**Light More Light**

#include<cmath>

#include<iostream>

using namespace std;

int main()

{

unsigned int n;

unsigned int x;

while(1)

{

cin>>n;

if(n==0)

break;

x=sqrt(n);

if(n==x)

cout<<"yes\n";

else

cout<<"no\n";

}

return 0;

}

**CarmichaelNumber**

// Carmichael or not.

#include <iostream>

using namespace std;

// utility function to find gcd of two numbers

int gcd(int a, int b)

{

if (a < b)

return gcd(b, a);

if (a % b == 0)

return b;

return gcd(b, a % b);

}

// utility function to find pow(x, y) under given modulo mod

int power(int x, int y, int mod)

{

if (y == 0)

return 1;

int temp = power(x, y / 2, mod) % mod;

temp = (temp \* temp) % mod;

if (y % 2 == 1)

temp = (temp \* x) % mod;

return temp;

}

// This function receives an integer n and finds if it's a Carmichael number

bool isCarmichaelNumber(int n)

{

for (int b = 2; b < n; b++) {

// If "b" is relatively prime to n

if (gcd(b, n) == 1)

// And pow(b, n-1)%n is not 1, return false.

if (power(b, n - 1, n) != 1)

return 0;

}

return 1;

}

// Driver function

int main()

{

cout << isCarmichaelNumber(500) << endl;

return 0;

}

**Euclid Algoritham**

#include <bits/stdc++.h>

using namespace std;

int x,y,d;

// ax + by = d store x, y, and d as global variables

void extendedEuclid(int a, int b)

{

if (b == 0) {

x = 1; y = 0; d = a; return;

} // base case

extendedEuclid(b, a % b); // similar as the original gcd

int x1 = y;

int y1 = x - (a / b) \* y;

x = x1;

y = y1;

}

int main() {

int a,b;

//while(scanf("%d %d",&a,&b) != EOF) {

extendedEuclid(17,17);

printf("%d %d %d\n",x,y,d);

}

**Factovisior**

|  |
| --- |
| int p[5500], pt = 0; |
| void sieve() { |
| int i, j, m[50000] = {}; |
| for(i = 2; i < 50000; i++) |
| if(m[i] == 0) { |
| p[pt++] = i; |
| for(j = i+i; j < 50000; j += i) |
| m[j] = 1; |
| } |
| } |
| int solve(int n, int m) { |
| int i; |
| for(i = 0; i < pt && p[i]\*p[i] <= m; i++) { |
| if(m%p[i] == 0) { |
| int cnt = 0; |
| while(m%p[i] == 0) |
| cnt++, m /= p[i]; |
| long long tmp = p[i], cnt2 = 0; |
| while(tmp <= n) { |
| cnt2 += n/tmp; |
| tmp \*= p[i]; |
| } |
| if(cnt2 < cnt) return 0; |
| } |
| } |
| if(m != 1) { |
| if(n < m) return 0; |
| } |
| return 1; |
| } |
| int main() { |
| sieve(); |
| int n, m; |
| while(scanf("%d %d", &n, &m) == 2) { |
| if(solve(n, m)) |
| printf("%d divides %d!\n", m, n); |
| else |
| printf("%d does not divide %d!\n", m, n); |
| } |
| return 0; |
| } |

**Summetion of four primes**

#include <bits/stdc++.h>

using namespace std;

int isPrime(int x)

{

int s = sqrt(x);

for (int i = 2; i <= s; i++)

if (x % i == 0)

return 0;

return 1;

}

void Num(int x, int& a, int& b)

{

for (int i = 2; i <= x / 2; i++) {

if (isPrime(i) && isPrime(x - i)) {

a = i;

b = x - i;

return;

}

}

}

void generate(int n)

{

if (n <= 7)

cout << "Impossible" << endl;

int a, b;

if (n % 2 != 0) {

Num(n - 5, a, b);

cout << "2 3 " << a << " " << b << endl;

}

else {

Num(n - 4, a, b);

cout << "2 2 " << a << " " << b << endl;

}

}

int main()

{

int n = 28;

generate(n);

return 0;

}**Smith Number**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int count(int n) {

char s[12];

sprintf(s, "%d", n);

int i, sum = 0;

for(i = 0; s[i]; i++)

sum += s[i]-'0';

return sum;

}

int C(int n) {

int i, ans;

int digits1 = count(n), digits2 = 0, flag = 1;

ans = n;

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

if(n%i == 0) {

int time = 0;

while(n%i == 0) {

time++;

n /= i;

}

digits2 += time\*count(i);

flag = 0;

}

}

if(n != 1)

digits2 += count(n);

if(digits1 == digits2 && flag == 0) {

printf("%d\n", ans);

return 1;

}

return 0;

}

int main() {

int i;

int T, n;

scanf("%d", &T);

while(T--) {

scanf("%d", &n);

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

if(C(i) == 1)

break;

}

return 0;

}

**Marbles**

#include <bits/stdc++.h>

using namespace std;

#define ll long long

ll x, y, d;

void extendedEuclid(ll a, ll b) {

if(b==0) { x=1; y=0; d=a; return;

}

extendedEuclid(b, a%b);

ll y1 = x-(a/b)\*y;

x = y;

y = y1;

}

int main() {

ll v,n1,n2,c1,c2;

while(scanf("%lld",&v),v){

scanf("%lld %lld %lld %lld",&c1,&n1,&c2,&n2);

extendedEuclid(n1,n2);

if (v%d != 0) {

printf("failed\n");

} else {

x \*= v/d;

y \*= v/d;

n2 /= d, n1 /= d;

ll lowerbound=ceil(-(double)x/n2);

ll upperbound=floor((double)y/n1);

if(lowerbound<=upperbound) {

// compare cost

ll res1 = c1\*(x+n2\*lowerbound) + c2\*(y-n1\*lowerbound);

ll res2 = c1\*(x+n2\*upperbound) + c2\*(y-n1\*upperbound);

if (res1 < res2) {

printf("%lld %lld\n",(x+n2\*lowerbound), (y-n1\*lowerbound));

} else {

printf("%lld %lld\n",(x+n2\*upperbound), (y-n1\*upperbound));

}

} else

printf("failed\n");

}

}

}

**QUEUE**

#include<stdio.h>

#include<string.h>

#include<iostream>

#include<algorithm>

#define LL long long

using namespace std;

LL dp[22][22][22];

void fun()

{

int N,P,R;

dp[1][1][1]=1;

for(N=2;N<=13;N++)

{

for(P=1;P<=N;P++)

{

;

for(R=1;R<=N;R++)

{

dp[N][P][R]=dp[N-1][P][R-1]+dp[N-1][P-1][R]+(N-2)\*dp[N-1][P][R];

}

}

}

}

int main()

{

int t;

scanf("%d",&t);

fun();

while(t--)

{

int N,P,R;

scanf("%d%d%d",&N,&P,&R);

printf("%lld\n",dp[N][P][R]);

}

}

**Hanoi tower trouble again**

#include <bits/stdc++.h>

using namespace std;

// O(1), mathematical proof by induction of 1,2,3...

// another approach is greedy simulatioN always pick existing leftmost peg

int main()

{

int t;

int n;

cin >> t;

while(t--){

cin >> n;

cout << (n+1)\*(n+1)/2-1 << endl;

}

}

**Disting Subsequence**

#include<bits/stdc++.h>

using namespace std;

int dp[1001][1001];

int findans(string &a,string &b,int n,int m)

{

if( n==0 && m>0) // returnig 0 if m is 0 and m is positive

return 0;

if(m==0) // returning the 1 when m is zero which means we have to count this also

return 1;

if(dp[n][m]!=-1) // returning the value if already calculated

return dp[n][m];

// if both values are equal then we have to make two calls

// first -> finding the another character for the current charcters

// second -> considering the both the current character ann calling for

// next character of the both the string

if(a[n-1]==b[m-1])

return dp[n][m]=findans(a,b,n-1,m)+findans(a,b,n-1,m-1);

else

return dp[n][m]=findans(a,b,n-1,m); // if values are not equal then checking for next character in the string s

}

int numDistinct(string s, string t) {

int n=s.size();

int m=t.size();

memset(dp,-1,sizeof(dp));

return findans(s,t,n,m);

}

int main()

{

string s="babgbag";

string t="bag";

cout<<numDistinct(s,t);

return 0;

}