GetCodes : Part 1.1

Brain Craft Ltd - SUB Inter University Programming Contest 2019

Aaman007

METROPOLITAN UNIVERSITY , Sylhet

Table of Contents

***************** Data Structure *****************

Array Compression / Mapping

Problem Name : **Points in Segments (II)**

Problem Link : http://lightoj.com/volume_showproblem.php?problem=1089

Solution :

```
using namespace std;
int main()
{
    int t, n , q , l , r , cas = 1;

    scanf("%d", &t);

    while(t--)
    {
        scanf("%d%d",&n,&q);

        map<int,int>cum;
        int mx = 0 , ans[q+2];

        for(int i=0;i<n;i++)
        {
            scanf("%d%d",&l,&r);
            cum[l]++;
            cum[r+1]--;
        }

        for(int i=0;i<q;i++)
        {
            scanf("%d",&ans[i]);
            cum[ans[i]] += 0;
        }
        int sum = 0;


        for(auto &i : cum)
        {
            sum += i.second;
            i.second = sum;
        }

        printf("Case %d:\n",cas++);
        for(int i=0;i<q;i++)
            printf("%d\n",cum[ans[i]]);
    }
}
```

Binary Indexed Tree / BIT / Fenwick Tree

Problem Name : Curious Robin Hood

Problem Link : http://lightoj.com/volume_showproblem.php?problem=1112

Solution :

```
const int MAX = 1e5+10;
int tree[MAX];

void update(int idx,int x,int n)
{
    while(idx <= n)
    {
        tree[idx] += x;
        idx += idx&(-idx);
    }
}
int query(int idx)
{
    int sum = 0;
    while(idx > 0)
    {
        sum += tree[idx];
        idx -= idx&(-idx);
    }
    return sum;
}
int main()
{
    int t,n,q,type,x,y,cas=1;

    scanf("%d",&t);

    while(t--)
    {
        memset(tree,0,sizeof tree);

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

        int a[n+5];

        for(int i=1;i<=n;i++)
        {
            scanf("%d",&a[i]);
            update(i,a[i],n);
        }
        printf("Case %d:\n",cas++);

        while(q--)
        {
            scanf("%d",&type);

            if(type == 1)
            {
                scanf("%d",&x);
                x++;
                update(x,-a[x],n);
                printf("%d\n",a[x]);
                a[x] = 0;
            }
            else if(type == 2)
            {
                scanf("%d%d",&x,&y);
                x++;
                a[x] += y;
                update(x,y,n);
            }
            else
            {
                scanf("%d%d",&x,&y);
                x++ , y++;
                printf("%d\n",query(y)-query(x-1));
            }
        }
    }
}
```

Segment Tree : Lazy Propagation

Problem Name : Computing Fast Average

Problem Link : http://lightoj.com/volume_showproblem.php?problem=1183

Solution :

```
const int MAX = 4e5+100;

struct Tree
{
    int sum, prop;
};

Tree tree[MAX];

void build(int pos,int l,int r)
{
    if(l == r)
    {
        tree[pos].sum = tree[pos].prop = 0;
        return;
    }

    int mid = (l+r)/2, left = 2*pos, right = 2*pos+1;

    build(left,l,mid);
    build(right,mid+1,r);

    tree[pos].sum = tree[pos].prop = 0;
}

void propagate(int pos,int l,int r)
{
    if(tree[pos].prop == -1)
        return;
    int mid = (l+r)/2, left = 2*pos, right = 2*pos+1;

    tree[pos].sum = (r-l+1)*tree[pos].prop;

    if(l != r)
    {
        if(left < MAX)
            tree[left].prop = tree[pos].prop;
        if(right < MAX)
            tree[right].prop = tree[pos].prop;
    }

    tree[pos].prop = -1;

}
void update(int pos,int l,int r,int L,int R,int v)
{
    propagate(pos,l,r);

    if(l > R || r < L)
        return;
    else if(l >= L && r <= R)
    {
        tree[pos].prop = v;
        propagate(pos,l,r);
        return;

    }

    int mid = (l+r)/2, left = 2*pos, right = 2*pos+1;

    update(left,l,mid,L,R,v);
    update(right,mid+1,r,L,R,v);

    tree[pos].sum = tree[left].sum+tree[right].sum;

}
int query(int pos,int l,int r,int L,int R)
{
    propagate(pos,l,r);

    if(l > R || r < L)
        return 0;
    else if(l >= L && r <= R)
        return tree[pos].sum;

    int mid = (l+r)/2, left = 2*pos, right = 2*pos+1;

    int x = query(left,l,mid,L,R);
    int y = query(right,mid+1,r,L,R);

    return x+y;

}

int main()
{
    int t,cas=1;

    scanf("%d",&t);

    while(t--)
    {

        int n,m,type,l,r,v;

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

        build(1,1,n);
        printf("Case %d:\n",cas++);
        for(int i=0; i<m; i++)
        {
            scanf("%d%d%d",&type,&l,&r);

            if(type == 2)
            {
                int sum = query(1,1,n,l+1,r+1);
                int tot = (r-l+1);
                int gcd = __gcd(sum,tot);

                sum /= gcd , tot /= gcd;

                if(sum%tot == 0)
                    printf("%d\n",sum/tot);
                else
                    printf("%d/%d\n",sum,tot);
            }
            else
            {
                scanf("%intd",&v);
                update(1,1,n,l+1,r+1,v);
            }
        }
    }
}
```

Policy Based Data Structure

```
// Program showing a policy-based data structure.
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
#include <iostream>
using namespace __gnu_pbds;
using namespace std;

// a new data structure defined. Please refer below
// GNU link : https://goo.gl/WVDL6g
typedef tree<int, null_type, less<int>, rb_tree_tag,
             tree_order_statistics_node_update>
    new_data_set;

// Driver code
int main()
{
    new_data_set p;
    p.insert(5);
    p.insert(2);
    p.insert(6);
    p.insert(4);

    // value at 3rd index in sorted array.
    cout << "The value at 3rd index ::"
         << *p.find_by_order(3) << endl;

    // index of number 6
    cout << "The index of number 6::"
         << p.order_of_key(6) << endl;

    // number 7 not in the set but it will show the
    // index number if it was there in sorted array.
    cout << "The index of number seven ::"
         << p.order_of_key(7) << endl;

    return 0;
}
```

Sparse Table (RMQ)

Maximum value in a range :

```
const int MAX = 505;

int P[MAX][20];

void buildSparseTable(int arr[], int n)
{
    for(int i = 1; i <= n; i++)
        P[i][0] = arr[i];

    for(int j = 1; (1 << j) <= n; j++)
    {
        for(int i = 1; (i + (1 << j) - 1) <= n; i++)
            P[i][j] = max(P[i][j - 1] , P[i + (1 << (j - 1))][j - 1]);
    }
}

int query(int L, int R)
{
    int log = log2(R - L + 1);

    return max(P[L][log] , P[R - (1 << log) + 1][log]);
}

int main()
{
    int t,cas=1;

    cin >> t;

    while(t--)
    {
        int n,q,u,v;

        cin >> n >> q;

        int a[n+2];

        for(int i=1;i<=n;i++)
            cin >> a[i];

        buildSparseTable(a,n);

        while(q--)
        {
            cin >> u >> v;

            cout << query(u,v) << endl;
        }
    }
}
```

Minimum value in a range :

```
const int MAX = 505;

int P[MAX][20];

void buildSparseTable(int arr[], int n)
{
    for(int i = 1; i <= n; i++)
        P[i][0] = arr[i];


    for(int j = 1; (1 << j) <= n; j++)
    {
        for(int i = 1; (i + (1 << j) - 1) <= n; i++)
            P[i][j] = min(P[i][j - 1] , P[i + (1 << (j - 1))][j - 1]);
    }
}
int query(int L, int R)
{
    int log = log2(R - L + 1);

    return min(P[L][log] , P[R - (1 << log) + 1][log]);
}
```

Sliding RMQ

```
vector<int> slidingRMQ(int a[],int n,int k)
{
    deque<int>d;
    vector<int>res;

    for(int i=0;i<n;i++)
    {
        while(!d.empty() && d.front() >= a[i])
            d.pop_front();

        d.push_front(a[i]);

        if(i>=k && a[i-k] == d.back())
            d.pop_back();

        if(i >= k-1)
            res.push_back(d.back());
    }
    return res;
}

int main()
{
    int n,k;
    cin >> n >> k;
    int a[n+2];

    for(int i=0;i<n;i++)
        cin >> a[i];

    vector<int> res = slidingRMQ(a,n,k);

    for(auto i : res)
        cout << i << " ";
    cout << endl;
}
```

****************String Matching ******************

Knuth-Morris-Pratt (KMP)

KMP code to check if a **pattern** appears in a **text** :

```
const int MAX = 1e6;

vector<int> createLPSArray(string pattern)
{
    vector<int> lps(pattern.size());

    int index = 0;
    for(int i=1;i<pattern.size();)
    {
        if(pattern[index] == pattern[i])
        {
            lps[i] = index+1;
            index++ , i++;
        }
        else
        {
            if(index)
                index = lps[index-1];
            else
                lps[i] = index , i++;
        }
    }

    return lps;
}


bool KMP(string text,string pattern)
{
    vector<int> lps = createLPSArray(pattern);

    int i = 0 , j = 0;

    while(i < text.size())
    {
        if(text[i] == pattern[j])
            i++ , j++;
        else
        {
            if(j)
                j = lps[j-1];
            else
                i++;
        }

        if(j == pattern.size())
            return true;
    }

    return false;
}

int countKMP(string text,string pattern)
{
    vector<int> lps = createLPSArray(pattern);

    int i = 0 , j = 0 , cnt = 0;
    while(i < text.size())
    {
        if(text[i] == pattern[j])
            i++ , j++;
        else
        {
            if(j)
                j = lps[j-1];
            else
                i++;
        }

        if(j == pattern.size())
        {
            cnt++;
            j = lps[j-1];
        }
    }

    return cnt;
}
int main()
{
    FastRead

    string text , pattern;

    cin >> text;

    while(cin >> pattern)
    {
        if(KMP(text,pattern))
        {
            cout << "Found!!\n";
            cout << pattern << " appears " << countKMP(text,pattern) << " times\n";
        }
        else
            cout << "Not Found!!\n";
    }
}
```

****************** Graph Theory *****************

Strongly Connected Components (Kosaraju's Algo)

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

const int MAX = 1e3+10;

vector < int > graph[MAX] , reverseGraph[MAX] , components[MAX];
bool vis[MAX];
int scc[MAX] , compCount;
stack<int>nodes;

void DFS(int src)
{
    vis[src] = 1;
    for(auto i : graph[src])
    {
        if(!vis[i])
            DFS(i);
    }
    nodes.push(src);
}
```

```
void DFS2(int src)
{
    vis[src] = 1;
    for(auto i : reverseGraph[src])
    {
        if(!vis[i])
            DFS2(i);
    }
    components[compCount].push_back(src);
    scc[src] = compCount;
}
void init()
{
    compCount = 1;
    for(int i=1;i<MAX;i++)
        graph[i].clear() , reverseGraph[i].clear() , components[i].clear();
}
void addEdge(int u,int v)
{
    graph[u].push_back(v);
    reverseGraph[v].push_back(u);
}
void kosaraju_SCC(int n)
{
    memset(vis,0,sizeof vis);
    for(int i=1;i<=n;i++)
    {
        if(!vis[i])
            DFS(i);
    }
    memset(vis,0,sizeof vis);
    while(nodes.size())
    {
        int top = nodes.top();
        nodes.pop();
        if(!vis[top])
        {
            DFS2(top);
            compCount++;
        }
    }
}



void print_SCCs()
{
    for(int i=1;i<compCount;i++)
    {
        cout << "Component " << i << ":\n";
        for(auto j : components[i])
            cout << j << " -> ";
        cout << endl;
    }
}
int main()
{
    int n,m,u,v;
    init();
    cin >> n >> m;
    for(int i=0;i<m;i++)
    {
        cin >> u >> v;
        addEdge(u,v);
    }
    kosaraju_SCC(n);
    print_SCCs();
}
```