Graph

1.ArticulationPoint :

```
const ll N = 1e5+7;
const ll inf = 1e9+7;
vector<ll>g[N];
bool vis[N];
ll in[N];
ll low[N];
ll point[N];
ll n,m;
ll timer=0,ans=0;
ll dfs(ll u,ll p=-1)
{
    timer++;
    vis[u]=true;
    in[u]=timer;
    low[u]=in[u];
    ll child=0;
    for(auto v : g[u] )
    {
        if(p==v)continue;
        if(!vis[v])
        {
            dfs(v,u);

            low[u]=min(low[u],low[v]);

            if(low[v]>=in[u]&&p!=-1&&!point[u])
            {
                point[u]=true;
                ans++;
            }
            child++;
        }
        else
            low[u]=min(low[u],in[v]);
    }
    if(p==-1&&child>1)// if u is a root node
        ans++;
}
```

2. Bridges :

```
const ll N = 1e5+7;
vector<ll>g[N];
bool vis[N];
ll in[N];
ll low[N];
ll col[N];
ll n,m;
ll timer=0;
vector<pair<ll,ll> >bridgEdge;
ll dfs(ll u,ll p)
{
    timer++;
    vis[u]=true;
    in[u]=timer;
    low[u]=timer;
    for(auto v : g[u] )
    {
        if(p==v)continue;

        if(!vis[v])
        {
            dfs(v,u);

            if(low[v]>in[u])
            {
               bridgEdge.push_back({v,u});
            }
            low[u]=min(low[u],low[v]);
        }
        else
        {
            low[u]=min(low[u],in[v]);
        }
    }
}
```

3.Bellman Ford :

```
const ll N=10010;
ll edges[N][3];
ll n,m;
void BellmanFord(ll s)
{
    // maske all dis[i] = inf
    ll dis[n+5];
    for(ll i=1; i<=n; i++)
        dis[i]=1e18;
    dis[s]=0;
    for(ll i=0; i<n-1; i++)
    {
        for(ll j=0; j<m; j++)
        {
            ll x=edges[j][0];
            ll y=edges[j][1];
            ll w=edges[j][2];
            if(dis[x]+w<dis[y])
            {
                dis[y]=dis[x]+w;
            }
        }
    }
    for(ll j=0; j<m; j++)
    {
        ll x=edges[j][0];
        ll y=edges[j][1];
        ll w=edges[j][2];
        if(dis[x]+w<dis[y])
        {
            cout<<"graph contain negative cycle"<<"\n";
            return;
        }
    }
    for(ll i=1; i<=n; i++)
        cout<<dis[i]<<ss;
}
```

4.Floyd_Warshall :

```
const ll N=1010;
ll n,m;
ll dis[N][N];
void Floyd_Warshall()
{
    // make_all dis[i][j] = inf
    for(ll k=1; k<=n; k++)
    {
        for(ll i=1; i<=n; i++)
        {
            for(ll j=1; j<=n; j++)
            {
                if(i==j)
                    dis[i][j]=0;
                dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
            }
        }
    }
}
```

5.Bfs :

```
int dx[] = {0 , 0 , -1 , 1 , -1 , -1 , 1 , 1 , 0} ;
int dy[] = {1 ,-1 , 0 , 0 , -1 , 1 , -1 , 1 , 0} ;
int dx[] = {0 , 0 , -1 , 1} ; // right , left , forward , backward
int dy[] = {1 , -1 , 0 , 0} ;
bool check(int x , int y) {
    if(x <= n && y <= m && x >= 1 && y >= 1)
        return 1 ;
    return 0 ;
}
vll v[10];
ll visited[10];
ll level[10];
void bfs(ll u)
{
    visited[u]=1;
    level[u]=0;
    queue<ll>q;
    q.push(u);
```

```
   while(!q.empty())
   {
       u=q.front();
       visited[u]=1;
       q.pop();

       for(ll i=0; i<zz(v[u]); i++)
       {
           ll p=v[u][i];
           if(!visited[p])
           {
               q.push(p);
               visited[p]=1;
               level[p]=level[u]+1;

           }
       }
   }

}
```

6.Dijkstra :

```
const ll N = 100010;

vector<pii>adj[N];
ll vis[N];
ll dis[N];
ll inf = 1e16;


void dij(ll u)
{

  // make all dis[i] = inf
  priority_queue<pii>q;
  dis[u] = 0;
  q.push({0LL, u});
```

```
    while(!q.empty())
    {

        u = q.top().second;
        q.pop();

        for(pii p2 : adj[u])
        {
            ll v = p2.fi;
            ll w = p2.se;

             if(dis[u] + w < dis[v])
             {
                 dis[v] = dis[u] + w;
                 q.push({-dis[v], v});
             }
        }


    }
}
```

7.Mst :

```
struct edge
{
    ll a;
    ll b;
    ll w;
};

edge ar[10010];

ll pr[10000];


bool cmp(edge p1,edge p2)
{
    return p1.w<p2.w;

}

ll find(int k)
{
```

```
    if(pr[k]==-1)
        return k;

    return pr[k]=find(pr[k]);
}

void merge(ll a,ll b)
{
    pr[a]=b;
}
int main()
{

     ios::sync_with_stdio(0);
    cin.tie(0);

    int n,m,a,b,w;
    cin>>n>>m;

    for(ll i=1; i<=n; i++)
    {
        pr[i]=-1;
    }

    for(ll i=0; i<m; i++)
    {
        cin>>ar[i].a>>ar[i].b>>ar[i].w;

    }
    sort(ar,ar+m,cmp);
    ll sum=0;
    for(ll i=0; i<m; i++)
    {
        a=find(ar[i].a);
        b=find(ar[i].b);

        if(a!=b)
        {
            sum+=ar[i].w;
            merge(a,b);
        }
    }

    cout<<sum<<nn;}
```

8.Lca :

```
const ll N = 10010;
vll adj[N];
ll n,m,q,tin[N],tout[N],bl[20][N],timer = 0;

void dfs(ll u, ll p)
{
    tin[u] = ++timer;
    bl[0][u] = p;
    for(ll i = 1; i < 20; i++)
    {
        bl[i][u] = bl[i-1][bl[i-1][u]];
    }
    for(ll j : adj[u])
        if(j != p)
            dfs(j,u);
    tout[u] = ++timer;
}

bool is_ancestor(ll u, ll v)
{
    return tin[u] <= tin[v] and tout[u] >= tout[v];
}

ll lca(ll u, ll v)
{
    if(is_ancestor(u,v))
        return u;
    if(is_ancestor(v,u))
        return v;
    for(ll i = 19; i >= 0; i--)
    {
        if(!is_ancestor(bl[i][u], v))
            u = bl[i][u];
    }
    return bl[0][u];
}
```

9.Top_Sort :

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

10.BinaryLifting :

```
vector<int>adj[N];
ll n;
ll BL[20][N];
void dfs(ll u,ll p)
{
    BL[0][u] = p;
    for(ll j : adj[u])
    {
        if(j==p)
            continue;
        dfs(j,u);
    }
}
int main()
{

    ll q; cin>>n>>q;
```

```
for(int i = 2; i<=n; i++)
{
    int num;
    cin>>num;
    adj[num].push_back(i);
    adj[i].push_back(num);
}
memset(BL, -1, sizeof(BL));
dfs(1,-1);
for(ll power  = 1; power < 20; power++)
{
    for(ll node = 1; node <= n; node++)
    {
        // node er 2 ^ power ke ase
        // eita ber korar jonno ki ki jana lagbe ???

        // amader node 2 ^ (power - 1) information jana lagbe
        if(BL[power-1][node] != -1 && BL[power-1][BL[power-1][node]]!=-1)
        {
            BL[power][node] = BL[power-1][BL[power-1][node]];
        }
    }

}
while(q--)
{
    ll x,k;
    cin>>x>>k;
    ll ans = -1;
    for(ll i = 19; i >= 0; i--)
    {
        if((1 << i) <= k)
        {
            if(BL[i][x] != - 1)
            {
                k -= (1 << i);
                x = BL[i][x];
            }
        }
    }
    if(k==0)
    {
        ans = x;       }
   cout<<ans<<"\n";   }}
```

String

1.Trie

```
struct node
{

    bool endmark;

    node *next[10];

    string name;

    node()
    {
        endmark = false;

        for(ll i=0; i < 10; i++)
        {
            next[i] = NULL;
        }
    }

};
node *root = new node();
void Insert(string &name, string str, ll len)
{
    node *cur = root;

    for(ll i = 0; i< len; i++)
    {
        ll id = str[i] - '0';

        if(cur -> next[id] == NULL)
        {

            cur -> next[id] = new node();
        }

        cur = cur->next[id];
    }
```

```
    cur-> endmark = true;
    cur->name = name;

    //cout<<cur->name<<nn;
}

string Search(string &str, ll len)
{
    node *cur = root;

    for(ll i = 0; i < len; i++)
    {
        ll id = str[i] - '0';
        if(cur->next[id] == NULL)
            return  "-1";
        cur = cur->next[id];
    }
    if(cur->endmark)
    {
        return cur->name + " " + str;
    }
    while(cur->endmark == 0)
    {
        bool flag = 0;

        for(ll i = 0; i < 10; i++)
        {
            if(cur ->next[i])
            {
                str.pb(char(i + '0'));
                cur = cur->next[i];

                flag = 1;
                break;
            }
        }
        if(flag == 0)
            return "-1";
    }

    return cur->name + " " + str;
}
```

2. Hashing :

```
const ll N = 1e6 + 9;
const ll B = 29;
const ll MOD = 1e9 + 7;

ll power[N], inverse[N], _B;
ll prefix[N];

string s;


ll bigMod(ll b, ll e)
{
    if(e == 0)
        return 1;
    if(e == 1)
        return b;

    if(e%2==0)
    {
        ll ret = bigMod(b, e/2);
        return ret * ret % MOD;
    }
    else
        return b * bigMod(b, e - 1) % MOD;
}

// 0 index range hash
ll range_hash(ll i,ll j)
{
    ll ret = prefix[j+1] - prefix[i];
    if(ret < 0)
        ret += MOD;
    ret = ret * inverse[i] % MOD;

    return ret;
}


void init()
{
    power[0] = 1;
    for(ll i=1; i < N ; i++)
```

```
    {
        power[i] = (power[i-1] * B ) % MOD;
    }

    _B = bigMod(B, MOD - 2);
    inverse[0] = 1;
    for(ll i=1; i < N; i++)
    {
        inverse[i] = (inverse[i-1] * _B) % MOD;
    }

}



int main()
{

//    ios::sync_with_stdio(0);
//    cin.tie(0);

    init();

    cin>>s;
    ll n = zz(s);

    prefix[0] = 0;
    for(ll i = 0; i < zz(s) ; i++)
    {
        prefix[i+1] = (prefix[i] + (s[i] - 'a' + 1) * power[i]) % MOD;
    }

    for(ll i=0; i<n-1; i++)
    {
        if(range_hash(0,i) == range_hash(n-(i+1),n-1))
        {
            cout<<i+1<<" ";
        }
    }
    cout<<nn;


}
```

Data Structure

1. SegmentTree :

```
const ll N = 200010;
ll n,q;
ll a[N];
ll Tree[N*4];
void build(ll node, ll b, ll e)
{
    if(b == e)
    {
        Tree[node] = a[b];
        return;
    }
    ll m = (b+e)/2;
    build(node * 2, b, m);
    build(node * 2 + 1, m+1, e);
    Tree[node] = Tree[node * 2] + Tree[node * 2 + 1];
}
void update(ll node, ll b, ll e,ll idx,ll value)
{
    if(b > idx || e < idx)
        return;
    if(b == e && b == idx)
    {
        Tree[node] = value;
        return;
    }
    ll m = (b+e)/2;
    update(node * 2, b, m,idx, value);
    update(node * 2 + 1, m+1, e,idx, value);

    Tree[node] = Tree[node * 2] + Tree[node * 2 + 1];
}
ll query(ll node, ll b, ll e, ll l, ll r)
{
    if(e < l || b > r)
        return 0;
    if(b >= l and e <= r)
        return Tree[node];
    ll m = (b+e)/2;
    return query(node * 2, b, m, l, r) + query(node * 2 + 1, m+1, e, l, r);
}
```

```
// class
/*
    1. constructor
    2. init
    3. build
    4. take careod INF9 and INF18
*/
#define INF9        2147483647
#define INF18       9223372036854775806
template <typename T> struct SegmentTree
{
    vector <T> seg;
    vector <T> lazy;
    vector <T> ar;
    int type, up;
    SegmentTree()
    {
        type = 0;
        up = 0;
    }
    SegmentTree(int tp, int u)
    {
        type = tp;
        up = u;
    }
    void Init(int N)
    {
        seg.assign(N << 2, 0);
        lazy.assign(N << 2, 0);
    }
    void Init(vector <T> &s)
    {
        Init(s.size() + 1);
        ar = s;
    }
    void PushDown(int cur, int left, int right)
    {
        if(type==0)
        {
            if (up == 1) seg[cur] += (right - left + 1) * lazy[cur];
            else seg[cur] = (right - left + 1) * lazy[cur];
        }
        else if(type==1)
        {
```

```
        if(up == 1) seg[cur] +=lazy[cur];
        else seg[cur]=lazy[cur];
    }
    else
    {
        if(up == 1) seg[cur] +=lazy[cur];
        else seg[cur]=lazy[cur];
    }

    if (left != right)
    {
        if (up == 1)
        {
            lazy[cur << 1] += lazy[cur];
            lazy[cur << 1 | 1] += lazy[cur];
        }
        else
        {
            lazy[cur << 1] = lazy[cur];
            lazy[cur << 1 | 1] = lazy[cur];
        }
    }
    lazy[cur] = 0;
}
T Merge(T x, T y)
{
    if (type == 0) return x + y;
    if (type == 1) return max(x, y);
    if (type == 2) return min(x, y);
}
void Build(int cur, int left, int right)
{
    lazy[cur] = 0;
    if (left == right)
    {
        seg[cur] = ar[left];
        return;
    }
    int mid = (left + right) >> 1;
    Build(cur << 1, left, mid);
    Build(cur << 1 | 1, mid + 1, right);
    seg[cur] = Merge(seg[cur << 1], seg[cur << 1 | 1]);
}
void Update(int cur, int left, int right, int pos, T val)
```

```
    {
        Update(cur, left, right, pos, pos, val);
    }

    void Update(int cur, int left, int right, int l, int r, T val)
    {
        if (lazy[cur] != 0) PushDown(cur, left, right);
        if (l > right || r < left) return;
        if (left >= l && right <= r)
        {
            if (up == 0) lazy[cur] = val;
            else lazy[cur] += val;
            PushDown(cur, left, right);
            return ;
        }
        int mid = (left + right) >> 1;
        Update(cur << 1, left, mid, l, r, val);
        Update(cur << 1 | 1, mid + 1, right, l, r, val);
        seg[cur] = Merge(seg[cur << 1], seg[cur << 1 | 1]);
    }
    T Query(int cur, int left, int right, int l, int r)
    {
        if (l > right || r < left)
        {
            if (type == 0) return 0;
            if (type == 1) return -INF18;
            if (type == 2) return INF18;
        }
        if (lazy[cur] != 0) PushDown(cur, left, right);
        if (left >= l && right <= r) return seg[cur];
        int mid = (left + right) >> 1;
        T p1 = Query(cur << 1, left, mid, l, r);
        T p2 = Query(cur << 1 | 1, mid + 1, right, l, r);
        return Merge(p1, p2);
    }
};
//for sum = 0, max = 1, min = 2, for assignment update send 0 or 1 for increment.
SegmentTree <ll> tree1(2, 1);
```

3. Mo's algorithm :

```
const ll maxn=1e4;
int k,ans[maxn],a[maxn],sum;
struct query
{
    ll index,l,r;
    bool operator < (const query &other) const
    {
        ll block_own = l/k;
        ll block_other = other.l/k;

        if(block_own==block_other)
        {
            return r < other.r;
        }
        return block_own < block_other;
    }
} mos[maxn];
void add(ll index)
{
    sum+=a[index];
}
void remove(ll index)
{
    sum-=a[index];
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll n;
    cin>>n;
    k=sqrt(n);
    for(ll i=0; i<n; i++)
    {
        cin>>a[i];
    }
    ll q;
    cin>>q;
    for(ll i=0; i<q; i++)
    {
        cin>>mos[i].l>>mos[i].r;
```

```
        mos[i].index=i;
    }
    sort(mos,mos+q);
    ll L=0,R=-1;
    for(ll i=0; i<q; i++)
    {
        while(L<mos[i].l) remove(L++);
        while(L>mos[i].l) add(--L);

        while(R>mos[i].r) remove(R--);
        while(R<mos[i].r) add(++R);

        ans[mos[i].index] = sum;
    }

    for(ll i=0; i<q; i++)
    {
        cout<<ans[i]<<ss;
    }
    cout<<nn;

}
```

4. PBDS :

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using    namespace __gnu_pbds;
typedef tree<int, null_type, less<ll>, rb_tree_tag, tree_order_statistics_node_update>
ordered_set;

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    ordered_set os;
    for(ll i=1;i<=5;i++) os.insert(i);
//  how many numbers are smaller than a given value(7)
   cout<<os.order_of_key(9)<<nn;
//  if the given numbers are sorted in ascending order, what is the k'th number
   cout<<*os.find_by_order(4)<<nn;


}
```

Number Theory

1.PrimeFactor :

```
const ll N=30;
ll mod=1e9+7;
ll spf[N];
void sieve()
{
    // marking smallest prime factor for every
    // number to itself

    for(ll i=1; i<N; i++)
        spf[i]=i;
    // marking SPF for all numbers divisible by i
    for(ll i=2; i*i<N; i++)
    {
        if(spf[i]==i)
        {
            for(ll j=i*i; j<N; j+=i)
            {
                if(spf[j]==j)
                    spf[j]=i;
            }
        }
    }

}
vll getfact(ll x)
{
    vll fact;

    while(x!=1)
    {
        fact.pb(spf[x]);
        x/=spf[x];
    }

    return fact;
}
```

2.nPr_nCr :

```
const ll N = 5e5 + 7, mod = 998244353 ;
ll POW(ll a, ll b, ll mod)
{
    a %= mod;
    ll r = 1;
    for(ll i = b; i > 0; i >>= 1)
    {
        if(i & 1)
            r = (r * a) % mod;
        a = (a * a) % mod;
    }
    return r;
}
ll f[N];
ll nCr(ll n, ll r)
{
    if(n < r)
        return 0;
    return f[n] * POW(f[n - r] * f[r], mod - 2, mod) % mod;
}
ll nPr(ll n, ll r)
{
    return nCr(n, r) * f[r] % mod;
}
void init()
{
    f[0] = 1;
    for(ll  i = 1; i < N; i++)
    {
        f[i] = (f[i - 1] * i) % mod;
    }

}
```

3. Notes :

1. if n is odd all the divisor of n will be odd.
2. if d divide n than d divide also (n-d).
3. if n is even and is not a power of 2, it means that n has an odd divisor.

// Permutation..

1 . for n objects how many distinct permutations exists ?
$n (n - 1) (n - 2) (n - 3)...(1) = n!$

2. if an integer $n >= 0$, n factorial denoted n! .. is defined as

$0! = 1$
$n! = n (n - 1) (n - 2)..(1)$, for $n >= 1$

3. if there are n distinct objects , the number of permutations of size k , with $1 <= k <= n$, for the n objects is

$P(n , k) = n (n - 1) (n - 2) (n - 3) ... (n - k + 1)$

or

$P(n , k) = n! / (n - k)!$

4. $P(n , n) = n!$

5 . permutations with repeatation :

If we have n1 indistingushable objects of a first type , n2 indistingushable objects of a second type ,, and Nr indistingushable objects of kth type , where n1 + n2 + .. + nr = n , then there are ,

$P(n , n) = n! / (n1!n2!..nk!)$

// GCD

1 : GCD property :

For non-negative integers a and b, where a and b are not both zero, provable by considering the Euclidean algorithm in base n :

// gcd((n^a) − 1, (n^b) − 1) = (n^gcd(a,b)) − 1

// If gcd(x,n)=1, then gcd(n−x,n)=1

2 : GCD and LCM relations , It is based on the formula that ,

// LCM(A , B) × GCD(A , B) = A × B

3 : Number of divisors (NOD) of a number N can be calculated using Prime power factorization.

Let , N = P1^a1 * P2^a2 * P3^a3 * ... * Pn^ak , is the prime power factorization of a number N , where P is the prime number and

a is number of times occurs that prime number.

Then , NOD(N) defines as :

// NOD(N) = (a1 + 1) * (a2 + 1) * (a3 + 1) * ... * (ak + 1)

4 : Sum of divisors (SOD) of a number N can be calculated using prime power factorization.

Let , N = P1^a1 * P2^a2 * P3^a3 * ... * Pn^ak , s the prime power factorization of a number N , where P is the prime number and

a is number of times occurs that prime number.

Then , SOD(N) defines as :

// SOD(N) = (P1^0 + P1^1 + p1^2 + ... + P1^a1) * (P2^0 + P2^1 + P2^2 + ... + P2^a2) * (Pk^0 + Pk^1 + ... + Pk^ak)

5 : Logarithm base calculation :

//logB(x)=logC(x) / logC(B)

6 : Trailing zeros in N! in decimal number system ,

Let , a is frequency of 2 in N! prime factorisation and b is frequency of 5 in N! prime factorisation..

Then ,

// Number of Trailing zeros = min(a , b) ;

7 : Trailing zeros in N! in different base system :

We find number of trailing zero using the following steps:

Factorize the base B

If $B = p_1^{a_1} \times p_2^{a_2} \ldots \times p_k^{a_k}$, then find out occurance of x_i=factorialPrimePower(p_i).

But we can’t use x_i directly. In order to create B we will need to combine each p_i into $p_i^{a_i}$. So we divide each x_i by a_i.

Number of trailing zero is MIN($x_1, x_2, \ldots, x_k$).

8 : Leading Numbers :

We need to execute the following steps to find the first K leading digits of a number x (in our problem x=N!):

Find the log value of the number whose leading digits we are seeking. $y=\log_{10}(x)$.

Decompose y into two parts. Integer part p and fraction part q.

The answer is $\lfloor 10^q \times 10^{K-1} \rfloor$.

9 : Euler Phi Extension Theorem

Theorem:

Given a number N, let d be a divisor of N. Then the number of pairs a,N, where $1 \le a \le N$ and $\gcd(a,N)=d$, is $\phi(N / d)$.

10 : Euler Phi Divisor Sum Theorem

Theorem:

For a given integer N, the sum of Euler Phi of each of the divisors of N equals to N, i.e, $N = \sum_{d \mid N} \phi(d)$

11 : Eulers Totient Function (Eulers Phi):

Given an integer N, how many numbers less than or equal N are there such that they are coprime to N?

A number X is coprime to N if gcd(X,N) = 1.

// $\phi(n) = n \times ((p_1 - 1) / p_1) \times ((p_2 - 1) / p_2) \ldots \times ((p_k - 1) / p_k)$

12 : Given two sequences of numbers $A=[a_1,a_2,\ldots,a_n]$ and $M=[m_1,m_2,\ldots,m_n]$, find the smallest solution to the following linear congruence equations if it exists:

$x \equiv a_1 \pmod{m_1}$

$x \equiv a_2 \pmod{m_2}$

…

$x \equiv a_n \pmod{m_n}$

13 : GCD Sum Function – g(n)

Given a positive integer N, find the value of g(N), where

$g(n) = \gcd(1,n) + \gcd(2,n) + \gcd(3,n) + \cdots + \gcd(n,n) = \sum_{i=1}^{n} \gcd(i,n)$

// there is a direct formula for calculating the value of g(n).

If the prime factorization of n is $p1^{a1} \times p2^{a2} \times \ldots * pk^{ak}$, then

$g(n) = i=0 \text{ to } k \prod ((ai + 1)pi^{ai}) – (ai * pi^{(ai−1)})$

14 : Sum of Co-prime Numbers of an Integer

Problem

Given a number N, find the sum of all numbers less than or equal to N that are co-prime with N.

Let us define a function f(n), which gives us sum of all numbers less than or equal to n that are co-prime to n.

Then we can calculate the value of f(n) with the following formula:

// f(n)= (phi(n) * n) / 2

Eulers Phi Properties..

1. If p is a prime number, then $gcd(p,q)=1$ for all $1 \le q<p$. Therefore we have:
$\phi(p)=p−1$.
2. If p is a prime number and $k \ge 1$, then there are exactly pk/p numbers between 1 and pk that are divisible by p. Which gives us:
$\phi(p^k)=p^k−p^k−1$.
3. If a and b are relatively prime, then:
$\phi(ab)=\phi(a) \cdot \phi(b)$.
4. In general, for not coprime a and b, the equation
$\phi(ab)=\phi(a) \cdot \phi(b) \cdot d\phi(d)$
with $d=gcd(a,b)$ holds.
5.

Fibonacci properties...

1 . Cassini's identity:
$(Fn−1)*(Fn+1) − F2n= (−1)^n$

2 . The "addition" rule:
$Fn+k=(Fk * Fn+1) + (Fk−1 * Fn)$

3 . Applying the previous identity to the case k=n, we get:
$F2n= Fn * (Fn+1 + Fn−1)$

4 . From this we can prove by induction that for any positive integer k, Fnk is multiple of Fn.

5 . The inverse is also true: if Fm is multiple of Fn, then m is multiple of n

6 . GCD identity:
$GCD(Fm,Fn) = FGCD(m,n)$