# Template For ICPC


Author: HKing

Email: 1470042308@qq.com

# 目录

# 1 数据结构

## 1.1 Trie 树

```
int son[N][26], cnt[N], idx;
// 0号点既是根节点，又是空节点
// son[][]存储树中每个节点的子节点
// cnt[]存储以每个节点结尾的单词数量

// 插入一个字符串
void insert(char *str) {
  int p = 0;
  for (int i = 0; str[i]; i++) {
    int u = str[i] - 'a';
    if (!son[p][u])
      son[p][u] = ++idx;
    p = son[p][u];
  }
  cnt[p]++;
}

// 查询字符串出现的次数
int query(char *str) {
  int p = 0;
  for (int i = 0; str[i]; i++) {
    int u = str[i] - 'a';
    if (!son[p][u])
      return 0;
    p = son[p][u];
  }
  return cnt[p];
}

/* ---------------------------------------- */
const int N=1e5+10;
struct Node{
    int ne[26];
    int cnt;
}node[N];
int idx;
// 插入字符串s
void Insert(string s){
    int p=0,i=0;
    while(i<s.length()){
        if(!node[p].ne[s[i]-'a'])
            node[p].ne[s[i]-'a']=++idx;
        p=node[p].ne[s[i]-'a'];
        ++i;
    }
    node[p].cnt++;
}
// 查询字符串s出现的次数
```

```
int query(string s){
   int p=0,i=0;
   while(i<s.length()){
      if(node[p].ne[s[i]-'a']==0)
         return 0;
      p=node[p].ne[s[i]-'a'];
      ++i;
   }
   return node[p].cnt;
}
```

## 1.2 treap

```
/*
 * @Author: ACCXavier
 * @Date: 2021-06-17 00:53:47
 * @LastEditTime: 2021-06-17 14:28:12
 * Bilibili:https://space.bilibili.com/7469540
 * 题目地址:https://www.acwing.com/problem/content/description/255/
 * @keywords: Treap 平衡树
 */
#include <iostream>
using namespace std;

const int N = 100010, INF = 1e8;

int n;

struct Node {
   int l, r; // 左右儿子
   int key; // 二叉搜索树权值
   int val; // 大根堆的随机权值
   int cnt; // 当前节点的key的重复个数
   int size; // 当前节点的子孙节点个数
} tr[N]; //空间O(N)

//Treap在以关键码构成二叉搜索树的同时，还满足堆的性质
//且堆的权重随机,这使得treap的期望复杂度是logn

int root, idx;// 根节点序号 和 序号

//更新父节点size信息,用儿子节点
void pushup(int p) {
   tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
}

//创建一个叶节点
int get_node(int key){
   tr[++ idx].key = key;
   tr[idx].val = rand();//随机值
   tr[idx].cnt = tr[idx].size = 1;//cnt,size
```

```
39      return idx;
40  }
41
42  //初始化平衡树 左右哨兵
43  //y总:如果查询的结果可能不存在，那加上哨兵之后可以保证查询的结果一定存在，
44  //就不需要在查询过程中特判无解的情况了。
45  void build(){
46      get_node(-INF),get_node(INF);
47      root = 1,tr[1].r = 2;//+inf > -inf,+inf在-inf右边
48      pushup(root);//更新root的size
49  }
50
51  //右旋
52  void zig(int &p){ //根变了,传引用
53  // 旋转的时候传root,root会变化,我们希望root还是真正的root,故用root
54  // p始终指向根
55
56
57      int q = tr[p].l; //q是左儿子
58      tr[p].l = tr[q].r,//p的左儿子是q的右儿子
59      tr[q].r = p,//q的右儿子是p
60      p = q;//p再变回根
61      pushup(tr[p].r),//更新p.r
62
63      pushup(p);//不需要更新p是因为看着右旋图,右旋之前y左是A+B,y右是C,旋过之后实际上y左是A,y右是B+C,
64      //但是不更新p,由于有 tr[q].r = p,相当于y的size由左侧的A+B和右侧的C构成,不影响最终size
65      //但是p的r要更新,因为r由B和C构成(r旋前只有C)
66
67  }
68
69  //左旋
70  void zag(int &p){
71      int q = tr[p].r;
72      tr[p].r = tr[q].l;//p的右儿子是q的左儿子
73      tr[q].l = p;//q的左儿子是p
74      p = q;//p再变回根
75      pushup(tr[p].l);
76      pushup(p);
77
78  }
79
80  //插入值key,从根开始
81
82  void insert(int &p,int key)//p是每一层根节点的指针
83  {
84      if(!p) p = get_node(key);// 不存在根,则构造(最底层时构造节点)
85      //由于这里是引用,传过来的是A节点的左或右,get_node之后A节点的左和右就是get_node的返回值idx,故完成了连接
86
87
88      else if (tr[p].key == key)tr[p].cnt ++; // 刚好key和p的key相等, 则直接增加cnt
89      else if (tr[p].key > key){ // 当前节点值大于key,说明应该在左子树插入
90          insert(tr[p].l,key);
91
```

```
92          //由于在左子树插入,插入左侧后左子树val可能大于根节点,左大右旋,保证堆的性质
93          if(tr[tr[p].l].val > tr[p].val) zig(p);
94      }
95      else{
96          insert(tr[p].r,key);
97          //右大左旋
98          if(tr[tr[p].r].val > tr[p].val) zag(p);
99      }
100     pushup(p);//p是每一层的根,指针,自底向上更新p
101 }
102
103 void remove(int &p,int key){
104     if(!p) return ;//不存在要删除的值
105     if(tr[p].key == key){ //要删除当前节点
106         if(tr[p].cnt > 1)tr[p].cnt --;
107         else if (tr[p].l || tr[p].r){ // 当前节点只有一个可以且有左儿子或右儿子
108         //注意rand函数>=0,左子树为空等价于idx = 0的点,其val为0
109
110             if(!tr[p].r||tr[tr[p].l].val > tr[tr[p].r].val){
111             //只存在左儿子(左val>右val_0)或左val>右val
112                 zig(p);//左val大右旋
113                 remove(tr[p].r,key);
114             }
115             else//若存在右儿子且左val<右val
116             //(左儿子也可能不存在,
                    不存在的话左儿子的val就是0,肯定<=右儿子的val(val最小为0),这个模板隐含了判存在操作)
117             {
118                 zag(p);//右大左旋
119                 remove(tr[p].l,key);
120             }
121         }
122         else //不存在左右子树,是叶子节点
123             p = 0;//空节点
124
125     }else if (tr[p].key > key)remove(tr[p].l,key);//去左侧删
126     else remove(tr[p].r,key);//右侧删
127     pushup(p);// 自底向上更新p的size
128 }
129
130 //没有修改,不需要引用
131
132 int get_rank_by_key(int p, int key) // 通过数值找排名
133 {
134     if (!p) return 0;
135     if (tr[p].key == key) return tr[tr[p].l].size + 1;//左子树的size +
          1(同样的数值中最靠左的)
136     if (tr[p].key > key)return get_rank_by_key(tr[p].l,key);//大了,去左子树找
137     //去右边找的时候找的是在右子树中的排名,需要加上左子树和根的cnt
138     return tr[tr[p].l].size + tr[p].cnt + get_rank_by_key(tr[p].r,key);
139 }
140
141 int get_key_by_rank(int p, int rank) // 通过排名找数值
142 {
```

```
143     if(!p) return INF;
144     if(tr[tr[p].l].size >= rank) return get_key_by_rank(tr[p].l,rank);
145     //左边的个数>=rank,说明数值在左边
146     if(tr[tr[p].l].size + tr[p].cnt >= rank)return
            tr[p].key;//左子树个数不够,加上当前cnt又多了,那就是当前数值
147     return get_key_by_rank(tr[p].r,rank - tr[tr[p].l].size - tr[p].cnt);
            //去右子树中找数值,排名应该先减去左子树size+cnt
148 }
149
150
151 int get_prev(int p, int key) // 找到严格小于key的最大数
152 {
153     if(!p) return -INF;
154     if(tr[p].key >= key) return get_prev(tr[p].l,key);//当前大于key,右子树不考虑
155     return max(tr[p].key,get_prev(tr[p].r,key));
            //当前key<key,不错,左子树都小于key所以不如key更好,所以考虑当前key和右子树
156
157 }
158 int get_next(int p, int key) // 找到严格大于key的最小数
159 {
160     if(!p) return INF;
161     if(tr[p].key <= key)return get_next(tr[p].r,key); //当前key小于key,左子树不考虑
162     return min(tr[p].key,get_next(tr[p].l,key));
            //当前key>key,不错,右子树都大于key所以不如key更好,所以考虑当前key和左子树
163 }
164
165
166
167 int main(){
168     build();
169     int n;
170     scanf("%d",&n);
171     while(n --){
172         int op,x;
173         scanf("%d%d",&op,&x);
174         if(op == 1) insert(root,x);
175         else if (op == 2) remove(root,x);
176         else if (op == 3) printf("%d\n",get_rank_by_key(root,x) -
                1);//查排名,有-INF,排名要-1
177         else if (op == 4) printf("%d\n",get_key_by_rank(root,x +
                1));//查排名为x的数值,有-INF,内部排名为x + 1
178         else if (op == 5) printf("%d\n",get_prev(root,x));
179         else printf("%d\n",get_next(root,x));
180     }
181     return 0;
182
183 }
```

## 1.3 Splay

```
1 const int N=1e5+10;
```

```
struct Node{
    int s[2],p,v;
    int size,flag;
    void init(int _v,int _p){
        v=_v,p=_p;
        size=1;
    }
}tr[N];
int root,idx;
int n,m;
void pushup(int u){
    tr[u].size=tr[tr[u].s[0]].size+tr[tr[u].s[1]].size+1;
}
void pushdown(int u){
    if(tr[u].flag){
        swap(tr[u].s[0],tr[u].s[1]);
        tr[tr[u].s[0]].flag^=1;
        tr[tr[u].s[1]].flag^=1;
        tr[u].flag=0;
    }
}
void rotate(int x){
    int y=tr[x].p,z=tr[y].p;
    int k=tr[y].s[1]==x;
    tr[z].s[tr[z].s[1]==y]=x,tr[x].p=z;
    tr[y].s[k]=tr[x].s[k^1],tr[tr[x].s[k^1]].p=y;
    tr[x].s[k^1]=y,tr[y].p=x;
    pushup(y),pushup(x);
}
void splay(int x,int k){
    while(tr[x].p!=k){
        int y=tr[x].p,z=tr[y].p;
        if(z!=k)
            if((tr[y].s[1]==x)^(tr[z].s[1]==y))
                rotate(x);
            else
                rotate(y);
        rotate(x);
    }
    if(!k)
        root=x;
}
void insert(int v){
    int u=root,p=0;
    while(u) p=u,u=tr[u].s[v>tr[u].v];
    u=++idx;
    if(p) tr[p].s[v>tr[p].v]=u;
    tr[u].init(v,p);
    splay(u,0);
}
int get_k(int k){
    int u=root;
    while(true){
```

```
55        pushdown(u);
56        if(tr[tr[u].s[0]].size>=k)u=tr[u].s[0];
57        else if(tr[tr[u].s[0]].size+1==k) return u;
58        else k-=tr[tr[u].s[0]].size+1,u=tr[u].s[1];
59    }
60    return -1;
61 }
62 void output(int u){
63    pushdown(u);
64    if(tr[u].s[0]) output(tr[u].s[0]);
65    if(tr[u].v>=1&&tr[u].v<=n)cout<<tr[u].v<<' ';
66    if(tr[u].s[1]) output(tr[u].s[1]);
67 }
68 int main()
69 {
70    IOS;
71    cin>>n>>m;
72    for(int i=0;i<=n+1;++i)insert(i);
73    while(m--){
74        int l,r;
75        cin>>l>>r;
76        l=get_k(l),r=get_k(r+2);
77        splay(l,0),splay(r,l);
78        tr[tr[r].s[0]].flag^=1;
79    }
80    output(root);
81    return 0;
82 }
```

## 1.4 fhqtreap

```
1  const int N=1e5+10;
2  int root,idx;
3  struct Node{
4     int l,r;
5     int v,key;
6     int size;
7     void init(int _v){
8        v=_v,key=rand();
9        size=1;
10    }
11 }tr[N];
12 void pushup(int u){
13    tr[u].size=tr[tr[u].l].size+tr[tr[u].r].size+1;
14 }
15 // 按值分裂(x中的节点的值都<=v)
16 void split(int u,int v,int &x,int &y){
17    if(!u)x=y=0;
18    else{
19       if(tr[u].v<=v){
20          x=u;
```

```
21          split(tr[u].r,v,tr[u].r,y);
22      }else{
23          y=u;
24          split(tr[u].l,v,x,tr[u].l);
25      }
26      pushup(u);
27   }
28}
29// 合并x与y
30int merge(int x,int y){
31   if(!x||!y)return x+y;
32   if(tr[x].key>tr[y].key){ //这里条件可以随便写，只要是随机就行
33      tr[x].r=merge(tr[x].r,y);
34      pushup(x);
35      return x;
36   }else{
37      tr[y].l=merge(x,tr[y].l);
38      pushup(y);
39      return y;
40   }
41}
42// 插入值为v的节点
43void insert(int v){
44   int x,y;
45   split(root,v,x,y);//按值v分裂为x、y两棵树
46   tr[++idx].init(v);//新节点
47   root=merge(merge(x,idx),y);//x与新节点合并，再与y合并
48}
49// 删除值为v的一个节点
50void del(int v){
51   int x,y,z;
52   split(root,v,x,z);//先按值v分裂为x与z
53   split(x,v-1,x,y);//再将x按值v-1分裂为x与y(y上节点值都为v)
54   y=merge(tr[y].l,tr[y].r);//合并y的左右子树=>删除y这个节点
55   root=merge(merge(x,y),z);
56}
57// 查询值为v的节点的排名
58int get_rk(int v){
59   int rk,x,y;
60   split(root,v-1,x,y);
61   rk=tr[x].size+1;
62   root=merge(x,y);
63   return rk;
64}
65// 查询排名为k的节点的编号
66int get_val(int k){
67   int u=root;
68   while(u){
69      if(tr[tr[u].l].size>=k)u=tr[u].l;
70      else if(tr[tr[u].l].size+1==k)return u;
71      else k-=tr[tr[u].l].size+1,u=tr[u].r;
72   }
73   return -1;
```

```
}
// 查询值为v的节点的前驱(编号)
int pre(int v){
    int u,x,y;
    split(root,v-1,x,y);
    u=x;
    while(tr[u].r)u=tr[u].r;
    root=merge(x,y);
    return u;
}
// 查询值为v的节点的后继(编号)
int nxt(int v){
    int u,x,y;
    split(root,v,x,y);
    u=y;
    while(tr[u].l)u=tr[u].l;
    root=merge(x,y);
    return u;
}
int main()
{
    IOS;
    int m;
    W(m){
        int t,x;
        cin>>t>>x;
        switch(t){
            case 1:
                insert(x);
                break;
            case 2:
                del(x);
                break;
            case 3:
                cout<<get_rk(x)<<'\n';
                break;
            case 4:
                cout<<tr[get_val(x)].v<<'\n';
                break;
            case 5:
                cout<<tr[pre(x)].v<<'\n';
                break;
            case 6:
                cout<<tr[nxt(x)].v<<'\n';
                break;
        }
    }
    return 0;
}
```

## 1.5 fhqtreap（区间操作）

```
const int N=1e5+10;
int root,idx;
struct Node{
    int l,r;
    int v,key;
    int size;
    int rev;// 区间操作的懒标记
    void init(int _v){
        v=_v,key=rand();
        size=1;
    }
}tr[N];
void pushup(int u){
    tr[u].size=tr[tr[u].l].size+tr[tr[u].r].size+1;
}
void pushdown(int u){
    if(tr[u].rev){
        tr[tr[u].l].rev^=1;
        swap(tr[tr[u].l].l,tr[tr[u].l].r);
        tr[tr[u].r].rev^=1;
        swap(tr[tr[u].r].l,tr[tr[u].r].r);
        tr[u].rev=0;
    }
}
// 按大小size分裂(x中包含前size个节点)
void split(int u,int size,int &x,int &y){
    if(!u)x=y=0;
    else{
        pushdown(u);// 分裂前下传懒标记
        if(tr[tr[u].l].size<size){
            x=u;
            split(tr[u].r,size-tr[tr[u].l].size-1,tr[u].r,y);
        }else{
            y=u;
            split(tr[u].l,size,x,tr[u].l);
        }
        pushup(u);
    }
}
// 合并x与y
int merge(int x,int y){
    if(!x||!y)return x+y;
    if(tr[x].key>tr[y].key){ //这里条件可以随便写，只要是随机就行
        pushdown(x);// 合并前下传懒标记
        tr[x].r=merge(tr[x].r,y);
        pushup(x);
        return x;
    }else{
        pushdown(y);// 合并前下传懒标记
        tr[y].l=merge(x,tr[y].l);
        pushup(y);
        return y;
```

```
    }
}
// 翻转区间[l,r]
void reverse(int l,int r){
    int x,y,z;
    split(root,l-1,x,y);// root按大小l-1分裂为x与y
    split(y,r-l+1,y,z);// y按大小r-l+1分裂为y与z ==> y就是所要操作的区间
    tr[y].rev^=1;
    swap(tr[y].l,tr[y].r);
    root=merge(merge(x,y),z);
}
// 插入值为v的节点
void insert(int v){
    int x,y;
    split(root,v,x,y);//按值v分裂为x、y两棵树
    tr[++idx].init(v);//新节点
    root=merge(merge(x,idx),y);//x与新节点合并，再与y合并
}
// 删除值为v的一个节点
void del(int v){
    int x,y,z;
    split(root,v,x,z);//先按值v分裂为x与z
    split(x,v-1,x,y);//再将x按值v-1分裂为x与y(y上节点值都为v)
    y=merge(tr[y].l,tr[y].r);//合并y的左右子树=>删除y这个节点
    root=merge(merge(x,y),z);
}
// 查询值为v的节点的排名
int get_rk(int v){
    int rk,x,y;
    split(root,v-1,x,y);
    rk=tr[x].size+1;
    root=merge(x,y);
    return rk;
}
// 查询排名为k的节点的编号
int get_val(int k){
    int u=root;
    while(u){
        if(tr[tr[u].l].size>=k)u=tr[u].l;
        else if(tr[tr[u].l].size+1==k)return u;
        else k-=tr[tr[u].l].size+1,u=tr[u].r;
    }
    return -1;
}
// 查询值为v的节点的前驱(编号)
int pre(int v){
    int u,x,y;
    split(root,v-1,x,y);
    u=x;
    while(tr[u].r)u=tr[u].r;
    root=merge(x,y);
    return u;
}
```

```
// 查询值为v的节点的后继(编号)
int nxt(int v){
    int u,x,y;
    split(root,v,x,y);
    u=y;
    while(tr[u].l)u=tr[u].l;
    root=merge(x,y);
    return u;
}
void output(int u){
    pushdown(u);
    if(tr[u].l)output(tr[u].l);
    cout<<tr[u].v<<" ";
    if(tr[u].r)output(tr[u].r);
}
int main()
{
    IOS;
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;++i)
        insert(i);
    while(m--){
        int l,r;
        cin>>l>>r;
        reverse(l,r);
    }
    output(root);
    cout<<'\n';
    return 0;
}
```

## 1.6 树链剖分

```
int id[N],idx;
int fa[N],sz[N],dep[N],son[N],tp[N];
LL p[N],cnt[N],np[N],ncnt[N];
void dfs1(int u,int d){
    if(ne[u].size()==0)
        leaf.push_back({0,u});
    dep[u]=d,sz[u]=1;
    for(auto s: ne[u]){
        dfs1(s,d+1);
        sz[u]+=sz[s];
        if(sz[son[u]]<sz[s])son[u]=s;
    }
}
void dfs2(int u,int t){
    tp[u]=t,id[u]=++idx,ncnt[idx]=cnt[u],np[idx]=p[u];
    if(son[u])dfs2(son[u],t);
    for(auto s: ne[u]){
```

```
        if(s!=son[u])
            dfs2(s,s);
    }
}
LL query_path_sum(int u,int v){
    LL res=0;
    while(tp[u]!=tp[v]){
        if(dep[tp[u]]<dep[tp[v]])swap(u,v);
        res+=query_s(1,id[tp[u]],id[u]);
        u=fa[tp[u]];
    }
    if(u!=v){
        if(dep[u]<dep[v])swap(u,v);
        res+=query_s(1,id[v]+1,id[u]);
    }
    return res;
}
```

## 1.7 主席树

```
const int N=2e5+10;
int a[N],root[N],idx;
int n,m;
vector<int> v;
struct Node{
    int l,r,cnt;
}tr[4*N+N*16];
int get(int x){
    return lower_bound(v.begin(),v.end(),x)-v.begin();
}
void pushup(int u){
    tr[u].cnt=tr[tr[u].l].cnt+tr[tr[u].r].cnt;
}
int build(int l,int r){
    int p=idx++;
    tr[p].cnt=0;
    if(l==r)
        return p;
    int mid=l+r>>1;
    tr[p].l=build(l,mid);
    tr[p].r=build(mid+1,r);
    pushup(p);
    return p;
}
int insert(int p,int l,int r,int x){
    int q=idx++;
    tr[q]=tr[p];
    if(l==r){
        tr[q].cnt++;
        return q;
    }
```

```
    int mid=l+r>>1;
    if(x<=mid)
        tr[q].l=insert(tr[p].l,l,mid,x);
    else
        tr[q].r=insert(tr[p].r,mid+1,r,x);
    pushup(q);
    return q;
}
int query(int p,int q,int l,int r,int k){
    if(l==r)
        return l;
    int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;
    int mid=l+r>>1;
    if(cnt>=k)
        return query(tr[p].l,tr[q].l,l,mid,k);
    else
        return query(tr[p].r,tr[q].r,mid+1,r,k-cnt);
}
int main()
{
    IOS;
    cin>>n>>m;
    for(int i=1;i<=n;++i){
        cin>>a[i];
        v.push_back(a[i]);
    }
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    root[0]=build(0,v.size()-1);
    for(int i=1;i<=n;++i)
        root[i]=insert(root[i-1],0,v.size()-1,get(a[i]));
    while(m--){
        int l,r,k;
        cin>>l>>r>>k;
        cout<<v[query(root[l-1],root[r],0,v.size()-1,k)]<<'\n';
    }
    return 0;
}
```

## 1.8 线段树分裂

```
const int N=2e5+10;
int idx,root[N];
struct Node{
    int l,r;
    LL v;
}tr[N];
void pushup(int u){
    tr[u].v=tr[tr[u].l].v+tr[tr[u].r].v;
}
void modify(int l,int r,int& u,int idx,int x){
```

```
    if(!u)u=++idx;
    tr[u].v+=x;
    if(l==r)return;
    int mid=l+r>>1;
    if(idx<=mid)modify(l,mid,tr[u].l,idx,x);
    else modify(mid+1,r,tr[u].r,idx,x);
    // if(l==r){
    // tr[u].v+=x;
    // return;
    // }
    // int mid=l+r>>1;
    // if(idx<=mid)modify(l,mid,tr[u].l,idx,x);
    // else modify(mid+1,r,tr[u].r,idx,x);
    // pushup(u);
}
// x与y合并==>x
void merge(int &x,int y){
    if(!x||!y)x|=y;
    else{
        tr[x].v+=tr[y].v;
        merge(tr[x].l,tr[y].l);
        merge(tr[x].r,tr[y].r);
    }
}
// 将k分裂出区间[x,y]
int split(int l,int r,int &k,int x,int y){
    int u=++idx;
    if(x<=l&&y>=r){
        tr[u]=tr[k];
        k=0;
    }else{
        int mid=l+r>>1;
        if(x<=mid)
            tr[u].l=split(l,mid,tr[k].l,x,y);
        if(y>r)
            tr[u].r=split(mid+1,r,tr[k].r,x,y);
        pushup(k),pushup(u);
    }
    return u;
}
LL query(int l,int r,int u,int x,int y){
    if(x<=l&&y>=r)
        return tr[u].v;
    int mid=l+r>>1;
    LL res=0;
    if(x<=mid)
        res+=query(l,mid,tr[u].l,x,y);
    if(y>mid)
        res+=query(mid+1,r,tr[u].r,x,y);
    return res;
}
LL kth(int l,int r,int u,int k){
    if(l==r)
```

```
64          return u;
65      int mid=l+r>>1;
66      if(k<=tr[tr[u].l].v)
67          return kth(l,mid,tr[u].l,k);
68      else
69          return kth(mid+1,r,tr[u].r,k-tr[tr[u].l].v);
70  }
71  int main()
72  {
73      // IOS;
74      int n,m;
75      cin>>n>>m;
76      for(int i=1;i<=m;++i){
77          int x;
78          cin>>x;
79          modify(1,n,root[1],i,x);
80      }
81      int last=1;
82      while(m--){
83          int t,x,y,z;
84          cin>>t>>x>>y;
85          switch(t){
86              case 0:
87                  cin>>z;
88                  root[++last]=split(1,n,root[x],y,z);
89                  break;
90              case 1:
91                  merge(root[x],root[y]);
92                  break;
93              case 2:
94                  cin>>z;
95                  modify(1,n,root[x],z,y);
96                  break;
97              case 3:
98                    cin>>z;
99                    cout<<query(1,n,root[x],y,z)<<'\n';
100                   break;
101           case 4:
102                   if(y>tr[root[x]].v)cout<<-1<<'\n';
103                   else cout<<kth(1,n,root[x],y)<<'\n';
104                   break;
105         }
106     }
107     return 0;
108 }
```

## 1.9 LCA

```
1  void bfs(int root) // 预处理倍增数组
2  {
3      memset(depth, 0x3f, sizeof depth);
```

```
    depth[0] = 0, depth[root] = 1; // depth存储节点所在层数
    int hh = 0, tt = 0;
    q[0] = root;
    while (hh <= tt)
    {
        int t = q[hh ++ ];
        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            if (depth[j] > depth[t] + 1)
            {
                depth[j] = depth[t] + 1;
                q[ ++ tt] = j;
                fa[j][0] = t; // j的第二次幂个父节点
                for (int k = 1; k <= 15; k ++ )
                    fa[j][k] = fa[fa[j][k - 1]][k - 1];
            }
        }
    }
}

int lca(int a, int b) // 返回a和b的最近公共祖先
{
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 15; k >= 0; k -- )
        if (depth[fa[a][k]] >= depth[b])
            a = fa[a][k];
    if (a == b) return a;
    for (int k = 15; k >= 0; k -- )
        if (fa[a][k] != fa[b][k])
        {
            a = fa[a][k];
            b = fa[b][k];
        }
    return fa[a][0];
}
```

# 2 其他

## 2.1 二分

```
bool check(int x) {/* ... */} // 检查x是否满足某种性质

// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid; // check()判断mid是否满足性质
        else l = mid + 1;
```

```
    }
    return l;
}
// 区间[l, r]被划分成[l, mid − 1]和[mid, r]时使用:
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid − 1;
    }
    return l;
}
```

## 2.2 template

```
/*************************************************************************
 > Author: HKing
 > Mail: 1470042308@qq.com
*************************************************************************/

#include <iostream>
#include <algorithm>
#include <cmath>
#include <string>
#include <map>
#include <unordered_map>
#include <cstring>
#include <vector>
#include <queue>
#include <stack>
#include <set>
#define IOS ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
#define re register
#define endl '\n'
#define out(n) cout<<n<<' '
#define outl(n) cout<<n<<endl
#define sd(n) scanf("%d", &n)
#define sdd(n, m) scanf("%d%d", &n, &m)
#define sddd(n, m, k) scanf("%d%d%d", &n, &m, &k)
#define pd(n) printf("%d\n", (n))
#define pdd(n, m) printf("%d %d\n", n, m)
#define pddd(n, m, k) printf("%d %d %d\n", n, m, k)
#define sld(n) scanf("%lld", &n)
#define sldd(n, m) scanf("%lld%lld", &n, &m)
#define slddd(n, m, k) scanf("%lld%lld%lld", &n, &m, &k)
#define pld(n) printf("%lld\n", n)
#define pldd(n, m) printf("%lld %lld\n", n, m)
#define plddd(n, m, k) printf("%lld %lld %lld\n", n, m, k)
#define sf(n) scanf("%lf", &n)
```

```
35 #define sff(n, m) scanf("%lf%lf", &n, &m)
36 #define sfff(n, m, k) scanf("%lf%lf%lf", &n, &m, &k)
37 #define ss(str) scanf("%s", str)
38 #define ps(str) printf("%s", str)
39 #define x first
40 #define y second
41 #define lc u<<1
42 #define rc u<<1|1
43 #define pi acos(-1)
44 #define de(c, n) \
45     for (int i = 0; i < n; ++i) \
46         cout << c; \
47     cout << endl
48 #define debug(a) cout << #a << '=' << a << endl
49 #define INF_INT 0x3f3f3f3f
50 #define INF_LONG 4557430888798830399
51 #define mem(ar, num) memset(ar, num, sizeof(ar))
52 #define me(ar) memset(ar, 0, sizeof(ar))
53 #define all(v) v.begin(),v.end()
54 #define max3(a,b,c) max(a,max(b,c))
55 #define min3(a,b,c) min(a,min(b,c))
56 #define lowbit(x) (x & (-x))
57 #define gcd(a, b) __gcd(a, b)
58 #define lcm(a, b) a / gcd(a, b) * b
59 #define qpow(a, k, p) ({LL s = 1; while(k > 0) {if (k & 1)s = s * a % p; a = a * a
      % p; k >>= 1;} s; })
60 #define inv(a,p) ({ LL q=p-2;qpow(a,q,p);})
61 #define W(t) cin >> t; while(t--)
62 using namespace std;
63 typedef long long LL;
64 typedef unsigned long long ULL;
65 typedef pair<int, int> PII;
66 typedef pair<int, PII> PIII;
67 typedef pair<LL, LL> PLL;
68 typedef pair<LL, PLL> PLLL;
69
70 int main()
71 {
72     IOS;
73
74     return 0;
75 }
```