CodeChef Discussion

ANUCBC - Editorial

29

4

PROBLEM LINK:

Practice
Contest

Author: Anudeep Nekkanti
Tester: Shiplu Hawlader and Mahbubul Hasan
Editorialist: Lalit Kundu

DIFFICULTY:

MEDIUM

PREREQUISITES:

Dynamic Programming
Precomputation

PROBLEM:

Given N numbers, and a number M, we need to find number of subsets such that sum of elements of subset is divisible by M.
1 <= N <= 100000
1 <= M <= 100
At most 3 test cases with 30 queries in each test. That is 90 queries at most.

EXPLANATION:

If you are not familiar with dynamic programming, read topcoder tutorial here first.
So, here we define a state DP[i][j] which stores the number of ways to choose subsets from $A_0...A_i$ such that subset sum modulo M is j.
Now,

```
DP[i][j]= DP[i-1][j] + // we don't choose the i'th element
          DP[i-1][(j-A[i])%M]  // we choose the i'th element
```

Our answer will be DP[N-1][0]. But here, complexity will be N*M for each query, which will time out.

The advantage here we have is that M<=100. A[i] doesn't matter to us. What matters is what A[i]%M is. So,

```
for i=0 to M-1:
     count[i]=number of A[j] such that A[j]%m=i
```

The above array count, can be calculated using hashing.
Again, we have to use dynamic programming.

We define a new state DP[i][j] which stores, the number of subsets which consists of only those A_j's whose modulo M is less than or equal to i, and subset sum modulo M is j.
choose(n,r) is number of ways to choose r elements out of n distinct elements ie. (n!)/((r!)*(n-r)!) Now,

```
DP[i][j] =
DP[i-1][(j-i*0)%M]*choose(count[i],0)+// we don't pick any one of the count[i] choices we have.
DP[i-1][(j-i*1)%M]*choose(count[i],1)+// we can pick any one of the count[i] choices we have.
DP[i-1][(j-i*2)%M]*choose(count[i],2)+// we can pick any two of the count[i] choices we have.
.
.
.
DP[i-1][(j-i*count[i])%M]*choose(count[i],count[i])// we pick all of the count[i] choices we have
```

But, the complexity here will still remain N*M. We can rewrite the above as:

```
DP[i][j] =
DP[i-1][(j-0)%M]*summation[k=0 to count[i], where (k*i)%M==0]{choose(count[i],k)} +
DP[i-1][(j-1)%M]*summation[k=0 to count[i], where (k*i)%M==1]{choose(count[i],k)} +
.
.
```

```
.
DP[i-1][(j-(M-1))%M]*summation[k=0 to count[i], where (k*i)%M==(M-1)]{choose(count[i],k)}
```

We can precompute all the summation terms above for each query in O(N).
So, complexity will be O(M*M*M) for each query.

AUTHOR'S AND TESTER'S SOLUTIONS:

To be updated soon.

april14 anucbc medium dynamic-prog editorial

This question is marked "community wiki".

asked 14 Apr '14, 15:38
darkshadows ◆◆
2.9k●88●133●148
accept rate: 6%

edited 14 Apr '14, 16:41
wittyceaser
3.3k●19●42●69

---

13 Answers:

oldest newest most voted

1 2 next »

8

@wittcyeaser We can precompute all the summation terms for each query in O(N)
following code will help you understand how to do it ...

```
summation[M] = {0};
for (int k = 0; k <= count[i]; k++) {
    summation[(k*i)%M] += choose(count[i],k);
}
```

For each i we need to compute this summation[] only once and after precomputing this summation[] array we can use it for filling all dp[i][j] values in following way ..

```
for (int j = 0; j < M; j++) {
   dp[i][j] = 0;
   for (int k = 0; k < M; k++) {
      dp[i][j] += dp[i-1][j-k]*summation[k];
   }
}
```

basically this procedure will take O(M*M) time for each i

link | award points

edited 14 Apr '14, 21:34

answered 14 Apr '14, 17:48
dawnavd
251●4●5●10
accept rate: 0%

You have incremented i while computing summation, here k is always 0. If the loop is in terms of k, then we should have to compute summation for i=0 to M-1, so the complexity becomes O(MN), and the solution wont be accepted. I am not sure if I am correct. If I am wrong someone please point out my mistake.

jawad (14 Apr '14, 19:37)

oh sorry that was a mistake
now its correct ...

dawnavd (14 Apr '14, 20:58)

Then what is the value of i while calculating summation

jawad (14 Apr '14, 21:03)

"i" will go from 1 to M-1
dp[0][0] = 2^count[0]
dp[0][j] = 0
or If you want to run "i" from 0 to M-1 then
dp[-1][0] = 1
dp[-1][j] = 0

dawnavd (14 Apr '14, 21:11)

1 No that loop will run for count[i] time for each i
and i varying from 1 to M-1
so in total it'll run sigma(count[i]) times
which is O(N)

dawnavd (14 Apr '14, 21:19)

showing 5 of 8 show all

4

@anudeep2011 really a nice question followed by a equally nice editorial.
That's what I like about codechef...:)

link | award points

edited 14 Apr '14, 15:49

answered 14 Apr '14, 15:42
ronakymca
1.1k●3●12●23
accept rate: 19%

Can someone please explain the last step where the 'summation' term is introduced?

3

link | award points

answered 14 Apr '14, 17:27
wittyceaser
3.3k●19●42●69
accept rate: 16%

1 let m = 30 and 7 occurs 1000 times. Thus by using 7 there will be 1001 (one 0 sum) different possible sums. But we know m = 30 hence they will boil down to max of 30 different sums. Hence instead of calculating for each possible sum (which would take O(1001 X 30) for one row). We pre-compute for all different sums less than 30 (in O(10001)) and than in O(30 * 30) we can fill a row.

vishfrnds (14 Apr '14, 18:53)

Thanks @vishfrnds

wittyceaser (15 Apr '14, 07:25)

My O(NM) solution barely passes (in Java) Probably should have made TL a bit more strict; I didn't even realize I needed to optimize further. At least there is a lot to be learned from this editorial.

2

link | award points

edited 14 Apr '14, 23:24

answered 14 Apr '14, 23:22
lg5293
331●2●8
accept rate: 14%

@lg5293:Can you explain your solution.

herman (16 Apr '14, 09:59)

My solution is basically the first solution presented in the editorial (the one that should supposedly time out).

lg5293 (17 Apr '14, 02:44)

ok.Thanks.

herman (17 Apr '14, 21:54)

What about the complexity to calculate choose(n,k)? Is there an algorithm which takes O(1) time? While solving the problem, I had the presumption that calculating it would require O(N*N)(using dynamic programming, since count[i] can be atmost N) in the worst case and hence I eliminated the option of using combinations

1

link | award points

edited 14 Apr '14, 22:06

answered 14 Apr '14, 22:02
midhul
76●5
accept rate: 0%

1 yeah choose(n,k) % P can be calculated in O(1) with some pre computation
precomputate fact[i] array which is (i!)%P
and ifact[i] array which is Modular multiplicative inverse of i! modulo P
then choose(n,k)%P = (fact[n]*ifact[n-k]*ifact[k])%P
precomputation of these arrays can be done in in O(NlogP)

dawnavd (14 Apr '14, 22:20)

Thanks! That seemed quite straight forward (in the sense that this is how we naturally calculate choose n,k (using the formula)). So we can compute nck with just O(n) pre-computation (with an extra logP for inverse calculation :))

midhul (14 Apr '14, 22:34)

Just miss the last O(m*m*m)trick next time sure

0

link | award points

answered 14 Apr '14, 16:28
unsungwarrior
16●2
accept rate: 0%

Ok, so basically:

0

1. count[i] is pre-computed in O(N).
2. Each query is answered in O(M*M):

```
    i. If i'th element is chosen, we can choose M different modular sums - O(M)
    ii. The number of ways of choosing the k'th modular sum has to be found, for which we have to scan
all 'M' values in the count[ ] array - O(M)
```

Hence, overall complexity is O(N + M*M*M).

Is that correct?

link | award points

answered 15 Apr '14, 07:39
wittyceaser
3.3k●19●42●69
accept rate: 16%

Yes, per query the complexity is O(N + M*M*M)

anudeep2011 (19 Apr '14, 14:57)

Dynamic programming problems are really very hard to think..

0

link | award points

answered **16 Apr '14, 21:17**

princerk
708•7•11•23
accept rate: 5%

Can someone please explain this. As choose(count[i],k1) and choose(count[i],k2) can go to indices in the summation array and later on get added up while filling up the dp[I][J] array. So how can we make sure that k1 + k2 has a size less than the total size of count[i] ?

0

link | award points

answered **17 Apr '14, 16:42**

vermashubhang
1.5k•1•10•25
accept rate: 23%

We actually do it in order. We have M classes of numbers. We start from class 0 and go till class M-1, at each class we decide on how many numbers we need to take from that class, we also need the remainder when we move to next state. We move from state DP[i][carry] to DP[i+1][NewCarry], where i is the class and carry is the current sum modulo M

anudeep2011 (19 Apr '14, 14:59)

Did the O(mmm) thing, but used top-down (recursive) DP. Got TLE and got stuck...

0

link | award points

answered **18 Apr '14, 03:14**

caioaao
91•2•6
accept rate: 0%

1 Give me link. I will see. Also note that Modulo operator is very costly and so excessive use might have resulted in TLE.

anudeep2011 (19 Apr '14, 15:00)

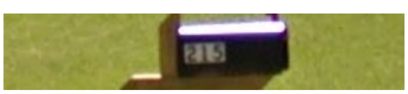
http://www.codechef.com/viewsolution/3759961 (sorry for the late reply, I was traveling)

caioaao (25 Apr '14, 15:58)

1 2 **next »**

Your answer

[hide preview] community wiki



Type the text
Privacy & Terms

Post Your Answer

About CodeChef | About Directi | CEO's Corner
CodeChef Campus Chapters | CodeChef For Schools | Contact Us