АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА КУЛИ

Струва ни се, че най-добре е да започнем обясненията, като разгледаме идеите за

Решения за една кула

В основата на всички решения лежи идеята кулата да се постави на всички възможни места в редицата от сгради и за всяко място да се преброят сградите, които ще получават съобщения. Мястото, на което се поставя кулата ще определяме с израза „непосредствено след сграда с номер *m*", което означава, че кулата се поставя между сгради с номера *m* и *m+1* или след сграда с номер *N*. Ясно е, че, ако кулата се постави преди първата сграда, то броят на сградите, които получават съобщенията от нея, ще бъде равен на 0.

Наивно решение със сложност $O(N^2)$

Да си представим, че сме поставили кулата непосредствено след сграда с номер *m*. Ще броим сградите, които ще получават съобщение от нея, движейки се по редицата от кулата към сграда номер 1. Ясно е, че придвижването трябва да продължи или докато обходим всички сгради, или докато срещнем сграда, по-висока от кулата, тъй като след нея никоя сграда няма да получава сигнал. Другото нещо, което трябва да се съобрази е, че сградите, които ще получават сигнал ще бъдат разположени в настващ ред на своите височини при този начин на обхождане (от кулата към първата сграда). Това ни дава основание да броим по следния начин: поддържаме една променлива, в която пазим текущия максимум на височините на обходените сгради. Всеки път, когато този максимум се смени, т.е. срещнем сграда, по-висока от досегашната най-висока сграда, която сме срещнали, към брояча на сградите, които ще получават съобщения, добавяме 1.

Променяйки *m* от 1 до *N* и пресмятайки всеки път броя на сградите, които ще получават съобщения при позиция на кулата, непосредствено след сграда с номер *m*, лесно намираме максималния брой сгради, които ще получават съобщения при подходящо разположение на кулата.

Тъй като всяко обхождане на сградите от кулата до сграда 1 или до по-висока от кулата сграда е със сложност $O(m)$ и тъй като *m* се променя от 1 до *N*, то общата сложност на алгоритъма е $O(N^2)$.

Следващото решение, което ще разгледаме също е със сложност $O(N^2)$, но, както ще видим, използва идея, която впоследствие ще ни доведе до линейно решение в случай на една кула.

Решение със сложност $O(N^2)$, използващо стек

Отново си представяме, че сме поставили кулата непосредствено след сграда с номер *m*. Този път ще броим сградите, които ще получават нейните съобщения, започвайки от първата сграда и движейки се последователно до сграда с номер *m*. Ще казваме, че сграда с номер *j* покрива сграда с номер *i* $(j>i)$, ако $h_j>h_i$ и между тях няма сграда, която е по-висока от сграда с номер *i*. В светлината на това определение нашата

задача е да намерим максималния брой сгради, които кулата може да покрива при избор на подходяща позиция.

Идеята на това решение е, тръгвайки от началото на редицата от сгради и движейки се към кулата, постоянно да следим кои сгради вече са покрити от друга сграда – ясно е, че те няма как да бъдат покрити от кулата и да получават съобщения от нея. Когато стигнем до кулата ще искаме да имаме списък на обходените сгради, които още не са покрити, подреден в нарастващ ред на височините им. Тогава кулата ще може да покрие ония от тях, които са с по-малка височина от нея.

За да реализираме тази идея, ще използваме стек, в който във всеки момент на обхождането ще се намират височините на ония сгради, които още не са покрити от друга сграда. Височините в стека ще бъдат подредени в нарастващ ред от върха към „дъното“ му, както ще стане ясно от следващите изречения. Нека сме стигнали до сграда с номер *j*. Последователно премахваме от върха на стека всички сгради (по-точно височините им) с височина, по-малка от тази на сграда номер *j* – те ще бъдат покрити от сграда номер *j* и не могат да получават съобщения от кулата. След като стигнем до сграда във върха на стека с по-голяма височина от тази на сграда *j* или до „дъното“ на стека, добавяме височината на сграда с номер *j* във върха на стека.

Да напомним, че сме поставили кулата непосредствено след сграда с номер *m*. Тогава обхождаме сградите от номер 1 до номер *m*, изпълнявайки описаните операции със стека. След като добавим височината на сграда номер *m* в стека, сме готови да преброим колко сгради ще покрие кулата, т.е. колко сгради ще получават съобщенията ѝ. За целта премахваме последователно от върха на стека всички височини, по-малки от височината на кулата (да напомним, че те са сортирани в нарастващ ред от върха към дъното), като при всяко премахване прибавяме единица към брояча на покриваните от кулата сгради.

Променяйки *m* от 1 до *N* и пресмятайки всеки път броя на сградите, които ще получават съобщения при позиция на кулата, непосредствено след сграда с номер *m*, лесно намираме максималния брой сгради, които ще получават съобщения при подходящо разположение на кулата.

Тъй като намирането на броя покривани сгради при всяко *m* е със сложност $O(m)$ и тъй като *m* приема стойности от 1 до *N*, то общата сложност на алгоритъма е $O(N^2)$. Въпреки че сложността на това решение е същата, както и на предното, то е добро с това, че позволява усъвършенстване до линейно решение.

Решение със сложност $O(N)$

За да достигнем до линейно решение, трябва по-добре да си представим как „работи“ идеята с използване на стек. Възлов тук е въпросът: „трябва ли за всяка позиция на кулата, непосредствено след сграда с номер *m*, когато *m* се променя от 1 до *N*, да започваме отначало пресмятането на броя сгради, които се покриват от кулата, т.е. ще получават съобщения от нея?“. Нека си представим, че имаме брояч *Lc* на сградите, които се покриват от кулата. Как ще се променя стойността му, когато преместим кулата с една позиция към края на редицата от сгради и тя застане непосредствено зад сграда с номер *m*. Обработвайки сграда с номер *m*, първо вадим от стека последователно ония сгради, които са с по-малка височина от нея – те ще бъдат

покрити от тази сграда и ще станат недостъпни за сигналите от кулата. Ако между тях има сгради, чиято височина е по-малка от тази на кулата, те са били преброени в някакъв момент, при влизането си в стека, като такива които кулата ще покрие, ако бъде поставена непосредствено след тях (вижте следващия параграф). Така че, за всяка сграда с по-малка височина от кулата, която се вади от стека, понеже ще бъде покрита от сграда с номер *m,* трябва да намалим *Lc* с 1.

Последното нещо, което трябва да съобразим е, че ако сградата с номер *m* е по-ниска от кулата, то *Lc* ще се увеличи с 1 в момента, в който вкарваме височината и в стека, тъй като кулата ще я покрие (между нея и сграда с номер *m* няма други сгради).

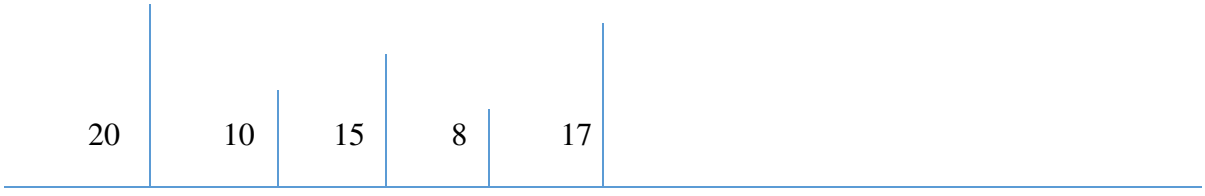
Тези съображения водят до алгоритъм, при който *Lc* се пресмята веднага след преместването на кулата с една позиция към края на редицата от сгради. Новоизчислената стойност на *Lc* трябва да се сравни с текущия намерен максимум на този брояч и, ако е по-голяма, да се смени текущия максимум. Когато кулата обходи всички възможни позиции, то текущият максимум ще дава отговора на задачата. Изменение в стойността на *Lc* може да настъпи, когато височината на сграда се вкарва или вади от стека. Тъй като височината на всяка сграда се вкарва в стека точно веднъж и се вади от стека най-много веднъж, то този алгоритъм има сложност *O(N)***.**

Сега да преминем към обяснение на решенията за повече от една кула**.**

Решения за *K* кули

Най-лесният начин да решим задачата за *K* кули е да я решаваме *K* пъти за всяка кула поотделно. В зависимост от това, кой от гореописаните алгоритми за решаване на задачата за една кула използваме, ще получим решения със сложност $O(K*N^2)$ и *O(K*N).* Първото ще покрие ония 20% от тестовете, в които $N \leq 1000$, $K \leq 20$ и ще донесе 20 точки. То е реализирано във файл **towers_kn2.cpp**. Второто ще покрие тези 20% и още 30% от тестовете, в които $N \leq 1\,000\,000$, $K \leq 20$ и ще донесе 50 точки. То е реализирано във файл **towers_kn.cpp.**

За да решим задачата по-ефективно, нека отново осмислим стека, който използвахме в решенията по-горе. Ще казваме, че една сграда се намира в стека, ако нейната височина е в стека. Във всеки момент сграда, която се намира в стека, е „затрупана“ от други сгради (може и нула на брой), които са с по-големи номера от нея (намират се надясно от нея в реда от сгради), имат по-малка височина от нея и са подредени във възходящ ред на височините им, започвайки от върха на стека и стигайки до нея. Нека измерваме „затрупаността“ на една сграда в стека, в даден момент от обработката на редицата от сгради, с броя сгради, които я „затрупват“ в този момент +1 (за нея самата). Тази величина непрекъснато се мени. Да поясним това с пример: нека имаме сгради с височини 20, 10, 15, 8 и 17.



Да проследим как се изменя стекът и съответно „затрупаноста“ на всяка сграда в него.

Състояния на стек

1	2	3	4	5	6
празен	20	10 20	15 20	8 15 20	17 20

„Затрупаност“ на сградите

Състояние на стек / Сграда с височина	1	2	3	4	5	6
20	0	1	2	2	3	2
10	0	0	1	0	0	0
15	0	0	0	1	2	0
8	0	0	0	0	1	0
17	0	0	0	0	0	1

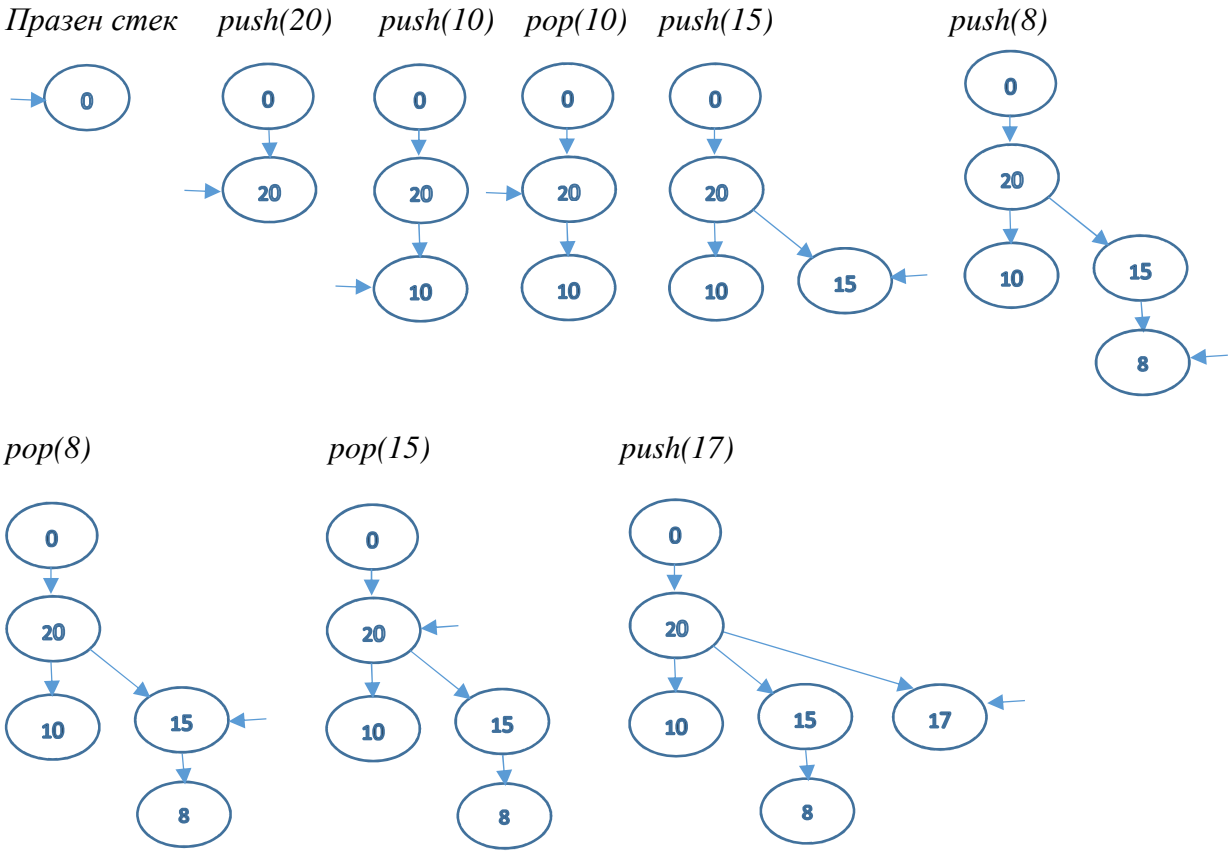
Нека сега имаме кула с височина *h*. Разглеждайки таблицата със “затрупаностите” на сградите, можем да формулираме следното твърдение: максималният брой сгради, които ще получават съобщения от кула с височина *h,* при оптимално нейно разположение, е равен на максималната „затрупаност“, която се среща за сграда, по-ниска от кулата, т.е. сграда с височина, по-малка от *h*.

В примера, който разглеждаме (сградите се идентифицират с височините им):

h на кула	Макс. брой сгради, получаващи съобщения	След коя сграда е кулата	Кои сгради получават съобщения
21	3	8	8, 15, 20
16	2	8	8,15
19	2	8	8,15
11	1	10	10

И така, трябва ни структура, в която да пазим състоянията на стека през цялото време на обработка на редицата от сгради, бърз начин да изчисляваме максималните „затрупаности“ на сградите и бърз начин за всяка кула да определяме сграда, която е по-ниска от нея и има най-голяма максимална „затрупаност“.

Подходящ за целта е персистентен стек, който се реализира с дърво, описващо всички състояния на стека. В него операцията *push* се реализира с добавяне на листо на върха, който сочи указателят, а операцията *pop* – с преместване на указателя, за да сочи към родителя на върха, който *pop*-ваме. Дървото има фиктивен корен – ако указателят сочи към него, това означава, че в момента стекът е празен. Ето как се променя дървото, чрез което се реализира персистентния стек за горния пример:



Когато са обработени всички сгради, получаваме дърво, при помощта на което можем да определим максималната „затрупаност“ на всяка сграда – това е броят на върховете по пътя от върха на дървото, изобразяващ сградата, до най-отдалеченото от него листо (включително върха на сградата и листото). В горния пример за сграда с височина 20 този брой е 3, за сграда с височина 15 – 2, и т.н.
Това дърво се строи за време *O(N)*, тъй като всяка сграда влиза в стека точно един път и излиза от него най-много един път.
При такава структура максималната „затрупаност“ за всички сгради се изчислява с обхождане в дълбочина на дървото за време *O(N)*.

Сега остава за всяка кула да вземем най-голямата „затрупаност“ измежду максималните затрупаности на всички сгради, които са по-ниски от нея. За целта сортираме сградите и кулите в нарастващ ред по тяхната височина (сложност *O(N*logN) + O(K*logK)*).
Нека за всяка сграда в някакво поле, напр. *score*, сме записали намерената за нея максимална „затрупаност“. Ако за всяка кула търсим максималната стойност на *score* за сградите, по-ниски от нея, ще отидем на сложност *O(N*K)*, което е лошо. Това може да се избегне, като линейно, за всяка сграда, в *score* запишем най-голямата максимална „затрупаност“, която се е срещнала до нея в сортираните по височина сгради (включително и за самата нея). След това правим асинхронно линейно обхождане на двата сортирани по височина масива от кули и сгради и за всяка кула отговорът е в *score* на последната, по-ниска от нея сграда. Сложността на решението е *O(N*logN) + O(K*logK)*. Това решение носи 100 точки и е реализирано във файл **towers.cpp**.

Автори: Руско Шиков, Йордан Чапъров