Code library

Mesbah Tanvir

This algorithms note is for competitive programming contest.The codes of this note might not be readable, short code having fast running time is main target here. Though it is tested [1] might content several bugs.

---

[1] Tested on several online judge like lightoj, codeforces,SPOJ.

Table of contents

Stable Marriage **35**

Treap

Data Structure

```
#include <bits/stdc++.h>
using namespace std;
typedef struct node{
    int prior,Size;
    /// add extra information if necessary
    int val;//value stored in the array
    int Max;
    struct node *l,*r;
}node;
typedef node* pnode;
int sz(pnode t){
    return t?t->Size:0;
}
void upd_sz(pnode t){
    if(t)t->Size=sz(t->l)+1+sz(t->r);
}
void reset(pnode t){
    if(t){
        /// modify here
        t->Max =t->val;
    }
    //no need to reset lazy coz when we call this lazy would itself be propagated
}
void combine(pnode& t,pnode l,pnode r){//combining two ranges of segtree
    if(!l || !r)return void(t = l?l:r);
    /// modify here
    /// values elsewhere , cause value will change on the fly
    t->Max = max(l->Max,r->Max);
    /// end here
}
void operation(pnode t){//operation of segtree
    if(!t)return;
    reset(t);
    combine(t,t->l,t);
    combine(t,t,t->r);
}
void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
    if(!t)return void(l=r=NULL);
    int curr_pos = add + sz(t->l);
    if(curr_pos<=pos)//element at pos goes to left subtree(l)
        split(t->r,t->r,r,pos,curr_pos+1),l=t;
    else
        split(t->l,l,t->l,pos,add),r=t;
```

```
    upd_sz(t);
    operation(t);
}
void Merge(pnode &t,pnode l,pnode r){ //l->leftarray,r->rightarray,t->resulting
array
    if(!l || !r) t = l?l:r;
    else if(l->prior>r->prior) Merge(l->r,l->r,r),t=l;
    else    Merge(r->l,l,r->l),t=r;
    upd_sz(t);
    operation(t);
}
pnode init(int val){
    pnode ret = (pnode)malloc(sizeof(node));
    ret->l=ret->r=NULL;
    ret->prior=rand();ret->Size=1;
    /// modify here
    ret->val=val;
    ret->Max=val;
    /// end here
    return ret;
}
int range_query(pnode t,int l,int r){//[l,r]
    pnode L,mid,R;
    split(t,L,mid,l-1);
    split(mid,t,R,r-l);//note: r-l!!
    /// modify here
    int ans = t->Max;
    /// end here
    Merge(mid,L,t);
    Merge(t,mid,R);
    return ans;
}
void insert_at(pnode &t,int i, int val) // i should be less than or e
{
        pnode lft , rgh;
        split(t,lft,rgh,i-1);
        Merge(lft,lft,init(val));
        Merge(t,lft,rgh);
}
void insert_end(pnode &t,int val){
        pnode it = init(val);
        reset(it);
        Merge(t,t,it);
}
void delete_at(pnode &t,int i){
        pnode lft , rgh,ex;
        split(t,lft,rgh,i);
        split(lft,lft,ex,i-1);
        free(ex);
        Merge(t,lft,rgh);
}
void replace_at(pnode &t,int i,int v){
        pnode lft , rgh,ex;
```

```
        split(t,lft,rgh,i);
        split(lft,lft,ex,i-1);
        ex->val=v;
        reset(ex);
        Merge(lft,lft,ex);
        Merge(t,lft,rgh);
}
int main(){
        pnode head;
        head = init(0);
        int n;
        scanf("%d",&n);
        for(int i=0;i<n;i++){
                char c ;
                scanf(" %c",&c);
                if(c=='A'){
                        int x , y;
                        scanf("%d %d",&x,&y);
                        insert_at(head,y,x);
                }
                else if(c=='Q'){
                        int a , b ;
                        scanf("%d %d",&a,&b);
                        int ans = range_query(head,a,b);
                        printf("%d\n",ans);

                }
        }
        return 0;
}
```

Treap [Lazy]

Data Structure

```
typedef struct node{
        int prior,Size;
        int val;//value stored in the array
        /// add extra information if necessary
        int sum;//whatever info you want to maintain in segtree for each node
        int lazy;//whatever lazy update you want to do
        struct node *l,*r;
}node;
typedef node* pnode;
int sz(pnode t){
        return t?t->Size:0;
}
void upd_sz(pnode t){
        if(t)t->Size=sz(t->l)+1+sz(t->r);
}
```

```
void lazy(pnode t){
        if(!t || !t->lazy)return;
        /// modify here
        t->val+=t->lazy;//operation of lazy
        /// calculating update result
        t->sum+=t->lazy*sz(t);
        // Propagate lazy
        if(t->l)t->l->lazy+=t->lazy;if(t->r)t->r->lazy+=t->lazy;
        t->lazy=0;
}
void reset(pnode t){
        if(t)t->sum = t->val;//no need to reset lazy
}
void combine(pnode& t,pnode l,pnode r){//combining two ranges of segtree
        if(!l || !r)return void(t = l?l:r);
        // modify here
        t->sum = l->sum + r->sum;
        // end here
}
void operation(pnode t){//operation of segtree
        if(!t)return;
        reset(t);// assuming it now represents a single element of the array
        lazy(t->l);lazy(t->r);//imp:propagate lazy before combining t->l,t->r;
        combine(t,t->l,t);combine(t,t,t->r);
}
void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
        if(!t)return void(l=r=NULL);
        lazy(t);
        int curr_pos = add + sz(t->l);
        if(curr_pos<=pos)//element at pos goes to left subtree(l)
                split(t->r,t->r,r,pos,curr_pos+1),l=t;
        else    split(t->l,l,t->l,pos,add),r=t;
        upd_sz(t);operation(t);
}
void Merge(pnode &t,pnode l,pnode r){ //l->leftarray,r->rightarray,t->resulting
array
        lazy(l);lazy(r);
        if(!l || !r) t = l?l:r;
        else if(l->prior>r->prior)Merge(l->r,l->r,r),t=l;
        else    Merge(r->l,l,r->l),t=r;
        upd_sz(t);operation(t);
}
pnode init(int val){
        pnode ret = (pnode)malloc(sizeof(node));
        ret->l=ret->r=NULL;ret->prior=rand();ret->Size=1;
        /// modify here
        ret->val=val;ret->sum=val;ret->lazy=0;
        /// end here
        return ret;
}
int range_query(pnode t,int l,int r){//[l,r]
        pnode L,mid,R;
        split(t,L,mid,l-1);split(mid,t,R,r-l);//note: r-l!!
```

```
        /// modify here
        int ans = t->sum;
        /// end here
        Merge(mid,L,t);Merge(t,mid,R);
        return ans;
}
void range_update(pnode t,int l,int r,int val){//[l,r]
        pnode L,mid,R;
        split(t,L,mid,l-1);split(mid,t,R,r-l);//note: r-l!!
        /// modify here
        t->lazy+=val; //lazy_update
        /// end here
        Merge(mid,L,t);Merge(t,mid,R);
}
```

Implicit Persistent Segment Tree

```
/* implicit persistent segment tree */
#include <bits/stdc++.h>
#define MAX 1000000005
#define MAXN 2000000
using namespace std;
int a[MAXN],l[MAXN],r[MAXN],tree[MAXN],root[MAXN],free_index;
int update(int s , int e , int id,int i){
        int cur_id =++free_index;
        if(s==e){
                tree[cur_id]=tree[id]+1;
                return cur_id;
        }
        int mid = (s+e)/2;
        l[cur_id] = l[id] , r[cur_id] = r[id];
        if(i<=mid) l[cur_id] = update(s, mid, l[id], i);
        else r[cur_id] = update(mid+1, e, r[id], i);
        tree[cur_id] = tree[l[cur_id]] + tree[r[cur_id]];
        return cur_id;
}
int query(int s , int e, int id, int i){
        if(s==e  && e==i) return tree[id];
        int mid = (s + e)/2;
        if( i<= mid) return query(s, mid, l[id], i);
        else return tree[ l[id] ] + query(mid+1, e, r[id], i);
}
int main(){
        int n ,m;
        scanf("%d",&n);
        for(int i = 1; i <= n; i++ )   scanf("%d", &a[i]);
        for(int i = 1; i <= n; i++ )   root[i] = update(0, MAX, root[i-1], a[i]);
        scanf("%d", &m);
```

```
        int last_ans = 0;
        for(int i = 1; i <= m; i++){
                int x , y, z;
                scanf("%d %d %d", &x, &y, &z);
                x = x^last_ans; y = y^last_ans; z = z^last_ans;
                if( y > n ) y = n;
                if( x < 1 ) x = 1;
                if( y < x ) last_ans = 0;
                else {
                        int pre = query(0, MAX, root[x-1], z);
                        int now = query(0, MAX, root[y], z);
                        last_ans = (y - x + 1 - now + pre);
                }
                printf("%d\n", last_ans);
        }
        return 0;
}
```

Binary Indexed Tree

Data Structure

```
/*     binary indexed tree maintain 1 based indexing   */
#define MAX 1000

int n, Tree[MAX], a[MAX];
void update(int bit, int val){
       while(bit <= n){
              Tree[bit] += val; bit += (bit) & (-bit);
       }
}
int query(int bit, int sum=0){
       while(bit){
              sum += Tree[bit]; bit -= (bit) & (-bit);
       }
       return sum;
}
```

Binary Indexed Tree [2D]

Data Structure

```
//update and query function for 2D bit.
//MAX is the maximum possible value.
//bit[][] holds the 2D binary indexed tree
void update(int x, int y, int v) {
    int y1;
    while(x <= MAX) {
        y1 = y;
        while(y1 <= MAX) {
```

```
            bit[x][y1] += v;
            y1 += (y1 & -y1);
        }
        x += (x & -x);
    }
}
int readsum(int x, int y) {
    int v = 0, y1;
    while(x > 0) {
        y1 = y;
        while(y1 > 0) {
            v += bit[x][y1];
            y1 -= (y1 & -y1);
        }
        x -= (x & -x);
    }
    return v;
}
```

Mo's Algorithm

Data Structure

```
struct info{
        int s, e, i;
};
info res[MAX]; int sqr;
bool cmp(info a, info b){
        if(a.s/sqr == b.s/sqr) return a.e < b.e;
        return  (a.s/sqr < b.s/sqr);
}
void add(long long x){ /*change according to addition of new element*/}
void del(long long x){ /*change according to deletion of new element*/}
void mo_algo(){
        int l=0, r=0;
        for(int i=0; i<q; i++){
                while(r<=res[i].e){
                        add(a[r]);      r++;
                }
                while(l>res[i].s){
                        l--;      add(a[l]);
                }
                while(r>res[i].e+1){
                        r--;      del(a[r]);
                }
                while(l<res[i].s){
                        del(a[l]);      l++;
                }
                ans[res[i].i]= result;
```

```
        }
}
void getdata(){
        scanf("%d",&n);
        for(int i=0; i<n; i++) scanf("%d",&a[i]);
}
void getQuery(){
        scanf("%d",&q);
        for(int i=0; i<q; i++){
                int s, e;
                scanf("%d %d",&s,&e);
                res[i].i=i; res[i].s=s-1; res[i].e=e-1;
        }
}
void solve(){
        sqr = sqrt(n); sort(res,res+q,cmp); mo_algo();
}
int main(){
        getdata(); getQuery(); solve();
        return 0;
}
```

Suffix Array

String

```
const int MAXN = 1 << 20;
const int MAXL = 20;
// n, number of character
// step log (n)
// MAXN maximum number of  character
// MAXL log(MAXN)
int n, step, mv, suffix[MAXN], tmp[MAXN];
int sum[MAXN], cnt[MAXN], tab[MAXL][MAXN];
char str[MAXN],str1[MAXN],str2[MAXN];

inline bool soman(const int &u, const int &v){
    if(!step) return str[u] == str[v];
    int step_1 =step-1;
    if(tab[step_1][u] != tab[step_1][v]) return false;
    int a = u + mv < n ? tab[step_1][u+mv] : -1;
    int b = v + mv < n ? tab[step_1][v+mv] : -1;
    return a == b;
}

void update(){
    int i, rnk;
    int *temp =tab[step];
    for(i = 0; i < n; i++) sum[i] = 0;
    for(i = rnk = 0; i < n; i++){
        int is = suffix[i] = tmp[i];
        if(i && !soman(is, suffix[i-1])){
```

```
            temp[is] = ++rnk;
            sum[rnk+1] = sum[rnk];
        }
        else temp[is] = rnk;
        sum[rnk+1]++;
    }
}
void Sort()
{
    for(int i = 0; i < n; i++) cnt[i] = 0;
    memset(tmp, -1, sizeof tmp);
    for(int i = 0; i < mv; i++){
        int idx = tab[step - 1][n - i - 1];
        int x = sum[idx];
        tmp[x + cnt[idx]] = n - i - 1;
        cnt[idx]++;
    }
    for(int i = 0; i < n; i++){
        int idx = suffix[i] - mv;
        if(idx < 0)continue;
        idx = tab[step-1][idx];
        int x = sum[idx];
        tmp[x + cnt[idx]] = suffix[i] - mv;
        cnt[idx]++;
    }
    update();
    return;
}

inline bool cmp(const int &a, const int &b){
    char cc = str[a] , dd =str[b];
    if(cc!=dd) return cc<dd;
    return false;
}

void SortSuffix(){
    for(int i = 0; i < n; i++) tmp[i] = i;
    sort(tmp, tmp + n, cmp);
    step = 0;
    update();
    ++step;
    for(mv = 1; mv < n; mv <<= 1)  Sort(),step++;
    step--;
    for(int i = 0; i <= step; i++) tab[i][n] = -1;
}

inline int lcp(int u, int v){
    if(u == v) return n - u;
    int ret, i;
    for(ret = 0, i = step; i >= 0; i--){
        if(tab[i][u] == tab[i][v])
        {
            ret += 1<<i;
```

```
            u += 1<<i;
            v += 1<<i;
        }
    }
    return ret;
}
```

Prefix Function

String

```
int pf [ MAX ] ;
void prefixFunction( const char * a, int n ){
    int i, sp ;
    pf [0] = 0 ;
    for(i=1 ; i<n ; i++ ){
        sp = pf[i-1] ;
        while( a[sp] != a[i] && sp ) sp = pf[sp-1];
        if( sp ) pf[i] = sp + 1 ;
        else pf[i] = ( a[i] == a[sp]) ;
    }
}
```

Z Function

Z Algorithm

```
int z[1000000];
void z_function(const char * str, int n){
    int  l=-1, r=-1, i;
    z[0]=0;
    for(i=1; i<n; i++){
        if(i>r){
            l=r=i;
            while(l<n && str[r]==str[r-l]) 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 && str[r-l]==str[r]) r++;
                z[i]=r-l;r--;
            }
        }
    }
```

```
}
```

Trie [Array]

String

```
#include <bits/stdc++.h>

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

#define MAX     100
#define RANGE   26
struct Trie{
        int trie[MAX][RANGE], cnt[MAX][RANGE], sz, ROOT;
        void init(){
                memset(trie,0,sizeof(trie));
                memset(cnt,0,sizeof(cnt));
                sz =0;ROOT=0;
        }
        int scale(char c){
                return (c-'a');
        }
        void  Insert(char s[], int SZ){
                int node = ROOT;
                for(int i=0; i<SZ; i++){
                        int next = scale(s[i]);
                        if(trie[node][next]==0) trie[node][next] = ++sz;
                        cnt[node][next]++;
                        node = trie[node][next];
                }
        }
        bool Query(char s[], int SZ){
                int node = ROOT;
                for(int i=0; i<SZ; i++){
                        int next = scale(s[i]);
                        if(!trie[node][next]|| !cnt[node][next]){
                                return false;
                        }
                        node= trie[node][next];
                }
                return true;
        }
        void Delete(char s[], int SZ){
                int node = ROOT;
                for(int i=0; i<SZ; i++){
                        int next = scale(s[i]);
                        if(!cnt[node][next]) return ;
                        cnt[node][next]--;
                        node = trie[node][next];
                }
```

```
        }
};
```

Heavy Light Decomposition

Graph

```
#define LG 20
#define MAX 100005
#define left s,(s+e)/2,n+n
#define right 1+(s+e)/2,e,n+n+1

vector < int > G[MAX];
int pr[MAX][LG],depth[MAX],subTree[MAX],chainID[MAX],chainHEAD[MAX];
int aPOS[MAX],COST[MAX],a[MAX, start , chain, n,m;
vector < pair < int  , int > > extra;
map < pair<int , int >  , bool > MAP;

// calculating depth , size of subtree and first parent
void dfs(int u,int p,int d){
        pr[u][0]=p;
        depth[u]=d;
        subTree[u]=1;
        int SZ = G[u].size();
        for(int i=0;i<SZ;i++){
                int v = G[u][i];
                if(v!=p){
                        dfs(v,u,d+1);
                        subTree[u]+=subTree[v];
                }
        }
}
// dfs must be called
void LCA_Preprocess(){
        for(int i=1;i<LG;i++)
                for(int j=1;j<=n;j++)
                        pr[j][i]=pr[pr[j][i-1]][i-1];
}
// u--a->lca , v--b->lca
int LCA(int u , int v,int &a,int &b){
        bool flag = false;
        if(depth[u]<depth[v]) {
                swap(u,v);
                flag = true;
        }
        if(depth[u]!=depth[v])
                for(int i=LG-1;i>=0;i--)
                        if(depth[u]-(1<<i)>=depth[v])
                                u = pr[u][i];
        if(u==v) return u;
        for(int i=LG-1;i>=0;i--)
                if(pr[u][i]!=pr[v][i])
```

```
                        u = pr[u][i],v = pr[v][i];
        if(flag) a = v, b = u;
        else     a = u, b = v;
        return pr[u][0];
}
int getImidiate(int u , int v){ // v is some kind of parent
        for(int i=LG-1;i>=0;i--){
                if(depth[u]-(1<<i)>depth[v]){
                        u = pr[u][i];
                }
        }
        return u;
}
// decompose tree
// aPOS[u] : position of node u  in final tree
// chainID[u] : chain number where node u belongs to
// chainHEAD[u] : head node of chainID[u]
void HLD(int u,int p,int chainNUM){
        aPOS[u] =start; a[start++]=u; chainID[u]=chainNUM;
        if(chainHEAD[chainID[u]]==-1) chainHEAD[chainID[u]]=u;
        int SZ = G[u].size() , idx=-1, val;
        for(int i=0;i<SZ;i++){
                int v = G[u][i];
                if(v==p) continue;
                if(idx==-1 || subTree[v]>val){
                        idx = v;
                        val = subTree[v];
                }
        }
        if(idx!=-1) HLD(idx,u,chainNUM);
        for(int i=0;i<SZ;i++){
                int  v=G[u][i];
                if(v==p || v==idx) continue;
                chain++; HLD(v,u,chain);
        }
}
// segment tree part
struct node{ };
node tree[4*MAX];
void merge_node(int s, int e, int n){ }
void build(int s, int e, int n){
        if(s==e){ return ; }
        build(left);build(right);
        merge_node(s,e,n);
}
void update(int s, int e, int n, int l, int r,int add){
        if(s>r || e<l) return ;
        if(s>=l && e<=r){ return;}
        update(left,l,r,add);update(right,l,r,add);
        merge_node(s,e,n);
}
// traverse through chain and go up from u to v
void go_fix(int u , int v,int x){
```

```
        // v must be some kind of parent of u
        while(1){
                if(chainID[u]==chainID[v]){ // u , v are in same chain
                        update(aPOS[v],aPOS[u],x);
                        break;
                }
                else { // different chain
                        update(aPOS[chainHEAD[chainID[u]]],aPOS[u],x);
                        u = pr[chainHEAD[chainID[u]]][0];
                }
        }
}
void fixTree(int u , int v , int x){
        int a =0, b=0, lca = LCA(u,v,a,b);
        if(u==v) return ;
        if(lca==u){
                u = getImidiate(v,u); go_fix(v,u,x);
                return ;
        }
        if(lca==v){
                v = getImidiate(u,v);go_fix(u,v,x);
                return ;
        }
        go_fix(u,a,x);go_fix(v,b,x);
        return ;
}
```

Lowest Common Ancestor

LCA

```
#define MAX 10010
#define LG 16
int depth[MAX],pr[MAX][LG],cost[MAX][LG],n;
vector < int > G[MAX],C[MAX];
void dfs(int u, int p, int d)
{
        depth[u]=d;
        int SZ = G[u].size();
        for(int i=0;i<SZ;i++){
                int v = G[u][i];
                if(v==p) continue;
                pr[v][0]=u;
                cost[v][0]=C[u][i];
                dfs(v,u,d+1);
        }
}
```

```
void preprocessLCA()
{
        for(int i=1;i<LG;i++){
                for(int j=1;j<=n;j++){
                       pr[j][i] = pr[pr[j][i-1]][i-1];
                        cost[j][i] = cost[j][i-1] + cost[pr[j][i-1]][i-1];
                }
        }
}
int getLCA(int u , int v , int &res){
        res=0;
        if(u==v) return v;
        if(depth[u]<depth[v]) swap(u,v);
        for(int i=LG-1;i>=0;i--){
                if(depth[u]-(1<<i)>=depth[v]){
                        res+=cost[u][i];
                        u = pr[u][i];
                }
        }
        if(v==u) return u;
        for(int i =LG-1;i>=0;i--){
                if(pr[u][i]!=pr[v][i]) {
                        res+=cost[u][i];
                        u = pr[u][i];
                        res+=cost[v][i];
                        v = pr[v][i];
                }
        }
        res+=cost[u][0]+cost[v][0];
        return pr[v][0];
}
```

Articulation Point

Graph

```
// status[v] will store true if node v is an articulation point
// vis[v] will store visiting time of node v
// low[v] will store the minimum time that can be reached
int vis[MAX] , low[MAX] ,cut[MAX] , TIME;
vector < int > G[MAX];
void ArticulationPoint(int u , int p){
    int child =0, v, i;
    vis[u] = low[u] = ++TIME;
    for(i = 0; i < G[u].size(); i++){
        v = G[u][i];
        if(v == p) continue;
        if(~vis[v]) low[u] = min(low[u],vis[v]);
        else {
            ++child;
            ArticulationPoint(v, u);
            low[u]= min(low[v], low[u]);
```

```
            if(low[v] >= vis[u]) cut[u] = 1;
        }
    }
    if(p == -1) cut[u] = (child>1);
}
```

Bridge

Graph

```
// result will store bridge edges in random order
// vis[v] will store visiting time of node v
// low[v] will store the minimum time that can be reached
int vis[MAX] , low[MAX] , TIME;
vector < pair  < int , int > > result;
vector < int > G[MAX];
void Bridge(int u , int p){
    int v, i;
    vis[u] = low[u] = ++TIME;
    for(i = 0; i < G[u].size(); i++){
        v = G[u][i];
        if(v == p) continue;
        if(vis[v] != -1) low[u] = min(low[u],vis[v]);
        else {
            Bridge(v, u);
            low[u]= min(low[v], low[u]);
            if(low[v] > vis[u]) result.push_back(make_pair(u ,v)));
        }
    }
}
```

Konig's Theorem	In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

Vertex Cover	Simply BPM is ans. (Konig's Theorem)
Path Cover	Take two copy of each node. (N-BPM).

Edmonds Karp [MAXFLOW]

Graph

```
// 1 based graph
int edge[MAXN][MAXN], pr[MAXN], vis[MAXN];
void bfs(int s, int t, int n){
```

```
        memset(pr,0,sizeof(pr));
        memset(vis,0,sizeof(vis));
        queue < int > Q; Q.push(s); vis[s]=1;
        while(!Q.empty()){
                int u = Q.front(); Q.pop();
                for(int i=0; i<n; i++){
                        if(edge[u][i]>0){
                                if(vis[i]==0){
                                        vis[i]=1; pr[i]=u; Q.push(i);
                                }
                        }
                }
        }
}
int path_find(int s, int t, int n){
        if(pr[t]==0) return 0;
        int min_value=INT_MAX,cur=t;
        while(cur!=s){
                int nex = pr[cur];
                min_value = min(min_value,edge[nex][cur]);
                cur = nex;
        }
        cur=t;
        while(cur!=s){
                int nex = pr[cur];
                edge[nex][cur]-=min_value;
                edge[cur][nex]+=min_value;
                cur = nex;
        }
        return min_value;
}
int Flow(int s , int t , int n){
        int result=0; bfs(s,t,n);
        int res= path_find(s,t,n);
        while(res){
                result+=res;
                bfs(s,t,n);
                res = path_find(s,t,n);
        }
        return result;
}
```

Hopcroft Karp [BPM]

Graph

```
/* Author : Bidhan Roy| Complexity : O (|E|sqrt|V|)| 1 based indexing */
namespace hopcroftKarp{
        #define MAXN 50010 // Maximum possible Number of nodes
```

```
        #define MAXE 500010 // Maximum possible Number of edges*3
        #define INF (1<<29)
        int ptr[MAXN], next[MAXE], zu[MAXE], n, m, match[MAXN], D[MAXN], q[MAXN];
        void init(int _n){  // initialization _n=number of nodes
                n=_n; m=0;
                memset(ptr, ~0, sizeof(int)*(n+1)); memset(match, 0,
sizeof(match));
                memset(D, 0, sizeof(D)); memset(q, 0, sizeof(q));
                // memset(next,0,sizeof(next)); // memset(zu,0,sizeof(zu));
        }
        void add_edge(int u,int v){   // Adding edge between u and v
                next[m]=ptr[u]; ptr[u]=m; zu[m]=v; ++m;
        }
        bool bfs(){
                int u, v; register int i; int qh=0, qt=0;
                for(i=1; i<=n; i++){
                        if(!match[i]) { D[i]=0;q[qt++]=i; }
                        else D[i]=INF;
                }
                D[0]=INF;
                while(qh<qt){
                        u=q[qh++];
                        if(u!=0){
                                for(i=ptr[u]; ~i; i=next[i]){
                                        v=zu[i];
                                        if(D[match[v]]==INF){
                                                D[match[v]]=D[u]+1;
                                                q[qt++]=match[v];
                                        }
                                }
                        }
                }
                return D[0]!=INF;
        }
        bool dfs(int u){
                int v;register int i;
                if(u){
                        for(i=ptr[u]; ~i; i=next[i]){
                                v=zu[i];
                                if(D[match[v]]==D[u]+1){
                                        if(dfs(match[v])){
                                                match[v]=u;match[u]=v;
                                                return true;
                                        }
                                }
                        }
                        D[u]=INF;
                        return false;
                }
                return true;
        }
        int run(){
                int matching=0;register int i;
```

```
                while(bfs())
                        for(i=1; i<=n; i++)
                                if(!match[i] && dfs(i))
                                        matching++;
                return matching;
        }
        #undef MAXN
        #undef INF
};
```

Dinitz [MAXFLOW]

Graph

```
struct MaxFlow {
        // MAXV: Number of vertex| MAXE: Number of Edge
        // F_INF: greater than MAXIMUM flow| INF : 1e7
        // i64: long long| SET(x): memset(-1)
        int V, E;int start[MAXV], next[MAXE], v[MAXE];
        int used[MAXV], level[MAXV];int cap[MAXE], flow[MAXE];
        MaxFlow(int n) {
                int i; V = n; E = 0;
                memset(start, -1, sizeof(start));
        }
        void add_edge(int x, int y, int c) {
                cap[E] = c;flow[E] = 0;v[E] = y;next[E] = start[x]; start[x] =
E;++E;
                cap[E] = 0;flow[E] = 0;v[E] = x;next[E] = start[y];start[y] = E;++
E;
        }
        bool bfs(int s, int t) {
                memset(level, -1, sizeof(level));
                queue< int > q;
                q.push(s); level[s] = 0;
                while (!q.empty()) {
                        int x = q.front(); q.pop();
                        for (int i = start[x]; i != -1; i = next[i])
                                if (level[v[i]] == -1 && cap[i] > flow[i]) {
                                        q.push(v[i]);
                                        level[v[i]] = level[x] + 1;
                                }
                }
                return (level[t] != -1);
        }
        int dfs(int s, int t, int f) {
                if (s == t) return f;
                for (int &i = used[s]; i != -1; i = next[i])
                //if (level[v[i]] > level[s] && cap[i] > flow[i]) { // should be
same
                        if (level[v[i]] == level[s] + 1 && cap[i] > flow[i]) {
                                int temp = dfs(v[i], t, min(f, cap[i] - flow[i]));
                                if (temp > 0) {
```

```
                                        flow[i] += temp; flow[i^1] -= temp;
                                        return temp;
                                }
                        }
                return 0;
        }
        int maxflow(int s, int t) {
                while (bfs(s, t)) {
                        for (int i = 0; i < V; ++ i) used[i] = start[i];
                        while (dfs(s, t, F_INF) != 0);
                }
                int ret = 0;
                for (int i = start[s]; i != -1; i = next[i]) ret += flow[i];
                return ret;
        }
};
```

Strongly Connected Component

Graph

```
/*
SCC (Tarjan) in O(|v| + |e|)
Input:
G[] is a input directed graph with n nodes in range [1,n]
Output:
Component[i] holds the component id to which node i belongs
components: total number of components in the graph
*/

int Stack[MAX], top;
int Index[MAX], Lowlink[MAX], Onstack[MAX];
int Component[MAX];
int idx, components;
vector< int > G[MAX];

void tarjan(int u) {
    int v, i;
    Index[u] = Lowlink[u] = idx++;
    Stack[top++] = u;
    Onstack[u] = 1;
    for(i = 0; i < SZ(G[u]); i++) {
        v = G[u][i];
        if(Index[v]==-1) {
            tarjan(v);
            Lowlink[u] = min(Lowlink[u], Lowlink[v]);
        }
        else if(Onstack[v]) Lowlink[u] = min(Lowlink[u], Index[v]);
    }
    if(Lowlink[u] == Index[u]) {
```

```
        components++;
        do {
            v = Stack[--top];
            Onstack[v] = 0;
            Component[v] = components;
        } while(u != v);
    }
}

void findSCC(int n) {
    components = top = idx = 0;
    SET(Index); CLR(Onstack); MEM(Lowlink, 0x3f);
    for(int i = 1; i <= n; i++) if(Index[i]==-1) tarjan(i);
}
```

Disjoint Set

Graph

```
/*
disjoint set data-structure
implements union by rank and path compression
*/

struct DisjointSet {
    int *root, *rank, n;
    DisjointSet(int sz) {
        root = new int[sz+1];
        rank = new int[sz+1];
        n = sz;
    }
    ~DisjointSet() {
        delete[] root;
        delete[] rank;
    }
    void init() {
        for(int i = 1; i <= n; i++) {
            root[i] = i;
            rank[i] = 0;
        }
    }
    int find(int u) {
        if(u != root[u]) root[u] = find(root[u]);
        return root[u];
    }
    void merge(int u, int v) {
        int pu = find(u);
        int pv = find(v);
        if(rank[pu] > rank[pv]) root[pv] = pu;
        else root[pu] = pv;
        if(rank[pu]==rank[pv]) rank[pv]++;
    }
```

```
};
```

Euclid/Extended Euclid

Math

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
struct Euclid{
        ll x , y  , d;
        Euclid(ll a=0, ll b=0, ll c=0){
                x=a , y=b , d =c;
        }
        static ll gcd(ll a, ll b){
                if(!b) return a;
                return Gcd(b,a%b);
        }
        static Euclid Extend(ll a, ll b){
                if( !b ) return Euclid(1, 0, a);
                Euclid E = Extend(b,a%b);
                return Euclid(E.y, E.x - (a/b) * 1LL * E.y, E.d);
        }
        static ll ModInv(ll a, ll m){
                ll r = Extend(a,m).x;
                return r>=0?r:r+m;
        }

};
int main(){
}
```

Tricks

if $ax_1 + by_1 = c$ is any solution, then all solutions are of the form

$$x = x_1 - r\frac{b}{\gcd(a,b)}, \qquad y = y_1 + r\frac{a}{\gcd(a,b)}$$

Bitwise Sieve

Math

```
#include <bits/stdc++.h>
using namespace std;
/*
        PRIME_UPTO/10 is asymptotic change as need
        405 ms, 67328 KB for PRIME_UPTO = 10^8
*/
#define PRIME_UPTO 100000000
int isPrime[PRIME_UPTO>>5], Prime[PRIME_UPTO/10],nPrime;
inline bool checkPrime(int i){
        return ( ( isPrime[ i>>5 ]) & (1 << (i&31)) );
}
inline void setPrime(int i){
        isPrime[ i>>5 ] = ( ( isPrime[ i>>5 ] ) | (1 << ( i&31 ) ));
}
void SieveBit(){
        int Sqrt = sqrt(PRIME_UPTO), i, j;
        for( i = 4; i < PRIME_UPTO; i += 2) setPrime(i);
        for( i = 3; i <= Sqrt; i += 2)
                if( !checkPrime(i) )
                        for( j = i*i; j < PRIME_UPTO; j += i<<1)
                                setPrime(j);
        j = 0;
        nPrime = 0, Prime[j++] = 2;
        for( i = 3; i < PRIME_UPTO; i += 2)
                if( !checkPrime(i) )
                        Prime[j++] = i;
        nPrime = j;
}
int main(){
        SieveBit();
        for(int i = 0; i < 25; i++){
                cout<< Prime[i] <<endl;
        }
        return 0;
}
```

Josephus Recurrence

Math

```
/*
The first one is for K = 2 and the second one is general.
Note: first function returns 1 based index while second one is 0 based.
*/
int f(int n) {
    if(n == 1) return 1;
    return (f((n-(n&1))>>1)<<1) + ((n&1)?1:-1);
}
t
```

Matrix Exponent

```
typedef long long ll;
long long  MOD = 1000000007;
long long MOD_SQR = MOD*MOD;
const int MAT_SIZE = 55;
struct matrix
{
    int dim, mat[MAT_SIZE][MAT_SIZE];
    matrix(int SZ=0)
    {
        dim = SZ;
        memset(mat,0,sizeof(mat));
    }
    void init(int SZ = 0)
    {
        dim = SZ;
        memset(mat,0,sizeof(mat));
    }
    static matrix iden(int SZ = 0)
    {
        matrix r = matrix(SZ);
        for(int i = 0; i < SZ; i++) r.mat[i][i] = 1;
        return r;
    }
    matrix operator*(matrix b)
    {
        matrix r = matrix(dim);
        for(int i = 0; i < dim; i++)
        {
            for(int j = 0; j < dim; j++)
            {
                ll temp = 0;
                for(int k = 0; k < dim; k++)
                {
                    temp += mat[i][k] * 1LL * b.mat[k][j];
                    if(temp > MOD_SQR) temp -= MOD_SQR;
                }
                r.mat[i][j] = (temp % MOD);
            }
        }
        return r;
    }
    matrix operator^(ll p)
    {
        matrix x = *this;
        matrix r = matrix :: iden(dim);
        while (p)
        {
```

```
            if (p & 1) r = r * x;
            p = p >> 1, x = x * x;
        }
        return r;
    }
    void print()
    {
        for(int i = 0; i < dim; i++)
            for(int j = 0; j < dim; j++) cout<<mat[i][j]<<" \n"[j==dim-1];
    }
};
```

Rips Van winckle code

Problems

```
#include <bits/stdc++.h>
#define MAX 1500000
using namespace std;

struct info
{
    long long value;
    bool laz;
    bool SET;
    long long shuru;
    int diff;
    int cnt;

};
info tree[MAX];

#define SAME 1
// set propagation
#define SERI 2
// series propagation
#define NONE 0
// no propagation

#define left s,(s+e)/2,n+n
#define right (s+e)/2+1,e,n+n+1
#define call 1,250100,1

long long series_sum(long long a, long long  d, long long n)
{
    return n*a + d*((n-1)*n)/2;
}

long long next_num(long long a, long long d, long long n)
{
    return a+(n-1)*d;
```

```
}

void prop(long long s, long long e, long long n)
{

    if(tree[n].laz)
    {
        if(tree[n].SET)
        {
            tree[n].value = series_sum(tree[n].shuru,tree[n].diff,tree[n].cnt);
        }
        else tree[n].value += series_sum(tree[n].shuru,tree[n].diff,tree[n].cnt);
        if(s!=e)
        {
            if(tree[n].SET){
                tree[n+n].shuru=tree[n].shuru;
                tree[n+n+1].shuru =
next_num(tree[n].shuru,tree[n].diff,tree[n+n].cnt+1);
                tree[n+n].diff = tree[n+n+1].diff = tree[n].diff;
                tree[n+n].SET = tree[n+n+1].SET = 1;
                tree[n+n].laz=tree[n+n+1].laz=1;
            }
            else {
                tree[n+n].shuru += tree[n].shuru;
                tree[n+n+1].shuru +=
next_num(tree[n].shuru,tree[n].diff,tree[n+n].cnt+1);
                tree[n+n].diff += tree[n].diff;
                tree[n+n+1].diff += tree[n].diff;
                tree[n+n].laz=tree[n+n+1].laz=1;

            }



        }
        tree[n].laz = tree[n].shuru = tree[n].diff= tree[n].SET = 0;
    }
}
void lazy(long long s, long long e, long long n, long long l, long long r, long
long type, long long val)
{
    if(type == SAME)
    {
        tree[n].shuru = val;
        tree[n].diff = 0;
        tree[n].SET =1;
        tree[n].laz=1;
        return ;
    }
    else if(type == 2)
    {
        tree[n].shuru += (s-l+1);
        tree[n].diff += 1;
```

```
        tree[n].laz=1;
        return ;
    }
    else if(type == 3)
    {
        tree[n].shuru += (r-s+1);
        tree[n].diff -= 1;

        tree[n].laz=1;
        return ;
    }
}


void build(long long s, long long e, long long n)
{
    if(s==e)
    {
        tree[n].cnt=1;
        tree[n].value = tree[n].diff =tree[n].SET=tree[n].laz=0;
        return ;
    }
    build(left);
    build(right);
    tree[n].cnt = tree[n+n].cnt + tree[n+n+1].cnt;
}

void update(long long s, long long e, long long n, long long l, long long r, long
long type, long long val=0)
{
    prop(s,e,n);
    if(s>r || e<l) return ;
    if(s>=l && e<=r)
    {
        lazy(s,e,n,l,r,type,val);
        prop(s,e,n);
        return ;
    }
    update(left,l,r,type,val);
    update(right,l,r,type,val);
    tree[n].value = tree[n+n].value + tree[n+n+1].value;
}

long long query(long long s, long long e, long long n, long long l, long long r)
{
    prop(s,e,n);
    if(s>r || e<l) return 0;
    if(s>=l && e<=r)    return tree[n].value;
    return query(left,l,r) + query(right,l,r);
}
```

```
int main()
{
   // freopen("in.txt","r",stdin);
    // freopen("final_output.txt","w",stdout);
    long long tc, cn=0;
    scanf("%lld",&tc);
    while(tc--)
    {

        long long q;
        scanf("%lld",&q);
        memset(tree,0,sizeof(tree));
        build(call);

        printf("Case %lld:\n",++cn);

        while(q--)
        {
            char  c;
            long long l, r, v;
            scanf(" %c",&c);
            if(c=='A')
            {
                scanf("%lld %lld",&l,&r);
                update(call,l,r,2);
            }
            if(c=='B')
            {
                scanf("%lld %lld",&l,&r);
                update(call,l,r,3);
            }
            if(c=='S')
            {
                scanf("%lld %lld",&l,&r);
                long long res = query(call,l,r);
                printf("%lld\n",res);
                //break;
            }
            if(c=='C')
            {
                scanf("%lld %lld %lld",&l,&r,&v);
                update(call,l,r,1,v);
            }

        }
    }
    return 0;
}

/*
1
11
B 1 2
```

```
C 2 5 4
B 1 3
C 1 2 3
B 1 2
C 1 5 3
A 1 4
B 3 7
A 2 6
B 1 2
S 2 3
*/
```

Rectangle Union

Problems

```
#include <bits/stdc++.h>
#define left s,(s+e)/2, n+n
#define right (s+e)/2+1, e, n+n+1
#define call 1,a.size()-1,1
#define MAX    100000
using namespace std;

struct node
{
    int full_update, total_count,down_count,node_size;
    node()
    {
        full_update = total_count = down_count = node_size = 0;
        node_size = 1;
    }
};

node tree[4*MAX];
vector < int > a;

void merge_node(int s, int e, int n)
{
    tree[n].down_count = tree[n+n].total_count + tree[n+n+1].total_count;
    tree[n].node_size = tree[n+n].node_size + tree[n+n+1].node_size;
    if(tree[n].full_update) tree[n].total_count  = tree[n].node_size;
    else tree[n].total_count = tree[n].down_count;
}

void update_single_node(int s, int e, int n,int add)
{
    tree[n].full_update+=add;
    if(s!=e) merge_node(s,e,n);
    else
    {
```

```
        if(tree[n].full_update)
        {
            tree[n].node_size=a[s]-a[s-1];
            tree[n].total_count = a[s]-a[s-1];
            tree[n].down_count=0;
        }
        else {
            tree[n].node_size=a[s]-a[s-1];
            tree[n].total_count = tree[n].down_count=0;
        }

    }
}

void build(int s, int e, int n)
{
    if(s==e)
    {
        tree[n]=node();
        tree[n].node_size  = a[e]-a[e-1];

        return ;
    }
    build(left);
    build(right);
    merge_node(s,e,n);
}

void update(int s, int e, int n, int l, int r,int add)
{
    update_single_node(s,e,n,0);
    if(s>r || e<l) return ;
    if(s>=l && e<=r)
    {
        update_single_node(s,e,n,add);
        return;
    }
    update(left,l,r,add);
    update(right,l,r,add);
    merge_node(s,e,n);
}

struct info
{
    int prio, s, e,add;
    info()
    {
        prio =s=e=add=0;
    }
    info(int p, int ss,int ee,int a)
    {
        prio = p;
        s = ss;
```

```
        e = ee;
        add= a;
    }
};
info q[MAX];
bool cmp(info a, info b)
{
    return a.prio < b.prio ;
}

int main()
{
   // test();

    int test_case, case_number=0;
    scanf("%d",&test_case);
    while(test_case--)
    {
        a.clear();
        case_number++;
        int n;
        scanf("%d",&n);
        for(int i=0; i<n; i++)
        {
            int x1, x2,y1,y2;
            scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
            q[2*i] =     info(x1,y1,y2,1);
            q[2*i + 1] = info(x2,y1,y2,-1);
            a.push_back(y1);
            a.push_back(y2);
        }
        sort(a.begin(),a.end());
        map < int ,  int  > ma;
        for(int i=0;i<a.size();i++){
            ma[a[i]]=i;
        }
        build(call);
        long long ans=0;
        long long  pre=0;
        long long  last ;
        sort(q,q+2*n,cmp);

        for(int i=0; i<n+n; i++)
        {
            //cout<<ma[q[i].s]+1<<" "<<ma[q[i].e]<<" "<<endl;
            update(call,ma[q[i].s]+1,ma[q[i].e],q[i].add);
            ans+=pre*(q[i].prio-last);
            pre=tree[1].total_count;
            //cout<<pre<<endl;

            last = q[i].prio;

        }
```

```
        printf("Case %d: %lld\n",case_number,ans);

    }
    return 0;
}
```

Knight Distance

Miscellaneous

```
ll knight_move(ll x,ll y){
    ll a,b,z,c,d;
    x = abs(x) , y = abs(y);
    if(x < y) swap(x,y);
    if( x == 2 && y == 2 )return 4;
    if( x == 1 && y == 0 )return 3;

    if( y == 0 || (y << 1) < x ){
        c = y & 1;
        a = x - ( c << 1 ), b = a & 3;
        return ((a-b) >> 1) + b + c;
    }
    else{
        d = x - ( (x-y) >> 1 );
        z = ((d % 3) != 0 ), c = (x - y) & 1;
        return ((d/3) * 2 ) + c + (z * 2 * (1 - c));
    }
}
```

Largest Rectangle in Histogram

Problem: Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars.
Solution: For every bar X, we calculate the area with X as the smallest bar in the rectangle. Traverse all bars from left to right, maintain a stack of bars. Every bar is pushed to stack once. A bar is popped from stack when a bar of smaller height is seen. When a bar is popped, we calculate the area with the popped bar as smallest bar.

Greedy

```
int st[MAX] , top;
//given array must have zero as first element
// n+1 element with first zero
int histogram(int a[], int n){
    top=0;
    int i=1, res=0;
```

```
    a[++n]=0, st[top]=0;
    while(i<=n){
        if(a[st[top]]<=a[i]) st[++top]=i++;
        else{
            int bar = a[st[top]];
            --top;
            res  = max ( res , (i-st[top]-1)*bar);
        }
    }
    return res;
}
```

Stable Marriage

Overview: Given N men and N women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, all the marriages are "stable" Solution: The idea is to iterate through all free men while there is any free man available. Every free man goes to all women in his preference list according to the order. For every woman he goes to, he checks if the woman is free, if yes, they both become engaged. If the woman is not free, then the woman chooses either says no to him or dumps her current engagement according to her preference list. So an engagement done once can be broken if a woman gets better option.