|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***Aho Korasik***  #include<bits/stdc++.h>  using namespace std ;  char T[1000000+1] ;  char keyword[500] ;  char tmp[1000000+1] ;  int n ;  int freq[1000000+1] ;  int ans[1000000] ;  struct Trie {  int level,\_next[26] ; vector<int>patIdx ;  int parent ;  Trie() {  level = 0 ;  memset( \_next , 0 , sizeof \_next ) ;  patIdx.clear() ; parent = 0 ;  }  };  const int MAXC = 500\*500+5 ;  Trie Tree[MAXC] ; int treeIdx ;  void InsertTrie(char keyword[],int idx) {  int root = 0 ;  for( int i = 0 ; keyword[i] ; i++ ) {  char ch = keyword[i]-'a' ;  if( Tree[root].\_next[ch] == 0 ) {  Tree[root].\_next[ch] = treeIdx++ ;  }  Tree[Tree[root].\_next[ch]].level = Tree[root].level+1 ;  root = Tree[root].\_next[ch] ;  }  Tree[root].patIdx.push\_back(idx);  }  int FindParent( int src,int ch ) {  int par = Tree[src].parent ;  while( par > 0 && Tree[par].\_next[ch] == 0 ) {  par = Tree[par].parent ;  }  return par ;  }  void bfsOnTrie() {  queue<int>Q ;  for( int i =0 ; i < 26 ; i++ ) {  if( Tree[0].\_next[i] != 0 ) {  Q.push( Tree[0].\_next[i] ) ;  }  }  while( !Q.empty() ) {  int src = Q.front() ;  Q.pop();  for( int i = 0 ; i < 26 ; i++ ) {  if( Tree[src].\_next[i] != 0 ) {  int dest = Tree[src].\_next[i] ;  int par = FindParent(src,i) ;  Tree[dest].parent = Tree[par].\_next[i] ;  Q.push(dest) ;  }  }  }  }  bool comp( pair<int,int> a , pair<int,int> b ){  return a.first > b.first ;  }  void query() {  int root = 0 ;  memset( freq , 0 , sizeof freq ) ;  for( int i = 0 ; T[i] ; i++ ) {  int ch = T[i] - 'a' , par;  if( Tree[root].\_next[ch] == 0 ) {  int par = FindParent(root,ch) ;  root = Tree[par].\_next[ch] ;  } else {  root = Tree[root].\_next[ch] ;  }  freq[root]++ ;  } | | vector<pair<int,int> > tmp ;  for( int i = 0 ; i < treeIdx ; i++ ){  tmp.push\_back( make\_pair(Tree[i].level,i) ) ;  }  sort( tmp.begin() , tmp.end() , comp );  for( int i = 0 ; i < treeIdx ; i++ ){  freq[ Tree[tmp[i].second].parent ]+=freq[tmp[i].second] ;  }  for( int i = 1 ; i < treeIdx ; i++ ){  for( int j = 0 ; j < Tree[i].patIdx.size() ; j++ ){  ans[Tree[i].patIdx[j]] = freq[i] ;  }  }  }  int main() {  int cases,caseno=1 ;  scanf("%d",&cases ) ;  while( cases -- ) {  scanf("%d%s",&n,T) ;  treeIdx = 1 ;  for( int i = 0 ; i < n ; i++ ) {  scanf("%s",keyword) ;  InsertTrie(keyword,i) ;  }  bfsOnTrie() ;  query();  cout << "Case " << caseno++ << ":\n";  for(int i = 0 ; i < n ; i++ ){  cout << ans[i] << "\n" ;  }  for( int i = 0 ; i < MAXC ; i++ ) Tree[i] = Trie() ;  }  return 0 ;  }   |  |  | | --- | --- | | 2  5  ababacbabc  aba  ba  ac  a  abc  3  lightoj  oj  light  lit | Case 1:  2  3  1  4  1  Case 2:  1  1  0 |   ***BIT***  void update(int x, int y, int v) {  int y1;  while(x <= MAX) {  y1 = y;  while(y1 <= MAX) {  bit[x][y1] += v;  y1 += (y1 & -y1);  }  x += (x & -x);  }  }  int readsum(int x, int y) {  int v = 0, y1;  while(x > 0) {  y1 = y;  while(y1 > 0) {  v += bit[x][y1];  y1 -= (y1 & -y1);  }  x -= (x & -x);  }  return v;  }   Update(a, v) and Update(b+1, -v) | | To add v in range [a, b]: Update(a, v), Update(b+1, -v) in the first BIT and Update(a, v\*(a-1)) and Update(b+1, -v\*b) on the second BIT. To get sum in range [0, x]: you simply do Query\_BIT1(x)\*x - Query\_BIT2(x); Now you know how to find range sum for [a, b]. Just find sum(b) - sum(a-1) using the formula stated above.  ***CRT***  long long Extendex\_GCD( long long a , long long b ) {  long long x = 0 , y = 1 , g = b ;  long long m, n, q, r;  for (long long u=1, v=0; a != 0; g=a, a=r) {  q = g / a;r = g % a;m = x-u\*q;n = y-v\*q;x=u;  y=v; u=m;v=n;  }return y;  }  long long CRT(pair<long long int,long long int> arr[] , int n ) {  long long int N = 1 ;  for( int i = 0 ; i < n ; i++ ) {  N\*=arr[i].first ;  }  long long ans = 0 ;  for( int i = 0 ; i < n ; i++ ) {  long long b = Extendex\_GCD( arr[i].first , N/arr[i].first );  ans+=arr[i].second\*b\*(N/arr[i].first) ;  ans%=N ;  }  if( ans < 0 )ans += N ;  return ans ;  }printf("Case %d: %lld\n",caseno++,CRT(arr , n ));  Given a set of simultaneous congruences  x = a1 (mod n1)  x = a2 (mod n2)  …………….  …………….  x = ai (mod ni)  ***Disjoint Set***  struct DisjointSet {  int \*root, \*rank, n;  DisjointSet(int sz) {  root = new int[sz+1];  rank = new int[sz+1];  n = sz;  }  ~DisjointSet() {  delete[] root;  delete[] rank;  }  void init() {  for(int i = 1; i <= n; i++) {  root[i] = i;  rank[i] = 0;  }  }  int find(int u) {  if(u != root[u]) root[u] = find(root[u]);  return root[u];  }  void merge(int u, int v) {  int pu = find(u);  int pv = find(v);  if(rank[pu] > rank[pv]) root[pv] = pu;  else root[pu] = pv;  if(rank[pu]==rank[pv]) rank[pv]++;  }  }; | |
| ***Brige***  #include <cstdio>  #include <cstring>  #include <algorithm>  using namespace std;  const int MAXN = 10000;  const int MAXE = 40000;  int from[MAXE], to[MAXE], next[MAXE], fin[MAXN], bridge[MAXE];  int low[MAXN], vis[MAXN], used[MAXN], dfsTime, nEdge;  int compo[MAXN], deg[MAXN], nComp;  void addEdge(int u, int v) {  from[nEdge] = u, to[nEdge] = v, bridge[nEdge] = 0, next[nEdge] = fin[u], fin[u] = nEdge++;  from[nEdge] = v, to[nEdge] = u, bridge[nEdge] = 0, next[nEdge] = fin[v], fin[v] = nEdge++;  }  void findBridge(int u, int par) {  int i, v;  used[u] = 1;  vis[u] = low[u] = ++dfsTime;  for(i = fin[u]; i >= 0; i = next[i]) {  v = to[i];  if(v == par) continue;  if(used[v]) low[u] = min(low[u], vis[v]);  else {  findBridge(v, u);  low[u] = min(low[u], low[v]);  if(low[v] > vis[u]) bridge[i] = bridge[i^1] = 1;  }  }  }  int main() {  int test, cs, n, m, i, u, v;  scanf("%d", &test);  for(cs = 1; cs <= test; cs++) {  scanf("%d %d", &n, &m);  memset(fin, -1, sizeof(int)\*n);  memset(used, 0, sizeof(int)\*n);  for(i = nEdge = 0; i < m; i++) {  scanf("%d %d", &u, &v);  addEdge(u, v);  }  dfsTime = nComp = 0;  findBridge(0, -1);  return 0;  }  /\*  bridge separates two connected component.  in each component there is at least 2 path from a node to another node.  non-bicolorable connected graph has a cycle of odd length.  \*/  ***Articulation Point***  vector< int > G[MAX];  int low[MAX], vis[MAX], used[MAX], cut[MAX], dfstime;  void dfs(int u, int par = -1) {  int i, v, child = 0;  used[u] = 1;  vis[u] = low[u] = ++dfstime;  for(i = 0; i < G[u].size(); i++) {  v = G[u][i];  if(v == par) continue;  if(used[v]) low[u] = min(low[u], vis[v]);  else {  child++;  dfs(v, u);  low[u] = min(low[u], low[v]);  if(low[v] >= vis[u]) cut[u] = 1;  }  }  if(par == -1) cut[u] = (child > 1); | | ***Bi connected component***  /\*  G[][]: undirected graph  Separates bi-connected component by edges.  \*/  vector< int > G[MAX];  stack< pii > S;  int dfstime;  int low[MAX], vis[MAX], used[MAX];  void dfs(int u, int par) {  int v, i, sz = G[u].size();  pii e, curr;  used[u] = 1;  vis[u] = low[u] = ++dfstime;  for(i = 0; i < sz; i++) {  v = G[u][i];  if(v == par) continue;  if(!used[v]) {  S.push(pii(u, v));  dfs(v, u);  if(low[v] >= vis[u]) {  // new component  curr = pii(u, v);  do {  e = S.top();  S.pop();  // e is an edge in current bcc  } while(e != curr);  }  low[u] = min(low[u], low[v]);  } else if(vis[v] < vis[u]) {  S.push(pii(u, v));  low[u] = min(low[u], vis[v]);  }  }  }  ***Bellman Ford***  D[0] = 0 ;  bool isCycle = 0 ;  for( int i = 0 ; i < N ; i++ ) {  bool brk = 0 ;  for( int j = 0 ; j < R ; j++ ) {  int u = U[j] ;  int v = V[j] ;  if( D[u] + C[j] < D[v] ) {  brk = 1 ;  if( i == (N-1) )isCycle =1 ;  else D[v] = D[u] + C[j] ;  }  }  if( !brk ) break ;  }  ***Josephous***  /\*  The first one is for K = 2 and the second one is general.  Note: first function returns 1 based index while second one is 0 based.  \*/  int f(int n) {  if(n == 1) return 1;  return (f((n-(n&1))>>1)<<1) + ((n&1)?1:-1);  }  int f(int n, int k) {  if(n == 1) return 0;  return (f(n-1, k) + k)%n;  } | | ***Closest pair***  const int N = int(1e5) + 10;  typedef long long int64;  const int64 INF = 1LL << 60;  struct Point {  int64 x;  int64 y;  } point[N];  int n;  int tmpt[N];  bool cmpxy(const Point& a, const Point& b) {  if (a.x != b.x)return a.x < b.x;  return a.y < b.y;  }  bool cmpy(const int& a, const int& b) {  return point[a].y < point[b].y;  }  int64 dis2(int i, int j) {  return (point[i].x - point[j].x) \* (point[i].x - point[j].x)  + (point[i].y - point[j].y) \* (point[i].y - point[j].y);  }  int64 sqr(int64 x) {return x \* x;}  int64 Closest\_Pair(int left, int right) {  int64 d = INF;  if (left == right)  return d;  if (left + 1 == right)  return dis2(left, right);  int mid = (left + right) >> 1;  int64 d1 = Closest\_Pair(left, mid);  int64 d2 = Closest\_Pair(mid + 1, right);  d = min(d1, d2);  int i, j, k = 0;  for (i = left; i <= right; i++) {  if (sqr(point[mid].x - point[i].x) <= d)  tmpt[k++] = i;  }  sort(tmpt, tmpt + k, cmpy);  for (i = 0; i < k; i++) {  for (j = i + 1; j < k && sqr(point[tmpt[j]].y - point[tmpt[i]].y) < d;  j++) {  int64 d3 = dis2(tmpt[i], tmpt[j]);  if (d > d3)  d = d3;  }  }  return d;  }  int main() {  int n ;cin >> n ;int64 S[n+1] ;  S[0] = 0 ;  for( int i = 1 ; i <= n ; i++ ) {  int x , y ;  cin >> x >> y ;  point[i].x = x ;  point[i].y = y ;  }  cout << Closest\_Pair(1,n) << "\n" ;  return 0 ;  }  ***HASH:***  unsigned Hash( vector<pair<int,int> > V ) {  unsigned h = 2166136261;  for( int i = 0 ; i < V.size() ; i++ ) {  h = (h \* 16777619) ^ (V[i].first);  h = (h \* 16777619) ^ (V[i].second);  }  return h;  } | |
| ***Convex Hull***  /\*  1. Assign hull.n  2. Take input in hull.point  3. Call convexSort()  4. Call findHull()  5. Convex Hull is ready in hull.convex with hull.m points in it.  \*/  pll g;  vlong triArea ( pll a, pll b, pll c ) {  vlong area = a.ff \* b.ss + b.ff \* c.ss + c.ff \* a.ss;  area -= a.ff \* c.ss + b.ff \* a.ss + c.ff \* b.ss;  return area;  }  vlong sqDist ( pll a, pll b ) {  return ( SQ(a.ff-b.ff) + SQ(a.ss-b.ss ) );  }  bool convexCompare ( const pll &a, const pll &b ) {  vlong area = triArea ( g, a, b );  if ( area > 0 ) return true;  else if ( area == 0 && sqDist ( g, a ) < sqDist ( g, b ) ) return true;  else return false;  }  struct ConvexHull {  int n, m;  pll point[PPP], convex[PPP];  void convexSort() {  g = point[0];  FOR(i,0,n-1) {  if ( point[i].ff < g.ff ) g = point[i];  else if ( point[i].ff == g.ff && point[i].ss < g.ss ) g = point[i];  }  sort ( point, point + n, convexCompare );  }  void findHull() {  if ( n == 1 ) {  convex[0] = convex[1] = point[0];  m = 1;  return;  }  convex[0] = point[n-1];  convex[1] = point[0];  convex[2] = point[1];  int cur = 3;  for ( int i = 2; i < n; i++ ) {  vlong area = triArea ( convex[cur-2], convex[cur-1], point[i] );  if ( area > 0 ) {  convex[cur] = point[i];  cur++;  } else if ( area == 0 ) { ///Take action depending on what is required  /\*Left Vertical Line gets omitted. Manually handle it\*/  /\*convex[cur] = point[i];  cur++;\*/  ///If extra point needs to be removed  convex[cur-1] = point[i];  } else {  cur--;  i--;  }  }  m = cur - 1;  }  } hull; | | ***Convex Hull***  struct ConvexHull {  vector<point> points;  void clear() {  points.clear();  }  void addPoint(ll x, ll y) {  points.push\_back( point(x, y) );  }  void addPoint( point p) {  points.push\_back(p);  }  // 2D cross product.  // Return a positive value, if OAB makes a counter-clockwise turn,  // negative for clockwise turn, and zero if the points are collinear.  ll cross(const point &O, const point &A, const point &B) {  return (A.x - O.x) \* (B.y - O.y) - (A.y - O.y) \* (B.x - O.x);  }  // Returns a list of points on the convex hull in counter-clockwise order.  // Note: the last PointLL in the returned list is the same as the first one.  vector< point > getConvexHull(vector< point >& P) {  int n = P.size(), k = 0;  vector< point > H(2\*n);  // Sort points lexicographically  sort(P.begin(), P.end());  // Build lower hull  for (int i = 0; i < n; i++) {  while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--; // strictly counter-clockwise then < 0 (skipping the colinear points) else use <=0  H[k++] = P[i];  }  // Build upper hull  for (int i = n-2, t = k+1; i >= 0; i--) {  while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--; // strictly counter-clockwise then < 0 (skipping the colinear points) else use <=0  H[k++] = P[i];  }  H.resize(k);  //H.pop\_back(); // first point comes again . so pop  return H;  }  vector< point > getConvexHull() {  return getConvexHull(points);  }  };  ***Histogram***  i64 calc(int \*ht, int n) {  i64 ret = 0;  int top = 1, st[MAX], i;  ht[0] = st[0] = ht[++n] = 0;  for(i = 1; i <= n; i++) {  while(top > 1 && ht[st[top-1]] >= ht[i]) {  ret = \_max(ret, (i64)ht[st[top-1]]\*(i64)(i - st[top-2]-1));  top--;  }  st[top++] = i;  }  return ret;  } | | ***Gaussian Ellimination***  /\*  1. Set row and col of mat  2. Call rank() to perform gauss-elimination and find rank  3. Call isValid() to find if solution exists.  Careful about int a[x][x]. If mod^2 crosses int, take vlong  If mod is 2, it is better to use XOR since it a lot faster.  \*/  struct GAUSS {  int row, col;  vlong a[x][x];  int mod;  bool valid;  GAUSS() {  mod = xyz;  }  void clear () {  memset ( a, 0, sizeof a );  }  void isValid ( int st ) {  int i;  valid = true;  for ( i = st; i < row; i++ ) {  if ( a[i][col-1] ) {  valid = false;  return;  }  }  }  ///Return Rank of Matrix  ///Free variable = Variable - Rank or Col - Rank - 1  int rank() {  int i = 0, j = 0, k, r, u;  while(i < row && j < col - 1) {  r = i;  for(k = i; k < row; k++)  if(a[k][j]) {  r = k; ///Find non-zero coefficient  break;  }  if(a[r][j]) {  if(r != i) ///Swap row if required  for(k = 0; k < col; k++)  swap(a[r][k], a[i][k]);  ///Neutralize if required. Depends on whether double or modular division  vlong v = a[i][j];  v = modInv ( v, mod );  for ( u = j; u < col; u++ ) {  a[i][u] = ( a[i][u] \* v ) % mod;  }  for(u = i + 1; u < row; u++)  if(a[u][j]) { ///Eliminate  int v = a[u][j];  for(k = j; k < col; k++) {  a[u][k] = ( ( a[i][k] \* v ) - a[u][k] ) % mod; ///Change here if no mod  if ( a[u][k] < 0 ) a[u][k] += mod;  }  } i++;  }j++;  }return i;  }  void print() {  FOR(i,0,row-1) {  FOR(j,0, col-1) {  printf ( "%d ", a[i][j] );  }  nl;  }  }  } mat; | |
| ***Geometry Library***  #define MAXD 2  #define FOR(i,b,c) for(int i = b ; i < c ; i++ )  #define eps 1e-9  #define pi acos(-1.0)  #define SQ(x) ((x)\*(x))  #define MAXD 2  #define SQ(x) ((x)\*(x))  double cosineRule3Side ( double a, double b, double c ) {  double res = (SQ(a)+SQ(b)-SQ(c)) / (2\*a\*b);  if ( res < -1 ) res = -1;  if ( res > 1 ) res = 1;  return acos ( res );  }  struct myVec {  int d; //Dimension  double val[MAXD];//Contains value of each component  myVec() {  d=2;memset(val,0,sizeof(val));  }  myVec(double \_x,double \_y) {  d=2;  val[0]=\_x;  val[1]=\_y;  }  myVec(int \_d,double\*a) {  d=\_d;  for(int i=0; i<d; i++) {  val[i]=a[i];  }  }  myVec add ( myVec b ) {  myVec res;  for(int i=0; i<d; i++) res.val[i] = val[i] + b.val[i];  return res;  }  myVec sub ( myVec b ) {  myVec res;  for(int i=0; i<d; i++) res.val[i] = val[i] - b.val[i];  return res;  }  myVec mul ( double t ) {  myVec res;  for(int i=0; i<d; i++)res.val[i] = val[i] \* t;  return res;  }  myVec div ( double t ) {  myVec res;  for(int i=0; i<d; i++) res.val[i] = val[i] / t;  return res;  }  bool operator == ( myVec b ) {  for(int i=0; i<d; i++) if ( fabs ( val[i] - b.val[i] ) > eps ) return false;  return true;  }  myVec perp2D() {  myVec res = (\*this);  swap ( res.val[0], res.val[1] );  res.val[0] \*= -1;  return res;  }  double dot ( myVec v ) { //Finds \*this (dot) v  double res = 0;  for ( int i = 0; i < d; i++ ) res += val[i] \* v.val[i];  return res; | }  double length () { //Finds length of current vector  return sqrt ( this->dot( \*this ) );  }  myVec unitVec () {  return (\*this).div ( length() ); // v / ||v||  }  double angleBetween ( myVec b ) { //Angle between two vectors  double res = dot( b ) / ( length() \* b.length() );  if ( res > 1 ) res = 1;  if ( res < -1 ) res = -1;  return acos (res);  }  double polarAngle2D() { //Angle from x-axis  double res = atan2 ( val[1], val[0] );  if ( res + eps < 0 ) res += 2 \* pi;  return res;  }  double cross2D ( myVec v ) { //Cross the two values. Only for 2D. Z compo 0.  return val[0]\*v.val[1] - val[1]\*v.val[0];  }  };  struct myLine {  myVec a, b; //a is displacement, b is direction.  //Builds a line from two points  myLine() {}  myLine(myVec x, myVec y) {  a=x;b=y.sub(x);  }  myLine lineFromPoints ( myVec x, myVec y ) {  myLine m;  m.a = x;  m.b = y.sub ( x );  return m;  }  //Finds point on line, given t.  myVec atPos ( double t ) {  return a.add ( b.mul ( t ) ); // a + tb;  }  double lineToPointDistance ( myVec p, double t ) {  p = p.sub ( a ); //Take it to origin  t = b.dot ( p ) / ( b.length() \* b.length() ); //point of intersection  myVec x = b.mul ( t ); //tb  return ( p.sub(x).length() ); //xp length()  }  double segmentToPointDistance ( myVec p, double &t ) {  p = p.sub ( a ); //Take it to origin  t = b.dot ( p ) / ( b.length() \* b.length() );  if ( t + eps < 0 || t > 1 + eps ) { //Not on segment  return min ( p.length(), p.sub(b).length() );  }  myVec x = b.mul ( t ); //tb  return ( p.sub(x).length() ); //xp length()  } | | bool overlapParallel ( myLine l ) {  double p, q, r, s;  if ( b.val[0] == 0 ) {  p = a.val[1];  q = atPos(1).val[1];  r = l.a.val[1];  s = l.atPos ( 1 ).val[1];  if ( min ( r, s ) > max ( p, q ) ) return false;  if ( max ( r, s ) < min ( p, q ) ) return false;  return true;  } else {  p = a.val[0];  q = atPos(1).val[0];  r = l.a.val[0];  s = l.atPos ( 1 ).val[0];  if ( min ( r, s ) > max ( p, q ) ) return false;  if ( max ( r, s ) < min ( p, q ) ) return false;  return true;  }  }  char lineAndLineIntersection2D ( myLine l, double &t, double &s ) {  if ( b.cross2D ( l.b) == 0 ) {  if ( l.a.sub(a).cross2D(l.b) == 0 ) {  if ( overlapParallel ( l ) ) return 'o'; //overlaps  else return 'p'; //parallel  } else return 'd'; //disjoint and parallel  }  myVec w = a.sub ( l.a );  myVec p = l.b.perp2D(), z = b.perp2D();  t = -(w.dot(p))/p.dot(b); //for current line  s = w.dot(z)/z.dot(l.b); //for line l  return 'i';  }  double lineAndLineDistance2D ( myLine l ) {  double t, s; //First check if the intersect  char r = lineAndLineIntersection2D ( l, t, s );  if ( r == 'i' ) return 0; //Intersects. 0 distance.  //Parallel Lines  return lineToPointDistance ( l.a, t );  }  double lineAndSegmentDistance2D ( myLine l ) {  double t, s;  char r = lineAndLineIntersection2D ( l, t, s );  if ( r == 'i' && s + eps > 0 && s < 1 + eps ) {  return 0; //Valid intersection  }  double res = lineToPointDistance ( l.a, t );  res = min ( res, lineToPointDistance ( l.a.add(l.b), t ) );  return res;  } | | double segmentAndSegmentDistance2D ( myLine l ) {  double t, s;  char r = lineAndLineIntersection2D ( l, t, s );  if ( r =='i' && t+eps > 0 && t < 1 + eps && s + eps > 0 && s < 1 + eps ) {  return 0; //Valid intersection  }  double res = segmentToPointDistance ( l.a, t );  res = min ( res, segmentToPointDistance ( l.a.add(l.b), t ) );  res = min ( res, l.segmentToPointDistance ( a, t ) );  res = min ( res, l.segmentToPointDistance ( a.add ( b ), t ) );  return res;  }  myLine reflect ( myVec p, myVec norm ) {  myVec ap = p.sub ( a ); //Starting to Point of Reflection  norm = norm.unitVec();  double d = fabs ( ap.dot ( norm ) );  myVec m = p.add ( norm.mul ( d ) );  myVec h = m.sub ( a ).mul ( 2 );  m = a.add ( h );  myLine ray = ray.lineFromPoints ( p, m );  return ray;  }  }; |
| struct myCir {  myVec a;  double r;  myVec atPos ( double t ) {  myVec res;  res.val[0] = a.val[0] + r \* cos ( t );  res.val[1] = a.val[1] + r \* sin ( t );  return res;  }  char circleAndLineIntersection2D ( myLine l, double &t1, double &t2 ) {  double t3;  double d = l.lineToPointDistance ( a, t3 );  if ( d > r + eps ) return 'd';  if ( fabs ( d - r ) <= eps ) return 't';  myVec m = l.atPos ( t3 );  myVec am = m.sub ( a );  //Need to handle when line passes through center  double x = am.polarAngle2D();  double temp = d / r;  if ( temp > 1 ) temp = 1;  if ( temp < -1 ) temp = -1;  double theta = pi / 2 - asin ( temp ); //Using sin law find internal angle.  t1 = x + theta;  t2 = x - theta;  return 'i';  }  char sphereAndLineIntersect ( myLine l, double &t1, double &t2 ) {  double tp = 0;  double d = l.lineToPointDistance ( a, tp );  if ( d > r + eps ) return 'd';  if ( fabs ( d - r ) < eps ) {  t1 = tp;  return 't';  }  double chord = sqrt ( r \* r - d \* d );  t1 = tp - chord / l.b.length();  t2 = tp + chord / l.b.length();  return 'i';  }  char circleAndCircleIntersection2D ( myCir c2, double &t1, double &t2 ) {  myVec d = c2.a.sub ( a );  if ( d.length() > r + c2.r + eps ) return 'd'; //Case 1  if ( d.length() + c2.r + eps < r ) return 'd'; //Case 2  if ( a == c2.a && fabs ( r - c2.r ) <= eps ) {  if ( r == 0 ) {  t1 = 0;  return 't'; //Case 7  }  return 's'; //Case 6  }  if ( fabs ( d.length() - r - c2.r ) <= eps ||  fabs ( d.length() + c2.r - r ) <= eps ) {  t1 = d.polarAngle2D();  return 't'; //Case 3 and 4  }  double theta = cosineRule3Side ( r, d.length(), c2.r );  double m = d.polarAngle2D ();  t1 = m - theta;  t2 = m + theta;  return 'i'; //Case 5  }  int circleToCircleTangentLine (myCir c2,myLine &l1,myLine &l2,myLine &l3,myLine &l4) {  //First circle must be smaller or equal to second circle  if (r>c2.r + eps ) return c2.circleToCircleTangentLine ( \*this, l1, l2, l3, l4 );  myVec oo = c2.a.sub ( a );  double d = oo.length();  if ( fabs ( d ) < eps && fabs ( r - c2.r ) < eps ) //Infinite tangents  return -1; | | if ( d + r + eps < c2.r ) //No tangents  return 0;  double base = oo.polarAngle2D();  if ( fabs ( d + r - c2.r ) < eps ) { //Contains Circle  l1 = l1.lineFromPoints ( atPos ( base + pi ), atPos ( base + pi ) );  return 1;  }  double ang = pi - acos ( (c2.r - r ) / d );  l1 = l1.lineFromPoints ( atPos ( base + ang ), c2.atPos ( base + ang ) );  l2 = l2.lineFromPoints ( atPos ( base - ang ), c2.atPos ( base - ang ) );  if ( d + eps < r + c2.r ) return 2; //Circle intersects  if ( fabs ( d - r - c2.r ) < eps ) { //Circle tangent  l3 = l3.lineFromPoints ( atPos ( base ), atPos ( base ) );  return 3;  }  //Disjoint Circle  ang = acos ( ( c2.r + r ) / d );  l3 = l3.lineFromPoints ( atPos ( base + ang ), c2.atPos ( base + ang + pi ) );  l4 = l4.lineFromPoints ( atPos ( base - ang ), c2.atPos ( base - ang + pi ) );  return 4;  }  };  ***Heavy Light Decomposition:LOJ***  #include<bits/stdc++.h>  using namespace std ;  #define MAX 30100  #define LGMAX 16  int par[MAX], head[MAX], depth[MAX], nChild[MAX], heavyNd[MAX], mp[MAX] ;  int nEdge, bit\_n, tree[MAX], genies[MAX] ;  vector<int>g[MAX] ;  void update( int idx , int val) {  for( int j = idx ; j<= bit\_n ; j += (j & -j) ) {  tree[j] += val ;  }  }  int read( int idx ) {  int ret = 0 ;  for( int j = idx ; j ; j -= (j & -j ) ) {  ret += tree[j] ;  }  return ret ;  }  int atPos( int a ) {  a = mp[a] ;  return read(a)-read(a-1) ;  }  int rangeSum( int a , int b ) {  a = mp[a] , b = mp[b] ;  return read(b)-read(a-1) ;  }  void dfs( int u , int bc , int \_depth ) {  par[u] = bc ;  nChild[u] = 1 ;  depth[u] = \_depth ;  int mx = 0 ;  for( int i = 0 ; i < (int)g[u].size() ; i++ ) {  int v = g[u][i] ;  if( v != bc ) {  dfs(v,u,\_depth+1) ;  nChild[u]+=nChild[v] ;  if( nChild[v] > mx ) {  heavyNd[u] = v ;  mx = nChild[v] ;  }  } | | }  }  int query( int u , int v ) {  int ans = 0 ;  while( 1 ) {  int uChain = head[u] , vChain = head[v] ;  if( uChain == vChain ) {  if( depth[u] < depth[v] )  ans += rangeSum(u,v) ;  else ans += rangeSum(v,u) ;  break ;  } else if( depth[uChain] < depth[vChain] ) {  ans += rangeSum(vChain,v) ;  v = par[vChain] ;  } else {  ans += rangeSum(uChain,u) ;  u = par[uChain] ;  }  }  return ans ;  }  void buildChain(int u , bool isChainHead ) {  head[u] = isChainHead ? u : head[par[u]] ;  mp[u] = ++bit\_n ;  if( heavyNd[u] != -1 ) {  buildChain( heavyNd[u] , 0 ) ;  }  for( int i = 0 ; i < (int)g[u].size() ; i++ ) {  int v = g[u][i] ;  if( v == par[u] )continue ;  if( v != heavyNd[u] ) {  buildChain( v, 1 ) ;  }  }  }  void buildSum(int n) {  for( int i = 0 ; i < n ; i++ ) {  update(mp[i],genies[i]) ;  }  }  void init(int n) {  nEdge = 0 , bit\_n = 0;  for( int i = 0 ; i < n ; i++ )g[i].clear() ;  memset( tree , 0 , sizeof tree ) ;  memset( heavyNd , -1 , sizeof heavyNd ) ;  }  int main() {  int cases = readInt() ;  int caseno = 1 ;  while( cases -- ) {  int n = readInt() ;  init(n) ;  for( int i = 0 ; i < n ; i++ ) {  genies[i] = readInt() ;  }  for( int i = 0 ; i < n-1 ; i++ ) {  int u = readInt() ;int v = readInt() ;  g[u].push\_back(v) ;g[v].push\_back(u) ;  }  dfs( 0 , -1 , 0 ) ;  buildChain( 0 , 1 ) ;  buildSum( n ) ;  cout << "Case " << caseno++ << ":\n" ;  int q = readInt() ;  while( q -- ) {  int t = readInt() ;int u = readInt() ;  int v = readInt() ;  if( t ) {  update( mp[u] , v-atPos(u) ) ;  } else {  cout << query( u , v ) << "\n" ;  }  }  }  return 0 ;  } | |
| ***Heavy Light Decomposition-Q-tree1***  #include<bits/stdc++.h>  using namespace std;  int n,u,v,w;  int tree[4\*10004],  level[10004],  pr[10004],  head[10004],chainid[10004],  chainpos[10004],  edge[10004][3],  pos,nchain,a[10004],  child[10004];  vector<pair<int,int> >g[10004];  void init(int nd,int bb,int ee) {  if(bb==ee) {  tree[nd]=a[bb];  return;  }  int ll=nd<<1,rr=ll|1,mm=(bb+ee)>>1;  init(ll,bb,mm);  init(rr,mm+1,ee);  tree[nd]=max(tree[ll],tree[rr]);  return;  }  void update(int nd,int bb,int ee,int ii,int vv) {  if(bb>ii || ee<ii)return;  if(bb==ee && bb==ii) {  tree[nd]=vv;  return;  }  int ll=nd<<1,rr=ll|1,mm=(bb+ee)>>1;  update(ll,bb,mm,ii,vv);  update(rr,mm+1,ee,ii,vv);  tree[nd]=max(tree[ll],tree[rr]);  return;  }  int query(int nd,int bb,int ee,int ii,int jj) {  if(ii>jj)return 0;  if(bb>jj || ee<ii)return 0;  if(bb>=ii && ee<=jj)return tree[nd];  int ll=nd<<1,rr=ll|1,mm=(bb+ee)>>1;  return max(query(ll,bb,mm,ii,jj),query(rr,mm+1,ee,ii,jj));  }  void dfs(int s,int p) {  child[s]=1;  for(int i=0; i<(int)g[s].size(); i++) {  if(g[s][i].first!=p) {  level[g[s][i].first]=level[s]+1;  pr[g[s][i].first]=s;  dfs(g[s][i].first,s);  child[s]+=child[g[s][i].first];  }  }  return;  }  inline int lca(int x,int y) {  if(x==y)return 0;  while(true) {  if(chainid[x]==chainid[y])  return (level[x]<level[y])?x:y;  if(level[head[chainid[x]]]<level[head[chainid[y]]])  y=pr[head[chainid[y]]];  else x=pr[head[chainid[x]]];  }  return 0;  } | | void hld(int s,int p,int c) {  if(head[nchain]==-1) {  head[nchain]=s;  }  chainid[s]=nchain;  chainpos[s]=pos;  a[pos]=c;  pos++;  int heavyid,mx=-1,ww;  for(int i=0; i<(int)g[s].size(); i++) {  if(g[s][i].first!=p) {  if(mx<child[g[s][i].first]) {  mx=child[g[s][i].first];  heavyid=g[s][i].first;  ww=g[s][i].second;  }  }  }  if(mx!=-1) {  hld(heavyid,s,ww);  }  for(int i=0; i<(int)g[s].size(); i++) {  if(g[s][i].first!=p && g[s][i].first!=heavyid) {  nchain++;  hld(g[s][i].first,s,g[s][i].second);  }  }  return;  }  int hldquery(int x,int y) {  int ret=0;  while(true) {  if(chainid[x]==chainid[y]) {  ret=max(ret,  query(1,1,n,chainpos[y]+1,chainpos[x]));  break;  }  ret=max(ret,  query(1,1,n,  chainpos[head[chainid[x]]],chainpos[x]));  x=pr[head[chainid[x]]];  }  return ret;  }  /\*  ***Shanks baby step giant step***  for the equation: b = a^x % p where a, b, p known, finds x  works only when p is an odd prime  \*/  int shank(int a, int b, int p) {  int i, j, m;  long long c, aj, ami;  map< long long, int > M;  map< long long, int > :: iterator it;  m = (int)ceil(sqrt((double)(p)));  M.insert(make\_pair(1, 0));  for(j = 1, aj = 1; j < m; j++) {  aj = (aj \* a) % p;  M.insert(make\_pair(aj, j));  }  ami = modexp(modinv(a, p), m, p);  for(c = b, i = 0; i < m; i++) {  it = M.find(c);  if(it != M.end()) return i \* m + it->second;  c = (c \* ami) % p;  }  return 0;  } | | int main() {  ios\_base::sync\_with\_stdio(0);  cin.tie(0);  int t;  cin>>t;  while(t--) {  for(int i=1; i<=n; i++) {  g[i].clear();  }  pos=1;  nchain=1;  memset(head,-1,sizeof(head));  memset(pr,-1,sizeof(pr));  memset(tree,0,sizeof(tree));  cin>>n;  for(int i=1; i<n; i++) {  cin>>u>>v>>w;  g[u].push\_back(make\_pair(v,w));  g[v].push\_back(make\_pair(u,w));  edge[i][0]=u,edge[i][1]=v,edge[i][2]=w;  }  level[1]=0;  a[1]=0;  dfs(1,-1);  hld(1,-1,0);  init(1,1,n);  string s;  long long x,y;  while(cin>>s && s[0]!='D') {  cin>>x>>y;  if(s[0]=='Q') {  if(x==y) {  cout<<"0\n";  continue;  }  int \_lca=lca(x,y);  cout<<max(hldquery(x,\_lca),hldquery(y,\_lca))<<"\n";  } else {  u=edge[x][0];  v=edge[x][1];  if(pr[u]==v)swap(u,v);  update(1,1,n,chainpos[v],y);  a[chainpos[v]]=y;  }  }  }  return 0;  } | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Hn***  void precalc() {  arr[0] = 0 ;  arr[1] = 1 ;  for( int i = 2 ; i < 1000000 ; i++ ) {  arr[i] = arr[i-1]+ (1/double(i)) ;  }  }  #define gamma 0.57721566490153286060  651209008240243104215933593992  double Hn( double N ) {  if( N < 1000000 ) {  return arr[int(N)];  } else return (log(double(N))+log(double(N+1)))/2 + gamma ; ;  }  ---------------------------------  long long H( int n ) {  long long res = 0;  for( int i = 1; i <= n; i++ )  res = res + n / i;  return res;  }  long long a ;  scanf("%lld",&a ) ;  long long ans = 0 ;  long long LIM = sqrt( a ) ;  for( int i = 1 ; i <= LIM ; i++ ) {  ans += a/i ;  }  printf("Case %d: %lld\n",caseno++ , (ans<<1) - LIM\*LIM ) ;  ***//KMP p=A^m ;***  int main() {  ios\_base::sync\_with\_stdio(0);  cin.tie(nullptr);  string s;  while(cin>>s && s!=".") {  if(s.size()==1) {  cout<<"1\n";  continue;  }  int jump[s.size()];  jump[0]=0;  for(int i=1,q=0; i<(int)s.size(); i++) {  while(q>0 && s[i]!=s[q])q=jump[q-1];  if(s[i]==s[q])q++;  jump[i]=q;  }  int x=s.size()-jump[s.size()-1];  if(s.size()%x==0) {  cout<<s.size()/x<<"\n";  } else cout<<"1\n";  }  return 0;  } | ***//KMP***  int \_next[1000000+1] ;  void failure( string T ) {  int \_left = 0 , \_right ;  \_next[0] = 0 ;  for( \_right = 1 ; \_right < T.size() ; \_right ++ ) {  while( \_left > 0 && T[\_left] != T[\_right] ) {  \_left = \_next[\_left-1] ;  }  if( T[\_left] == T[\_right] )\_left++ ;  \_next[\_right] = \_left ;  }  }  int KMP(string T,string P){  int \_left = 0 , \_right , cnt = 0 ;  for( \_right = 0 ; \_right < T.size() ; \_right ++ ){  while( \_left > 0 && P[\_left] != T[\_right] ) \_left = \_next[\_left-1] ;  if( P[\_left] == T[\_right] )\_left ++ ;  if( \_left == P.size() ) {  cnt ++ ;  \_left = \_next[\_left-1] ;  }  }  return cnt ;  }  ***//Z***  int M[1000000] ;  int z[1000000] ;  string s ;  void calc() {  memset(z,0,sizeof z) ;  int L = 0, R = 0;  int n = s.size() ;  for (int i = 1; i < n ; i++) {  if (i > R) {  L = R = i;  while (R < n && s[R-L] == s[R]) R++;  z[i] = R-L;  R--;  } else {  int k = i-L;  if (z[k] < R-i+1) z[i] = z[k];  else {  L = i;  while (R < n && s[R-L] == s[R]) R++;  z[i] = R-L;  R--;  }  }  }  } | | ***// Text Repitation***  #include<bits/stdc++.h>  using namespace std ;  int \_next[1000000+2];  void failure( string s ){  \_next[0] = 0 ;  for( int i = 1,q=0 ; s[i] ; i++ ){  while( q > 0 && s[q] != s[i] )q = \_next[q-1] ;  if( s[q] == s[i] ) q++ ;  \_next[i] = q ;  }  }  int main() {  int n ;  string s ;  while(cin >> n >> s ) {  if( n == -1 && s =="\*")break ;  if( s.size() > n ){  cout << 0 << '\n' ;  continue ;  }else if( s.size()== n ){  cout << 1 << '\n' ;  continue ;  }  failure(s);  int k = \_next[s.size()-1] ;  cout << 1+(n-s.size())/(s.size()-k) << "\n" ;  }  return 0 ;  }  /\*  14 abcab  1000 abcde  1000000000 z  1 zzzzz  -1 \*  output  4  200  1000000000  0  14 “abcab” == “abcabcabcabcab”  ans is 4  \*/ |
| /\*  ***Manacher algorithm implementation.***  Application, largest palindromic substring, largest palindromic suffix  \*/  int lengths[MAX<<1];  int manacher(char \*buff, int len) {  int i, k, pallen, found, d, j, s, e;  k = pallen = 0;  for(i = 0; i < len; ) {  if(i > pallen && buff[i-pallen-1] == buff[i]) {  pallen += 2, i++;  continue;  }  lengths[k++] = pallen;  s = k - 2, e = s - pallen, found = 0; | | for(j = s; j > e; j--) {  d = j - e - 1;  if(lengths[j] == d) {  pallen = d;  found = 1;  break;  }  lengths[k++] = (d < lengths[j]? d : lengths[j]);  }  if(!found) { pallen = 1; i++; }  }  lengths[k++] = pallen;  return lengths[k-1];  } | |
| ***Matrix Expo***  #include<bits/stdc++.h>  using namespace std ;  class matrix {  public:  vector<vector<int> >arr;  matrix() {}  matrix(int N) {  vector<int>y;  for( int i = 0 ; i < N ; i ++ ) {  for( int j = 0 ; j < N ; j++ ) {  y.push\_back(0);  }  arr.push\_back(y);  }  }  matrix operator \*(const matrix &in) {  matrix ret ;  int N=this->arr.size();  ret=matrix(N);  for( int i = 0 ; i < N ; i++ ) {  for( int j = 0 ; j < N ; j++ )  for( int k = 0 ; k < N ; k++ ) {  ret.arr[i][j]+=(arr[i][k])\*(in.arr[k][j]) ;  ret.arr[i][j]%=10 ;  }  }  return ret ;  }  matrix operator ^( long long int POW) {  matrix ret;  int N=this->arr.size();  ret=matrix(N);  for( int i = 0 ; i < N ; i++ ) {  ret.arr[i][i] = 1 ;  }  matrix ME = \*this ;  while( POW ) {  if( POW&1 ) {  ret = ret \* ME ;  }  ME = ME \* ME ;  POW >>= 1 ;  }  return ret ;  }  matrix operator +(const matrix &x) {  matrix ret;  int N=this->arr.size();  ret=matrix(N);  for(int i=0; i<N; i++) {  for(int j=0; j<N; j++) {  ret.arr[i][j]=(arr[i][j]+x.arr[i][j])%10;  }  }  return ret;  }  }; | /\*  ***Trie***  All operation has complexity O(length).  MAX is number of different items.  \*/  struct trie {  trie \*next[MAX+1];  trie() { for(int i=0; i<=MAX; i++) next[i] = NULL; }  };  void insert(trie \*root, int \*seq, int len) {  trie \*curr = root;  for(int i = 0; i < len; i++) {  if(!curr->next[seq[i]]) curr->next[seq[i]] = new trie;  curr = curr->next[seq[i]];  }  if(!curr->next[MAX]) curr->next[MAX] = new trie;  }  bool found(trie \*root, int \*seq, int len) {  trie \*curr = root;  for(int i = 0; i < len; i++) {  if(!curr->next[seq[i]]) return false;  curr = curr->next[seq[i]];  }  if(!curr->next[MAX]) return false;  return true;  }  ***nCr***  void precalc() {  for(int i = 1 ; i <= 1000 ; i++) {  nCr[i][0]=1;  nCr[i][1]=i;  for(int j = 2 ; j <= i ; j++ ) {  nCr[i][j] = ( nCr[i-1][j-1] + nCr[i-1][j] ) % MOD;  }  }  }  long long Pow(long long n , long long k) {  long long ret = 1;  while( k ) {  if( k&1 ) {  ret \*= n;  ret %= MOD;  }  n \*= n;  n %= MOD;  k >>= 1;  }  return ret;  } | | ***Point in Polygon***  int crossproduct(pair<int,int> a , pair<int,int> b , pair<int,int>p ){  return ((a.x-p.x)\*(b.y-p.y)-(a.y-p.y)\*(b.x-p.x)) ;  }  bool isInsidePolygon( pair<int,int> P[] , pair<int,int> sample , int n ){  int cnt = 0 ;  bool f = 0 ;  for( int i = 0 , j = n-1 ; i < n ; j = i++ ){  int Xmin = min( P[i].x,P[j].x ) ;  int Xmax = max( P[i].x,P[j].x ) ;  int Ymin = min( P[i].y,P[j].y ) ;  int Ymax = ma  x( P[i].y,P[j].y ) ;  if( Xmin <= sample.x && sample.x <= Xmax && Ymin <= sample.y && sample.y <= Ymax){  if( crossproduct( P[i] , P[j] , sample ) == 0 )return true ;  }  if( crossproduct( P[i] , P[j] , sample ) < 0 ){  swap( P[i],P[j] ) ;  f = 1 ;  }  if( P[i].y >= sample.y && P[j].y < sample.y )cnt++ ;  if( f == 1 ){  f = 0 ;  swap( P[i],P[j] ) ;  }  }  return ( cnt & 1 ) ;  }  ***SIEVE***  #define MAX 10000000  unsigned flag[MAX/64];  vector<long long>prime ;  #define chkC(n) (flag[n>>6]&(1<<((n>>1)&31)))  #define setC(n) (flag[n>>6]|=(1<<((n>>1)&31)))  int lim;  void sieve() {  unsigned i, j, k;  flag[0]|=0;  int sqrtN = sqrt(MAX) ;  for(i=3; i<= sqrtN ; i+=2)  if(!chkC(i))  for(j=i\*i,k=i<<1; j<MAX; j+=k)  setC(j);  prime.push\_back(2);  for(i=3; i<MAX; i+=2)  if(!chkC(i))  prime.push\_back(i) ;  lim = prime.size() ;  } |
| ***Segmented Sieve***  /\*  Generates primes within interval [a, b] when b - a <= 100000  and 1 <= a <= b <= 2147483647  \*/  int base[MAX>>6], segment[RNG>>6], primes[LEN], prlen;  #define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))  #define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))  void sieve() {  int i, j, k;  for(i=3; i<LMT; i+=2) if(!chkC(base, i)) for(j=i\*i, k=i<<1; j<MAX; j+=k) setC(base, j);  for(i=3, prlen=0; i<MAX; i+=2) if(!chkC(base, i)) primes[prlen++] = i;  } | | int segmented\_sieve(int a, int b) {  int rt, i, k, cnt = (a<=2 && 2<=b)? 1 : 0;  if(b<2) return 0;  if(a<3) a = 3;  if(a%2==0) a++;  memset(segment, 0, sizeof segment);  for(i=0, rt=(int)sqrt((double)b); i < prlen && primes[i] <= rt; i++) {  unsigned j = primes[i] \* ( (a+primes[i]-1) / primes[i] );  if(j%2==0) j += primes[i];  for(k=primes[i]<<1; j<=b; j+=k) if(j!=primes[i]) setC(segment, (j-a));  }  for(i=0; i<=b-a; i+=2) if(!chkC(segment, i)) cnt++;  return cnt;  } | |

|  |  |  |
| --- | --- | --- |
| ***Pollard Rho***  #include<bits/stdc++.h>  using namespace std;  #define LL long long  /\*  This is not my code . I collect It from net. use it. save it as  a pollard rho in my library. Use as a API for problem solving.  http://www.csie.ntnu.edu.tw/~u91029/Prime.html  \*/  int p[5500], pt = 0;  void sieve() {  int mark[46340] = {};  int i, j;  for(i = 2; i < 46340; i++) {  if(mark[i] == 0) {  p[pt++] = i;  for(j = i+i; j < 46340; j += i)  mark[j] = 1;  }  }  }  LL modmultiply(LL a,LL b,LL c) {  LL x = 0,y = a%c;  while(b > 0) {  if(b%2 == 1) {  x = (x+y)%c;  }  y = (y\*2)%c;  b /= 2;  }  return x%c;  }  LL modpow(LL x, LL y, LL mod) {  LL ret = 1;// ret = x^y%mod;  while(y) {  if(y&1)  //ret = (ret\*x)%mod;  ret = modmultiply(ret, x, mod);  //x = (x\*x)%mod;  x = modmultiply(x, x, mod);  y >>= 1;  }  return ret;  }  int isprime(LL n) {  if(n == 2 || n == 3)  return 1;  if(n < 2 || (n&1) == 0)  return 0;  int i, a;  for(i = 0; i < 5; i++) {  a = rand()%(n-4)+2;  if(modpow(a, n-1, n) != 1)  return 0;  }  return 1;  }  LL gcd(LL x, LL y) {  if(!x) return y;  if(!y) return x;  if(x < 0) x = -x;  if(y < 0) y = -y;  LL t;  while(x%y)  t = x, x = y, y = t%y;  return y;  }  vector<LL> ret;  LL pollard\_rho(LL n, LL c) {  long long x = 2, y = 2;  do {  //x = (x\*x+c)%n;  x = (modmultiply(x, x, n)+c)%n | y = (modmultiply(y, y, n)+c)%n;  y = (modmultiply(y, y, n)+c)%n;  LL d = gcd(x-y, n);  if(d > 1) return d;  } while(true);  return n;  }  void small\_factorize(LL n) {  int i;  for(i = 0; i < pt && p[i]\*p[i] <= n; i++) {  if(n%p[i] == 0) {  while(n%p[i] == 0)  ret.push\_back(p[i]), n /= p[i];  }  }  if(n != 1)  ret.push\_back(n);  }  void factorize(LL n) {  if(n == 1) return;  if(isprime(n)) {  ret.push\_back(n);  return;  }  if(n < 1000000000) {  small\_factorize(n);  return;  }  int c;  LL d = n;  for(c = 2; d == n; c++) {  d = pollard\_rho(n, c);  }  factorize(d);  factorize(n/d);  }  int main() {  sieve();  int cases ;  scanf("%d",&cases ) ;  while( cases-- ) {  long long n ;  scanf("%lld", &n);  ret.clear();  factorize(n);  sort(ret.begin(), ret.end());  cout << n << " = " ;  /\*  for( int i = 0 ; i < ret.size() ; i++ )cout << ret[i] << " " ;  cout << "\n" ;  \*/  int cnt = 1 ;  for( int i = 1 ; i < ret.size() ; i++ ){  if( ret[i] == ret[i-1] )cnt ++ ;  else {  cout << ret[i-1] ;  if( cnt > 1 ){  cout << "^" << cnt ;  }  cout << " \* " ;  cnt = 1 ;  }  }  cout << ret[ret.size()-1] ;  if( cnt > 1 )cout << "^" << cnt ;  cout << "\n" ;  }  return 0;  } | ***RMQ***  #define MAX 100000+5  #define LGMAX 18  int ST[MAX][LGMAX];  void rmq( int n ){  for( int j = 1 ; 1<<j <= n ; j++ ){  for( int i = 0 ; i+(1<<j)-1 < n ; i++ ){  ST[i][j] = min( ST[i][j - 1] , ST[i+(1<<(j-1))][j - 1] ) ;  }  }  }  int main() {  int cases = readInt() ;  int caseno = 1 ;  while( cases -- ){  int N = readInt() ;  int Q = readInt() ;  for( int i = 0 ; i < N ; i++ ){  ST[i][0] = readInt() ;  }  rmq(N) ;  cout << "Scenario #" << caseno++ << ":\n" ;  while( Q -- ){  int x = readInt()-1 ;  int y = readInt()-1 ;  int lim = y-x+1 , lg = 0 ;  while( lim ){  lim >>= 1 ;  lg++ ;  }  lg-- ;  cout << min( ST[x][lg] , ST[y-(1<<lg)+1][lg] ) << "\n" ;  }  }  return 0 ;  }  ***Rotate Point***  pair<double , double> Rotate\_Point( pair<double,double> P ,pair<double,double> Ref ,double dist, double cosA ,double sinA ){  //if only theta comes then sinA= sin(theta) , conA = cos( theta ) ;  double a = sqrt( (P.first-Ref.first)\*(P.first-Ref.first) + (P.second-Ref.second)\*(P.second-Ref.second) ) ;  double x1 = (dist\*(Ref.first-P.first))/a ;  double y1 = (dist\*(Ref.second-P.second))/a ;  pair<double,double> ret ;  ret.first = x1\*cosA-y1\*sinA+P.first ;  ret.second = x1\*sinA+y1\*cosA+P.second ;  return ret ;  } |

|  |  |  |
| --- | --- | --- |
| ***Stable Marrige***  #include<bits/stdc++.h>  using namespace std ;  #define rep(i,n) for( int i = 0 ;i < n ; i++ )  int Temp[101] ;  int n , p;  inline int readInt() {  int ip = getchar\_unlocked(), ret = 0, flag = 1;  for(; ip < 48 || ip > 57; ip = getchar\_unlocked()) {  if(ip == 45) {  flag = -1;  ip = getchar\_unlocked();  break;  }  }  for(; ip > 47 && ip < 58; ip = getchar\_unlocked())  ret = ret \* 10 + ip - 48 ;  return flag \* ret;  }  int main() {  int cases , caseno = 1;  scanf("%d",&cases ) ;  while( cases -- ) {  //scanf("%d",&n) ;  n = readInt() ;  vector<deque<int> > Can ;  int Com[n][n] ;  for( int i = 1 ; i <= n ; i++ ) {  deque<int>d ;  for( int j = 1; j <= n ; j++ ) {  //scanf("%d",&p) ;  p = readInt() ;  d.push\_back(p-n-1) ;  }  Can.push\_back(d) ;  }  for( int i = 0 ; i < n ; i++ ) {  for( int j = 0; j < n ; j++ ) {  //scanf("%d",&p) ;  p = readInt() ;  Com[i][p-1] = j ;  }  }  int cnt = 0 ;  do {  memset( Temp , -1 , sizeof Temp ) ;  cnt = 0 ;  for( int i = 0 ; i < n ; i++ ) {  int q = Can[i][0] ;  if( Temp[q] == -1 )Temp[q] = i,cnt++ ;  else {  int p = Temp[q] ;  if( Com[q][i] < Com[q][p] ) {  Temp[q] = i ;  Can[p].pop\_front() ;  } else {  Temp[q] = p ;  Can[i].pop\_front() ;  }  }  }  } while( cnt != n ) ;  cout << "Case " << caseno++ << ":" ;  for(int i = 0 ; i < n ; i++ ) {  cout << " (" << Temp[i]+1 << " " << i+n+1 << ")" ;  }  cout << "\n";  }  } | /\*  ***Tarjan's offline LCA*** algorithm. For each pair of node's in P {u, v, qid},  it finds the LCA of the nodes in the rooted tree G (no edge to back to the parent.  The array ans holds the result for queries in orders defined by qid.  \*/    void lca(int u) {  int v, i, sz;  make\_set(u);  ancestor[find\_set(u)] = u;  sz = G[u].size();  for(i = 0; i < sz; i++) {  v = G[u][i];  lca(v);  union\_set(u, v);  ancestor[find\_set(u)] = u;  }  color[u] = 1;  sz = P[u].size();  for(i = 0; i < sz; i++) {  v = P[u][i].first;  if(color[v]) ans[P[u][i].second] = ancestor[find\_set(v)];  }  }  ***Tan of Line***  pair<int,int> TanOfLine( pair<int,int> a , pair<int,int> b )  {  const int INF = 1<<30 ;  pair<int,int> M ;  if( a.y == b.y ) {  M.x = INF ;  M.y = INF ;  } else {  M.x = a.x-b.x ;  M.y = a.y-b.y ;  int tmp = \_\_gcd( (M.x),(M.y) ) ;  M.x = M.x/tmp ;  M.y = M.y/tmp ;  }  return M ;  } | ***SCC***  #include<bits/stdc++.h>  using namespace std ;  int n , m ;  vector<int>G[20001] ;  vector<int>GR[20001] ;  int visited[20001] ;  int I[20001] , O[20001] , C[20001] ;  int scc = 0 ;  stack<int> RPost ;  void dfs( int node ) {  visited[node] = 1 ;  for( int i = 0 ; i < G[node].size() ; i++ ) {  if(visited[G[node][i]] == 0 ) {  dfs( G[node][i] ) ;  }  }  RPost.push( node ) ;  }  void dfs2( int node ) {  visited[node] = 1 ;  for( int i = 0 ; i < GR[node].size() ; i++ ) {  if( !visited[GR[node][i]] ) {  dfs2( GR[node][i] ) ;  }  }  C[node] = scc ;  }  int main() {  int cases , caseno = 1 ;  scanf("%d",&cases ) ;  while( cases -- ) {  scanf("%d%d",&n,&m ) ;  for( int i = 0 ; i < m ; i++ ) {  int u , v ;  scanf("%d%d",&u,&v) ;  G[u].push\_back( v ) ;  GR[v].push\_back( u ) ;  }  for( int i = 1 ; i <= n ; i++ ) {  if( !visited[i] ) {  dfs( i ) ;  }  }  scc = 0 ;  for( int i = 1 ; i <= n ; i++ )visited[i] = 0 ;  while( !RPost.empty() ) {  if( !visited[RPost.top()] ) {  scc++ ;  dfs2( RPost.top() ) ;  }  RPost.pop();  }  }  return 0 ;  } |
| ***Max Flow DinitZ***  #include<bits/stdc++.h>  using namespace std;  struct edge{  int v,ri;  long long c;  edge(){}  edge(int \_v,long long \_c,int \_ri){v=\_v,c=\_c,ri=\_ri;}  edge(int \_v,long long \_c){v=\_v,c=\_c;}  };  vector<edge>g[5003];  int source,sink,level[5003];  void addEdge(int u,int v,long long c){  //cout<<"addedge "<<u<<" "<<v<<" "<<c<<endl;  g[u].push\_back(edge(v,c,g[v].size()));  g[v].push\_back(edge(u,0,g[u].size()-1));  return;  }  bool bfs(){  memset(level,-1,sizeof(level));  level[source]=0;  queue<int>q;  q.push(source);  while(!q.empty()){  int src=q.front();  q.pop();  for(int i=0;i<(int)g[src].size();i++){  if(level[g[src][i].v]==-1 && g[src][i].c>0){  level[g[src][i].v]=level[src]+1;  q.push(g[src][i].v);  }  }  }  return level[sink]!=-1;  }  long long dfs(int src,long long minCap){  if(src==sink)return minCap;  long long x=0,y=0;  for(int i=0;i<(int)g[src].size();i++){  if(g[src][i].c<=0 || level[g[src][i].v]!=level[src]+1)continue;  y=dfs(g[src][i].v,min(g[src][i].c,minCap-x));  x+=y;  g[src][i].c-=y;  g[g[src][i].v][g[src][i].ri].c+=y;  if(x==minCap)break;  }  if(x==0)level[src]=0;  return x;  }  int main(){  ios\_base::sync\_with\_stdio(0);  cin.tie(nullptr);  int n,m,u,v,c;  cin>>n>>m;  source=1,sink=n;  for(int i=0;i<m;i++){  cin>>u>>v>>c;  addEdge(u,v,c);  addEdge(v,u,c);  }  long long flow=0;  while(bfs()){  flow+=dfs(source,1LL<<50);  }  cout<<flow<<"\n";  return 0;  } | ***DinitZ with Edge list***  #define \_min(x,y) (((x)<(y))?(x):(y))  struct edge{  int u,v,\_next,w;  long long c;  edge(){}  edge(int \_u,int \_v,int \_\_next,long long \_c,int \_w){u=\_u,v=\_v,\_next=\_\_next,c=\_c,w=\_w;}  }edgeList[120005];  int ee,start[5003],nn;  void addEdge(int u,int v,long long cuv,long long cvu,int w){  edgeList[ee]=edge(u,v,start[u],cuv,w);start[u]=ee;ee++;  edgeList[ee]=edge(v,u,start[v],cvu,-w);start[v]=ee;ee++;  return;  }  int source,sink,level[5003];  void init(int s,int t,int n){  source=s,sink=t,ee=0,nn=n;  memset(start,-1,sizeof(start));  }  bool bfs(){  memset(level,-1,sizeof(level));  level[source]=0;  int q[nn+nn],ii=0,jj=0;  q[jj]=source;  jj++;  while(ii<jj && level[sink]==-1){  int src=q[ii];  ii++;  for(int i=start[src];i!=-1;i=edgeList[i].\_next){  if(level[edgeList[i].v]==-1 && edgeList[i].c>0){  level[edgeList[i].v]=level[src]+1;  q[jj]=edgeList[i].v;  jj++;  }  }  }  return level[sink]!=-1;  }  long long dfs(int src,long long \_minCap){  if(src==sink)return \_minCap;  long long x=0,y=0;  for(int i=start[src];i!=-1;i=edgeList[i].\_next){  if(edgeList[i].c<=0 || level[edgeList[i].v]!=level[src]+1)continue;  y=dfs(edgeList[i].v,\_min(edgeList[i].c,\_minCap-x));  x+=y;  edgeList[i].c-=y;  edgeList[i^1].c+=y;  if(x==\_minCap)break;  }  if(x==0)level[src]=0;return x;  }  int main(){  ios\_base::sync\_with\_stdio(0);  cin.tie(nullptr);  int n,m,u,v,c;  cin>>n>>m;  init(1,n,n);  for(int i=0;i<m;i++){  cin>>u>>v>>c;  addEdge(u,v,c,c,0);  //addEdge(v,u,c);  }  long long flow=0;  while(bfs()){  flow+=dfs(source,1LL<<50);  }  cout<<flow<<"\n";  return 0;  } | ***Min Cost Max Flow***  /\*  min cost flow (bellman ford)  works only on directed graphs  handles multiple edges, cycles, loops  \*/  int src, snk, nNode, nEdge;  int fin[MAXN], pre[MAXN], dist[MAXN];  int cap[MAXE], cost[MAXE], next[MAXE], to[MAXE], from[MAXE];  inline void init(int \_src, int \_snk, int nodes) {  SET(fin);  nNode = nodes, nEdge = 0;  src = \_src, snk = \_snk;  }  inline void addEdge(int u, int v, int \_cap, int \_cost) {  from[nEdge] = u, to[nEdge] = v, cap[nEdge] = \_cap, cost[nEdge] = \_cost;  next[nEdge] = fin[u], fin[u] = nEdge++;  from[nEdge] = v, to[nEdge] = u, cap[nEdge] = 0, cost[nEdge] = -(\_cost);  next[nEdge] = fin[v], fin[v] = nEdge++;  }  bool bellman() {  int iter, u, v, i;  bool flag = true;  MEM(dist, 0x7f);  SET(pre);  dist[src] = 0;  for(iter = 1; iter < nNode && flag; iter++) {  flag = false;  for(u = 0; u < nNode; u++) {  for(i = fin[u]; i >= 0; i = next[i]) {  v = to[i];  if(cap[i] && dist[v] > dist[u] + cost[i]) {  dist[v] = dist[u] + cost[i];  pre[v] = i;  flag = true;  }  }  }  }  return (dist[snk] < INF);  }  int mcmf(int &fcost) {  int netflow, i, bot, u;  netflow = fcost = 0;  while(bellman()) {  bot = INF;  for(u = pre[snk]; u >= 0; u = pre[from[u]]) bot = min(bot, cap[u]);  for(u = pre[snk]; u >= 0; u = pre[from[u]]) {  cap[u] -= bot;  cap[u^1] += bot;  fcost += bot \* cost[u];  }  netflow += bot;  }  return netflow;  } |

|  |  |  |
| --- | --- | --- |
| /\*  ***Bipertite Matching***  G[] is the left-side graph, must be bipartite  match(n): n is the number of nodes in left-side set  and returns the maximum possible matching.  Left[] anf Right[] ar assigned with corresponding matches  \*/  vector < int > G[MAX];  bool visited[MAX];  int Left[MAX], Right[MAX];  bool dfs(int u) {  if(visited[u]) return false;  visited[u] = true;  int len = G[u].size(), i, v;  for(i=0; i<len; i++) {  v = G[u][i];  if(Right[v]==-1) {  Right[v] = u, Left[u] = v;  return true;  }  }  for(i=0; i<len; i++) {  v = G[u][i];  if(dfs(Right[v])) {  Right[v] = u, Left[u] = v;  return true;  }  }  return false;  }  int match(int n) {  int i, ret = 0;  bool done;  SET(Left); SET(Right);  do {  done = true; CLR(visited);  for(i=0; i<n; i++) {  if(Left[i]==-1 && dfs(i)) {  done = false;  }  }  } while(!done);  for(i=0; i<n; i++) ret += (Left[i]!=-1);  return ret;  } |  |  |