Nama : Fajar Satria

NRP : 05111940000083

Kelas : B

Resume Kuliah PAA – 17 Mei 2021

Algoritma Dynamic Programming biasa diselesaikan dengan metode top down. Akan tetapi metode tersebut tidak selalu lebih efisien daripada metode bottom up. Biasanya dalam competitive programming, seseorang akan memikirkan bagaimana cara menyelesaikan masalah sehingga pendekatan yang diambil adalah metode top down. Oleh karena itu program yang dihasilkan pun tidak selalu efisien. Sedangkan pada mata kuliah PAA ini, hal yang harus dicari adalah bagaimana cara menyelesaikan masalah dengan resource sekecil mungkin dan se-efisien mungkin. Sehingga metode bottom up akan menjadi pilihan terbaik. Berikut adalah beberapa soal yang dapat diselesaikan dengan Dynamic Programming bottom up.

a. Lanterns

PERMASALAHAN

You need to install night lighting along the road. Working team has already installed **n** lanterns, it remains only to turn them on. To save money, it was decided to turn on such number of lanterns so that it will be safe to ride on this road. Road is considered safe if any its section does not contain **k** neighboring switched off lanterns.

Your task is to figure out in how many ways you can turn on illumination on the road so as to go through it is safely. Two variants of illumination are considered different if there is at least one lantern that is turned on in the first version, and is turned off in the second, or vice versa. Because the answer may be very large, output the answer modulo $10^9 + 7$.

SOLUSI

Permasalahan inti pada soal ini adalah mencari banyaknya konfigurasi dari **n** lampu. Dengan ketentuan tidak boleh ada **k** lampu yang mati berturut-turut.

No	Lampu 1	Lampu 2	Lampu 3
1	0	0	0
2	0	0	1
3	0	1	0
4	1	0	0
5	0	1	1
6	1	0	1
7	1	1	0
8	1	1	1

misal n=3 dan k=3

Pada tabel diatas 0 adalah ketika lampu mati, dan 1 adalah ketika lampu menyala. Semua konfigurasi yang mungkin untuk $n = 3$ adalah 8. Namun karena terdapat 1 konfigurasi lampu yang mati berturut-turut sebanyak $k = 3$. Maka hasilnya adalah $res = 8 - 1 = 7$.

Untuk mempermudah memahami lebih lanjut mengenai hubungan **n** dan **k** maka dapat dibuat tabel seperti berikut.

		N								
		0	1	2	3	4	5	6	7	8
K	1	1	1	1	1	1	1	1	1	1
	2	1	2	3	5	8	13	21	34	55
	3	1	2	4	7	12	20	33	54	88
	4	1	2	4	8	15	26	44	73	120
	5	1	2	4	8	16	31	54	92	153
	6	1	2	4	8	16	32	63	110	188
	7	1	2	4	8	16	32	64	127	222
	8	1	2	4	8	16	32	64	128	255

Pada tabel diatas terbagi menjadi 4 bagian. Warna hijau menandakan terminal case $a = 0$ untuk semua k hasilnya adalah 1. Warna kuning menandakan $a < k$ maka hasilnya adalah $(2 * n_{a-1})$. Warna biru menandakan $a = k$ maka hasilnya adalah $(2 * n_{a-1}) - 1$. Warna putih menandakan $a > k$ maka hasilnya adalah $(2 * n_{a-1}) - n_{a-k-1}$.

Dikarenakan konfigurasi yang sangat banyak, maka nilai yang dihasilkan harus di modulo $10^9 + 7$. Agar tidak terjadi *overflow*, proses modulo dapat dijalankan disetiap operasi.

PSEUDOCODE

SOLVE(n, k)

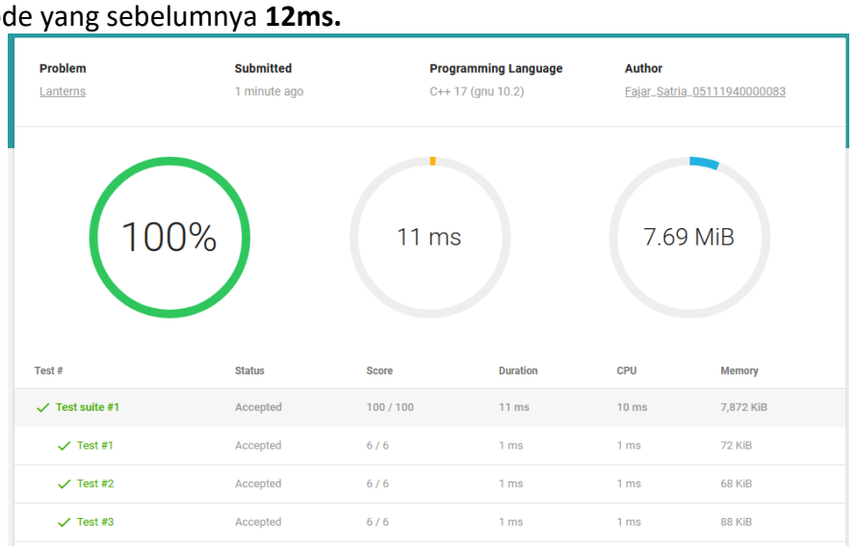
1. let dp[0..n] be a new array
2. MOD = 1000000007
3. dp[0] = 1
4. for a=1 to n
5. if(a<k)
6. dp[a] = (2 * dp[a-1] % MOD)
7. else if(a==k)
8. dp[a] = (2 * dp[a-1] % MOD) - 1
9. else
10. dp[a] = (2 * dp[a-1] % MOD) – (dp[a-k-1])
11. if dp[a]<0
12. dp[a] = dp[a] + MOD
13. return dp[n]

SOURCECODE

```
#include<cstdio>
using namespace std;
#define MOD 1000000007
long long n, k, dp[1000001];
int main(){
    scanf("%lld %lld",&n,&k);
    dp[0]=1;
    for(ll a=1;a<=n;a++){
        if(a<k)
            dp[a]=(2*dp[a-1]%MOD);
        else if(a==k)
            dp[a]=(2*dp[a-1]%MOD)-1;
        else{
            dp[a]=(2*dp[a-1]%MOD)-dp[a-k-1];
            if(dp[a]<0) dp[a]=dp[a]+MOD;
        }
    }
    printf("%lld\n",dp[n]);
    return 0;
}
```

SUBMISSIONS

Dengan sedikit memodifikasi pada **if** didapatkan solusi yang lebih cepat, yakni **11ms** dari kode yang sebelumnya **12ms.**



b. Ladders

PERMASALAHAN

Ladder is a set of cubes in which each over the top layer contains fewer blocks than the previous one. Count the number of ladders that can be built from **n** blocks.

SOLUSI

Permasalahan pada soal ini cukup mudah dipahami. Yakni menghitung banyaknya tangga minimal yang digunakan untuk setiap konfigurasi yang memenuhi syarat ladders. Atau susunan block, dimana block yang lebih atas harus lebih kecil atau sama dengan block dibawahnya.

n = 3

Pada gambar diatas, terlihat bahwa untuk $n = 3$ memiliki 2 konfigurasi yang mungkin yaitu {3} dan {2,1} sehingga tangga yang dibutuhkan adalah 2. Begitupun untuk $n = 4$, akan memiliki 2 konfigurasi {4} dan {2,2} sehingga tangga yang dibutuhkan adalah 2.

Dalam penghitungannya dapat menggunakan solusi dynamic programming bottom up dan jalannya bukan forward melainkan backward. Hal ini dilakukan agar tidak terjadi perulangan terus menerus pada nilai **i** yang kecil dari **dp**. Misalnya pada dp[2], seharusnya dp[1] tidak harus dihitung meskipun keduanya memiliki relasi rekurens. Jadi setiap dp[i] hanya perlu dihitung sebanyak 1x.

Solusi yang saya gunakan menggunakan **dp** sebanyak **n** maksimal atau **dp[101]** dari yang semula membutuhkan sebanyak **2n**. Hal ini dapat dilakukan dengan menambahkan statement if baru.

PSEUDOCODE

SOLVE(n)

1. let dp[0..n] be new array
2. dp[0] = 1
3. for a=1 to n

4. for j=n to 0
5. if dp[j]
6. if j+I <= 100
7. dp[j+i] = dp[j+i] + dp[j]
8. return dp[n]

SOURCECODE

```
#include <cstdio>
using namespace std;
int n,dp[101];
int main(){
    scanf("%d",&n);
    dp[0]=1;
    for(int i=1;i<=n;i++)
        for(int j=n;j>=0;j--)
            if(dp[j])
                if(j+i<=100)
                    dp[j+i] += dp[j];
    printf("%d\n",dp[n]);
    return 0;
}
```

SUBMISSIONS

Dengan sedikit menambahkan statement **if** didapatkan solusi yang menggunakan memory lebih sedikit. Dari yang semula 70 KiB menjadi 68 KiB.

#	Submit date	Lang	Time	CPU	Memory	State
9005484	May 17, 2021, 10:08:17 AM	C++ 17 (gnu 10.2)	1 ms	1 ms	68	✓ Accepted
9005451	May 17, 2021, 10:04:00 AM	C++ 17 (gnu 10.2)	1 ms	1 ms	70	✓ Accepted