Анализ

Под „хубави двойки“ разбирайте двойки от съседни различни числа.

Подзадача №1

Както винаги оставих подзадача с тестовите примери за обратна връзка от системата.

Подзадача №2

Разглеждаме всяка пермутация на редицата и намираме броя хубави двойки.

Постигната сложност по време и памет: $O(N! \times N)$ и $O(N)$

Имплементация: `flowers_17p.cpp`

Подзадача №3

Нека започнем решението с едно кратко доказателство.

Лема №1. Нека най-срещания вид цветя в редицата да се среща на cnt брой пъти. Тогава отговорът е $\leq \min(N-1, 2(N-cnt))$.

Всяко число може да участва в най-много две хубави двойки (съответно с двата си съседа). Нека най-често срещаната стойност в редицата е x. Броят числа различни от x са $N-cnt$ на брой, следователно броят двойки, в които участват числа различни x, са най-много $2(N-cnt)$. Забележете, че във всяка двойка участва поне едно число, различно от x. Следователно броят на двойките е $\leq 2(N-cnt)$. Тъй като има $N-1$ двойки съседни числа, отговорът винаги е $\leq \min(N-1, 2(N-cnt))$. С това лемата е доказана.

Ние винаги има начин да разположим нулите и единиците по такъв начин, че отговорът да е равен на $\min(N-1, 2(N-cnt))$. Нека без ограничение на общността нулите са поне колкото единиците. Тогава поставяме нулите на нечетните позиции и разглеждаме два варианта:

1) Нулите са точно колкото нечетните позиции. Тогава допълваме останалите позиции с единици и отговорът ни е $N-1$.
2) Нулите са повече от нечетните позиции. Тогава, ако N е четно, слагаме нула на последната позиция в редицата и допълваме останалите клетки по произволен начин. Така отговорът ни е $2(N-cnt)$, където cnt е броят единици.

Забележете, че не е възможно броят на нулите да е по-малък от броя на нечетните позиции. В противен случай единиците ще са по-често срещания елемент.

Постигната сложност по време и памет: $O(N)$ и $O(1)$

Имплементация: `flowers_18p.cpp`

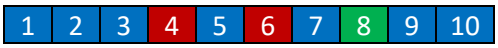
Подзадача №4

Нека продължим решението ни с още едно доказателство:

Лема №2. Нека най-срещания вид цветя в редицата да се среща на cnt брой пъти. Тогава отговорът е $\geq \min(N-1, 2(N-cnt))$.

Нека в редицата има k цветя, като най-често срещаното е със стойност x_1, като се среща cnt_1 пъти, второто е със стойност x_2 и се среща cnt_2 на брой пъти, …, k-тото е със стойност x_k и се среща cnt_k на брой пъти. Нека разгледаме два случая:

- $cnt_1 > \left\lceil \frac{N}{2} \right\rceil$. Това ще рече, че x_1 се среща повече на брой пъти от броя нечетни позиции в редицата. Тогава на всяка нечетна позиция поставяме x_1. След това, ако N е четно, на последната позиция отново поставяме x_1. След това разпределяме всички цветя произволно. По този начин получаваме решение с $2(N-cnt_1)$ на брой хубави двойки, следователно отговорът винаги е $\geq \min(N-1, 2(N-cnt))$. Едно примерно поставяне би било:



- $cnt_1 \leq \left\lceil \frac{N}{2} \right\rceil$. Това ще рече, че x_1 се среща най-много колкото е броя на нечетните позиции. Така първоначално разполагаме x_1 на първите нечетни позиции, след това като x_1 се изчерпа, продължаваме поставянето с x_2, след това продължаваме с x_3 и така докато не свършат нечетните позиции. Тогава нека сме достигнали i-тия вид цветя и сме поставили y цветя до последната нечетна позиция. Така ще трябва да доразположим $cnt_i - y$ цветя от вида x_i. Забележете, че тъй като първите cnt_1 нечетни позиции са с цветя от вида x_1, то първите $cnt_1 - 1$ четни позиции са заобградени от x_1. Заради това ако поставим цветята от вида x_i там, те ще образуват хубави двойки. Тъй като ние избрахме цветята, така че $cnt_1 \geq cnt_2 \geq \cdots \geq cnt_k$, ние ще трябва да разгледаме два случая:
  1. $cnt_i - y < cnt_1$. Тогава разполагаме остатъка от цветята от i-тия вид на първите четни позиции и произволно допълваме редицата. Така постигаме отговор $N-1$.
  2. $cnt_i - y = cnt_1$. Тъй като $cnt_i \leq cnt_1$, то $cnt_i = cnt_1$ и $y = 0$. Тъй като $y = 0$, цветята от вид $< i$ са успяли самостоятелно да изпълнят всичките нечетни позиции. Следователно където и да разположим останалите цветя, всичките им съседи ще са различни. Така отново постигаме отговор $N-1$.

С това намерихме конструкция, с която винаги постигаме отговор, който е равен на $\min(N-1, 2(N-cnt))$. Това означава, че отговорът винаги е $\geq \min(N-1, 2(N-cnt))$.

От **Лема №1** следва, че отговорът е $\leq \min(N-1, 2(N-cnt))$, а от **Лема №2** – че отговорът е $\geq \min(N-1, 2(N-cnt))$. Това означава, че отговорът винаги е $= \min(N-1, 2(N-cnt))$. Така остава само да се намери най-често срещаната стойност. Тъй като в тази подзадача има достатъчно памет да си запаметим редицата, ние може да я сортираме и да направим разделяне на последователности.

Постигната сложност по време и памет: $\mathrm{O}(N \log_2 N)$ и $\mathrm{O}(N)$

Имплементация: `flowers_65p.cpp`

Подзадача №5

Всъщност, какъв трябва да е най-често срещания елемент, така че отговорът да не бъде $N-1$? Той трябва да се среща повече от всички останали взети заедно, иначе казано да бъде *мажорант*. Така задачата ни се свежда до това да намерим алгоритъм, по който да проверим дали в редицата има мажорант и ако има, да видим колко пъти се среща. Когато аз бях Е група ☺ се беше дала задача major на НОИЗ D група, която е точно това. Хората, които са решавали задачи от минали години биха могли много по-лесно да направят текущата стъпка в решението. Нека разгледаме алгоритъма, който може да използваме:

- Поддържаме текущ кандидат за мажорант major и балансиращ параметър cnt. Първоначално $\mathrm{major} = -1$ и $\mathrm{cnt} = 0$.
- Обхождаме редицата отляво-надясно
  - Ако текущото число е равно на major, увеличаваме cnt с едно.
  - В противен случай:
    - Ако $\mathrm{cnt} > 0$, намаляваме cnt с едно.
    - В противен случай, като $\mathrm{cnt} = 0$, променяме кандидата за мажоранта на текущия елемент и фиксираме $\mathrm{cnt} = 1$.

Защо това работи? Нека, за да се демонстрират операциите, които този алгоритъм върши, да заменим всички елементи в редицата, равни на мажоранта, с $+1$, а всички останали с -1. Тогава:

- Истинският мажорант ще се възкачи като кандидат за мажорант, ако има положителен префикс на редицата. Тъй като има повече $+1$-ци от -1-ци, винаги съществува такъв префикс.
- Истинският мажорант ще се запази като кандидат за мажорант, ако суфикът, започващ от момента на възкачване, има неотрицателен сбор.

Нека моментът на възкачване е i, p е префиксната сума до елемент i и s е суфиксната сума, започваща от първия елемент, за който кандидат за мажорант ще бъде истинският мажорант, а именно $i+1$. Тъй като p е първият положителен префикс, $p = 1$. За да се запази истинският мажорант до края, трябва $s \geq 0$. Тъй като $s + p > 0$, то $s + 1 > 0 \Rightarrow s > -1 \Rightarrow s \geq 0$. С това вярността на алгоритъма е доказана.

Има още един алгоритъм, чрез който може да намерим мажорантата, който е по-мултифункционален от показания горе. Ние може за всяка позиция на числата в редицата (единици, десетици, стотици, …) да пазим колко пъти се среща всяка цифра от 0 до 9. За целта ще ни трябват 90 променливи. След това разглеждаме всяка позиция в числата (единиците, десетиците и т.н.) като намираме коя цифра (от 0 до 9) се среща най-често. Тази цифра ще е на мажорантата. Така още по-лесно може да изпълним тази част от задачата ни. Най-оптимално е да се реализира този алгоритъм

в двоична бройна система, като там ще ни трябват 30 променливи. Ще намерите две имплементации, съответно в десетична и двоична бройна система.

Добре, сега остана да разберем защо редицата ни е дадена два пъти. Проблемът на алгоритмите горе са, че те единствено те намират стойността мажорант, при наличие на такъв в дадена редица, но не намират колко пъти се среща в нея. Заради това, като сме намерили мажоранта, ние въвеждаме редицата повторно и преброяваме колко пъти се среща. Така също проверяваме дали намерената стойност от алгоритмите горе е мажорант. С това задачата ни е решена.

Постигната сложност по време и памет: $\mathrm{O}(N)$ и $\mathrm{O}(1)$

Имплементация на първия алгоритъм: `flowers_100p.cpp`

Имплементации на втория алгоритъм: `flowersDec_100p.cpp` и `flowersBin_100p.cpp`

Допълнение към авторовото решение

Задачата има начин да се реши като се въвежда редицата само веднъж. Представете си входния поток като `txt` файл. В него, за да знае `cin` откъде да въвежда, той си пази нещо като курсор за това къде се намира. Като прочитате променлива, вие местите курсора по-надясно в този `txt` файл. Има команда, с която може да преместите този курсор в началото на файла, като по този начин сами може да прочетете входа отначало. Командата е `cin.seekg(0)`, като `0` е „мястото“, където искате да се намира курсора. Когато бях осми клас, аз използвах тази команда по един интересен начин на втория кръг на олимпиадата, за да добутам една задача за `100`, тъй че може и на вас да ви е от полза ☺. Ако питате защо не съм дал задачата с едно въвеждане на редицата, считам, че знаенето на детайли от STL от типа на `cin.seekg(0)` няма да е добър показател за знанията на състезателите в шести клас, заради това счетох за най-добър този компромисен вариант между целта и сложността на задачата.

Постигната сложност по време и памет: $\mathrm{O}(N)$ и $\mathrm{O}(1)$

Имплементация на първия алгоритъм: `flowersSeekg_100p.cpp`

Автор: Борис Михов