目录

[线性筛 埃拉托斯特尼筛法 nloglogn 2](#_Toc45033599)

[欧拉筛 包括Ф 2](#_Toc45033600)

[洲阁筛 3](#_Toc45033601)

[杜教筛μФ 5](#_Toc45033602)

[Min-25筛 求素数个数 7](#_Toc45033603)

[素数个数范围1e11 hdu5901 7](#_Toc45033604)

[Miller-Rabin判断素数 10](#_Toc45033605)

[最长公共前缀长度 11](#_Toc45033606)

[AC自动机 12](#_Toc45033607)

[Manacher马拉车 最长回文子串O(n) 14](#_Toc45033608)

[后缀数组luogu3809 16](#_Toc45033609)

[后缀自动机 18](#_Toc45033610)

[回文自动机luogu1659 19](#_Toc45033611)

[SPFA 21](#_Toc45033612)

[Dijkstra 23](#_Toc45033613)

[Dijkstra堆优化 25](#_Toc45033614)

[Dijkstra配对堆（？？？） 26](#_Toc45033615)

[Bellman-Ford 28](#_Toc45033616)

# 线性筛 埃拉托斯特尼筛法 nloglogn

int Eratosthenes(int n) {

int p = 0;

for (int i = 0; i <= n; ++i) is\_prime[i] = 1;

is\_prime[0] = is\_prime[1] = 0;

for (int i = 2; i <= n; ++i) {

if (is\_prime[i]) {

prime[p++] = i; // prime[p]是i,后置自增运算代表当前素数数量

for (int j = i \* i; j <= n;

j += i) // 因为从 2 到 i - 1 的倍数我们之前筛过了，这里直接从 i

// 的倍数开始，提高了运行速度

is\_prime[j] = 0; //是i的倍数的均不是素数

}

}

return p;

}

# 欧拉筛 包括Ф

void init() {

phi[1] = 1;

for (int i = 2; i < MAXN; ++i) {

if (!vis[i]) {

phi[i] = i - 1;

pri[cnt++] = i;

}

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

if (1ll \* i \* pri[j] >= MAXN) break;

vis[i \* pri[j]] = 1;

if (i % pri[j]) {

phi[i \* pri[j]] = phi[i] \* (pri[j] - 1);

} else {

// i % pri[j] == 0

// 换言之，i 之前被 pri[j] 筛过了

// 由于 pri 里面质数是从小到大的，所以 i 乘上其他的质数的结果一定会被

// pri[j] 的倍数筛掉，就不需要在这里先筛一次，所以这里直接 break

// 掉就好了

phi[i \* pri[j]] = phi[i] \* pri[j];

break;

}

}

}

}

# 洲阁筛

//#include<iostream>

#include<cstring>

#include<cstdlib>

#include<cstdio>

//#include<map>

#include<math.h>

//#include<time.h>

//#include<complex>

#include<algorithm>

using namespace std;

#define LL long long

int T; LL n;

int m=320000;

#define maxn 640011

LL prime[maxn],f[maxn],sf[maxn],s[maxn],i0[maxn],who[maxn],g[maxn],ff[maxn],mi[maxn],ci[maxn];

int lp,lw; bool notprime[maxn];

#define maxh 1000007

struct Hash

{

int first[maxh],le; struct Edge{LL to,v; int next;}edge[maxn];

void clear(LL n) {le=2; for (LL i=1;i<=n;i=n/(n/i)+1) first[n/i%maxh]=0;}

void insert(LL x,LL y) {int z=x%maxh; Edge &e=edge[le]; e.to=x; e.v=y; e.next=first[z]; first[z]=le++;}

LL find(LL x) {int z=x%maxh; for (int i=first[z];i;i=edge[i].next) if (edge[i].to==x) return edge[i].v; return -1;}

}h;

void pre(int &n)

{

lp=0; f[1]=sf[1]=1;

for (int i=2;i<=n;i++)

{

if (!notprime[i]) {prime[++lp]=i; f[i]=4; mi[i]=i; ci[i]=1;}

sf[i]=sf[i-1]+f[i]; s[i]=s[i-1]+!notprime[i];

for (int j=1,tmp;j<=lp && 1ll\*i\*prime[j]<=n;j++)

{

notprime[tmp=i\*prime[j]]=1;

if (i%prime[j]) {f[tmp]=f[i]\*4; mi[tmp]=prime[j]; ci[tmp]=1;}

else {f[tmp]=f[i/mi[i]]\*(3\*ci[i]+4); mi[tmp]=mi[i]\*prime[j]; ci[tmp]=ci[i]+1; break;}

}

}

lp--; n=prime[lp+1]-1;

}

void mg()

{

lw=0; h.clear(n);

for (LL i=1;i<=n;i=n/(n/i)+1) lw++,who[lw]=g[lw]=n/i,h.insert(who[lw],lw);

memset(i0,0,sizeof(i0));

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

for (int j=1;j<=lw && prime[i]\*prime[i]<=who[j];j++)

{

int k=h.find(who[j]/prime[i]);

g[j]-=g[k]-(i-1-i0[k]);

i0[j]=i;

}

}

void mff()

{

for (int j=1;j<=lw;j++) ff[j]=1;

for (int i=lp;i;i--)

for (int j=1;j<=lw && 1ll\*prime[i]\*prime[i]<=who[j];j++)

{

if (prime[i+1]>who[j]) ff[j]=1;

else if (prime[i+1]\*prime[i+1]>who[j]) ff[j]=(s[min((LL)m,who[j])]-s[prime[i+1]-1])\*4+1;

LL tmp=prime[i],k2;

for (int c=1;tmp<=who[j];tmp\*=prime[i],c++)

{

LL k=who[j]/tmp;

if (prime[i+1]>k) k2=1;

else if (prime[i+1]\*prime[i+1]>k) k2=(s[min((LL)m,k)]-s[prime[i+1]-1])\*4+1;

else k2=ff[h.find(k)];

ff[j]+=k2\*(3\*c+1);

}

}

}

int main()

{

scanf("%d",&T);

pre(m);

while (T--)

{

scanf("%lld",&n);

if (n<=m) {printf("%lld\n",sf[n]); continue;}

LL ans=0,last; mg(); mff();

for (int i=1,id;i<=m;i=last+1)

{

LL tmp=n/i,kk;

if (prime[lp+1]>tmp) kk=1;

else id=h.find(tmp),kk=g[id]-(lp-i0[id]);

ans+=(sf[last=min((LL)m,n/tmp)]-sf[i-1])\*(kk-1)\*4;

}

printf("%lld\n",ans+ff[1]);

}

return 0;

}

# 杜教筛μФ

#include<bits/stdc++.h>

#include<tr1/unordered\_map>

#define N 6000010

using namespace std;

template<typename T>inline void read(T &x)

{

x=0;

static int p;p=1;

static char c;c=getchar();

while(!isdigit(c)){if(c=='-')p=-1;c=getchar();}

while(isdigit(c)) {x=(x<<1)+(x<<3)+(c-48);c=getchar();}

x\*=p;

}

bool vis[N];

int mu[N],sum1[N],phi[N];

long long sum2[N];

int cnt,prim[N];

tr1::unordered\_map<long long,long long>w1;

tr1::unordered\_map<int,int>w;

void get(int maxn)

{

phi[1]=mu[1]=1;

for(int i=2;i<=maxn;i++)

{

if(!vis[i])

{

prim[++cnt]=i;

mu[i]=-1;phi[i]=i-1;

}

for(int j=1;j<=cnt&&prim[j]\*i<=maxn;j++)

{

vis[i\*prim[j]]=1;

if(i%prim[j]==0)

{

phi[i\*prim[j]]=phi[i]\*prim[j];

break;

}

else mu[i\*prim[j]]=-mu[i],phi[i\*prim[j]]=phi[i]\*(prim[j]-1);

}

}

for(int i=1;i<=maxn;i++)sum1[i]=sum1[i-1]+mu[i],sum2[i]=sum2[i-1]+phi[i];

}

int djsmu(int x)

{

if(x<=6000000)return sum1[x];

if(w[x])return w[x];

int ans=1;

for(int l=2,r;l>=0&&l<=x;l=r+1)

{

r=x/(x/l);

ans-=(r-l+1)\*djsmu(x/l);

}

return w[x]=ans;

}

long long djsphi(long long x)

{

if(x<=6000000)return sum2[x];

if(w1[x])return w1[x];

long long ans=x\*(x+1)/2;

for(long long l=2,r;l<=x;l=r+1)

{

r=x/(x/l);

ans-=(r-l+1)\*djsphi(x/l);

}

return w1[x]=ans;

}

int main()

{

int t,n;

read(t);

get(6000000);

while(t--)

{

read(n);

printf("%lld %d\n",djsphi(n),djsmu(n));

}

return 0;

}

# Min-25筛 求素数个数

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

j=n/(n/i);w[++m]=n/i;

if (w[m]<=Sqr) id1[w[m]]=m;

else id2[n/w[m]]=m;

g[m]=(w[m]-1)%mod;

}

for (int j=1;j<=tot;++j)

for (int i=1;i<=m&&pri[j]\*pri[j]<=w[i];++i){

int k=(w[i]/pri[j]<=Sqr)?id1[w[i]/pri[j]]:id2[n/(w[i]/pri[j])];

g[i]=(g[i]-g[k]+j-1)%mod;g[i]=(g[i]+mod)%mod;

}

# 素数个数范围1e11 hdu5901

#include<cstdio>

#include<cmath>

using namespace std;

#define LL long long

const int N = 5e6 + 2;

bool np[N];

int prime[N], pi[N];

int getprime()

{

int cnt = 0;

np[0] = np[1] = true;

pi[0] = pi[1] = 0;

for(int i = 2; i < N; ++i)

{

if(!np[i]) prime[++cnt] = i;

pi[i] = cnt;

for(int j = 1; j <= cnt && i \* prime[j] < N; ++j)

{

np[i \* prime[j]] = true;

if(i % prime[j] == 0) break;

}

}

return cnt;

}

const int M = 7;

const int PM = 2 \* 3 \* 5 \* 7 \* 11 \* 13 \* 17;

int phi[PM + 1][M + 1], sz[M + 1];

void init()

{

getprime();

sz[0] = 1;

for(int i = 0; i <= PM; ++i) phi[i][0] = i;

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

{

sz[i] = prime[i] \* sz[i - 1];

for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];

}

}

int sqrt2(LL x)

{

LL r = (LL)sqrt(x - 0.1);

while(r \* r <= x) ++r;

return int(r - 1);

}

int sqrt3(LL x)

{

LL r = (LL)cbrt(x - 0.1);

while(r \* r \* r <= x) ++r;

return int(r - 1);

}

LL getphi(LL x, int s)

{

if(s == 0) return x;

if(s <= M) return phi[x % sz[s]][s] + (x / sz[s]) \* phi[sz[s]][s];

if(x <= prime[s]\*prime[s]) return pi[x] - s + 1;

if(x <= prime[s]\*prime[s]\*prime[s] && x < N)

{

int s2x = pi[sqrt2(x)];

LL ans = pi[x] - (s2x + s - 2) \* (s2x - s + 1) / 2;

for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];

return ans;

}

return getphi(x, s - 1) - getphi(x / prime[s], s - 1);

}

LL getpi(LL x)

{

if(x < N) return pi[x];

LL ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;

for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans -= getpi(x / prime[i]) - i + 1;

return ans;

}

LL lehmer\_pi(LL x)

{

if(x < N) return pi[x];

int a = (int)lehmer\_pi(sqrt2(sqrt2(x)));

int b = (int)lehmer\_pi(sqrt2(x));

int c = (int)lehmer\_pi(sqrt3(x));

LL sum = getphi(x, a) +(LL)(b + a - 2) \* (b - a + 1) / 2;

for (int i = a + 1; i <= b; i++)

{

LL w = x / prime[i];

sum -= lehmer\_pi(w);

if (i > c) continue;

LL lim = lehmer\_pi(sqrt2(w));

for (int j = i; j <= lim; j++) sum -= lehmer\_pi(w / prime[j]) - (j - 1);

}

return sum;

}

int main()

{

init();

LL n;

while(~scanf("%lld",&n))

{

printf("%lld\n",lehmer\_pi(n));

}

return 0;

}

# Miller-Rabin判断素数

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int prime[10]={2,3,5,7,11,13,17,19,23,29};

int Quick\_Multiply(int a,int b,int c) //快速积（和快速幂差不多）

{

long long ans=0,res=a;

while(b)

{

if(b&1)

ans=(ans+res)%c;

res=(res+res)%c;

b>>=1;

}

return (int)ans;

}

int Quick\_Power(int a,int b,int c) //快速幂，这里就不赘述了

{

int ans=1,res=a;

while(b)

{

if(b&1)

ans=Quick\_Multiply(ans,res,c);

res=Quick\_Multiply(res,res,c);

b>>=1;

}

return ans;

}

bool Miller\_Rabin(int x) //判断素数

{

int i,j,k;

int s=0,t=x-1;

if(x==2) return true; //2是素数

if(x<2||!(x&1)) return false; //如果x是偶数或者是0,1，那它不是素数

while(!(t&1)) //将x分解成(2^s)\*t的样子

{

s++;

t>>=1;

}

for(i=0;i<10&&prime[i]<x;++i) //随便选一个素数进行测试

{

int a=prime[i];

int b=Quick\_Power(a,t,x); //先算出a^t

for(j=1;j<=s;++j) //然后进行s次平方

{

k=Quick\_Multiply(b,b,x); //求b的平方

if(k==1&&b!=1&&b!=x-1) //用二次探测判断

return false;

b=k;

}

if(b!=1) return false; //用费马小定律判断

}

return true; //如果进行多次测试都是对的，那么x就很有可能是素数

}

int main()

{

int x;

scanf("%d",&x);

if(Miller\_Rabin(x)) printf("Yes");

else printf("No");

return 0;

}

# 最长公共前缀长度

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<algorithm>

#include<cmath>

using namespace std;

char sa[1100000],sb[1100000];

int lena,lenb;

int p[1100000],ex[1100000];

//p数组是用来让B串自己匹配自己的

void exkmp()

{

p[1]=lenb;

int x=1;

while(sb[x]==sb[x+1]&&x+1<=lenb) x++;//因为我们p[1]是具有一定性，所以我们不能直接用，所以要先暴力求出p[2]

p[2]=x-1;

int k=2;

for(int i=3;i<=lenb;i++)

{

int pp=k+p[k]-1,L=p[i-k+1];//pp实际上是p

if(i+L<pp+1) p[i]=L;//i-k+L<pp-k+1化简后i+L<pp

else

{

int j=pp-i+1;

if(j<0) j=0;

while(sb[j+1]==sb[i+j]&&i+j<=lenb) j++;

p[i]=j;

k=i;

}

}

x=1;

while(sa[x]==sb[x]&&x<=lenb) x++;//ex[1]并不具有一定性，所以我们暴力求出ex[1]

ex[1]=x-1;

k=1;

for(int i=2;i<=lena;i++)

{

int pp=k+ex[k]-1,L=p[i-k+1];

if(i+L<pp+1) ex[i]=L;

else

{

int j=pp-i+1;

if(j<0) j=0;

while(sb[j+1]==sa[i+j]&&i+j<=lena&&j<=lenb) j++;

ex[i]=j;

k=i;

}

}

}

int main()

{

scanf("%s%s",sa+1,sb+1);

lena=strlen(sa+1);lenb=strlen(sb+1);

exkmp();

for(int i=1;i<lena;i++) printf("%d ",ex[i]);

printf("%d\n",ex[lena]);

return 0;

}

# AC自动机

#include <queue>

#include <cstdlib>

#include <cmath>

#include <cstdio>

#include <string>

#include <cstring>

#include <iostream>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 2\*1e6+9;

int trie[maxn][26]; //字典树

int cntword[maxn]; //记录该单词出现次数

int fail[maxn]; //失败时的回溯指针

int cnt = 0;

void insertWords(string s){

int root = 0;

for(int i=0;i<s.size();i++){

int next = s[i] - 'a';

if(!trie[root][next])

trie[root][next] = ++cnt;

root = trie[root][next];

}

cntword[root]++; //当前节点单词数+1

}

void getFail(){

queue <int>q;

for(int i=0;i<26;i++){ //将第二层所有出现了的字母扔进队列

if(trie[0][i]){

fail[trie[0][i]] = 0;

q.push(trie[0][i]);

}

}

//fail[now] ->当前节点now的失败指针指向的地方

////tire[now][i] -> 下一个字母为i+'a'的节点的下标为tire[now][i]

while(!q.empty()){

int now = q.front();

q.pop();

for(int i=0;i<26;i++){ //查询26个字母

if(trie[now][i]){

//如果有这个子节点为字母i+'a',则

//让这个节点的失败指针指向(((他父亲节点)的失败指针所指向的那个节点)的下一个节点)

//有点绕,为了方便理解特意加了括号

fail[trie[now][i]] = trie[fail[now]][i];

q.push(trie[now][i]);

}

else//否则就让当前节点的这个子节点

//指向当前节点fail指针的这个子节点

trie[now][i] = trie[fail[now]][i];

}

}

}

int query(string s){

int now = 0,ans = 0;

for(int i=0;i<s.size();i++){ //遍历文本串

now = trie[now][s[i]-'a']; //从s[i]点开始寻找

for(int j=now;j && cntword[j]!=-1;j=fail[j]){

//一直向下寻找,直到匹配失败(失败指针指向根或者当前节点已找过).

ans += cntword[j];

cntword[j] = -1; //将遍历国后的节点标记,防止重复计算

}

}

return ans;

}

int main() {

int n;

string s;

cin >> n;

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

cin >> s ;

insertWords(s);

}

fail[0] = 0;

getFail();

cin >> s ;

cout << query(s) << endl;

return 0;

}

# Manacher马拉车 最长回文子串O(n)

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

char s[11000002];

char s\_new[21000002];//存添加字符后的字符串

int p[21000002];

int Init() {//形成新的字符串

int len=strlen(s);//len是输入字符串的长度

s\_new[0]='$';//处理边界，防止越界

s\_new[1]='#';

int j=2;

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

s\_new[j++]=s[i];

s\_new[j++]='#';

}

s\_new[j]='\0';//处理边界，防止越界（容易忘记）

return j;// 返回s\_new的长度

}

int Manacher() {//返回最长回文串

int len=Init();//取得新字符串的长度， 完成向s\_new的转换

int max\_len=-1;//最长回文长度

int id;

int mx=0;

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

if(i<mx)

p[i]=min(p[2\*id-i],mx-i);//上面图片就是这里的讲解

else p[i]=1;

while(s\_new[i-p[i]]==s\_new[i+p[i]])//不需边界判断，因为左有'$'，右有'\0'标记；

p[i]++;//mx对此回文中点的贡献已经结束，现在是正常寻找扩大半径

if(mx<i+p[i]) {//每走移动一个回文中点，都要和mx比较，使mx是最大，提高p[i]=min(p[2\*id-i],mx-i)效率

id=i;//更新id

mx=i+p[i];//更新mx

}

max\_len=max(max\_len,p[i]-1);

}

return max\_len;

}

int main()

{

scanf("%s",&s);

printf("%d",Manacher());

return 0;

}

# 后缀数组luogu3809

读入一个长度为 n 的由大小写英文字母或数字组成的字符串，把这个字符串的所有非空后缀按字典序从小到大排序，然后按顺序输出后缀的第一个字符在原串中的位置(1-n)。

#include<iostream>

#include<cstdio>

#include<cstring>

#define rint register int

#define inv inline void

#define ini inline int

#define maxn 1000050

using namespace std;

char s[maxn];

int y[maxn],x[maxn],c[maxn],sa[maxn],rk[maxn],height[maxn],wt[30];

int n,m;

inv putout(int x) {

if(!x) {

putchar(48);

return;

}

rint l=0;

while(x) wt[++l]=x%10,x/=10;

while(l) putchar(wt[l--]+48);

}

inv get\_SA() {

for (rint i=1; i<=n; ++i) ++c[x[i]=s[i]];

//c数组是桶

//x[i]是第i个元素的第一关键字

for (rint i=2; i<=m; ++i) c[i]+=c[i-1];

//做c的前缀和，我们就可以得出每个关键字最多是在第几名

for (rint i=n; i>=1; --i) sa[c[x[i]]--]=i;

for (rint k=1; k<=n; k<<=1) {

rint num=0;

for (rint i=n-k+1; i<=n; ++i) y[++num]=i;

//y[i]表示第二关键字排名为i的数，第一关键字的位置

//第n-k+1到第n位是没有第二关键字的 所以排名在最前面

for (rint i=1; i<=n; ++i) if (sa[i]>k) y[++num]=sa[i]-k;

//排名为i的数 在数组中是否在第k位以后

//如果满足(sa[i]>k) 那么它可以作为别人的第二关键字，就把它的第一关键字的位置添加进y就行了

//所以i枚举的是第二关键字的排名，第二关键字靠前的先入队

for (rint i=1; i<=m; ++i) c[i]=0;

//初始化c桶

for (rint i=1; i<=n; ++i) ++c[x[i]];

//因为上一次循环已经算出了这次的第一关键字 所以直接加就行了

for (rint i=2; i<=m; ++i) c[i]+=c[i-1]; //第一关键字排名为1~i的数有多少个

for (rint i=n; i>=1; --i) sa[c[x[y[i]]]--]=y[i],y[i]=0;

//因为y的顺序是按照第二关键字的顺序来排的

//第二关键字靠后的，在同一个第一关键字桶中排名越靠后

//基数排序

swap(x,y);

//这里不用想太多，因为要生成新的x时要用到旧的，就把旧的复制下来，没别的意思

x[sa[1]]=1;

num=1;

for (rint i=2; i<=n; ++i)

x[sa[i]]=(y[sa[i]]==y[sa[i-1]] && y[sa[i]+k]==y[sa[i-1]+k]) ? num : ++num;

//因为sa[i]已经排好序了，所以可以按排名枚举，生成下一次的第一关键字

if (num==n) break;

m=num;

//这里就不用那个122了，因为都有新的编号了

}

for (rint i=1; i<=n; ++i) putout(sa[i]),putchar(' ');

}

inv get\_height() {

rint k=0;

for (rint i=1; i<=n; ++i) rk[sa[i]]=i;

for (rint i=1; i<=n; ++i) {

if (rk[i]==1) continue;//第一名height为0

if (k) --k;//h[i]>=h[i-1]-1;

rint j=sa[rk[i]-1];

while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;

height[rk[i]]=k;//h[i]=height[rk[i]];

}

putchar(10);

for (rint i=1; i<=n; ++i) putout(height[i]),putchar(' ');

}

int main() {

gets(s+1);

n=strlen(s+1);

m=122;

//因为这个题不读入n和m所以要自己设

//n表示原字符串长度，m表示字符个数，ascll('z')=122

//我们第一次读入字符直接不用转化，按原来的ascll码来就可以了

//因为转化数字和大小写字母还得分类讨论，怪麻烦的

get\_SA();

//get\_height();

}

# 后缀自动机

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

struct lxy{

int to[26],p,len,k,num;

}a[400005];

int cnt=1,las=1,len;

int tax[200005];

int tp[400005];

char s[200005];

long long ans;

void insert(int c,int w){

a[++cnt].len=w;a[cnt].num=1;

int i;for(i=las;a[i].to[c]==0&&i!=0;i=a[i].p) a[i].to[c]=cnt;

las=cnt;

if(i==0){

a[cnt].p=1;return;

}

int q=a[i].to[c],nq;

if(a[i].len+1==a[q].len){

a[cnt].p=q;return;

}

nq=cnt+1;for(int j=i;a[j].to[c]==q;j=a[j].p) a[j].to[c]=nq;

a[nq]=a[q];a[nq].num=0;a[nq].len=a[i].len+1;

a[q].p=nq;a[cnt].p=nq;las=cnt;cnt++;

}

void querytp(){

for(int i=1;i<=cnt;i++) tax[a[i].len]++;

for(int i=1;i<=len;i++) tax[i]+=tax[i-1];

for(int i=1;i<=cnt;i++) tp[tax[a[i].len]--]=i;

}

void matchit(int u,int pos,int l){

a[u].k++;ans-=1ll\*a[u].num\*(a[u].len-l);

if(s[pos]==0) return;

for(;a[u].to[s[pos]-'a']==0&&u!=0;u=a[u].p);

if(u==0) matchit(1,pos+1,0);

else matchit(a[u].to[s[pos]-'a'],pos+1,min(l,a[u].len)+1);

}

int main()

{

scanf("%s",s+1);len=strlen(s+1);

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

insert(s[i]-'a',i);

querytp();

for(int i=cnt;i>=1;i--) a[a[tp[i]].p].num+=a[tp[i]].num;

scanf("%s",s+1);

matchit(1,1,0);

for(int i=cnt;i>=1;i--) a[a[tp[i]].p].k+=a[tp[i]].k,ans+=1ll\*a[tp[i]].k\*a[tp[i]].num\*(a[tp[i]].len-a[a[tp[i]].p].len);

printf("%lld",ans);

}

# 回文自动机luogu1659

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

typedef long long ll;

const int maxn=1e6+100,P=19930726;

struct node{

int len,fail,son[26],siz;

node(){

len=fail=0;

for(int i=0;i<=25;i++)

son[i]=0;

}

};

node prt[maxn];

int n,last,len,num;

ll ans=1,k;

char s[maxn];

ll poww(ll x,int y){

ll base=1;

while(y){

if(y&1) base\*=x,base%=P;

x\*=x,x%=P;

y>>=1;

}

return base;

}

bool cmp(node x,node y){

return x.len>y.len;

}

int getfail(int x){

while(s[n-prt[x].len-1]!=s[n]) x=prt[x].fail;

return x;

}

void extend(int x){

int cur=getfail(last);

if(!prt[cur].son[x]){

int now=++num;

prt[now].len=prt[cur].len+2;

prt[now].fail=prt[getfail(prt[cur].fail)].son[x];

prt[cur].son[x]=now;

}

prt[prt[cur].son[x]].siz++;

last=prt[cur].son[x];

}

int main(){

scanf("%d%d",&len,&k);

scanf("%s",s);

last=num=1,prt[1].len=-1;

prt[0].fail=prt[1].fail=1;

for(n=0;n<len;n++) extend(s[n]-'a');

for(int i=num;i>=2;i--)

prt[prt[i].fail].siz+=prt[i].siz,prt[prt[i].fail].siz%=P;

sort(prt+1,prt+num+1,cmp);

int now=1;

while(k){

if(now>num){

printf("-1\n");

return 0;

}

if(prt[now].len%2==0){

now++;

continue;

}

if(prt[now].siz<k){

k-=prt[now].siz;

ans\*=poww(prt[now].len,prt[now].siz)%P;

ans%=P;

now++;

}

else{

ans\*=poww(prt[now].len,k)%P;

ans%=P;

k=0;

}

}

printf("%lld\n",ans);

}

# SPFA

#include <bits/stdc++.h>

#define inf 0x3f3f3f3f

using namespace std;

const int M=10005;

struct A{

int y,time,next;

}a[M<<1];

int pre[M],cent=0;//链式前向星数组

int vis[M],ven[M],nums[M];

//SPFS数组，vis记录最短路，ven记录是否在队列，nums记录入队次数

void add(int x,int y,int k)//链式前向星，加入节点

{

a[cent].y=y, a[cent].time=k, a[cent].next=pre[x];

pre[x]=cent++;

}

bool SPFA(int s,int n)

{

queue <int> q;

memset(vis,inf,sizeof(vis));

memset(ven,0,sizeof(ven));

memset(nums,0,sizeof(nums));

vis[s]=0;//初始化距离

ven[s]=1,nums[s]++;//标记s节点在队列，队列次数+1

q.push(s);

while(!q.empty())

{

int x=q.front();

q.pop();//出队

ven[x]=0;//标记不在队列

for(int i=pre[x]; ~i; i=a[i].next)//遍历与x节点连通的点

{

int y=a[i].y;

if(vis[y]>vis[x]+a[i].time)//更新

{

vis[y]=vis[x]+a[i].time;

if(!ven[y])

//由于更新了节点，所以后续以这个为基础的最短路，也要更新下

//所以如果在队列就不用加入，不在的话加入更新后续节点

{

q.push(y);

ven[y]=1;//标记这个节点在队列中

nums[y]++;//记录加入次数

if(nums[y]>n)//如果这个点加入超过n次，说明存在负圈，直接返回

return false;

}

}

}

}

return true;

}

int main()

{

int n,m,t;

int b[M],c[M];

while(cin>>n>>m>>t)

{

cent=0;

memset(pre,-1,sizeof(pre));

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

{

int x,y,k;

cin>>x>>y>>k;

add(x,y,k);

add(y,x,k);

}

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

{

cin>>b[i];

}

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

{

cin>>c[i];

}

int minn=inf;

for(int i=0;i<m;i++)//遍历多个找寻最小

{

SPFA(b[i],n);

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

minn=min(minn,vis[c[j]]);

}

cout<<minn<<endl;

}

}

# Dijkstra

typedef struct \_graph

{

char vexs[MAX]; // 顶点集合

int vexnum; // 顶点数

int edgnum; // 边数

int matrix[MAX][MAX]; // 邻接矩阵

}Graph, \*PGraph;

// 边的结构体

typedef struct \_EdgeData

{

char start; // 边的起点

char end; // 边的终点

int weight; // 边的权重

}EData;

/\*

\* Dijkstra最短路径。

\* 即，统计图(G)中"顶点vs"到其它各个顶点的最短路径。

\*

\* 参数说明：

\* G -- 图

\* vs -- 起始顶点(start vertex)。即计算"顶点vs"到其它顶点的最短路径。

\* prev -- 前驱顶点数组。即，prev[i]的值是"顶点vs"到"顶点i"的最短路径所经历的全部顶点中，位于"顶点i"之前的那个顶点。

\* dist -- 长度数组。即，dist[i]是"顶点vs"到"顶点i"的最短路径的长度。

\*/

void dijkstra(Graph G, int vs, int prev[], int dist[])

{

int i,j,k;

int min;

int tmp;

int flag[MAX]; // flag[i]=1表示"顶点vs"到"顶点i"的最短路径已成功获取。

// 初始化

for (i = 0; i < G.vexnum; i++)

{

flag[i] = 0; // 顶点i的最短路径还没获取到。

prev[i] = 0; // 顶点i的前驱顶点为0。

dist[i] = G.matrix[vs][i];// 顶点i的最短路径为"顶点vs"到"顶点i"的权。

}

// 对"顶点vs"自身进行初始化

flag[vs] = 1;

dist[vs] = 0;

// 遍历G.vexnum-1次；每次找出一个顶点的最短路径。

for (i = 1; i < G.vexnum; i++)

{

// 寻找当前最小的路径；

// 即，在未获取最短路径的顶点中，找到离vs最近的顶点(k)。

min = INF;

for (j = 0; j < G.vexnum; j++)

{

if (flag[j]==0 && dist[j]<min)

{

min = dist[j];

k = j;

}

}

// 标记"顶点k"为已经获取到最短路径

flag[k] = 1;

// 修正当前最短路径和前驱顶点

// 即，当已经"顶点k的最短路径"之后，更新"未获取最短路径的顶点的最短路径和前驱顶点"。

for (j = 0; j < G.vexnum; j++)

{

tmp = (G.matrix[k][j]==INF ? INF : (min + G.matrix[k][j])); // 防止溢出

if (flag[j] == 0 && (tmp < dist[j]) )

{

dist[j] = tmp;

prev[j] = k;

}

}

}

// 打印dijkstra最短路径的结果

printf("dijkstra(%c): \n", G.vexs[vs]);

for (i = 0; i < G.vexnum; i++)

printf(" shortest(%c, %c)=%d\n", G.vexs[vs], G.vexs[i], dist[i]);

}

# Dijkstra堆优化

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<int, int> pii;

const int N = 1e5 + 10;

const ll inf = (ll)1e16;

vector <pii> V[N];

int n, m;

bool vis[N];

ll dis[N];

struct Node{

int id;

ll d;

Node(){}

Node(int id, ll d):id(id),d(d){}

bool operator < (const Node &A)const{

return d > A.d;

}

};

void dijkstra(int st){

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

vis[i] = 0;

dis[i] = inf;

}

dis[st] = 0;

priority\_queue<Node> Q;

Q.push(Node(st, 0));

Node nd;

while(!Q.empty()){

nd = Q.top(); Q.pop();

if(vis[nd.id]) continue;

vis[nd.id] = true;

for(int i=0; i<V[nd.id].size(); i++){

int j = V[nd.id][i].first;

int k = V[nd.id][i].second;

if(nd.d + k < dis[j] && !vis[j]){

dis[j] = nd.d + k;

Q.push(Node(j, dis[j]));

}

}

}

}

int main(){

int x, y, z, st, ed, cas = 0;

scanf("%d", &cas);

while(cas--){

scanf("%d%d%d", &n, &m, &st);

for(int i=1; i<=n; i++) V[i].clear();

while(m--){

scanf("%d%d%d", &x, &y, &z);

V[x].push\_back(make\_pair(y, z));

//V[y].push\_back(make\_pair(x, z));

}

dijkstra(st);

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

if(i==1) printf("%d", dis[i]);

else printf(" %d", dis[i]);

}

}

# Dijkstra配对堆（？？？）

#include<bits/stdc++.h>

#include<ext/pb\_ds/priority\_queue.hpp>

using namespace std;

using namespace \_\_gnu\_pbds;

typedef long long ll;

const int mn = 100005;

const int maxn = 200005 ;

const int inf = 2147483647;

typedef \_\_gnu\_pbds::priority\_queue< pair<int,int> ,\

greater< pair<int,int> >,pairing\_heap\_tag > heap;

heap::point\_iterator id[mn];//记录下每个点的迭代器

heap q;

inline int read(){

int x=0;

char ch=getchar();

while(ch>'9' || ch<'0') ch=getchar();

while(ch>='0' && ch<='9') x=x\*10+ch-'0',ch=getchar();

return x;

}

struct edge{int to, next, dis;};

edge e[maxn \* 2];

int head[mn], edge\_max;

int n, m, st, dis[mn];

void add(int x, int y, int z){

e[++edge\_max].to=y;

e[edge\_max].dis=z;

e[edge\_max].next=head[x];

head[x]=edge\_max;

}

void dij(int x){

for(int i=1; i<=n; i++) dis[i] = inf;

dis[x]=0;

id[x] = q.push(make\_pair(0, x));//每次push会返回新加入点的迭代器

while(!q.empty()){

int now = q.top().second;

q.pop();

for(int i=head[now]; i; i=e[i].next){

if(e[i].dis+dis[now] < dis[e[i].to]){

dis[e[i].to] = dis[now]+e[i].dis;

if(id[e[i].to]!=0) //如果在堆中

q.modify(id[e[i].to], make\_pair(dis[e[i].to], e[i].to));//修改权值

else id[e[i].to] = q.push(make\_pair(dis[e[i].to], e[i].to));//加入堆

}

}

}

}

int main(){

int x, y, z;

n=read(), m=read(), st=read();

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

x=read(), y=read(), z=read();

add(x, y, z);

}

dij(st);

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

if(i==1) printf("%d", dis[i]);

else printf(" %d", dis[i]);

}

# Bellman-Ford

struct Node{

int v,dis;

};

vector<Node> Adj[MAXV];

int n;

int d[MAXV];

bool Bellman(int s){

fill(d,d+MAXV,INF);

d[s]=0;

for(int i=0;i<n-1;i++){

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

for(int j=0;j<Adj[u].size();j++){

int v=Adj[u][j].v;

int dis=Adj[u][j].dis;

if(d[u]+dis<d[v]){

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

}

}

}

}

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

for(int j=0;j<Adj[u].size();j++){

int v=Adj[u][j].v;

int dis=Adj[u][j].dis;

if(d[u]+dis<d[v]){

return false;

}

}

}

return true;

}