Codebook

1. **Priority Queue:**

priority\_queue<int> pr;     //max\_priority\_queue priority\_queue<int, vector<int>, greater<int>> pr; // min\_priority\_queue priority\_queue<Student, vector<Student>, cmp> pq; // custom Class

***Compare Class for priority queue:***

class Student { public: string name; int roll; int marks; Student(string name, int roll, int marks){ this->name = name; this->roll = roll; this->marks = marks; } };

class cmp

{

public:

    bool operator()(Student a, Student b)

    {

        if (a.marks == b.marks) return a.roll > b.roll;

        else    return a.marks < b.marks;

    }

};

1. **Custom\_Hash for unordered\_map**:

struct chash

{

    static uint64\_t splitmix64(uint64\_t x)

{

        x += 0x9e3779b97f4a7c15;

        x = (x ^ (x >> 30)) \* 0xbf58476d1ce4e5b9;

        x = (x ^ (x >> 27)) \* 0x94d049bb133111eb;

        return x ^ (x >> 31);

    }

    size\_t operator()(uint64\_t x) const

{

        static const uint64\_t FIXED\_RANDOM = chrono::steady\_clock::now().time\_since\_epoch().count();

        return splitmix64(x + FIXED\_RANDOM);

    }

    size\_t operator()(pair<uint64\_t,uint64\_t> x) const

{

        static const uint64\_t FIXED\_RANDOM = chrono::steady\_clock::now().time\_since\_epoch().count();

        return splitmix64(x.first + FIXED\_RANDOM)^(splitmix64(x.second + FIXED\_RANDOM) >> 1);

    }

    //for vector I can also do the similarly with lower collision

    long long operator()(string s) const

{

        const int p = 31;

        //for only lowercase p=31,for lowercase+uppercase p=53

        const int m = 1e9 + 9;

        long long hash\_value = 0;

        long long p\_pow = 1;

        for (char c : s)

{

            hash\_value = (hash\_value + (c - 'a' + 1) \* p\_pow) % m;

            p\_pow = (p\_pow \* p) % m;

        }

        return hash\_value;

    }

};

**5) Binary Search:**

* ***closest from left: (1 indexed)***

ll x;   cin>>x; auto it=upper\_bound(all(v),x); cout<<it-v.begin()<<nl;

* ***closest from right: (1 indexed)***

ll x;   cin>>x; auto it=lower\_bound(all(v),x); cout<<it-v.begin()+1<<nl;

* ***Binary Search on Real Number***:

ld l=0; // good ld r=1e18; //bad

    for(int i=0;i<=100;i++)

    {

        ld mid=(l+r)/2;

        if(good(mid))   l=mid;

        else r=mid;

    }

    cout<<fixed<<setprecision(6)<<l<<nl;

**6) Two Pointers:**

***General Code:***

ll L=0;

for(ll R = 0; R < n; R++)

{

    add(v[R]); // this function will recalculate after updating right pointer

    while( !good() )

    {

        remove(v[L]); // this function will recalculate after updating left pointer

        L++;

    }

    ans+=R-L+1; // what is needed do here

}

***If we can add element and recalculate the function then we also can remove and recalculate the function using following technique:***

struct stack

{

    vector<ll>s,smin={LLONG\_MAX},smax={LLONG\_MIN};

    void push(ll x)

    {

        s.pb(x);

        smin.pb(::min(smin.back(),x));

        smax.pb(::max(smax.back(),x));

    }

    ll pop()

    {

        ll res=s.back();

        s.ppb();

        smin.ppb();

        smax.ppb();

        return res;

    }

    bool empty()

    {

        return s.empty();

    }

    ll min()

    {

        return smin.back();

    }

    ll max()

    {

        return smax.back();

    }

};

::stack s1,s2;

**7) Bit Masking: (max array size 20)**

for (int bits = 0; bits < (1 << n); bits++)

    {

        ll sum\_of\_subset = 0;

        for (int i = 0; i < n; i++)

        {

            if (bits & (1 << i)) sum\_of\_subset += v[i];

            else sum\_of\_subset -= v[i];

        }

    }

**8) Next Permutation:**

ll n;   cin >> n; string s;

    while (true){

        s += "47";

        sort(s.begin(), s.end());

        if (stoll(s) >= n)

        {

            cout << stoll(s) << nl;

            return 0;

        }

        while (next\_permutation(s.begin(), s.end()))

        {

            if (stoll(s) >= n)

            {

                cout << stoll(s) << nl;

                return 0;

            }

        }

    }

**9) N er ith bit set ki na:**

ll n,i; cin >> n >> i;

    if (n & (1 << i))   cout << "true" << nl;

    else    cout << "false" << nl;

**10) Decimal to binary:**

long long int x;

    cin >> x;

    bool flag = false;

    // for int it start from 30

    // for long long int it start from 62 then 1 should be 1LL to avoid overflow

    // or to avoid overflow we can write ((x >> i) & 1) equivalent of ( x & (1LL << i))

    for (int i = 62; i >= 0; i--)

    {

        if (((x >> i) & 1) != 0) // be careful about brackets

        {

            cout << "1";

            flag = true;

        }

        else if (flag)  cout << "0";

    }

**11) flipping all bits of a number:**

unsigned int x; cin >> x;

        x = ~x;

        cout << x << nl;

**12) reversing bits of a number using bitset:**

unsigned int n; cin >> n;

       bitset<32> x(n);

       bitset<32> rev(n);

       for (int i = 0; i < 32; i++)

       {

            rev[31 - i] = x[i];

       }

       unsigned int ans = rev.to\_ulong();

       cout << ans << nl;

**17) Modular of power/ Binary Exponentiation:**

ll biexp(ll a,ll b){

    ll result=1;

    while(b>0)

    {

        if(b&1) result=(result\*a)%mod;

        a=(a\*a)%mod;

        b/=2;

    }

    return result;

}

**14) Divide & Conquer: (Merge Sort)**

void merge(ll l,ll mid,ll r)

{

    ll n1=mid-l+1,n2=r-mid;

    vector<ll>a(n1),b(n2);

    for(ll i=0;i<n1;i++)    a[i]=v[l+i];

    for(ll i=0;i<n2;i++)    b[i]=v[mid+1+i];

//here you will have two sorted array (left & right part ). And here you can do your calculations.

    ll i=0,j=0,k=l;

    while(i<n1 && j<n2)

    {

        if(a[i]<b[j])

        {

            v[k]=a[i];

            k++,i++;

        }

        else

        {

            v[k]=b[j];

            k++,j++;

        }

    }

    while(i<n1)

    {

        v[k]=a[i];

        k++,i++;

    }

    while(j<n2)

    {

        v[k]=b[j];

        k++,j++;

    }

}

void mergesort(ll l,ll r)

{

    if(l<r)

    {

        ll mid=(l+r)/2;

        mergesort(l,mid);

        mergesort(mid+1,r);

        merge(l, mid, r);

    }

}

void solve()

{

    ll n;   cin>>n;

    v.resize(n);

    for (ll i=0;i<n;i++)  cin>>v[i];

    mergesort(0,n-1);

    for(ll i=0;i<n;i++)  cout<<v[i]<<" ";

    cout<<nl;

}

**15. Compressing integer(range):**

vector<pair<ll,ll>>v(n);

    set<ll>st;

    for(ll i=0;i<n;i++)

    {

        cin>>v[i].fi>>v[i].sec;

        st.insert(v[i].fi),st.insert(v[i].fi+1);

        st.insert(v[i].sec),st.insert(v[i].sec+1);

    }

    vector<ll>compress;

    for(ll x:st)    compress.pb(x);

    unordered\_map<ll,ll,custom\_hash>mp;

    for(ll i=0;i<compress.size();i++)   mp[compress[i]]=i;

    for(ll i=0;i<n;i++) v[i].fi=mp[v[i].fi],v[i].sec=mp[v[i].sec];

**16. Modular formula’s:**

* ab^c % m = ab^c % n % m [n=m-1]
* (a + b) % m = ( ( a % m ) + ( b % m ) ) % m
* (a \* b) % m = ( ( a % m ) \* ( b % m ) ) %m
* (a – b) % m = ( ( a % m ) – ( b % m ) + m ) % m
* (a / b) % m = ( ( a % m ) \* ( b-1 % m ) ) % m
* B-1 % m = power ( b , m-2)

17. **2D prefix sum**:

ll n,m,q; cin>>n>>m>>q;

    vector<vector<ll>>v(n,vector<ll>(m));

    for(ll i=0;i<n;i++)

        for(ll j=0;j<m;j++) cin>>v[i][j];

    vector<vector<ll>>prefix(n,vector<ll>(m));

    for(ll i=0;i<n;i++)

    {

        for(ll j=0;j<m;j++)

        {

            prefix[i][j]=v[i][j];

            if(i!=0)    prefix[i][j]+=prefix[i-1][j];

            if(j!=0)    prefix[i][j]+=prefix[i][j-1];

            if(i!=0 && j!=0)    prefix[i][j]-=prefix[i-1][j-1];

        }

    }

    while(q--)

    {

        ll a,b,c,d; cin>>a>>b>>c>>d;

        a--,b--,c--,d--;

        ll ans=prefix[c][d];

        if(a!=0 && b!=0)    ans+=prefix[a-1][b-1];

        if(a!=0)    ans-=prefix[a-1][d];

        if(b!=0)    ans-=prefix[c][b-1];

        cout<<ans<<nl;

    }

**18. Pascal’s triangle:**

// max possible size 66 //tried and tested

ll C[66][66];

void pascal\_triangle()

{

    C[0][0]=1;

    for(ll i=1;i<=66;i++)

    {

        C[i][0]=C[i][i]=1;

        for(ll j=1;j<i;j++)

        {

            C[i][j]=C[i-1][j-1]+C[i-1][j];

        }

    }

}

**19. nCk:**

ll nCk(ll n,ll k)   // it will work for comperatively bigger number

{

    double c=1;

    for(ll i=1;i<=k;i++)

    {

        c=c\*(n-i+1)/i;

    }

    return ll(c+0.01);

}

void solve{

ll n,k; cin>>n>>k;

    // nCk==nC(n-k)

    ll C=nCk(n,min(k,n-k));

    cout<<C<<nl;

}

**20. Cycle Detection Directed Graph:**

const int N=1e5+5;

vector<int> adj[N];

bool vis[N]={false};

bool pathVis[N]={false};

bool cycle=false;

void dfs(int parent)

{

    vis[parent]=true;

    pathVis[parent]=true;

    for(int child : adj[parent])

    {

        if(pathVis[child])

        {

            cycle=true;

        }

        if(!vis[child])

        {

            dfs(child);

        }

    }

    pathVis[parent]=false;

}

int main()

{

    int n,e;    cin>>n>>e;

    for(int i=0;i<e;i++)

    {

        int a,b;    cin>>a>>b;

        adj[a].push\_back(b);

    }

    for(int i=0;i<n;i++)

    {

        if(!vis[i])

        {

            dfs(i);

        }

    }

    if(cycle)   cout<<"cycle detected"<<endl;

    else cout<<"cycle not detected"<<endl;

    return 0;

}

**21. Cycle Detection Undirected Graph**:

const int N=100;

vector<int> adj[N];

bool vis[N]={false};

int parentArray[N]={-1};

bool cycle=false;

void dfs(int parent)

{

    vis[parent]=true;

    for(int child:adj[parent])

    {

        if(vis[child] && child!=parentArray[parent])  cycle=true;

        if(!vis[child])

        {

            parentArray[child]=parent;

            dfs(child);

        }

    }

}

int main()

{

    int n,e;    cin>>n>>e;

    for(int i=0;i<e;i++)

    {

        int a,b;    cin>>a>>b;

        adj[a].push\_back(b);

        adj[b].push\_back(a);

    }

    for(int i=0;i<n;i++)

    {

        if(!vis[i]) dfs(i);

    }

    if(cycle)   cout<<"Cycle Detected"<<endl;

    else cout<<"Cycle not Detected"<<endl;

**22. Dijkstra. (weighted single source shortest path):**  
  
const int N=100;

vector<pair<int,int>> adj[N];

int dis[N];

class cmp

{

    public:

        bool operator()(pair<int,int>a,pair<int,int>b)

        {

            return a.second>b.second;

        }

};

// dijkstra works on monotonic type function

// like sum

// it will not work on xor

// because xor is not monotonic

// it doesn't gurantee that big number will always give big xor value

// or small number will always give smanll xor value

void dijkstra(int src)      //O((V+E)\*logE)

{

    priority\_queue<pair<int,int>,vector<pair<int,int>>,cmp>pq;

    dis[src]=0;

    pq.push({src,0});

    while(!pq.empty())

    {

        auto [parent,weight]=pq.top();

        pq.pop();

        for(auto [child,x]:adj[parent])

        {

            if(weight+x < dis[child])

            {

                dis[child]=weight+x;

                pq.push({child,weight+x});

            }

        }

    }

}

int main()

{

    int n,e;    cin>>n>>e;

    for(int i=0;i<e;i++)

    {

        int u,v,w;  cin>>u>>v>>w;

        adj[u].push\_back({v,w});

        adj[v].push\_back({u,w});

    }

    fill(dis,dis+N,INT\_MAX);

    dijkstra(0);

    for(int i=0;i<n;i++)    cout<<dis[i]<<" ";

    cout<<endl;

**23. Bellman Ford: (Single source shortest path with negative cycle)**  
  
 struct edge {

    int u,v,w;

};

int main()

{

    int n,e;    cin>>n>>e;

    vector<edge>edgeList;

    for(int i=0;i<e;i++)

    {

        int u,v,w;  cin>>u>>v>>w;

        edgeList.push\_back({u,v,w});

        edgeList.push\_back({v,u,w});    //only difference between

                        //directed and undirected is here

    }

    vector<int>dis(n,INT\_MAX);  //Time Complexity O(n\*e)

    int src=0;

    dis[src]=0;

    for(int i=1;i<n;i++)

    {

        for(auto [u,v,w]:edgeList)

        {

            if(dis[u]!=INT\_MAX && dis[u]+w < dis[v])

            {

                dis[v]=dis[u]+w;

            }

        }

    }

    bool cycle=false;

    for(auto [u,v,w]:edgeList)

    {

        if(dis[u]!=INT\_MAX && dis[u]+w < dis[v])

        {

            cycle = true;

            break;

            dis[v]=dis[u]+w;

        }

    }

    if(cycle)   cout<<"Negative Cycle, No Answer"<<endl;

    else

    {

        for(int i=0;i<n;i++) cout<<dis[i]<<" ";

        cout<<endl;

    }

**24. Floyd Warshall: (all source shortest path)**

int main()

{

    int n,e;    cin>>n>>e;

    vector<vector<int>>adj(n,vector<int>(n,INT\_MAX));

    for(int i=0;i<n;i++)    adj[i][i]=0;

    for(int i=0;i<e;i++)

    {

        int u,v,w;  cin>>u>>v>>w;

        adj[u][v]=w;

    }

    for(int k=0;k<n;k++)

    {

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<n;j++)

            {

                if(k==i || k==j)    continue;

                if(adj[i][k]==INT\_MAX || adj[k][j]==INT\_MAX)    continue;

                if(adj[i][k]+adj[k][j] < adj[i][j]) //i to j via k (i->k->j < i->j)

                {

                    adj[i][j]=adj[i][k]+adj[k][j];

                }

            }

        }

    }

    //logic of negative cycle detection

    //where i==j the distance for it should be always zero(0)

    // if this value become less than 0 then its a negative cycle

    // like....

    bool cycle=false;

    for(int i=0;i<n;i++)

    {

        if(adj[i][i]<0)

        {

            cycle=true;

            break;

        }

    }

    // Floyd Warshall works for directed graph

    // in undirected graph floyd warshal fails to detect negative cycle

    // because if any weight is negative (like -1) its go u->v with -1 and v->u -1.

    // but in undirected graph you need atleast 3 node to for a cycle

    //if there is no negative cycle then floyd warshall can work on undirected

    if(cycle)   cout<<"Negative Cycle Detected"<<endl;

    else

    {

        for(int i=0;i<n;i++)    // works for positive or negative weight

        {

            for(int j=0;j<n;j++)

            {

                cout<<adj[i][j]<<" ";

            }

            cout<<endl;

        }

    }

**25. Cycle Detection using dsu (undirected graph):**

const int N=1e5+5;

vector<int>parent(N);

vector<int>group\_size(N);

void initialize(int n)

{

    for(int i=1;i<=n;i++)

    {

        parent[i]=-1;

        group\_size[i]=1;

    }

}

int dsu\_find(int node)

{

    if(parent[node]==-1)    return node;

    int leader=dsu\_find(parent[node]);

    parent[node]=leader;

    return leader;

}

void union\_by\_size(int node1,int node2)

{

    int leader1=dsu\_find(node1);

    int leader2=dsu\_find(node2);

    if(group\_size[leader1]>group\_size[leader2]) swap(leader1,leader2);

    parent[leader1]=leader2;

    group\_size[leader2]+=group\_size[leader1];

}

int main()

{

    int n,e;    cin>>n>>e;

    initialize(n);

    bool cycle=false;

    for(int i=0;i<e;i++)

    {

        int u,v;    cin>>u>>v;

        int leader1=dsu\_find(u);

        int leader2=dsu\_find(v);

        if(leader1==leader2)

        {

            cycle=true;

        }

        else

        {

            union\_by\_size(u,v);

        }

    }

    if(cycle)   cout<<"Cycle Detected"<<endl;

    else cout<<"No Cycle"<<endl;

**25. MST (krushkal):**  
 /\* MST mane eita na je ekta vertex theke arekta vertex e jaoyar

minimum cost er rasta ber kora. Er mane holo prottekta vertex theke

jeno prottekta vertex e jaoya jay sobcheye total kom khoroce. \*/

#include<bits/stdc++.h>

using namespace std;

struct edge {

    int u,v,w;

};

bool cmp(edge a,edge b)

{

    return a.w < b.w;

}

const int N=1e5+5;

vector<int>parent(N);

vector<int>level(N);

void initialize(int n)

{

    for(int i=0;i<n;i++)

    {

        parent[i]=-1;

        level[i]=0;

    }

}

int dsu\_find(int node)

{

    if(parent[node]==-1)    return node;

    return parent[node]=dsu\_find(parent[node]);

}

void union\_by\_level(int node1,int node2)

{

    int leader1=dsu\_find(node1);

    int leader2=dsu\_find(node2);

    if(level[leader1]>level[leader2])

    {

        parent[leader2]=leader1;

    }

    else if(level[leader1]<level[leader2])

    {

        parent[leader1]=leader2;

    }

    else

    {

        parent[leader1]=leader2;

        level[leader2]++;

    }

}

int main()

{

    int n,e;    cin>>n>>e;

    initialize(n);

    vector<edge>edgeList(e);

    for(int i=0;i<e;i++)

    {

        int u,v,w;  cin>>u>>v>>w;

        edgeList[i]={u,v,w};

    }

    sort(edgeList.begin(),edgeList.end(),cmp);

    int cost=0;

    for(auto [u,v,w]:edgeList)

    {

        int leader1=dsu\_find(u);

        int leader2=dsu\_find(v);

        if(leader1==leader2)

        {

            //same group or connected

            continue;

        }

        else

        {

            // not connected, so connect them

            union\_by\_level(leader1,leader2);

            cost+=w;

        }

    }

    cout<<cost<<endl;

    return 0;

}

**26. Policy Based Data Structure:**

i. Ordered Set:

#include<bits/extc++.h>

using namespace \_\_gnu\_pbds;

typedef tree <int,null\_type,less<int>,rb\_tree\_tag,tree\_order\_statistics\_node\_update> oset;

// oset\_name.order\_of\_key(x);   number of element less than x

// \*oset\_name.find\_by\_order(k);  kth element 0 based if(end(oset\_name)!=oset\_name.find\_by\_order(4))<<nl; //true->ache //false->nai

ii. Gp Hash Table:

#include<bits/extc++.h>

// gp\_hash\_table<key,value,chash>name;

**27. Sieve of Eratosthenis:**

**i. Linear Sieve:**

const int N=1e7+7;  //ideally can calculate upto 1e8

bitset<N>is\_prime;

vector<int>primes;

void sieve()

{

    is\_prime.set(); //set all bits to 1

    is\_prime[0]=is\_prime[1]=0;

    for(int i=2;i\*i<=N;i++)

    {

        if(is\_prime[i])

        {

            for(int j=i\*i;j<N;j+=i)  is\_prime[j]=0;

        }

    }

    for(int i=2;i<N;i++)

        if(is\_prime[i]) primes.push\_back(i);

}

**ii. Bitwise Sieve:**

#define IS\_COMPOSITE(id, composite) (composite[(id) >> 5] & (1 <<((id) & 31)))

#define SET\_COMPOSITE(id, composite) (composite[(id) >> 5] |= (1 <<((id) & 31)))

const int N = 100000000;

// Only odd numbers

const int LIMIT = (N >> 1);

const int ARRAY\_SIZE = (LIMIT >> 5) + 1;

unsigned int composite[ARRAY\_SIZE];

vector<int> primes;

void bitwise\_sieve()

{

    for (int i = 3; i \* i <= N; i += 2) {

        if (!IS\_COMPOSITE(i >> 1, composite)) {

            for (int j = i \* i; j <= N; j += (i << 1)) {

                SET\_COMPOSITE(j >> 1, composite);

            }

        }

    }

    primes.push\_back(2);

    for (int i = 3; i <= N; i += 2) {

        if (!IS\_COMPOSITE(i >> 1, composite)) primes.push\_back(i);

    }

}

**iii.** **Segmented Sieve**:

vector<char> segmentedSieve(ll L,ll R)

{

    ll N=sqrtl(R);

    vector<char>is\_prime(N+1,'1');

    is\_prime[0]=is\_prime[1]='0';

    for(ll i=2;i\*i<=N;i++)

    {

        if(is\_prime[i]=='1')

        {

            for(ll j=i\*i;j<=N;j+=i) is\_prime[j]='0';

            //here we can keep j==N because vector size is N+1

        }

    }

    vector<ll>primes;

    for(ll i=2;i<=N;i++)

        if(is\_prime[i]) primes.push\_back(i);

    // precalculate primes upto sqrt(R).

    vector<char>rangePrime(R-L+1,'1');

    for(ll el:primes)

    {

        ll st=max(el\*el,((L+el-1)/el)\*el);

        for(ll j=st;j<=R;j+=el) rangePrime[j-L]='0';

    }

    if(L==1)    rangePrime[0]='0';

    return rangePrime;

    //segmented sieve part

}

void solve()

{

    ll L,R; cin>>L>>R;

    vector<char>primes=segmentedSieve(L,R);

    for(ll i=0;i<primes.size();i++)

    {

        if(primes[i]=='1') cout<<L+i<<" ";

    }

    cout<<nl;

}

**28. All about big number:**string removeLeadingZeros(string str) {

    int i = (str[0] == '-') ? 1 : 0;

    while (i < str.size() - 1 && str[i] == '0') ++i;

    return (str[0] == '-' ? "-" : "") + str.substr(i);

}

string absolute(string str) {

    return (str[0] == '-') ? str.substr(1) : str;

}

bool isEqual(string a, string b) {

    return removeLeadingZeros(absolute(a)) == removeLeadingZeros(absolute(b));

}

bool isGreater(string a, string b) {

    a = removeLeadingZeros(absolute(a));

    b = removeLeadingZeros(absolute(b));

    if (a.size() != b.size()) return a.size() > b.size();

    return a > b;

}

bool isGreaterOrEqual(string a, string b) {

    return isGreater(a, b) || isEqual(a, b);

}

bool isLess(string a, string b) {

    return !isGreaterOrEqual(a, b);

}

// helps in (a+b) function

string addStrings(string a, string b) {

    string res;

    int carry = 0;

    reverse(a.begin(), a.end());

    reverse(b.begin(), b.end());

    for (int i = 0; i < max(a.size(), b.size()) || carry; ++i) {

        int digit = carry;

        if (i < a.size()) digit += a[i] - '0';

        if (i < b.size()) digit += b[i] - '0';

        res += digit % 10 + '0';

        carry = digit / 10;

    }

    reverse(res.begin(), res.end());

    return res;

}

// helps in (a-b)   function

string subtractStrings(string a, string b) {

    string res;

    bool neg = false;

    if (isGreater(b, a)) {

        swap(a, b);

        neg = true;

    }

    reverse(a.begin(), a.end());

    reverse(b.begin(), b.end());

    int carry = 0;

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

        int digit = a[i] - '0' - carry;

        if (i < b.size()) digit -= b[i] - '0';

        if (digit < 0) digit += 10, carry = 1;

        else carry = 0;

        res += digit + '0';

    }

    reverse(res.begin(), res.end());

    res = removeLeadingZeros(res);

    return (neg ? '-' + res : res);

}

// (a+b)

string bigIntAdd(string a, string b) {

    bool negA = a[0] == '-', negB = b[0] == '-';

    a = absolute(a); b = absolute(b);

    if (negA == negB) {

        string res = addStrings(a, b);

        return negA ? '-' + res : res;

    }

    return negA ? subtractStrings(b, a) : subtractStrings(a, b);

}

// (a-b)

string bigIntSubtract(string a, string b) {

    bool negA = a[0] == '-', negB = b[0] == '-';

    a = absolute(a); b = absolute(b);

    if (!negA && !negB) return subtractStrings(a, b);

    if (negA && negB) return subtractStrings(b, a);

    string res = addStrings(a, b);

    return negA ? '-' + res : res;

}

// (a\*b)

string multiplyStrings(string a, string b) {

    bool neg = (a[0] == '-') ^ (b[0] == '-');

    a = absolute(a); b = absolute(b);

    vector<int> res(a.size() + b.size(), 0);

    for (int i = a.size() - 1; i >= 0; --i)

        for (int j = b.size() - 1; j >= 0; --j)

            res[i + j + 1] += (a[i] - '0') \* (b[j] - '0');

    for (int i = res.size() - 1; i > 0; --i) {

        res[i - 1] += res[i] / 10;

        res[i] %= 10;

    }

    string s;

    for (int digit : res) s += digit + '0';

    s = removeLeadingZeros(s);

    return (neg && s != "0") ? '-' + s : s;

}

// (a/b)

string divideStrings(string a, string b) {

    if (b == "0" || b == "-0") throw invalid\_argument("Division by zero");

    bool neg = (a[0] == '-') ^ (b[0] == '-');

    a = absolute(a); b = absolute(b);

    string res, curr;

    for (char c : a) {

        curr += c;

        curr = removeLeadingZeros(curr);

        int count = 0;

        while (isGreaterOrEqual(curr, b)) {

            curr = subtractStrings(curr, b);

            count++;

        }

        res += count + '0';

    }

    res = removeLeadingZeros(res);

    return (neg && res != "0") ? '-' + res : res;

}

// here both are integer but pow is string

// just converst base and exponents to string

// pow(a,b) here a is string and b is int

string powerIntBase(string base, int exponent) {

    bool neg = base[0] == '-' && (exponent % 2 == 1);

    base = absolute(base);

    string result = "1";

    while (exponent > 0) {

        if (exponent % 2 == 1)

            result = multiplyStrings(result, base);

        base = multiplyStrings(base, base);

        exponent /= 2;

    }

    return (neg ? '-' + result : result);

}

// pow(a,b) here both are string

string powerStrBase(string base, string exponent) {

    bool neg = base[0] == '-' && ((exponent.back() - '0') % 2 == 1);

    base = absolute(base);

    string result = "1";

    while (exponent != "0") {

        if ((exponent.back() - '0') % 2 == 1)

            result = multiplyStrings(result, base);

        base = multiplyStrings(base, base);

        exponent = divideStrings(exponent, "2");

    }

    return (neg ? '-' + result : result);

}

// gcd (a,b)

string bigIntGCD(string a, string b) {

    a = absolute(a);

    b = absolute(b);

    while (b != "0") {

        string temp = b;

        b = divideStrings(a, b);

        b = subtractStrings(a, multiplyStrings(b, temp)); // b = a % b

        a = temp;

    }

    return a;

}

// lcm = (a \* b) / gcd(a,b)

string bigIntLCM(string a, string b) {

    string gcd = bigIntGCD(a, b);

    string mult = multiplyStrings(absolute(a), absolute(b));

    return divideStrings(mult, gcd);

}

// Modular operation (a mod m), both positive; returns string

string bigIntMod(string a, string m) {

    a = absolute(a);

    m = absolute(m);

    string rem = "0";

    for (char c : a) {

        rem = multiplyStrings(rem, "10");

        rem = bigIntAdd(rem, string(1, c));

        while (isGreaterOrEqual(rem, m)) rem = subtractStrings(rem, m);

    }

    return rem;

}

// Modular exponentiation: (base^exponent) mod mod

string bigIntModPow(string base, string exponent, string mod) {

    base = bigIntMod(base, mod);

    string result = "1";

    while (exponent != "0") {

        if ((exponent.back() - '0') % 2 == 1)

            result = bigIntMod(multiplyStrings(result, base), mod);

        base = bigIntMod(multiplyStrings(base, base), mod);

        exponent = divideStrings(exponent, "2");

    }

    return result;

}

// Integer square root of big number (floor)

string bigIntSqrt(string n) {

    if (n == "0" || n == "1") return n;

    string low = "1", high = n, ans = "1";

    while (isLess(low, high) || isEqual(low, high)) {

        string mid = divideStrings(bigIntAdd(low, high), "2");

        string sq = multiplyStrings(mid, mid);

        if (isGreater(sq, n)) {

            high = subtractStrings(mid, "1");

        } else {

            ans = mid;

            low = addStrings(mid, "1");

        }

    }

    return ans;

}

**29. Math Series formula:**

i. ; sum of first n natural number

ii. ; sum of first n squares

iii. ; sum of first n cubes

iv. ; sum of the product of two consecutive number

v. ; sum of the product of four consecutive number

vi. ; sum of even numbers upto n

vii. ; sum of odd numbers upto n

viii. ; sum of first n value of AP(arithmetic progression)

first term a, difference d;

ix. ; sum of first n value of GP(geometric progression)

first term a, ratio r ( r≠1)

x. ; bionomial coefficient ,

30. KMP:

vector<int> buildLPS(const string &pattern) {

    int n = pattern.size();

    vector<int> lps(n, 0);

    int len = 0;

    for (int i = 1; i < n;) {

        if (pattern[i] == pattern[len]) {

            lps[i++] = ++len;

        } else {

            if (len != 0)

                len = lps[len - 1];

            else

                lps[i++] = 0;

        }

    }

    return lps;

}

// KMP search function

pair<int, vector<int>> KMPSearch(const string &text, const string &pattern) {

    vector<int> lps = buildLPS(pattern);

    vector<int> matchIndexes;

    int i = 0, j = 0;

    int count = 0;

    while (i < (int)text.size()) {

        if (text[i] == pattern[j]) {

            i++; j++;

        }

        if (j == (int)pattern.size()) {

            matchIndexes.push\_back(i - j);

            count++;

            j = lps[j - 1]; // allow for overlapping matches

        } else if (i < (int)text.size() && text[i] != pattern[j]) {

            if (j != 0)

                j = lps[j - 1];

            else

                i++;

        }

    }

    return {count, matchIndexes};

}

31. 0-1 Knapsack:

int n,sum;  cin>>n>>sum;

    vector<int>v(n);

    for(int i=0;i<n;i++)    cin>>v[i];

    vector<vector<int>>dp(n+1,vector<int>(sum+1));

    dp[0][0]=1;

    for(int i=1;i<=sum;i++) dp[0][i]=0;

    for(int i=1;i<=n;i++)

    {

        for(int j=0;j<=sum;j++)

        {

            if(v[i-1]<=j)   dp[i][j]=dp[i-1][j-v[i-1]]+dp[i-1][j];

            else    dp[i][j]=dp[i-1][j];

        }

    }

    cout<<dp[n][sum]<<endl;

/\*  dp vector's ith row and jth coloumn means how many subset we have to make

        sum equal j using the first i elements. \*/

32. Unbound Knapsack:

int n,w;    cin>>n>>w;

    vector<int>value(n),weight(n);

    for(int i=0;i<n;i++) cin>>value[i];

    for(int i=0;i<n;i++) cin>>weight[i];

    vector<vector<int>>dp(n+1,vector<int>(w+1));

    for(int i=0;i<=n;i++)   dp[i][0]=0;

    for(int i=0;i<=w;i++)   dp[0][i]=0;

    for(int i=1;i<=n;i++)

    {

        for(int j=1;j<=w;j++)

        {

            if(weight[i-1]<=j)  dp[i][j]=max(dp[i][j-weight[i-1]]+value[i-1],dp[i-1][j]);

            else dp[i][j]=dp[i-1][j];

        }

    }

    cout<<dp[n][w]<<endl;

33. Longest Common Subsequence Printing:

string a,b; cin>>a>>b;

    vector<vector<int>>dp(a.size()+1,vector<int>(b.size()+1));

    for(int i=0;i<=a.size();i++)    dp[i][0]=0;

    for(int i=0;i<=b.size();i++)    dp[0][i]=0;

    for(int i=1;i<=a.size();i++)

    {

        for(int j=1;j<=b.size();j++)

        {

            if(a[i-1]==b[j-1])  dp[i][j]=dp[i-1][j-1]+1;

            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

        }

    }

    cout<<dp[a.size()][b.size()]<<endl;

    int i=a.size(),j=b.size();

    string s;

    while(i!=0 && j!=0)

    {

        if(a[i-1]==b[j-1])  s.push\_back(a[i-1]),i--,j--;

        else

        {

            if(dp[i-1][j]>=dp[i][j-1])  i--;

            else j--;

        }

    }

    reverse(s.begin(),s.end());

    cout<<s<<endl;

34. Longest Common Substring Printing:

string a,b; cin>>a>>b;

    vector<vector<int>>dp(a.size()+1,vector<int>(b.size()+1));

    for(int i=0;i<=a.size();i++)    dp[i][0]=0;

    for(int i=0;i<=b.size();i++)    dp[0][i]=0;

    for(int i=1;i<=a.size();i++)

    {

        for(int j=1;j<=b.size();j++)

        {

            if(a[i-1]==b[j-1])  dp[i][j]=dp[i-1][j-1]+1;

            else dp[i][j]=0;

        }

    }

    int mx=INT\_MIN,r=-1,c=-1;

    for(int i=0;i<=a.size();i++)

    {

        for(int j=0;j<=b.size();j++)

        {

            if(dp[i][j]>mx)

            {

                mx=dp[i][j];

                r=i,c=j;

            }

        }

    }

    cout<<mx<<endl;

    string s;

    while(dp[r][c]!=0)

    {

        s.push\_back(a[r-1]);

        r--,c--;

    }

    reverse(s.begin(),s.end());

    cout<<s<<endl;

35. LIS – Longest Increasing Subsequence:

void solve()

{

    ll n;   cin>>n;

    vector<ll>v(n);

    for(ll i=0;i<n;i++) cin>>v[i];

    vector<ll>dp(n+1,LLONG\_MAX);

    dp[0]=LLONG\_MIN;

    vector<ll>parent(n,-1),idx(n+1,-1);

    for(int i=0;i<n;i++)

    {

        int j = lower\_bound(all(dp), v[i]) - dp.begin();

        if (v[i] < dp[j])

        {

            dp[j] = v[i];

            parent[i] = idx[j-1];

            idx[j] = i;

        }

    }

    int length=0, lastIdx=-1;

    for(ll i=1;i<=n;i++)

    {

        if (dp[i]!=LLONG\_MAX && i>length)

        {

            length=i;

            lastIdx=idx[i];

        }

    }

    cout<<length<<nl;

    vector<ll> lis;

    while (lastIdx != -1)

    {

        lis.push\_back(v[lastIdx]);

        lastIdx = parent[lastIdx];

    }

    reverse(all(lis));

    for (ll el:lis) cout<<el<<" ";

    cout<<nl;

}

int32\_t main()

{

    ios\_base::sync\_with\_stdio(false);

    cin.tie(NULL);

    cout.tie(NULL);

    ll t = 1;

    // cin>>t;

    while(t--) solve();

    return 0;

}

// here dp[i] means min value of a[j] 0<=j<=n which gives

// increasing subsequence of leanth i.

// for(ll j=1;j<=n;j++) //loop replaced by binary search

// {

//     if(dp[j-1]<v[i] && v[i]<dp[j])

//     {

//         dp[j]=v[i];

//         parent[i] = idx[j-1];

//         idx[j] = i;

//     }

// }

// lexicographically smallest lagle lower bound

36. Number of Divisors:

ll numberOfDivisors(ll num)

{

    ll total=1;

    for(ll i=2;i\*i<=num;i++)

    {

        ll e=0;

        while(num%i==0)

        {

            e++;

            num/=i;

        }

        total\*=(e+1);

    }

    if(num>1)   total\*=2;

    return total;

}

37. Sum of Divisors:

ll sumOfDivisors(ll num)

{

    ll total=1;

    for(ll i=2;i\*i<=num;i++)

    {

        ll e=0;

        while(num%i==0)

        {

            e++;

            num/=i;

        }

        ll sum=0,pow=1;

        while(e>=0)

        {

            sum+=pow;

            pow\*=i;

            e--;

        }

        total\*=sum;

    }

    if(num>1)   total\*=(num+1);

    return total;

}

38. Prime Check & Composite Check & Count Divisors upto 1e18:

using u64 = uint64\_t;

// fallback for multiplying large number under mod

u64 mul\_mod(u64 a, u64 b, u64 mod)

{

    u64 res = 0;

    a = a % mod;

    b = b % mod;

    while (b > 0){

        if (b & 1) res = (res + a) % mod;

        a = (a + a) % mod;

        b /= 2;

    }

    return res;

}

u64 binpower(u64 base, u64 e, u64 mod) {

    u64 result = 1;

    base %= mod;

    while (e) {

        if (e & 1) result = mul\_mod(result, base, mod);

        base = mul\_mod(base, base, mod);

        e >>= 1;

    }

    return result;

}

bool check\_composite(u64 n, u64 a, u64 d, int s) {

    u64 x = binpower(a, d, n);

    if (x == 1 || x == n - 1) return false;

    for (int r = 1; r < s; r++) {

        x = mul\_mod(x, x, n);

        if (x == n - 1) return false;

    }

    return true;

}

bool MillerRabin(u64 n) { // returns true if n is prime, else false.

    if (n < 2) return false;

    int r = 0;

    u64 d = n - 1;

    while ((d & 1) == 0) {

        d /= 2;

        r++;

    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {

        if (n == (u64)a) return true;

        if (check\_composite(n, a, d, r)) return false;

    }

    return true;

}

ll countDivUpto1e18(ll num)

{

    ll N = 1000000;

    vector<char> is\_prime(N + 1, true);

    is\_prime[0] = is\_prime[1] = false;

    vector<ll> primes;

    for (ll i = 2; i <= N; i++) {

        if (is\_prime[i]) {

            primes.push\_back(i);

            for (ll j = i \* i; j <= N; j += i) is\_prime[j] = false;

        }

    }

    ll total = 1;

    for (ll el : primes) {

        if (el \* el \* el > num) break;

        ll e = 0;

        while (num % el == 0) {

            e++;

            num /= el;

        }

        total \*= (e + 1);

    }

    ll sqr = sqrtl(num);

    if (MillerRabin(num)) total \*= 2;

    else if (sqr \* sqr == num && MillerRabin(sqr)) total \*= 3;

    else if (num != 1) total \*= 4;

    return total;

}

void solve()

{

    ll n;

    cin >> n;

    cout << countDivUpto1e18(n) << nl;

}