POJ3237 Tree

[POJ 3237 Tree （树链剖分）](http://www.cnblogs.com/kuangbin/p/3263822.html)

Tree

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 5000MS |  | **Memory Limit:** 131072K |
| **Total Submissions:** 2825 |  | **Accepted:** 769 |

Description

You are given a tree with *N* nodes. The tree’s nodes are numbered 1 through *N* and its edges are numbered 1 through *N* − 1. Each edge is associated with a weight. Then you are to execute a series of instructions on the tree. The instructions can be one of the following forms:

|  |  |
| --- | --- |
| CHANGE *i* *v* | Change the weight of the *i*th edge to *v* |
| NEGATE *a* *b* | Negate the weight of every edge on the path from *a* to *b* |
| QUERY *a* *b* | Find the maximum weight of edges on the path from *a* to *b* |

Input

The input contains multiple test cases. The first line of input contains an integer *t* (*t* ≤ 20), the number of test cases. Then follow the test cases.

Each test case is preceded by an empty line. The first nonempty line of its contains *N* (*N* ≤ 10,000). The next *N* − 1 lines each contains three integers *a*, *b* and c, describing an edge connecting nodes *a* and *b* with weight *c*. The edges are numbered in the order they appear in the input. Below them are the instructions, each sticking to the specification above. A lines with the word “DONE” ends the test case.

Output

For each “QUERY” instruction, output the result on a separate line.

Sample Input

1

3

1 2 1

2 3 2

QUERY 1 2

CHANGE 1 3

QUERY 1 2

DONE

Sample Output

1

3

Source

题目大意：指定一颗树上有3个操作：询问操作,询问a点和b点之间的路径上最长的那条边的长度；取反操作，将a点和b点之间的路径权值都取相反数；变化操作，把某条边的权值变成指定的值。

解体思路：这个思路很明显的路径剖分，完全就是来练写路径剖分的。

树链剖分+线段树实现

1 /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2 Author :kuangbin

3 Created Time :2013/8/17 4:04:42

4 File Name :F:\2013ACM练习\专题学习\数链剖分\POJ3237Tree.cpp

5 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

6

7 #include <stdio.h>

8 #include <string.h>

9 #include <iostream>

10 #include <algorithm>

11 #include <vector>

12 #include <queue>

13 #include <set>

14 #include <map>

15 #include <string>

16 #include <math.h>

17 #include <stdlib.h>

18 #include <time.h>

19 using namespace std;

20

21 const int MAXN = 100010;

22 struct Edge

23 {

24 int to,next;

25 }edge[MAXN\*2];

26 int head[MAXN],tot;

27 int top[MAXN];//top[v]表示v所在的重链的顶端节点

28 int fa[MAXN]; //父亲节点

29 int deep[MAXN];//深度

30 int num[MAXN];//num[v]表示以v为根的子树的节点数

31 int p[MAXN];//p[v]表示v与其父亲节点的连边在线段树中的位置

32 int fp[MAXN];//和p数组相反

33 int son[MAXN];//重儿子

34 int pos;

35 void init()

36 {

37 tot = 0;

38 memset(head,-1,sizeof(head));

39 pos = 0;

40 memset(son,-1,sizeof(son));

41 }

42 void addedge(int u,int v)

43 {

44 edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;

45 }

46 void dfs1(int u,int pre,int d) //第一遍dfs求出fa,deep,num,son

47 {

48 deep[u] = d;

49 fa[u] = pre;

50 num[u] = 1;

51 for(int i = head[u];i != -1; i = edge[i].next)

52 {

53 int v = edge[i].to;

54 if(v != pre)

55 {

56 dfs1(v,u,d+1);

57 num[u] += num[v];

58 if(son[u] == -1 || num[v] > num[son[u]])

59 son[u] = v;

60 }

61 }

62 }

63 void getpos(int u,int sp) //第二遍dfs求出top和p

64 {

65 top[u] = sp;

66 p[u] = pos++;

67 fp[p[u]] = u;

68 if(son[u] == -1) return;

69 getpos(son[u],sp);

70 for(int i = head[u] ; i != -1; i = edge[i].next)

71 {

72 int v = edge[i].to;

73 if(v != son[u] && v != fa[u])

74 getpos(v,v);

75 }

76 }

77

78 //线段树

79 struct Node

80 {

81 int l,r;

82 int Max;

83 int Min;

84 int ne;

85 }segTree[MAXN\*3];

86 void build(int i,int l,int r)

87 {

88 segTree[i].l = l;

89 segTree[i].r = r;

90 segTree[i].Max = 0;

91 segTree[i].Min = 0;

92 segTree[i].ne = 0;

93 if(l == r)return;

94 int mid = (l+r)/2;

95 build(i<<1,l,mid);

96 build((i<<1)|1,mid+1,r);

97 }

98 void push\_up(int i)

99 {

100 segTree[i].Max = max(segTree[i<<1].Max,segTree[(i<<1)|1].Max);

101 segTree[i].Min = min(segTree[i<<1].Min,segTree[(i<<1)|1].Min);

102 }

103 void push\_down(int i)

104 {

105 if(segTree[i].l == segTree[i].r)return;

106 if(segTree[i].ne)

107 {

108 segTree[i<<1].Max = -segTree[i<<1].Max;

109 segTree[i<<1].Min = -segTree[i<<1].Min;

110 swap(segTree[i<<1].Min,segTree[i<<1].Max);

111 segTree[(i<<1)|1].Max = -segTree[(i<<1)|1].Max;

112 segTree[(i<<1)|1].Min = -segTree[(i<<1)|1].Min;

113 swap(segTree[(i<<1)|1].Max,segTree[(i<<1)|1].Min);

114 segTree[i<<1].ne ^= 1;

115 segTree[(i<<1)|1].ne ^= 1;

116 segTree[i].ne = 0;

117 }

118 }

119 void update(int i,int k,int val) // 更新线段树的第k个值为val

120 {

121 if(segTree[i].l == k && segTree[i].r == k)

122 {

123 segTree[i].Max = val;

124 segTree[i].Min = val;

125 segTree[i].ne = 0;

126 return;

127 }

128 push\_down(i);

129 int mid = (segTree[i].l + segTree[i].r)/2;

130 if(k <= mid)update(i<<1,k,val);

131 else update((i<<1)|1,k,val);

132 push\_up(i);

133 }

134 void ne\_update(int i,int l,int r) // 更新线段树的区间[l,r]取反

135 {

136 if(segTree[i].l == l && segTree[i].r == r)

137 {

138 segTree[i].Max = -segTree[i].Max;

139 segTree[i].Min = -segTree[i].Min;

140 swap(segTree[i].Max,segTree[i].Min);

141 segTree[i].ne ^= 1;

142 return;

143 }

144 push\_down(i);

145 int mid = (segTree[i].l + segTree[i].r)/2;

146 if(r <= mid)ne\_update(i<<1,l,r);

147 else if(l > mid) ne\_update((i<<1)|1,l,r);

148 else

149 {

150 ne\_update(i<<1,l,mid);

151 ne\_update((i<<1)|1,mid+1,r);

152 }

153 push\_up(i);

154 }

155 int query(int i,int l,int r) //查询线段树中[l,r] 的最大值

156 {

157 if(segTree[i].l == l && segTree[i].r == r)

158 return segTree[i].Max;

159 push\_down(i);

160 int mid = (segTree[i].l + segTree[i].r)/2;

161 if(r <= mid)return query(i<<1,l,r);

162 else if(l > mid)return query((i<<1)|1,l,r);

163 else return max(query(i<<1,l,mid),query((i<<1)|1,mid+1,r));

164 push\_up(i);

165 }

166 int findmax(int u,int v)//查询u->v边的最大值

167 {

168 int f1 = top[u], f2 = top[v];

169 int tmp = -100000000;

170 while(f1 != f2)

171 {

172 if(deep[f1] < deep[f2])

173 {

174 swap(f1,f2);

175 swap(u,v);

176 }

177 tmp = max(tmp,query(1,p[f1],p[u]));

178 u = fa[f1]; f1 = top[u];

179 }

180 if(u == v)return tmp;

181 if(deep[u] > deep[v]) swap(u,v);

182 return max(tmp,query(1,p[son[u]],p[v]));

183 }

184 void Negate(int u,int v)//把u-v路径上的边的值都设置为val

185 {

186 int f1 = top[u], f2 = top[v];

187 while(f1 != f2)

188 {

189 if(deep[f1] < deep[f2])

190 {

191 swap(f1,f2);

192 swap(u,v);

193 }

194 ne\_update(1,p[f1],p[u]);

195 u = fa[f1]; f1 = top[u];

196 }

197 if(u == v)return;

198 if(deep[u] > deep[v]) swap(u,v);

199 return ne\_update(1,p[son[u]],p[v]);

200 }

201 int e[MAXN][3];

202 int main()

203 {

204 //freopen("in.txt","r",stdin);

205 //freopen("out.txt","w",stdout);

206 int T;

207 int n;

208 scanf("%d",&T);

209 while(T--)

210 {

211 init();

212 scanf("%d",&n);

213 for(int i = 0;i < n-1;i++)

214 {

215 scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);

216 addedge(e[i][0],e[i][1]);

217 addedge(e[i][1],e[i][0]);

218 }

219 dfs1(1,0,0);

220 getpos(1,1);

221 build(1,0,pos-1);

222 for(int i = 0;i < n-1; i++)

223 {

224 if(deep[e[i][0]] > deep[e[i][1]])

225 swap(e[i][0],e[i][1]);

226 update(1,p[e[i][1]],e[i][2]);

227 }

228 char op[10];

229 int u,v;

230 while(scanf("%s",op) == 1)

231 {

232 if(op[0] == 'D')break;

233 scanf("%d%d",&u,&v);

234 if(op[0] == 'Q')

235 printf("%d\n",findmax(u,v));//查询u->v路径上边权的最大值

236 else if(op[0] == 'C')

237 update(1,p[e[u-1][1]],v);//改变第u条边的值为v

238 else Negate(u,v);

239 }

240 }

241 return 0;

242 }