**Ravenclaw Team’s NOTE**

**===MATH===**

* (p-1)! %p=p-1 //wilson theorem neu p la ngto
* Phi(n) = mul of (pi-1)\*(p^(ki-1)) ,n=p1^k1 \* p2\*k2\*…\*pm^km //pi is primes
* Neu gcd(a,n) =1 thi a^phi(n) %n =1
* Neu gcd(m,n)=1 thi phi(m,n) =phi(m) \*phi(n) //(day la ham nhan tinh)
* a^-1 %mod =a^(mod-2) %mod //neu a,mod ngto cung nhau
* nCk = n! / (k! \*(n-k)!)
* nPk = n ! / (n-k) !
* sum(i^2) =i(i+1)(2i+1)/6
* sum(a^i) =(a^n-1)/(a-1)
* loga(b) =loge(a) /loge(b)
* catalan(n) = (2n) ! / ((n+1) ! n !) = (2n)Cn /(n+1)

1 1 2 5 14 42

* S=n+ m/2 -1; //n la soluong diem co toa do nguyen trong da giac

M la so diem co toa do nguyen tren canh da giac

* d= sqrt( (x1-x2)^2+(y1-y2)^2 )

//khoang cach diem (x0,y0) den duong thang ax+by+c=0

* d= abs(ax0 + by0 +c) / sqrt(a^2+b^2)
* 2 duong thang vuong goc nhau <=> a1\*a2+b1\*b2=0

dien tich tam giac:

* S= sqrt(p\*(p-a)\*(p-b)\*(p-c)) = 1/2 b\*c\*sinA =p\*r=(abc)/4R //r,R la ban kinh duong tron noi tiep,ngoai tiep
* dien tich hinh tron s= pi\*r^2
* chu vi hinh tron c = 2pi\*r
* dien tich mat cau s= 4 pi r^2
* the tich khoi cau v=4/3 \*pi\*r^3

Grundy:

* G(x) =mex(all(G(yi))), //yi la cac cau hinh co the den dc tu x bang 1 buoc di chuyen
* He thuc luong (vuong tai A)

BC^2 = AB^2 + AC^2 (Định lí Pitago)=>AB^2 = BC^2 – AC^2

AB^2 = BH.BC

AC^2 = CH.BC

AH^2 = BH.CH

AB.AC = BC.AH

1/AH^2 = 1/AB^2 + 1/AC^2

* Dinh ly sin: a/sinA = b/sinB = c/sinC = 2R
* Diện tích = B/2 + I – 1

B = số lượng điểm nằm ở trên cạnh đa giác (Boundary)

I = số lượng điểm nằm trong đa giác (Inside)

* V - E + F = 2, với:

V = số đỉnh của khối

E = số cạnh của khối

F = số mặt của khối

* Tinh luy thua 1 so:

int power(int x, int n)

{  
If (n == 0) return 1;  
if (n % 1) return power(x, n-1) \* x;  
int temp = power(x, n/2);  
return temp\*temp;

}

**===FUNCTION===**

* **Noi Suy**:

int InterPolationSearch(int arr[], int n, int x)

{

  int left = 0;

  int right = n-1;

  while (left <= right && x >= arr[left] && x <= arr[right])

  {

    double val1 = (double) (x - arr[left]) / (arr[right]-arr[left]);

    int val2 = (right-left);

    int pos = left + val1\*val2;

    if (arr[pos] == x)

      return pos;

    if (arr[pos] < x)

      left = pos + 1;

    else

      right = pos - 1;

  }

  return -1;

}

* **NextBit String**:

void NextBitString()

{

int i = n-1;

while (i>=0&& x[i]!=0)

{

x[i] = 0;

i--;

}

if(i>=0)

x[i] = 1;

else check = false;

}

-**Sinh To Hop**:  
int a[100],n,k,stop=0;  
void khoitao()  
{

for(int i=1;i<=k;i++)  
a[i]=i;  
}  
void in()  
{  
for(int i=1;i<=k;i++)  
cout<<a[i]<<" ";  
cout<<endl;  
}  
void sinh()  
{  
int i=k;  
while(a[i]==n-k+i) i--;  
if(i==0) stop=1;  
else a[i]++;

int p=a[i];

while(i<=k) a[i++]=p++;

}

void tohop()

{

while(stop==0)

{

in(); sinh();

}

}

main()

{

cin>>n>>k; khoitao(); tohop();

}

* **Phi Ham Euler**

int eulerPhi(int n) { // = n (1-1/p1) ... (1-1/pn)

if (n == 0) return 0;

int ans = n;

for (int x = 2; x\*x <= n; ++x) {

**if** (n **%** x **==** 0) {

ans -= ans / x;

while (n % x == 0) n /= x;

}

}

if (n > 1) ans -= ans / n;

return ans;

}

* **Base-n to decima:**

int val(char c)

{

if (c >= '0' && c <= '9')

return (int)c - '0';

else

return (int)c - 'A' + 10;

}

// Function to convert a number from given base 'b'

// to decimal

int toDeci(char \*str, int base)

{

int len = strlen(str);

int power = 1; // Initialize power of base

int num = 0; // Initialize result

int i;

// Decimal equivalent is str[len-1]\*1 +

// str[len-1]\*base + str[len-1]\*(base^2) + ...

for (i = len - 1; i >= 0; i--)

{

// A digit in input number must be

// less than number's base

if (val(str[i]) >= base)

{

printf("Invalid Number");

return -1;

}

num += val(str[i]) \* power;

power = power \* base;

}

return num;

}

* **decimal to base-n:**

char reVal(int num)

{

if (num >= 0 && num <= 9)

return (char)(num + '0');

else

return (char)(num - 10 + 'A');

}

// Utility function to reverse a string

void strev(char \*str)

{

int len = strlen(str);

int i;

for (i = 0; i < len/2; i++)

{

char temp = str[i];

str[i] = str[len-i-1];

str[len-i-1] = temp;

}

}

// Function to convert a given decimal number

// to a base 'base' and

char\* fromDeci(char res[], int base, int inputNum)

{

int index = 0;  // Initialize index of result

 // Convert input number is given base by repeatedly

// dividing it by base and taking remainder

 while (inputNum > 0)

{

        res[index++] = reVal(inputNum % base);

       inputNum /= base;

}

res[index] = '\0';

   // Reverse the result

strev(res);

    return res;

}

* **Chat Tam Phan:**

int ternary\_search(int l,int r, int x)

{

if(r>=l)

{

int mid1 = l + (r-l)/3;

int mid2 = r - (r-l)/3;

if(ar[mid1] == x)

return mid1;

if(ar[mid2] == x)

return mid2;

if(x<ar[mid1])

return ternary\_search(l,mid1-1,x);

else if(x>ar[mid2])

return ternary\_search(mid2+1,r,x);

else

return ternary\_search(mid1+1,mid2-1,x);

}

return -1;

}

* **Dem so luong so nguyen to nho hon n (10^10)**

const int N = 100005;  
const int M = 1000000007;  
bool np[N];  
int p[N], pp=0;  
  
**void eratos() {**  
    np[0]=np[1]=true;  
    for (int i=2; i\*i<N; i++) if (!np[i])  
        for (int j=i\*i; j<N; j+=i) np[j]=true;  
    for (int i=2; i<N; i++)  
    if (!np[i]) p[++pp]=i;  
}  
  
**long power(long a, long k) {**  
    long P = 1;  
    while (k) {  
        if (k&1) P=P\*a;  
        k/=2; a=a\*a;  
    }  
    return P;  
}  
  
**long power(long a, long k, long M) {**  
    long P=1;  
    for (a=a%M; k; k/=2)  
    { if (k&1) P=P\*a%M; a=a\*a%M; }  
    return P;  
}  
  
**long root(long n, long k) {**  
    long x = pow(n, 1.0/k);  
    while (power(x, k)%M==power(x, k, M) && power(x, k)<n) x++;  
    while (power(x, k)%M!=power(x, k, M) || power(x, k)>n) x--;  
    return x;  
}  
  
map<long, long> Phi[N];  
  
**long phi(long x, int a) {**  
    if (Phi[a].count(x)) return Phi[a][x];  
    if (a==1) return (x+1)/2;  
    long Result = phi(x, a-1) - phi(x/p[a], a-1);  
    return Phi[a][x] = Result;  
}  
  
**long pi(long x) {**  
    if (x<N)  
        return upper\_bound(p+1, p+pp+1, x) - (p+1);  
    long a = pi(root(x, 4));  
    long b = pi(root(x, 2));  
    long c = pi(root(x, 3));  
    long Sum = phi(x, a) + (b+a-2)\*(b-a+1)/2;  
    for (int i=a+1; i<=b; i++)  
        Sum -= pi(x/p[i]);  
    for (int i=a+1; i<=c; i++) {  
        long bi = pi(root(x/p[i], 2));  
        for (int j=i; j<=bi; j++)  
        Sum -= pi(x/p[i]/p[j]) - (j-1);  
    }  
    return Sum;  
}

* **Đoạn con có tổng lớn nhât:**

f[0] = 0;

    for (int i = 1; i <= n; ++i) f[i] = f[i-1] + a[i];

    int maxsum = 0, f\_min = 0;

    for (int i = 1; i <= n; ++i) {

        f\_min = min(f\_min, f[i]);

        maxsum = max(maxsum, f[i] - f\_min);

    }

* **Unique:**

vector<int> a(5);

vector<int>::iterator it;

UNIQUE

it=unique(a.begin(),a.end());

a.resize(distance(a.begin(),it));

FIND

// tim vi tri so 30

it = find (a.begin(), a.end(), 30);

if(it!=a.end())

cout<<it-a.begin();

else cout<<”khong co”;

COUNT

//dem co bao nhieu so 1

Int Mycount=count(a.begin(),a.end(),1);

Cout<<Mycount;

* **====STRING===**

**-palindrome tree:**

#define MAXN  1000

struct Node

{

    // store start and end indexes of current

    // Node inclusively

    int start, end;

    // stores length of substring

    int length;

    // stores insertion Node for all characters a-z

    int insertEdg[26];

    // stores the Maximum Palindromic Suffix Node for

    // the current Node

    int suffixEdg;

};

// two special dummy Nodes as explained above

Node root1, root2;

// stores Node information for constant time access

Node tree[MAXN];

// Keeps track the current Node while insertion

int currNode;

string s;

int ptr;

void insert(int idx)

{

//STEP 1//

    /\* search for Node X such that s[idx] X S[idx]

       is maximum palindrome ending at position idx

       iterate down the suffix link of currNode to

       find X \*/

    int tmp = currNode;

    while (true)

    {

        int curLength = tree[tmp].length;

        if (idx - curLength >= 1 and s[idx] == s[idx-curLength-1])

            break;

        tmp = tree[tmp].suffixEdg;

    }

    /\* Now we have found X ....

     \* X = string at Node tmp

     \* Check : if s[idx] X s[idx] already exists or not\*/

    if(tree[tmp].insertEdg[s[idx]-'a'] != 0)

    {

        // s[idx] X s[idx] already exists in the tree

        currNode = tree[tmp].insertEdg[s[idx]-'a'];

        return;

    }

    // creating new Node

    ptr++;

    // making new Node as child of X with

    // weight as s[idx]

    tree[tmp].insertEdg[s[idx]-'a'] = ptr;

    // calculating length of new Node

    tree[ptr].length = tree[tmp].length + 2;

    // updating end point for new Node

    tree[ptr].end = idx;

    // updating the start for new Node

    tree[ptr].start = idx - tree[ptr].length + 1;

//STEP 2//

    /\* Setting the suffix edge for the newly created

       Node tree[ptr]. Finding some String Y such that

       s[idx] + Y + s[idx] is longest possible

       palindromic suffix for newly created Node.\*/

    tmp = tree[tmp].suffixEdg;

    // making new Node as current Node

    currNode = ptr;

    if (tree[currNode].length == 1)

    {

        // if new palindrome's length is 1

        // making its suffix link to be null string

        tree[currNode].suffixEdg = 2;

        return;

    }

    while (true)

    {

        int curLength = tree[tmp].length;

        if (idx-curLength >= 1 and s[idx] == s[idx-curLength-1])

            break;

        tmp = tree[tmp].suffixEdg;

    }

    // Now we have found string Y

    // linking current Nodes suffix link with s[idx]+Y+s[idx]

    tree[currNode].suffixEdg = tree[tmp].insertEdg[s[idx]-'a'];

}

// driver program

int main()

{

    // initializing the tree

    root1.length = -1;

    root1.suffixEdg = 1;

    root2.length = 0;

    root2.suffixEdg = 1;

    tree[1] = root1;

    tree[2] = root2;

    ptr = 2;

    currNode = 1;

    // given string

    s = "abcbab";

    int l = s.length();

    for (int i=0; i<l; i++)

        insert(i);

    // printing all of its distinct palindromic

    // substring

    cout << "All distinct palindromic substring for "

         << s << " : \n";

    for (int i=3; i<=ptr; i++)

    {

        cout << i-2 << ") ";

        for (int j=tree[i].start; j<=tree[i].end; j++)

            cout << s[j];

        cout << endl;

    }

    return 0;

}

**-suffix tree**

class Node

{

public:

char ch;

unordered\_map<char, Node\*> children;

vector<int> indexes; //store the indexes of the substring from where it starts

Node(char c):ch(c){}

};

int maxLen = 0;

string maxStr = "";

void insertInSuffixTree(Node\* root, string str, int index, string originalSuffix, int level=0)

{

root->indexes.push\_back(index);

// it is repeated and length is greater than maxLen

// then store the substring

if(root->indexes.size() > 1 && maxLen < level)

{

maxLen = level;

maxStr = originalSuffix.substr(0, level);

}

if(str.empty()) return;

Node\* child;

if(root->children.count(str[0]) == 0) {

child = new Node(str[0]);

root->children[str[0]] = child;

} else {

child = root->children[str[0]];

}

insertInSuffixTree(child, str.substr(1), index, originalSuffix, level+1);

}

int main()

{

string str ;

cin>>str;

Node\* root = new Node('@');

//insert all substring in suffix tree

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

string s = str.substr(i);

insertInSuffixTree(root, s, i, s);

}

cout << maxLen << "->" << maxStr << endl;

return 1;

}

**XÓA**

str.erase(str.begin() + 3): xóa 1 kí tự tại vị trí 3

str.erase(x, y); *// Xóa đi y ký tự bắt đầu tại vị trí x.*

str.erase(str.begin() + x); *// Xóa đi ký tự tại vị trí x.*

str.erase(str.begin() + x, str.begin() + y); *// Xóa đi các ký tự từ chỉ số x đến chỉ số y - 1*

**THÊM**

str.insert(str.begin() + 3, **'y'**): thêm 1 kí tự **'y'** tại vị trí số 3

str.insert(x, **"y"**, z); *// Thêm z ký tự từ chuỗi y vào vị trí x. Nếu z có độ dài lớn hơn độ dài chuỗi y thì tiếp tục thêm vào 1 khoảng trắng và sau đó lại bắt đầu thêm vào các ký tự của chuỗi từ vị trí 0 --> sao cho đủ z thì thôi.*

str.insert(x, y, **'z'**); *// Thêm y lần ký tự z vào vị trí x.*

str.insert(str.begin() + x, **'y'**); *// Thêm ký tự y vào vị trí x.*

str.insert(str.begin() + x, y, **'z'**); *// Thêm y lần ký tự z vào vị trí x.*

str.insert(str.begin() + x, str2.begin() + y, str2.begin() + z); *// Thêm chuỗi ký tự con của chuỗi str2 bắt đầu từ vị trí y cho đến vị trí z - 1 vào vị trí x trong chuỗi str.*

============ **LẤY CHUỖI CON**

string str2 = str.substr(x, y); *// Lấy ra chuỗi con của chuỗi str bắt đầu từ vị trí x và lấy ra y ký tự.*

string str2 = str.substr(x); *// Lấy ra chuỗi con của chuỗi str bắt đầu từ vị trí x cho đến cuối chuỗi.*

============ **TÌM KIẾM**

**int** found = str.find(**'x'**);

**if**(found != string::npos) => trả về found chính là vị trí xuất hiện đầu tiên của ký tự x trong chuỗi str.

**else** => Không tồn tại ký tự x trong chuỗi str.

**int** found = str.find(**"x"**);

**if**(found != string::npos) => trả về found chính là vị trí xuất hiện đầu tiên của chuỗi x trong chuỗi str.

**else** => Không tồn tại chuỗi x trong chuỗi str.

**int** found = str.find(**"x"**, y, z);

**if**(found != string::npos) => trả về found chính là vị trí xuất hiện đầu tiên của chuỗi x lấy ra z ký tự và sẽ bắt đầu tìm kiếm từ vị trí y trong chuỗi str.

**else** => Không tồn tại chuỗi x trong chuỗi str.

**===GRAPH===**

* **Tim cay khung nho nhat**

typedef pair<int, int> ii;  
const int N=100005, oo=0x3c3c3c3c;  
int n, m, d[N];  
vector<int> a[N], b[N];  
  
**int prim(int u) {**  
    int Sum = 0;  
    priority\_queue<ii> qu;  
    for (int i=1; i<=n; i++) d[i]=oo;  
    qu.push(ii(0, u)); d[u]=0;  
      
    while (qu.size()) {  
        ii Pop=qu.top(); qu.pop();  
        int u=Pop.second, du=-Pop.first;  
        if (du!=d[u]) continue;  
        Sum+=d[u]; d[u]=0;  
          
        for (int i=0; int v=a[u][i]; i++)  
        if (d[v] > b[u][i]) {   
            d[v]=b[u][i];   
            qu.push(ii(-d[v], v));   
        }  
    }  
    return Sum;  
}  
  
**main() {**  
    scanf("%d%d", &n, &m);  
    for (int i=1; i<=m; i++) {  
        int x, y, z;  
        scanf("%d%d%d", &x, &y, &z);  
        a[x].push\_back(y);  
        b[x].push\_back(z);  
        a[y].push\_back(x);  
        b[y].push\_back(z);  
    }  
    for (int i=1; i<=n; i++)  
        a[i].push\_back(0);  
    cout << prim(1) << endl;  
}

-**Dijstra**:

void dijkstra(){  
    priority\_queue <ii, vector<ii>, greater<ii> > pq;  
    int i, u, v, du, uv;  
  
    for (i=1; i<=n; i++) d[i] = oo;  
    d[1] = 0;  
    pq.push(ii(0, 1));  
  
    while (pq.size()){  
        u=pq.top().second;  
        du=pq.top().first;  
        pq.pop();  
        if (du!=d[u]) continue;  
  
        for (i=0; v=a[u][i].second; i++)  
        {  
            uv=a[u][i].first;  
            if (d[v]>du+uv) {  
                d[v]=du+uv;  
                pq.push(ii(d[v], v));  
            }  
        }  
    }  
  
 }

* **Đếm số chu trình qua M cạnh, N đỉnh**

long long n,a=1,b,z, N, M;

int main(){

cin>>N>>M;

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

z=b;

b=((N-1)\*a)%1000000007;

a=((N-2)\*a+z)%1000000007;

}

cout<<b;

}

greater<ll>

|  |  |
| --- | --- |
| **===BIGNUM===**  -------------------------------------- JAVA -------------------------------------------------------  Scanner cin = **new** Scanner(System.***in***);  BigInteger n = **new** BigInteger(cin.next());  **int** t = cin.nextInt();  System.***out***.print(n);  **Phương pháp** | |
| **Công cụ sửa đổi và loại** | **Phương pháp và mô tả** |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**abs**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#abs())()  Trả về một BigInteger có giá trị là giá trị tuyệt đối của BigInteger này. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**add**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#add(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this + val). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**and**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#and(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this & val). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**andNot**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#andNot(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this & ~val). |
| int | [**bitCount**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#bitCount())()  Trả về số lượng bit trong biểu diễn bổ sung của hai BigInteger này khác với bit dấu của nó. |
| int | [**bitLength**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#bitLength())()  Trả về số lượng bit trong biểu diễn hai phần bù tối thiểu của BigInteger này, *ngoại trừ* một bit dấu. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**clearBit**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#clearBit(int))(int n)  Trả về một BigInteger có giá trị tương đương với BigInteger này với bit được chỉ định đã bị xóa. |
| int | [**compareTo**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#compareTo(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  So sánh BigInteger này với BigInteger được chỉ định. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**divide**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#divide(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this / val). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html)[] | [**divideAndRemainder**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#divideAndRemainder(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một mảng gồm hai BigIntegers chứa (this / val) theo sau (this % val). |
| double | [**doubleValue**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#doubleValue())()  Chuyển đổi BigInteger này thành a double. |
| boolean | [**equals**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#equals(java.lang.Object))([**Object**](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) x)  So sánh BigInteger này với Object được chỉ định cho đẳng thức. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**flipBit**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#flipBit(int))(int n)  Trả về một BigInteger có giá trị tương đương với BigInteger này với bit được chỉ định được lật. |
| float | [**floatValue**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#floatValue())()  Chuyển đổi BigInteger này thành a float. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**gcd**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#gcd(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là ước số chung lớn nhất của abs(this)và abs(val). |
| int | [**getLowestSetBit**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#getLowestSetBit())()  Trả về chỉ số của một bit ngoài cùng bên phải (thứ tự thấp nhất) trong BigInteger này (số bit 0 ở bên phải của một bit ngoài cùng bên phải). |
| int | [**hashCode**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#hashCode())()  Trả về mã băm cho BigInteger này. |
| int | [**intValue**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#intValue())()  Chuyển đổi BigInteger này thành một int. |
| boolean | [**isProbablePrime**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#isProbablePrime(int))(int certainty)  Trả về truenếu BigInteger này có thể là số nguyên tố, falsenếu nó chắc chắn là hợp số. |
| long | [**longValue**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#longValue())()  Chuyển đổi BigInteger này thành a long. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**max**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#max(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về mức tối đa của BigInteger này và val. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**min**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#min(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về mức tối thiểu của BigInteger này và val. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**mod**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#mod(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) m)  Trả về một BigInteger có giá trị là (this mod m). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**modInverse**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#modInverse(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) m)  Trả về một BigInteger có giá trị là (this-1 mod m) . |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**modPow**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#modPow(java.math.BigInteger,%20java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) exponent, [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) m)  Trả về một BigInteger có giá trị là ( mod m lũy thừa này ) . |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**multiply**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#multiply(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this \* val). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**negate**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#negate())()  Trả về một BigInteger có giá trị là (-this). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**nextProbablePrime**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#nextProbablePrime())()  Trả về số nguyên đầu tiên lớn hơn số này BigIntegercó lẽ là số nguyên tố. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**not**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#not())()  Trả về một BigInteger có giá trị là (~this). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**or**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#or(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this | val). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**pow**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#pow(int))(int exponent)  Trả về một BigInteger có giá trị là ( số mũ này ) . |
| static [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**probablePrime**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#probablePrime(int,%20java.util.Random))(int bitLength, [**Random**](https://docs.oracle.com/javase/7/docs/api/java/util/Random.html) rnd)  Trả về một BigInteger dương có lẽ là số nguyên tố, với bitLpm được chỉ định. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**remainder**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#remainder(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this % val). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**setBit**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#setBit(int))(int n)  Trả về một BigInteger có giá trị tương đương với BigInteger này với tập bit được chỉ định. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**shiftLeft**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#shiftLeft(int))(int n)  Trả về một BigInteger có giá trị là (this << n). |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**shiftRight**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#shiftRight(int))(int n)  Trả về một BigInteger có giá trị là (this >> n). |
| int | [**signum**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#signum())()  Trả về hàm đăng nhập của BigInteger này. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**subtract**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#subtract(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this - val). |
| boolean | [**testBit**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#testBit(int))(int n)  Trả về truenếu và chỉ khi bit được chỉ định được đặt. |
| byte[] | [**toByteArray**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#toByteArray())()  Trả về một mảng byte chứa biểu diễn hai phần bù của BigInteger này. |
| [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) | [**toString**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#toString())()  Trả về đại diện Chuỗi thập phân của BigInteger này. |
| [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) | [**toString**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#toString(int))(int radix)  Trả về đại diện Chuỗi của BigInteger này trong cơ số đã cho. |
| static [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**valueOf**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#valueOf(long))(long val)  Trả về một BigInteger có giá trị bằng với giá trị được chỉ định long. |
| [**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) | [**xor**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html#xor(java.math.BigInteger))([**BigInteger**](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html) val)  Trả về một BigInteger có giá trị là (this ^ val). |

-------------------------------------- CPP -------------------------------------------------------  
  
#define long long long  
#define f1(i,n) for (int i=1; i<=n; i++)  
#define f0(i,n) for (int i=0; i<n; i++)  
  
typedef vector<int> vi;  
const int BASE=10000;  
  
**void fix(vi &a){**  
    a.push\_back(0);  
    f0(i,a.size()-1) {   
        a[i+1]+=a[i]/BASE; a[i]%=BASE;   
        if (a[i]<0) { a[i]+=BASE; a[i+1]--; }  
    }  
    while (a.size()>=2 && a.back()==0) a.pop\_back();  
}  
  
**void operator += (vi &a, const vi &b) {**  
    a.resize(max(a.size(), b.size()));  
    f0(i,b.size()) a[i]+=b[i];  
    fix(a);  
}  
  
**vi operator \* (const vi &a, const vi &b){**  
    vi c(a.size()+b.size());  
    f0(i,a.size()) f0(j,b.size()) c[i+j]+=a[i]\*b[j];  
    return fix(c), c;  
}  
  
**vi to\_vi(int x) // x<BASE**  
    { return vi(1,x); }  
  
**void operator -= (vi &a, const vi &b)**  
    { f0(i,b.size()) a[i]-=b[i]; fix(a); }  
**void operator \*= (vi &a, int x) // x<BASE**  
    { f0(i,a.size()) a[i]\*=x; fix(a); }  
**vi operator + (vi a, const vi &b)**  
    { a+=b; return a; }  
**vi operator - (vi a, const vi &b)**  
    { a-=b; return a; }  
**vi operator \* (vi a, int x) // x<BASE**  
    { a\*=x; return a; }  
      
**bool operator < (const vi &a, const vi &b){**  
    if (a.size() != b.size()) return a.size()<b.size();  
    for (int i=a.size()-1; i>=0; i--)  
    if (a[i]!=b[i]) return a[i]<b[i];  
    return false;  
}  
  
**void divide(const vi &a, int x, vi &q, int &r) {**  
    q.clear(); r=0;  
    for (int i=a.size()-1; i>=0; i--) {  
        r=r\*BASE+a[i];  
        q.push\_back(r/x); r%=x;  
    }  
    reverse(q.begin(), q.end());  
    fix(q);  
}  
  
**vi operator / (const vi &a, int x)**  
    { vi q; int r; divide(a, x, q, r); return q; }  
**int operator % (const vi &a, int x)**  
    { vi q; int r; divide(a, x, q, r); return r; }  
      
**istream& operator >> (istream& cin, vi &a){**  
    string s; cin >> s;  
    a.clear(); a.resize(s.size()/4+1);  
    f0(i,s.size()) {  
        int x = (s.size()-1-i)/4; // <- log10(BASE)=4  
        a[x]=a[x]\*10+(s[i]-'0');  
    }  
    return fix(a), cin;  
}  
  
**ostream& operator << (ostream& cout, const vi &a){**  
    printf("%d", a.back());  
    for (int i=(int)a.size()-2; i>=0; i--)  
    printf("%04d", a[i]);  
    return cout;  
}  
  
**void test\_fib(int n){**  
    vi a=to\_vi(1), b=to\_vi(1);  
    f1(i,n/2) {  
        a+=b; b+=a;  
        cout << "F[" << i\*2+1 << "]=" << a << endl;  
        cout << "F[" << i\*2+2 << "]=" << b << endl;  
    }  
}  
  
**void test\_fact(int n){**  
    vi P=to\_vi(1);  
    f1(i,n) {  
        P\*=i;  
        cout << i << "!= " << P << endl;  
    }  
}  
  
**void test\_divide(){**  
    vi a; int x;  
    for (;;){  
        cout << "Input a big integer and a small integer (<10000)" << endl;  
        if (cin >> a >> x); else break;  
        cout << "a=" << a << " x=" << x << endl;  
        vi q=a/x; int r=a%x;  
        cout << "a/x=" << q << "; a%x=" << r << endl;  
        vi a0 = q\*to\_vi(x)+to\_vi(r);  
        assert(a0==a && !(a0<a) && !(a0>a));  
    }  
}  
  
**main(){**  
    cout << "Press Enter to run test\_fib()" << endl;  
    cin.ignore(1); test\_fib(100);  
    cout << "Press Enter to run test\_fact()" << endl;  
    cin.ignore(1); test\_fact(100);  
    cout << "Press Enter to run test\_divide()" << endl;  
    cin.ignore(1); test\_divide();  
}

**====BLABLO===**

**-bitwise**

//Kiem tra xem n có phai giai thua cua 2 hay không:

ll n=3;

n > 0?(n & (n - 1)) == 0: false;

return 0;

//Kiem tra x và y có cùng dau hay không:

(x ^ y) >= 0;

* **Python**:

Input list: a = list(map(int, input().split()))

Length: len(string)

Lam tron: round(x, b) --- b:so can lam tron

Ham: def temham(\*num): \*truyen nhieu so