

Algorithm__Template\Data__Structure.cpp

```

1  /**
2   * 树状数组 BIT
3   * 线段树 SegTree
4   * 并查集 DisjointSet
5   * 最近公共祖先 LCA
6   * 分块 block
7   * ST表 st
8   *
9  */
10 #include <iostream>
11 #include <vector>
12 #include <string>
13 #include <cstring>
14 #include <set>
15 #include <map>
16 #include <queue>
17 #include <ctime>
18 #include <random>
19 #include <sstream>
20 #include <numeric>
21 #include <stdio.h>
22 #include <algorithm>
23 using namespace std;
24
25 #define rep(i,x,n) for(int i = x; i <= n; i++)
26
27 typedef long long LL;
28 typedef pair<int,int> PII;
29
30 const int INF = 0x3f3f3f3f;
31 const int N = 1e5 + 21;
32
33 namespace golitter {
34 namespace BIT {
35     /** url: https://ac.nowcoder.com/acm/contest/61132/L
36      底部确定，顶部无穷大
37      最外面的结点是2的n次方，如上图1，2，4，8的结点
38      奇数的结点一定是叶子结点
39      数组一定要从1开始
40
41
42      树状数组 离线处理
43      查询种类数，维护区间内仅有一个或两个此种元素
44      一般都考虑离线询问
45      将询问的区间按照右端点小在前排序
46
47      */
48     const int N = 5e5 + 21;
49     int n,m;
50     int tr[N];
51     int lowbit(int x) {
52         return x & -x;
53     }
54     void add(int x, int c) {
55         // if(!x) return; // 如果 树状数组 离线处理 记得**
56         https://www.luogu.com.cn/problem/P4113

```

```

56     for(; x < N; x += lowbit(x)) tr[x] += c;
57 }
58 LL sum(int x) {
59     LL res = 0;
60     for(; x; x -= lowbit(x)) res += tr[x];
61     return res;
62 }
63 void solve() {
64     cin>>n>>m;
65     rep(i,1,n) {
66         int x; cin>>x;
67         add(i, x);
68     }
69     rep(i,1,m) {
70         int a,b,c; cin>>a>>b>>c;
71         if(a == 1) {
72             add(b,c);
73         } else {
74             cout<<sum(c) - sum(b-1)<<endl;
75         }
76     }
77 }
78
79 }}
80
81 namespace golitter {
82 namespace SegTree {
83     /**
84     * 小区间的值更新大区间的值
85     * 问题满足：区间加法： [1, r] 可以用 [1, mid] 和 [mid+1, r]的值产生
86     *     不满足的问题：区间的众数，区间最长连续问题，最长不下降问题
87     *
88     * 解题步骤： 建树，
89     */
90
91     int w[N],n,m; // 注意 w[N] 开LL ( https://www.luogu.com.cn/problem/P2357
92     struct adt {
93         int l,r;
94         LL sum,add;
95     }tr[N << 2];
96     // 左子树
97     inline int ls(int p) {return p<<1; }
98     // 右子树
99     inline int rs(int p) {return p<<1|1; }
100    // 向上更新
101    void pushup(int u) {
102        tr[u].sum = tr[ls(u)].sum + tr[rs(u)].sum;
103    }
104    // 向下回溯时，先进行更新
105    void pushdown(int u) { // 懒标记，该节点曾经被修改，但其子节点尚未被更新。
106        auto &root = tr[u], &right = tr[rs(u)], &left = tr[ls(u)];
107        if(root.add) {
108            right.add += root.add; right.sum += (LL)(right.r - right.l + 1)*root.add;
109            left.add += root.add; left.sum += (LL)(left.r - left.l + 1)*root.add;
110            root.add = 0;
111        }
112    }
113 }
114 // 建树
115 void build(int u, int l, int r) {

```

```

116     if(l == r) tr[u] = {l, r, w[r], 0};
117     else {
118         tr[u] = {l,r}; // 容易忘
119         int mid = l + r >> 1;
120         build(ls(u), l, mid), build(rs(u), mid + 1, r);
121         pushup(u);
122     }
123 }
124 // 修改
125 void modify(int u, int l, int r, int d) {
126     if(tr[u].l >= l && tr[u].r <= r) {
127         tr[u].sum += (LL)(tr[u].r - tr[u].l + 1)*d;
128         tr[u].add += d;
129     }
130     else {
131         pushdown(u);
132         int mid = tr[u].l + tr[u].r >> 1;
133         if(l <= mid) modify(ls(u), l ,r, d);
134         if(r > mid) modify(rs(u), l, r, d);
135         pushup(u);
136     }
137 }
138 // 查询
139 LL query(int u, int l, int r) {
140     if(tr[u].l >= l && tr[u].r <= r) {
141         return tr[u].sum;
142     }
143     pushdown(u);
144     int mid = tr[u].l + tr[u].r >> 1;
145     LL sum(0);
146     if(l <= mid) sum = query(ls(u), l, r);
147     if(r > mid ) sum += query(rs(u), l, r);
148     return sum;
149 }
150 void solve() {
151     cin>>n>>m;
152     for(int i = 1; i <= n; ++i) cin>>w[i];
153     build(1, 1, n);
154     // cout<<tr[1].sum<<endl;
155     int xx, yy;
156     int op;
157     while(m--) {
158         cin>>op>>xx>>yy;
159         if(op == 1) {
160             cin>>op;
161             modify(1, xx, yy, op);
162         } else {
163             cout<<query(1, xx, yy); puts("");
164         }
165     }
166 }
167 /**
168  * 建树时 向下回溯对父节点进行更新
169  * modify时 向下回溯要先对左右节点进行更新，之后再对父节点进行更新
170  * query时 向下回溯要先对左右节点进行更新
171  *
172  */
173
174 namespace mul_and_add {
175

```

```

176
177     // 先 * 后 +
178
179     const int N = 2e5 + 21;
180     int w[N];
181     int mod;
182     struct SegTree {
183         int l, r;
184         LL sum,mul,add;
185     }tr[N << 2];
186     inline int ls(int r) {return r << 1; }
187     inline int rs(int r) {return r << 1 | 1; }
188     void pushup(int u) {
189         tr[u].sum = (tr[ls(u)].sum + tr[rs(u)].sum) % mod;
190     }
191     void pushdown(int u) {
192         auto &root = tr[u], &right = tr[rs(u)], &left = tr[ls(u)];
193         // 先计算子节点和 子.sum * 根.mul + 根.add * (子节点区间范围)
194         right.sum = (right.sum * root.mul + root.add * (right.r - right.l + 1)) % mod;
195         left.sum = (left.sum * root.mul + root.add * (left.r - left.l + 1)) % mod;
196         // 之后更新子节点lazy # mul 子.mul = 子.mul * 根.mul
197         right.mul = (right.mul * root.mul) % mod;
198         left.mul = (left.mul * root.mul) % mod;
199         // 最后更新子节点lazy # add 子.add = 子.add * 根.mul + 根.add
200         left.add = (left.add * root.mul + root.add) % mod;
201         right.add = (right.add * root.mul + root.add) % mod;
202         // 根.add = 0 根.mul = 1
203         root.add = 0;
204         root.mul = 1;
205     }
206     void build(int u, int l, int r) {
207         if(l == r) tr[u] = {l,r,w[r],1,0};
208         else {
209             int mid = l + r >> 1;
210             tr[u] = {l,r,0,1,0};
211             build(ls(u),l, mid); build(rs(u), mid + 1, r);
212             pushup(u);
213         }
214     }
215     void modify_mul(int u, int l, int r, int k) {
216         if(tr[u].l >= l && tr[u].r <= r) {
217             // 加 = 加 * k
218             // 乘 = 乘 * k
219             // 和 = 和 * k
220             tr[u].add = (tr[u].add * k) % mod;
221             tr[u].mul = (tr[u].mul * k) % mod;
222             tr[u].sum = (tr[u].sum * k) % mod;
223             return ;
224         }
225         pushdown(u);
226         int mid = tr[u].l + tr[u].r >> 1;
227         if(l <= mid) modify_mul(ls(u), l,r,k);
228         if(r > mid) modify_mul(rs(u),l,r,k);
229         pushup(u);
230     }
231     void modify_add(int u, int l, int r, int k) {
232         if(tr[u].l >= l && tr[u].r <= r) {
233             // 加 = 加 + k
234             // 和 = 和 + k * (区间长度)
235             tr[u].add = (tr[u].add + k) % mod;

```

```

236         tr[u].sum = (tr[u].sum + k * (tr[u].r - tr[u].l + 1)) % mod;
237         return ;
238     }
239     pushdown(u);
240     int mid = tr[u].l + tr[u].r >> 1;
241     if(l <= mid) modify_add(ls(u), l,r,k);
242     if(r > mid) modify_add(rs(u),l,r,k);
243     pushup(u);
244 }
245 LL query(int u, int l, int r) {
246     if(tr[u].l >= l && tr[u].r <= r) {
247         return tr[u].sum;
248     }
249     LL sum = 0;
250     pushdown(u);
251     int mid = tr[u].l + tr[u].r >> 1;
252     if(l <= mid) sum = (sum + query(ls(u),l,r) ) % mod;
253     if(r > mid) sum = (sum + query(rs(u),l,r)) % mod;
254     return sum;
255 }
256 void solve() {
257     int n,q; cin>>n>>q>>mod;
258     rep(i,1,n) cin>>w[i];
259     build(1,1,n);
260     while(q--) {
261         int op,l,r; cin>>op>>l>>r;
262         int k;
263         if(op == 1) {
264             cin>>k;
265             modify_mul(1,l,r,k);
266         } else if(op == 2) {
267             cin>>k;
268             modify_add(1,l,r,k);
269         } else {
270             cout<<query(1,l,r)<<endl;
271         }
272     }
273 }
274
275 }
276
277 /**
278 eat k: 吃掉当前的第k个零食。右边的零食全部往左移动一位（编号减一）。
279 query i j: 查询当前第i个零食到第j个零食里面美味度最高的和最低的零食的美味度。
280 */
281 namespace dynamic_maximum {
282
283 const int INF = 0x3f3f3f3f;
284 // https://www.luogu.com.cn/problem/P6011
285 // https://ac.nowcoder.com/acm/problem/208250
286 /**
287 * 动态删点求区间最值
288 */
289 const int N = 2e6 + 21;
290 int w[N];
291 struct SegTree {
292     int l,r,num,mi,ma; // num 记录这一段区间内的有效长度
293 }tr[N << 2];
294 inline int ls(int u) {return u << 1; }
295 inline int rs(int u) {return u << 1 | 1; }

```

```

296
297 void pushup(int u) {
298     tr[u].ma = max(tr[ls(u)].ma, tr[rs(u)].ma);
299     tr[u].mi = min(tr[ls(u)].mi, tr[rs(u)].mi);
300     tr[u].num = tr[ls(u)].num + tr[rs(u)].num;
301 }
302
303 void build(int u, int l, int r) {
304     if(l == r) tr[u] = {l, r, 1, w[r], w[r]};
305     else {
306         tr[u] = {l, r};
307         int mid = l + r >> 1;
308         build(ls(u), l, mid), build(rs(u), mid + 1, r);
309         pushup(u);
310     }
311 }
312
313 void del(int u, int l, int r, int x) {
314     if(l == r) { // 删点, 将该点 经过最大值min, 最小值max, num = 0, 进行忽略 (删除
315         tr[u].mi = INF;
316         tr[u].ma = -INF;
317         tr[u].num = 0;
318         return ;
319     }
320     int mid = l + r >> 1;
321     // 如果左区间实际长度大于x, 表示x在左
322     if(tr[ls(u)].num >= x) del(ls(u), l, mid, x);
323     // 否则在右, 在右时, 需要将x 删除掉左区间实际长度
324     else del(rs(u), mid + 1, r, x - tr[ls(u)].num);
325     pushup(u);
326 }
327 int query_mi(int u, int l, int r) {
328     if(l <= 1 && r >= tr[u].num) {
329         return tr[u].mi;
330     }
331     int lnum = tr[ls(u)].num;
332     int tmp = INF;
333     if(l <= lnum) {
334         tmp = query_mi(ls(u), l, r);
335     }
336     if(r > lnum) {
337         tmp = min(tmp, query_mi(rs(u), l - lnum, r - lnum));
338     }
339     return tmp;
340 }
341 int query_ma(int u, int l, int r) { // 同理
342     if(l <= 1 && r >= tr[u].num) {
343         return tr[u].ma;
344     }
345     int lnum = tr[ls(u)].num;
346     int tmp = -INF;
347     if(l <= lnum) {
348         tmp = query_ma(ls(u), l, r);
349     }
350     if(r > lnum) {
351         tmp = max(tmp, query_ma(rs(u), l - lnum, r - lnum));
352     }
353     return tmp;
354 }
355 void inpfile();

```

```

356 void solve() {
357     int n,m; cin>>n>>m;
358     rep(i,1,n) w[i] = fread();
359     build(1,1,n);
360     while(m--) {
361         int opt; cin>>opt;
362         if(opt == 1) {
363             int k; cin>>k;
364             del(1,1,n,k);
365         } else {
366             int l,r; l = fread(), r = fread();
367             // cout<<query_mi(1,l,r)<<" "<<query_ma(1,l,r)<<endl;
368             printf("%d %d\n", query_mi(1,l,r), query_ma(1,l,r));
369         }
370     }
371 }
372
373 }
374
375 namespace scanning_line {
376
377     const int N = 2e5 + 21;
378     /**
379      * 操作一：将某个区间 [l,r] + k
380      * 操作二：整个区间中，长度大于0的区间总长度是多少
381      *
382      * 线段树中的节点信息：
383      *     1. cnt 当前区间整个被覆盖次数
384      *     2. 不考虑祖先节点cnt的前提下， cnt > 0 的区间总长
385      */
386     int w[N];
387     int n;
388     struct Segment { // 线段
389         double x, y1, y2;
390         int k;
391         // 按横坐标进行排序
392         bool operator<(const Segment& rhs) const {
393             return x < rhs.x;
394         }
395     } seg[N << 1];
396     vector<double> native;
397     int find(double y) {
398         return lower_bound(all(native), y) - native.begin();
399     }
400     struct SegTree {
401         //线段树的节点tr[u]表示的线段树Node区间[tr[u].l,tr[u].r]维护离散化后的区间 --> [y_l, y_r
+ 1]
402         int l,r,cnt;
403         double len;
404     } tr[N << 3];
405     inline int ls(int u) {return u << 1; }
406     inline int rs(int u) {return u << 1 | 1; }
407     void pushup(int u) {
408         if(tr[u].cnt) tr[u].len = (native[tr[u].r + 1] - native[tr[u].l]);
409         else if(tr[u].l != tr[u].r) {
410             tr[u].len = tr[ls(u)].len + tr[rs(u)].len;
411         } else tr[u].len = 0;
412     }
413     void build(int u, int l, int r) {
414         if(l == r) tr[u] = {l,r,0,0};

```

```

415     else {
416         tr[u] = {l,r};
417         int mid = l + r >> 1;
418         build(ls(u),l,mid), build(rs(u),mid+1,r);
419     }
420 }
421 void modify(int u, int l, int r, int k) {
422     if(tr[u].l >= l && tr[u].r <= r) {
423         tr[u].cnt += k;
424         pushup(u);
425     } else {
426         int mid = tr[u].l + tr[u].r >> 1;
427         if(l <= mid) modify(ls(u), l,r,k);
428         if(r > mid) modify(rs(u), l,r,k);
429         pushup(u);
430     }
431 }
432 int T = 1;
433 void inpfiler();
434 void solve() {
435     native.clear();
436     int seglen = 0;
437     rep(i,1,n) {
438         double x1,x2,y1,y2; cin>>x1>>y1>>x2>>y2;
439         seg[seglen++] = {x1,y1,y2,1};
440         seg[seglen++] = {x2,y1,y2,-1};
441         native.push_back(y1), native.push_back(y2);
442     }
443     sort(all(native));
444     native.erase(unique(all(native)), native.end());
445     // 离散化后纵坐标有2n个点, 2n-1个区间, 构建线段树, 线段树的节点维护这些区间tr[i] -->
    [y_i, y_i+1], 所以线段树的节点个数与区间个数相同2n-1
446     //从1号点开始建线段树, 对应的离散化后的坐标的取值范围是0~ys.size()-2 --> 2n-1个
447     build(1,0, native.size() - 2);
448     sort(seg, seg + n * 2);
449     double ans = 0;
450     rep(i,0,n * 2 - 1) {
451         ans += tr[1].len * (seg[i].x - seg[i-1].x);
452         modify(1, find(seg[i].y1), find(seg[i].y2) - 1, seg[i].k);
453     }
454     printf("Test case #%d\n",T++);
455     printf("Total explored area: %.2lf\n\n",ans);
456 }
457 }
458
459 }}
460
461
462
463
464
465 namespace golitter {
466     namespace DisjointSet {
467         #include <unordered_map>
468
469         const int N = 234;
470         int fa[N];
471
472         // 朴素并查集
473         int find(int x) {

```



```

474     return fa[x] == x ? x : fa[x] = find(fa[x]);
475 }
476 void solve() {
477     int n,m; cin>>n>>m;
478     for(int i = 1; i <= n; ++i) fa[i] = i;
479     while(m--) {
480         int a,b; char opt[2];
481         scanf("%s%d%d", opt, &a, &b);
482         if(opt[0] == 'M') {
483             fa[find(a)] = find(b); // 合并
484         } else {
485             if(find(a) == find(b)) {
486                 puts("Yes");
487             } else puts("No");
488         }
489     }
490 }
491 // url: https://blog.csdn.net/m0_63794226/article/details/126697871
492 // 维护size的并查集 | 按秩合并
493 int fa[N], size[N],n;
494 // p[]存储每个点的祖宗节点, size[]只有祖宗节点的有意义, 表示祖宗节点所在集合中的点的数量
495
496 // 返回x的祖宗节点
497 int find(int x) {
498     if(fa[x] != x) fa[x] = find(fa[x]);
499     return fa[x];
500 }
501 // 初始化, 假定节点编号是1~n
502 void init() {
503     for(int i = 1; i <= n; ++i) fa[i] = i, size[i] = 1;
504 }
505 // 合并a和b所在的两个集合
506 // 合并时的小优化 -- 将一棵点数与深度都较小的集合树连接到一棵更大的集合树下。
507 // 在实际代码中, 即便不使用启发式合并, 代码也能够在规定时间内完成任务。
508 void merge(int a, int b) {
509     int pa = find(a), pb = find(b);
510     if(pa == pb) return ;
511     if(size[pa] > size[pb]) swap(pa, pb); // 保证小的合到大的里
512     fa[pa] = pb;
513     size[pb] += size[pa];
514 }
515
516 // 维护祖先节点距离的并查集 | 带权并查集
517 // problem: https://codeforces.com/contest/1850/problem/H
518 int fa[N], d[N];
519 //p[]存储每个点的祖宗节点, d[x]存储x到p[x]的距离
520 // 返回x的祖宗节点
521 int find(int x) {
522     if(fa[x] != x) {
523         int r = find(fa[x]);
524         d[x] += d[fa[x]];
525         fa[x] = r;
526     }
527     return fa[x];
528 }
529 // 初始化, 假定节点编号是1~n
530 void init() {
531     for(int i = 1; i <= n; ++i) fa[i] = i, d[i] = 0;
532 }
533 // 合并a和b所在的两个集合

```

```

534 int distance;
535 void merge(int a, int b, int t) {
536     int pa = find(a), pb = find(b);
537     fa[pa] = pb;
538     d[pa] = d[b] - d[a] + t;
539     /**
540      * 解释:
541      *   a .. pa           b .. pb
542      * a ---> pa           b ---> pb
543      * a -->pa   --> pb
544      *           b --|
545      * dist(pa -> pb) == d[b] + t - dist(a -> pa) = d[a];
546      * d[pa] = d[b] - d[a] + t;
547      */
548     // fa[find(a)] = find(b);
549     // d[find(a)] = distance; // 根据具体问题, 初始化find(a)的偏移量
550 }
551
552 /**
553  * 离散化处理:
554  *   https://www.acwing.com/solution/content/112727/
555  * int find(int x, unordered<int,int>& umii) {
556  *     return umii[x] == x ? x : umii[x] = find(umii[x], umii);
557  * }
558  * 合并: umii[ find(x, umii)] = umii[ find(y, umii)];
559  * 判断是否联通: find(x, umii) == find(y, umii)
560 */
561
562 }}
563
564
565
566 namespace golitter {
567 namespace LCA { // https://www.luogu.com.cn/problem/P8805#submit 加前缀和 求树上两点之间距
    离
568 // 板子
569 // https://www.acwing.com/problem/content/1173/
570 // https://www.luogu.com.cn/problem/P3379
571 namespace beizeng {
572
573 const int N = 10e5 + 21;
574 const int M = 2*N;
575 int h[M], e[M], ne[M], idx;
576 int dep[N], root, n, m, t;
577 int fa[N][20];
578 void add(int u, int v) { //
579     e[idx] = v, ne[idx] = h[u], h[u] = idx++;
580 }
581 void bfs() { // 找深度 + 预处理
582     memset(dep, 0x3f, sizeof(dep));
583     dep[0] = 0, dep[root] = 1;
584     queue<int> q;
585     q.push(root);
586     while (q.size())
587     {
588         int x = q.front(); q.pop();
589         for(int i = h[x]; ~i; i = ne[i]) {
590             int y = e[i];
591             if(dep[y] > dep[x]) {
592                 dep[y] = dep[x] + 1;

```

```

593         q.push(y);
594         fa[y][0] = x;
595         for(int k = 1; k <= t; k++) {
596             fa[y][k] = fa[ fa[y][k-1]][k-1];
597         }
598     }
599 }
600 }
601
602 }
603 int lca(int x, int y) {
604     if(dep[y] > dep[x]) swap(x,y); // 让x深度最大, 从x到y找
605     for(int k = t; k >= 0; --k) {
606         if(dep[ fa[x][k]] >= dep[y]) x = fa[x][k];
607     }
608     if(x == y) return x;
609     for(int k = t; k >= 0; --k) {
610         if(fa[x][k] != fa[y][k]) {
611             x = fa[x][k], y = fa[y][k];
612         }
613     }
614     return fa[x][0];
615 }
616 void solve() {
617     t = 15;
618     cin>>n>>m>>root;
619     memset(h, -1, sizeof(h));
620     for(int i = 1; i < n; ++i) {
621         int u,v; cin>>u>>v;
622         add(u,v); add(v,u);
623     }
624     bfs();
625     for(int i = 0; i < m; ++i) {
626         int u,v; cin>>u>>v;
627         cout<<lca(u,v)<<endl;
628     }
629 }
630 }
631
632 namespace tarjan {
633     // 离线 O(n + m)
634
635     const int N = 1e6 + 21;
636     int fa[N], e[N], h[N], ne[N], w[N], idx, dist[N], vis[N], ans[N];
637     vector<PII> ask[N];
638     int find(int x) {return x == fa[x] ? x : fa[x] = find(fa[x]); }
639     void add(int u, int v, int a) {
640         e[idx] = v, w[idx] = a, ne[idx] = h[u], h[u] = idx++;
641     }
642     void dfs(int u, int fu) {
643         for(int i = h[u]; ~i; i = ne[i]) {
644             int y = e[i];
645             if(y == fu) continue;
646             dist[y] = dist[u] + w[i];
647             dfs(y,u);
648         }
649     }
650 /**
651  * 树中节点分为三类:
652  * 1. 已经访问完毕并且回溯的节点。在这些节点上标记一个整数

```

```

653 * 2. 已经开始递归, 但尚未回溯的节点。这些节点就是当前正在访问的节点x以及x的祖先。标记为1
654 * 3. 尚未访问过的节点。没有标记。
655 * 对于正在访问的节点x, 它到根节点的路径已经标记为1. 若y是已经访问完毕并且回溯的节点, 则
    LCA(x,y) 就是y向上走到根, 第一个遇到的标记为1的节点。
656 */
657 void tarjan(int u) {
658     vis[u] = 1;
659     for(int i = h[u]; ~i; i = ne[i]) {
660         int y = e[i];
661         if(vis[y]) continue;
662         tarjan(y);
663         fa[y] = u;
664     }
665     for(auto t: ask[u]) {
666         int y = t.vf, id = t.vs;
667         if(vis[y] == 2) {
668             int lca = find(y); // 最近的公共祖先
669             // ans[id] = dist[u] + dist[y] - 2 * dist[anc]; // 求距离
670             ans[id] = lca;
671         }
672     }
673     vis[u] = 2;
674 }
675 void inpf();
676 void solve() {
677     memset(h, -1, sizeof(h));
678     int n,m; cin>>n>>m; int s; cin>>s;
679     for(int i = 1; i < n; ++i) {
680         int u,v,a; cin>>u>>v; a = 1;
681         add(u,v,a); add(v,u,a);
682     }
683     for(int i = 0; i < m; ++i) {
684         int u,v; cin>>u>>v;
685         if(u != v) {
686             ask[u].pb({v,i});
687             ask[v].pb({u,i});
688         } else ans[i] = u; // 如果求的是最近公共祖先, 重复的话就是本身咯
689     }
690     rep(i,1,n) fa[i] = i;
691     dfs(s,-1);
692     tarjan(s);
693     rep(i,0,m-1) cout<<ans[i]<<endl;
694 }
695
696 }
697
698 }
699 }
700
701 namespace TCS { // 树链剖分
702
703 /**
704 * url: https://www.luogu.com.cn/problem/solution/P3384
705 * 树链剖分的思想是:对于两个不在同一重链内的节点,让他们不断地跳,使得他们处于同一重链上
706 *
707 * 如何跳:
708 * 用第二次dfs中记录的top数组, ** x 到 top[x] 中的节点在线段树上是连续的。
709 * 结合dep数组即可。
710 *

```

```

711 * 选择x 和 y点dep较大的点开始跳（假设较大点是x），让x节点直接跳到 top[x]，然后在线段树上更新。
712 * 最后两个节点一定是处于同一条重链的，再直接在线段树上处理即可。
713 *
714 *
715 */
716 /* ----- */
717 const int N = 2e6 + 21;
718
719 // - `fa(x)`：表示节点`x`在树上的父亲
720 // - `dep(x)`：表示节点`x`在树上的深度
721 // - `siz(x)`：表示节点`x`的子树的节点个数
722 // - `son(x)`：表示节点`x`的重儿子
723 // - `top(x)`：表示节点`x`所在**重链**的顶部节点（深度最小）
724 // - `dfn(x)`：表示节点`x`的**DFS序**，也是其在线段树中的编号
725 // - `rnk(x)`：表示DFS序所对应的节点编号，有`rnk(dfn(x)) = x`
726
727 int fa[N], dep[N], siz[N], son[N], top[N], dfn[N], rnk[N];
728 int h[N], e[N], ne[N], w[N], dist[N], idx, cnt;
729 int p;
730 void inpfiler();
731
732 void add(int u, int v) {
733     e[idx] = v, ne[idx] = h[u], h[u] = idx++;
734 }
735 /* ----- 树链剖分 两次dfs ----- */
736
737 // 找出 fa dep siz son
738 void dfs1(int u) {
739     // if(dep[u])
740     son[u] = -1; // 重儿子设置为-1
741     siz[u] = 1; // 当前u节点大小为1（它本身）
742     for(int i = h[u]; ~i; i = ne[i]) {
743         int y = e[i];
744         if(y == fa[u]) continue; // **
745         if(!dep[y]) { // 如果深度没有，则可以接着往下遍历
746             dep[y] = dep[u] + 1; // 求出深度
747             fa[y] = u; // 为y设置父亲节点
748             dfs1(y); // 递归 y
749             siz[u] += siz[y]; // 当前节点u增加子节点个数
750             if(son[u] == -1 || siz[y] > siz[son[u]]) son[u] = y; // 更新重儿子
751         }
752     }
753 }
754
755 // 求出 top dfn rnk
756 void dfs2(int u, int t) {
757     top[u] = t; // 设置节点u的顶部节点为t
758     cnt++;
759     dfn[u] = cnt; // 在线段树中的编号
760     rnk[cnt] = u; // DFS序对应的节点编号
761     if(son[u] == -1) return; // 如果son[u] = -1，表示是叶子节点
762     dfs2(son[u], t); // 优先对重儿子进行DFS，保证同一条重链上的点DFS序连续
763     for(int i = h[u]; ~i; i = ne[i]) {
764         int y = e[i];
765         // 当不是u的重儿子，也不是u的父亲节点
766         // 那就是新的重链
767         if(y != son[u] && y != fa[u]) dfs2(y, y);
768     }
769 }

```

```

769     }
770 }
771
772 // 求lca
773 int lca(int u, int v) {
774     // 当两个点的重链顶点不一样时，表示是两个不同的重链
775     // 深度大的向上跳
776     // 跳到重链顶点的父亲节点
777     while(top[u] != top[v]) {
778         if(dep[ top[u]] > dep[ top[v]]) {
779             u = fa[ top[u]];
780         } else {
781             v = fa[ top[v]];
782         }
783     }
784     return dep[u] > dep[v] ? v : u;
785 }
786
787 /* ----- 线段树 [ 区间修改 区间求和 板子 ] -----
788 -----*/
789 // ( 裸线段树：树中点映射到线段树重
790 struct SegTree {
791     int l,r;
792     LL sum, add;
793 }tr[N << 2];
794 inline int ls(int u) {return u << 1; }
795 inline int rs(int u) {return u << 1 | 1; }
796 void pushup(int u) {
797     tr[u].sum = (tr[ls(u)].sum + tr[rs(u)].sum) % p;
798 }
799 void pushdown(int u) {
800     auto &root = tr[u], &left = tr[ls(u)], &right = tr[rs(u)];
801     if(root.add) {
802         left.add += root.add; left.sum += (left.r - left.l + 1) * root.add;
803         left.add %= p; left.sum %= p;
804         right.add += root.add; right.sum += (right.r - right.l + 1) * root.add;
805         right.add %= p; right.sum %= p;
806         root.add = 0;
807     }
808 }
809 void build(int u, int l, int r) {
810     if(l == r) tr[u] = {l,r,w[r],0};
811     else {
812         tr[u] = {l,r};
813         int mid = l + r >> 1;
814         build(ls(u), l, mid), build(rs(u), mid + 1, r);
815         pushup(u);
816     }
817 }
818 void modify(int u, int l, int r, int k) {
819     if(tr[u].l >= l && tr[u].r <= r) {
820         tr[u].add += k;
821         tr[u].add %= p;
822         tr[u].sum += (tr[u].r - tr[u].l + 1) * k;
823         tr[u].sum %= p;
824         return ;
825     }
826     pushdown(u);
827     int mid = tr[u].l + tr[u].r >> 1;
828     if(l <= mid) modify(ls(u),l,r,k);

```

```

828     if(r > mid) modify(rs(u), l, r,k);
829     pushup(u);
830 }
831 LL query(int u, int l, int r) {
832     if(tr[u].l >= l && tr[u].r <= r) {
833         return tr[u].sum;
834     }
835     int mid = tr[u].l + tr[u].r >> 1;
836     LL sum = 0;
837     pushdown(u);
838     if(l <= mid) sum = query(ls(u), l,r);
839     if(r > mid) sum += query(rs(u),l,r);
840     return sum;
841 }
842 /* ----- 树链剖分 -----
-----*/
843 // 求树 从 x 到 y 结点最短路径上所有节点的值之和
844 LL treesum(int x, int y) {
845     LL ans = 0;
846     // 如果x 和 y两个点对应重链顶点不一样，就向上跳
847     while(top[x] != top[y]) {
848         // 让 x 向上跳
849         if(dep[ top[x]] < dep[ top[y]]) swap(x,y);
850         // 查询这条重链的和
851         // dfn -- 对应 树中点在线段树中的映射
852         // top -- 对应重链顶点
853         ans = (ans + query(1,dfn[ top[x]], dfn[x])) % p;
854         // 让 x等于它重链顶点的父亲节点
855         x = fa[ top[x]];
856     }
857     // 让 x 在左边
858     if(dep[x] > dep[y]) swap(x,y);
859     // 处理 x 和 y 在同一条重链的区间和
860     ans = (ans + query(1, dfn[x], dfn[y])) % p;
861     return ans;
862 }
863
864 // 将树从 x 到 y 结点 最短路径上所有节点的值都加上k
865 // 同上 treeadd
866 void treeadd(int x, int y, int k) {
867     while(top[x] != top[y]) {
868         if(dep[ top[x]] < dep[ top[y]]) swap(x,y);
869         modify(1, dfn[ top[x]], dfn[x], k);
870         x = fa[ top[x]];
871     }
872     if(dep[x] > dep[y]) swap(x,y);
873     modify(1, dfn[x], dfn[y], k);
874 }
875 int a[N];
876 void solve() {
877     memset(h, -1, sizeof(h));
878     int n,m,r; cin>>n>>m>>r>>p;
879     for(int i = 1; i <= n; ++i) cin>>a[i];
880     for(int i = 1; i < n; ++i) {
881         int u,v; cin>>u>>v;
882         add(u,v), add(v,u);
883     }
884     /* ----- dfs * 2-----*/
885     dfs1(r);
886     dfs2(r,r);

```

```

887      /* ----- 将对应的在线段树中的位置 and 值进行设置 ----- */
888      for(int i = 1; i <= n; ++i) w[ dfn[i]] = a[i];
889      /* ----- 建树 ----- */
890      build(1,1,n);
891
892      /* ----- 查询 ----- */
893      while(m--) {
894          int opt; cin>>opt;
895          int x,y,z;
896          if(opt == 1) {
897              cin>>x>>y>>z;
898              treeadd(x,y,z);
899          } else if(opt == 2) {
900              cin>>x>>y;
901              cout<<treesum(x,y) % p<<endl;
902          } else if(opt == 3) {
903              cin>>x>>z;
904              // 以 x 为根节点的子树内所有节点值都加上z
905              modify(1, dfn[x], dfn[x] + siz[x] - 1, z);
906
907          } else {
908              cin>>x;
909              // 求以 x 为根节点的子树内所有的节点值之和
910              cout<<query(1, dfn[x], dfn[x] + siz[x] - 1) % p<<endl;
911          }
912      }
913  }
914  }
915
916  namespace golitter {
917      namespace block {
918          /**
919           *
920           * https://www.bilibili.com/video/BV1ms411t7xu/?spm\_id\_from=333.337.search-card.all.click&vd\_source=13dfbe5ed2deada83969fafa995ccff6
921           * 范围比线段树广，不必考虑区间可加性~
922           * belong 表示这个数在哪个块里面
923           * block 块大小
924           * lb 数所在的左边界
925           * rb 数所在的右边界
926           *
927          */
928          const int N = 2e5 + 21;
929          int belong[N], block, lb[N], rb[N], n, num;
930          void build() {
931              block = sqrt(n);
932              num = n / block; if(n%block) num++;
933              for(int i = 1; i <= num; ++i) { // 下标从1开始
934                  lb[i] = (i-1)*block + 1, rb[i] = min(i*block, n);
935              }
936              for(int i = 1; i <= n; ++i) {
937                  belong[i] = (i-1)/block + 1;
938              }
939          }
940      }
941  }
942
943  namespace golitter {
944      namespace st {
945

```



```
946  const int N = 1e5 + 21;
947  int st[N][25];
948  int a[N], n, m;
949  int mn[N];
950
951  void st_init() {
952      for(int i = 1; i <= n; ++i) {
953          st[i][0] = a[i];
954      }
955      for(int j = 1; (1<<j) <= n; ++j) {
956          for(int i = 1; i + (1 << j) - 1 <= n; ++i) {
957              st[i][j] = max(st[i][j-1], st[i + (1 << (j - 1))][j-1]);
958          }
959      }
960      for(int len = 1; len <= n; ++len) {
961          int k = 0;
962          while(1 <<(k+1) <= len) {
963              k++;
964          }
965          mn[len] = k;
966      }
967  }
968  int st_query(int l, int r) {
969      int k = mn[r - l + 1];
970      return max(st[l][k], st[r - (1<<k) + 1][k]);
971  }
972
973  }}
974
```