

```

1 #####
2 #
3 #           データ構造
4 #
5 #####
6
7 class UnionFind:
8     def __init__(self,N): # 頂点数 N
9         self.table = [i for i in range(N)] # 親 table[x] == x で根
10        self.rank  = [1 for i in range(N)] # 木の長さ
11        self.size  = [1 for i in range(N)] # 集合のサイズ
12
13    def Find(self,x): #xの根を返す
14        if self.table[x] == x:
15            return x
16        else:
17            self.table[x] = self.Find(self.table[x]) #親の更新
18            self.size[x] = self.size[self.table[x]]
19            return table[x]
20
21    def Unite(self,x,y,w): #xとyをdiff(x,y)=W で繋げる
22        w = w - self.weight(y) + self.weight(x)
23        x,y = self.Find(x), self.Find(y)
24        sx,sy = self.Size(x), self.Size(y)
25        if x == y: return
26        if self.rank[x] > self.rank[y]:
27            self.table[y] = x
28            self.size[x] = sx + sy
29        else:
30            self.table[x] = y
31            self.size[y] = sx + sy
32            if self.rank[x] == self.rank[y]:
33                self.rank[y] += 1
34
35    def Check(self,x,y):
36        return self.Find(x) == self.Find(y)
37
38    def Size(self,x):
39        return self.size[self.Find(x)]
40
41
42
43 class BinaryIndexedTree():
44     def __init__(self,N):
45         self.N = N
46         self.bit = [0]*(self.N+1)
47
48     def add(self,a,w):
49         x = a
50         while x <= self.N:
51             self.bit[x] += w
52             x += x & -x
53
54     def sum(self,a):
55         tmp = 0
56         x = a
57         while x > 0:
58             tmp += self.bit[x]
59             x -= x & -x
60         return tmp
61
62
63
64 class SegmentTree:
65     def __init__(self,N,d):
66         self.NN = 1
67         while self.NN < N:
68             self.NN *= 2
69         self.SegTree = [d]*(self.NN*2-1)
70
71     def update(self,i,x): #iの値をxに更新
72         i += self.NN - 1

```

```

73         self.SegTree[i] = x
74         while i>0:
75             i = (i-1)//2
76             self.SegTree[i] = self.process(self.SegTree[i*2+1],self.SegTree[i*2+2])
77
78     def query(self,a,b,k=0,l=0,r=None): #[A,B]の値, 呼ぶときはquery(a,b)
79         if r == None: r = self.NN
80         if r <= a or b <= l: #完全に含まない
81             return INF
82         elif a <= l and r <= b : #完全に含む
83             return self.SegTree[k]
84         else: #交差する
85             vl = self.query(a,b,k*2+1,l,(l+r)//2)
86             vr = self.query(a,b,k*2+2,(l+r)//2,r)
87             return(self.process(vl,vr))
88
89     def process(self,x,y): #x,yが子の時, 親に返る値
90         return min(x,y)
91
92
93
94     class PotentialUnionFind:
95         def __init__(self,N): # 頂点数 N
96             self.table = [i for i in range(N)] # 親 table[x] == x で根
97             self.rank = [1 for i in range(N)] # 木の長さ
98             self.size = [1 for i in range(N)] # 集合のサイズ
99             self.diffweight = [0 for i in range(N)]
100
101         def Find(self,x): #xの根を返す
102             if self.table[x] == x:
103                 return x
104             else:
105                 root = self.Find(self.table[x]) #親の更新
106                 self.size[x] = self.size[self.table[x]]
107                 self.diffweight[x] += self.diffweight[self.table[x]]
108                 self.table[x] = root
109                 return root
110
111         def Unite(self,x,y,w): #xとyをDiff(x,y)=W で繋げる
112             w = w - self.Weight(y) + self.Weight(x)
113             x,y = self.Find(x), self.Find(y)
114             sx,sy = self.Size(x), self.Size(y)
115             if x == y: return
116             if self.rank[x] > self.rank[y]:
117                 self.table[y] = x
118                 self.size[x] = sx + sy
119                 self.diffweight[y] = w
120             else:
121                 self.table[x] = y
122                 self.size[y] = sx + sy
123                 self.diffweight[x] = -w
124                 if self.rank[x] == self.rank[y]:
125                     self.rank[y] += 1
126
127         def Check(self,x,y):
128             return self.Find(x) == self.Find(y)
129
130         def Size(self,x):
131             return self.size[self.Find(x)]
132
133         def Weight(self,x): # 重さ(根からの距離)
134             self.Find(x)
135             return self.diffweight[x]
136
137         def Diff(self,x,y): # 繋がってる二点間距離
138             return self.Weight(y) - self.Weight(x)
139
140
141
142
143
144
145
146

```

```

147
148 # 優先度付きキュー
149 class HeapQueue:
150     def __init__(self, flag): # flag == true 最小値pop / False 最大値pop
151         if flag: self.inv = 1
152         else: self.inv = -1
153         self.hq = []
154         heapq.heapify(self.hq)
155
156     def push(self, x):
157         heapq.heappush(self.hq, self.inv*x)
158
159     def pop(self):
160         return self.inv * heapq.heappop(self.hq)
161
162
163 # 動的中央値管理
164 class DynamicMedian:
165     def __init__(self):
166         self.Hq = HeapQueue(True) # Lqより多く
167         self.Hv = None
168         self.Lv = None
169         self.Lq = HeapQueue(False)
170         self.N = 0
171
172     def pop(self, x):
173         if self.N == 0:
174             self.Hv = x
175         elif self.N == 1:
176             if x <= self.Hv:
177                 self.Lv = x
178             else:
179                 self.Lv, self.Hv = self.Hv, x
180         elif self.N%2 == 0: # Hq と Lq が同数
181             if self.Hv <= x:
182                 self.Hq.push(x)
183             elif self.Lv < x < self.Hv:
184                 self.Hq.push(self.Hv)
185                 self.Hv = x
186             elif x <= self.Lv:
187                 self.Hq.push(self.Hv)
188                 self.Hv = self.Lv
189                 self.Lv = self.Lq.pop()
190                 self.Lq.push(x)
191         else: # Hq が一つ多い
192             if self.Hv <= x:
193                 self.Lq.push(self.Lv)
194                 self.Lv = self.Hv
195                 self.Hv = self.Hq.pop()
196                 self.Hq.push(x)
197             elif self.Lv < x < self.Hv:
198                 self.Lq.push(self.Lv)
199                 self.Lv = x
200             elif x <= self.Lv:
201                 self.Lv.push(x)
202         self.N += 1
203
204     def median(self):
205         if self.N%2 == 0:
206             return (self.Hv+self.Lv)/2
207         else:
208             return self.Hv
209

```