Red John is Back ☆

We can convert the first part of the problem into a recurrence relation i.e. F(N) = F(N-1) + F(N-4) with the base case : F(0) = F(1) = F(2) = F(3) = 1.

Here, F(N) represents the number of ways of tiling the 4xN rectangle with 4x1 and 1x4 tiles. The explanation goes as :

If we place a 4x1 tile, then we just need to solve for F(N-1).

If we place a 1x4 tile, then we can't place a 4x1 tile under it. Basically, we will have to place a set of four 1x4 tiles together, hence we solve forF(N-4).

Finally, we take the sum and get the total number of configuration i.e. M.

Once we get M, we need to to calculate the number of primes less than or equal to M. We use Sieve of Eratosthenes(http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes) to do the same.

The time complexity of the first part of the problem is O(n) since we can calculate M using Dynamic Programming. The time complexity of the second part is O(n log log n). Hence, the overall time complexity for the problem becomes O(n log log n).

Problem Setter's code:

C++

```
  #include <iostream>
#include <vector>
#include <set>
#include <cmath>

using namespace std;

vector<int> sieve(int n)
{
    set<int> primes;
    vector<int> vec;

    primes.insert(2);

    for(int i=3; i<=n ; i+=2)
    {
        primes.insert(i);
    }

    int p=*primes.begin();
    vec.push_back(p);
    primes.erase(p);

    int maxRoot = sqrt(*(primes.rbegin()));

    while(primes.size() > 0)
    {
        if(p > maxRoot)
        {
            while(primes.size() > 0)
            {
                p=*primes.begin();
                vec.push_back(p);
                primes.erase(p);
            }
            break;
        }

        int i = p*p;
        int temp = (*(primes.rbegin()));

        while(i<=temp)
```

```
        {
            primes.erase(i);
            i += p;
            i += p;
        }

        p=*primes.begin();
        vec.push_back(p);
        primes.erase(p);
    }

    return vec;
}

int main()
{
    int t, N, M, P, i, count, res;
    vector<int> prime;
    vector<int>::iterator it;

    cin >> t;

    prime = sieve(217286);

    int a[41];
    for(i=0;i<4;i++)
        a[i] = 1;
    for(i=4;i<=40;i++)
        a[i] = a[i-1] + a[i-4];

    int numprime[41];
    for(i=0;i<41;i++)
    {
        count = 0;
        for(it=prime.begin();it!=prime.end();it++)
        {
            if(*it <= a[i])
                count++;
            else
                break;
        }
        numprime[i] = count;
        count = 0;
    }

    while(t--)
    {
        cin >> N;
        res = numprime[N];
        cout << res << endl;
    }

    return 0;
}
```

Feedback

Was this editorial helpful?

Yes **No**