**2018.7.20模板补充打印**

**1.图论和数据结构**

/\*\***2-SAT**\*\*/

int n;

vector<int> g[maxn\*2];

bool mark[maxn\*2];

int s[maxn\*2],c;

bool dfs(int x){

if(mark[x^1]) return false;

if(mark[x]) return true;

mark[x]=true;

s[c++]=x;

for(int i=0;i<(int)g[x].size();i++)

if(!dfs(g[x][i])) return false;

return true;

}

void init(int n){

for(int i=0;i<n\*2;i++) g[i].clear();

memset(mark,0,sizeof(mark));

}

void add\_clause(int x,int y){//这个函数随题意变化

g[x].push\_back(y);//选了x就必须选y^1

}

bool solve(){

for(int i=0;i<n\*2;i+=2)

if(!mark[i]&&!mark[i+1]){

c=0;

if(!dfs(i)){

while(c>0) mark[s[--c]]=false;

if(!dfs(i+1)) return false;

}

}

return true;

}

/\***RMQ** HDU-4123\*/

void ST(int n) {

for (int j = 1; (1 << j) <= n; j++) {

for (int i = 1; i + (1 << j) - 1 <= n; i++) {

dp[i][j][0] = max(dp[i][j - 1][0], dp[i + (1 << (j - 1))][j - 1][0]);

dp[i][j][1] = min(dp[i][j - 1][1], dp[i + (1 << (j - 1))][j - 1][1]);

}

}

}

int RMQ(int l, int r) {

int k = 0;

while ((1 << (k + 1)) <= r - l + 1) k++;

return max(dp[l][k][0], dp[r - (1 << k) + 1][k][0])-min(dp[l][k][1], dp[r - (1 << k) + 1][k][1]);

}

/\***树状数组**\*/

int lowbit(int x){

return x&(-x);

}

int sum(int x){//求和

int ret=0;

while(x>0){

ret+=tree[x];

x-=lowbit(x);

}

return ret;

}

void update(int x,int d){//更新节点信息

if(x<0) return;

while(x<=n){

tree[x]+=d;

x+=lowbit(x);

}

}

/\*\***二维树状数组**\*/

int lowbit(int x){

return x&(-x);

}

void add(int x,int y){

for(int i=x;i<=n;i+=lowbit(i))

for(int j=y;j<=n;j+=lowbit(j))

tree[i][j]++;

}

int sum(int x,int y){

int ret=0;

for(int i=x;i>0;i-=lowbit(i))

for(int j=y;j>0;j-=lowbit(j))

ret+=tree[i][j];

return ret;

}

**2.数学**

/\*\***组合数**预处理\*/

for(int j=0;j<50;j++)

c[0][j]=1;

for(int i=1;i<50;i++)

for(int j=0;j<50;j++)

{

if(i==j) c[i][j]=1;

else if(j<i) c[i][j]=0;

else c[i][j]=(c[i-1][j-1]+c[i][j-1])%mod;

// cout<<j<<" "<<i<<" "<<c[i][j]<<endl;

}

/\***数位dp**  HDU-2089 不要62 \*/

int bit[30];

ll dp[30][2][2];

ll dfs(int pos,int st,int flag){

if(pos==0) return 1;

if(flag&&dp[pos][st][flag]!=-1)

return dp[pos][st][flag];

int u=flag?9:bit[pos];

ll ans=0;

for(int i=0;i<=u;i++){

if(i==4) continue;

else if(st==0&&i==2) continue;

else if(i!=6) ans+=dfs(pos-1,1,flag||i<u);

else if(i==6) ans+=dfs(pos-1,0,flag||i<u);

}

return ans;

}

ll solve(int n){

int len=0;

while(n){

bit[++len]=n%10;

n/=10;

}

return dfs(len,1,0);

}

/\*\***容斥原理**\*\*/

/\*\*

计算小于等于x的与p互质的数的个数

vector<int>d[p]中存p的质因子

\*\*/

ll cal(ll x,int p){

int m=d[p].size();

ll num=0;

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

ll ans=1;int ant=0;

for(int j=0;j<m;j++){

if(i&(1<<j)){

ans\*=d[p][j];

ant++;

}

}

if((ant-1)%2) num-=(x/ans);

else num+=(x/ans);

}

return x-num;

}

**3.字符串**

/\*\*\*\***AC自动机**HUD-2222\*\*\*\*/

struct trie{

int next[maxn][26],fail[maxn],ed[maxn];

int root,cnt;

int newnode(){

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

next[cnt][i]=-1;

ed[cnt++]=0;

return cnt-1;

}

void init(){

cnt=0;

root=newnode();

}

void inser(char\* buf){

int len=strlen(buf);

int now=root;

for(int i=0;i<len;i++){

if(next[now][buf[i]-'a']==-1)

next[now][buf[i]-'a']=newnode();

now=next[now][buf[i]-'a'];

}

ed[now]++;

}

void build(){

queue<int>que;

fail[root]=root;

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

if(next[root][i]==-1)

next[root][i]=root;

else{

fail[next[root][i]]=root;

que.push(next[root][i]);

}}

while(!que.empty()){

int now=que.front();

que.pop();

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

if(next[now][i]==-1)

next[now][i]=next[fail[now]][i];

else{

fail[next[now][i]]=next[fail[now]][i];

que.push(next[now][i]);

}

}

}

}

int query(char\* buf){

int len=strlen(buf);

int now=root;

int res=0;

for(int i=0;i<len;i++){

now=next[now][buf[i]-'a'];

int temp=now;

while(temp!=root){

res+=ed[temp];

ed[temp]=0;

temp=fail[temp];

}

}

return res;

}

};

char buf[maxn];

trie ac;

int main(){

ios\_base::sync\_with\_stdio(false);cin.tie(NULL);cout.tie(NULL);

int t;

cin>>t;

while(t--){

int n;

cin>>n;

ac.init();

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

cin>>buf,ac.inser(buf);

ac.build();

cin>>buf;

cout<<ac.query(buf);

}

return 0;

}

**4.一些操作**

/\*\***离散化操作**\*/

vector<int>v;

int getid(int x){

return lower\_bound(v.begin(),v.end(),x)-v.begin()+1;

}

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

scanf("%d",&a[i]),v.push\_back(a[i]);

sort(v.begin(),v.end()),v.erase(unique(v.begin(),v.end()),v.end());

getid(x);//取x离散化后的下标

v[id];//找到其离散前的元素

/\***bitset**\*/

bitset<100000>bt;

bt<<1;//整体移位

bt|=10;

bt.count();//b中置为1的二进制位的个数

bt.size();//b中二进制位的个数

bt[pos];//访问b中在pos处的二进制位

bt.test(pos);//b中在pos处的二进制位是否为1？

bt.set();//把b中所有二进制位都置为1

bt.set(pos);//把b中在pos处的二进制位置为1

bt.reset();//把b中所有二进制位都置为0

bt.reset(pos);//把b中在pos处的二进制位置为0

/\*\***java快速读入**\*\*/

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.\*;

import java.nio.Buffer;

import java.util.\*;

public class Main {

public static void main(String[] args) {

int n;

FastScanner sc=new FastScanner();

PrintWriter pw=new PrintWriter(System.out);

n = sc.nextInt();

int ans;

pw.println(ans);

pw.flush();

}

}

class FastScanner{

BufferedReader br;

StringTokenizer st;

public FastScanner()

{

try{

br=new BufferedReader(new InputStreamReader(System.in),32768);

st=new StringTokenizer("");

}

catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

}

public boolean hasNext() {

while(!st.hasMoreTokens()){

String line=nextLine();

if(line==null)

return false;

st=new StringTokenizer(line);

}

return true;

}

public String next()

{

while(!st.hasMoreTokens()){

st=new StringTokenizer(nextLine());

}

return st.nextToken();

}

public int nextInt(){

return Integer.parseInt(next());

}

public long nextLong(){

return Long.parseLong(next());

}

public double nextDouble(){

return Double.parseDouble(next());

}

public String nextLine(){

String line="";

try{

line=br.readLine();

}catch (Exception e) {

e.printStackTrace();

// TODO: handle exception

}

return line;

}

}