({-dp[i].size(),i});
98     }
99     return dp[q.top().second];
100 }
101 };//NTT::getwn();

```

7.3 FWT

```
namespace FWT
```

```

2 {
3     const int p=998244353;
4     const ll inv2=(p+1)/2;
5     void fwt(vector<int> &a,int n,int f,int v)
6     {
7         for(int d=1;d<n;d<=1)
8         {
9             for(int m=d<<1,i=0;i<n;i+=m)
10             {
11                 for(int j=0;j<d;j++)
12                 {
13                     ll x=a[i+j],y=a[i+j+d];
14                     if(v==1)
15                     {
16                         if(f==1) a[i+j]=(x+y)%p,a[i+j+d
17                             ]=(x-y+p)%p;//xor
18                         else if(f==2) a[i+j]=(x+y)%p;//
19                             and
20                         else if(f==3) a[i+j+d]=(x+y)%p;
21                             //or
22                     }
23                     else
24                     {
25                         if(f==1) a[i+j]=(x+y)*inv2%p,a[
26                             i+j+d]=(x-y+p)%p*inv2%p;//
27                             xor
28                         else if(f==2) a[i+j]=(x-y+p)%p;
29                             //and
30                         else if(f==3) a[i+j+d]=(y-x+p)%
31                             p;//or
32                     }
33                 }
34             }
35         }
36     }
37
38     vector<int> XOR(vector<int> a,vector<int> b)
39     {
40         int n,len;
41         n=a.size();
42         for(len=1;len<n;len<=1);
43         a.resize(len,0);
44         b.resize(len,0);
45         fwt(a,len,1,1);
46         fwt(b,len,1,1);
47         for(int i=0;i<len;i++) a[i]=l1l*a[i]*b[i]%p;
48         fwt(a,len,1,-1);
49         return a;
50     }
51
52     vector<int> AND(vector<int> a,vector<int> b)
53     {
54         int n,len;
55         n=a.size();
56         for(len=1;len<n;len<=1);

```

```

49     a.resize(len,0);
50     b.resize(len,0);
51     fwt(a,len,2,1);
52     fwt(b,len,2,1);
53     for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
54     fwt(a,len,2,-1);
55     return a;
56 }
57 vector<int> OR(vector<int> a,vector<int> b)
58 {
59     int n,len;
60     n=a.size();
61     for(len=1;len<n;len<=<=1);
62     a.resize(len,0);
63     b.resize(len,0);
64     fwt(a,len,3,1);
65     fwt(b,len,3,1);
66     for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
67     fwt(a,len,3,-1);
68     return a;
69 }
70 }

```

7.4 拉格朗日插值

```

1 namespace polysum {
2     #define rep(i,a,n) for (int i=a;i<n;i++)
3     #define per(i,a,n) for (int i=n-1;i>=a;i--)
4     const int D=101000;
5     ll a[D],tmp[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h
        [D][2],C[D];
6     ll powmod(ll a,ll b){ll res=1;a%=mod;assert(b
        >=0);for(;b>=1){if(b&1)res=res*a%mod;a=a*
        a%mod;}return res;}
7     ll calcn(int d,ll *a,ll n) { // a[0].. a[d] a[n]
8         if (n<=d) return a[n];
9         p1[0]=p2[0]=1;
10        rep(i,0,d+1) {
11            ll t=(n-i+mod)%mod;
12            p1[i+1]=p1[i]*t%mod;
13        }
14        rep(i,0,d+1) {
15            ll t=(n-d+i+mod)%mod;
16            p2[i+1]=p2[i]*t%mod;
17        }
18        ll ans=0;
19        rep(i,0,d+1) {
20            ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%
                mod*a[i]%mod;
21            if ((d-i)&1) ans=(ans-t+mod)%mod;
22            else ans=(ans+t)%mod;
23        }
24        return ans;

```

```

25     }
26     void init(int M) {
27         f[0]=f[1]=g[0]=g[1]=1;
28         rep(i,2,M+5) f[i]=f[i-1]*i%mod;
29         g[M+4]=powmod(f[M+4],mod-2);
30         per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
31     }
32     ll polysum(ll n,ll *a,ll m) { // a[0].. a[m] \
        sum_{i=0}^{n-1} a[i]
33         rep(i,0,m+1) tmp[i]=a[i];
34         tmp[m+1]=calcn(m,tmp,m+1);
35         rep(i,1,m+2) tmp[i]=(tmp[i-1]+tmp[i])%mod;
36         return calcn(m+1,tmp,n-1);
37     }
38     ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[
        m] \sum_{i=0}^{n-1} a[i]*R^i
39         if (R==1) return polysum(n,a,m);
40         a[m+1]=calcn(m,a,m+1);
41         ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
42         h[0][0]=0;h[0][1]=1;
43         rep(i,1,m+2) {
44             h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
45             h[i][1]=h[i-1][1]*r%mod;
46         }
47         rep(i,0,m+2) {
48             ll t=g[i]*g[m+1-i]%mod;
49             if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,
                p4=((p4-h[i][1]*t)%mod+mod)%mod;
50             else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i
                ][1]*t)%mod;
51         }
52         c=powmod(p4,mod-2)*(mod-p3)%mod;
53         rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
54         rep(i,0,m+2) C[i]=h[i][0];
55         ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
56         if (ans<0) ans+=mod;
57         return ans;
58     }
59 } // polysum::init();

```

8 矩阵

8.1 矩阵类

```

1 const ll mod=1e9+7;
2 struct Matrix
3 {
4     static const int N=;
5     int n;
6     ll c[N][N];
7     Matrix(){}
8     Matrix(int _n,ll v=0)
9     {

```

```

10     int i,j;
11     n=_n;
12     for(i=0;i<n;i++)
13     {
14         for(j=0;j<n;j++)
15         {
16             c[i][j]=v;
17         }
18     }
19 }
20 void init_identity_matrix() {for(int i=0;i<n;i
    ++) c[i][i]=1;}
21 Matrix operator *(const Matrix &b)const
22 {
23     int i,j,k;
24     Matrix res(n);
25     for(k=0;k<n;k++)
26     {
27         for(i=0;i<n;i++)
28         {
29             if(!c[i][k]) continue;
30             for(j=0;j<n;j++)
31             {
32                 res.c[i][j]+=c[i][k]*b.c[k][j];
33                 if(res.c[i][j]>=mod) res.c[i][j]%=
                    mod;
34             }
35         }
36     }
37     return res;
38 }
39 };
40 Matrix matqpow(Matrix a,ll b)
41 {
42     Matrix res(a.n);
43     res.init_identity_matrix();
44     while(b)
45     {
46         if(b&1) res=res*a;
47         a=a*a;
48         b>>=1;
49     }
50     return res;
51 }

```

8.2 高斯消元

8.2.1 浮点数方程组

```

1 struct Gauss
2 {
3     const double eps=1e-7;
4     double mp[][];
5     int gauss(int n,int m)

```

```

6     {
7         int i,j,k,pos,r;
8         double tmp;
9         r=0;
10        for(k=1;k<=m;k++)
11        {
12            pos=r+1;
13            if(pos>n) return -1; // no solution
14            for(i=pos+1;i<=n;i++)
15            {
16                if(fabs(mp[i][k])>fabs(mp[pos][k]))
17                    pos=i;
18            }
19            if(fabs(mp[pos][k])<eps) continue;
20            r++;
21            swap(mp[pos],mp[r]);
22            tmp=mp[r][k];
23            for(j=k;j<=m+1;j++) mp[r][j]/=tmp;
24            for(i=r+1;i<=n;i++)
25            {
26                tmp=mp[i][k];
27                for(j=k;j<=m+1;j++)
28                {
29                    mp[i][j]-=mp[r][j]*tmp;
30                }
31            }
32            return r;
33        }
34        int work(int n,int m,double *res)
35        {
36            int i,j,cnt;
37            cnt=gauss(n,m);
38            if(cnt==-1) return -1;
39            for(i=cnt+1;i<=n;i++)
40            {
41                if(fabs(mp[i][m+1])>eps)
42                {
43                    // no solution
44                    return -1;
45                }
46            }
47            if(cnt<m)
48            {
49                // multi solution
50                return 1;
51            }
52            res[m]=mp[m][m+1];
53            for(i=m-1;i>=1;i--)
54            {
55                res[i]=mp[i][m+1];
56                for(j=i+1;j<=m;j++)
57                {
58                    res[i]-=mp[i][j]*res[j];

```

```

59     }
60 }
61 return 0;
62 }
63 }gs;
64 /*
65 (mp[1][1]*x1) + (mp[1][2]*x2) + ... + (mp[1][m]*xm)
66   = mp[1][m+1]
67 (mp[2][1]*x1) + (mp[1][2]*x2) + ... + (mp[2][m]*xm)
68   = mp[2][m+1]
69 ...
70 (mp[n][1]*x1) + (mp[n][2]*x2) + ... + (mp[n][m]*xm)
71   = mp[n][m+1]
72
73 x x x x | x
74 0 x x x | x
75 0 0 x x | x
76 0 0 0 x | x
77
78 O(n*m^2) m<=n
79
80 gs.work(n,m,res); mp[1..n][1..m+1], res[1..m]
81
82 -1: no solution
83 0 : one solution
84 1 : multi solution
85 */

```

8.2.2 异或方程组

```

1 struct Gauss
2 {
3     bitset<> mp[];
4     int gauss_jordan(int n,int m)
5     {
6         int i,j,k,pos,r;
7         r=0;
8         for(k=1;k<=m;k++)
9         {
10             pos=r+1;
11             if(pos>n) return -1; // no solution
12             while(pos<n&&!mp[pos][k]) pos++;
13             if(!mp[pos][k]) continue;
14             r++;
15             swap(mp[pos],mp[r]);
16             for(i=1;i<=n;i++)
17             {
18                 if(i!=r&&mp[i][k]) mp[i]^=mp[r];
19             }
20         }
21         return r;
22     }
23     int work(int n,int m,int *res)
24     {

```

```

25     int i,j,cnt;
26     cnt=gauss_jordan(n,m);
27     if(cnt==-1) return -1;
28     for(i=cnt+1;i<=n;i++)
29     {
30         if(mp[i][m+1])
31         {
32             // no solution
33             return -1;
34         }
35     }
36     if(cnt<m)
37     {
38         // multi solution
39         return 1;
40     }
41     for(i=1;i<=m;i++) res[i]=mp[i][m+1];
42     return 0;
43 }
44 }gs;
45 /*
46 (mp[1][1]*x1) xor (mp[1][2]*x2) xor ... xor (mp[1][
47   m]*xm) = mp[1][m+1]
48 (mp[2][1]*x1) xor (mp[1][2]*x2) xor ... xor (mp[2][
49   m]*xm) = mp[2][m+1]
50 ...
51 (mp[n][1]*x1) xor (mp[n][2]*x2) xor ... xor (mp[n][
52   m]*xm) = mp[n][m+1]
53
54 a:0/1 x:0/1
55
56 x 0 0 0 | x
57 0 x 0 0 | x
58 0 0 x 0 | x
59 0 0 0 x | x
60
61 O((n*m^2)/64)
62
63 gs.work(n,m,res); mp[1..n][1..m+1], res[1..m]
64
65 -1: no solution
66 0 : one solution
67 1 : multi solution
68 */

```

8.2.3 同余方程组

```

1 namespace Gauss
2 {
3     int p;
4     ll mp[905][905],sol[905];
5     void set_mod(int _p)
6     {

```

```

7      p=_p;
8      mem(mp,0);
9      mem(sol,0);
10     }
11     ll pow2(ll a,ll b)
12     {
13         ll res=1;
14         while(b)
15         {
16             if(b&1) res=res*a%p;
17             a=a*a%p;
18             b>>=1;
19         }
20         return res;
21     }
22     ll inv(ll x){return pow2(x,p-2);}
23     ll lcm(ll a,ll b){return a/__gcd(a,b)*b;}
24     int gauss(int n,int m)
25     {
26         int r,c,id,i,j;
27         ll tmp,ta,tb;
28         r=c=0;
29         while(r<n&&c<m)
30         {
31             id=r;
32             for(i=r+1;i<n;i++)
33             {
34                 if(abs(mp[i][c])>abs(mp[id][c])) id=i;
35             }
36             if(id!=r)
37             {
38                 for(i=0;i<m;i++) swap(mp[r][i],mp[id][i]);
39             }
40             if(abs(mp[r][c])!=0)
41             {
42                 for(i=r+1;i<n;i++)
43                 {
44                     if(abs(mp[i][c])==0) continue;
45                     tmp=lcm(abs(mp[i][c]),abs(mp[r][c]))
46                         );
47                     ta=tmp/abs(mp[i][c]);
48                     tb=tmp/abs(mp[r][c]);
49                     if(mp[i][c]*mp[r][c]<0) tb=-tb;
50                     for(j=c;j<m;j++)
51                     {
52                         mp[i][j]=(mp[i][j]*ta-mp[r][j]*
53                             tb)%p;
54                         if(mp[i][j]<0) mp[i][j]+=p;
55                     }
56                 }
57                 r++;
58             }
59         }
60         return r;
61     }

```

```

58     }
59     for(i=r;i<n;i++)
60     {
61         if(mp[i][m]!=0) return -1;//no solution
62     }
63     // if(r<m) return m-r;//multi solution
64     for(i=m-1;~i;i--)
65     {
66         tmp=mp[i][m];
67         for(j=i+1;j<m;j++)
68         {
69             if(mp[i][j]==0) continue;
70             tmp=(tmp-mp[i][j]*sol[j])%p;
71             if(tmp<0) tmp+=p;
72         }
73         sol[i]=tmp*inv(mp[i][i])%p;
74     }
75     return 0;
76 }
77 }
78 using namespace Gauss;
79 //set_mod();

```

8.3 线性基

8.3.1 线性基

```

1 struct Base
2 {
3     #define type ll
4     #define mx 60
5     type d[mx+3];
6     void init()
7     {
8         memset(d,0,sizeof(d));
9     }
10    bool insert(type x)
11    {
12        for(int i=mx;~i;i--)
13        {
14            if(!(x&(1LL<<i))) continue;
15            if(!d[i])
16            {
17                d[i]=x;
18                break;
19            }
20            x^=d[i];
21        }
22        return x>0;
23    }
24    type ask_max()
25    {
26        type res=0;
27        for(int i=mx;~i;i--)

```

```

28     {
29         if((res^d[i])>res) res^=d[i];
30     }
31     return res;
32 }
33 void merge(Base x)
34 {
35     for(int i=mx;~i;i--)
36     {
37         if(x.d[i]) insert(x.d[i]);
38     }
39 }
40 #undef type
41 }bs;

```

8.3.2 带删除线性基

```

1 struct Base
2 {
3     #define type ll
4     #define mx 60
5     type d[mx+3];
6     int p[mx+3],cnt;
7     void init()
8     {
9         memset(d,0,sizeof(d));
10        cnt=0;
11    }
12    bool insert(type x,int pos=0)
13    {
14        int i;
15        for(i=mx;~i;i--)
16        {
17            if(!(x&(1LL<<i))) continue;
18            if(!d[i])
19            {
20                cnt++;
21                d[i]=x;
22                p[i]=pos;
23                break;
24            }
25            if(p[i]<pos)
26            {
27                swap(d[i],x);
28                swap(p[i],pos);
29            }
30            x^=d[i];
31        }
32        return x>0;
33    }
34    type query_max(int pos=-1)
35    {
36        int i;
37        type res=0;

```

```

38        for(i=mx;~i;i--)
39        {
40            if(p[i]>=pos)
41            {
42                if((res^d[i])>res) res^=d[i];
43            }
44        }
45        return res;
46    }
47    type query_min(int pos=-1)
48    {
49        for(int i=0;i<=mx;i++)
50        {
51            if(d[i]&&p[i]>=pos) return d[i];
52        }
53        return 0;
54    }
55    void merge(Base x)
56    {
57        if(cnt<x.cnt)
58        {
59            swap(cnt,x.cnt);
60            swap(d,x.d);
61            swap(p,x.p);
62        }
63        for(int i=mx;~i;i--)
64        {
65            if(x.d[i]) insert(x.d[i]);
66        }
67    }
68    //kth min
69    //first use rebuild()
70    type tp[mx+3];
71    void rebuild()
72    {
73        int i,j;
74        cnt=0;
75        for(i=mx;~i;i--)
76        {
77            for(j=i-1;~j;j--)
78            {
79                if(d[i]&(1LL<<j)) d[i]^=d[j];
80            }
81        }
82        for(i=0;i<=mx;i++)
83        {
84            if(d[i]) tp[cnt++]=d[i];
85        }
86    }
87    type kth(type k)
88    {
89        type res=0;
90        if(k>=(1LL<<cnt)) return -1;
91        for(int i=mx;~i;i--)

```

```

92     {
93         if(k&(1LL<<i)) res^=tp[i];
94     }
95     return res;
96 }
97 };

```

9 博弈

9.1 sg 函数

```

1  int sg[MAX],a[MAX],n;
2  int dfs(int x)
3  {
4      if(sg[x]!=-1) return sg[x];
5      int i,j,flag[105]={0};
6      for(i=1;i<=n;i++)
7      {
8          if(x>=a[i])
9          {
10             dfs(x-a[i]);
11             flag[sg[x-a[i]]]=1;
12         }
13     }
14     for(i=0;;i++)
15     {
16         if(!flag[i])
17         {
18             j=i;
19             break;
20         }
21     }
22     return sg[x]=j;
23 }

```

9.2 结论

1. 阶梯博弈

0 层为终点的阶梯博弈，等价于奇数层的 nim，偶数层的移动不影响结果

2.SJ 定理

对于任意一个 Anti-SG 游戏，如果我们规定当局面中所有的单一游戏的 SG 值为 0 时，游戏结束。

先手必胜当且仅当：

- (1) 游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1;
- (2) 游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函

数大于 1。

3.k-nim

问题描述: 给定 n 堆石子，每堆 a_i 个，每次可以从最多 k 堆中拿走任意个，先拿完的人胜利

结论: 先手必败, 当且仅当将每个 a_i 写成二进制，对每一个二进制位，这一位为 1 的 i 的个数为 s , $s \bmod (k+1) = 0$

4. 树上删边博弈

结论: 叶子 sg 为 0, 非叶子 sg 为所有子节点的 sg+1 的异或和

10 dp

10.1 LIS

```

1  struct LIS
2  {
3      #define type int
4      const type inf=INF;
5      vector<type> work(vector<type> a,bool strict)
6      {
7          int i,pos,len,n;
8          n=a.size();
9          vector<type> b(n,inf);
10         vector<int> tmp(n),res;
11         for(i=0;i<n;i++)
12         {
13             // strict: lower_bound
14             //not strict: upper_bound
15             if(strict) pos=lower_bound(b.begin(),b.
16                 end(),a[i])-b.begin();
17             else pos=upper_bound(b.begin(),b.end(),a[
18                 i])-b.begin();
19             b[pos]=a[i];
20             tmp[i]=pos;
21         }
22         len=lower_bound(b.begin(),b.end(),inf)-b.
23             begin();
24         for(i=n-1;~i;i--)
25         {
26             if(!len) break;
27             if(tmp[i]+1==len)
28             {
29                 len--;
30                 res.push_back(i);
31             }
32         }
33     }
34 }

```



```

30     return res;
31 }
32 #undef type
33 }lis;

```

```

25     return res;
26 };
27     return dfs(tot-1,1,0,1);
28 }

```

10.2 LPS

```

1 int dp[105][105];
2 void LPS(char *s,int n) // s[1..n]
3 {
4     int i,len,l,r;
5     memset(dp,0,sizeof dp);
6     for(i=1;i<=n;i++) dp[i][i]=1;
7     for(len=2;len<=n;len++)
8     {
9         for(l=1;l+len-1<=n;l++)
10        {
11            r=l+len-1;
12            if(s[l]==s[r]) dp[l][r]=dp[l+1][r-1]+2;
13            else dp[l][r]=max(dp[l+1][r],dp[l][r-1]);
14        }
15    }
16 }

```

10.3 数位 dp

```

1 const int DIG=20+2;
2 ll dp[DIG][2];
3 ll gao(ll x)
4 {
5     const int base=10;
6     int p[DIG],tot=0;
7     if(x==-1) return 0;
8     while(1)
9     {
10        p[tot++]=x%base;
11        x/=base;
12        if(!x) break;
13    }
14    function<ll(int,int,int,int)> dfs=[&](int pos,
15        int lead,int sta,int limit)->ll
16    {
17        if(pos==--1) return ;
18        if(!limit&&!lead&&dp[pos][sta]!=-1) return dp[
19            pos][sta];
20        ll res=0;
21        for(int i=(limit?p[pos]:base-1);~i;i--)
22        {
23            res+=dfs(pos-1,lead&&i==0&&pos,,limit&&i==
24                p[pos]);
25        }
26        if(!limit&&!lead) dp[pos][sta]=res;
27    }
28 }

```

11 杂项

11.1 FastIO

```

1 namespace fastIO{
2     #define BUF_SIZE 100000
3     #define OUT_SIZE 100000
4     #define ll long long
5     //fread->read
6     bool IOerror=0;
7     // inline char nc(){char ch=getchar();if(ch==--1)
8         IOerror=1;return ch;}
9     inline char nc(){
10        static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*
11            pend=buf+BUF_SIZE;
12        if(p1==pend){
13            p1=buf;pend=buf+fread(buf,1,BUF_SIZE,
14                stdin);
15            if(pend==p1){IOerror=1;return -1;}
16        }
17        return *p1++;
18    }
19    inline bool blank(char ch){return ch==' '||ch=='
20        \n' || ch=='\r' || ch=='\t';}
21    template<class T> inline bool read(T &x){
22        bool sign=0;char ch=nc();x=0;
23        for(;blank(ch);ch=nc());
24        if(IOerror)return false;
25        if(ch=='-')sign=1,ch=nc();
26        for(;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
27        if(sign)x=-x;
28        return true;
29    }
30    inline bool read(double &x){
31        bool sign=0;char ch=nc();x=0;
32        for(;blank(ch);ch=nc());
33        if(IOerror)return false;
34        if(ch=='-')sign=1,ch=nc();
35        for(;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
36        if(ch=='.'){
37            double tmp=1; ch=nc();
38            for(;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x
39                +=tmp*(ch-'0');
40        }
41        if(sign)x=-x;
42        return true;
43    }
44    inline bool read(char *s){

```

```

80     template<class T>void print(T x){Ostream.print(x
        );};
81     inline void print(char x){Ostream.out(x);}
82     inline void print(char *s){Ostream.print(s);}
83     inline void print(string s){Ostream.print(s.
        c_str());};
84     inline void print(const char *s){Ostream.print(s
        );};
85     inline void print(double x,int y){Ostream.print(
        x,y);};
86     template<class T,class... U>void print(const T&
        h,const U&... t){print(h);print(t...);};
87     void println(){print('\n');};
88     template<class T,class... U>void println(const T
        & h,const U&... t){print(h);println(t...);};
89     inline void flush(){Ostream.flush();};
90     #undef ll
91     #undef OUT_SIZE
92     #undef BUF_SIZE
93 };
94 using namespace fastIO;

```

```

1  ll mul2(ll x,ll y,ll p)
2  {
3      ll res=(x*y-ll((long double)x/p*y+1.0e-8)*p);
4      return res<0?res+p:res;
5  }

```

```

1 struct FAST_MOD // Montgomery modular
    multiplication
2 {
3     #define mod_byte 64 // 32 or 64
4
5     #if mod_byte==32
6         #define itype int32_t
7         #define utype uint32_t
8         #define utype2 uint64_t
9     #else
10        #define itype int64_t
11        #define utype uint64_t
12        #define utype2 __uint128_t
13    #endif
14
15    static utype mod,r,r2;
16    utype x;
17    static utype init(utype x){return reduce(utype2(
        x)*r2);}
18    static utype get_mod(){return mod;}
19    static void set mod(utype m)

```

```

20 {
21     mod=r*m;
22     assert((mod&1)&&mod>2);
23     for(int i=0;i<5;i++) r*=2-r*m;
24     r2=-utype2(m)%m;
25 }
26 static utype reduce(utype2 x)
27 {
28     utype y=utype(x>>mod_byte)-utype((utype2(
29         utype(x)*r)*mod)>>mod_byte);
30     return itype(y)<0?y+mod:y;
31 }
32 FAST_MOD():x(0){}
33 FAST_MOD(utype n):x(init(n)){}
34 FAST_MOD operator+(const FAST_MOD &a)const {
35     return FAST_MOD(*this)+=a;
36 }
37 FAST_MOD operator-(const FAST_MOD &a)const {
38     return FAST_MOD(*this)-=a;
39 }
40 FAST_MOD operator*(const FAST_MOD &a)const {
41     return FAST_MOD(*this)*=a;
42 }
43 FAST_MOD& operator+=(const FAST_MOD &a)
44 {
45     x+=a.x-mod;
46     if(itype(x)<0) x+=mod;
47     return *this;
48 }
49 FAST_MOD& operator-=(const FAST_MOD &a)
50 {
51     x=x>=a.x?x-a.x:mod-a.x+x;
52     return *this;
53 }
54 FAST_MOD& operator*=(const FAST_MOD &a)
55 {
56     x=reduce(utype2(x)*a.x);
57     return *this;
58 }
59 FAST_MOD& operator++()
60 {
61     x++;
62     if(x==mod) x=0;
63     return *this;
64 }
65 FAST_MOD& operator++(int)
66 {
67     x++;
68     if(x==mod) x=0;
69     return *this;
70 }
71 FAST_MOD& operator--()
72 {
73     if(x==0) x=get_mod();
74     x--;
75     return *this;
76 }

```

```

70 FAST_MOD& operator--(int)
71 {
72     if(x==0) x=get_mod();
73     x--;
74     return *this;
75 }
76 // friend bool operator<(const FAST_MOD& a,const
77     FAST_MOD& b){return reduce(a.x)<reduce(b.x);}
78 // friend bool operator>(const FAST_MOD& a,const
79     FAST_MOD& b){return reduce(a.x)>reduce(b.x);}
80 friend bool operator==(const FAST_MOD& a,const
81     FAST_MOD& b){return a.x==b.x;}
82 friend bool operator!=(const FAST_MOD& a,const
83     FAST_MOD& b){return a.x!=b.x;}
84 friend ostream &operator<<(ostream &os, const
85     FAST_MOD &a){return os<<reduce(a.x);}
86 utype get_val()const {return reduce(x);}
87 };
88 utype FAST_MOD::mod,FAST_MOD::r,FAST_MOD::r2;
89 #undef itype
90 #undef utype
91 #undef utype2
92 typedef FAST_MOD mint;
93 /*
94 mint::set_mod(p);
95 */

```

11.4 xor_sum(1,n)

```

1 ll xor_sum(ll n)
2 {
3     ll t=n&3;
4     if (t&1) return t/2ull^1;
5     return t/2ull^n;
6 }

```

11.5 约瑟夫环 kth

```

1 ll kth(ll n,ll m,ll k)
2 {
3     if(m==1) return k;
4     ll res=(m-1)%(n-k+1);
5     for(ll i=n-k+2,stp=0;i<=n;i+=stp,res+=stp*m)
6     {
7         if(res+m>=i)
8         {
9             res=(res+m)%i;
10            i++;
11            stp=0;
12        }
13        else
14        {
15            stp=(i-res-2)/(m-1);

```

```

16         if(i+stp>n)
17         {
18             res+=(n-(i-1))*m;
19             break;
20         }
21     }
22 }
23 return res+1;
24 }

```

11.6 判断星期几

```

1 int judge(int y,int m,int d)
2 {
3     int res;
4     if(m==1||m==2) m+=12,y--;
5     if((y<1752)||((y==1752&&m<9)||((y==1752&&m==9&&d
6         <3))) res=(d+2*m+3*(m+1)/5+y+y/4+y/5)%7;
7     else res=(d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7;
8     return res+1;
9 }

```

11.7 整数三分

```

1 while(l<r)
2 {
3     lm=l+(r-l)/3;
4     rm=r-(r-l)/3;
5     tl=cal(lm);
6     tr=cal(rm);
7     // tl>tr:min tl<tr:max
8     if(tl>tr) l=lm+1;
9     else r=rm-1;
10 }

```

11.8 有根树与 prufer 序列的转换

```

1 //TODO

```

11.9 网格整数点正方形个数

```

1 struct node
2 {
3     int x,y;
4     void input(){scanf("%d%d",&x,&y);}
5 }p[511];
6 int main()
7 {
8     int n,i,j,ans;
9     while(~scanf("%d",&n))
10    {

```

```

11    map<pair<int,int>,int> mp;
12    for(i=0;i<n;i++)
13    {
14        p[i].input();
15        mp[MP(p[i].x,p[i].y)]=1;
16    }
17    ans=0;
18    for(i=0;i<n;i++)
19    {
20        for(j=i+1;j<n;j++)
21        {
22            int a,b,c,d,e,f,g,h;
23            a=p[i].x;
24            b=p[i].y;
25            c=p[j].x;
26            d=p[j].y;
27            e=a+b-c-d;
28            f=-a+b+c+d;
29            g=a-b+c+d;
30            h=a+b-c+d;
31            if(abs(e%2)+abs(f%2)+abs(g%2)+abs(h%2)
32                ==0)
33            {
34                if(mp[MP(e/2,f/2)]&&mp[MP(g/2,h/2)]
35                    ) ans++;
36            }
37        }
38    }
39    printf("%d\n",ans/2);
40    return 0;
41 }

```

11.10 模拟退火

```

1 /*简单版
2 1. 模拟退火求费马点->复杂版
3
4 2.求矩形区域内一点到各点距离之和最短时间复杂度
5
6 cnt*c1*c2*n
7
8 */
9 int sgn(double x)
10 {
11     if(fabs(x)<eps) return 0;
12     else return x>0?-1:1;
13 }
14 struct Point
15 {
16     double x,y;
17     Point(){}
18     Point(double a,double b)

```

```

19 {
20     x=a;
21     y=b;
22 }
23 void input()
24 {
25     scanf("%lf%lf",&x,&y);
26 }
27 };
28 typedef Point Vector;
29 Vector operator -(Vector a,Vector b){return Vector(
    a.x-b.x,a.y-b.y);}
30 double dot(Vector a,Vector b){return a.x*b.x+a.y*b.
    y;}
31 double dist(Point a,Point b){return sqrt(dot(a-b,a-
    b));}
32 double lx,ly;//矩形区域(0,0)-(lx,ly)
33 int check(double x,double y)
34 {
35     if(sgn(x)<0||sgn(y)<0||sgn(x-lx)>0||sgn(y-ly)>0)
        return 1;
36     return 0;
37 }
38 double Rand(double r,double l)
39 {
40     return(rand()%((int)(l-r)*1000))/(1000.0+r);
41 }
42 double getres(Point t,Point *p,int n)//求距离之和
43 {
44     double res=0;
45     for(int i=0;i<n;i++)
46     {
47         res+=dist(t,p[i]);
48     }
49     return res;
50 }
51 pair<Point,double> SA(Point *p,int n)//模拟退火
52 {
53     srand(time(0));//重置随机种子
54     const double k=0.85;//退火常数
55     const int c1=30;//随机取点的个数
56     const int c2=50;//退火次数
57     Point q[c1+10];//随机取点
58     double dis[c1+10];//每个点的计算结果
59     int i,j;
60     for(i=1;i<=c1;i++)
61     {
62         q[i]=Point(Rand(0,lx),Rand(0,ly));
63         dis[i]=getres(q[i],p,n);
64     }
65     double tmax=max(lx,ly);
66     double tmin=1e-3;
67     // int cnt计算外层循环次数=0;//
68     while(tmax>tmin)

```

```

69 {
70     for(i=1;i<=c1;i++)
71     {
72         for(j=1;j<=c2;j++)
73         {
74             double ang=Rand(0,2*PI);
75             Point z;
76             z.x=q[i].x+cos(ang)*tmax;
77             z.y=q[i].y+sin(ang)*tmax;
78             if(check(z.x,z.y)) continue;
79             double temp=getres(z,p,n);
80             if(temp<dis[i])
81             {
82                 dis[i]=temp;
83                 q[i]=z;
84             }
85         }
86     }
87     // cnt++;
88     tmax*=k;
89 }
90 // cout<<cnt*c1*c2*n<<endl时间复杂度;//
91 int pos=1;
92 for(i=2;i<=c1;i++)
93 {
94     if(dis[i]<dis[pos])
95     {
96         pos=i;
97     }
98 }
99 pair<Point,double> res;
100 res=make_pair(q[pos],dis[pos]);
101 return res;
102 }

```

11.11 斐波那契 01 串的第 k 个字符

```

1 int fib[MAX],tot;
2 void init_fib(ll limt)
3 {
4     assert(limt>=0);
5     tot=1;
6     fib[0]=fib[1]=1;
7     for(int i=2;;i++)
8     {
9         if(fib[i-1]>limt-fib[i-2])
10        {
11            tot=i-1;
12            break;
13        }
14        fib[i]=fib[i-1]+fib[i-2];
15    }
16 }

```

```

17 //S(0)="0", S(1)="1", S(i)=S(i-1)+S(i-2)
18 //FibString: S(i) [i>=1]
19 int get_kth_FibString(ll k)
20 {
21     int i;
22     for(i=tot;i>=2;i--)
23     {
24         if(k>fib[i]) k-=fib[i];
25     }
26     return k==1;
27 }
28 int is_fib(ll x)
29 {
30     for(int i=1;i<=tot;i++)
31     {
32         if(x==fib[i]) return 1;
33     }
34     return 0;
35 }

```

11.12 光速幂

```

1 struct light_speed_pow
2 {
3     #define type int
4     int n,sq;
5     type res[MAX][2],val;
6     void init(int _n,type _val)
7     {
8         n=_n;
9         val=_val;
10        sq=sqrt(n)+1;
11        res[0][0]=res[0][1]=1;
12        for(int i=1;i<=sq;i++) res[i][0]=1ll*res[i-1][0]*val%mod;
13        for(int i=1;i<=sq;i++) res[i][1]=1ll*res[i-1][1]*res[sq][0]%mod;
14    }
15    type qpow(int exp)
16    {
17        if(exp<=sq) return res[exp][0];
18        return 1ll*res[exp/sq][1]*res[exp-exp/sq*sq][0]%mod;
19    }
20    #undef type
21 }lsp;
22 /*
23 val^exp
24 O(sqrt max_exp)-O(1)
25 lsp.init(max_exp,val);
26 */

```

11.13 倍增求等比数列和

```

1 ll cal(ll a0,ll q,ll n,ll p)
2 {
3     int i;
4     ll tmp[33],sum[33],res,now;
5     tmp[0]=0;
6     tmp[1]=q;
7     for(i=2;i<=30;i++) tmp[i]=tmp[i-1]*tmp[i-1]%p;
8     sum[0]=1;
9     for(i=1;i<=30;i++) sum[i]=sum[i-1]*(1+tmp[i])%p;
10    res=0;
11    now=1;
12    for(i=30;~i;i--)
13    {
14        if((n>>i)&1)
15        {
16            res=(res+now*sum[i])%p;
17            now=(now*tmp[i+1])%p;
18        }
19    }
20    return a0*res%p;
21 }

```

11.14 矩阵旋转 90 度

```

1 int n,m,a[MAX][MAX],tmp[MAX][MAX];
2 void rotate90()
3 {
4     int i,j;
5     for(i=1;i<=n;i++)
6     {
7         for(j=1;j<=m;j++)
8         {
9             tmp[i][j]=a[i][j];
10        }
11    }
12    for(i=1;i<=n;i++)
13    {
14        for(j=1;j<=m;j++)
15        {
16            a[j][i]=tmp[n-i+1][j];
17        }
18    }
19    swap(n,m);
20 }

```

12 附录

12.1 NTT 常用模数

$$r * 2^k + 1, r, k, g$$

3,1,1,2

5,1,2,2
 17,1,4,3
 97,3,5,5
 193,3,6,5
 257,1,8,3
 7681,15,9,17
 12289,3,12,11
 40961,5,13,3
 65537,1,16,3
 786433,3,18,10
 5767169,11,19,3
 7340033,7,20,3
 23068673,11,21,3
 104857601,25,22,3
 167772161,5,25,3
 469762049,7,26,3
 998244353,119,23,3
 1004535809,479,21,3
 2013265921,15,27,31
 2281701377,17,27,3
 3221225473,3,30,5
 75161927681,35,31,3
 77309411329,9,33,7
 206158430209,3,36,22
 2061584302081,15,37,7
 2748779069441,5,39,3
 6597069766657,3,41,5
 39582418599937,9,42,5
 79164837199873,9,43,5
 263882790666241,15,44,7
 1231453023109121,35,45,3
 1337006139375617,19,46,3
 3799912185593857,27,47,5
 4222124650659841,15,48,19
 7881299347898369,7,50,6
 31525197391593473,7,52,3
 180143985094819841,5,55,6
 1945555039024054273,27,56,5
 4179340454199820289,29,57,3

```

1  LBasis intersection(const LBasis &a, const LBasis &
    b){
2      LBasis ans, c = b, d = b;
3      ans.init();
4      for (int i = 0; i <= 32; i++){
5          ll x = a.d[i];
6          if(!x)continue;
7          int j = i;
8          ll T = 0;
9          for(; j >= 0; --j){
10             if((x >> j) & 1)
11                 if(c.d[j]) {x ^= c.d[j]; T ^= d.d[j];}
12                 else break;
13             }
14             if(!x) ans.d[i] = T;
15             else {c.d[j] = x; d.d[j] = T;}
16         }
17         return ans;
18     }
  
```

12.2 线性基求交