Team notebook

August 28, 2016

Contents

1 Data Structure **1**
- 1.1 BIT . . . 1
- 1.2 Disjoint$_{S}et$ . . . 2
- 1.3 Mo's$_{A}lgo$ . . . 2
- 1.4 SQRT$_{D}ecomposition$ . . . 4
- 1.5 Segment$_{T}ree$ . . . 5

2 Graph **5**
- 2.1 Articulation Point . . . 5
- 2.2 Edmonds$_{K}arp$ . . . 6
- 2.3 Floyed$_{W}arshall$ . . . 7
- 2.4 Kruskal . . . 8
- 2.5 scc$_{K}osaraju$ . . . 9

3 String **10**
- 3.1 KMP . . . 10
- 3.2 Z$_{a}lgo$ . . . 11

1 Data Structure

1.1 BIT

```
using vi = vector < int >;
using vii = vector < vi >;

struct BIT_2D {
    int n;
    vii tree;
    BIT_2D () {}
    BIT_2D ( int _n ) : n( _n ), tree( _n, vi( _n, 0 ) ) {}
    ~BIT_2D () {}
    void update_y( int x, int y, int v ) {
        for( ; y<n; y+=(y&-y) ) {
            tree[x][y] += v;
        }
    }

    void update( int x, int y, int v ) {
        for( ; x<n; x+=(x&-x) ) {
            update_y( x, y, v );
        }
    }

    int query_y( int x, int y ) {
        int ret = 0;
        for( ; y; y-=(y&-y) ) {
            ret += tree[x][y];
        }
        return ret;
    }

    int query( int x, int y ) {
        int ret = 0;
        for( ; x; x-=(x&-x) ) {
            ret += query_y( x, y );
        }
        return ret;
    }

    int query( int x1, int y1, int x2, int y2 ) {
        return ( query( x2, y2 ) - query( x2, y1-1 ) - query( x1-1, y2 ) +
            query( x1-1, y1-1 ) );
```

```
    }
}

struct BIT {
    int n;
    vi tree;

    BIT () {}
    BIT ( int _n ) : n( _n ), tree( _n, 0 ) {}
    ~BIT () {}
    void update( int x, int v ) {
        for( ; x<n; y+=(x&-x) ) {
            tree[x] += v;
        }
    }

    int query( int x ) {
        int ret = 0;
        for( ; x; x-=(x&-x) ) {
            ret += tree[x];
        }
        return ret;
    }

    int query( int x, int y, int x2, int y2 ) {
        return ( query( y ) - query( x-1 ) );
    }
}
```

1.2 Disjoint$_{S}et$

```
/**
    Implementation of Disjoint-Set Union Data Structure
    Running time:
        O(nlog(n))
    Usage:
        - call make_set() to reset the set
        - call find_rep() to get the set of the vertex
        - call union_() to merge to sets
    Input:
        - n, number of sets
    Tested Problems:
      UVA:
        10608 - Friends
        11503 - Virtual Friends
        10583 - Ubiquitous Religions
**/

struct Disjoint_Set {
    int n;
    vector < int > par, cnt, rank;

    Disjoint_Set( int n ) : n(n), rank(n), par(n), cnt(n) {}

    void make_set() {
        for(int i=0; i<n; i++) {
            par[i] = i;
            cnt[i] = 1;
            rank[i] = 0;
        }
    }

    int find_rep( int x ) {
        if(x != par[ x ]) {
            par[ x ] = find_rep( par[ x ] );
        }
        return par[ x ];
    }

    int union_( int u, int v ) {
        if( ( u = find_rep( u ) ) != ( v = find_rep( v ) ) ) {
            if( rank[ u ] < rank[ v ] ) {
                cnt[ v ] += cnt[ u ];
                par[ u ] = par[ v ];
                return cnt[v];
            } else {
                rank[ u ] = max( rank[ u ], rank[ v ] + 1 );
                cnt[ u ] += cnt[ v ];
                par[ v ] = par[ u ];
            }
        }
        return cnt[u];
    }
};
```

1.3 Mo's$_{A}lgo$

```
/**
    Implementation of Mo's Algo with SQRT-Decomposition Data Structure
    Running time:
        O( ( n + q ) * sqrt( n ) * f() )
    Mo's Algo is a algorithm to process queries offline
    For it to work, this condition must be satisified:
        1) There can be no updates in the array
        2) All queries must be known beforehand
    Tested Problems:
      CF:
        220B - Little Elephant and Array
**/
#include <bits/stdc++.h>
using namespace std;

using piii = pair < pair < int, int >, int >;
const int mx = 1e5 + 1;
int BLOCK_SIZE;
int n, m;
int calc;
int ar[mx];
int ans[mx];
unordered_map < int, int > cnt;
piii query[mx];

struct {
    bool operator()( const piii &a, const piii &b ) {
        int block_a = a.first.first / BLOCK_SIZE;
        int block_b = b.first.first / BLOCK_SIZE;
        if( block_a != block_b ) {
            return block_a < block_b;
        }
        return a.first.second < b.first.second;
    }
} cmp;

void add( int x ) {
    calc -= ( cnt[x] == x ? 1 : 0 );
    cnt[x]++;
    calc += ( cnt[x] == x ? 1 : 0 );
}

void remove( int x ) {
    calc -= ( cnt[x] == x ? 1 : 0 );
    cnt[x]--;
    calc += ( cnt[x] == x ? 1 : 0 );
}

int main() {
    #ifdef LU_SERIOUS
        freopen( "in.txt", "r", stdin );
//        freopen( "out.txt", "w+", stdout );
    #endif // LU_SERIOUS
    while( ~scanf( "%d %d", &n, &m ) ) {

        BLOCK_SIZE = sqrt( n );
        cnt.clear();
        calc = 0;

        for( int i=0; i<n; i++ ) scanf( "%d", ar+i );

        for( int i=0; i<m; i++ ) {
            scanf( "%d %d", &query[i].first.first, &query[i].first.second
                );
            query[i].second = i;
        }

        sort( query, query+m, cmp );

        int mo_l = 0, mo_r = -1;

        for( int i=0; i<m; i++ ) {
            int left = query[i].first.first - 1;
            int right = query[i].first.second - 1;

            while( mo_r < right ) {
                mo_r++;
                add( ar[mo_r] );
            }

            while( mo_r > right ) {
                remove( ar[mo_r] );
                mo_r--;
            }

            while( mo_l < left ) {
                remove( ar[mo_l] );
                mo_l++;
            }
```

```
            while( mo_l > left ) {
                mo_l--;
                add( ar[mo_l] );
            }

            ans[ query[i].second ] = calc;
        }

        for( int i=0; i<m; i++ ) {
            printf( "%d\n", ans[i] );
        }
    }
    return 0;
}
```

1.4 SQRT$_D$ecomposition

```
/**
    Implementation of SQRT-Decomposition Data Structure
    Running time:
        O( ( n + q ) * sqrt( n ) * f() )
    Usage:
        - call int() to initialize the array
        - call update() to update the element in a position
        - call query() to get ans from segment [L...R]
    Input:
        - n, number of elements
        - n elements
        - q queries
    Tested Problems:
      lightOJ:
        1082 - Array Queries
**/
#include <bits/stdc++.h>
using namespace std;

const int mx = 1e5 + 1;
const int sz = 1e3 + 1;
const int inf = 1e9;
int BLOCK_SIZE;
int n, q, t, cs, x, y;
int BLOCKS[sz];
int ar[mx];

int getID( int idx ) {
    return idx / BLOCK_SIZE;
}

void init() {
    for( int i=0; i<sz; i++ ) BLOCKS[i] = inf;
}

void update( int idx, int val ) {
    int id = getID( idx );
    BLOCKS[id] = min( val, BLOCKS[id] );
}

int query( int l, int r ) {
    int le = getID( l );
    int ri = getID( r );
    int ret = inf;

    if( le == ri ) {
        for( int i=l; i<=r; i++ ) {
            ret = min( ret, ar[i] );
        }
        return ret;
    }

    for( int i=l; i<(le+1)*BLOCK_SIZE; i++ ) ret = min( ret, ar[i] );
    for( int i=le+1; i<ri; i++ ) ret = min( ret, BLOCKS[i] );
    for( int i=ri*BLOCK_SIZE; i<=r; i++ ) ret = min( ret, ar[i] );

    return ret;
}

int main() {
    #ifdef LU_SERIOUS
        freopen( "in.txt", "r", stdin );
//      freopen( "out.txt", "w+", stdout );
    #endif // LU_SERIOUS
    scanf( "%d", &t );
    for( cs=1; cs<=t; cs++ ) {
        scanf( "%d %d", &n, &q );
        BLOCK_SIZE = sqrt( n );
        init();
        for( int i=0; i<n; i++ ) {
```

```
            scanf( "%d", &ar[i] );
            update( i, ar[i] );
        }
        printf( "Case %d:\n", cs );
        for( int i=0; i<q; i++ ) {
            scanf( "%d %d", &x, &y );
            printf( "%d\n", query( x-1, y-1 ) );
        }
    }
    return 0;
}
```

1.5 Segment$_{T}$ree

```
struct info {
    int prop, sum;
} tree[ mx * 3 ];

void update( int node, int b, int e, int i, int j, int x ) {
//    cerr << b << " " << e << " " << i << " " << j << " " << x << "\n";
    if( i > e || j < b ) {
        return;
    }
    if( b >= i && e <= j ) {
        tree[node].sum = ( e - b + 1 ) * x;
        tree[node].prop = x;
        return;
    }

    int left = node << 1;
    int right = left | 1;
    int mid = (b + e) >> 1;

    if( tree[node].prop != -1 ) {
        tree[left].sum = ( mid - b + 1 ) * tree[node].prop;
        tree[right].sum = ( e - mid ) * tree[node].prop;
        tree[node].sum = tree[left].sum + tree[right].sum;
        tree[left].prop = tree[node].prop;
        tree[right].prop = tree[node].prop;
        tree[node].prop = -1;
    }

    update(left, b, mid, i, j, x);
    update(right, mid + 1, e, i, j, x);

    tree[node].sum = tree[left].sum + tree[right].sum;
}

int query( int node, int b, int e, int i, int j ) {
    if( i > e || j < b ) {
        return 0;
    }
    if( b >= i and e <= j ) {
        return tree[node].sum;
    }

    int left = node << 1;
    int right = left | 1;
    int mid = (b + e) >> 1;

    if( tree[node].prop != -1 ) {
        tree[left].sum = ( mid - b + 1 ) * tree[node].prop;
        tree[right].sum = ( e - mid ) * tree[node].prop;
        tree[node].sum = tree[left].sum + tree[right].sum;
        tree[left].prop = tree[node].prop;
        tree[right].prop = tree[node].prop;
        tree[node].prop = -1;
    }

    int p1 = query( left, b, mid, i, j );
    int p2 = query( right, mid + 1, e, i, j );


    return p1 + p2;
}
```

2 Graph

2.1 Articulation Point

```
/**
An O(V+E) approach:
-------------------
- perform a DFS on the graph.
- compute d(i) and low(i) for each vertex 1 ... i
```

```
       d(i):  dfs number of i, represents the discovery time.
       low(i): the least dfn reachable from i throguh a path consisting
            of zero or
               more edges follwoing by zero or one back edges.
- vertex u is an AP if and only if:
       - u is the root of the dfs tree and has at least two children.
       - u is not the root and has a child v for which low(v) >= d(u).
 **/
#include <bits/stdc++.h>
using namespace std;

const int mx = 1e4 + 10;
vector < int > G[mx];
int tim, root, n, m, a, b;
int ap[mx], vis[mx], low[mx], d[mx], par[mx];

void ap_dfs(int u) {
    tim++;
    int cnt = 0;
    low[u] = tim;
    d[u] = tim;
    vis[u] = 1;
    int v;
    for(int i=0; i<G[u].size(); i++) {
        v = G[u][i];
        if( v == par[u] ) continue;
        if( !vis[v] ) {
            par[u] = v;
            ap_dfs( v );
            low[u] = min( low[u], low[v] );
            /// d[u] < low[v] if bridge is needed
            if( d[u] <= low[v] && u != root ) {
                ap[u] = 1;
            }
            cnt++;
        } else {
            low[u] = min( low[u], d[v] );
        }
        if( u == root && cnt > 1 ) ap[u] = 1;
    }
}

int main() {

       return 0;
}
```

2.2 Edmonds$_K arp$

```
/**
    Implementation of Edmonds-Karp max flow algorithm
    Running time:
        O(|V|*|E|^2)
    Usage:
        - add edges by add_edge()
        - call max_flow() to get maximum flow in the graph
    Input:
        - n, number of nodes
        - directed, true if the graph is directed
        - graph, constructed using add_edge()
        - source, sink
    Output:
        - Maximum flow
    Tested Problems:
      CF:
        653D - Delivery Bears
      UVA:
        820 - Internet Bandwidth
        10330 - Power Transmission
**/

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

const int INF = 1e9;

struct edmonds_karp {
    int n;
    vector < int > par;
    vector < bool > vis;
    vector < vector < int > > graph;

    edmonds_karp () {}
    edmonds_karp( int _n ) : n( _n ), par( _n ), vis( _n ), graph( _n,
        vector< int > ( _n, 0 ) ) {}
    ~edmonds_karp() {}

    void add_edge( int from, int to, int cap, bool directed ) {
```

```
    this->graph[ from ][ to ] += cap;
    this->graph[ to ][ from ] = directed ? graph[ to ][ from ] + cap :
        graph[ to ][ from ] ;
}

bool bfs( int src, int sink ) {
    int u;
    fill( vis.begin(), vis.end(), false );
    fill( par.begin(), par.end(), -1 );
    vis[ src ] = true;
    queue < int > q;
    q.push( src );
    while( !q.empty() ) {
        u = q.front();
        q.pop();
        if( u == sink ) return true;
        for(int i=0; i<n; i++) {
            if( graph[u][i] > 0 and not vis[i] ) {
                q.push( i );
                vis[ i ] = true;
                par[ i ] = u;
            }
        }
    }
    return par[ sink ] != -1;
}

int min_val( int i ) {
    int ret = INF;
    for( ; par[ i ] != -1; i = par[ i ] ) {
        ret = min( ret, graph[ par[i] ][ i ] );
    }
    return ret;
}

void augment_path( int val, int i ) {
    for( ; par[ i ] != -1; i = par[ i ] ) {
        graph[ par[i] ][ i ] -= val;
        graph[ i ][ par[i] ] += val;
    }
}

int max_flow( int src, int sink ) {
    int min_cap, ret = 0;
    while( bfs( src, sink ) ) {
        augment_path( min_cap = min_val( sink ), sink );
        ret += min_cap;
    }
    return ret;
}
};
```

2.3 Floyed$_{W}arshall$

```
/**
    Implementation of Floyd Warshall Alogrithm
    Running time:
        O( |v| ^ 3 )
    Input:
        - n, number vertex
        - graph, inputed as an adjacency matrix
    Tested Problems:
      UVA:
        544 - Heavy Cargo - MaxiMin path
        567 - Risk - APSP
**/

using vi = vector < int >;
using vvi = vector < vi >;

/// mat[i][i] = 0, mat[i][j] = distance from i to j, path[i][j] = i
void APSP( vvi &mat, vvi &path ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                if( mat[ from ][ via ] + mat[ via ][ to ] < mat[ from ][
                    to ] ) {
                    mat[ from ][ to ] = mat[ from ][ via ] + mat[ via ][ to
                        ];
                    path[ from ][ to ] = path[ via ][ to ];
                }
            }
        }
```

```
    }
}

/// prints the path from i to j
void print( int i, int j ) {
    if( i != j ) {
        print( i, path[i][j] );
    }
    cout << j << "\n";
}

/// check if negative cycle exists
bool negative_cycle( vvi &mat ) {
    APSP( mat );
    return mat[0][0] < 0;
}

void transtitive_closure( vvi &mat ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                mat[ from ][ to ] |= ( mat[ from ][ via ] & mat[ via ][ to
                    ] );
            }
        }
    }
}

/// finding a path between two nodes that maximizes the minimum cost
void mini_max( vvi &mat ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                mat[ from ][ to ] = min( mat[ from ][ to ], max( mat[ from
                    ][ via ], mat[ via ][ to ] ) );
            }
        }
    }
}

/// finding a path between two nodes that minimizes the maximum cost
/// eg: max load a truck can carry from one node to another node where
/// the paths have weight limit
void maxi_min( vvi &mat ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                mat[ from ][ to ] = max( mat[ from ][ to ], min( mat[ from
                    ][ via ], mat[ via ][ to ] ) );
            }
        }
    }
}
```

2.4 Kruskal

```
/**
    Implementation of Kruskal's minimum spanning tree algorithm
    Running time:
        O(|E|log|V|)
    Usage:
        - initialize by calling init()
        - add edges by add_edge()
        - call kruskal() to generate minimum spanning tree
    Input:
        - n, number of nodes, provided when init() is called
        - graph, constructed using add_edge()
    Output:
        - weight of minimum spanning tree
        - prints the mst
    Tested Problems:
        UVA:
            1208 - Oreon
```

```
*/

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

struct edge {
    int u, v, cost;
    bool operator < (const edge& other) const{
        if( other.cost == this->cost ) {
        if( other.u == this->u ) {
            return other.v > this->v;
        } else {
            return other.u > this->u;
        }
        } else {
            return other.cost > this->cost;
        }
    }
};

vector< edge > edges;
vector < int > par, cnt, rank;
int N;

void init( int n ) {
    N = n;
    par.resize( n );
    cnt.resize( n );
    rank.resize( n );
}

void add_edge( int u, int v, int c ) {
    edges.push_back( { u, v, c } );
}

void make_set() {
    for(int i=0; i<N; i++) {
        par[i] = i;
        cnt[i] = 1;
        rank[i] = 0;
    }
}

int find_rep( int x ) {
    if(x != par[ x ]) {
        par[ x ] = find_rep( par[ x ] );
    }
    return par[ x ];
}

int kruskal() {
    int ret = 0;
    make_set();
    sort( edges.begin(), edges.end() );
    cout << "Case " << ++cs << ":\n";
    for( edge e : edges ) {
        int u = e.u;
        int v = e.v;
        if( ( u = find_rep( u ) ) != ( v = find_rep( v ) ) ) {
            if( rank[ u ] < rank[ v ] ) {
                cnt[ v ] += cnt[ u ];
                par[ u ] = par[ v ];
            } else {
                rank[ u ] = max( rank[ u ], rank[ v ] + 1 );
                cnt[ u ] += cnt[ v ];
                par[ v ] = par[ u ];
            }
            cout << city[ e.u ] << "-" << city[ e.v ] << " " << e.cost <<
                 "\n";
            ret += e.cost;
        }
    }
    return ret;
}
```

2.5 $\mathbf{scc}_K osaraju$

```
#include <bits/stdc++.h>
using namespace std;
int p, t;
bool vis[1001];
vector<int> G[1001], gT[1001];
map<string,int> mp;
stack < int > top_sorted;

void dfs_top_sort(int u) {
       vis[u] = true;
       for(int v: G[u]) {
```

```
                if(!vis[v]) {
                        dfs_top_sort( v );
                }
        }
        top_sorted.push( u );
}

void top_sort() {
        for(int i=1; i<=p; i++) {
                if(!vis[i]) {
                        dfs_top_sort(i);
                }
        }
}

void dfs_kosaraju(int u) {
        vis[u] = true;
        for(int v: gT[u]) {
                if(!vis[v]) {
                        dfs_kosaraju( v );
                }
        }
}

int kosaraju() {
        memset( vis, false, sizeof(vis) );
        top_sort();
        int u, ret = 0;
        memset( vis, false, sizeof(vis) );
        while(!top_sorted.empty()) {
                u = top_sorted.top();
                top_sorted.pop();
                if(!vis[u])
                        dfs_kosaraju( u ), ret++;
        }
        return ret;
}
```

3 String

3.1 KMP

```
/// complexity : o( n + m )
///solution reference loj 1255 Substring Frequency
#include <bits/stdc++.h>
using namespace std;

int t;
const int mx = 1e6 + 10;
char a[mx], b[mx];
int table[mx], lenA, lenB;

void hash_table( char *s ) {
        table[ 0 ] = 0;
        int i = 1, j = 0;
        while( i < lenB ) {
                if( s[i] == s[j] ) {
                        j++;
                        table[ i ] = j;
                        i++;
                } else {
                        if( j ) {
                                j = table[ j - 1 ];
                        } else {
                                table[ i ] = 0;
                                i++;
                        }
                }
        }
}

int kmp( char *s, char *m ) {
        hash_table( m );
        int i = 0, j = 0;
        int ans = 0;
        while( i < lenA ) {
                while( i < lenA && j < lenB && s[i] == m[j] ) {
                        i++;
                        j++;
                }
                if( j == lenB ) {
                        j = table[ j - 1 ];
                        ans++;
                } else if( i < lenA && s[i] != m[j] ) {
                        if( j ) {
                                j = table[ j - 1 ];
                        } else {
```

```
                        i++;
                    }
                }
        }
        return ans;
}

int main() {
#ifdef LU_SERIOUS
        freopen("in.txt", "r", stdin);
#endif // LU_SERIOUS
        scanf( "%d", &t );
        for(int cs=1; cs<=t; cs++) {
                lenA = 0; lenB = 0;
                scanf("%s", &a);
                scanf("%s", &b);
                lenA = strlen( a );
                lenB = strlen( b );
                printf( "Case %d: %d\n", cs, kmp( a, b ) );
        }
        return 0;
}
```

3.2 $\mathbf{Z}_{algo}$

```
int L = 0, R = 0;
for( int i = 1; i < n; i++ ) {
        if ( i > R ) {
                L = R = i;
                while ( R < n && s[R-L] == s[R] ) R++;
                z[i] = R-L; R--;
        } else {
        int k = i-L;
                if ( z[k] < R-i+1 ) z[i] = z[k];
                else {
                        L = i;
                        while ( R < n && s[R-L] == s[R] ) R++;
                        z[i] = R-L; R--;
                }
        }
}
int maxz = 0, res = 0;
for ( int i = 1; i < n; i++ ) {
        if ( z[i] == n-i && maxz >= n-i ) { res = n-i; break; }
        maxz = max( maxz, z[i]) ;
}
```