Sherlock and Coprime Subset

Attempted by: **207** / Accuracy: **59%** / Maximum Score: **30** / ★★★★★ 4 Votes

Tag(s): Algorithms, Bitmask, Dynamic Programming, Medium

Problem Description

You are given N numbers in an array where each number can be in range [1, 50]. You need to find out the largest possible subset of numbers that are coprime. Two numbers are coprime if they dont share any prime factors between them (or GCD of both of them is 1). The problem asks to find largest subset such that no two pair of numbers in that subset shares a prime factor.

Solution

There are 50 numbers in the array, so there can 2^{50} possible subsets, that we need to consider. Of course if we are going to check each of the subsets, one by one checking if that has no co prime pairs, it will take a lot of time. We need to think of something else. When we are doing the above process, we are redoing lots of things. This is where Dynamic Programming comes into picture, to store and resuse the computed information.

Firstly, observe that there are only 15 prime numbers inside the first 50 natural numbers. Any number in the array will have only prime factors among these 15 number. This does mean we can store these as masks (or seive). Lets say we have number 6, with factors 2 and 3. Then we will set the first two bits of the mask. This means that in future, if we want to extend this set (of number 6), then we should avoid taking numbers that have factors of already set bits in this set (here the first two bits).

Okay, so here is the idea then. Firstly compute for each number in the array, the indices of the prime factors. So if number has a factor 2, we will store 1 in the list, if 5 then we will store 3, as 5 is the 3rd prime. Now lets define our DP recurrence. We will maintain a 2D array, dp[index][mask] which says, given a mask *mask*, what is the largest subset of co primes that can be generated from all the numbers in and after *index* position. So dp[index][mask] gives the answer to the question, what is the largest subset on and after index. This enalbes us to create a function rec(index, mask) whose initial call as (0, 0) {first 0 says all numbers from 0 and afterwards, second 0 says initally mask is empty} will give us the answer to the problem. There will be two options to consider, weather include the current index to the subset, or dont include it.

- Dont include - Simply return rec(index + 1, mask).
- Include - Firstly check if the current index is suitable to be included or not. Run a loop and check if mask is set for a position in the prime factor list we created in the beginning for this index. If there is any prime which is already in the mask, we cannot include to simply return the answer to rec(index + 1, mask). Othwerise, this is fit to be included, so tamper the mask to set all the bits for the positions which are prime factors of this number. Then returned value should be 1 + rec(index + 1, newMask). 1 indicates we included this number, and passed the newMask (setting all appropriate bits) and getting answer from rec(index + 1, newMask).

Ofcourse the answer should be the maximum of both the cases and that should be the return value of our call to rec(index, mask).

Finally, a few things worhty to note. We can check if bit is set or not, by using bitwise AND [mask & (1 << primePosition)]. Similarly, we can set a bit by using bitwise OR [mask & (1 << primePosition)]. To avoid recacluation of the same function rec(index, mask), we can maintain a table initialized by -1. Firstly we ch if dp[index][mask] is -1 or not. If not we return becuase that has been already calcualted. Othwerise w

calculate as above and also store dp[index][mask], so that we can reuse it later. Complexity ? Because each entry of the table is computed only once, We can make atmost (N * 2^{15}) calls to the function. Inside the function we just run a loop for size of prime list for that index which can be atmost 15. Therefore, overall complexity is O(N * 15 * 2^{15}). See testers code for neat implementation of the idea.

Author Solution by Lalit Kundu

```
#include<bits/stdc++.h>
using namespace std;
#define assn(n,a,b) assert(n<=b && n>=a)
#define pb push_back
#define mp make_pair
#define clr(x) x.clear()
#define sz(x) ((int)(x).size())
#define F first
#define S second
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,b) for(i=0;i<b;i++)
#define rep1(i,b) for(i=1;i<=b;i++)
#define pdn(n) printf("%d\n",n)
#define sl(n) scanf("%lld",&n)
#define sd(n) scanf("%d",&n)
#define pn printf("\n")
typedef pair<int,int> PII;
typedef vector<PII> VPII;
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long LL;
#define MOD 1000000007
LL mpow(LL a, LL n)
{LL ret=1;LL b=a;while(n) {if(n&1)
    ret=(ret*b)%MOD;b=(b*b)%MOD;n>>=1;}
return (LL)ret;}
int prime[16]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
int dp[102][(1<<15)+10]={};
int arr[105];
int p,n;
int solve(int cur, int mask)
{
    if(cur==n || mask==p)return 0;
    if(dp[cur][mask]!=-1)return dp[cur][mask];
    int q=solve(cur+1,mask);
    if(arr[cur]==1)return dp[cur][mask]=q+1;
    int temp=mask;
```

```
    for(int i=0; i<15; i++)
        if(arr[cur]%prime[i]==0)
        {
            if(mask&(1<<i))return dp[cur][temp]=q;
            mask=mask|(1<<i);
        }
    return dp[cur][temp]=max(1+solve(cur+1,mask),q);
}
int main()
{
    int t;
    cin >> t;
    assn(t,1,10);
    cout << t << endl;
    while(t--)
    {
        cin >> n;
        cout << n << endl;
        assn(n,1,50);
        for(int i=0; i<n; i++)
        {
            cin >> arr[i];
            if(i!=n-1)cout << arr[i] << " ";
            else cout << arr[i] << "\n";
            assn(arr[i],1,50);
        }
        p=(1<<n)-1;
        memset(dp,-1,sizeof(dp));
//         cout << solve(0,0) << endl;
    }
    return 0;
}
```

Tester Solution

```
// asdasdasda as dasd
#include <bits/stdc++.h>
using namespace std;
#define FREP(b) for(int i=0;i<b;i++)
#define REP(a,b,c) for(int a=b;a<c;a++)
#define asd(x)                cout<<__LINE__<<" :: "<<#x<< ": "<<x<<endl;
#define asdf(x, y)            cout<<__LINE__<<" :: "<<#x<< ": "<<x<<" | "
<<#y<< ": "<<y<<endl;
typedef pair<int,int> ii;
typedef long long LL;

int dp[53][1 << 15], prime[15] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47},
A[53], n;
vector<int> G[51];
void scan(){
    cin >> n;
    REP(i, 0, n) cin >> A[i];
}
```

```
int rec(int id, int mask){
    if(id == n) return 0;
    int &ret = dp[id][mask];
    if(ret != -1) return ret;

    if(A[id] == 1) return ret = 1 + rec(id + 1, mask);
    ret = rec(id + 1, mask);
    REP(i, 0, G[A[id]].size()){
        int u = G[A[id]][i];
        if(mask & (1 << u)) return ret;
        mask |= (1 << u);
    }
    ret = max(ret, 1 + rec(id + 1, mask));
    return ret;
}

int solve(){
    REP(i, 0, 51){
        G[i].clear();
        REP(j, 0, 15){
            if(prime[j] > i) break;
            if(i % prime[j] == 0) G[i].push_back(j);
        }
    }
    memset(dp, -1, sizeof dp);
    return rec(0, 0);
}

int main(){
    int test;
    cin >> test;
    while(test--){
        scan();
        cout << solve() << endl;
    }
    return 0;
}
```

?