

Nama : Fajar Satria

NRP : 05111940000083

Kelas : B

### Resume Kuliah PAA – 21 Juni 2021

Pada pertemuan kali ini, difokuskan untuk mengeksplorasi bagian divide and conquer. Berikut adalah beberapa soal yang cukup menarik untuk dibahas.

#### a. AU7\_2 - SERVERS

##### PERMASALAHAN

There are  $N$  ( $1 \leq N \leq 100000$ ) servers and each has a fixed serving time  $T_i$  irrespective for any single task.

There are  $M$  ( $1 \leq M \leq 1000000000$ ) tasks which needs to processed in sequential order one after another.

The tasks need not be processed immediately, if the servers are free. They can wait and assigned to a faster server if necessary.

Find a schedule such that the total time needed for processing all tasks is minimized.

**Example:**  $N=2, M=6, T_1 = 7, T_2 = 10$

Optimal Schedule:

server1 processes task1 and server2 processes task2.

after 7 seconds, server1 is free and task3 is assigned to server1.

after 10 seconds. server2 is free and task4 is assigned to server2.

after 14 seconds. server1 is free and task5 is assigned to server1.

after 20 seconds. server2 is free and task6 is not assigned to server2. It waits for 1 second.

then after 21 seconds server1 is free and task6 is assigned to server1.

After 28 seconds all tasks are completed.

On other hand, if task6 was immediately assigned to server2 without waiting the total time would have been 30seconds.

##### SOLUSI

Terdapat beberapa solusi yang dapat digunakan untuk menyelesaikan permasalahan ini. Namun permasalahan pada soal ini sebenarnya sudah terlihat cukup berat. Hal tersebut terlihat dari banyaknya task yang bisa mencapai 1000000000.

Solusi yang pertama adalah dengan menggunakan Algoritma Greedy. Yaitu dengan memanfaatkan *priority\_queue structure*. Jadi untuk tiap task, dilakukan pemilihan server yang dapat menyelesaikan task tercepat. Namun solusi ini masih belum cukup optimal karena masih memiliki *time complexity*  $O(M \log N)$ .

Solusi yang kedua adalah dengan menggunakan prinsip Binary Search. Yaitu dengan menduga sebuah waktu untuk dapat menyelesaikan semua task. Rentang waktu yang mungkin adalah antara 0 second sampai dengan  $t_{min} * m$  second. Dimana  $t_{min}$  adalah lamanya pengerjaan pada server yang paling cepat dan  $m$  adalah banyaknya task.

Untuk mengatur **low/high** pada binary search adalah dengan melakukan pengecekan apakah waktu sebanyak **mid** dapat digunakan untuk menyelesaikan **m** task pada **n** server. Jika waktu sebanyak **mid** cukup, maka **high=mid**, selain itu **low=mid**. Hal tersebut dilakukan berulang ulang jika **high** masih lebih besar dari **low**. Sehingga **high** merupakan solusi akhir pada permasalahan ini. Solusi ini optimal karena memiliki *time complexity*  $O(N \log M)$ .

## PSEUDOCODE

ENOUGH (T,time,n,m)

1. cnt = 0
2. for i=0 to n-1
3.     here = time / T[i]
4.     if here>=m or here+cnt>=m
5.         return true
6.     cnt = cnt + here
7. if cnt>=m
8.     return true
9. else
10.    return false

SOLVE(T,n,m)

1. Tmin = 100000
2. for i=0 to n-1
3.     if T[i]<tmin
4.         Tmin = T[i]
5. Low = 0
6. high = Tmin\*m
7. while high > low
8.     mid = (low+high)/2
9.     if ENOUGH(T,mid,n,m)
10.         high = mid
11.    else
12.         low = mid
13. return high

## SOURCECODE

```
#include <cstdio>
#define MAXN 100002
typedef long long LL;
int T,n,m;
int t[MAXN],tmin;
template<typename T>
T getNum() {
    T res=0;
    char c;
    while(1){
        c = getchar_unlocked();
        if(c==' ' || c=='\n') continue;
        else break;
    }
    res=c-'0';
    while(1){
        c = getchar_unlocked();
        if(c>='0' && c<='9') res=10*res+c-'0';
        else break;
    }
    return res;
}
bool enough(LL time){
    LL cnt=0;
    for(int i=0;i<n;i++){
        LL here = time / t[i];
        if(here>=m || here+cnt>=m)
            return true;
        cnt+=here;
    }
    return cnt>=m;
}
```

```
int main() {
    T=getNum<int>();
    while(T--){
        n=getNum<int>();
        m=getNum<int>();
        tmin=100000;
        for(int i=0;i<n;i++){
            t[i]=getNum<int>();
            if(t[i]<tmin) tmin=t[i];
        }
        LL low=0,high=(LL)tmin*m;
        while(high-low>1){
            LL mid = (low+high)/2;
            (enough(mid)?high:low)=mid;
        }
        printf("%lld\n",high);
    }
    return 0;
}
```

### SUBMISSIONS

Saya melakukan beberapa submission.

Pada bagian range binary search





untuk high menggunakan **t[0]\*m** mendapatkan waktu 0.15 tanpa fast IO

Sedangkan

Untuk high menggunakan **tmin\*m** malah mendapatkan waktu 0.25 tanpa fast IO

Padahal dalam teori, menggunakan **tmin** akan memiliki waktu yang lebih singkat.

Dan ketika menggunakan fast IO keduanya mendapatkan waktu 0.12.

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
28088613		2021-06-21 10:32:17	SERVERS	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.12	5.3M	CPP
28088201		2021-06-21 08:52:42	SERVERS	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.12	5.4M	CPP
28088161		2021-06-21 08:48:00	SERVERS	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.25	5.4M	CPP
28088140		2021-06-21 08:46:10	SERVERS	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.15	5.4M	CPP

### b. BWB - Black and White beads

#### PERMASALAHAN

Well its now time for some serious task . There are lots of beads available and in two colours , namely white and black . So there are many beads of both the colours . One has to make a string of beads by joining beads with end to end . But there are some constraint , and you have to follow that constraint. While making beads , you have to make sure that there should not be K beads of black colour consecutively and also there should not be any bead string which has black bead in front ,i.e it must have white bead in front . So the task is quite simple , find all possible ways of making a string of bead of length N which satisfies the above constraint.

#### SOLUSI

Permasalahan pada soal ini cukup mudah dipahami. Yakni mencari banyaknya kemungkinan untuk menyusun beads. Terdapat 2 aturan yaitu tidak boleh ada **k** warna hitam berturut-turut dan untuk awalan harus berwarna putih.

Solusi permasalahan ini adalah dengan menggunakan prinsip inklusi eksklusi. Yakni untuk mencari banyaknya konfigurasi yang benar adalah dengan menghitung semua konfigurasi dikurangi dengan konfigurasi yang salah.

Berikut adalah tabel **dp[K][N]** atau tabel konfigurasi yang tidak memenuhi aturan.

		N								
		1	2	3	4	5	6	7	8	9
K	1	1	3	7	15	31	63	127	255	511
	2	0	1	3	8	19	43	94	201	423
	3	0	0	1	3	8	20	47	107	238
	4	0	0	0	1	3	8	20	48	111
	5	0	0	0	0	1	3	8	20	48
	6	0	0	0	0	0	1	3	8	20
	7	0	0	0	0	0	0	1	3	8
	8	0	0	0	0	0	0	0	1	3
	9	0	0	0	0	0	0	0	0	1

Tabel diatas didapatkan dari komputasi tiap N dan K. Untuk penjabaran perhitungannya adalah sebagai berikut.

Misal untuk  $K=3$  dan  $N=\{1,2,3,4,5,6\}$ . Karena memerlukan relasi rekuren untuk Dynamic Programming, maka tabel dibedakan menjadi 2 warna yaitu warna mocca melambangkan  **$2 \times$  (banyak konfigurasi salah sebelumnya)** dan warna biru melambangkan **konfigurasi salah yang baru**.

- Pada N=1 dan N=2 menghasilkan 0 karena pasti semua konfigurasi memenuhi.
- Pada N=3, menghasilkan 1 konfigurasi salah.
- Pada N=4, menghasilkan 3 konfigurasi salah,  $2 * \text{konfigurasi}[3] + 1 = 2 * 1 + 1$ .
- Pada N=5, menghasilkan 8 konfigurasi salah,  $2 * 3 + 2$ .
- Pada N=6, menghasilkan 20 konfigurasi salah,  $2 * 8 + 4$ .

Dst.

		I																			
1	2	3			4				5					6							
K=3	NA	NA	1	1	1	0	1	1	1	0	0	1	1	1	0	0	0	1	1	1	
						1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	
						1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	
										1	0	1	1	1	0	1	0	1	1	1	
										1	1	1	1	1	0	1	1	1	1	1	
										1	1	1	1	0	0	1	1	1	1	0	
										1	1	1	0	0	0	1	1	1	0	0	
										1	1	1	0	1	0	1	1	0	1		
																1	0	0	1	1	1
																1	0	1	1	1	1
																1	0	1	1	1	0
																1	1	0	1	1	1
																1	1	1	1	1	1
																1	1	1	1	1	0
																1	1	1	1	0	0
																1	1	1	1	0	1
													1	1	1	0	0	0			
													1	1	1	0	0	1			
													1	1	1	0	1	0			
													1	1	1	0	1	1			

Kembali lagi ke inklusi eksklusi, untuk mencari konfigurasi yang benar adalah dengan mencari semua konfigurasi dikurangi konfigurasi yang salah. Semua konfigurasi adalah  $2^N$ . Konfigurasi yang salah dapat diambil dari tabel **dp[K][N-1]**. Karena solusi harus dimodulo **10e9+7** maka tiap tiap iterasi pada tabel **dp** harus dilakukan modulo. Dan untuk menghindari salah hitung konfigurasi yang benar, maka ketika jumlah konfigurasi benar adalah **kurang dari 0**, maka harus ditambah **10e9+7**.

### PSEUDOCODE

```
SOLVE(N,K)
1. let dp[0..K][0..N] be a new 2D array
2. let pow[0..10000] be a new array
3. MOD = 1000000007
4. pow[0] = 1
5. pow[1] = 2
6. for i=2 to 10000
7.     pow[i] = (pow[i-1]*2)%MOD
8. for i=1 to 100
9.     for j=1 to 10000
10.        if i>j
11.            dp[i][j] = 0
12.        else if i==j
13.            dp[i][j] = 1
14.        else
15.            temp = pow[ j - (i+1) ] - dp[i][ j - (i+1) ]
16.            if temp<0
17.                temp = temp + MOD
18.            dp[i][j] = (( 2 * dp[i][j-1]) % MOD + temp ) %MOD
19.
20. temp = pow[N-1]-dp[K][N-1]
21. if temp<0
22.     temp = temp + MOD
23. return temp
```

### SOURCECODE

```
#include <cstdio>
#define MOD 1000000007
template<typename T>
T getNum() {
    T res=0;
    char c;
    while(1){
        c = getchar_unlocked();
        if(c==' ' || c=='\n') continue;
        else break;
    }
    res=c-'0';
    while(1){
        c = getchar_unlocked();
        if(c>='0' && c<='9') res=10*res+c-'0';
        else break;
    }
    return res;
}
int dp[101][10001];
int main() {
    int pow[10001];
    pow[0]=1;
```

```
pow[1]=2%MOD;
for(int i=2;i<=10000;i++)
    pow[i]=(pow[i-1]*2)%MOD;
for(int i=1;i<=100;i++){
    for(int j=1;j<=10000;j++){
        if(i>j){
            dp[i][j]=0;
        }else if(i==j){
            dp[i][j]=1;
        }else{
            int temp=pow[j-(i+1)] - dp[i][j-(i+1)];
            if(temp<0) temp+=MOD;
            dp[i][j]=((2*dp[i][j-1])%MOD+temp)%MOD;
        }
    }
}
int T,N,K;
T=getNum<int>();
while(T--){
    N=getNum<int>();
    K=getNum<int>();
    int temp=pow[N-1]-dp[K][N-1];
    if(temp<0) temp+=MOD;
    printf("%d\n",temp);
}
return 0;
}
```

SUBMISSIONS

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
28088374		2021-06-21 09:30:50	Black and White beads	<div>accepted</div> <div><a href="#">edit</a> <a href="#">ideone it</a></div>	0.02	5.3M	CPP