Coding-Library

August 15, 2023

Contents

1. Bits . . . 2
1..1 Bit Manipulation . . . 2
2. Data Stuctures . . . 2
2..1 BIT . . . 2
2..2 Backup Segment Tree . . . 2
2..3 MO's Algorithm . . . 2
2..4 PQ . . . 3
2..5 Segment Tree Lazy . . . 3
2..6 Segment Tree . . . 4
3. Graph . . . 4
3..1 0-1 BFS . . . 4
3..2 Bellman-Ford . . . 4
3..3 DSU . . . 4
3..4 Dijkstra . . . 5
3..5 Floyd . . . 5
3..6 Kruskal MST . . . 5
3..7 LCA . . . 5
3..8 Tree-Diameter . . . 6
4. MISC . . . 6
4..1 Coordinate compression . . . 6
4..2 Ternary Search . . . 6
4..3 in128 . . . 7
5. Number theory . . . 7
5..1 Chinese Remainder . . . 7
5..2 Euler's totient . . . 7
5..3 Fast Power . . . 7
5..4 Josephus . . . 7
5..5 Sieve . . . 7
5..6 nCr with Mod Inverse . . . 8
5..7 nPr . . . 8
6. Strings . . . 8
6..1 KMP . . . 8
6..2 Trie . . . 8
6..3 Z . . . 8

1. Bits

1..1 Bit Manipulation

```
y = (x & (1 << i)) // Get the i-th bit
x |= (1 << i)      // Set the i-th bit
x &= ~(1 << i)     // Clear the i-th bit
(x && !(x &(x-1))) // if power of 2
1 << (32 - __builtin_clz (n - 1)) // Next power of 2
n & (-n)           // LSB
```

2. Data Stuctures

2..1 BIT

```
struct BIT {
    vector<int> T;
    int n;

    BIT(int n) {
        this->n = n + 1;
        T.assign(n + 1, 0);
    }

    BIT(vector<int> a) : BIT(a.size()) {
        for (int i = 0; i < a.size(); i++)
            add(i, a[i]);
    }

    ll sum(int idx) {
        ll ret = 0;
        for (++idx; idx > 0; idx -= idx & -idx)
            ret += T[idx];
        return ret;
    }

    void add(int idx, int delta) {
        for (++idx; idx < n; idx += idx & -idx)
            T[idx] += delta;
    }
};
```

2..2 Backup Segment Tree

```
struct SegmentTree{
private:
    vector<int>seg;int sz,skip=INT_MAX;
    int merge(int a,int b)
    {
        return min(a,b);
    }
    void update(int l,int r,int node,int idx,int val)
    {
        if(l==r)
        {
            seg[node]=val;
            return;
        }
        int mid=l+r>>1;
        if(mid<idx)
        {
            update(mid+1,r,2*node+2,idx,val);
        }
        else
        {
            update(l,mid,2*node+1,idx,val);
        }
        seg[node]=merge(seg[2*node+1],seg[2*node+2]);
    }
    int query(int l,int r,int node,int lx,int rx)
    {
        if(l>rx||r<lx)return skip;
        if(l>=lx&&r<=rx)return seg[node];
        int mid=l+r>>1;
        int a=query(l,mid,2*node+1,lx,rx);
        int b=query(mid+1,r,2*node+2,lx,rx);
        return merge(a,b);
    }
public:
    SegmentTree(int n)
    {
        sz=1;
        while(sz<=n)sz<<=1;
        seg=vector<int>(sz<<1,skip);
    }
    void update(int idx,int val)
    {
        update(0,sz-1,0,idx,val);
    }
    int query(int l,int r)
    {
        return query(0,sz-1,0,l,r);
    }
};
```

2..3 MO's Algorithm

```
void remove(idx); // TODO: remove value at idx from data
     structure
void add(idx);   // TODO: add value at idx from data
     structure
int get_answer(); // TODO: extract the current answer of the
      data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the
         range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}

//~ Based on the problem we can use a different data
     structure and modify the add/remove/get_answer
```

```
    functions accordingly. For example if we are asked to
    find range sum queries then we use a simple integer as
    data structure, which is
//~  $0$  at the beginning. The add function will simply add
     the value of the position and subsequently update the
    answer variable. On the other hand remove function will
     subtract the value at position and subsequently update
     the answer variable. And get_answer just returns the
    integer.

//~ For answering mode-queries, we can use a binary search
    tree (e.g. map<int, int>) for storing how often each
    number appears in the current range, and a second
    binary search tree (e.g. set<pair<int, int>>) for
    keeping counts of the numbers (e.g. as count-number
    pairs) in order. The add method removes the current
    number from the second BST, increases the count in the
    first one, and inserts the number back into the second
    one. remove does the same thing, it only decreases the
    count. And get_answer just looks at second tree and
    returns the best value in
//~ $O(1) $ .
```

2..4 PQ

```
// based on first part in ascending and second part in
// descending first basis
class Compare {
public:
    bool operator()(PII below, PII above)
    {
        if (below.first > above.first) {
            return true;
        }
        else if (below.first == above.first
                 && below.second < above.second) {
            return true;
        }

        return false;
    }
};

    priority_queue<PII, vector<PII>, Compare> ds;
```

2..5 Segment Tree Lazy

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

/*
Segment tree with lazy propagation
lazy[index] - array for updating the values.
tree[index] - array for finding the sum of elements in a
     particular range.
Basic technique used - update the array elements range when
     they are required to be updated
        till then keep hold the sum which is to be updated in
             that range .
qs - query starting
qe - query ending
ss - segment starting
se - segment ending
2*index+1 and 2*index+2 as childs and index as their parent.
*/

#define ll long long

#define MAX 200010
ll a[MAX] , t[4*MAX] , lazy[4*MAX];
void propagate(int ss ,int se, int idx)
{
    t[idx] += lazy[idx]*(se-ss+1);
    // *(se-ss+1) to add the inc value of children nodes to
         curr node
    // only in sum or similar queries
    if(ss!=se)
    {
        lazy[2*idx+1]+= lazy[idx];
        lazy[2*idx+2]+= lazy[idx];
    }
    lazy[idx] = 0 ;
}
void update(int ss, int se, int qs, int qe, int idx ,ll
     value)
{
    if(lazy[idx]!=0)
        propagate(ss, se, idx);
    if(qs>se || qe<ss)
        return;
    if(qs<=ss && qe>=se)
    {
        lazy[idx] = value;
        propagate(ss, se, idx);
        return ;
    }
    int mid = (ss+se)/2;
    update(ss, mid , qs, qe, 2*idx+1 , value);
    update(mid+1, se, qs ,qe ,2*idx+2 , value);
    t[idx] = t[2*idx+1] + t[2*idx+2];
}
ll get(int ss, int se, int qs, int qe, int idx)
{
    if(lazy[idx]!=0)
        propagate(ss, se, idx);
    if(qs>se || qe<ss)
        return 0;
    if(qs<=ss && qe>=se)
        return t[idx];

    int mid = (ss+se)/2;
    return get(ss, mid, qs ,qe , 2*idx+1) + get(mid+1 , se,
         qs , qe, 2*idx+2);
}
void build(int ss, int se, int idx)
{
    if(ss==se)
    {
        t[idx] = a[ss];
        return;
    }
    int mid= (ss+se)/2;
    build(ss , mid , 2*idx+1);
    build(mid+1, se,2*idx+2);
    t[idx] = t[2*idx+1] + t[2*idx+2];
}

int main()
{



    int n, q;
    cin >> n >> q;
 for(int i = 0; i < 4*MAX; i++)
   lazy[i] = 0 ;
 for(int i =0; i < n; i++)
  cin >> a[i];
 build(0 , n-1, 0);
    while(q--)
    {
  int x; cin >> x;
  if(x == 1)
  {
   int l, r, v;
   cin >> l >> r >> v;
   update(0, n-1, l - 1, r -1, 0, v);
  }
  else
```

```
 {
  int ind; cin >> ind;
  cout << get(0, n-1, ind - 1, ind -1, 0) << '\n';
 }
}

}
```

2..6 Segment Tree

```
vt<ll>T;
void update(ll node, ll val){
 T[node] = val;
 for(int i = node/2 ; i > 0; i /= 2)
  T[i] = min(T[i*2], T[i * 2 + 1]);
}
ll range_min (ll node, ll ql, ll qr, ll node_l, ll node_r){
  if(node_l>= ql && node_r <= qr) return T[node];
  if(node_l > qr || node_r < ql) return LONG_LONG_MAX;
  ll mid = (node_l + node_r) / 2;
  return min(range_min(2*node, ql, qr, node_l, mid),
   range_min(2*node + 1, ql, qr, mid + 1, node_r));
}
void Solve()
{
 ll n, q;
  cin >> n >> q;
 int temp = 1;
 int in_size = n;
 while(temp <= n)
  temp <<= 1;
 n = temp;
 T.assign(2 * n, LONG_LONG_MAX);
 for(int i = 0; i < in_size; i++) cin >> T[i + n];
 for(int i = n - 1 ; i >= 1; i--) T[i] = min(T[2*i], T[2*i +
       1]);

 while(q--)
 {
  ll x, l, r;
  cin >> x >> l >> r;
  if(x == 1) update(l + n - 1, r); // point update set v[l]
       = r
  if(x == 2)
  cout << range_min(1, l, r, 1, n) << nl;
 }

}
```

3. Graph

3..1 0-1 BFS

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

3..2 Bellman-Ford

```
#define ar array
#define ll long long

const int MAX_N = 2.5e3 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e15;

int n, m, par[MAX_N];
vector<ar<ll,2>> adj[MAX_N];
vector<ll> dist;

void bellman_ford(int s) {
    dist.assign(n + 1, LINF);
    dist[s] = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int u = 1; u <= n; u++) {
            for (auto [v, w] : adj[u]) {
                if (dist[u] + w < dist[v]) {
                    par[v] = u;
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
}

void cycle_detect() {
    int cycle = 0;
    for (int u = 1; u <= n; u++) {
        for (auto [v, w] : adj[u]) {
            if (dist[u] + w < dist[v]) {
                cycle = v;
                break;
            }
        }
    }
    if (!cycle) cout << "NO\n";
    else {
        cout << "YES\n";
        // backtrack to print the cycle
        for (int i = 0; i < n; i++) cycle = par[cycle];
        vector<int> ans; ans.push_back(cycle);
        for (int i = par[cycle]; i != cycle; i = par[i]) ans.
             push_back(i);
        ans.push_back(cycle);
        reverse(ans.begin(), ans.end());
        for (int x : ans) cout << x << " ";
        cout << "\n";
    }
}

void solve() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        adj[u].push_back({v, w});
    }
    bellman_ford(1);
    cycle_detect();
}
```

3..3 DSU

```
struct dsu {
    vt<int> par, sz;
    explicit dsu(int n)
    {
        par.assign(n+1, 0);
        iota(all(par), 0);
        sz.assign(n+1, 1);

    }
    int get_par(int x) {
```

```
        if (par[x] == x) return x;
        return par[x] = get_par(par[x]);
    }
    bool join(int a, int b) {
        a = get_par(a), b = get_par(b);
        if (a == b) return false;
        if (sz[a] > sz[b]) swap(a, b);
        par[a] = par[b];
        sz[b] += sz[a];
        return true;
    }
};
```

3..4 Dijkstra

```
vector<int> dist; // answer -> dist[destination]
vector<vector<pair<int,int>>> g; // {cost, to}

void djikstra(int src = 1) {
    dist.resize(g.size(), INFINITY);
    priority_queue<pair<int,int>, vector<pair<int,int>>,
         greater<pair<int,int>>> pq;
    pq.push({dist[src] = 0, src});
    while(!pq.empty()) {
        auto curr = pq.top(); pq.pop();
        for(auto to : g[curr.second])
            if(dist[to.second] > curr.first + to.first)
                pq.push({dist[to.second] = curr.first + to.
                     first, to.second});
    }
}
```

3..5 Floyd

```
// Find all pair shortest paths
// Time complexity: O(n^3)
// Problem link: https://cses.fi/problemset/task/1672

#include <bits/stdc++.h>

using namespace std;

#define ar array
#define ll long long

const int MAX_N = 500 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e15;

int n, m, q;
ll dist[MAX_N][MAX_N];

void floyd_warshall() { // 4 lines
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist
                     [k][j]);
}

void solve() {
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dist[i][j] = (i == j) ? 0 : LINF;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        dist[u][v] = dist[v][u] = min(dist[u][v], (ll)w);
    }
    floyd_warshall();
    while (q--) {
        int u, v; cin >> u >> v;
        cout << (dist[u][v] < LINF ? dist[u][v] : -1) << "\n"
             ;
    }
}
```

3..6 Kruskal MST

```
vector<int> parent, rank;

void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
parent.resize(n);
rank.resize(n);
for (int i = 0; i < n; i++)
    make_set(i);

sort(edges.begin(), edges.end());

for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e);
        union_sets(e.u, e.v);
    }
}
```

3..7 LCA

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
```

```
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

3..8 Tree-Diameter

```
void DFS( vector<int> graph[] , int node , int d )
{
    // marking the node as visited
    vis[node] = 1 ;


    // if the distance from root is greater then maximum
        Distance then updating the maximum value of distance
    // also storing the maxNode no. as this node is now at
        the farthest distance
    if( d > maxD )
    {
        maxNode = node ;
        maxD = d ;
    }

     // applying the standard dfs
    for( auto x : graph[node] )
    {
        if( vis[x] == 0 )
        {
            DFS( graph , x , d+1 ) ;
        }
    }

}



// Applyting DFS from node 1
DFS( graph , 1 , 1 ) ;
// we could have choosen any node in the graph but for
     simplicity we have choosen node 1


// making every node unvisited as we will be applying DFS
maxD = -1 ;
for( int i = 1 ; i<=8 ; i++ )
{
    vis[i] = 0 ;
}

// applying the dfs for the second time as this will give
     the diameter of the tree
DFS( graph , maxNode , 1 ) ;


// Now printing the maximum diameter of the tree
cout<<maxD<<" is the diameter of the tree "<<endl;
```

4. MISC

4..1 Coordinate compression

```
//method 1

void compress(vector<ll>&a,int start)
{
    int n=a.size();
    vector<pair<ll,ll>>pairs(n);
    for(int i=0;i<n;i++)
    {
        pairs[i]={a[i],i};
    }
    sort(pairs.begin(),pairs.end());
    int nxt=start;
    for(int i=0;i<n;i++)
    {
        if(i>0&&pairs[i-1].first!=pairs[i].first)
        {
            nxt++;
        }
        a[pairs[i].second]=nxt;
    }
}

//method 2
int getCompressedIndex(int a) {
 return lower_bound(v.begin(), v.end(), a) - v.begin();
}

 //========= COORDINATE COMPRESSION =======
 sort(v.begin(), v.end());
 indices.erase(unique(v.begin(), v.end()), v.end());

//  another method :

// use map to store value of ind and map to store ind of val
```

4..2 Ternary Search

```
int ternarySearch(int left, int right) {
    while (right - left >= 3) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (func(mid1) < func(mid2)) {
            right = mid2;
        } else {
            left = mid1;
        }
    }

    int minValue = func(left);
    for (int i = left + 1; i <= right; ++i) {
        minValue = min(minValue, func(i));
    }

    return minValue;
}
```

```
double ternarySearch(double left, double right, double error
    ) {
    while (right - left > error) {
        double mid1 = left + (right - left) / 3;
        double mid2 = right - (right - left) / 3;
        double f_mid1 = func(mid1);
        double f_mid2 = func(mid2);

        if (f_mid1 < f_mid2) {
            right = mid2;
        } else {
            left = mid1;
        }
    }

    return (left + right) / 2;
}
```

4..3 in128

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
bool cmp(__int128 x, __int128 y) { return x > y; }
```

5. Number theory

5..1 Chinese Remainder

```
// k is size of num[] and rem[]. Returns the smallest
// number x such that:
//  x % num[0] = rem[0],
//  x % num[1] = rem[1],
//  ..................
//  x % num[k-2] = rem[k-1]
// Assumption: Numbers in num[] are pairwise coprime
// (gcd for every pair is 1)
int findMinX(int num[], int rem[], int k)
{
    int x = 1; // Initialize result

    // As per the Chinese remainder theorem,
    // this loop will always break.
    while (true)
    {
        // Check if remainder of x % num[j] is
        // rem[j] or not (for all j from 0 to k-1)
        int j;
        for (j=0; j<k; j++ )
            if (x%num[j] != rem[j])
               break;

        // If all remainders matched, we found x
        if (j == k)
            return x;

        // Else try next number
        x++;
    }

    return x;
}
```

5..2 Euler's totient

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

5..3 Fast Power

```
ll power(ll b, ll p, ll mod) {
    ll res = 1;
    b = b % mod;
    if (b == 0) return 0;
    while (p)
    {
        if (p & 1)
            res = (res * b) % mod;
        p >>= 1;
        b = (b * b) % mod;
    }
    return res;
}
```

5..4 Josephus

```
int josephus(int n, int k) {
    if (n == 1)
        return 1;
    if (k <= (n + 1) / 2) {
        if (2 * k > n) return 2 * k % n;
        else return 2 * k;
    }
    int c = josephus(n >> 1, k - ( n + 1) / 2);
    if (n & 1) return 2 * c + 1;
    else return 2 * c - 1;
}
```

5..5 Sieve

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
```

```
    }
}
```

5..6 nCr with Mod Inverse

```
ll modInverse(ll n, ll mod)
{
    return power(n, mod - 2, mod);
}

ll nCr(ll n, ll r, ll mod)
{
    if (n < r)
        return 0;
    if (r == 0)
        return 1;
    ll fac[n + 1];
    fac[0] = 1;
    for (int i = 1; i <= n; i++)
        fac[i] = (fac[i - 1] * i) % mod;
    return (fac[n] * modInverse(fac[r], mod) % mod
            * modInverse(fac[n - r], mod) % mod)
        % mod;
}
```

5..7 nPr

```
int nPr(int n, int r)
{
 ll ans = 1;
 for (int i = 0; i < (n - r); i++)
  ans *= (n - i);
}
```

6. Strings

6..1 KMP

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

6..2 Trie

```
struct Trie{
    struct Node{
        Node*child[26];
        int IsEnd,Prefix;
        Node(){
            memset(child,0,sizeof child);
            IsEnd=Prefix=0;
        }
    };
    Node*root=new Node();
    void insert(string &s)
    {
        Node*cur=root;
        for(auto it:s)
        {
            int idx=it-'a';
            if(cur->child[idx]==0)
            {
                cur->child[idx]=new Node();
            }
            cur=cur->child[idx];
            cur->Prefix++;
        }
        cur->IsEnd++;
    }
    bool SearchWord(string &s)
    {
        Node*cur=root;
        for(auto it:s)
        {
            int idx=it-'a';
            if(cur->child[idx]==0)return 0;
            cur=cur->child[idx];
        }
        return cur->IsEnd;
    }
    int CountWord(string &s)
    {
        Node*cur=root;
        for(auto it:s)
        {
            int idx=it-'a';
            if(cur->child[idx]==0)return 0;
            cur=cur->child[idx];
        }
        return cur->IsEnd;
    }
    int CountPrefix(string &s)
    {
        Node*cur=root;
        for(auto it:s)
        {
            int idx=it-'a';
            if(cur->child[idx]==0)return 0;
            cur=cur->child[idx];
        }
        return cur->Prefix;
    }
};
```

6..3 Z

```
//~ The Z-function for this string is an array of length
//~  $n$  where the
//~  $i$ -th element is equal to the greatest number of
     characters starting from the position
//~  $i$  that coincide with the first characters of
//~  $s$ .

//~ In other words,
//~ $z[i] $  is the length of the longest string that is, at
      the same time, a prefix of
//~  $s$  and a prefix of the suffix of
//~  $s$  starting at
//~  $i$ .
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}

//~ String compression
```

```
//~ Given a string
//~  $s$  of length
//~  $n$ . Find its shortest "compressed" representation,
     that is: find a string
//~  $t$  of shortest length such that
//~  $s$  can be represented as a concatenation of one or
     more copies of
//~  $t$ .

//~ A solution is: compute the Z-function of
//~  $s$ , loop through all
//~  $i$  such that
//~  $i$  divides
//~  $n$ . Stop at the first
//~  $i$  such that
//~ $i + z[i] = n$. Then, the string
//~  $s$  can be compressed to the length
//~  $i$ .
```