acm

ICPC International Collegiate Programming Contest

NULL Coders

Notebook de Algoritmos[1]

[1] **NOTAS PARA EDIÇÃO:** Inserir os códigos dentro de uma tabela 1x1 através do complemento “Code Books” com o tema “googlecode” e sem background.

ÍNDICE

1. TEMPLATE

```
#include <bits/stdc++.h>
using namespace std;
#define MAXN 10005 // alterar para diferentes problemas
#define INF 0x3F3F3F3F
#define pb push_back
#define mb make_pair
typedef vector<int> vi;
typedef pair <int, int> pii;
typedef vector<pii> vii;

int main() {
    return 0;
}
```

2. GRAFOS

2.1. DFS

```
vector<int> g[MAXN];
bool vis[MAXN];
void dfs(int s) {
   vis[s] = true;
   for (int i = 0; i < g[s].size(); i++) {
       int y = g[s][i];
       if (!vis[y])
           dfs(y);
   }
}
```

2.2. BFS

```
vector<int> g[MAXN];
int dist[MAXN];
void bfs(int s) {
    memset(dist, INF, sizeof(dist));
    dist[s] = 0;
    queue<int> q;
    q.push(s);

    while (!q.empty())  {
        int x = q.front();
        int d = dist[x];
        q.pop();

        for (int i = 0; i < g[x].size(); i++) {
            int y = g[x][i];
            if (d + 1 < dist[y]) {
                dist[y] = d + 1;
                q.push(y);
            }
        }
    }
}
```

2.3.1. Dijkstra com Matriz de Adjacência - O(V²)

```
int graph[MAXN][MAXN]; // inicializar com -1
int dist[MAXN];
int pred[MAXN];
bool vis[MAXN];
int n;

void dijkstra(int p) {
    int i, v, c;
    for (i = 0; i < MAXN; i++) dist[i] = INF;
    memset(vis, 0, sizeof vis);
    memset(pred, -1, sizeof pred);
    dist[p] = 0;
    v = p;

    while (!vis[v]) {
        vis[v] = true;
        for (i = 0; i < n; i++)
            if (graph[v][i] != INF) {
                c = graph[v][i];
                if (dist[i] > dist[v] + c) {
                    dist[i] = dist[v] + c;
                    pred[i] = v;
                }
            }

        v = 0;
        c = INF;

        for (i = 1; i < n; i++)
            if (!vis[i] && c > dist[i]) {
                c = dist[i];
                v = i;
            }
    }
}
```

2.3.2. Dijkstra com Lista de Adjacência - O((V+E)logV)

```
vector < pair <int, int > > g[MAXN];
int dist[MAXN];

void dijkstra(int s) {
      memset(dist, INF, sizeof dist);
      priority_queue < pair < int, int > > pq;
```

```
        dist[s] = 0;
        pq.push(make_pair(0, s));

        while (!pq.empty()) {
                int x = pq.top().second;
                int d = -pq.top().first;
                pq.pop();

                if (d > dist[x]) continue;

                for (int i = 0; i < g[x].size(); i++) {
                        int y = g[x][i].first;
                        int nd = g[x][i].second + d;

                        if (dist[y] > nd) {
                                dist[y] = nd;
                                pq.push(make_pair(-nd, y));
                        }
                }
        }
}
```

2.4. Floyd Warshall - $O(N^3)$

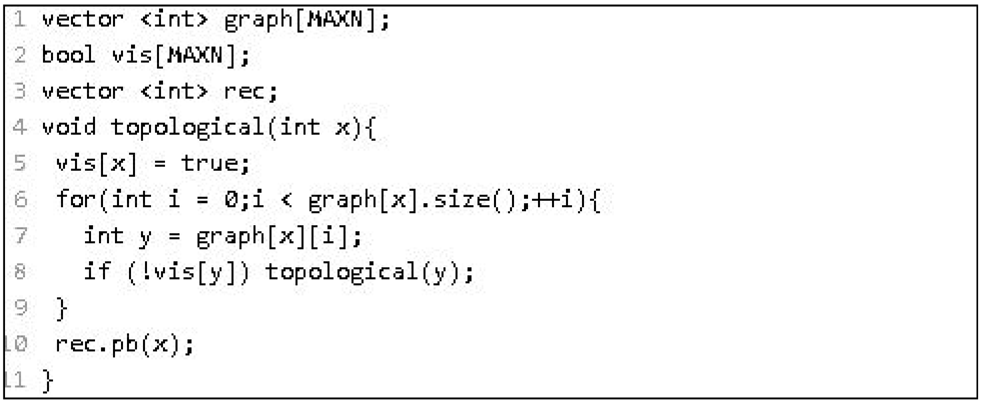
```
int n;
int dist[MAXN][MAXN];
void floyd(){
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
```

2.5. Kruskal Minimal Spanning Tree - $O(E + \log V)$

```
struct node{
    int x,y,w;
    node(){};
    node (int _x, int _y, int _w){
        x=_x;y=_y;w=_w;
    }
    bool operator < (const node sx) const{
        return w < sx.w;
    }
};

int n;
vector<node> graph;
int root [MAXN];

void iniset(){
    for (int i = 0 ; i < MAXN;i++)
```

```
        root[i]=i;
}

int findset(int x){
    if (root[x] ==x)return x;
    return root[x]=findset(root[x]);
}

long long int kruskal(){
    iniset();
    sort(graph.begin(),graph.end());
    long long int mst=0;
    for (int i = 0 ; i < graph.size();i++){
        int x=graph[i].x;
        int y=graph[i].y;
        int w=graph[i].w;
        int fx=findset(x);
        int fy=findset(y);

        if (fx!=fy){
            root[fx]=fy;
            mst+=w;
        }
    }
    return mst;
}
// pra inserir os nodes: graph.push_back(node(x, y, w));
```

2.6. Topological Sort - O(V+E)



```
vector <int> graph[MAXN];
bool vis[MAXN];
vector <int> rec;
void topological(int x){
 vis[x] = true;
 for(int i = 0;i < graph[x].size();++i){
   int y = graph[x][i];
   if (!vis[y]) topological(y);
 }
 rec.pb(x);
}
```

2.7. Tarjan’s Algorithm - O(V+E)

```
vector <int> g[MAXN];
int n, m;
int pred[MAXN], low[MAXN], id[MAXN];
// id[X] = strongly connected component index of X
stack <int> stk;
int preordercounter;
int numSCCs;
```

```
void dfs(int u){
 pred[u] = low[u] = preordercounter++;
 stk.push(u);

 for(int i=0;i<g[u].size();i++){
   int y = g[u][i];
   if(pred[y]==-1) dfs(y);

   low[u] = min(low[u], low[y]);
 }

 if(pred[u]==low[u]){
   while(1){
     int y = stk.top(); stk.pop();
     id[y] = numSCCs;
     low[y] = n+1;
     if(u==y) break;
   }
   numSCCs++;
 }
}

bool tarjan(){
 memset(pred, -1, sizeof pred);
 numSCCs = preordercounter = 0;
 for(int i=1;i<=n;i++){
   if(pred[i]==-1) dfs(i);
 }
 return (numSCCs==1); //se o grafo é fortemente conexo
}
```

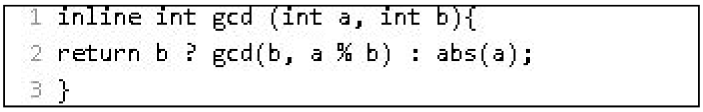
2.8. Bridges Detection - O(V+E)

```
int n, p=1, ans;
vector <int> g[MAXN];
int disc[MAXN], low[MAXN], parent[MAXN];

void dfs(int x){
 disc[x] = low[x] = p++;
 for(int i=0;i<g[x].size();i++){
   int y = g[x][i];
```

```
    if(disc[y]==-1){
      parent[y]=x;
      dfs(y);
      low[x] = min(low[x], low[y]);
      if(low[y]>disc[x]) ans++;
    }
    else if (y != parent[x])low[x] = min(low[x], disc[y]);
  }
}

int bridges(){
  for(int i=0;i<n;i++){
    if(disc[i]==-1) dfs(i);
  }
  // ans contem a resposta
}
```

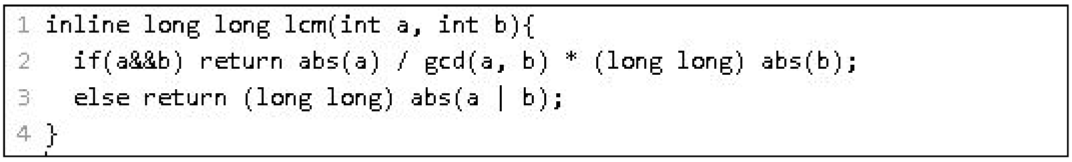
3.Matemática

3.1. GCD



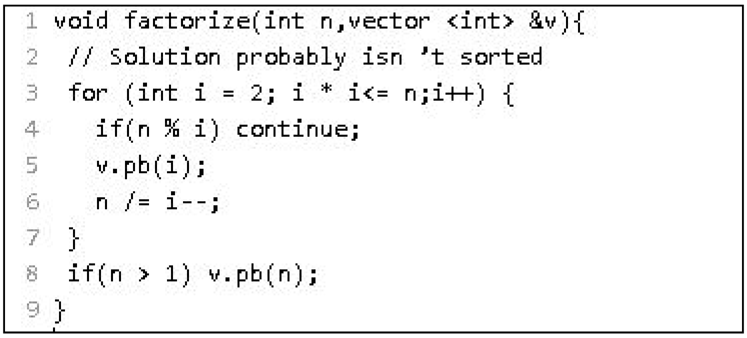
```
inline int gcd (int a, int b){
return b ? gcd(b, a % b) : abs(a);
}
```

3.2. LCM



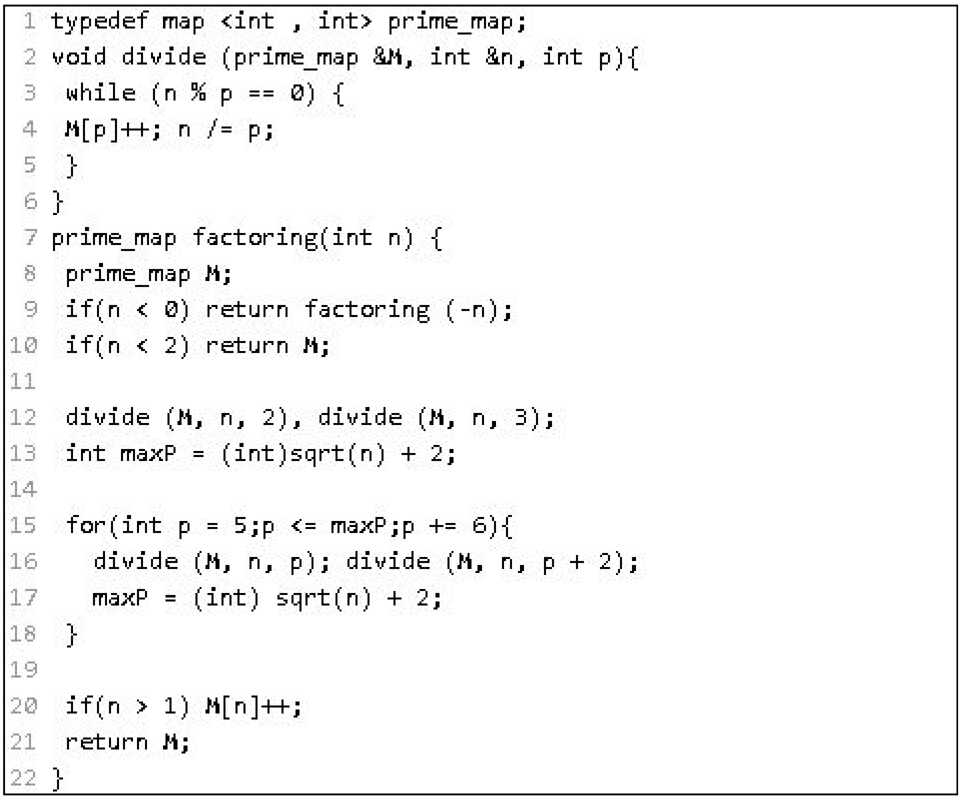
```
inline long long lcm(int a, int b){
  if(a&&b) return abs(a) / gcd(a, b) * (long long) abs(b);
  else return (long long) abs(a | b);
}
```

3.3. Factoring



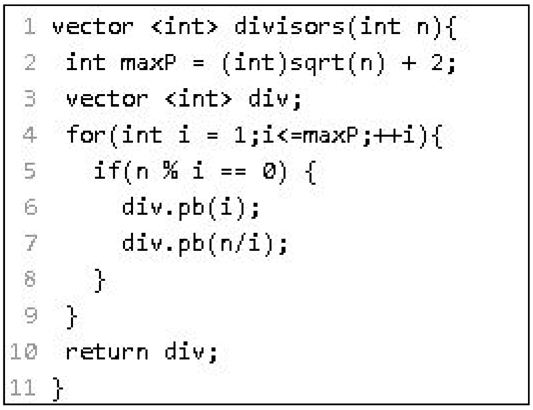
```
void factorize(int n,vector <int> &v){
 // Solution probably isn 't sorted
 for (int i = 2; i * i<= n;i++) {
   if(n % i) continue;
   v.pb(i);
   n /= i--;
 }
 if(n > 1) v.pb(n);
}
```

3.4. Factoring (Grouped Factors)



```
typedef map <int , int> prime_map;
void divide (prime_map &M, int &n, int p){
 while (n % p == 0) {
 M[p]++; n /= p;
 }
}
prime_map factoring(int n) {
 prime_map M;
 if(n < 0) return factoring (-n);
 if(n < 2) return M;

 divide (M, n, 2), divide (M, n, 3);
 int maxP = (int)sqrt(n) + 2;

 for(int p = 5;p <= maxP;p += 6){
   divide (M, n, p); divide (M, n, p + 2);
   maxP = (int) sqrt(n) + 2;
 }

 if(n > 1) M[n]++;
 return M;
}
```

3.5. Divisors



```
vector <int> divisors(int n){
 int maxP = (int)sqrt(n) + 2;
 vector <int> div;
 for(int i = 1;i<=maxP;++i){
   if(n % i == 0) {
     div.pb(i);
     div.pb(n/i);
   }
 }
 return div;
}
```

Fibonnaci - O(Log(n))

```
unsigned long long int f[MAXN];
unsigned long long int fib(int n) {
    if (n == 0) return 0;
    if (n == 1 || n == 2) return (f[n] = 1);
    if (f[n]) return f[n];
    unsigned long long int k = (n & 1) ? (n + 1) / 2 : n / 2;
    f[n] = (n & 1) ? (fib(k) * fib(k) + fib(k - 1) * fib(k - 1)) : (2 * fib(k - 1)
+ fib(k)) * fib(k);
    return f[n];
}
```

Counting Sort - O(N)

```
void countingSort(long long int *arr, long long int n, long long int minr, long
long int maxr) {
    long long int j, z = 0, freq[maxr - minr + 1];
    memset(freq, 0, sizeof(freq));
    for (j = 0; j < n; j++)
        freq[arr[j] - minr]++;
    for (j = 0; j <= maxr - minr; j++)
        while (freq[j]-- > 0)
            arr[z++] = j + minr;
}
```

Maior soma sequencial em um conjunto (Kodane) - O(N)

```
int kodane(vector<int> v){
   int n = v.size();
   int sum=0, ans =0;
   for (int i = 0 ; i < n;i++){
       sum += v[i];
       ans = max(ans, sum);
       if (sum <0) sum=0;
   }
   return ans;
}
```

Fenwick Tree - O (Log(N))

```
vector <int> nums;
void update(int i, int v){
       while (i < nums.size()){
             nums[i] +=v;
             i+= i &-i;
       }
}

int sum(int i){
       int ans=0;
       while (i > 0){
             ans+=nums[i];
             i-= i &-i;
       }
       return ans;
}

int sum_range(int i, int j){
       return sum(j)-sum(i-1);
}
```

Convex Hull - (Edgar)

```
typedef struct Point{
  int x, y;
  Point(){};
  Point(int _x, int _y){
      x = _x; y = _y;
  }
  Point operator - (Point o){
      return Point(x-o.x, y-o.y);
  }
  int operator % (Point o){
      return x*o.y - y*o.x;
  }
  bool operator < (Point o)const{
      if(x < o.x) return true;
      else if(x == o.x && y < o.y) return true;
      else return false;
  }
}Point;
vector <Point> pts;
int tam;

double dist(Point a, Point b){
  double x = (a.x-b.x)*(a.x-b.x);
  double y = (a.y-b.y)*(a.y-b.y);
  return sqrt(x+y);
}
double cima(){
  double ans=0;
  vector <Point> ch;
  for(int i=0;i<tam;i++){
      while(ch.size()>=2 && (ch[ch.size()-1]-ch[ch.size()-2]) %
(pts[i]-ch[ch.size()-2]) >= 0){
      ch.pop_back();
      }
      ch.pb(pts[i]);
  }
  for(int i=1;i<ch.size();i++){
      ans += dist(ch[i-1], ch[i]);
  }
  return ans;
}
double baixo(){
  double ans=0;
  vector <Point> ch;
  for(int i=0;i<tam;i++){
      while(ch.size()>=2 && (ch[ch.size()-1]-ch[ch.size()-2]) %
(pts[i]-ch[ch.size()-2]) <= 0){
      ch.pop_back();
      }
      ch.pb(pts[i]);
  }
  for(int i=1;i<ch.size();i++){
      ans += dist(ch[i-1], ch[i]);
  }
  return ans;
}
```

```
//Dar sort(pts.begin(), pts.end()); na main
```

Convex Hull - (Ricardo)

```
struct Point {
        int x, y;
        int operator%(Point o) {
                return (x - o.x) * (x - o.x) + (y - o.y) * (y - o.y);
        }
};
vector<Point> points;

int orientation(Point p, Point q, Point r) {
        int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
        if (val == 0) return 0;
        return (val > 0) ? 1 : 2;
}

int compare(const void *vp1, const void *vp2) {
        Point *p1 = (Point *)vp1, *p2 = (Point *)vp2;
        int o = orientation(points[0], *p1, *p2);

        if (o == 0) return (points[0] % *p2 >= points[0] % *p1) ? -1 : 1;
        return (o == 2) ? -1 : 1;
}

vector<Point> convexHull()
{
        int n = points.size(), min = 0, m=1;
        vector<Point> ans;

        for (int i = 1; i < n; i++)
                if (points[i].y < points[min].y || (points[i].y == points[min].y &&
points[i].x < points[min].x))
                        min = i;
        swap(points[0], points[min]);
        qsort(&points[1], n - 1, sizeof(Point), compare);

        for (int i = 1; i < n; i++) {
                while (i < n - 1 && orientation(points[0], points[i], points[i + 1])
== 0)
                        i++;
                points[m] = points[i];
                m++;
        }

        if (m < 3) return ans;

        for (int i = 0; i < m; i++) {
                while (i > 2 && orientation(ans[ans.size() - 2], ans.back(),
points[i]) != 2)
                        ans.pop_back();
```

```
             ans.push_back(points[i]);
       }
       return ans;
}

double perimeter(vector<Point> points) {
       double ans = 0;
       for (int i = 0; i < points.size() - 1; i++)
             ans += sqrt(points[i] % points[i + 1]);
       return ans += sqrt(points[points.size() - 1] % points[0]);
}
```

LCA

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define mp make_pair
#define INF 0x3F3F3F3F
#define MAXN 20002
typedef long long int ll;

typedef struct edge{
  int a, b, w;
  edge(){};
  edge(int _a, int _b, int _w){
       a = _a; b = _b; w = _w;
  }
  bool operator < (const edge ex) const{
       return ex.w < w;
  }
}Edge;
vector <Edge> g;
int dist[MAXN][23], root[MAXN], n, m;
int memo[MAXN][23], h[MAXN];
vector <pair<int, int> > graph[MAXN];

void dfs(int x){
  for (int i = 0;i < graph[x].size();i++){
       int y = graph[x][i].first;
       if (y == memo[x][0]) continue;

       h[y] = h[x] + 1;
       dist[y][0] = graph[x][i].second;
       memo[y][0] = x;
       //printf("h[%d] = %d\n", y, h[y]);
       dfs(y);
  }
}
void build_lca(){
  for(int j=1;j<23;j++){
      for(int i=1;i<=n;i++){
      memo[i][j] = memo[memo[i][j-1]][j-1];
      dist[i][j] = min(dist[i][j-1], dist[memo[i][j-1]][j-1]); //Alterar
      //printf("Pulo do %d de %d: %d - Peso: %d\n", i, j, memo[i][j], dist[i][j]);
      }
  }
}

int lca(int x, int y){
  if(h[y] > h[x]) swap(x, y);
  int d = h[x]-h[y];
  int ans = INF;
  //printf("x:%d y:%d\n", x, y);
  for(int i=0;i<23;i++){
      if(d & (1 << i)){
       //printf("dist[%d][%d]=%d\n",x, i, dist[x][i]);
       ans = min(ans, dist[x][i]);
       x = memo[x][i];
      }
  }
  if(x==y) return ans;
  for(int i=22;i>=0;i--){
      if(memo[x][i]!=memo[y][i]){
      ans = min(ans, min(dist[y][i], dist[x][i]));
      x = memo[x][i];
      y = memo[y][i];
      }
  }
  return min(ans, min(dist[x][0],dist[y][0]));
}
```

Number to string / String to number

```
#include <bits/stdc++.h>
using namespace std;
#define MAXN 3001
#define pb push_back

int main(){
 double i;
 scanf("%lf", &i);
  //Number to string
 ostringstream str1;
 str1 << i;
 string b = str1.str();
 cout << b << endl;
  //String to number
 stringstream str2(b);
 str2 >> i;
 printf("%lf", i);
}
```

Segtree (Edgar)

```
int st[4*MAXN];
void build(int id, int l, int r){
    if(l==r){
        st[id] = a[l-1]; return;
    }
    int mid = (l+r)>>1;
    int nxt = id<<1;
    build(nxt, l, mid);
    build(nxt+1, mid+1, r);
    st[id] = st[nxt] + st[nxt+1];
}
void update(int id, int l, int r, int pos, int val){
    if(pos < l || pos > r) return;
    if(l==r){
        st[id] = val; return;
    }
    int mid = (l+r)>>1;
    int nxt = id<<1;
    update(nxt, l, mid, pos, val);
    update(nxt+1, mid+1, r, pos, val);
    st[id] = st[nxt] + st[nxt+1];
}
int query(int id, int l, int r, int ll, int rr){
    if(rr < l || ll > r) return 0;
    if(l >= ll && r <= rr) return st[id];
    int mid = (l+r)>>1;
    int nxt = id<<1;
    return query(nxt, l, mid, ll, rr) + query(nxt+1, mid+1, r, ll, rr);
}
```

SegTree (Ricardo)

```
int t[MAXN * 2], n;

void build(){
    for (int i = n - 1; i > 0; i--) t[i] = t[i << 1] + t[i << 1 | 1];
}

void update(int i, int val){
    for (t[i += n] = val; i > 1; i >>= 1) t[i >> 1] = t[i] + t[i ^ 1];
}
```

```
int query(int l, int r){
    int ans = 0;
    for (l += n, r += n; l < r; l >>=1, r >>=1){
        if (l & 1) ans += t[l++];
        if (r & 1) ans += t[--r];
    }
    return ans;
}
```

Segtree com Lazy Propagation

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

//Gustavo Guerra, Federal University of Itajubá
// ---------------------------------------------------
//This can be adapted to run with the
//iterative (non-recursive) version of a segment tree,
//but I'm too lazy
// ---------------------------------------------------
//Segment Tree
//With Lazy Propagation

const int MAXN = 101010;

int st[4*MAXN];
int v[MAXN];
int lazy[4*MAXN];

void build(int at, int l, int r){

    if(l == r){
       st[at] = v[l];
       return;
    }

    int mid = (l + r)>>1;

    build((at<<1), l, mid);
    build((at<<1) + 1, mid + 1, r);

    st[at] = st[at<<1] + st[(at<<1) + 1];
}

//node "at" comprehends interval [l , r]
//updating interval [lu, ru] with val "x"

void update(int at, int l, int r, int lu, int ru, int x){

    if(lazy[at]){
        st[at] += (r - l + 1) * lazy[at]; //updates node with lazy array value

        if(l != r){ // spread lazyness to children
              lazy[at<<1] += lazy[at];
              lazy[(at<<1) + 1] += lazy[at];
        }

        lazy[at] = 0;
    }

    if(r < lu || l > ru) return;

    if(lu <= l && r <= ru){

        st[at] += (r - l + 1) * x; // updates node

        if(l != r){ // updates children's lazyness
              lazy[at<<1] += x;
              lazy[(at<<1) + 1] += x;
        }

        return;

    }

    int mid = (l + r)>>1;

    update((at<<1), l, mid, lu, ru, x);
    update((at<<1) + 1, mid + 1, r, lu, ru, x);

    st[at] = st[at<<1] + st[(at<<1) + 1];
}

//node "at" comprehends interval [l , r]
//querying interval [lq , rq]

int query(int at, int l, int r, int lq, int rq){

    if(lazy[at]){

        st[at] += (r - l + 1) * lazy[at]; // update lazy array

        if(l != r){ // spread lazyness to children
              st[at<<1] += lazy[at];
              st[(at<<1) + 1] += lazy[at];
        }

        lazy[at] = 0;
    }

    if(r < lq || l > rq) return 0;
    if(lq <= l && r <= rq) return st[at]; //we can return immediately if updated

    int mid = (l + r)>>1;

    return query((at<<1), l, mid, lq, rq) + query((at<<1) + 1, mid + 1, r, lq, rq);
}
```