```
// Heavy Light Decomposition

void dfs(int u, int Parent) {
   parent[u] = Parent;            // Parent of u
   ChainSize[u] = 1;              // Number of child (initially the size is 1, contains only 1 node. itself) (resued
                                  // array in hld)
   for(int i = 0; i < SIZE(G[u]); ++i) {
      int v = G[u][i];
      if(v == Parent)                          // if the connected node is parent, skip
         continue;
      level[v] = level[u]+1;                   // level of the child node is : level of parent node + 1
      dfs(v, u);
      ChainSize[u] += ChainSize[v];            // Increment the child numbers
      if(nextNode[u] == -1 || ChainSize[v] > ChainSize[nextNode[u]])
         nextNode[u] = v;           // next selected node of u (select the node which has more child, (HEAVY))
   }
}

void hld(int u, int Parent) {
   chain[u] = ChainNo;               // Chain Number
   num[u] = all++;                   // Numbering all nodes

   if(ChainSize[ChainNo] == 0)          // if this is the first node
      top[ChainNo] = u;                 // mark this as the root node of the n'th chain

   ChainSize[ChainNo]++;

   if(nextNode[u] != -1)             // if this node has a child, go to it
      hld(nextNode[u], u);           // the next node is included in the chain

   for(int i = 0; i < SIZE(G[u]); ++i) {
      int v = G[u][i];
      if(v == Parent || v == nextNode[u])     // if this node is parent node or, this node is already included in
         continue;                            // the chain, skip
      ChainNo++;                               // this is a new (light) chain, so increment the chain no. counter
      hld(v, u);
   }
}


int GetSum(int u, int v) {
   int res = 0;
   while(chain[u] != chain[v]) {                        // While two nodes are not in same chain
      if(level[top[chain[u]]] < level[top[chain[v]]])   // u is the chain which's topmost node is deeper
         swap(u, v);
      int start = top[chain[u]];
      res += read(num[u]) - read(num[start]-1);         // Run query in u node's chain
      u = parent[start];                                // go to the upper chain of u
   }

   if(num[u] > num[v])
      swap(u, v);
   res += read(num[v]) – read(num[u]-1);               // Sum from node chain u to v
   return res;
}


void updateNodeVal(int u, int Val) {       // Modify node value
   update(num[u], -val[u]);               // Update value of chain (in which the node is)
```

```
    val[u] = Val;
    update(num[u], Val);
}
// Heavy light Decomposition End

int main() {
    int u, v, q, Val, t, c;
    sf("%d", &t);

    for(int Case = 1; Case <= t; ++Case) {
        sf("%d", &n);                       // number of nodes
        for(int i = 1; i <= n; ++i)         // value of each node
            sf("%d", &val[i]);
        for(int i = 1; i < n; ++i) {        // tree edges
            sf("%d %d", &u, &v);
            u++, v++;                       // node starts from 0 to n-1
            G[u].pb(v);
            G[v].pb(u);
        }

        MaxVal = n+1;
        memset(tree, 0, sizeof tree);
        memset(nextNode, -1, sizeof nextNode);
        ChainNo = 1, all = 1;
        dfs(1, 1);
        memset(ChainSize, 0, sizeof ChainSize);     // array reused in hld
        hld(1, 1);
        init(n);
        pf("Case %d:\n", Case);
        sf("%d", &q);                       // number of query

        while(q--) {
            sf("%d", &c);                   // 0 to query, 1 to update
            if(c == 0) {
                sf("%d %d", &u, &v);
                u++, v++;
                pf("%d\n", GetSum(u, v));
            }
            else {                          // assign new value to node u
                sf("%d %d", &u, &Val);
                u++;
                updateNodeVal(u, Val);
            }
        }

        for(int i = 1; i <= n; ++i)
            G[i].clear();
    }
    return 0;
}
```