**Here you will find the following topics:**

**#DS:**

1. BIT(2D)

2. RMQ(1D & 2D)

3. Implicit Segment Tree

4. SQRT Decomposition

5. MO’s

6. TRIE

**#NetWorkFlow:**

1. Hopcroft-karp BPM

2. Kuhn’s

3. Edmond’s

**#Divide and Conquer:**

1. Mat Expo

**#String:**

1.Suffix Array

**#MATH:**

1. EGCD

2. Mod by Square of Prime

3. Lucas Theorem

\*\*\*\*\*\*\*\*\*\*\*\*\*\* **DATA STRUCTURES** \*\*\*\*\*\*\*\*\*\*\*\*\*

**TRIE:**

struct node

{

bool endMark;

node \* next[27];

node ()

{

endMark = false;

for(int i = 0; i < 26; i++)

{

next[i] = NULL;

}

}

} \*root;

void inser(string str)

{

int sz = str.size();

node \*curr = root;

for(int i = 0; i < sz ; i++)

{

int id = str[i] - 'a';

if(curr->next[id]==NULL)

{

curr->next[id] = new node();

}

curr = curr->next[id];

}

curr->endMark = true;

}

bool search(string str)

{

int sz = str.size();

node \*current = root;

for(int i = 0;i < sz ; i++)

{

int id = str[i] - 'a';

if(current->next[id] == NULL)

{

return false;

}

current = current->next[id];

}

return current->endMark;

}

void del(node \*current)

{

for(int i = 0; i < 26; i++)

{

if(current->next[i] != NULL)

{

del(current->next[i]);

}

}

delete(current);

}

\*\*\*\*\*\*\*\*\***2D BIT**\*\*\*\*\*\*

const int mxn = 1008;

int bit[mxn+3][mxn+3];

bool vis[mxn+3][mxn+3];

void update(int idx , int idy , int v)

{

int x = idx;

while(x <= mxn)

{

int y = idy;

while( y <= mxn)

{

bit[x][y] +=v;

y += (y&-y);

}

x += (x&-x);

}

}

int query(int idx ,int idy)

{

int ret = 0;

int x = idx;

while(x)

{

int y = idy;

while(y)

{

ret += bit[x][y];

y -= (y&-y);

}

x -= (x&-x);

}

return ret;

}

int solve(int x1 , int y1 , int x2 , int y2)

{

int ans = query(x2 , y2) - query(x2 , y1-1) - query(x1-1 , y2) + query(x1-1 , y1-1);

return ans;

}

\*\*\*\*\*\*\*\*\*\*\* **1D RMQ** \*\*\*\*\*\*\*\*\*\*\*

const int mxn = 1e5;

#define MOD 1e9+7

int n , m;

int ar[mxn+10];

int table[20][mxn+10];

void RMQ()

{

for(int ir = 0; ir < n; ir++)

{

table[0][ir] = ar[ir];

}

for(int jr = 1 ; (1<<jr) <= n; jr++)

{

for(int ir = 0; ir + (1<<jr)-1 < n ; ir++)

{

table[jr][ir] = min(table[jr-1][ir] , table[jr-1][ir+(1<<(jr-1))]);

}

}

}

int query(int x , int y)

{

int len = y-x+1;

len = log(double(len))/log(2.0D);

return min( table[len][x] , table[len][y+1- (1<<len)]);

}

\*\*\*\*\*\*\*\*\*\*\* **2D RMQ** \*\*\*\*\*\*\*\*\*\*\*

typedef long long ll;

typedef pair< int , int > pii;

typedef vector< int > vi;

#define rept(i , n) for(int i = 0; i < n ; i++)

#define loop(i, n) for(int i = 1; i <= n ; i++)

const int mxn = 1e3;

#define MOD 1e9+7

int ar[mxn+10][mxn+10] ;

int spTable[12][12][mxn+10][mxn+10];

int n ,m;

ll sum[mxn+10][mxn+10];

void RMQ()

{

for(int jr =0; (1 << jr) <= n; jr++)

{

for(int jc = 0; (1 << jc) <= m ; jc++)

{

for(int ir = 0; ir+(1<<jr)-1 < n; ir++)

{

for(int ic =0; ic+(1<<jc)-1 < m; ic++)

{

if(jr==0 && jc == 0)

{

spTable[jr][jc][ir][ic] = ar[ir][ic];

}

else if(jr==0)

{

spTable[jr][jc][ir][ic] = max( spTable[jr][jc-1][ir][ic] , spTable[jr][jc-1][ir][ic+(1<<(jc-1))]);

}

else if(jc==0)

{

spTable[jr][jc][ir][ic] = max( spTable[jr-1][jc][ir][ic] , spTable[jr-1][jc][ir+(1<<(jr-1))][ic]);

}

else

{

spTable[jr][jc][ir][ic] = max( spTable[jr-1][jc-1][ir][ic], max( spTable[jr-1][jc-1][ir + (1<<(jr-1))][ic],max(spTable[jr-1][jc-1][ir][ic+(1<<(jc-1))] , spTable[jr-1][jc-1][ir+(1<<(jr-1))][ic+(1<<(jc-1))])));

}

}

}

}

}

}

ll query(int x1 , int y1 , int x2, int y2)

{

int lenx = x2 - x1+1;

int leny = y2 - y1 + 1;

lenx = log( (double)lenx)/log(2.0D);

leny = log((double)leny) / log(2.0D);

ll ans = max( (long long)spTable[lenx][leny][x1][y1] ,(long long) spTable[lenx][leny][x1][y2+1-(1<<leny)]);

ans = max( ans ,(long long) max( (long long)spTable[lenx][leny][x2+1-(1<<lenx)][y1] , (long long)spTable[lenx][leny][x2+1-(1 << lenx)][y2+1-(1 << leny)]));

return ans;

}

\*\*\*\*\*\*\*\* **Sqrt Decomposition** \*\*\*\*\*\*\*

const int mxn = 100000;

struct data

{

int st , en;

deque<int > dq;

int fr[mxn+3];

} backet[325];

int ar[mxn+3] , bcnt ;

int const lm = 1500;

void decomposition(int n)

{

bcnt = 0;

for(int i =0 , c = 0; i < n ; i++, c++)

{

if(c==lm)

{

backet[bcnt].en = i-1;

c = 0;

bcnt++;

backet[bcnt].st = i;

}

backet[bcnt].dq.push\_back(ar[i]);

backet[bcnt].fr[ar[i]]++;

}

backet[bcnt].en = n-1;

}

void update(int l , int r)

{

int lb = (int)l/lm;

int rb = (int) r/lm;

int val = backet[rb].dq[r-backet[rb].st];

backet[rb].dq.erase( backet[rb].dq.begin()+ r%lm);

backet[rb].fr[val]--;

backet[lb].dq.insert(backet[lb].dq.begin()+l%lm, val);

backet[lb].fr[val]++;

for(int i = lb+1; i <= rb; i++)

{

backet[i].dq.push\_front(backet[i-1].dq.back());

backet[i].fr[backet[i].dq.front()]++;

backet[i-1].fr[backet[i].dq.front()]--;

backet[i-1].dq.pop\_back();

}

}

int query(int l , int r , int k)

{

int lb = (int) l/lm;

int rb = (int) r/lm;

int ret = 0;

for(int i = l - backet[lb].st; i <= min( backet[lb].en - backet[lb].st , r - backet[lb].st); i++)

{

ret +=(backet[lb].dq[i]==k);

}

if(lb!=rb){

for(int i = 0; i <= r - backet[rb].st; i++)

{

ret += (backet[rb].dq[i]==k);

}

for(int i = lb+1; i < rb; i++)

{

ret += backet[i].fr[k];

}

}

return ret;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **MO’s** \*\*\*\*\*\*\*\*\*\*\*\*\*

int left = 0;

int right = -1;

while(right < p.first)

{

right++;

add(right);

}

while(right > p.first)

{

remov(right);

right--;

}

while(left< p.second)

{

remov(left);

left++;

}

while(left > p.second)

{

left--;

add(left);

}

\*\*\*\*\*\*\*\*\*\*\*\*\* **Implicit Segment Tree** \*\*\*\*\*\*\*\*

typedef long long ll;

const ll mod = 1e9+7;

struct NODE

{

ll sum;

NODE\* left , \*right;

NODE()

{

left = NULL;

right = NULL;

sum = 0;

}

};

void update(NODE \*node , ll beg , ll endd , ll pos , ll val)

{

if(beg==pos&&endd==pos)

{

node->sum =val;

return;

}

ll mid = (beg + endd)>>1;

if(node->left==NULL)

{

node->left = new NODE();

}

if(node->right==NULL)

{

node->right = new NODE();

}

if(pos <= mid)

update(node->left, beg , mid , pos , val);

else

{

update(node->right, mid+1, endd , pos , val);

}

node->sum = node->left->sum + node->right->sum;

return;

}

ll query(NODE\* node , ll beg ,ll endd, ll l , ll r)

{

if(node == NULL || l > r)

return 0;

if(l==beg && endd==r)

return node->sum;

ll mid = (beg+endd)>>1;

ll r1 =query(node->left , beg, mid, l , min(mid,r));

ll r2= query(node->right, mid+1, endd, max(l, mid+1), r);

return r1+r2;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NETWORK FLOW \*\*\*\*\*\*\*\*\*\*\*\*\*\*

**– Edmond’s Bro:**

const int mxn = 1e2+10;

int flowMatrix[mxn][mxn] , maxFlow , curFlow, src , sink;

bool vis[mxn];

vector< int > path;

void augment(int v , int minEdge)

{

if(v==src)

{

curFlow = minEdge;

return;

}

if(path[v]!=-1)

{

augment(path[v] , min(minEdge , flowMatrix[path[v]][v]));

flowMatrix[path[v]][v] -= curFlow;

flowMatrix[v][path[v]] += curFlow;

}

}

void EdmondsBro(int n)

{

maxFlow = 0;

while(true)

{

curFlow = 0;

memset(vis , false , sizeof vis);

dis[src] = 0;

vis[src] = true;

queue<int> Q;

Q.push(src);

path.assign(mxn , -1);

while(!Q.empty())

{

int u = Q.front();

Q.pop();

if(u==sink)

{

break;

}

for(int v = 1; v <= n; v++)

{

if(flowMatrix[u][v] > 0 && !vis[v])

{

vis[v] = true;

dis[v] = dis[u] + 1;

Q.push(v);

path[v] = u;

}

}

}

augment(sink , INT\_MAX);

if(curFlow==0)

break;

maxFlow += curFlow;

}

}

**--- Kuhn’s Bro:**

const int mxn = 1100;

int matchR[mxn] , matchL[mxn];

bool vis[mxn];

vector<int > adj[mxn];

int n , m;

int kuhnBro(int u)

{

vis[u] = true;

for(int i = 0; i < (int)adj[u].size(); i++)

{

int v = adj[u][i];

if(matchR[v]==-1 || (!vis[matchR[v]] && kuhnBro(matchR[v])))

{

matchL[u] = v;

matchR[v] = u;

return 1;

}

}

return 0;

}

int main()

{

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

memset(matchL,-1,sizeof matchL);

memset(matchR,-1,sizeof matchR);

int tot\_cnt =0;

for(int i = 1; i <= n; i++)

{

memset(vis , false ,sizeof vis);

tot\_cnt +=kuhnBro(i);

}

}

**–- Hopcroft-Karp:**

const int mxn = 2\*1e4;

const int inf = INT\_MAX;

const int NIL = 0;

int cntU , cntV;

int matchU[mxn] , matchV[mxn];

int dis[mxn];

vector< int > adj[mxn];

int n, m;

bool bfs()

{

queue< int > Q;

for(int i = 1; i <= cntU; i++)

{

if(matchU[i]==NIL)

{

dis[i] = 0;

Q.push(i);

}

else

{

dis[i] = inf; }}

dis[NIL] = inf;

while(!Q.empty())

{

int u = Q.front();

Q.pop();

if(dis[u] < dis[NIL])

{

for(int i = 0; i < adj[u].size(); i++)

{

int v = adj[u][i];

if(dis[matchV[v]] == inf)

{

dis[matchV[v]] = dis[u] + 1;

Q.push(matchV[v]);

}

}

}

}

return (dis[NIL] != inf);

}

bool dfs(int u)

{

if(u!= NIL)

{

for(int i = 0; i < adj[u].size(); i++)

{

int v = adj[u][i];

if(dis[matchV[v]] == dis[u]+1)

{

if(dfs(matchV[v])==true)

{

matchU[u] = v;

matchV[v] = u;

return true;

}

}

}

dis[u] = inf;

return false;

}

return true;

}

int HopCroftKarpBro()

{

for(int i = 0; i <= cntU; i++)

{

matchU[i] = NIL;

}

for(int i = 0; i <= cntV; i++)

{

matchV[i] = NIL;

}

int tot\_cnt = 0;

while(bfs())

{

for(int i = 1; i <= cntU; i++)

{if(matchU[i]==NIL && dfs(i)) tot\_cnt++;}}return tot\_cnt;}

\*\*\*\*\*\*\*\*\*\*\* DIVIDE AND CONQUER \*\*\*\*\*\*\*\*\*\*

**---Matrix Expo:**

\*\*\*\*\*\*\* Mat. Expo\*\*\*\*

const int mxn = 2;

typedef unsigned long long ull;

struct Matrix

{

ull mat[mxn][mxn];

};

Matrix mul(Matrix a , Matrix b)

{

Matrix ans;

for(int i = 0; i < mxn; i++)

{

for(int j = 0; j < mxn; j++)

{

ans.mat[i][j] = 0;

for(int k = 0; k < mxn; k++)

{

ans.mat[i][j] += a.mat[i][k]\*b.mat[k][j];

}

}

}

return ans;

}

Matrix expo(Matrix base , int p)

{

if(p==0)

{

Matrix ret;

for(int i = 0; i < mxn; i++)

{

for(int j = 0; j < mxn; j++)

{

ret.mat[i][j] = (i==j);

}

}

return ret;

}

Matrix x = expo(base , p/2);

x= mul(x , x);

if(p&1)

{

x = mul(x , base);

}

return x;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\* **STRING** \*\*\*\*\*\*\*\*\*\*\*\*

**--Suffix Array:**

#include<bits/stdc++.h>

using namespace std;

const int mxn = 262144;

const int mxlg = 22;

int suffix[mxlg][mxn] , Rank[mxn] , lcp[mxn];

struct entry

{

int nr[2],p;

}L[mxn];

bool cmp(entry const a , entry const b)

{

return a.nr[0] == b.nr[0] ? (a.nr[1] < b.nr[1] ? true : false) : a.nr[0] < b.nr[0] ? true : false;

}

string str;

int lg , ln , stp;

void build\_suffix\_array()

{

ln = str.size();

for(int i=0; i < ln; i++)

{

suffix[0][i] = str[i] - 'a';

}

for(stp = 1, lg = 1; (lg >> 1) < ln; stp++ , lg <<=1)

{

for(int i = 0 ; i < ln; i++)

{

L[i].nr[0] = suffix[stp-1][i];

L[i].nr[1] = i + lg < ln ? suffix[stp-1][i+lg] : -1;

L[i].p = i;

}

sort(L, L + ln, cmp);

for(int i = 0; i < ln; i++)

{

suffix[stp][L[i].p] = i > 0 && L[i].nr[0] == L[i-1].nr[0] && L[i].nr[1] == L[i-1].nr[1] ? suffix[stp][L[i-1].p] : i ;

}

}

stp-=1;

for(int i = 0; i < ln; i++)

{

cout<<suffix[stp][i]<<" ";

}

cout<<endl;

}

void big\_o\_LCP()

{

for(int i =0 ; i < ln; i++)

{

Rank[suffix[stp][i]] = i;

}

int now = 0;

for(int i = 0; i < ln; i++)

{

now = max(now-1 , 0);

if(suffix[stp][i]==ln-1)

{

now = 0;

continue;

}

int j = Rank[suffix[stp][i]+1];

while( i + now < ln && j + now < ln && str[i+now] == str[j+now])

now++;

lcp[suffix[stp][i]] = now;

}

}

int nlogn\_lcp(int x , int y)

{

int k , ret = 0;

if(x==y)

return ln-x;

for(int k = stp; k >= 0 && x < ln && y < ln; k--)

{

if(suffix[k][x] == suffix[k][y])

{

x += ( 1 << k);

y +=( 1 << k);

ret += (1<<k);

}

}

return ret;

}

int main()

{

cin>>str;

build\_suffix\_array();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MATH \*\*\*\*\*\*\*\*\*\***

**–--EGCD:**

void extendedEuclid(int A, int B) {

if(B == 0) {

d = A;

x = 1;

y = 0;

}

else {

extendedEuclid(B, A%B);

int temp = x;

x = y;

y = temp - (A/B)\*y;

}

}

**---- MOD by Square of Prime:**

typedef vector< int > VI;

typedef vector< pair<int , int > > VPI;

VI spfact[5] , invMod , pp , rem , phi ;

int pop;

VPI prime;

int bigmod(int b, int p, int m)

{

if(p==0) return 1;

int x = bigmod(b , p/2 , m);

x = (x\*x)%m;

if(p&1)

x = (x\*b)%m;

return x;

}

int power(int b , int p)

{

if(p==0) return 1;

int x = power(b , p/2);

x \*=x;

if(p&1) x \*= b;

return x;

}

int POPrime(int n , int p)

{

if(n==0) return 0;

return POPrime(n/p , p) + n/p;

}

int FOPrime(int n , int idx , int mod)

{

if(n== 0) return 1;

if(n < mod)

return spfact[idx][n]\* FOPrime(n/prime[idx].first , idx , mod);

int tmp = n/prime[idx].first;

int tmp3 = n/mod;

int tmp1 = spfact[idx][mod-1];

int tmp2 = n%mod;

tmp2 = spfact[idx][tmp2];

return (bigmod(tmp1 , tmp3 , mod)\*FOPrime(tmp , idx , mod)\*tmp2)%mod;

}

int comMOD(int n, int r , int idx)

{

int en = POPrime(n , prime[idx].first);

int er = POPrime(r , prime[idx].first);

int enr = POPrime(n-r , prime[idx].first);

if(en >= er + enr + prime[idx].second) return 0;

long long mulPrime = en - (er + enr);

mulPrime = power( prime[idx].first , mulPrime);

int tmpMod = power(prime[idx].first , prime[idx].second);

en = FOPrime(n , idx , power(prime[idx].first, prime[idx].second));

er = FOPrime(r , idx , power(prime[idx].first , prime[idx].second));

enr = FOPrime(n-r , idx, power(prime[idx].first, prime[idx].second));

return (((mulPrime\*bigmod((er\*enr)%tmpMod , phi[idx]-1 , tmpMod))%tmpMod)\*en)%tmpMod;

}

long long solve(int n , int r)

{

for(int i = 0; i < 4; i++)

{

rem.push\_back(comMOD(n , r , i));

}

long long ret = 0LL;

for(int i = 0; i < 4; i++)

{

ret = ret + (rem[i]\*invMod[i]\*pp[i])%pop;

//cout<<ret<<endl;

ret = ret % pop;

}

return ret;

}

int gcd(int a , int b)

{

if(b==0) return a;

return gcd(b , a%b);

}

VI sfcts(int m)

{

VI tmp;

tmp.assign(m , 1);

for(int i = 1; i< m; i++)

{

if(gcd(i , m)==1) tmp[i] = (tmp[i-1]\*i)%m;

else tmp[i] = tmp[i-1];

}

return tmp;

}

void init()

{

prime.push\_back(make\_pair(3 , 3));

prime.push\_back(make\_pair(11 , 1));

prime.push\_back(make\_pair(13 , 1));

prime.push\_back(make\_pair(37, 1));

phi.push\_back(18);

phi.push\_back(10);

phi.push\_back(12);

phi.push\_back(36);

pp.push\_back(11\*13\*37);

pp.push\_back(27\*13\*37);

pp.push\_back(27\*11\*37);

pp.push\_back(27\*11\*13);

pop = 27\*11\*13\*37;

for(int i = 0; i < 4; i++)

{

invMod.push\_back(bigmod(pp[i] , phi[i]-1, power(prime[i].first , prime[i].second)));

}

for(int i = 0; i < 4; i++)

spfact[i] = sfcts(power(prime[i].first , prime[i].second));

}

**--Lucas Theorem:**

int ncr[50][50][50];

int NCR(int n, int r, int m){

if(n < r) return 1;

if(r==0)

return ncr[n][r][m] = 1;

if(n==r)

return ncr[n][r][m] = 1;

if(r==1)

return ncr[n][r][m] = n;

if(ncr[n][r][m]!=-1)

return ncr[n][r][m];

return ncr[n][r][m] = (NCR(n-1 , r, m) + NCR(n-1 , r-1 , m))%m;

}int lucas(int N , int R , int m)

{

if(N < R)

return 1;if(R==0)return 1;return (lucas(N/m , R/m , m)\*NCR(N%m , R%m , m))%m;}