Team Name: ULAB_Disjoint_Skill_Union
Team Member 1: Rakib Ahmed
Team Member 2: Md. Amanullah
Team Member 3: Ahmed Shahriar Tanvir

```
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x))
#define fast_io ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
typedef long long ll;
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
```

sieve_and_seg_sieve:

```
bool isprime[M];
vector<ll>prime;
void sieve_of_eratosthenes() {
  for (ll i = 3; i * i <= M; i++) {
    if (!isprime[i]) { for (ll j = i * i; j <= M; j += i) isprime[j] = true; } }
  prime.push_back(2);
  for (ll i = 3; i <= M; i += 2) if (!isprime[i])prime.push_back(i); }
void segmented_sieve(ll l, ll r) {
  ll size = r - l + 1;
  bool is_segsive[size];
  for (ll i = 0; i < size; i++) is_segsive[i] = true;
  for (ll i = 0; prime[i]*prime[i] <= r; i++) {
    ll curren_prime = prime[i];
    ll base = (l / curren_prime) * curren_prime;
    if (base < l) base += curren_prime;
    for (ll j = base; j <= r; j += curren_prime) is_segsive[j - l] = false;
    if (base == curren_prime)is_segsive[base - l] = true; }
  for (ll i = 0; i < size; i++) if (is_segsive[i] && i + l != 1) cout << i + l << endl; }
```

mobius:

```
short int mobius[MAX];
void sieve() { mobius[1] = 1; for (register int i = 1; i < MAX; ++i) for (register int j = i + i; j < MAX; j += i) mobius[j] -= mobius[i]; }
```

Matrix Exponentiation:

```
#define Mat_type int
class Matrix {
  public:
  int M_row, M_col;
  vector<vector<Mat_type>> mat;
  Matrix(int _M_row, int _M_col) {
    M_row = _M_row;
    M_col = _M_col;
    mat.clear();
    mat.resize(M_row, vector<Mat_type>(M_col, 0));
    if (M_row == M_col) { for (int i = 0; i < M_row; i++) mat[i][i] = 1; } } //identity matrix
  Matrix operator* (const Matrix& other) { // '*' operator overloading
    int n_r = M_row;
    int n_c = other.M_col;
    Matrix new_mat(n_r, n_c);
    for (int i = 0; i < n_r; i++) {
      for (int j = 0; j < n_c; j++) {
        int sum = 0;
        for (int k = 0; k < M_col; k++) sum += mat[i][k] * other.mat[k][j]; }
        new_mat.mat[i][j] = sum; }
    return new_mat; }
  Matrix operator^(ll p) { // mat^p
    Matrix res(M_row, M_col);
    Matrix x = *this; //copy current obj(matrix)
    while (p) {
      if (p & 1) res = res * x;
      p /= 2;
      x = x * x; }
    return res; }
  Matrix operator + ( Matrix b) { // for square matrices
    int r = M_row;
    int c = M_col;
    Matrix res(r, c);
    for (int i = 0; i < r; i++) for (int j = 0; j < c; j++) res.mat[i][j] = mat[i][j] + b.mat[i][j];
    return res; }
  Matrix operator - (Matrix b) { // for square matrices
    int r = M_row;
    int c = M_col;
    Matrix res(r, c);
    for (int i = 0; i < r; i++) for (int j = 0; j < c; j++) res.mat[i][j] = mat[i][j] - b.mat[i][j];
    return res; } }; // a = result which is in obj.mat[0][0]
```

Binary Exponentiation

```
ll binaryExponentiation(ll x, ll n) { if (n == 0) return 1; else if (n % 2 == 0) return binaryExponentiation(x * x, n / 2); else return x * binaryExponentiation(x * x, (n - 1) / 2); }
```

Extended GCD:

```
ll gcdExtended(ll a, ll b, ll *x, ll *y) {
  if (a == 0) { *x = 0; *y = 1; return b; }
  ll x1, y1;
  ll gcd = gcdExtended(b % a, a, &x1, &y1);
  *x = y1 - (b / a) * x1;
  *y = x1;
  return gcd; }

ll expo(ll a, ll b, ll mod) {ll res = 1; while (b > 0) {if (b
& 1)res = (res * a) % mod; a = (a * a) % mod; b = b
>> 1;} return res;}
ll modInvPow(ll a, ll b, ll p) {return
modPow(modPow(a, p - 2, p), b, p) % p;}
```

Maximum Divisors in Range

```
//Finds x such that x <= n and has maximum
number of divisors
#include <bits/stdc++.h>
using namespace std;
#define ULL unsigned long long int
ULL n, res, idx;
int p, primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
31, 37, 41, 43, 47, 53, 59, 61, 67, 71};
ULL mul(ULL a, ULL b){
  ULL res = 0;
  while (b){
    if (b & 1LL) res = (res + a);
    if (res > n) return 0;
    a = (a << 1LL);
    b >>= 1LL;
  }
  return res;
}
void backtrack(int i, int lim, ULL val, ULL r){
  if ((r > res) || (r == res && val < idx)) res = r, idx =
val;
  if (i == p) return;

  ULL x = val;
  for (int d = 1; d <= lim; d++){
    x = mul(x, primes[i]);
    if (x == 0) return;
    backtrack(i + 1, d, x, r * (d + 1));
  }
}
int main(){
  /* Tested for <= 10^18 */
  p = sizeof(primes) / sizeof(int);
  res = 0;
  scanf("%llu", &n);
  backtrack(0, 100, 1, 1);
  printf("%llu = %llu\n", idx, res);
}
```

Fibonacci Combination Sums

```
#define MOD 1000000007
#define MAXK 205
ll bigMOD(ll n,ll r){
  if(r==0) return 1LL;
  ll ret = bigMOD(n,r/2);
  ret = (ret*ret)%MOD;
  if(r%2==1) ret = (ret*n)%MOD;
  return ret; }
ll inverse(ll x) {return bigMOD(x%MOD, MOD - 2);}
ll cMOD(ll x) {x%=MOD; return (x>=0) ? x :
x+MOD;}
struct cong{
  ll a, b;
  cong (ll x, ll y) {a = cMOD(x); b = cMOD(y);}
  cong operator + (const cong &x) const { return
cong(a + x.a, b + x.b); }
  cong operator - (const cong &x) const { return
cong(a - x.a, b - x.b); }
  cong operator * (const cong &x) const {
    ll c = x.a;
    ll d = x.b;
    return cong((a * c) + (5 * b * d), (a * d) + (b *
c)); }
  cong operator / (ll x) const { return cong(a *
inverse(x), b * inverse(x)); }
  cong operator / (const cong g) const {
    ll c = g.a;
    ll d = g.b;
    ll x = cMOD(a*c - 5*b*d) * inverse(c*c -
5*d*d);
    ll y = cMOD(a*d - b*c) * inverse(5*d*d - c*c);
    return cong(x, y); } };
cong bigMOD(cong n, ll r) {
  if(r==0) return cong(1, 0);
  cong ret = bigMOD(n,r/2);
  ret = ret * ret;
  if(r%2 == 1) ret = ret * n;
  return ret; }
cong geosum(cong x, ll n) {
  if(x.a == 1 && x.b == 0) return cong(n+1, 0);
  return (bigMOD(x, n+1) - cong(1, 0)) / (x -
cong(1, 0)); }
cong alpha = cong(1, 1) / 2;
cong beta  = cong(1, -1) / 2;
ll c[MAXK][MAXK];
ll a[MAXK][MAXK];
ll fibPower(ll n, int m) {
  cong ans = cong(0, 0);
  for(int j = 0; j <= m; j++) {
    cong t = bigMOD(alpha, j) * bigMOD(beta, m -
j);
    cong x = cong(c[m][j], 0) * geosum(t, n);
    if((m - j) & 1) ans = ans - x;
    else ans = ans + x; }
  ans = ans * bigMOD((cong(0, 1) / 5), m);
  return ans.a; }
void PreCalc(){
  for(int i = 0; i < MAXK; i++) {
    c[i][0] = 1;
    for(int j = 1; j <= i; j++) {
      c[i][j] = c[i - 1][j] + c[i - 1][j - 1];
      c[i][j] %= MOD; } }
  a[0][0] = 1;
  for(int i = 1; i < MAXK; i++) {
    a[i][0] = 1;
```

```
        for(int j = 1; j <= i+1; j++) {
            a[i][j] = a[i - 1][j] + i * a[i - 1][j - 1];
            a[i][j] %= MOD; } } }
// query(n,k) = sum of C(Fib(i),k) in range 0 <= i <=n
ll query(ll n, int k) {
    if(n < 0) return 0;
    ll ans = 0;
    for(int i = 0; i <= k; i++) {
        ll x = a[k - 1][i] * (fibPower(n + 2, k - i) - 1);
        if(i & 1) ans = cMOD(ans - x);
        else ans = cMOD(ans + x); }
    for(int i = 1; i <= k; i++) {ans *= inverse(i); ans %=
MOD;}
    return ans; }
int main(){
    PreCalc();
    ll k, l, r;
    scanf("%lld %lld %lld",&k,&l,&r);
    ll ans = query(r, k) - query(l-1, k);
    cout << cMOD(ans) << endl; }
```

Divisor Count by prime factorization

```
#count diviosr vector<int> primes; // we'll preload
primes once at the beginning int countDivisor(int
n) { int divisor = 1; for (int i = 0; i < primes.size();
i++) { if (n % primes[i] == 0) { int cnt = 1; while (n %
primes[i] == 0) { n /= primes[i]; cnt++; } divisor *=
cnt; } } return divisor; }
```

phi_function:

```
vector<ll>phi(M);
void phic() {
    for (int i = 2; i <= M; i++) phi[i] = i;
    for (int i = 2; i <= M; i++) { if (phi[i] == i) for (int j
= i; j <= M; j += i) phi[j] -= (phi[j] / i); }
    phi[0] = 0;
    phi[1] = 1; }
```

Factorials with nCr

```
int fact[N], invfact[N];
int pow(int a, int b, int m) {
            int ans=1;
            while(b) {
                        if(b&1) ans=(ans*a)%m;
                        b/=2;
                        a=(a*a)%m; }
            return ans; }
int modinv(int k) { return pow(k, MOD-2, MOD); }
void precompute() {
            fact[0]=fact[1]=1;
            for(int i=2;i<N;i++) {
                        fact[i]=fact[i-1]*i;
                        fact[i]%=MOD; }
            invfact[N-1]=modinv(fact[N-1]);
            for(int i=N-2;i>=0;i--) {
                        invfact[i]=invfact[i+1]*(i+1);
                        invfact[i]%=MOD; } }
int nCr(int x, int y) {
            if(y>x) return 0;
            int num=fact[x];
            num*=invfact[y];
            num%=MOD;
            num*=invfact[x-y];
            num%=MOD;
            return num; }
```

#Euler totient upto a number n

```
def phi_range(n):
    n += 1
    phi = [i for i in range(n)]
    for i in range(2, n):
        if phi[i] == i:
            for j in range(i, n, i):
                phi[j] -= phi[j]//i
    return phi
```

Sum of Geometric Series in O((log(n)) ^ 2)

```
// it calculates
(base^1+base^2+base^3+base^4+.........+base^n)%
MOD in O((log(n))^2) time.
// if n==0 , you have to manually return/add 1 for
that
ll func(ll base, ll n) {
    if (n == 1) return bigMod(base, n);
    if (n % 2 == 0) {
        ll temp = func(base, n / 2);
        return (temp + (bigMod(base, n / 2) * temp));}
    return (bigMod(base, n) + func(base, n - 1)); }
```

Chinese Remainder Theorem:

```
#define no_eqn 20
ll modPow(ll a, ll b, ll MOD) {
    if (b == 0) return 1LL;
    if (b % 2 == 0) {
        ll temp = modPow(a, b / 2, MOD) % MOD;
        return (temp * temp) % MOD; }
    return (a * modPow(a, b - 1, MOD)) % MOD; }
ll modInv(ll a, ll b) { return modPow(a, b - 2, b); }
struct point { int val, m; };
point ara[no_eqn];
ll CRT(ll sz) {
    ll x = 0; // x=val[i](mod m[i])
    ll M = 1;
    for (int i = 0; i < sz; i++) M *= ara[i].m;
    ll Midx[14];
    ll MInv[14];
    for (int i = 0; i < sz; i++) {
        Midx[i] = M / ara[i].m;
        MInv[i] = modInv(Midx[i], ara[i].m); }
    for (int i = 0; i < sz; i++) x = (x + (ara[i].val *
Midx[i] * MInv[i]) % M) % M; //
x=val[i]*Inv(Mi)*(Mi) where Mi means
(m0*m1*m2*...*mn)/mi;
    return x; }
```

PBDS:

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree <int,null_type,less< int >,rb_tree_tag,tree_order_statistics_node_update> ordered_set;
int arr[100010], brr[100010];
int main() {
  ordered_set x;
  int n;
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) scanf("%d", &arr[i]);
  for (int i = 1; i <= n; i++) scanf("%d", &brr[i]);
  for (int i = 1; i <= n; i++) {
    x.insert(arr[i]);
    printf("%d\n", *x.find_by_order(brr[i] - 1)); }
  return 0; }
```

nth Catalan Number:

```
void catalan(int n){
cpp_int cat_ = 1;
cout << cat_ << " ";
for (cpp_int i = 1; i < n; i++){
cat_ *= (4 * i - 2);cat_ /= (i + 1);
cout << cat_ << " ";}}catalan(n);
```

Kadane algo:

```
int maxSumSubArray(vector<int>&A) {
  int n = A.size();
  int local_max = 0;
  int global_max = INT_MIN;
  for (int i = 0; i < n; i++) {
    local_max = max(A[i], A[i] + local_max);
    global_max = max(global_max, local_max); }
  return global_max; }
cout << maxSumSubArray(A);
```

LCA:

1) K - th node from node to node .

2) K - th node from node

3) Lowest Common Ancestor of two given nodes and

4) Max Edge Value in a weighted tree from node to node.

```
#define MAX 10001
#define LOGN 14 // be VERY VERY CAREFUL while setting this!
#define pb push_back
vector<int>gr[MAX + 10];
vector<int>cost[MAX + 10]; // omit this for unweighted graph
bool vis[MAX + 10];
int level[MAX + 10], khoroch[MAX + 10];
int par[MAX + 10][20], weight[MAX + 10][20];
int node, a, b, c;
void dfs(int u){
  vis[u] = true;
  for (int i = 0; i < gr[u].size(); i++) {
    int v = gr[u][i];
    if (!vis[v]) {
      par[v][0] = u;
      level[v] = level[u] + 1;
      khoroch[v] = khoroch[u] + cost[u][i]; // omit this for unweighted graph
      dfs(v); } }
  return; }
void reset() {
  memset(par, -1, sizeof(par));
  memset(vis, false, sizeof(vis));
  memset(level, false, sizeof(level));
  memset(khoroch, false, sizeof(khoroch)); // omit this for unweighted graph
  for (int i = 0; i < MAX; i++) {
    gr[i].clear();
    cost[i].clear(); // omit this for unweighted graph
  } }
void makeSparseTable() {
  dfs(1);
  for (int j = 1; (1 << j) < node; j++) {
    for (int i = 1; i <= node; i++) {
      if (par[i][j - 1] != -1) {
        par[i][j] = par[par[i][j - 1]][j - 1]; // par[i][j] is defined by : (2^j)-th parent of node i
        weight[i][j] = max(weight[i][j - 1], weight[p[i][j - 1]][j - 1]); } } } }
  return; }
int LCA(int a, int b) {
  if (a == b) return a;
  if (par[a][0] == par[b][0]) return par[a][0];
  if (level[b] > level[a]) swap(b, a);
  for (int i = LOGN; i >= 0; i--) {
    if (par[a][i] != -1) {
      if (level[a] - (1 << i) >= level[b]) a = par[a][i];
} }
  if (a == b) return a;
  for (int i = LOGN; i >= 0; i--) {
    if (par[b][i] != -1 && par[a][i] != -1 && par[b][i] != par[a][i]) {
      b = par[b][i];
      a = par[a][i]; } }
  return par[a][0]; }
```

Trie:

```
struct trieNode {
  bool isEnd;
```

```
   trieNode* child[26];
   trieNode() {
      isEnd = 0;
      for (int i = 0; i < 26; i++) child[i] = NULL; } };
void Insert(trieNode* root, string &str) {
   trieNode* curr = root;
   for (int i = 0; i < (int)str.size(); i++) {
      int ch = str[i] - 'a'; // current character of
string
      if (curr -> child[ch] == NULL) // curr theke 'ch'
borabor kono rasta ache kina, NULL maane rasta
nai
      { curr -> child[ch] = new trieNode(); } // rasta
create kortesi
      // ei scope mean kore je rasta create kora
hoye gese / aage thekei rasta ase
      curr = curr -> child[ch]; }
   curr -> isEnd = true; }
bool Search(trieNode* root, string &str){
   trieNode* curr = root;
   for (int i = 0; i < (int)str.size(); i++) {
      int ch = str[i] - 'a';
      if (curr -> child[ch] == NULL) return false;
      curr = curr -> child[ch]; }
   return curr && curr -> isEnd; }
string str;
void dfs(trieNode* root) {
   if (root && root->isEnd) cout << str << "\n";
   if (root == NULL) return;
   for (int i = 0; i < 26; i++) {
      if (root -> child[i]) {
         str.push_back(i + 'a');
         dfs(root -> child[i]);
         str.pop_back(); } } } }
int main() {
   trieNode* root = new trieNode();
   string str = "alice";
   Insert(root, str);
   str = "bob";
   Insert(root, str);
   str = "all";
   Insert(root, str);
   str = "boss";
   Insert(root, str);
   str = "bossy";
   Insert(root, str);
   str = "";
   dfs(root);
   return 0; }
```

Lazy propagation:

```
#define left st, (st + en) / 2, nd + nd
#define right ((st + en) / 2) + 1, en, nd + nd + 1
ll tree[4 * MAX + 5], lazy[4 * MAX + 5], arr[4 * MAX
+ 5];
void pushDown(int st, int en, int nd) {
   if (!lazy[nd] || st == en) return;
   int mid = (st + en) / 2;
   tree[nd + nd] += (mid - st + 1) * lazy[nd];
   tree[nd + nd + 1] += (en - mid) * lazy[nd];
   lazy[nd + nd] += lazy[nd];
   lazy[nd + nd + 1] += lazy[nd];
   lazy[nd] = 0; }
void update(int st, int en, int nd, int L, int R, int val)
{
   pushDown(st, en, nd);
   if (R < st || en < L) return;
   if (L <= st && en <= R) {
      tree[nd] = tree[nd] + (val * (en - st + 1));
      lazy[nd] += val;
      return; }
   update(left, L, R, val); /// left
   update(right, L, R, val); /// right
   tree[nd] = tree[nd + nd] + tree[nd + nd + 1]; }
ll query(int st, int en, int nd, int L, int R) {
   pushDown(st, en, nd);
   if (R < st || en < L) return 0;
   if (L <= st && en <= R) return tree[nd];
   return query(left, L, R) + query(right, L, R); }
int main() {
   int n, T, q, L, R, val, com, cs = 1;
   scanf("%d", &T);
   while (T--) {
      scanf("%d %d", &n, &q);
      for (int i = 0; i <= 4 * n; i++) arr[i] = lazy[i] =
tree[i] = 0;
      printf("Case %d:\n", cs++);
      while (q--) {
         scanf("%d", &com);
         if (com == 0) {
            scanf("%d %d %d", &L, &R, &val);
            update(0, n - 1, 1, L, R, val); }
         else {
            scanf("%d %d", &L, &R);
            printf("%lld\n", query(0, n - 1, 1 , L , R)); }
} }
   return 0; }
```

Miller:

```
ll rp(ll a, ll b, ll mod) {
   ll res = 0;
   while (b > 0) {
      if (b % 2 == 1) res = (res + a) % mod;
      a = (a << 1) % mod;
      b = (b >> 1); }
   return res; }
ll binpower(ll base, ll e, ll mod) {
   ll result = 1;
   base %= mod;
   while (e) {
      if (e & 1) result = rp(result, base, mod);
      base = rp(base, base, mod);
      e >>= 1; }
   return result; }
```

```
bool check_composite(ll n, ll a, ll d, int s) {
   ll x = binpower(a, d, n); /// i) a^d % n
   if (x == 1 || x == n - 1) return false; /// n probably prime
   for (int r = 0; r < s - 1; r++) {
      x = rp(x, x, n); /// ii) x^2 % n
      if (x == n - 1) return false; /// n probably prime
   } return true; /// n composite
};
bool MillerRabin(ll n)   // returns true if n is prime, else returns false.
{
   if (n < 2) return false;
   if (n == 2) return true;
   if (n % 2 == 0) return false;
   int s = 0;
   ll d = n - 1;
   while ((d & 1) == 0) { d >>= 1; s++; }
   vector<int>bases{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
   for (int a : bases) /// O(|base| * (logN)^2) = O(12 * 3600)
   {
      if (n == a) return true;
      if (check_composite(n, a, d, s)) return false;
   }
   return true; /// highly probable that n is a prime number
}
```

NcR:

```
void printNcR(int n, int r) {
   ll p = 1, k = 1;
   if (n - r < r) r = n - r;
   if (r != 0) {
      while (r) {
         p *= n;
         k *= r;
         ll m = __gcd(p, k);
         p /= m;
         k /= m;
         n--;
         r--; } }
   else p = 1;
   cout << p << endl; }
```

LIS:

```
/* Finds only LIS. LDS can be found by simply multiplying the whole input array with -1.
For Longest Non-Decreasing sequence, simply use upper_bound().
Complexity: NlogK */
struct LIS {
   int bbb[NSIZE + 10];
   int calculateLIS ( int arr[], int lisVal[], int n ) {
      FOR(i, 0, n) { bbb[i] = inf; }
      bbb[0] = -inf;
      int mx = 0;
      FOR(i, 0, n - 1) {
         int v = arr[i];
         int pos = lower_bound ( bbb, bbb + mx + 1, v ) - bbb;
         lisVal[i] = pos;
         bbb[pos] = v;
         mx = MAX(mx, pos); }
      return mx; } } lis;
```

K_th_NODE:

```
int kthNode(int a1, int an, int kth) {
   int w = LCA(a1, an);
   int d1, dn;
   int from;
   d1 = level[a1] - level[w] + 1;
   dn = level[an] - level[w] + 1;
   if (d1 == kth) return w;
   else if (d1 > kth) {
      from = a1;
      kth--; }
   else {
      from = an;
      kth = d1 + dn - kth - 1; }
   int lg = LOGN;
   while (kth > 0 && lg >= 0) {
      if ((1 << lg) <= kth) {
         from = par[from][lg];
         kth -= (1 << lg); }
      --lg; }
   return from; }
```

Segment_tree:

```
int build(int a[], int v, int tl, int tr) {
   if (tl == tr) return st[v] = a[tl];
   int tm = (tl + tr) >> 1;
   return st[v] = min(build(a, v << 1, tl, tm), build(a, (v << 1) + 1, tm + 1, tr)); }
int query(int v, int tl, int tr, int l, int r) {
   if (l > r) return INF;
   if (l == tl && r == tr)  return st[v];
   int tm = (tl + tr) >> 1;
   return min(query(v << 1, tl, tm, l, min(r, tm)), query((v << 1) + 1, tm + 1, tr, max(l, tm + 1), r)); }
void update(int v, int tl, int tr, int pos, int new_val)
{
   if (tl == tr) { st[v] = new_val; return; }
   int tm = (tl + tr) >> 1;
   if (pos <= tm) update(v << 1, tl, tm, pos, new_val);
   else update((v << 1) + 1, tm + 1, tr, pos, new_val);
   st[v] = min(st[(v << 1)], st[(v << 1) + 1]); }
build(a, 1, 0, n - 1);
update(1, 0, n - 1, k - 1, u);
query(1, 0, n - 1, a - 1, b - 1)
```

Segment_tree_lazy:

```
#define M 100005
#define left start,(start+end)/2,node+node
#define right (end+start)/2+1,end,node+node+1
#define VI vector<long long>
VI tree(4 * M);
VI lazy(4 * M);
VI arr(M + 5);
/* ll arr[M]; ll tree[4 * M]; ll lazy[4 * M]; */
void Tree_build(ll start, ll end, ll node) {
  if (start == end) { tree[node] = arr[start]; return;
}
  Tree_build(left);
  Tree_build(right);
  tree[node] = tree[node + node] + tree[node +
node + 1]; }
void update(ll start, ll end, ll node, ll L, ll R, ll value)
{
  if (lazy[node] != 0) {
    tree[node] += (end - start + 1) * lazy[node];
    if (start != end) {
      lazy[node + node] += lazy[node];
      lazy[node + node + 1] += lazy[node]; }
    lazy[node] = 0; }
  if (start > R || end < L) return;
  if (start >= L && end <= R) {
    tree[node] += (end - start + 1) * value;
    if (start != end) {
      lazy[node + node] += value;
      lazy[node + node + 1] += value; }
    return; }
  update(left, L, R, value);
  update(right, L, R, value);
  tree[node] = tree[node + node] + tree[node +
node + 1]; }
ll query(ll start, ll end, ll node, ll L , ll R) {
  if (start > R || end < L)return 0;
  if (lazy[node] != 0) {
    tree[node] += (end - start + 1) * lazy[node];
    if (start != end) {
      lazy[node + node] += lazy[node];
      lazy[node + node + 1] += lazy[node]; }
    lazy[node] = 0; }
  if (start >= L && end <= R) return tree[node];
  return query(left, L, R) + query(right, L, R); }
fill(tree.begin(), tree.end(), 0);
fill(lazy.begin(), lazy.end(), 0);
update(0, n - 1, 1, l, r, v);
query(0, n - 1, 1, l, r)
```

Dynamic Segment Tree

```
/* Implicit segment tree with addition on the
interval and getting the value of some
element.Works on the intervals like [1..10^9]. */
struct Node {
  ll sum;
  Node *l, *r;
  Node() : sum(0), l(NULL), r(NULL) { } };
void add(Node *v, int l, int r, int q_l, int q_r, ll val) {
  if (l > r || q_r < l || q_l > r) return;
  if (q_l <= l && r <= q_r) {
    v -> sum += val;
    return; }
  int mid = (l + r) >> 1;
  if (v -> l == NULL) v -> l = new Node();
  if (v -> r == NULL) v -> r = new Node();
  add(v -> l, l, mid, q_l, q_r, val);
  add(v -> r, mid + 1, r, q_l, q_r, val); }
ll get(Node *v, int l, int r, int pos) {
  if (!v || l > r || pos < l || pos > r) return 0;
  if (l == r) return v -> sum;
  int mid = (l + r) >> 1;
  return v -> sum + get(v -> l, l, mid, pos) + get(v ->
r, mid + 1, r, pos); }
int n, m, t, x, y, val;
char c;
int main() {
  Node *root = new Node();
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &x);
    add(root, 0, n - 1, i, i, x); }
  scanf("%d", &m);
  for (int i = 0; i < m; i++) {
    scanf("\n%c", &c);
    if (c == 'a') {
      scanf("%d%d%d", &x, &y, &val);
      add(root, 0, n - 1, --x, --y, val);
    } else {
      scanf("%d", &x);
      printf("%I64d ", get(root, 0, n - 1, --x)); } }
  return 0; }
```

bellmand_ford:

```
vector<tuple<int, int, long long > > EdgeList;long
long dist[MX];int n, m;
bool bellman(int u) {memset(dist, INF, sizeof
dist);const long long LLINF = dist[0];
dist[u] = 0LL;for (int i = 1, updated = true; i < n &&
updated; i++) {
updated = false;for (auto [u, v, w] : EdgeList) {if
(dist[u] == LLINF) continue;
// u is still not reachable, so correct
implementation **requires** to skip it
if (dist[u]+w < dist[v]) { dist[v] = min(LLINF,
dist[u]+w); //necessary to limit distance to INF to
avoid possible overflow
updated = true;}}}// following code is needed only
to check negative cycle
for (auto [u, v, w] : EdgeList) {if (dist[u] == LLINF)
continue;if (dist[u] + w < dist[v]) return false;
}return true;}
int main() {scanf("%d %d", &n, &m); // number of
nodes and edges
for (auto [i, u, v, w] = tuple{0, 0, 0, 0LL}; i < m; i++)
{ //input nodes
scanf("%d %d %lld", &u, &v,
&w);EdgeList.emplace_back(tuple{u, v, w});}
bellman(1);//1 based index was used for testingfor
(int i = 1; i <= n; i++) printf("%lld ", dist[i]);
printf("\n");
return 0;}
```

Bipartite Matching (Hopcroft Karp)

```
#define mset0(x) memset(x,0,sizeof(x))
const int maxN = 50000 ;
const int maxM = 50000 ;
struct HopcroftKarp {
   int vis[maxN], level[maxN], ml[maxN],
mr[maxM];
   vector<int> edge[maxN]; // constructing edges
for left part only
   void init(int n) {
      for (int i = 1; i <= n; ++i) edge[i].clear();
   }
   void add(int u, int v) {
      edge[u].push_back(v);
   }
   bool dfs(int u) {
      vis[u] = true;
      for (vector<int>::iterator it = edge[u].begin();
it != edge[u].end(); ++it) {
         int v = mr[*it];
         if (v == -1 || (!vis[v] && level[u] < level[v]
&& dfs(v))) {
            ml[u] = *it;
            mr[*it] = u;
            return true;
         }
      }
      return false;
   }
   int matching(int n) { // n for left
      mset0(vis);
      mset0(level);
      memset(ml, -1,sizeof(ml));
      memset(mr, -1,sizeof(mr));
      for (int match = 0;;) {
         queue<int> que;
         for (int i = 1; i <= n; ++i) {
            if (ml[i] == -1) {
               level[i] = 0;
               que.push(i);
            } else level[i] = -1;
         }
         while (!que.empty()) {
            int u = que.front();
            que.pop();
            for (vector<int>::iterator it =
edge[u].begin(); it != edge[u].end(); ++it) {
               int v = mr[*it];
               if (v != -1 && level[v] < 0) {
                  level[v] = level[u] + 1;
                  que.push(v);
               }
            }
         }
         for (int i = 1; i <= n; ++i) vis[i] = false;
         int d = 0;
         for (int i = 1; i <= n; ++i) if (ml[i] == -1 &&
dfs(i)) ++d;
         if (d == 0) return match;
         match += d;
      }
   }
};
```

Dijkstra:

```
int n, m;long long dist[MX];bool processed[MX];
vector < vector<tuple<int, long long> > >  AdjList;
void dijkstra(int u ) {priority_queue< tuple<long
long, int>,
vector<tuple<long long, int> >, greater<tuple<long
long, int> > > pq;
memset(dist, INF, sizeof dist);dist[u] =
0;pq.push(tuple {0, u});
while(!pq.empty()) {auto [d, u] = pq.top();
pq.pop();
if (processed[u]) continue;   //important as push in
pq does not replace previous entry
processed[u] = true;for (auto [v, w] : AdjList[u]) {
if (dist[u] + w < dist[v]) {dist[v] = dist[u] + w;
pq.push(tuple {dist[v], v});}}}}
int main() {scanf("%d %d", &n, &m); // number of
nodes and edges
AdjList.resize(n+1, vector<tuple<int, long long>
>());
for (auto [i, u, v, w] = tuple{0, 0, 0, 0LL}; i < m; i++)
{ //input nodes
scanf("%d %d %lld", &u, &v, &w);
AdjList[u].emplace_back(tuple {v, w});}
dijkstra(1);//1 based index was used for testing
for (int i = 1; i <= n; i++) printf("%lld ", dist[i]);
printf("\n");
return 0;}
```

floyed_worshall:

```
int n, m;long long AdjMat[MX][MX];
void floydWarshall () {scanf("%d %d", &n, &m);
//Matrix initailization//1 based indexing
for (int i = 1; i <= n; i++) {
memset(AdjMat[i], INF, sizeof AdjMat[i]); //All
distance set to INF
AdjMat[i][i] = 0LL;  //except distance to the node
itself}
//Input distaces;long long w;
for (int i = 0, u, v; i < m; i++) {
scanf("%d %d %lld", &u, &v, &w);
AdjMat[u][v] = min(AdjMat[u][v], w); //storing
minimum distance is important if multiple edges
exits between two nodes
//AdjMat[v][u] = min(AdjMat[v][u], w); //if
bidirectional use this, but be careful of negative
edge}
//Algorithm ;for (int k = 1; k <= n; k++) {   //k is the
intermidiate node
for (int i = 1; i <= n; i++) {
for (int j = 1; j <= n; j++) {
AdjMat[i][j] = min(AdjMat[i][j], AdjMat[i][k] +
AdjMat[k][j]);}}}}
int main () {//test code;
floydWarshall();for (int i = 1; i <= n; i++){
for (int j = 1; j <= n; j++)
printf("%lld ", AdjMat[i][j]); printf("\n");}return 0;}
```

Articulation Bridge

```
#define MAX 100005
#define f first
#define s second
int cnt;
vector<vector<int> > graph(MAX);
int d[MAX],md[MAX],vis[MAX];
map<pair<int,int>,int> bridges;
int all_bridges(int now,int from) {
   d[now]=md[now]=cnt++;
   vis[now]=1;
   int i;
   for(i=0; i<graph[now].size(); i++) {
      if(graph[now][i]==from) continue;
      if(vis[graph[now][i]]) {
         md[now]=min(md[now],d[graph[now][i]]);
         continue;
      }

md[now]=min(md[now],all_bridges(graph[now][i],
now));
      if(md[graph[now][i]]>d[now])

bridges[{min(now,graph[now][i]),max(now,graph[
now][i])}]=1;
   }
   return md[now];
}
```

fenwick_tree:

```
int n;long long bit[MX];
void add(int i, long long delta) {
for (; i < n; i |= (i+1)) bit[i] += delta;
// following is 1 based indexed implementation
// for (; i <= n; i += i&-i) bit[i] += delta;}
long long query(int i) {
return i < 0 ? 0LL : bit[i] + query((i&(i+1))-1);
// 1 based indexed implementation
// return i <= 0? 0LL : bit[i] + query(i - (i & -i)); }
int main() {scanf("%d %d", &n);
long long a[n];for (int i = 0; i < n; i++) {
scanf("%lld", &a[i]);add(i, a[i]);}}
```

Topological Sort

```
#define pb push_back
#define pf            printf
#define sf            scanf
#define sn(a)                  scanf("%lld",&a)
#define snn(a,b)               scanf("%lld
%lld",&a,&b)
#define snnn(a,b,c)     scanf("%lld %lld
%lld",&a,&b,&c)
#define M 105
vector<vector<int> > graph(M);
bool visited[M];
vector<int>topsort;
int node, edge;
stack<int>sk;
// topological sort
void dfs(int u) {
            visited[u] = true;
            for (unsigned i = 0; i < graph[u].size(); i++)
if (!visited[graph[u][i]]) dfs(graph[u][i]);
            sk.push(u); }
int main() {
            while (cin >> node >> edge, edge != 0 ||
node != 0) {
                        memset(visited, 0,
sizeof(visited));
                        graph.clear();
                        while (!sk.empty()) sk.pop();
                        while (edge--) {
                                    int u, v;
                                    cin >> u >> v;
                                    graph[u].pb(v); }
                        for (int i = 1; i <= node; i++) if
(!visited[i]) dfs(i);
                        int x = sk.size(), xx = 1;
                        while (!sk.empty()) {
                                    cout << sk.top();
                                    if (xx != x)cout << " ";
                                    xx++;
                                    sk.pop(); }
                        cout << endl; }
            return 0; }
```

MO's algo:

```
int ara[MAX+5];int L = 0, R = -1, sum = 0,
ans[MAX];
struct query{int id,l,r,mod;
query() {};} query[MAX+5];
void add(int x,int mod){
sum+=(ara[x]%mod);}
void rem(int x,int mod){sum-=(ara[x]%mod);}
int main()
{int n,q;cin >> n >> q;
for(int i=0; i<n; i++) cin>>ara[i];
for(int i=0; i<q; i++){
cin >> query[i].l >> query[i].r >> query[i].mod;
query[i].id = i;}
for(int i=0; i<q; i++){
while(query[i].r>R) add(++R,query[i].mod);
while(query[i].l>L) rem(L++,query[i].mod);
while(query[i].r<R) rem(R--,query[i].mod);
while(query[i].l<L) add(--L,query[i].mod);
cout<<sum<<endl;}return 0;}
```

floyed_cycle_finding:

```
int mu, lam;
inline int f(int x) {
return (x * (x+1)) % 11;}
void floyd(int x0) {
int tortoise = f(x0);
int hare = f(f(x0));
while(hare != tortoise) {
tortoise = f(tortoise);
hare = f(f(hare));}
mu = 0;tortoise = x0;
while(hare != tortoise) {
tortoise = f(tortoise);
hare = f(hare);++mu;}
lam = 1;hare = f(tortoise);
while(hare != tortoise) {
hare = f(hare);++lam;}
// updated lam, mu instead of returning the pair }
int main(){int x0;cin >> x0;
floyd(x0);cout << mu << " " << lam << "\n";
for(int i=0; i<20; ++i) {
cout << x0 << " ";x0 = f(x0);
} cout << endl;return 0;}
```

SSC:

```
/* adj[] is the main graph
trans[] strores tranpose graph
ind[u] will store the component number where u
belongs to */
const int maxn = 100000 + 7;    // 1e5
vector<int> adj[maxn], trans[maxn];
int ind[maxn], vis[maxn], idx = 0;
stack<int> st;
void dfs(int u) {vis[u] = 1;
for(int i=0; i<(int) adj[u].size(); ++i) {
int v = adj[u][i];
if(!vis[v]) dfs(v);}st.push(u);}
void dfs2(int u) {nd[u] = idx;
for(int i=0; i<(int) trans[u].size(); ++i) {
int v = trans[u][i];
if(!ind[v]) dfs2(v);}}
int scc(int n) {memset(vis, false, sizeof vis);
while(!st.empty()) st.pop();
for(int i = 1; i <= n; i++) {
if(!vis[i]) dfs(i);}
for(int u = 1; u <= n; u++) {
for(int i=0; i<(int) adj[u].size(); ++i) {
int v = adj[u][i];trans[v].push_back(u);}}
idx = 0;memset(ind, 0, sizeof ind);
while(!st.empty()) {
int u = st.top(); st.pop();
if(ind[u]) continue;
++idx;dfs2(u); }return idx;}
int main() {int t, tc=0;
scanf("%d", &t);
while(t--) {int n, m;
scanf("%d %d", &n, &m);
for(int i=1; i<=n; ++i) adj[i].clear(), trans[i].clear();
while(m--) {int u, v;scanf("%d %d", &u, &v);
adj[u].push_back(v);}
int res = scc(n);
printf("Case %d: %d\n", ++tc, res);
}return 0;}
```

MST:

```
struct edge {int u, v, w;
edge() { }
edge(int uu, int vv, int ww) { u=uu, v=vv, w=ww; }
bool operator < (const edge &p) const { return w <
p.w; }};

const int N = 100000 + 5;
int par[N], rep[N];
vector< edge > e;
inline int Find(int r) {
return par[r] == r ? r : par[r] = Find(par[r]);}
int mst(int n) {
if(n == 1) return 0;
sort(e.begin(), e.end());
for(int i=1; i<=n; ++i) par[i] = i, rep[i] = 1;
int res = 0, cnt = 0;
for(int i=0; i<(int) e.size(); ++i) {
int u = Find(e[i].u);int v = Find(e[i].v);
if(u != v) {
 if(rep[u] > rep[v]) par[v] = u, ++rep[u];
else par[u] = v, ++rep[v];res += e[i].w;
if(++cnt == n-1) return res;}}return -1;}
int main() {int n, m;cin >> n >> m;
while(m--) {int u, v, w;
cin >> u >> v >> w;
e.push_back(edge(u, v, w));}
int res = mst(n);cout << res;
return 0;}
```

Articulation point:

```
vector<int>gr[666666];
int
clr[666666],t,d[666666],low[666666],pre[666666],
cut[666666];
void dfs(int v){clr[v]=1;
t++;d[v]=t;
for(int i=0; i<gr[v].size(); i++)
{nt w=gr[v][i];if(clr[w]==0)
{pre[w]=v;dfs(w);
if(low[w]>=d[v]) cut[v]=1;
if(low[w]<low[v]) low[v]=low[w];}
if(d[w]<low[v]) low[v]=d[w];
}clr[v]=2;t++;}
int main()
{int a,b,V,E;
scanf("%d%d",&V,&E);
for(int i=0; i<E; i++){
scanf("%d%d",&a,&b);
gr[a].pb(b);gr[b].pb(a);}
memset(low,127,sizeof(low));
memset(d,127,sizeof(d));
for(int i=0; i<V; i++){
if(clr[i]==0){dfs(i);
if(gr[i].size()>1) cut[i]=1;
else cut[i]=0;}}
for(int i=0; i<V; i++)
{if(cut[i]!=0) printf("%d\n",i);
}return 0;}
```

DSU:

```
class UnionFind:
  def __init__(self, n):
    self._num_of_set = n
    self._set_size = [1] * n
    self._rank = [0] * n
    self._parent = [i for i in range(n)]
  def findSet(self, i):
    if self._parent[i] != i:
      self._parent[i] = self.findSet(self._parent[i])
    return self._parent[i]
  def isSameSet(self, i, j):
    return self.findSet(i) == self.findSet(j)
  def numOfDisjointSet(self):
    return self._num_of_set
  def sizeOfSet(self, i):
    return self._set_size[self.findSet(i)]
  def unionSet(self, i, j):
    if self.isSameSet(i, j):
      return
    pi, pj = self.findSet(i), self.findSet(j)
    if self._rank[pi] > self._rank[j]:
      self._parent[pj] = pi
      self._set_size[pi] += self._set_size[pj]
    else:
      self._parent[pi] = pj
      self._set_size[pj] += self._set_size[pi]
      if self._rank[pi] == self._rank[pj]:
        self._rank[pj] += 1
    self._num_of_set -= 1
```

Geo_Library:

```
//datatype definitions
typedef double T;typedef complex<T> pt;
#define x real()
#define y imag()
//output helpful for debugging
ostream& operator<< (ostream& os, pt p) { return
os << "(" << p.x << ", " << p.y << ")"; }
//translation, rotation and transformation
pt translate(pt v, pt p) { return v + p; } //translate p
by v
pt scale(pt c, T factor, pt p) { return c + (p - c) *
factor; }
pt rotate(pt p, T angle) { return p * polar(T(1),
angle); }
pt perp(pt p) { return { -p.y, p.x}; }
pt linearTransform(pt p, pt q, pt r, pt fp, pt fq) {
return fp + (r - p) * (fq - fp) / (q - p); }
//dot, cross and derivatives
T dot(pt v, pt w) { return (conj(v) * w).x; }
T cross(pt v, pt w) { return (conj(v) * w).y; }
tuple<T, T> dotcross(pt v, pt w) { pt p = conj(v) * w;
return {p.x, p.y}; }
bool isPerp(pt v, pt w) { return dot(v, w) == 0; }
double angle(pt v, pt w) {
return acos(clamp(dot(v, w) / abs(v) / abs(w), T(-
1), T(1)));}
T orient(pt a, pt b, pt c) { return cross(b - a, c - a); }
bool inAngle(pt a, pt b, pt c, pt p) {
assert(orient(a, b, c) != 0);
if (orient(a, b, c) < 0) swap(b, c);
return orient(a, b, p) >= 0 && orient(a, c, p) <= 0;}
double orientedAngle(pt a, pt b, pt c) {
return orient(a, b, c) >= 0 ? angle(b - a, c - a) : 2 *
M_PI - angle(b - a, c - a);}
bool isConvex(vector<pt> p) {bool hasPos = false,
hasNeg = false;
for (int i = 0, n = p.size(); i < n; i++) {
int o = orient(p[i], p[(i + 1) % n], p[(i + 2) % n]);
if (o > 0) hasPos = true;else if (o < 0) hasNeg = true;
}return !(hasNeg && hasPos);}
//Graham Scan
void convex_hull(vector<pt>& a) {
if (a.size() == 1)return;
sort(a.begin(), a.end(), [](pt a, pt b) {
return a.x < b.x || (a.x == b.x && a.y < b.y);
});pt p1 = a[0], p2 = a.back();
vector<pt> up, down;
up.push_back(p1);down.push_back(p1);
for (int i = 1; i < (int)a.size(); i++) {
if (i == a.size() - 1 || orient(p1, a[i], p2) < 0) {
while (up.size() >= 2 && !(orient(up[up.size() - 2],
up[up.size() - 1], a[i]) < 0))
up.pop_back();up.push_back(a[i]);}
if (i == a.size() - 1 || orient(p1, a[i], p2) > 0) {
while (down.size() >= 2 &&
!(orient(down[down.size() - 2], down[down.size() -
1], a[i]) > 0))
down.pop_back();down.push_back(a[i]);}}
a.clear();for (int i = 0; i < (int)up.size(); i++)
a.push_back(up[i]);for (int i = down.size() - 2; i > 0;
i--)
```

```
a.push_back(down[i]);}

//Polar Sort
bool half(pt p) {
assert(p.x != 0 && p.y != 0);
return p.y > 0 || (p.y == 0 && p.x < 0);}
void polarSort(vector<pt> &v) {
sort(v.begin(), v.end(), [](pt v, pt w) {
return make_tuple(half(v), 0) <
make_tuple(half(w), cross(v, w));
});}
struct line {pt v; T c;
line (pt v, T c) : v(v), c(c) {}
line (T a, T b, T c) : v( {b, -a}), c(c) {}
line (pt p, pt q) : v(q - p), c(cross(v, p)) {}
T side (pt p) { return cross(v, p) - c; }
double dist (pt p) { return abs(side(p)) / abs(v); }
double sqDist (pt p) { return side(p) * side(p) /
dot(v, v); }
line perpThrough (pt p) { return {p, p + perp(v)}; }
bool cmpProj (pt p, pt q) { return dot(v, p) < dot(v,
q); }
line translate (pt t) { return {v, c + cross(v, t)}; }
line shiftLeft(T dist) { return {v, c + dist * abs(v)}; }
pt proj(pt p) { return p - perp(v) * side(p) / dot(v,
v); }
pt refl(pt p) { return p - perp(v) * T(2) * side(p) /
dot(v, v); }
};
bool intersect(line l1, line l2, pt &out) {
T d = cross(l1.v, l2.v);if (d == 0) return false;
out = (l2.v * l1.c - l1.v * l2.c) / d;
return true;}
//bisector of angle
line bisector(line l1, line l2, bool interior) {
assert(cross(l1.v, l2.v) != 0);
T sign = interior ? T(1) : T(-1);
return { l2.v / abs(l2.v) + l1.v / abs(l1.v) * sign,
l2.c / abs(l2.v) + l1.c / abs(l1.v) * sign };}
// Segment functions
bool inDisk(pt a, pt b, pt p) { return dot(a - p, b - p)
<= 0; }
bool onSegment(pt a, pt b, pt p) {
return orient(a, b, p) == 0 && inDisk(a, b, p);}
bool properIntersect(pt a, pt b, pt c, pt d, pt &out)
{
T oa = orient(c, d, a);
T ob = orient(c, d, b);
T oc = orient(a, b, c);
T od = orient(a, b, d);
if (oa * ob < 0 && oc * od < 0) {
out = (a * ob - b * oa) / (ob - oa);
return true;}return false;}
//segment - point distance
double segPoint(pt a, pt b, pt p) {
if (a != b) {line l(a, b);
if (l.cmpProj(a, p) && l.cmpProj(p, b)) return
l.dist(p);
}return min(abs(p - a), abs(p - b));}
//segment - segment distance
double segSeg(pt a, pt b, pt c, pt d) {
pt dummy;
if (properIntersect(a, b, c, d, dummy)) return 0;
return min({segPoint(a, b, c), segPoint(a, b, d),
segPoint(c, d, a), segPoint(c, d, b)}) ;}
//Polygons
double areaTriangle(pt a, pt b, pt c) { return
abs(cross(b - a, c - a)) / 2.0;}
double areaPolygon(vector<pt> p) {
double area = 0.0;
for (int i = 0, n = p.size(); i < n; i++) area +=
cross(p[i], p[(i + 1) % n]);
return area;}
bool above(pt a, pt p) { return p.y >= a.y; }
bool crossesRay(pt a, pt p, pt q) {return (above(a,
q) - above(a, p) ) * orient(a, p, q) > 0;}
bool inPolygon(vector<pt> p, pt a, bool strict =
true) {
int numOfCrossings = 0;
for (int i = 0, n = p.size(); i < n; i++) {
if (onSegment(p[i], p[(i + 1) % n], a)) return !strict;
numOfCrossings += crossesRay(a, p[i], p[(i + 1) %
n]);
}return numOfCrossings & 1;}
// Winding number
double angleTravelled(pt a, pt p, pt q) {
double amplitude = angle(p - a, p - q);
return orient(a, p, q) > 0 ? amplitude : -
amplitude;}
int windingNumber (vector<pt> p, pt a) {
// undefined if a is on the polygon
double amplitude = 0;
for (int i = 0, n = p.size(); i < n; i++) amplitude +=
angleTravelled(a, p[i], p[(i + 1) % n]);
return round(amplitude / (2 * M_PI));}
//Circle
pt circumCenter(pt a, pt b, pt c) {
b -= a, c -= a; assert(cross(b, c) != 0); //no
cirumcircle if A, B, C aligned
return a + perp(b * abs(c * c) - c * abs(b * b)) /
cross(b, c) / T(2);}
int circleLine(pt o, double r, line l, pair<pt, pt>
&out) {
double h2 = r * r - l.sqDist(o);
if (h2 >= 0) {pt p = l.proj(o);
pt h = l.v * sqrt(h2) / abs(l.v);
out = {p - h, p + h};}
int sgn = (double(0) < h2) - (h2 < double(0));
return 1 + sgn;}
int circleCircle(pt o1, double r1, pt o2, double r2,
pair<pt, pt> &out) {
pt d = o2 - o1; double d2 = abs(d * d);
if (d2 == 0) {assert(r1 != r2); return 0;}
```

```
double pd = (d2 + r1 * r1 - r2 * r2) / 2;
double h2 = r1 * r1 - pd * pd / d2;
if (h2 >= 0) {
pt p = o1 + d * pd / d2, h = perp(d) * sqrt(h2 / d2);
out = {p - h, p + h};}
int sgn = (double(0) < h2) - (h2 < double(0));
return 1 + sgn;}
int tangents(pt o1, double r1, pt o2, double r2,
bool inner, vector<pair<pt, pt> > &out) {
if (inner) r2 = -r2;pt d = o2 - o1;
double dr = r1 - r2, d2 = abs(d * d), h2 = d2 - dr *
dr;
if (d2 == 0 || h2 < 0) {assert(h2 != 0); return 0;}
for (double sign : { -1, 1}) {
pt v = (d * dr + perp(d) * sqrt(h2) * sign) / d2;
out.push_back({o1 + v * r1, o2 + v * r2});
}return 1 + (h2 > 0);}
int main() {
pt a{1, 2}, b{ -3, 1}, e{0, 0};
cout << a + b << ", " << a*T(-1) << endl;
cout << abs(a) << ", " << arg(a) << endl;
cout << polar(2.0, -M_PI / 4) << endl;
auto [d, c] = dotcross(a, b);
cout << d << " " << c << endl;
cout << dot(a, b) << " " << cross(a, b) << endl;
cout << angle(a, b) << " " << isPerp(a, b) << endl;
cout << "Circum center: " << circumCenter(a, b, e)
<< endl;
return 0;}
```

Counting closest pair of Points (Convex hull)

```
int n;
struct Points {
	double x, y;
	Points() {}
	Points(double x, double y) : x(x), y(y) { }
	bool operator<(const Points &a) const {
return x < a.x; } };
bool comp1(const Points &a, const Points &b) {
return a.x < b.x; }
bool comp2(const Points &a, const Points &b) {
return a.y < b.y; }
void printPoint(Points a) { cout << a.x << " " << a.y
<< endl; }
Points P[10005];
typedef set<Points, bool(*)(const Points&, const
Points&)> setType;
typedef setType::iterator setIT;
setType s(&comp2);
double euclideanDistance(const Points &a, const
Points &b) {
// prnt((double)(a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-
b.y));
	return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) *
(a.y - b.y); }
map<double, map<double, int> > CNT;
int main() {
	while ((cin >> n) && n) {
		FOR(i, 0, n) cin >> P[i].x >> P[i].y;
		sort(P, P + n, comp1);
		FOR(i, 0, n) {
	// printPoint(P[i]);
			s.insert(P[i]);
			CNT[P[i].x][P[i].y]++; }
// To check repeated points :/
// for(auto it: s) printPoint(it);
		double ans = 10000;
		int idx = 0;
		FOR(j, 0, n) {
// cout<<"Point now: "; printPoint(P[j]);
			if (CNT[P[j].x][P[j].y] > 1)
ans = 0;
			Points it = P[j];
			while (it.x - P[idx].x >
ans) {
				s.erase(P[idx]);
				idx++; }
			Points low = Points(it.x,
it.y - ans);
			Points high = Points(it.x,
it.y + ans);
			setIT lowest =
s.lower_bound(low);
			if (lowest != s.end()) {
				setIT highest =
s.upper_bound(high);
				for (setIT now
= lowest; now != highest; now++) {

	double cur =
sqrt(euclideanDistance(*now, it));
// prnt(cur);
						if (cur
== 0) continue;
// cout<<"Here:"<<endl;
// printPoint(*now); printPoint(it); prnt(cur);
						if (cur
< ans) ans = cur; } }
			s.insert(it); }
// cout<<"Set now:"<<endl;
// for(auto I: s) printPoint(I);
		if (ans < 10000) cout <<
setprecision(4) << fixed << ans << endl;
		else prnt("INFINITY");
		s.clear();
		CNT.clear(); }
	return 0; }
```

Maximum Points to Enclose in a Circle of Given Radius with Angular Sweep

```
typedef pair<double,bool> pdb;
#define START 0
#define END 1
struct PT {
    double x, y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &p)  const {
return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p)  const { return
PT(x-p.x, y-p.y); }
    PT operator * (double c)     const { return
PT(x*c,   y*c  ); }
    PT operator / (double c)     const { return
PT(x/c,   y/c  ); } };
PT p[505];
double dist[505][505];
int n, m;
void calcDist() { FOR(i,0,n) FOR(j,i+1,n)
dist[i][j]=dist[j][i]=sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)
+(p[i].y-p[j].y)*(p[i].y-p[j].y)); }
// Returns maximum number of points enclosed
by a circle of radius 'radius'
// where the circle is pivoted on point 'point'
// 'point' is on the circumfurence of the circle
int intelInside(int point, double radius) {
    vector<pdb> ranges;
    FOR(j,0,n) {
        if(j==point ||
dist[j][point]>2*radius) continue;
        double a1=atan2(p[point].y-
p[j].y,p[point].x-p[j].x);
        double
a2=acos(dist[point][j]/(2*radius));
        ranges.pb({a1-a2,START});
        ranges.pb({a1+a2,END}); }
    sort(ALL(ranges));
    int cnt=1, ret=cnt;
    for(auto it: ranges) {
        if(it.second) cnt--;
        else cnt++;
        ret=max(ret,cnt); }
    return ret; }
// returns maximum amount of points enclosed by
the circle of radius r
// Complexity: O(n^2*log(n))
int go(double r) {
    int cnt=0;
    FOR(i,0,n) cnt=max(cnt,intelInside(i,r));
    return cnt; }
```

Point in Polygon Binary Search

```
int sideOf(const PT &s, const PT &e, const PT &p)  {
    ll a = cross(e-s,p-s);
    return (a > 0) - (a < 0); }
bool onSegment(const PT &s, const PT &e, const
PT &p)  {
    PT ds = p-s, de = p-e;
    return cross(ds,de) == 0 && dot(ds,de) <=
0; }
/*
Main routine
Description: Determine whether a point t lies
inside a given polygon (counter-clockwise order).
The polygon must be such that every point on the
circumference is visible from the first point in the
vector.
It returns 0 for points outside, 1 for points on the
circumference, and 2 for points inside.
*/
int insideHull2(const vector<PT> &H, int L, int R,
const PT &p) {
    int len = R - L;
    if (len == 2) {
        int sa = sideOf(H[0], H[L], p);
        int sb = sideOf(H[L], H[L+1], p);
        int sc = sideOf(H[L+1], H[0], p);
        if (sa < 0 || sb < 0 || sc < 0)
return 0;
        if (sb==0 || (sa==0 && L == 1) ||
(sc == 0 && R == (int)H.size())) return 1;
        return 2; }
    int mid = L + len / 2;
    if (sideOf(H[0], H[mid], p) >= 0) return
insideHull2(H, mid, R, p);
    return insideHull2(H, L, mid+1, p); }
int insideHull(const vector<PT> &hull, const PT &p)
{
    if ((int)hull.size() < 3) return
onSegment(hull[0], hull.back(), p);
    else return insideHull2(hull, 1,
(int)hull.size(), p); }
```

Rectangle Union

```
struct info {
    int x, ymin, ymax, type;
    info(){}
    info(int x, int ymin, int ymax, int type) :
x(x), ymin(ymin), ymax(ymax), type(type) { }
    bool operator < (const info &p) const {
return x<p.x; } };
vector<info> in;
int n, x, y, p, q, m;
vi take;
int Lazy[4*MAX], Tree[4*MAX];
void update(int node, int l, int r, int ymin, int ymax,
int val) {
    if(take[l]>ymax || take[r]<ymin) return;
    if(ymin<=take[l] && take[r]<=ymax) {
        Lazy[node]+=val;
        if(Lazy[node])
Tree[node]=take[r]-take[l];
        else
Tree[node]=Tree[lc]+Tree[rc];
        return; }
    if(l+1>=r) return;
    int mid=(l+r)/2;
```

```
        update(lc,l,mid,ymin,ymax,val);
        update(rc,mid,r,ymin,ymax,val);
        if(Lazy[node]) Tree[node]=take[r]-take[l];
        else Tree[node]=Tree[lc]+Tree[rc]; }
ll solve() {
        take.clear(); ms(Tree,0); ms(Lazy,0);
        take.pb(-1);
        FOR(i,0,in.size()) {
                take.pb(in[i].ymin);
                take.pb(in[i].ymax); }
        SORT(take);
        take.erase(unique(ALL(take)),take.end());
        m=take.size()-1;
        // VecPrnt(take);
        update(1,1,m,in[0].ymin,in[0].ymax,in[0].type);
        int prv=in[0].x; ll ret=0;
        FOR(i,1,in.size()) {
                ret+=(ll)(in[i].x-prv)*Tree[1];
                prv=in[i].x;

        update(1,1,m,in[i].ymin,in[i].ymax,in[i].type); }
        return ret; }
int main()
{
  int test, cases=1;
  scanf("%d", &test);
  while(test--) {
        scanf("%d", &n);
        in.clear();
        FOR(i,0,n) {
                scanf("%d%d%d%d", &x, &y, &p, &q);
                in.pb(info(x,y,q,1));
                in.pb(info(p,y,q,-1)); }
        SORT(in);
        ll ans=solve();
        printf("Case %d: %lld\n", cases++, ans); }
  return 0; }
```

Game theory

```
const int N = 300 + 7;int grundy[N];
int mex(vector<int> v) {
sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());
for(int i=0; i<(int) v.size(); ++i) {
if(v[i] != i) return i;
}return v.size();}
// returns the grundy value of the game
// with a strip of length n
int f(int n) {if(n == 0) return 0;
if(grundy[n] != -1) return grundy[n];
vector<int> vx;
for(int i=1; i<=n; ++i) {
int lf = max(0, i - 2);                    // left side strip length
int rt = max(0, n - i - 1);      // right side strip length
int cur = f(lf) xor f(rt);
vx.push_back(cur);}
grundy[n] = mex(vx);return grundy[n];}
int main() {
memset(grundy, -1, sizeof grundy);
/*
for(int i=1; i<=100; ++i) {
for(int len=1; len<=100; ++len) {
int j = i + len;bool flag = true;
for(int k=0; k<=100; ++k) {
if(f(i + k) != f(j + k)) {
flag = false;break;}}if(flag) {
cout << "Pattern starts from " << i << "\n";
cout << "Cycle length = " << len << "\n";
return 0;}}}
*/
int n;while(cin >> n) {
if(n < 52) {
if(f(n)) cout << "White\n";
else cout << "Black\n";
}else {
n = n - 52;n %= 34;
if(f(n + 52)) cout << "White\n";
else cout << "Black\n";
}}return 0;}
```

KMP code:

```
vector<int> prefix(const string& s) {
  int n = s.size();
  vector<int> pi(n);
  pi[0] = 0;
  int d = 0;
  for (int i = 1; i < n; ++i) {
    while (d > 0 and s[d] != s[i]) d = pi[d - 1];
    if (s[i] == s[d]) d++;
    pi[i] = d; }
  return pi; }
vector<int> kmp(const string& T, const string& P) {
  auto pi = prefix(P);
  vector<int> ocr;
  int d = 0;
  for (int i = 0; i < (int) T.size(); ++i) {
    while (d > 0 and P[d] != T[i]) d = pi[d - 1];
    if (T[i] == P[d]) d++;
    // current pi value (for text) is d
    if (d == (int) P.size()) {
      ocr.push_back(i - P.size() + 1);
      d = pi[d - 1]; } }
  return ocr; }
nbr_occer = kmp(str, Pattern);
```

Z_algorithm:

```
vector<int> z_function(string s) {
  int n = (int) s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1; }
    return z; }
```

Minimum Lexicographical Rotation:

```
int minimumExpression(string s) {
    s = s + s;
    int i = 0, j = 1, k = 0, len = s.size();
    while (i + k < len && j + k < len) {
        if (s[i + k] == s[j + k]) k++;
        else if (s[i + k] < s[j + k]) { j = max(j + k + 1, i +
1); k = 0; }
        else { i = max(i + k + 1, j + 1); k = 0; } }
    return min(i, j); }
```

double hashing:

```
const int BASE1 = 313;
const int MOD1 = 1e9 + 7;   // ensure this is a
prime
const int BASE2 = 1009;
const int MOD2 = 1e9 + 9;   // ensure this is a
prime
const int MAX = 1e6 + 7;
int hpsum[MAX], basep[MAX];
// precomputation O(|s|)
void init(const string& s) {
    hpsum[0] = 0, basep[0] = 1;
    for (int i = 0; i < (int) s.size(); ++i) {
        hpsum[i + 1] = (1LL * hpsum[i] * BASE1 + s[i])
% MOD1;
        basep[i + 1] = (1LL * basep[i] * BASE1) %
MOD1; } }

// query substring hash value O(1)
int h(int l, int r) {
    int sub = (hpsum[r + 1] - 1LL * hpsum[l] *
basep[r - l + 1]) % MOD1;
    if (sub < 0) sub += MOD1;
    return sub; }
// calling process
    string s;
    init(s);
    h(l, r)
```

Dynamic Hashing

```
const int B = 1249;
const int MOD = 1e9 + 7;
const int N = 1e6 + 7;
int base_pwr[N];                // base powers
struct Seg {int tr[4 * N];
Seg() { }
void build(int at, int l, int r, const string& s) {
if(l == r) {tr[at] = s[l] % MOD;
return ;}
int lc = (at << 1), rc = ((at << 1) ^ 1), mid = (l + r) / 2;
build(lc, l, mid, s);
build(rc, mid + 1, r, s);
tr[at] = (1ll * tr[lc] * base_pwr[r-mid] + tr[rc]) %
MOD;}
void update(int at, int l, int r, const int p, const int
v) {
if(l == r) {tr[at] = v % MOD;return ;}
int lc = (at << 1), rc = ((at << 1) ^ 1), mid = (l + r) / 2;
if(p <= mid) update(lc, l, mid, p, v);
else update(rc, mid + 1, r, p, v);
tr[at] = (1ll * tr[lc] * base_pwr[r-mid] + tr[rc]) %
MOD;}
int seg_seg_ins(int a, int b, int p, int q) {
if(a > q or b < p) return 0;
return min(b, q) - max(a, p) + 1;}
int query(int at, int l, int r, const int lo, const int hi)
{
if(l > hi or r < lo) return 0;
if(l >= lo and r <= hi) return tr[at];
int lc = (at << 1), rc = ((at << 1) ^ 1), mid = (l + r) / 2;
int q1 = query(lc, l, mid, lo, hi);
int q2 = query(rc, mid + 1, r, lo, hi);
int ret = (1ll * q1 * base_pwr[seg_seg_ins(mid + 1,
r, lo, hi)] + q2) % MOD;
return ret;}};
void precal() {base_pwr[0] = 1;
for(int i=1; i<N; ++i) base_pwr[i] = (1ll *
base_pwr[i-1] * B) % MOD;}

int main() {precal();string s;
cin >> s;Seg seg;
seg.build(1, 0, s.size() - 1, s);
int l, r;while(cin >> l >> r) {
cerr << "h: " << seg.query(1, 0, s.size() - 1, l, r) <<
"\n";
}return 0;}
```

String matching using hashing

```
const int BASE = 313;
const int MOD = 1e9 + 7;   // ensure this is a prime
const int MAX = 1e6 + 7;
int hpsum[MAX], basep[MAX];
// precomputation O(|s|)
void init(const string& s) {hpsum[0] = 0, basep[0] =
1;
for(int i=0; i<(int) s.size(); ++i) {
hpsum[i + 1] = (1LL * hpsum[i] * BASE + s[i]) %
MOD;
basep[i + 1] = (1LL * basep[i] * BASE) % MOD;}}
// query substring hash value O(1)
int h(int l, int r) {
int sub = (hpsum[r + 1] - 1LL * hpsum[l] * basep[r -
l + 1]) % MOD;
if(sub < 0) sub += MOD;return sub;}
vector<int> match(const string& T, const string&
P) {init(P);
int hp = h(0, P.size() - 1);init(T);vector<int> ocr;
for(int i=0; i + (int) P.size() <= (int) T.size(); ++i) {
```

```
if(h(i, i + P.size() - 1) == hp)
{ocr.push_back(i);}}return ocr;}
int main() {string txt, pat;
cin >> txt >> pat;auto ocr = match(txt, pat);
for(int p : ocr) cout << "Occured at index: " << p << "\n";return 0;}
```

Next_permutation:

```
void permute(string str){
sort(str.begin(), str.end());
do {cout << str << endl;
} while (next_permutation(str.begin(), str.end()));}
```

Ternary Search

```
typedef double Tf;
const Tf EPS = 1e-12;
struct Pt {Tf x, y;
friend istream& operator >> (istream& is, Pt& p) {
return is >> p.x >> p.y; }};

Tf dist(Pt a, Pt b) {
return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));}

Tf f(Tf t, const Pt a, const Pt b, const Pt c, const Pt d) {
Pt lf;lf.x = a.x + t * (b.x - a.x);
lf.y = a.y + t * (b.y - a.y);Pt rt;
rt.x = c.x + t * (d.x - c.x);
rt.y = c.y + t * (d.y - c.y);
return dist(lf, rt);}

int main() {int t, tc = 0;
cin >> t;while(t--) {
Pt a, b, c, d;
cin >> a >> b >> c >> d;
Tf l = 0, r = 1;while(r - l > EPS) {
Tf m1 = l + (r - l) / 3;
Tf m2 = r - (r - l) / 3;
if(f(m1, a, b, c, d) > f(m2, a, b, c, d)) {
l = m1;}else {r = m2;}}
Tf res = f(l, a, b, c, d);
cout << "Case " << ++tc << ": " << fixed << setprecision(10) << res << "\n";
}return 0;}
```

Binary indexed tree

```
ll Tree[MAX];
// This is equivalent to calculating lower_bound on prefix sums array
// LOGN = log(N)
int bit_search(int v) {
        int sum = 0;
        int pos = 0;
        for(int i=LOGN; i>=0; i--) {
                if(pos + (1 << i) < N and sum + Tree[pos + (1 << i)] < v) {
                        sum += Tree[pos + (1 << i)];
                        pos += (1 << i); } }
        // +1 because 'pos' will have position of largest value less than 'v'
        return pos + 1; }
void update(int idx, ll x) {
        // Let, n is the number of elements and our queries are
        // of the form query(n)-query(l-1), i.e range queries
        // Then, we should never put N or MAX in place of n here.
        while(idx<=n) {
                Tree[idx]+=x;
                idx+=(idx&-idx); } }
ll query(int idx) {
        ll sum=0;
        while(idx>0) {
                sum+=Tree[idx];
                idx-=(idx&-idx); }
        return sum; }

int main() {
        // For point update range query:
        // Point update: update(x,val);
        // Range query (a,b): query(b)-query(a-1);
        // For range update point query:
        // Range update (a,b): update(a,v); update(b+1,-v);
        // Point query: query(x);
        // Let's just consider only one update: Add v to [a, b] while the rest elements of the array is 0.
        // Now, consider sum(0, x) for all possible x, again three situation can arise:
        // 1. 0 ≤ x < a : which results in 0
        // 2. a ≤ x ≤ b : we get v * (x - (a-1))
        // 3. b < x < n : we get v * (b - (a-1))
        // This suggests that, if we can find v*x for any index x, then we can get the sum(0, x) by subtracting T from it, where:
        // 1. 0 ≤ x < : Sum should be 0, thus, T = 0
        // 2. a ≤ x ≤ : Sum should be v*x-v*(a-1), thus, T = v*(a-1)
        // 3. b < x < n : Sum should be 0, thus, T = -v*b + v*(a-1)
        // As, we can see, knowing T solves our problem, we can use another BIT to store this additive amount from which we can get:
        // 0 for x < a, v*(a-1) for x in [a..b], -v*b+v(a-1) for x > b.
        // Now we have two BITs.
        // To add v in range [a, b]: Update(a, v), Update(b+1, -v) in the first BIT and Update(a, v*(a-1)) and Update(b+1, -v*b) on the second BIT.
        // To get sum in range [0, x]: you simply do Query_BIT1(x)*x - Query_BIT2(x);
        // Now you know how to find range sum for [a, b]. Just find sum(b) - sum(a-1) using the formula stated above.
    return 0; }
```

<u>**Tips and tricks**</u>

# Number of ways two knights can be placed such that they don't attack in n*n chess board. number of ways to place with accack and don't is= 4*(n-1)*(n-2) number of ways to place can accack each others is=((n*n)*((n*n)-1))/2 so number of ways to place con't accack each others is = (((n*n)*((n*n)-1))/2)-(4*(n-1)*(n-2))

#Find last 3(x) digit of N^p
-> use N^p%10^k
-> N^p%1000
-> here k digit=k zero after one(1)

# number of digit of n
dig=log10(n)+1;
->number of digit of 2^n=nlog2+1
-> number of digit of N!
dig+=log10(N(1-n))
-> and in different base
dig+=log10(N(1-n))/log10(base)

#sum of divsior logic
(r^(n+1)-1)/r-1;here r=2 and n=3 when 2^3=8.

//Number of subsequence

```
Formula 1 :   2^n
Formula 2 :   C(n,0)+C(n,1)+......+C(n,n)
Formula 3 :   f(n)=2*f(n-1)  //take s[n] or don't take
```

//Number of distinct subsequence

```
Formula 1 :   f(n) = 2*f(n-1)-f(m)  //here, m=index of previous occurrence of s[n]. m=0 if s[n] not found previously
```

//Longest Common Subsequence

```
Formula 1 :   f(i,j)=0, if(i==0 || j==0)
        f(i,j)=1+f(i-1,j-1), if(X[i]==Y[j])
        f(i,j)=max(f(i,j-1),f(i-1,j)), if(X[i]!=Y[j])
```

Space Optimization trick: [2][N]
Time Optimization trick: use LIS in O(nlogn) if at most one string contains repetitions of characters

//**Longest Repeating Subsequence**
//**Longest Subsequence that occurs at least twice in a string without overlapping**

```
Formula 1 :   f(i,j)=0, if(i==0 || j==0)
        f(i,j)=1+f(i-1,j-1), if(s[i]==s[j] and i!=j)
        f(i,j)=max(f(i,j-1),f(i-1,j)), if(s[i]!=s[j] or i==j)
```

//**Edit Distance**
//**Operations: Insert,Delete,Replace**

```
Formula 1 :   f(i,j)=j /* insert Y[1...j] in X */, if(i==0)
        f(i,j)=i /* insert X[1...i] in Y */, if(j==0)
        f(i,j)=f(i-1,j-1), if(X[i]==Y[j])
        f(i,j)=1+min(f(i,j-1),f(i-1,j),f(i-1,j-1)), if(X[i]!=Y[j])
```

//**Edit Distance**
//**Operations: Insert,Delete**

```
Formula 1 :   f(i,j)=Size(x)+Size(Y)-LCS(X,Y)
```

//**Number of palindromic subsequence**

```
Formula 1 :   f(i,j)=1, if(i==j)
        f(i,j)=f(i+1,j)+f(i,j-1)-f(i+1,j-1)+ [f(i+1,j-1)+1], if(s[i]==s[j])
        f(i,j)=f(i+1,j)+f(i,j-1)-f(i+1,j-1), if(s[i]!=s[j])
```

//**Number of distinct palindromic subsequence**

```
Formula 1 :   f(i,j,x)=0, if(i==j and s[i]!=x)
        f(i,j,x)=1, if(i==j and s[i]=x)
        f(i,j,x)=f(i+1,j,x)+f(i,j-1,x)-f(i+1,j-1,x), if(s[i]!=x or s[j]!=x)
        f(i,j,x)=2 + sum of all y in a-z (f(i+1,j-1,y)), if(s[i]==x and s[j]==x)
        //2 added for xx and x[longest sequence]x
```

//**Longest Palindromic Subsequence**

```
Formula 1 :   f(i,j)=0, if(i>j)
        f(i,j)=1, if(i==j)
        f(i,j)=f(i+1,j-1)+2, if(s[i]==s[j])
        f(i,j)=max(f(i+1,j),f(i,j-1)), if(s[i]!=s[j])
```

//**Longest Common Substring**

```
Formula 1 :   f(i,j)=0, if(i==0 || j==0)
        f(i,j)=1+f(i-1,j-1), if(X[i]==Y[j])
        f(i,j)=0, if(X[i]!=Y[j])
        ans=max of all f(i,j)
```

//**Longest Palnidromic Substring**

```
Formula 1 :   isPal(i,j)=0, if(i>j)
        isPal(i,j)=1, if(i==j)
        isPal(i,j)=2+isPal(i+1,j-1), if(i<j and s[i]==s[j])
        isPal(i,j)=0, if(i<j and s[i]!=s[j])
```