Team Reference
Universidad Central de las Villas : KFP

Miembros del Equipo: Rafael Fernández Morera, Ruddy Guerrero Alvarez

Contents

1. Estructura de Datos

1.1. Arbol Binario Indexado.

```
# include <cstdio>
 using namespace std;

 int tm, op, p;
 typedef long long ll;

 struct date{
   int save[10005];

   void update(int p, ll v){
      for(int i = p; i <= tm; i += i& -i)
         save[i] += v;
   }

   ll query(int p){
      int sum=0;
      for(int i = p; i > 0; i -= (i & -i))
         sum += save[i];
      return sum;
   }
}bit;

int main(){

  scanf("%d", &tm);
  while(1){

     scanf("%d %d", &op, &p);

     if(op == -1)
        return 0;

     if(op)
        bit.update(p);
     else
        bit.print(p);
  }

}
```

1.2. Segment Tree.

```
# include <iostream>
 # include <algorithm>
 # define oo 1 << 29
 # define RANG 30000000
 using namespace std;

 char c;
 int r1, r2, r3, i, Q;

 struct S_Tree{
   int n;
   int elements[5005];
   int T[RANG], Mk[RANG];

   int Build(int x, int xend, int P = 1){

      if(x == xend)
         return T[P] = elements[x];

      int pv = (x+xend)/2;
      return T[P] = Build(x, pv, P*2) + Build(pv+1, xend, P*2+1);
   }

   void Lazy_propagation(int x, int xend, int P){
      if(x == xend)
         return;

      int pv = (x+xend)/2;

      T[P*2] += (pv - x + 1)* Mk[P];
      T[P*2+1] += (xend - pv )* Mk[P];

      Mk[P*2] += Mk[P];
      Mk[P*2+1] += Mk[P];

      Mk[P] = 0;
```

```
   }

   int Query(int x, int xend, int P = 1){

      if(r2 < x || xend < r1)
         return 0;

      if(Mk[P])
         Lazy_propagation(x, xend, P);

      if(r1 <= x && xend <= r2)
         return T[P];

      int pv = (x+xend)/2;
      return Query(x, pv, P*2) + Query(pv+1, xend, P*2+1);
   }

   int Update(int x, int xend, int P = 1){

      if(Mk[P])
         Lazy_propagation(x, xend, P);

      if(r2 < x || xend < r1)
         return T[P];

      if(r1 <= x && xend <= r2){
         Mk[P] += r3;
         T[P] += ((xend-x)+1)*r3;
         return T[P];
      }

      int pv = (x+xend)/2;
      return T[P] = Update(x, pv, P*2) + Update(pv+1, xend, P*2+1);
   }
}St;

int main(){
   cin >> St.n;
   for(i = 1; i <= St.n; i++)
      cin >> St.elements[i];
  St.Build(1, St.n);
   cin >> Q;
   while(Q--){
      cin >> c >> r1 >> r2;
      if(c == 'Q')
         cout << St.Query(1, St.n) << endl;
      else{
         cin >> r3;
         St.Update(1, St.n);
      }
   }

return 0; }
```

1.3. **Range Min-Max Quering.**

```
# include <cstdio>
 # include <cmath>
 # include <algorithm>
 using namespace std;

 int mat[5005][20];
 int n, p2, p1, q;

 void Build_RMQ(){

   int cc = (int) log2(n);
   int p = n, a, i, j;
   for(i = 1; i <= cc; i++){
      a = 1 << (i-1);
      p -= a;
      for(j = 1; j <= p; j++)
         mat[j][i] = min(mat[j][i-1], mat[j+a][i-1]);
   }
}

void find_RMQ(){
     int c = (int)log2(p2-p1);
     printf("%d\n", min(mat[p1][c], mat[p2-(1<<c)+1][c]));
}

int main(){


   scanf("%d %d", &n, &q);
   for(int i = 1; i <= n; i++)
      scanf("%d", &mat[i][0]);
```

```
   Build_RMQ();

   while(q--){
      scanf("%d %d", &p1, &p2);
      find_RMQ();
   }

 return 0;
}
```

1.4. Lowest Comon Antecesor.

```
# include <bits/stdc++.h>
 # define RANG 1000005
 using namespace std;

 int i, cn, q, x, y;
 vector <int> v[RANG];

 struct LCA {
   int T[100005][20], L[100005];

   void DFS(int np, int prev){
      L[np] = L[prev]+1;
      int l = v[np].size();
      for(int i = 0; i < l; i++){
         int nh = v[np][i];
         if(nh != prev)
            DFS(nh, np);
      }
    }

   void BFS(int np){
      queue <int> Q;
      Q.push(np);
      L[np] = 1;
      int l, nh;
      while(!Q.empty()){
         np = Q.front();
         Q.pop();

         l = v[np].size();
         for(int i = 0; i < l; i++){
            nh = v[np][i];
            if(L[nh] == 0){
               L[nh] = L[np]+1;
               Q.push(nh);
            }
         }
      }

   }

   void Build(int n){
      BFS(1);
      int lg = log2(n);
      for(int j = 1; j <= lg; j++)
         for(int i = 1; i <= n; i++)
            if(T[i][j-1] != -1)
               T[i][j] = T[T[i][j-1]][j-1];
   }

    int Query(int x, int y){
      int sol = 0;
      if(L[x] < L[y])swap(x, y);

      int lg = (int)log2(L[x]);
      for(int i = lg; i >= 0; i--)
         if(L[x] - (1 << i) >= L[y] && T[x][i])
            x = T[x][i], sol += (1 << i);

      if(x == y)return sol;

      for(int i = lg; i >= 0; i--)
         if(T[x][i] != T[y][i] && T[x][i])
            x = T[x][i], y = T[y][i], sol += (1 << i);

      return sol+2;
      return T[x][0];
    }
}Lc;

int main(){

   scanf("%d", &cn);
   for(i = 2; i <= cn; i++){//Leyendo padre
      scanf("%d", &Lc.T[i][0]);
      v[Lc.T[i][0]].push_back(i);
   }
```

```
   Lc.Build(cn);
   scanf("%d", &q);
   while(q--){
      scanf("%d %d", &x, &y);
      printf("%d\n", Lc.Query(x, y));
   }
```

1.5. Heavy Ligth Descomposition+Segmente Tree+Lowest Common Antecesor.

```
# include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> par;

vector <par> v[10005];
vector <int> indx[10005];
int subsize[10005], chainHead[10005], chainIndx[10005];
int posInBase[10005], otherEnd[10005], chainNo, cont;

St -> estructura segment tree. Build-Query-Update+Lazy Propagation
LC -> Lowest Common Antecesor. Level[n], T[n][log n]. Build-Query
//Inicializar Level y subsize
 void DFS(int np, int prev, int depth = 0){
   Lc.Level[np] = depth;
   Lc.T[np][0] = prev;
   subsize[np] = 1;
   int l = v[np].size();
   for(int i = 0; i < l; i++){
      int nh = v[np][i].first;
      if(nh != prev){
         otherEnd[indx[np][i]] = nh;
         DFS(nh, np, depth+1);
         subsize[np] += subsize[nh];
      }
   }
 }
//Descomposition Hevy Ligth
 void HDL(int np, int nc, int prev){
   if(chainHead[chainNo] == -1)
      chainHead[chainNo] = np;

   chainIndx[np] = chainNo;
   posInBase[np] = cont;
Posicion que sera usada en el Segment Tree
   St.elements[cont++] = nc;

   int nh = -1, newc, l = v[np].size();
   for(int i = 0; i < l; i++){
      if(v[np][i].first == prev)continue;
      if(nh == -1 || subsize[nh] < subsize[v[np][i].first]){
         nh = v[np][i].first;
         newc = v[np][i].second;
      }
   }
   if(nh != -1)
      HDL(nh, newc, np);

   for(int i = 0; i < l; i++)
      if(nh != v[np][i].first && v[np][i].first != prev){
         chainNo++;
         HDL(v[np][i].first, v[np][i].second, np);
      }

}
 int query_up(int u, int v){
   int uchain = chainIndx[u], vchain = chainIndx[v], ans = -1;

   while(uchain != vchain){
      ans = max(ans, St.query(0, cont-1, 1, posInBase[chainHead[uchain]],
                              posInBase[u]));
      u = Lc.T[chainHead[uchain]][0];
      uchain = chainIndx[u];
   }

   ans = max(ans, St.query(0, cont-1, 1, posInBase[v]+1, posInBase[u]));

   return ans;
}
int query(int x, int y){
   int lca = Lc.Query(x, y);
   return max(query_up(x, lca), query_up(y, lca));
}
void update(int i, int val){
   int x = otherEnd[i];
   x = posInBase[x];
   St.elements[x] = val;
   St.update(0, cont-1, 1, x);
}
```

```
int n, i, a, b, c, tc;
char arr[50];

int main(){
     scanf("%d", &n);
     cont = 0;
     for(i = 1; i < n; i++){
        scanf("%d %d %d", &a, &b, &c);
        v[a].push_back((par){b, c});
        v[b].push_back((par){a, c});
        indx[a].push_back(i);
        indx[b].push_back(i);
     }

     fill(chainHead, chainHead+10002, -1);
     chainNo = 0;
     DFS(1, -1);
     HDL(1, -1, -1);
     St.Build(0, cont-1);
     Lc.Build(n);
}
```

1.6. Centroid Descomposition+Lowest Common Antecesor.

```
# include <bits/stdc++.h>
using namespace std;

const int oo = 1 << 30;
int subsize[100005], Ant[100005], sol[100005], ref_pos, n, x, y;
vector <int> v[100005];
bool mk[100005];

void DFS1(int np, int prev){
  subsize[np] = 1;
  int l = v[np].size();
  for(int i = 0; i < l; i++){
     int nh = v[np][i];
     if(nh != prev && !mk[nh]){
        DFS1(nh, np);
        subsize[np] += subsize[nh];
     }
  }
}

int DFS2(int np, int prev){

   int l = v[np].size();
   for(int i = 0; i < l; i++){
     int nh = v[np][i];
     if(nh != prev && !mk[nh] && subsize[nh] > subsize[ref_pos]/2)
        return DFS2(nh, np);
   }
   return np;
}

void Descomposition(int root, int prev){
  ref_pos = root;
  DFS1(root, root);
  int centroid = DFS2(root, root);
  Ant[centroid] = prev;
  mk[centroid] = true;
  int l = v[centroid].size();
  for(int i = 0; i < l; i++){
     int nh = v[centroid][i];
     if(!mk[nh])
        Descomposition(nh, centroid);
  }
}
// LC -> tipo LCA, buscar implementacion arriba.

void Update(int x){
  int y = x;
  while(y > 0){
     sol[y] = min(sol[y], Lc.Query(x, y));
     y = Ant[y];
  }
}

int Query(int x){
  int y = x, ans = oo;
  while(y > 0){
     ans = min(ans, Lc.Query(y, x)+sol[y]);
     y = Ant[y];
  }
  return ans;
}
int Q;
```

```
int main(){

  scanf("%d %d", &n, &Q);
  for(int i = 1; i < n; i++){
     scanf("%d %d", &x, &y);
     v[x].push_back(y);
     v[y].push_back(x);
  }

  fill(sol, sol+n+1, oo);
  Lc.Build(n);
  Descomposition(1, -1);
  Update(1);

  while(Q--){
     scanf("%d %d", &x, &y);
     if(x == 1)
        Update(y);
     else
        printf("%d\n", Query(y));

  }
return 0;
}
```

1.7. Trie.

```
# include <cstdio>
# include <cstring>
using namespace std;

int n, q, i, j, ls, sol;
char s[505];

struct Trie{
  Trie *son[255];
  int end;
}T, *p = &T;

int main(){

  scanf("%d", &n);
  for(i = 1; i <= n; i++){
     scanf("%s", &s);
     ls = strlen(s);
     p = &T;
     for(j = 0; j < ls; j++){
        if(p -> son[s[j]] == NULL)
           p -> son[s[j]] = new Trie();
        p = p -> son[s[j]];
     }
  }


scanf("%d", &q);
  for(i = 1; i <= q; i++){
     scanf("%s", &s);
     ls = strlen(s);
     p = &T;
     for(j = 0; j < ls; j++){
        if(p -> son[s[j]] == NULL)
           break;
        p = p -> son[s[j]];
        if(j == ls-1)
           sol++;
     }
  }

  printf("%d", sol);
return 0;
}
```