

Алгоритъм

(codeforces.com/contest/1181/problem/D)

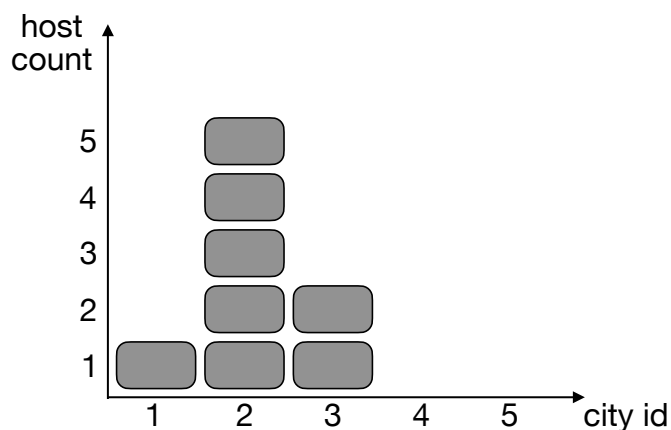
Да разгледаме следния вход:

```
8 5 4
3 2 1 2 2 3 2 2
11
16
10
34
```

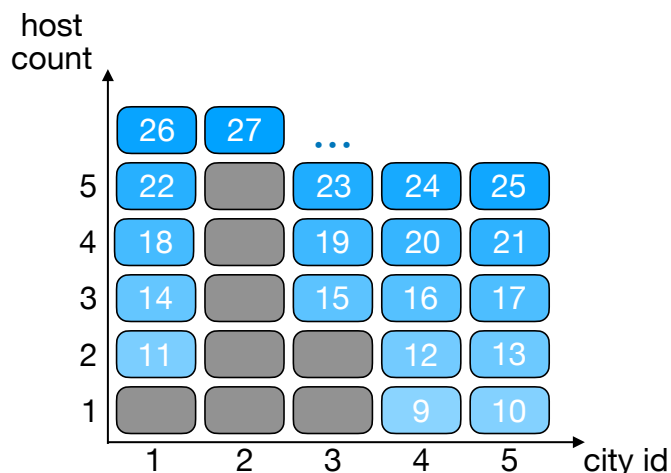
Имаме $n=8$ олимпиади, които са се провели през изминали години, както и реда на $8^{\text{те}}$ града, в които са се провели. Имаме още общия брой градове $m=5$, както и номерацията им и $q=4$ заявки.

Как бихме отговорили на заявките, ако не разполагаме с компютър (т.е. ръчно с груба сила)? Хубаво е да се отбележи, че заявките могат да достигнат до 10^{18} , което означава, че няма да може да отговорим на абсолютно всички възможни заявки и да ги съхраним и просто да ги адресираме, когато дойде някоя конкретна на входа.

Ще направим хистограма, в която по абсцисата са наредени градовете, сортирани по идентификационен номер, а по ординатата ще маркираме пътите, в които съответния град е бил домакин на олимпиадата. За конкретния пример по-горе, хистограмата ще има следния вид:



За да отговорим на въпроса кой е следващия град, който ще бъде домакин на олимпиадата е достатъчно да попълним едно блокче в хистограмата на съответния град и да го номерираме с годината, в която този град е бил домакин. Попълването на хистограмата става отдолу нагоре и отляво надясно. Това е така, тъй като по условие, с приоритет е градът с най-малко домакинства (отдолу нагоре), а от два града с еднакъв брой домакинства – с приоритет ще бъде този, който е с по-малък идентификационен номер (отляво надясно).



За заявка с аргумент k (година), отговора е номера на колоната (града), в която се намира $k^{\text{тото}}$ блокче при по-горе описаното попълване на хистограмата.

Какви наблюдения може да направим от прилагането на този алгоритъм? Първото нещо, което може да забележим е, че ако годината от заявката има номер k , който е по-голям от $m \cdot \text{maxCnt}$, където maxCnt е най-големия брой домакинства за даден град от първите n години, то отговора на тази заявка ще е просто $k \pmod m$. В случай, че модулното деление даде резултат 0, трябва просто да върнем m . Това е така, тъй като, ако $m \cdot \text{maxCnt} < k$, то това ще означава, че няма да ни се налага да пропускаме град при попълването на хистограмата. Всички градове вече ще са имали по maxCnt домакинства. С други думи, това което ни затруднява е да отговорим на заявките с $k \leq m \cdot \text{maxCnt}$.

За да отговорим на една такава заявка е необходимо да знаем колко колони сме пропускали при попълването до k или казано на обратно – колко колони сме запълвали при попълването до k . Тези колони ще натрупваме в декартово дърво (Treap), тъй като ключовете ще са индекси на градове, т.е. уникални числа и освен това в него лесно може да отговаряме на заявки от вида „кой е k -тия елемент в множеството“.

Когато дойде заявка k от подредените във възходящ ред по година заявки, отговора ще се изчисли като от k извадим броя блокчета необходими да „залеем“ блокчето което търсим от предходната заявка, разделим по модул от размера на декартовото дърво (т.е. на броя на колоните, в които сме попълвали блокчетата) и получим число order . След това запааметим на идентификационния номер на заявката – града, който е order по големина в декартовото дърво.

* под „залеем“ блокче z ще разбираме че сме в началото на реда, който се намира непосредствено над реда, в който се намира z .

По този начин ние не се интересуваме от реда, в който градовете от по-долните редове на хистограмата са идвали, а само от техния брой.

Да приложим описания алгоритъм по-горе на практика върху дадения ни вход.

Инициализация:

	cities						Сортираме градовете във възходящ ред първо по броя домакинства през първите n години и после по идентификационен номер.
hosted →	∅	0	0	1	2	5	
id →	∅	4	5	1	3	2	
	0	1	2	3	4	5	

	queries				Сортираме заявките във възходящ ред по година.
year →	10	11	16	34	
id →	2	0	1	3	
	0	1	2	3	

	ans				Създаваме масив, в който ще попълваме отговорите на заявките. Отговора на заявка k е ans[k]
value →	0	0	0	0	
index →	0	1	2	3	

Treap T



size=0

Създаваме празно декартово дърво T.

Инициализираме:

next=prev=n+1

queryIndex=0, cityIndex=1

Алгоритъм:

Докато размера на декартовото дърво е по-малък от m (т.е. докато колоните/градовете, които не прескачаме са по-малко от всички колони/градове):

- вземаме текущата заявка Q;
- ако не сме прескочили/подминали годината на заявката
 - запаметяваме prev=next и вкарваме текущия град в декартовото дърво (този град от следващия ред няма да го прескачаме, а ще трупаме блокчета над него, когато запълваме хистограмата);
 - актуализираме next като натрупваме блокчетата, които ще поставим в хистограмата, за да се озовем на реда над следващия град;
 - преминаваме на следващия град;
- ако годината на заявката е задмината, то ние може да я изчислим (тоест ако годината на заявката се е озовала между текущите два града от горния случай)
 - запаметяваме я в масива с отговори на позиция Q.id и е равна на $order+1^{-\text{вия}}$ град от декартовото дърво, където $order = Q.year - prev \pmod{T.size()};$
 - преминаваме на следващата заявка

cityIndex
↓

cities						
hosted →	∅	0	0	1	2	5
id →	∅	4	5	1	3	2
	0	1	2	3	4	5

queryIndex
↓

queries				
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	0	0
index →	0	1	2	3

next=prev=9
cityIndex+=1

Treap T



Добавяме град с id=4 в декартовото дърво, тъй като няма да го прескачаме докато запълваме хистограмата по-нагоре

Следващият град отново е с брой домакинства равен на нула, следователно next няма да се актуализира.

cityIndex
↓

cities						
hosted →	∅	0	0	1	2	5
id →	∅	4	5	1	3	2
	0	1	2	3	4	5

queryIndex
↓

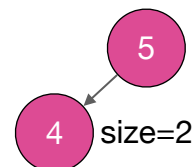
queries				
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	0	0
index →	0	1	2	3

next=11, prev=9
cityIndex+=1

Treap T



Добавяме град с id=5

Следващият град има 1 домакинство. Това означава, че ще умножим разликата между това домакинство и домакинствата от предния град (1-0) по броя на градовете, които няма да пропускаме. Те са толкова колкото е размера на декартовото дърво. Следователно, next+=(1-0)*2. next=11, което е на реда **над** текущата заявка за година 10. Следователно на следващата итерация ще може да отговорим на тази заявка.

cityIndex
↓

cities						
hosted →	∅	0	0	1	2	5
id →	∅	4	5	1	3	2
	0	1	2	3	4	5

queryIndex
↓

queries				
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

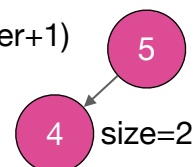
value →	0	0	5	0
index →	0	1	2	3

order=Q.year-prev=10-9=1

ans[Q.id]=T.kthSmallest(order+1)

queryIndex+=1

Treap T



Добавяме град с id=5

Отговаряме на текущата заявка и преместваме курсора на следващата заявка, за да проверим дали и за нея е изпълнено условието за „заливане“ на текущата заявка: Q.year<next.

cityIndex
↓

cities						
hosted →	∅	0	0	1	2	5
id →	∅	4	5	1	3	2
	0	1	2	3	4	5

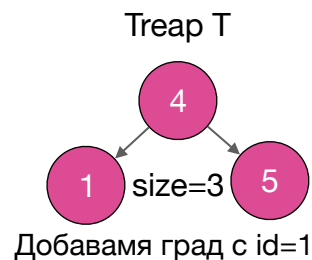
queryIndex
↓

queries				
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	5	0
index →	0	1	2	3

next=14, prev=11
cityIndex+=1



Следващият град има 2 домакинства. Това означава, че ще умножим разликата на тези две домакинства и домакинствата от текущия град (2-1) по броя на градове, които няма да пропускаме. Те са толкова колкото е размера на декартовото дърво. Следователно, next+=(2-1)*3. next=14, което е на реда **над** текущата заявка за година 11. Следователно на следващата итерация ще може да отговорим на тази заявка.

cityIndex
↓

cities						
hosted →	∅	0	0	1	2	5
id →	∅	4	5	1	3	2
	0	1	2	3	4	5

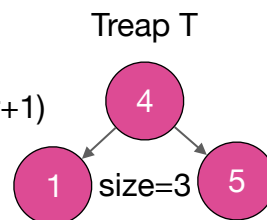
queryIndex
↓

queries				
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	1	0	5	0
index →	0	1	2	3

order=Q.year-prev=11-11=0
ans[Q.id]=T.kthSmallest(order+1)
queryIndex+=1



Отговаряме на текущата заявка и преместваме курсора на следващата заявка, за да проверим дали и за нея е изпълнено условието за „заливане“ на текущата заявка: Q.year<next.

cityIndex
↓

cities						
hosted →	∅	0	0	1	2	5
id →	∅	4	5	1	3	2
	0	1	2	3	4	5

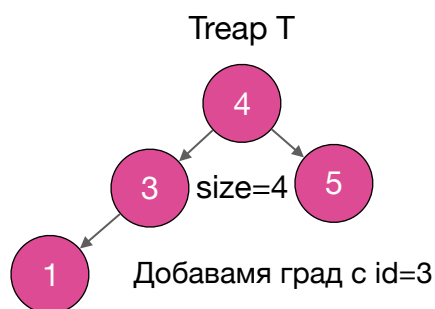
queryIndex
↓

queries				
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	5	0
index →	0	1	2	3

next=26, prev=14
cityIndex+=1



Следващият град има 5 домакинства. Това означава, че ще умножим разликата на тези 5 домакинства и домакинствата от текущия град (5-2) по броя на градове, които няма да пропускаме. Те са толкова колкото е размера на декартовото дърво. Следователно, next+=(5-2)*4. next=26, което е на реда **над** текущата заявка за година 16. Следователно на следващата итерация ще може да отговорим на тази заявка.

cityIndex
$$\text{order} = Q.\text{year} - \text{prev} = 16 - 14 = 2$$

```
ans[Q.id]=T.kthSmallest(order+1)
```

```
queryIndex+=1
```



queryIndex

Отговаряме на текущата заявка и преместваме курсора на следващата заявка, за да проверим дали и за нея е изпълнено условието за „заливане“ на текущата заявка: $Q.year < next$.

ans

Treap T

ans

```

graph TD
    4((4)) --> 2((2))
    4 --> 5((5))
    2 --> 1((1))
    2 --> 3((3))
    
```

size=5

6

Сложност по памет:

Използваме масив за градовете, заявките и отговорите на заявките. Освен това имаме и декартово дърво, което може да достигне максимален размер равен на броя на всички градове.

Общата сложност по памет е от порядъка на **$O(m+q)$** .