# Bfs

```
const int inf = 1e9; const
int N = 1e5 + 9;
vector<int> g[N]; int32_t
main() {   int n, m; cin >>
n >> m; for (int i = 1; i <=
m; i++) {     int u, v; cin >>
u >> v;
g[u].push_back(v);
g[v].push_back(u);
 }
queue<int> q;   vector<int> d(n + 1, inf),
par(n + 1, -1);
 q.push(1);   d[1] =
0;   while
(!q.empty()) {     int
u = q.front();
   q.pop();     for (auto
v: g[u]) {      if (d[u] +
1 < d[v]) {        d[v] =
d[u] + 1;        par[v] =
u;
      q.push(v);
    }
  }
 }
 if (d[n] == inf) {     cout <<
"IMPOSSIBLE\n";     return
0;
 }
 cout << d[n] + 1 << '\n';   int cur =
n;   vector<int> path;   while (cur !=
-1) {      path.push_back(cur);     cur
= par[cur];   } reverse(path.begin(),
path.end());   for (auto u: path) {
cout << u << ' ';
 }
 cout << '\n';
return 0;
}
```

# Cycle detecon

```
const int N = 5e5 + 9;
vector<pair<int, int>> g[N];
int vis[N], par[N], e_id[N];
vector<int> cycle; // simple cycle, contains edge
ids bool dfs(int u) {   if (!cycle.empty()) return 1;
vis[u] = 1;   for (auto [v, id] : g[u]) {     if (v !=
par[u]) {      if (vis[v] == 0) {        par[v] = u;
e_id[v] = id;        if (dfs(v)) return 1;
    }
    else if (vis[v] == 1) {        // cycle here
cycle.push_back(id);        for (int x = u; x !=
v; x = par[x]) {
cycle.push_back(e_id[x]);
      }
      return 1;
    }
  }
```

```
 }
 vis[u] = 2;   return
0; }int32_t main() {
int n, m; cin >> n >>
m;   for (int i = 1; i <=
m; i++) {     int u, v;
cin >> u >> v;     ++u;
++v;
g[u].push_back({v,
i});
 }
 for (int u = 1; u <= n; u++) {     if
(vis[u] == 0 and dfs(u)) {       cout <<
cycle.size() << '\n';       for (auto x:
cycle) cout << x - 1 << '\n';
return 0;
   }
 } cout << -1 << '\n';
return 0;
}
```

# Dijkstra

```
const int N =
1e5 + 10; const
int INF = 1e9 +
10; typedef
pair<int,int>iPai
r;
vector<pair<int,
int>>g[N];
vector<int>dist(N, INF);

void dijkstra(int s){
vector<int>vis(N, 0);
   priority_queue<iPair, vector<iPair>,
greater<iPair>>pq;

   dist[s] = 0;     pq.push({0,s}); // first cost
second node     while(!pq.empty()){
int cost  = pq.top().first;        int node =
pq.top().second;        pq.pop();

      for(auto child : g[node]){
int edge_cost = child.second;
int adj_node = child.first;

         if(cost + edge_cost < dist[adj_node]){
dist[adj_node] = cost + edge_cost;
pq.push({dist[adj_node], adj_node});
         }
      }
   }

}

int main() {     int node,
edge;     cin >> node >>
edge;
```

**AUST_Outliers**

```
   while(edge--){
int from, to, cost;
cin >> from >> to
>> cost;


      g[from].push_back({to,cost});
   }
   int source = 1;
dijkstra(source);
for(int i = 1; i <=
node; i++){
      cout<<source<<" to "<<i<<"
node
"<<dist[i]<<'\n';
   }
   cout<<'\n';
}
const int N = 3e5 + 9, mod =
998244353; int n, m;
vector<pair<int, int>> g[N],
r[N];
vector<long long>
dijkstra(int s, int t,
vector<int> &cnt) {
const long long inf =
1e18;
 priority_queue<pair<long long,
int>, vector<pair<long long, int>>,
greater<pair<long
long, int>>> q;
 vector<long long> d(n + 1, inf);
vector<bool> vis(n + 1, 0);
 q.push({0, s});
d[s] = 0;
 cnt.resize(n + 1, 0); // number of shortest
paths
 cnt[s] = 1;
while(!q.empty()) {
auto x = q.top();
   q.pop();    int u =
x.second;    if(vis[u])
connue;    vis[u] = 1;
for(auto y: g[u]) {      int v
= y.first;      long long w =
y.second;      if(d[u] + w <
d[v]) {       d[v] = d[u] + w;
      q.push({d[v], v});
cnt[v] = cnt[u];
    } else if(d[u] + w == d[v]) cnt[v] = (cnt[v] +
cnt[u]) % mod;
   }
 }
 return d;
}
int u[N], v[N], w[N]; int32_t
main() {
 int s, t;
 cin >> n >> m >> s >> t;   for(int i =
1; i <= m; i++) {  cin >> u[i] >> v[i] >>
```

**AUST_Outliers**

```
w[i];
g[u[i]].push_back({v[i],
w[i]});
r[v[i]].push_back({u[i],
w[i]});   } vector<int>
cnt1, cnt2;   auto d1 =
dijkstra(s, t, cnt1);   auto
d2 = dijkstra(t, s, cnt2);
long long ans = d1[t];
for(int i = 1; i <= m; i++)
{  int x = u[i], y = v[i];
long long nw = d1[x] +
w[i] + d2[y];
   if(nw == ans && 1LL * cnt1[x] *
cnt2[y] % mod
== cnt1[t]) cout << "YES\n";
   else if(nw - ans + 1 < w[i]) cout
<< "CAN " << nw - ans + 1 << '\n';
else cout << "NO\n";
 }
 return 0;
}
```

# Topological sort

```
const int
N = 1e5
+ 9;
vector<i
nt>
g[N]; bool vis[N];
vector<int> ord;
void dfs(int u) {
vis[u] = true;   for
(auto v: g[u]) {
   if (!vis[v]) {
    dfs(v);
   }
 }
 ord.push_back(u);
}
int32_t main() {   int n,
m; cin >> n >> m;
while (m--) {  int u, v;
cin >> u >> v;
g[u].push_back(v);
 }
 for (int i = 1; i <= n; i++) {
  if (!vis[i]) {  dfs(i);
   }
 }
 reverse(ord.begin(), ord.end());

 // check is feasible   vector<int>
pos(n + 1);   for (int i = 0; i < (int)
ord.size(); i++) {  pos[ord[i]] = i;
 }
```

```
 for (int u = 1; u
  <= n; u++) {  for
  (auto v: g[u]) {
  if (pos[u] >
  pos[v]) {  cout
  <<
  "IMPOSSIBLE\n
  ";  return 0;
     }
   }
 } // print the order
for (auto u: ord) cout
<< u << ' ';   cout <<
'\n';   return 0;
}
```

# Starng Template

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_conta
iner.hpp>
#include<ext/pb_ds/tree_policy.
hpp> using namespace
__gnu_pbds; using namespace
std; typedef long long ll ;
#define PI    acos(-
1.0)  #define  endl
"\n"
gp_hash_table<i
nt,int>mp;
template <typename T> using o_set = tree<T,
null_type, less<T>, rb_tree_tag,
tree_order_stascs_node_update>; int main()
{
        ios_base::sync_with_stdio(false);
        cin.e(NULL); o_set<int> se;
        se.insert(4);   se.insert(2);
        se.insert(5);
 // sorted set se = [2, 4, 5]
 cout << se.order_of_key(5) << '\n'; // number
of elements < 5
 cout << se.order_of_key(6) << '\n'; // number
of elements < 6
 cout << (*se.find_by_order(1)) << '\n'; // if you
imagine this as a 0-indexed vector, what is
se[1]?
 return 0;  int
  test_cases ;  cin >>
  test_cases ;  while
  (test_cases--)
   {
   }
   return 0 ;
}
```

# Double Hashing

```
const int N = 1e5 + 7;


int base1 = 137, base2 = 277, mod1
= 127657753, mod2 = 987654319;

int pref1[N], pref2[N], inv1[N],
inv2[N], pwr1[N], pwr2[N];


int binary_expo(long long n, long
long k, int mod) {

 int ans = 1 % mod; n %= mod; if (n
< 0) n += mod;

 while (k) {  if (k & 1) ans =

  (long long) ans * n % mod;  n

  = (long long) n * n % mod;  k

  >>= 1;

 }

 return ans;

}


void precalculate(){


   pwr1[0] = 1,

  pwr2[0] = 1;  inv1[0]

  = 1, inv2[0] = 1;

  int in_val1 = binary_expo(base1,
mod1 - 2, mod1);

  int in_val2 = binary_expo(base2,
mod2 - 2, mod2);

  //cout << in_val1 << " " << in_val2 << endl;
  for(int i = 1; i <= N; i++){

     pwr1[i] = (1LL * pwr1[i - 1]  * base1) %
mod1;

     pwr2[i] = (1LL * pwr2[i - 1]  * base2) %
mod2;

     inv1[i] = (1LL * inv1[i - 1]  * in_val1) %
mod1;

     inv2[i] = (1LL * inv2[i - 1]  * in_val2) %
mod2;

  }

}


pair<int,int> string_hash(string s){


  ll hash_value1 = 0, hash_value2 = 0;


  for(int i = 1; i <= s.size(); i++){

    hash_value1 += (1LL * s[i - 1] * pwr1[i]);

    hash_value1 %= mod1;  hash_value2 +=

    (1LL * s[i - 1] * pwr2[i]) ;  hash_value2

    %= mod2;

  }

  return {hash_value1, hash_value2};

}

void build_hash(string s){


  for(int i = 1; i <= s.size(); i++){

     pref1[i] = (pref1[i - 1] + (1LL * s[i - 1]  *
pwr1[i]) % mod1) % mod1;

     pref2[i] = (pref2[i - 1] + (1LL * s[i - 1]  *
```

```
pwr2[i]) % mod2) % mod2;
    }
}
pair<int,int> get_hash(int l, int
  r, int n){  assert((0 <= l) && (l
  <= r) && (r < n));  l++, r++;

   int hash_value1 = ( 1LL * (pref1[r]
- pref1[l - 1] + mod1) * inv1[l]) %
mod1;
   int hash_value2 = ( 1LL * (pref2[r]
- pref2[l - 1] + mod2) * inv2[l]) %
mod2;

    return {hash_value1,
   hash_value2};
}
pair<int,int> merge_hash(int l1, int
r1, int l2, int r2, int n){
   assert(0 <= l1  && l1 <= r1 && r1
< l2 && l2 <= r2 && r2 < n);

    pair<int,int> h1 = get_hash(l1, r1,
   n);  pair<int,int> h2 = get_hash(l2,
   r2, n);

    int pw = r1 - l1 + 1;

   int hash_value1 = (h1.first + ((1LL
* pwr1[pw] * h2.first) % mod1))%
mod1;
   int hash_value2 = (h1.second + ((1LL *
pwr2[pw] * h2.second) % mod2)) % mod2;

    return {hash_value1, hash_value2};

}
 //must call precalculate and build_hash first
 and Change N ;
```

## //Generator

```
ll rnd(ll low, ll high) {

   static mt19937
rng(chrono::steady_clock::no
w().time_since_epoch().coun
t());

uniform_int_distribution<ll>
dist(low, high);

   return dist(rng);
}
int main(){

   srand(time(0));
}
```

# Segment Tree

```
const int N = 1e5 + 7; ll
t[4*N], lazy[4*N]; int
arr[N]; void push(int
node, int start, int end){
        if(lazy[node] == 0){
                return;
        }
t[node] = t[node] + (lazy[node] *
(end - start +
1));
        if(start != end){
lazy[node * 2] += lazy[node] ;
lazy[node * 2 + 1] += lazy[node]
        }
        lazy[node] = 0;
}


void build(int node, int start, int
end){
        lazy[node] = 0;
if(start == end){
        t[node] =
arr[start];
                return ;
        }
        int mid = (start + end) / 2;
build(node*2, start, mid);
build(node*2 + 1, mid + 1, end);
t[node] = t[node * 2] + t[node*2 + 1];


}
void update(int node, int start, int end, int i, int
j, int value){
        push(node, start, end);
if(start > j || end < i)
                return;
        if(start >= i && j >= end){
                lazy[node] = value;
push(node, start, end);
                return;
        }
        int mid = (start + end) / 2;
update(node*2, start, mid, i, j, value );
        update(node*2 + 1, mid + 1, end, i, j,
value);  t[node] = t[node * 2] + t[node*2 +
1];
}
ll query(int node, int start, int end, int i, int j){
        push(node, start, end);
if(start > j || end < i)
                return 0;
        if(i <= start && end <= j){
                return t[node];
        }
        int mid = (start + end) / 2;
return query(node*2, start, mid, i, j) +
```

```
        query(node*2 + 1, mid + 1,
        end, i, j);
        }
```

# Prime factorizaon using seive

```
const int N =
1e6 + 9; int
spf[N];
int32_t
main() {   for
(int i = 2; i <
N; i++) {
spf[i] = i;
 }
 for (int i = 2; i <
N; i++) {     for
(int j = i; j < N; j
+= i) {      spf[j] =
min(spf[j], i);
   }
 }
 int q; cin >> q; // queries q
<= 1e6   while (q--) {
   int n; cin >> n; // find prime
factorizaon of n
<= 1e6
vector<int>
ans;     while
(n > 1) {
ans.push_back(spf[n]);
n /= spf[n];
   }
   for (auto x: ans) cout << x << ' '; cout << '\n';
 }
 return 0;
}
```

# ncr, npr

```
#include<bits/stdc++.h> using
namespace std; const int N = 1e6,
mod = 1e9 + 7; int power(long long
n, long long k) {
 int ans = 1 % mod; n %= mod; if (n < 0) n +=
mod;
 while (k) {     if (k & 1) ans = (long long) ans
* n % mod;     n = (long long) n * n % mod;
k >>= 1;
 }
 return ans;
}
int f[N], invf[N]; int nCr(int
n, int r) {   if (n < r or n < 0)
return 0;
 return 1LL * f[n] * invf[r] % mod * invf[n - r] %
mod;
```

```
} int nPr(int n, int r) {   if
(n < r or n < 0) return 0;
return 1LL * f[n] * invf[n
- r] % mod;
}
int32_t main() {
ios_base::sync_wi
th_stdio(0);
cin.e(0);   f[0] = 1;
 for (int i = 1; i <
N; i++) {     f[i] =
1LL * i * f[i - 1] %
mod;
 }
 invf[N - 1] = power(f[N - 1],
mod - 2);   for (int i = N - 2; i
>= 0; i--) {     invf[i] = 1LL *
invf[i + 1] * (i + 1) % mod;
 }
 cout << nCr(6, 2)
<< '\n';   cout <<
nPr(6, 2) << '\n';
return 0;
}
```

# nCr Using Binomial Theorem

```
const int N = 1010, mod = 1e9 + 7;
int C[N][N]; void ncr() {   C[0][0] =
1;   for (int n = 1; n < N; n++) {
C[n][0] = 1;     for (int k = 1; k <= n;
k++) {
     C[n][k] = (C[n - 1][k - 1] + C[n - 1][k]) % mod;
   }
 }
 cout << C[6][2] << '\n';
return 0;
}
```

# Dp minimum cost

```
int n, m, a[10][10], inf = 1e9 + 7;
int dp[10][10]; int min_cost(int i,
int j) {   if (j > m or i > n) return inf;
if (i == n and j == m) return a[i][j];
if (dp[i][j] != -1) return dp[i][j];
 return dp[i][j] = a[i][j] + min(min_cost(i + 1, j),
min_cost(i, j + 1));
} void path(int i, int j) {   cout << "("
<< i << ", " << j << ") -> ";   if (i == n
and j == m) return;   int right =
min_cost(i, j + 1);   int down =
min_cost(i + 1, j);   if (right <= down)
{     path(i, j + 1);
 }
```

```
els
e {
pa
th
(i
+
1,
j);
 }
}
```

# Dp on acyclic graph

```
#include<bits/stdc++.h>
using namespace std; const
int N = 1e5 + 9; int dp[N];
vector<int> g[N]; int rec(int
u) {  if (dp[u] != -1) return
dp[u];  int ans = 0;  for
(auto v: g[u]) {    ans =
max(ans, 1 + rec(v));
 }
 return dp[u] = ans;
}
// finding sizes of all subtrees
#include<bits/stdc++.h> using
namespace std;

const int N = 1e5 + 9; int sz[N];
vector<int> g[N]; void dfs(int u,
int p) {  sz[u] = 1;  for (auto v:
g[u]) {    if (v != p) {      dfs(v, u);
sz[u] += sz[v];
   }
 }
}
int32_t main() {  int n;
cin >> n;  for (int i = 1; i
< n; i++) {    int u, v; cin
>> u >> v;
g[u].push_back(v);
g[v].push_back(u);
 }
 dfs(1, 0);  for (int i = 1; i
<= n; i++) {    cout << sz[i]
<< ' ';
 }
 return 0;
}
```

# LCS

```
const int N =
3030; string a,
b; int
dp[N][N];
int lcs(int i, int j) {   if (i >= a.size() or
j >= b.size()) return 0;   if (dp[i][j] !=
-1) return dp[i][j];   int ans = lcs(i +
1, j);   ans = max(ans, lcs(i, j + 1));   if
(a[i] == b[j]) {     ans = max(ans, lcs(i
+ 1, j + 1) + 1);
 }
 return dp[i][j] = ans;
} void print(int i, int j) {   if (i >=
a.size() or j >= b.size()) return;   if
(a[i] == b[j]) {     cout << a[i];
print(i + 1, j + 1);     return;
 }   int x = lcs(i + 1, j);
int y = lcs(i, j + 1);   if
(x >= y) { print(i + 1, j);
}   else {     print(i, j +
1);
 }
}
```

# Longest increasing subsequence (n^2)

```
#include<bits/stdc++.h> using
namespace std;

const int N = 10010; int
a[N], dp[N]; int32_t
main() {   int n; cin >> n;
for (int i = 1; i <= n; i++) {
cin >> a[i];
 }
 for (int i = 1; i <= n; i++) {
dp[i] = 1;     for (int j = 1; j < i;
j++) {       if (a[j] < a[i]) {
dp[i] = max(dp[i], dp[j] + 1);
     }
   }
 }
 int ans = 0;   for (int i =
1; i <= n; i++) {     ans =
max(ans, dp[i]);
 }
 cout << ans << '\n';
return 0;}
```

# Biparte graph

```
int n,e,i,j;
```

**AUST_Outliers**

```
vector<vector<i
nt> > graph;
vector<int>
color; bool
vis[100011];
bool isBiparte()
{
   color[0] = 1; // Mark colour
as 1 for first vertex.     queue
<int> q;

q.push(
0);
while
(!q.emp
ty())
   {
      int temp = q.front();

q.pop();
for
(i=0;i<n
;i++)
      {
         if (graph[temp][i] && color[i]
== -1) //if there is an edge, and
colour is not assigned
         {
            color[i] = 1 - color[temp];
            q.push(i);
         }
         else if (graph[temp][i] && color[i] ==
color[temp]) // if there is an edge and both
verces have same colours
            return 0;                    // graph is
not biparte
      }
   }
   return 1;
}
int main()
{    int
x,y;
   cout<<"Enter number of verces and edges
respecvely:";    cin>>n>>e;    cout<<"\n";
graph.resize(n);    color.resize(n,-1);
memset(vis,0,sizeof(vis));    for(i=0;i<n;i++)
graph[i].resize(n);    for(i=0;i<e;i++)
   {
      cout<<"\nEnter edge verces of edge
"<<i+1<<" :";
cin>>x>>y;
      x--; y--;
graph[x][y]=1;
graph[y][x]=1;
   }
   if(isBiparte())
      cout<<"Yes, The given graph is Biparte.\n";
   else
```

```
    cout<<"No, The given
graoh is not Biparte.\n";
return 0;}
```

# Kruskal

```
const int MAX = 1e6-1;
int root[MAX]; const int
nodes = 4, edges = 5; pair
<long long, pair<int, int>
> p[MAX];
int parent(int a)
//find the parent of the given node
{

while(root[a]
!= a){
root[a] =
root[root[a]];
a = root[a];
  }
  return a;
}
void union_find(int a, int b)
//check if the given two verces are
in the same “union” or not
{
  int d =
parent(
a);   int
e =
parent(
b);   root[d] =
root[e];
}

long long kruskal()
{   int a, b;   long long cost,
minCost = 0;   for(int i = 0 ;
i < edges ; ++i){      a =
p[i].second.first;
    b = p[i].second.second;
cost = p[i].first;
    if(parent(a) != parent(b))
//only select edge if it does not create a cycle
(ie the two nodes forming it have different root
nodes)
    {
      minCost += cost;
union_find(a, b);
    }
  }
  return minCost;
}
int main()
{   int x,
y;
  long long weight, cost, minCost;
  for(int i = 0;i < MAX;++i)
//inialize the array groups
  {     root[i]
= i;   }
```

```
    sort(p, p + edges);
//sort the array of
edges    minCost =
kruskal();
  cout << "Minimum cost is:
"<< minCost << endl;    return
0;}
```

# Lca

```
int LCA(int a, int b,        const
std::vector<int>& depth,
const std::vector<int>& parent)
{
  if (depth[a] > depth[b])
std::swap(a, b);    while
(depth[a] != depth[b]) {
b = parent[b];
  }
  while (a !=
b){       b =
parent[b];
a = parent[a];
  }
  return a;
}
int main()
{    int tests;    std::cin >> tests;    for (int cas
= 1; cas <= tests; ++cas)
  {
    int  nodes;                  std::cin  >>  nodes;
std::vector<int>       depth              (nodes+1);
std::vector<int> parent (nodes+1);       depth[0]
= 0;       depth[1] = 1;
    for (int i = 1; i <= nodes; ++i)
    {          int n;
std::cin >> n;
while (n--)
        {             int dest;
std::cin >> dest;
depth[dest] = depth[i] + 1;
parent[dest] = i;
        }
    }
    int queries;       std::cin >> queries;
std::cout << "Case " << cas << ":\n";
while (queries--)
    {          int l, r;
std::cin >> l >> r;
       std::cout << LCA(l, r, depth, parent) <<
std::endl;
    }
  }
  return 0;
}
```

# Sparse table

```
#include<bits/stdc++.h>
using namespace std; const
int N = 1e5 + 9; int t[N][18],
a[N]; void build(int n) {
for(int i = 1; i <= n; ++i)
t[i][0] = a[i];   for(int k = 1; k
< 18; ++k) {  for(int i = 1; i +
(1 << k) - 1 <= n; ++i) {
     t[i][k] = min(t[i][k - 1], t[i + (1 <<
(k - 1))][k -
1]);
   }
 }
} int query(int l, int r) {   int k
= 31 - __builn_clz(r - l + 1);
return min(t[l][k], t[r - (1 <<
k) + 1][k]);
}
int32_t main() {
 int n;   cin >> n;   for(int
i = 1; i <= n; i++) cin >>
a[i];   build(n);   int q;
cin >> q;   while(q--) {
   in
  t l, r;  cin >> l
  >> r;  ++l;
   ++r;  cout << query(l, r)
  << '\n';
 }
 return 0;
}
```

# Extended Euclid

```
using ll = long long; ll
extended_euclid(ll a, ll b, ll &x, ll &y) {
if (b == 0) {  x = 1; y = 0;  return a;
 }   ll x1, y1;   ll d = extended_euclid(b, a
% b, x1, y1);   x = y1;   y = x1 - y1 * (a /
b);   return d;
} ll inverse(ll a, ll m)
{   ll x, y;
 ll g = extended_euclid(a, m, x, y);
if (g != 1) return -1;   return (x %
m + m) % m;
}
int32_t main() {   ll x = 100, m
= 37;   cout << inverse(x, m)
<< '\n';   return 0;}
```

# //2D prefix sum

```
const int N = 1005; int a[N][N],
pref[N][N]; int32_t main() {
ios_base::sync_with_stdio(0);
cin.tie(0);   int n, m; cin >> n >> m;
for (int i = 1; i <= n; i++) {     for (int j
= 1; j <= m; j++) {
     cin >> a[i][j];
   }
 }
 for (int i = 1; i <= n; i++) {     for
(int j = 1; j <= m; j++) {
     pref[i][j] = pref[i - 1][j] + pref[i][j - 1] - pref[i - 1][j - 1] +
a[i][j];
   }
 }
 int q; cin >> q;   while (q-
-) {
   int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;     int
ans = 0;
   ans = pref[x2][y2] - pref[x1 - 1][y2] - pref[x2][y1 - 1] +
pref[x1 - 1][y1 - 1];     cout << ans << '\n';
 }
 return 0;
}
```

# //big numbers

```
__int128 read() {
   __int128 x = 0, f = 1;     char
ch = getchar();     while (ch < '0'
|| ch > '9') {
      if (ch == '-') f = -1;     ch
   = getchar();
   }
   while (ch >= '0' && ch <= '9') {
      x = x * 10 + ch -
   '0';     ch =
   getchar();
   }
   return x * f;
}
void print(__int128 x) {
   if (x < 0) {
   putchar('-
   ');     x = -x;
   }
   if (x > 9) print(x / 10);
putchar(x % 10 + '0');
}
bool cmp(__int128 x, __int128 y) { return x > y; }
```

# //Prefix function :

```
vector<int>
prefix_function(string s) {     int n
= (int)s.length();     vector<int>
pi(n);     for (int i = 1; i < n; i++) {
int j = pi[i-1];        while (j > 0 &&
s[i] != s[j])           j = pi[j-1];        if
(s[i] == s[j])           j++;        pi[i] =
j;
   }
   return pi;
}
```

**CRT_primeland-s-currency:**

```
#include<bits/stdc++.h>
#define iamspeed
ios_base::sync_with_stdio(false);cin.tie(NULL);co
u t.tie(NULL); #define endl '\n' using namespace
std; typedef long long int ll; const ll
modd=1e9+7;
const ll N=1e6+5;
ll
x,y,z,n,m,k,w,sum,ans,cnt,cnt2,res,mn,mx,t,tt,q,i,
j
;
string
s,ss; int
main()
{
```

```
   ll n;
   cin>>n;
   ll a[n],m[n],M,M1[n],x,M1_inv[n];
M=1;     for(ll i=0; i<n; i++)
   {
      cin>>m[i]>>a[i];
M*=m[i];
   }
   for(ll i=0; i<n; i++)
   {
      M1[i]=M/m[i];        ll
y=(M1[i]*a[i])%M,z=0;
      ///use extended gcd or (euler's totient + mod
expo) to find mod inverse for practice purpose
      for(ll j=1;; j++)
      {
         ll ans=(M1[i]*j)%m[i];
z=(z+y)%M;
         if(ans==1)
         {
            M1_inv[i]=j;
            break;
         }
      }
      ///cerr<<m[i]<<' '<<a[i]<<' '<<M1[i]<<'
'<<M1_inv[i]<<a[i]*M1[i]*M1_inv[i]<<endl;
   }
   x=0;     for(ll i=0; i<n;
i++)
   {
      for(j=1;j<=M1_inv[i];j++)
      {
        x=(x+(M1[i]*a[i])%M)%M;
///x=(x+(((M1[i]*a[i])%M)*M1_inv[i])%M)%M may
cause overflow.
      }                    ///that's why I used a loop inside.
   }
   cout<<x<<endl;     return
0;
}
```