\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Unbounded knapsack

int knap(int W,int wt[],int val[],int n)

{

int i,w;

int k[n+1][W+1];

for(i=0;i<=n;i++)

{

for(w=0;w<=W;w++)

{

if(i==0||w==0)

k[i][w]=0;

else if(wt[i-1]<=w)

k[i][w]=max(val[i-1]+k[i-1][w-wt[i-1]],k[i-1][w]);

else k[i][w]=k[i-1][w];

}

}

return k[n][W];

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Fractional Knapsack

double fk(int w,struct Item arr[],int n)

{

int i;

sort(arr,arr+n,comp);

int curr\_weight=0;

double final\_value=0.0;

for(i=0;i<n;i++){

if(curr\_weight+arr[i].weight<=w)

{

curr\_weight+=arr[i].weight;

final\_value+=arr[i].value;

}

else {

int remain=w-curr\_weight;

final\_value+=remain\*(double)arr[i].value/arr[i].weight;

break;

}

}

return final\_value;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Modular exponentiation including string type very very large

ll ans(ll x,ll y)

{

ll res=1;

while(y>0)

{

if(y%2==1)

res=(res%M\*x%M)%M;

x=(x%M\*x%M)%M;

y=y>>1;

}

for(i=0;i<x.length();i++)

A=(10\*A+(x[i]-'0'))%M;

for(i=0;i<y.length();i++)

B=(10\*B+(y[i]-'0'))%1000000006;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Kruskal s Algo

#include <iostream>

#include <vector>

#include <utility>

#include <algorithm>

using namespace std;

const int MAX = 1e4 + 5;

int id[MAX], nodes, edges;

pair <long long, pair<int, int> > p[MAX];

void initialize()

{

for(int i = 0;i < MAX;++i)

id[i] = i;

}

int root(int x)

{

while(id[x] != x)

{

id[x] = id[id[x]];

x = id[x];

}

return x;

}

void union1(int x, int y)

{

int p = root(x);

int q = root(y);

id[p] = id[q];

}

long long kruskal(pair<long long, pair<int, int> > p[])

{

int x, y;

long long cost, minimumCost = 0;

for(int i = 0;i < edges;++i)

{

// Selecting edges one by one in increasing order from the beginning

x = p[i].second.first;

y = p[i].second.second;

cost = p[i].first;

// Check if the selected edge is creating a cycle or not

if(root(x) != root(y))

{

minimumCost += cost;

union1(x, y);

}

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

initialize();

cin >> nodes >> edges;

for(int i = 0;i < edges;++i)

{

cin >> x >> y >> weight;

p[i] = make\_pair(weight, make\_pair(x, y));

}

// Sort the edges in the ascending order

sort(p, p + edges);

minimumCost = kruskal(p);

cout << minimumCost << endl;

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PRIMS Algo

#include <iostream>

#include <vector>

#include <queue>

#include <functional>

#include <utility>

using namespace std;

const int MAX = 1e4 + 5;

typedef pair<long long, int> PII;

bool marked[MAX];

vector <PII> adj[MAX];

long long prim(int x)

{

priority\_queue<PII, vector<PII>, greater<PII> > Q;

int y;

long long minimumCost = 0;

PII p;

Q.push(make\_pair(0, x));

while(!Q.empty())

{

// Select the edge with minimum weight

p = Q.top();

Q.pop();

x = p.second;

// Checking for cycle

if(marked[x] == true)

continue;

minimumCost += p.first;

marked[x] = true;

for(int i = 0;i < adj[x].size();++i)

{

y = adj[x][i].second;

if(marked[y] == false)

Q.push(adj[x][i]);

}

}

return minimumCost;

}

int main()

{

int nodes, edges, x, y;

long long weight, minimumCost;

cin >> nodes >> edges;

for(int i = 0;i < edges;++i)

{

cin >> x >> y >> weight;

adj[x].push\_back(make\_pair(weight, y));

adj[y].push\_back(make\_pair(weight, x));

}

// Selecting 1 as the starting node

minimumCost = prim(1);

cout << minimumCost << endl;

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LAZY Propagation

void build(int node,int st,int end)

{

if(st==end)

{

tree[node]=a[st];

return;

}

int mid=(st+end)/2;

build(2\*node,st,mid);

build(2\*node+1,mid+1,end);

tree[node]=tree[2\*node]+tree[2\*node+1];

}

void updateRange(int node,int st,int end,int l,int r,int val)

{

if(lazy[node]!=0)

{

tree[node]+=(end-st+1)\*lazy[node];

if(st!=end){

lazy[2\*node]+=lazy[node];

lazy[2\*node+1]+=lazy[node];

}

lazy[node]=0;

}

if(r<st||l>end||st>end)

return;

if(l<=st&&r>=end)

{

tree[node]+=(end-st+1)\*val;

if(st!=end)

{

lazy[2\*node]+=val;

lazy[2\*node+1]+=val;

}

return;

}

int mid=(st+end)/2;

updateRange(2\*node,st,mid,l,r,val);

updateRange(2\*node+1,mid+1,end,l,r,val);

tree[node]=tree[2\*node]+tree[2\*node+1];

}

int queryRange(int node,int st,int end,int l,int r)

{

if(lazy[node]!=0)

{

tree[node]+=(end-st+1)\*lazy[node];

if(st!=end)

{

lazy[2\*node]+=lazy[node];

lazy[2\*node+1]+=lazy[node];

}

lazy[node]=0;

}

if(st>r||l>end)

return 0;

if(st>=l&&end<=r)

return tree[node];

int mid=(st+end)/2;

int p1=queryRange(2\*node,st,mid,l,r);

int p2=queryRange(2\*node+1,mid+1,end,l,r);

return(p1+p2);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXTENDED EUCLID

(A^-1)%M

void ee(ll a,ll b)

{

if(b==0){

d=a;

x=1;

y=0;

}

else

{

ee(b,a%b);

temp=x;

x=y;

y=temp-(a/b)\*y;

}

}

ll os(ll c,ll m)

{

ee(c,m);

return (x%m+m)%m;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DIAMETER

int dfs(int node,int par)

{

int mx1=0,mx2=0;

for(auto it:v[node])

{

if(it!=par&&it<=n)

{

int val=dfs(it,node);

if(val>= mx1)

{

mx2=mx1;

mx1=val;

}

else if(val>mx2)

mx2=val;

}

}

if(node==0)

return mx1+mx2;

else return mx1+1;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SIEVE

1. long long int q,l,r,i,j;
2. //memset(a,true,sizeof(a)); manually true banaaao
3. a[0]=a[1]=false;
5. for(i=2;i\*i<=N;i++){
6. if(a[i]==true){
7. for(j=i\*i;j<=N;j+=i)
8. a[j]=false;}
9. }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Segmented sieve

void simpleSieve(int limit, vector<int> &prime)

{

    // Create a boolean array "mark[0..n-1]" and initialize

    // all entries of it as true. A value in mark[p] will

    // finally be false if 'p' is Not a prime, else true.

    bool mark[limit+1];

    memset(mark, true, sizeof(mark));

    for (int p=2; p\*p<limit; p++)

    {

        // If p is not changed, then it is a prime

        if (mark[p] == true)

        {

            // Update all multiples of p

            for (int i=p\*2; i<limit; i+=p)

                mark[i] = false;

        }

    }

    // Print all prime numbers and store them in prime

    for (int p=2; p<limit; p++)

    {

        if (mark[p] == true)

        {

            prime.push\_back(p);

            cout << p << " ";

        }

    }

}

// Prints all prime numbers smaller than 'n'

void segmentedSieve(int n)

{

    // Compute all primes smaller than or equal

    // to square root of n using simple sieve

    int limit = floor(sqrt(n))+1;

    vector<int> prime;

    simpleSieve(limit, prime);

    // Divide the range [0..n-1] in different segments

    // We have chosen segment size as sqrt(n).

    int low = limit;

    int high = 2\*limit;

    // While all segments of range [0..n-1] are not processed,

    // process one segment at a time

    while (low < n)

    {

        if (high >= n)

           high = n;

        // To mark primes in current range. A value in mark[i]

        // will finally be false if 'i-low' is Not a prime,

        // else true.

        bool mark[limit+1];

        memset(mark, true, sizeof(mark));

        // Use the found primes by simpleSieve() to find

        // primes in current range

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

        {

            // Find the minimum number in [low..high] that is

            // a multiple of prime[i] (divisible by prime[i])

            // For example, if low is 31 and prime[i] is 3,

            // we start with 33.

            int loLim = floor(low/prime[i]) \* prime[i];

            if (loLim < low)

                loLim += prime[i];

            /\* Mark multiples of prime[i] in [low..high]:

                We are marking j - low for j, i.e. each number

                in range [low, high] is mapped to [0, high-low]

                so if range is [50, 100] marking 50 corresponds

                to marking 0, marking 51 corresponds to 1 and

                so on. In this way we need to allocate space only

                for range \*/

            for (int j=loLim; j<high; j+=prime[i])

                mark[j-low] = false;

        }

        // Numbers which are not marked as false are prime

        for (int i = low; i<high; i++)

            if (mark[i - low] == true)

                cout << i << " ";

        // Update low and high for next segment

        low = low + limit;

        high = high + limit;

    }

}

// Driver program to test above function

int main()

{

    int n = 100;

    cout << "Primes smaller than " << n << ":n";

    segmentedSieve(n);

    return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Counting no of multioples of a no in array

void countSieve(int arr[], int n)

{

    int MAX = \*max\_element(arr, arr + n);

    int cnt[MAX + 1];

    // ans is global pointer so that query function

    // can access it.

    ans = new int[MAX + 1];

    // Initialize both arrays as 0.

    memset(cnt, 0, sizeof(cnt));

    memset(ans, 0, (MAX + 1) \* sizeof(int));

    // Store the arr[] elements as index

    // in cnt[] array

    for (int i = 0; i < n; ++i)

        ++cnt[arr[i]];

    // Iterate over all multiples as 'i'

    // and keep the count of array[] ( In

    // cnt[] array) elements in ans[] array

    for (int i = 1; i <= MAX; ++i)

        for (int j = i; j <= MAX; j += i)

            ans[i] += cnt[j];

    return;

}

int countMultiples(int k)

{

    // return pre-calculated result

    return ans[k];

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Prime Factorisation

void primeFactors(int n)

{

    // Print the number of 2s that divide n

    while (n % 2 == 0)

    {

        cout << 2 << " ";

        n = n/2;

    }

    // n must be odd at this point. So we can skip

    // one element (Note i = i +2)

    for (int i = 3; i <= sqrt(n); i = i + 2)

    {

        // While i divides n, print i and divide n

        while (n % i == 0)

        {

            cout << i << " ";

            n = n/i;

        }

    }

    // This condition is to handle the case when n

    // is a prime number greater than 2

    if (n > 2)

        cout << n << " ";

}

All factors

void printDivisors(int n)

{

    // Note that this loop runs till square root

    for (int i=1; i<=sqrt(n); i++)

    {

        if (n%i == 0)

        {

            // If divisors are equal, print only one

            if (n/i == i)

                printf("%d ", i);

            else // Otherwise print both

                printf("%d %d ", i, n/i);

        }

    }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Catalan Number

// C++ program for nth Catalan Number

#include<iostream>

using namespace std;

// Returns value of Binomial Coefficient C(n, k)

unsigned long int binomialCoeff(unsigned int n, unsigned int k)

{

unsigned long int res = 1;

// Since C(n, k) = C(n, n-k)

if (k > n - k)

k = n - k;

// Calculate value of [n\*(n-1)\*---\*(n-k+1)] / [k\*(k-1)\*---\*1]

for (int i = 0; i < k; ++i)

{

res \*= (n - i);

res /= (i + 1);

}

return res;

}

// A Binomial coefficient based function to find nth catalan

// number in O(n) time

unsigned long int catalan(unsigned int n)

{

// Calculate value of 2nCn

unsigned long int c = binomialCoeff(2\*n, n);

// return 2nCn/(n+1)

return c/(n+1);

}

// Driver program to test above functions

int main()

{

for (int i = 0; i < 10; i++)

cout << catalan(i) << " ";

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Counting substngs with equal 0s and 1s ans 2s

#include <bits/stdc++.h>

using namespace std;

// Method to count number of substring which

// has equal 0, 1 and 2

int getSubstringWithEqual012(string str)

{

    int n = str.length();

    // map to store, how many times a difference

    // pair has occurred previously

    map< pair<int, int>, int > mp;

    mp[make\_pair(0, 0)] = 1;

    //  zc (Count of zeroes), oc(Count of 1s)

    //  and tc(count of twos)

    //  In starting all counts are zero

    int zc = 0, oc = 0, tc = 0;

    //  looping into string

    int res = 0;  // Initialize result

    for (int i = 0; i < n; ++i)

    {

        // increasing the count of current character

        if (str[i] == '0') zc++;

        else if (str[i] == '1') oc++;

        else tc++;  // Assuming that string doesn't contain

                    // other characters

        // making pair of differences (z[i] - o[i],

        // z[i] - t[i])

        pair<int, int> tmp = make\_pair(zc - oc,

                                       zc - tc);

        // Count of previous occurrences of above pair

        // indicates that the subarrays forming from

        // every previous occurrence to this occurrence

        // is a subarray with equal number of 0's, 1's

        // and 2's

        res = res + mp[tmp];

        // increasing the count of current difference

        // pair by 1

        mp[tmp]++;

    }

    return res;

}

//  driver code to test above method

int main()

{

    string str = "0102010";

    cout << getSubstringWithEqual012(str) << endl;

    return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

No of 0 1 ka sequence barabar number of them

matlab kitna aisa groups of any no possible jisme odd even baabar ho types ya 0 1 eg 10101111 me 6 ans

1. scn(n);
2. rep(i,0,n)
3. scn(ar[i]);
4. rep(i,0,n)
5. (ar[i]%2==0)?(ar[i]=-1):(ar[i]=1);
6. ll cnt=0,ans=0;
7. unordered\_map<ll,ll> m;
8. m[0]++;
9. rep(i,0,n)
10. {
11. cnt+=(ar[i]);
12. ans+=m[cnt];
13. m[cnt]++;
14. }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MINIMUM COIN CHANGE PROBLEM

int main()

{

int coins[4] = {2,4,6,7};

int n = 4,i,j;

int val = 4;

int tble[val+1];

for(i=1;i<=val;i++)

tble[i]=1000;

tble[0] = 0;

for(i=0;i<n;i++)

{

for(j = 1; j <= val; j++){

if(coins[i] <= j){

if(tble[j-coins[i]]+1<tble[j])

tble[j]=tble[j-coins[i]]+1;

}

}

}

if(tble[val]!=1000)

cout<<tble[val];

else cout<<"IMPOSSIBLE\n";

for(int i=0;i<=val;i++)

cout<<tble[i]<<" ";

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

KMP

#include<iostream>

using namespace std;

void findPrefix(string pattern, int m, int prefArray[]) {

   int length = 0;

   prefArray[0] = 0;     //first place is always 0 as no prefix

   for(int i = 1; i<m; i++) {

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

         length++;

         prefArray[i] = length;

      }else {

         if(length != 0) {

            length = prefArray[length - 1];

            i--;     //decrease i to avoid effect of increasing after iteration

         }else

            prefArray[i] = 0;

      }

   }

}

void kmpPattSearch(string mainString, string pattern, int \*locArray, int &loc) {

   int n, m, i = 0, j = 0;

   n = mainString.size();

   m = pattern.size();

   int prefixArray[m];    //prefix array as same size of pattern

   findPrefix(pattern, m, prefixArray);

   loc = 0;

   while(i < n) {

      if(mainString[i] == pattern[j]) {

         i++; j++;

      }

      if(j == m) {

         locArray[loc] = i-j;      //item found at i-j position.

         loc++;

         j = prefixArray[j-1];    //get the prefix length from array

      }else if(i < n && pattern[j] != mainString[i]) {

         if(j != 0)

            j = prefixArray[j-1];

         else

            i++;

      }

   }

}

int main() {

   string str = "AAAABAAAAABBBAAAAB";

   string patt = "AAAB";

   int locationArray[str.size()];

   int index;

   kmpPattSearch(str, patt, locationArray, index);

   for(int i = 0; i<index; i++) {

      cout << "Pattern found at location: " <<locationArray[i] << endl;

   }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sort By SEC

bool sortbysec(const pair<int,int> &a,

              const pair<int,int> &b)

{

    return (a.second < b.second);

}

sort(vect.begin(), vect.end(), sortbysec);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TOTIENT FN

**int phi(int n) {**

**int result = n;**

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

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

**while(n % i == 0)**

**n /= i;**

**result -= result / i;**

**}**

**}**

**if(n > 1)**

**result -= result / n;**

**return result;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

BIT

ll n,i,j,BIT[maxx];

void update(ll ind,ll val)

{

ind=ind+1;

while(ind<=5)// no of elements in array

{

BIT[ind]+=val;

ind+=ind&(-ind);

}

}

ll getsum(ll ind)

{

ll sum=0;

ind=ind+1;

while(ind>0)

{

sum+=BIT[ind];

ind-=ind&(-ind);

}

return sum;

}

im

{

ll arr[100]= {2, 1, 1, 3, 2};

/\*rep(i,0,12)

BIT[i]=0;\*/

//BIT[n+1];

// mem(BIT);

rep(i,0,5)

update(i,arr[i]);

cout<<getsum(4)<<"\n";// 0 se 4 tak ka sum

BIT[3]+=6;

update(3,6);

cout<<getsum(4)-getsum(0)<<"\n"; // 1 se 4 tak ka

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

COUNTING INVERSIONS(normal way)

#include<bits/stdc++.h>

using namespace std;

int getSum(int BITree[], int index)

{

int sum = 0; // Initialize result

// Traverse ancestors of BITree[index]

while (index > 0)

{

sum += BITree[index];

index -= index & (-index);

}

return sum;

}

void updateBIT(int BITree[], int n, int index, int val)

{

while (index <= n)

{

BITree[index] += val;

index += index & (-index);

}

}

int getInvCount(int arr[], int n)

{

int invcount = 0;

int maxElement = 0;

for (int i=0; i<n; i++)

if (maxElement < arr[i])

maxElement = arr[i];

int BIT[maxElement+1];

for (int i=1; i<=maxElement; i++)

BIT[i] = 0;

for (int i=n-1; i>=0; i--)

{

invcount += getSum(BIT, arr[i]-1);

updateBIT(BIT, maxElement, arr[i], 1);

}

return invcount;

}

int main()

{

int arr[] = {8, 4, 2, 1};

int n = sizeof(arr)/sizeof(int);

cout << "Number of inversions are : " << getInvCount(arr,n);

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Counting inversions (BEST WAY)

// C++ program to count inversions using Binary Indexed Tree

#include<bits/stdc++.h>

using namespace std;

int getSum(int BITree[], int index)

{

int sum = 0; // Initialize result

while (index > 0)

{

sum += BITree[index];

index -= index & (-index);

}

return sum;

}

void updateBIT(int BITree[], int n, int index, int val)

{

while (index <= n)

{

BITree[index] += val;

index += index & (-index);

}

}

void convert(int arr[], int n)

{

int temp[n];

for (int i=0; i<n; i++)

temp[i] = arr[i];

sort(temp, temp+n);

for (int i=0; i<n; i++)

{

arr[i] = lower\_bound(temp, temp+n, arr[i]) - temp + 1;

}

}

int getInvCount(int arr[], int n)

{

int invcount = 0; // Initialize result

convert(arr, n);

int BIT[n+1];

for (int i=1; i<=n; i++)

BIT[i] = 0;

for (int i=n-1; i>=0; i--)

{

invcount += getSum(BIT, arr[i]-1);

updateBIT(BIT, n, arr[i], 1);

}

return invcount;

}

int main() {

int arr[] = {8, 4, 2, 1};

int n = sizeof(arr)/sizeof(int);

cout << "Number of inversions are : " << getInvCount(arr,n);

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TOPOLOGICAL SORT

void dfs(ll u)

{

mark[u]=1;

for(auto it:v[u])

{

if(!mark[it])

dfs(it);

}

vv.pb(u);

}

im

{

ll e;

scn(n);

mark.assign(n+1,false);

scn(e);

while(e--)

{

ll x,y;

scn(x); scn(y);

v[x].pb(y);

}

for(i=0;i<n;i++)

{

if(!mark[i])

dfs(i);

}

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

cout<<vv[i]<<" ";

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TOPOLOGICAL LEXO ORDERING

vec v[maxx];

im

{

ll e;

scn(n); scn(e);

vector<ll> indegree(n+1,0);

while(e--)

{

ll x,y;

scn(x); scn(y);

v[x].pb(y);

}

for(ll u=1;u<=n;u++)

for(auto itr:v[u])

{

indegree[itr]++;

}

set<ll> s;

rep(i,1,n+1)

{

if(indegree[i]==0)

s.insert(i);

}

ll cnt=0;

vec fnl;

while(!s.empty())

{

ll u=\*s.begin();

s.erase(s.begin());

fnl.pb(u);

for(auto itr:v[u])

if(--indegree[itr]==0)

s.insert(itr);

cnt++;

}

for(i=0;i<fnl.size();i++)

cout<<fnl[i]<<" ";

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LOGG AND POW

ll logg(ll n){

1. ll c=-1;
2. **while**(n){
3. c++;
4. n/=3;
5. }
6. **return** c;
7. }
8. ll power(ll a,ll b)
9. {
10. ll res=1;
11. **while**(b){
12. **if**(b%2)
13. res=(res\*a);
14. a=(a\*a);
15. b>>=1;;
16. }
17. **return** res;
18. }
19. ll find(ll n){
20. vec v2,vis;
21. **while**(n){
22. ll lo=logg(n);
23. **//cout<<lo<<endl;**
24. v2.pb(lo);
25. ll p=power(3,lo);
26. n-=p;
27. }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LOGG AND POW

LOGG AND POW

1. ll logg(ll n){

2. ll c=-1;

3. while(n){

4. c++;

5. n/=3;

6. }

7. return c;

8. }

9. ll power(ll a,ll b)

10. {

11. ll res=1;

12. while(b){

13. if(b%2)

14. res=(res\*a);

15. a=(a\*a);

16. b>>=1;;

17. }

18. return res;

19. }

20. ll find(ll n){

21. vec v2,vis;

22. while(n){

23. ll lo=logg(n);

24. //cout<<lo<<endl;

25. v2.pb(lo);

26. ll p=power(3,lo);

27. n-=p;

28. }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Articulation Points and Bridges

void dfs(ll v,ll p)

{

vis[v]=true;

dis[v]=low[v]=tim++;

ll child=0;

for(int to:vect[v])

{

if(to==p)

continue;

if(vis[to])

{

low[v]=min(low[v],dis[to]);

}

else

{

dfs(to,v);

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

if(low[to]>=dis[v]&&p!=-1)

ans.pb(v);// stores APs

++child;

if(low[to]>dis[v])

br.pb(make\_pair(v,to));// stores edges

}

}

if(p==-1&&child>1)

ans.pb(v);

}

im

{

scn(n);

scn(e);

while(e--)

{

ll aa,bb;

scnl(aa,bb);

vect[aa].pb(bb);

vect[bb].pb(aa);

}

tim=0;

vis.assign(n,false);

dis.assign(n,-1);

low.assign(n,-1);

rep(i,0,n)

{

if(!vis[i])

dfs(i,-1);

}

sort(ans.begin(),ans.end());

auto ip=unique(ans.begin(),ans.end());

ans.resize(distance(ans.begin(),ip));

cout<<ans.size()<<"\n";

rep(i,0,ans.size()){

cout<<ans[i]<<" ";

}

cout<<"\n";

sort(br.begin(),br.end());

cout<<br.size()<<"\n";

rep(i,0,br.size()){

cout<<br[i].F<<" "<<br[i].S;

cout<<"\n";

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Strongly connected (0 based indexing)

vector<bool> mark;

vec order,comp;

vector<ll> vect[maxx];

vector<ll> rev[maxx];

void dfs1(ll v)

{

mark[v]=1;

for(auto it:vect[v])

{

if(!mark[it])

{

dfs1(it);

}

}

order.pb(v);

}

void dfs2(ll v)

{

mark[v]=1;

comp.pb(v);

for(auto it:rev[v])

{

if(!mark[it])

dfs2(it);

}

}

im

{

scnl(n,e);

while(e--)

{

ll a,b;

scnl(a,b);

vect[a].pb(b);

rev[b].pb(a);

}

mark.assign(n+1,false);

rep(i,0,n)

{

if(!mark[i])

dfs1(i);

}

mark.assign(n+1,false);

rep(i,0,n)

{

ll v=order[n-1-i];

if(!mark[v]){

dfs2(v);

for(ll j=0;j<comp.size();j++)

cout<<comp[j]<<" ";

comp.clear();

}

nl;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*