|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **//string matching**  int pi[200001]; string txt,pat;  void solve()  {  int i,j,res=0;  cin>>pat>>txt;  string st=pat+"#"+txt;  for(i=1;i<st.size();i++)  {  j=pi[i-1];  while(j>0 && st[i]!=st[j]) j=pi[j-1];  if(st[i]==st[j]) j++;  pi[i]=j;  }  for(i=pat.size();i<st.size();i++)  {  if(pi[i]==pat.size())  {  res++;  if(res>0) break;  }  }  }  **//LPS­\_ARRAY**  vector<int> lps;  void createLPS(string pat){  lps.push\_back(0);  int i = 0, j = 1;  for ( ; pat[j]; ){  if (pat[i] == pat[j]){  lps.push\_back(i+1);  i++;j++;  }  else{  if (i != 0) i = lps[i-1];  else {  j++;  lps.push\_back(0);  }  }  }  } | **//KMP**  char txt[1000009],pat[1000009];  void lps\_ar(char \*pat,int M,int \*lps)  {  int len=0; lps[0]=0; int i=1;  while(i<M)  {  if(pat[i]==pat[len]) len++,lps[i]=len,i++;  else  {  if(len!=0) len=lps[len-1];  else lps[i]=0,i++;  }  }  }  void KMPsearch(char \*txt,char \*pat)  {  int N=strlen(txt);  int M=strlen(pat);  int lps[M];  lps\_ar(pat,M,lps);  int i=0,j=0; total=0;  while(i<N)  {  //cout<<"kmp"<<endl;  if(pat[j]==txt[i]) i++,j++;  if(j==M)  {  //to print how many times match  total++;  //cout<<"found pattern at index: "<<i-j<<endl;  j=lps[j-1];  }  else if(i<N && pat[j]!=txt[i])  {  if(j!=0) j=lps[j-1];  else i++;  }  }  } | | | **//STRING MULTIPLACATION**  string multyply(string a, int b){  int carry = 0;  ans = "";  for(int i = 0; i < a.size(); i++){  carry = ((a[i] - '0') \* b + carry);  ans += carry % 10 + '0';  carry /= 10;  }  while(carry != 0){  ans += carry % 10 + '0';  carry /= 10;  }  return ans;  } |
| **//find nCr**  ll nCr(ll n,ll r)  {  ll p=1,q=1;  r=min(r,n-r);  if(r!=0)  {  while(r)  {  p\*=n;q\*=r;  ll x=\_\_gcd(p,q);  p/=x;q/=x;  n--;r--;  }  }  else p=1;  return p;  }  ///**print power**  ll power(ll x,ll n)  {  ll res=1;  while(n)  {  if(n&1) res\*=x;  x\*=x;  n>>=1;  }  return res;  }  ///**print power\_mod**  ll power\_mod(ll a,ll b)  {  ll res=1;  while(b)  {  if(b&1) res=(res\*1LL\*a)%MOD;  a=(a\*1LL\*a)%MOD;  b>>1;  }  return res;  }  //**GCD**  ll gcd(ll a,ll b){  if(b==0 || a==0) return 0;  if(b%a==0) return a;  else return gcd(b%a,a);  } | |  | **//EULET TOTIENT ( 1 to N) (time: nlog(n))**  void EulerTotient()  {  phi[1] = 1;  for (int i=2; i<MAX; i++)  {  if (!phi[i])  {  phi[i] = i-1;  for (int j = i\*2;j<MAX; j+=i)  {  if (!phi[ j]) phi[j] = j;  phi[ j] = (phi [ j]/i)\*(i-1);  } } } }  // **EULET TOTIENT (time:sqrt(n) \* log(n))**  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;  } | | |