**Basics of Programming(week-1)**

5. Recursion and Backtracking

1. A binary palindrome

#include<bits/stdc++.h>

using namespace std;

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

#define endl "\n"

#define ordered\_set tree<int, null\_type,less<int>, rb\_tree\_tag,tree\_order\_statistics\_node\_update>

#define FIO ios\_base::sync\_with\_stdio(false);cin.tie(NULL);cout.tie(NULL);

#define ll long long

#define vi vector<ll>

#define pb push\_back

#define F first

#define S second

#define all(v) (v).begin(),(v).end()

#define pii pair<ll,ll>

#define vii vector<pii>

#define vb vector<bool>

#define vvi vector<vi>

#define vvb vector<vb>

#define calc\_fact(n) tgamma(n+1)

#define inf LONG\_LONG\_MAX

#define MOD 1000000007

#define mod 998244353

string to\_binary(ll num){

if(num==0){

return "0";

}

string s = "";

while(num!=0){

s+=('0' + num%2);

num/=2;

}

return s;

}

ll get\_value(string s){

ll ans = 0;

for(ll i=0;i<s.length();i++){

if(s[i]=='1'){

ans+=(1LL<<i);

}

}

return ans;

}

signed main()

{

FIO;

set<ll> store = {0,1};

for(ll num=0;num<(1<<15);num++){

string now = to\_binary(num);

string rev = now;

reverse(all(rev));

// making even length palindrome

string curr = now + rev;

if(curr.back()=='1'){

store.insert(get\_value(curr));

}

curr = rev + now;

if(curr.back()=='1'){

store.insert(get\_value(curr));

}

// making odd length palindrome

curr = now + "1" + rev;

if(curr.back()=='1'){

store.insert(get\_value(curr));

}

curr = now + "0" + rev;

if(curr.back()=='1'){

store.insert(get\_value(curr));

}

curr = rev + "1" + now;

if(curr.back()=='1'){

store.insert(get\_value(curr));

}

curr = rev + "0" + now;

if(curr.back()=='1'){

store.insert(get\_value(curr));

}

}

ll t;

cin>>t;

while(t--){

ll n;

cin>>n;

ll ans = inf;

auto itr = store.upper\_bound(n);

if(itr!=store.end()){

ans = min(ans,(ll)abs(n-(\*itr)));

}

if(itr!=store.begin()){

itr--;

ans = min(ans,(ll)abs(n-(\*itr)));

}

cout<<ans<<endl;

}

}

1. Number of divisors

#define ll unsigned long long

#include<bits/stdc++.h>

using namespace std;

ll solve(ll n,ll k)

{

ll total;

ll b=0;

ll ans;

if(n<k)

{

return (n\*(n+1)/2);

}

else if((n/k)==1)

{

return n\*(n+1)/2-k+1;

}

else

{

total=n\*(n+1)/2;

ll divisors=n/k;

ll n2=divisors\*k;

b=b+solve(n2/k,k);

ll c=((divisors)\*(2\*k+(divisors-1)\*k))/2;

return (total-c+b);

}

}

int main()

{

ll testcase;

cin>>testcase;

while(testcase--)

{

ll n,k;

cin>>n>>k;

ll a=solve(n,k);

cout<<a<<endl;

}

}

1. Divide Number

#include<iostream>

#include<vector>

using namespace std;

long long int solve(long long int N)

{

long long int ans = -1;

vector<long long int> factors;

factors.clear();

long long int simplified[15] = {2,3,4,5,6,7,8,9,10,12,15, 18,20,24,42};

if(N % 2)

return ans;

for(int i = 0 ; i < 15 ; i ++)

{

if(N % simplified[i] == 0)

factors.push\_back(N/simplified[i]);

}

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

{

for(int b = 0 ; b < factors.size(); b++)

{

for(int c = 0 ; c < factors.size(); c++)

{

for(int d = 0 ; d < factors.size(); d++)

{

if(factors[a] + factors[b] + factors[c] + factors[d] == N)

{

if(factors[a] \* factors[b] \* factors[c] \* factors[d] > ans)

ans = factors[a] \* factors[b] \* factors[c] \* factors[d];

}

}

}

}

}

return ans;

}

int main()

{

int testCases;

cin>>testCases;

long long int N;

while(testCases)

{

cin>>N;

cout<<solve(N)<<endl;

testCases--;

}

return 0;

}

1. N-Queens

#include<bits/stdc++.h>

using namespace std;

// Print the 2D vector matrix

void print (vector <vector <bool> > &a ){

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

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

cout << a[i][j] << " " ;

}

cout << endl ;

}

cout << endl ;

}

// Check if a square is attacked by a queen

bool attacked (vector <vector <bool> > &a , int x , int y ){

// Rows check

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

if (a[i][y] == 1){

return true ;

}

}

// Columns check

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

if (a[x][i] == 1){

return true ;

}

}

// Right Diagonal Check

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

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

if ((x - y) == (i - j) && a[i][j] == 1){

return true ;

}

}

}

// Left downward Diagonal Check

for (int ii = x , jj = y; ii < a.size() && jj >= 0 ; ii++ , jj--){

if (a[ii][jj]) {

return true ;

}

}

// Left upward diagonal check

for (int ii = x , jj = y; jj < a.size() && ii >= 0 ; ii-- , jj++){

if (a[ii][jj]) {

return true ;

}

}

return false ;

}

// Place n - number of queens if Possible , else return false

bool place\_queens(vector < vector <bool> > &a , int n){

if (n == 0){

return true ;

}

else {

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

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

// If attacked then skip the square

if (attacked (a , i , j )){

continue ;

}

// if not attacked then place the queen here

else {

a[i][j] = 1 ;

if (place\_queens(a , n-1) == true) {

return true;

}

else {

a[i][j] = 0 ;

}

}

}

}

}

return false ;

}

// Main Code Here

int main()

{

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

int n ; cin >> n ;

// Vector of n - rows

vector <vector <bool> > a (n) ;

// Vector of n - columns

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

a[i] = vector <bool> (n) ;

}

if (place\_queens(a , n)){

print (a ) ;

}

else {

cout << "Not possible" ;

}

}