Тагове	На пълното решение	На частичните решения
	Побитови операции Наблюдения	Пълно изчерпване Динамично оптимиране

Анализ

Решения за частични точки

За да се изкарат до 50 точки, може да се използват различни пълни изчерпвания и динамични, като може да погледнете файловете `bitstack_20p.cpp` и `bitstack_50p.cpp` за да видите моите имплементации.

Решения за пълния брой точки

Преди да започна със същинското решение, ще спомена два факта, а именно че:

Нека с $p(x)$ обозначим броя единици в побитово представяне на x. Тогава:

- $p(x|y) \geq \max(p(x), p(y))$
- $p(x\&y) \leq \min(p(x), p(y))$

Това е съществено за решението.

Логично е да мислим задачата отзад-напред, като първо фиксираме последната операция, после предпоследната и т.н. Нека $p(a_N) > p(K)$. Тогава ако последната операция е *побитово или*, то последното останало число в стека ще има повече битове единици от K. Следователно няма как да бъдат равни. Следователно трябва да използваме *побитово и* за последната операция. Нека $p(a_N) < p(K)$. Тогава ако последната операция е *побитово и*, то последното останало число в стека ще има по-малко битове единици от K. Следователно няма как да бъдат равни. Следователно трябва да използваме *побитово или* за последна операция. Остана случая където $p(a_N) = p(K)$. Може лесно да се види, че ако $a_N \neq K$, то няма решение. Тогава нека да разгледаме две възможности:

- Използваме *побитово или* за последна операция. Тогава се стремим последното число да няма единици на позиции, на които a_N има нули. Няма по-добър начин да се подсигурим за това от това всички останали операции да са *побитово и*.
- Използваме *побитово и* за последна операция. Тогава се стремим последното число да няма нули на позиции, на които и a_N има единици. Няма по-добър начин да се подсигурим за това от това всички останали операции да са *побитово или*.

Супер, разгледахме случаите за a_N. Как продължаваме задачата? Нека последната ни операция да е *побитово или*. Нас вече не ни интересуват позициите на които a_N има единици, защото сме сигурни, че накрая ще имат единици. На всички останали позиции ще стои бита който е стоял на предпоследното число, останало в стека. Аналогично е когато се използва *побитово и* за последна операция – не ни интересуват позициите на които a_N има нули. На всички останали позиции отново ще стои бита който е стоял на предпоследното число, останало в стека. Защо просто не изтрием позициите, за които сме сигурни какъв ще е резултата? Така задачата остава същата за останалите битове. Сладко, нали ☺.

Постигната сложност: $\mathrm{O}(N)$

Имплементация: `bitstack_100p.cpp`

П.П: Чисто имплементационно може да се подходи по много начини, като аз вместо да трия битовете си поддържам неизтритите битове в променливата `active`. Може би се чудите как така постигнатата сложност е $\mathrm{O}(N)$, при положение че трябва да намираме $p(x)$ на всяка стъпка, което няма как да се случи за по-добро от $O(\log_2 a_i)$. Всъщност, това не е така! Има много изродски вградени функции, които смятат $p(x)$ за $\mathrm{O}(1)$, а именно:

- `__builtin_popcount(x)` – когато x е `int` променлива.
- `__builtin_popcountl(x)` – когато x е `long` променлива.
- `__builtin_popcountll(x)` – когато x е `long long` променлива.

Приятно, нали ☺.

Автор: Борис Михов