Дизайн и анализ на алгоритъма

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

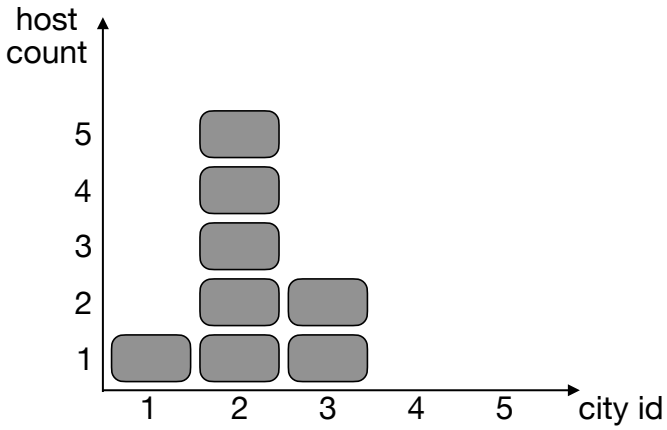
Да разгледаме следния вход:

```
8 5 4
3 2 1 2 2 3 2 2
1 1
1 6
1 0
3 4
```

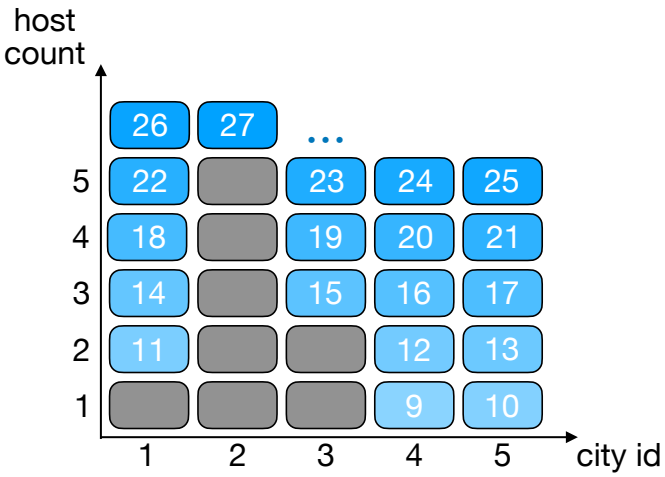
Имаме n=8 олимпиади, които са се провели през изминали години, както и реда на $8^{-те}$ града, в които са се провели. Имаме още общия брой градове m=5, както и номерацията им и q=4 заявки.

Как бихме отговорили на заявките, ако не разполагахме 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 отговотим 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 еднакакъв 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 $k^{-тото}$ 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*maxCnt, където maxCnt енай-големия брой домакинства за даден град от първите n години, то отговора на тази заявка ще е просто k (mod m). В случай, че модулното деление даде резулта 0, трябва просто да върнем m. Това е така, тъй като, ако m*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 $k \leq m*maxCnt$.

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 ḱ-тия елемент в множеството“.

Когато дойде заявка k от подредените във възходяш 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 блокчета) и получим число order. След това запаметим 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

hosted →	Ø	0	0	1	2	5
id →	Ø	4	5	1	3	2
	0	1	2	3	4	5

Сортираме градовете във възходящ ред първо по броя домакинства през първите n години и после по идентификационен номер.

Treap T

size=0

Създаваме празно декартово дърво T.

queries

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

Сортираме заявките във възходящ ред по година.

ans

value →	0	0	0	0
index →	0	1	2	3

Създаваме масив, в който ще попълваме отговорите на заявките. Отговора на заявка k е ans[k]

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-вия град от декартовото дърво, където order = Q.year - prev (mod T.size( ));
  - преминаваме на следващата заявка

cityIndex

cities

hosted →	Ø	0	0	1	2	5
id →	Ø	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

size=1

Добаваме 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 →	Ø	0	0	1	2	5
id →	Ø	4	5	1	3	2
	0	1	2	3	4	5

queryIndex

queries

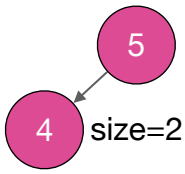
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	0	0
index →	0	1	2	3

next=11, prev=9
cityIndex+=1

Treap T



size=2

Добавамя 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 →	Ø	0	0	1	2	5
id →	Ø	4	5	1	3	2
	0	1	2	3	4	5

queryIndex

queries

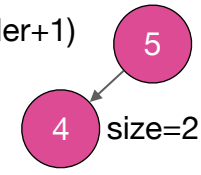
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

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



size=2

Добавамя 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 →	Ø	0	0	1	2	5
id →	Ø	4	5	1	3	2
	0	1	2	3	4	5

queryIndex

queries

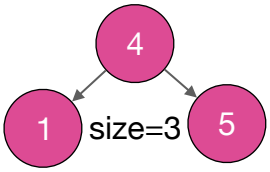
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	5	0
index →	0	1	2	3

next=14, prev=11
cityIndex+=1

Treap T



Добавамя град с id=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 заявката за година 11. Следователно на следващата итерация ще може да отговорим на тази заявка.

---

cityIndex

cities

hosted →	Ø	0	0	1	2	5
id →	Ø	4	5	1	3	2
	0	1	2	3	4	5

queryIndex

queries

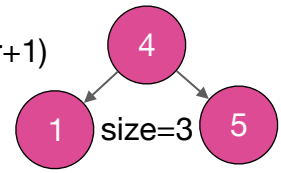
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

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

Treap T



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 →	Ø	0	0	1	2	5
id →	Ø	4	5	1	3	2
	0	1	2	3	4	5

queryIndex

queries

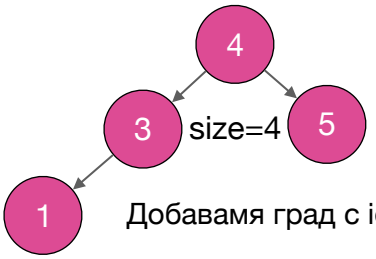
year →	10	11	16	34
id →	2	0	1	3
	0	1	2	3

ans

value →	0	0	5	0
index →	0	1	2	3

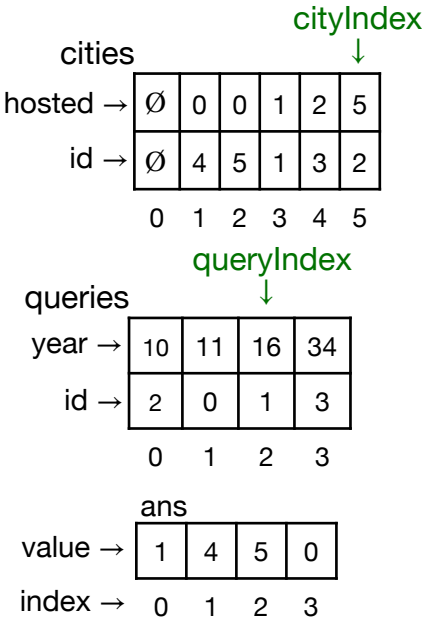
next=26, prev=14
cityIndex+=1

Treap T

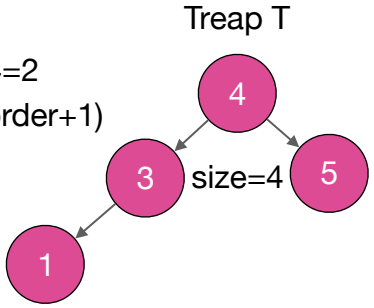


Добавамя град с id=3

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 заявката за година 16. Следователно на следващата итерация ще може да отговорим на тази заявка.

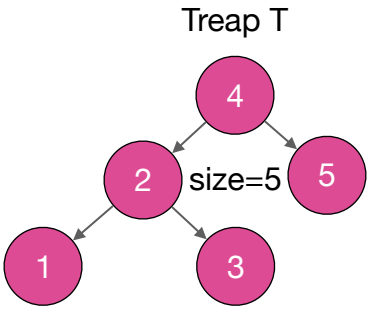


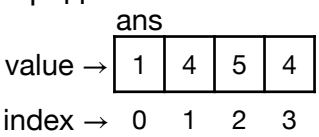
order=Q.year-prev=16-14=2
ans[Q.id]=T.kthSmallest(order+1)
queryIndex+=1



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.

Ако сме стигнали до тук, вече няма значение колко ще изчислим да е next, тъй като в декартовото дърво вече сме добавили всички градове/колони, което означава че от този момент натам, в хистограмата няма да прескачаме нито един град и отговора се пресмята тривиално с модулно деление на годината на заявката по броя на всички градове.





Времева сложност:

Сортирането на градовете ни струва O(m*log(m)), а сортирането на заявките ни струва O(q*log(q)) времева сложност. За прочитането и въвеждането на домакинствата от първите n града ще ни струва O(n) времева сложност.

В цикъла, в който се отговарят заявките се случва точно едно от двете:

- вмъква се елемент в декартово дърво, което може да достигне максимален размер m и инкрементираме индекса на градовете;
- взимаме order по големина елемент от декартовото дърво, което може да достигне максимален размер m и инкрементираме индекса на заявките;

И в двата случая правим една заявка със сложност O(log(m)) и инкрементираме един от индексите на градовете или заявките. Сумарно получаваме времева сложност за целия цикъл от порядъка на O((m+q)*log(m))

Следователно, цялата времева сложност на алгоритъма се свежда до O(max(q*log(q), n, (m+q)*log(m))), което в повечето случай ще резултира в **O((m+q)*log(m))**.

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)**.