Coloring

Тагове	На пълното решение	На подзадачите
	Наблюдения Паралелно двоично търсене	Малки и големи заявки

Анализ

Когато разглеждам конфигурация от оцветени клетки $p_1, p_2, \dots, p_k$ в анализа, аз винаги ще приемам, че Б.О.О. $p_i < p_{i+1}$ за $1 \leq i < k$. Очевидно няма значение в какъв ред гледаме клетките спрямо $\sum_{i=1}^{k} \sum_{j=1}^{k} |p_i - p_j|$.

Подзадача №1

Както винаги, оставих подзадача с тестовите примери, за да има по-добра обратна връзка от системата.

Подзадача №2

Може да се разгледа всяко подмножество на редицата с големина броя на оцветените клетки и да се намери $\sum_{i=1}^{k} \sum_{j=1}^{k} |p_i - p_j|$, както и минималната цена за да се достигне. Има стандартен начин да се сметне $\sum_{i=1}^{k} \sum_{j=1}^{k} |p_i - p_j|$ за $O(k)$ време, използва се например в B3. Points от НОИ-1 2022 година и C1. Pairs от Есенен СТИ 2020. За да довършим стандартното пълно изчерпване, ние трябва да направим и първото ни наблюдение.

Лема №1. По дадена начална конфигурация оцветени клетки $b_1, b_2, \dots, b_k$ и крайна конфигурация $e_1, e_2, \dots, e_k$, трансформирането с минимална цена от първата конфигурация към втората ще има струва $\sum_{i=1}^{k} |b_i - e_i|$ лева.

Ще означа трансформирането на една конфигурация към друга с $\rightarrow$. Очевидно две конфигурации са еквивалентни тогава и само тогава, когато $\sum_{i=1}^{k} |b_i - e_i| = 0$. Нека разгледаме как се променя $\sum_{i=1}^{k} |b_i - e_i|$ спрямо операциите, които извършваме:

1) Нека следващия ход в трансформацията е размяна на две оцветени клетки. Тогава $\sum_{i=1}^{k} |b_i - e_i|$ не се променя.
2) Нека следващия ход в трансформацията е размяна на оцветена и неоцветена клетка, където доближаваме клетка от b към съответната ѝ клетка от e. Тогава $\sum_{i=1}^{k} |b_i - e_i|$ намалява с 1.
3) Нека следващия ход в трансформацията е размяна на оцветена и неоцветена клетка, където отдалечаваме клетка от b към съответната ѝ клетка от e. Тогава $\sum_{i=1}^{k} |b_i - e_i|$ се увеличава с 1.

Тъй като една трансформация привършва, когато $\sum_{i=1}^{k} |b_i - e_i| = 0$, а с една операция може да намалим $\sum_{i=1}^{k} |b_i - e_i|$ с най-много 1, то цена на трансформация $\geq \sum_{i=1}^{k} |b_i - e_i|$.

Остана да докажем, че винаги съществува трансформация за цена точно равна на $\sum_{i=1}^{k} |b_i - e_i|$. Когато $k = 1$ е очевидно, че е оптимално директно да преместим b_1 към e_1 за $|b_1 - e_1|$лева.

Нека допуснем, че за някое $k \geq 2$, всяка конфигурация, с размер $len < k$, винаги има трансформация с цена $\sum_{i=1}^{len}|b_i - e_i|$. Нека разгледаме два случая за конфигурации с размер точно равен на k.

1) $b_k \leq e_k$. Така може да преместим b_k към e_k за $|b_k - e_k|$ лева и да направим $b_1, b_2, \dots, b_{k-1} \to e_1, e_2, \dots, e_{k-1}$. Тя няма да се пресича с $b_k \to e_k$, защото $\max(b_{k-1}, e_{k-1}) < e_k$. Така крайната цена е $\sum_{i=1}^{k-1}|b_i - e_i| + |b_k - e_k| = \sum_{i=1}^{k}|b_i - e_i|$.

2) $b_k > e_k$. Тогава може първо да приложим $b_1, b_2, \dots, b_{k-1} \to e_1, e_2, \dots, e_{k-1}$ и след това да приложим $b_k \to e_k$. Тези два процеса няма да се пресичат, защото $\min(b_k, e_k) > e_{k-1}$. Така крайната цена отново ще е $\sum_{i=1}^{k-1}|b_i - e_i| + |b_k - e_k| = \sum_{i=1}^{k}|b_i - e_i|$.

Следователно по индукция твърдението важи и намерихме констукция, по която трансформираме $b_1, b_2, \dots, b_k \to e_1, e_2, \dots, e_k$ за цена $\sum_{i=1}^{k}|b_i - e_i|$ и доказахме, че няма по-оптимална трансформация. С това доказахме лемата.

И това ни носи целите 15 точки. Юху!

Постигната сложност: $O(2^N \times N)$.

Имплементация: `coloring_15p.cpp`

Подзадача №3

Време е да направим и второто ни наблюдение.

Лема №2. По дадена конфигурация от оцветени клетки $p_1, p_2, \dots, p_k$, сборът $\sum_{i=1}^{k}\sum_{j=1}^{k}|p_i - p_j|$ приема минимална стойност тогава и само тогава, когато оцветените клетки са съседни. Нека допуснем противното – сборът може да е минимален, когато не са съседни. Представете си следната картинка:

Щом клетките не са съседни, то ще има поне две отделени последователности от оцветени клетки. Нека вземем най-лявата такава последователност (в случая 23) и я преместим до първата вдясно (в случая 5678), тоест в случая приложим размени $(3,4)$ и $(2,3)$. Така сборът $\sum_{i=1}^{k}\sum_{j=1}^{k}|p_i - p_j|$ очевидно ще намалее и **достигаме противоречие**. Следователно всяка конфигурация от несъседни клетки е неотпимална.

Остана да се докаже, че всяка конфигурация от съседни клетки е оптимална. Това ще рече, че всяка двойка конфигурации от съседни клетки имат равен сбор $\sum_{i=1}^{k}\sum_{j=1}^{k}|p_i - p_j|$. Нека допуснем, че две конфигурации $u_1, u_2, \dots, u_k$ и $v_1, v_2, \dots, v_k$, съответно започващи от $x+1$ и $y+1$, имат различен сбор. Тогава $u_i = x + i$ и $v_i = y + i$. Така:

$\sum_{i=1}^{k}\sum_{j=1}^{k}|u_i - u_j| = \sum_{i=1}^{k}\sum_{j=1}^{k}|x + i - (x + j)| = \sum_{i=1}^{k}\sum_{j=1}^{k}|i - j| = \sum_{i=1}^{k}\sum_{j=1}^{k}|i - j + y - y| = \sum_{i=1}^{k}\sum_{j=1}^{k}|y + i - (y + j)| = \sum_{i=1}^{k}\sum_{j=1}^{k}|v_i - v_j|$.

Следователно **достигаме противоречие**. С това лемата е доказана.

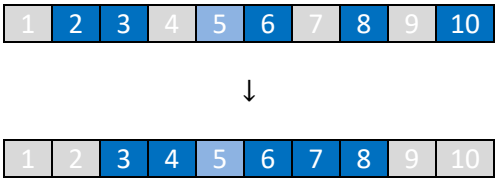
Така може за всяка заявка да разгледаме всяка възможна начална клетка и да пресметнем $\sum_{i=1}^{k}|b_i - e_i|$.

Постигната сложност: $O(QN^2)$.

Имплементация: `coloring_28p.cpp`

Подзадача №4

Тук има два варианта за оптимизация – да се оптимизира самото смятане на $\sum_{i=1}^{k}|b_i - e_i|$, което ще ни трябва в последствие, или да се използва **Лема №3**. Всъщност, вместо да избираме начална клетка, ние може да изберем *средна*, тоест такава, към която всички да се долепят. Може да разгледате примера отдолу за повече яснота:



В случая 5 е *средната* клетка. Може за упражнение да докажете, че конфигурацията, описана от всяка начална клетка, може да се опише и от средна такава и наобратно.

Лема №3. Винаги е оптимално да изберем *медианата* от позициите на оцветените клетки за *средна* клетка на крайната трансформация. Когато оцветените клетки са четен брой, няма значение коя медиана избираме.

1) Броят на оцветените клетки е четен, тоест $k = 2q + 1$. Нека означим средната клетка със s. Тогава цената е $\sum_{i=1}^{q}((s-1) - p_i - (q-i)) + \sum_{i=1}^{q}(p_{q+i} - (s+1) - (i-1)) = \sum_{i=1}^{q}\big(s - 1 - p_i - (q-i)\big) + \Big(p_{q+1} - (s+1) - (i-1)\Big) = \sum_{i=1}^{q} p_{q+i} - p_i - q - 1.$
Нека допуснем, че е по-оптимално средната клетка да е с x позиции наляво от s. Тогава всяка клетка ще я преместим с x позиции наляво от оригиналната трансформация. Така всяка клетка, преди старата средна клетка, ще бъде преместена с най-много x позиции по-малко, а всяка след старата средна клетка – с x позиции повече. Тъй като броят оцветени клетки преди и след старата средна клетка са равни, то в новия вариант, те няма да са по-малко, следователно **достигаме до противоречие**. Случая, когато новата средна клетка е вдясно на медианата е аналогичен, защото важат същите аргументи. Следователно винаги е оптимално да изберем медианата за нова средна клетка.
2) Броят на оцветените клетки е четен, тоест $k = 2q$. Нека фиксираме някоя произволна клетка между двете медиани за средна. Така цената за трансформацията ще е $\sum_{i=1}^{q}(s - p_i - (q-i)) + \sum_{i=1}^{q}(p_{q+i} - (s+1) - (i-1)) = \sum_{i=1}^{q}\big(s - p_i - (q-i)\big) + \Big(p_{q+1} - (s+1) - (i-1)\Big) = \sum_{i=1}^{q} p_{q+i} - p_i - q.$
Нека разгледаме случая, в който избираме за средна клетка някоя, която е преди първата медиана. Тогава отново ще местим клетките отляво с най-много x позиции по-малко, а тези вдясно – с x позиции повече.

   Аналогичен е случая, когато изберем клетка след втората медиана. Следователно е оптимално да изберем някоя от медианите за средна клетка.

Като изберем средната клетка остава само да се сметне отговорът за $O(N)$.

Постигната сложност: $O(QN)$.

Имплементация: `coloring_41p.cpp`

Метод за изчисление на отговора

Позовавайки се на разписването отгоре, ние достигнахме до извода, че ако $p_1, p_2, \dots, p_k$ са оцветените клетки и $q = \left\lfloor \frac{k}{2} \right\rfloor$, то отговорът е $\sum_{i=1}^{q} p_{q+i} - p_i - q = \sum_{i=1}^{q} (p_{q+i} - p_i) - q^2$ или $\sum_{i=1}^{q} p_{q+i} - p_i - q - 1 = \sum_{i=1}^{q} (p_{q+i} - p_i) - q(q+1)$ спрямо четността на k. Нека заявката ни е за x и y и да сме открили магически, че s е оптимална средна клетка. Нека за всяка стойност пазим клетките, които я съдържат, както и префиксни суми по тези клетки. Тогава, при нужда, може с две двоични да намерим кои клетки със стойност x и y спадат към вида p_i и съответно кои спадат към вида p_{q+i} от горния сбор. Така може относително лесно и бързо да смятаме отговорът по намерено s. Решенията, описани отдолу, се занимават точно с това.

Решение, използващо Малки и големи заявки

Тъй като решенията с малки и големи заявки в най-добрия случай имат сложност от порядъка на $O((N+Q)\sqrt{N})$, то това решение ще може да достигне до само петата подзадача.

Подзадача №5

Нека дефинираме *малка* стойност и *голяма* стойност. Малките стойности ще са тези, които се срещат до $\sqrt{N}$ пъти, а големите – всички останали. Нека стойност i се среща cnt_i пъти в редицата. Тъй като $cnt_1 + cnt_2 + \cdots + cnt_N = N$ и $cnt_{big} > \sqrt{N}$, ако big е голяма стойност, то ще има най-много $\approx \sqrt{N}$ големи стойности. Броят заявки, в които ще участва поне една голяма стойност са най-много $N\sqrt{N}$, заради това, ако намерим начин да ги преизчислим, ние ще трябва онлайн да отговаряме само на заявките с две малки стойности. Тъй като в най-лошия случай броя оцветени клетки при заявка от две малки стойности е $\approx 2\sqrt{N}$, то ще може да я отговорим по наивен начин. В моето решение използвам нещо, подобно на сливането при Merge sort. Преизчислението е по-интересната част.

Тъй като има най-много $\sqrt{N}$ големи стойности, то $O(N)$ начин за презичисление на всички заявки с нея ще ни свърши перфектна работа. Представете си следната картинка:

В случая, нея сините клетки са голямата стойност, за която изчисляваме отговорите на заявките, а останалите цветове са други стойности. Ние ще обхождаме редицата отляво-надясно и ще поддържаме колко често се среща всяка една стойност до момента. Една възможност е на всяка стъпка да проверяваме дали някоя от заявките е достигнала своята медиана, но това ще е твърде

бавно. Друга опция е да проверяваме за всяка заявка, към която спада текущата стойност, дали е достигнала медианата си, но това отново клони към $O(N^2)$. Нека вместо това за всяка стойност, различна от синята, да намерим първия момент, в който сме стъпили в нея и сме подминали медианата на заявката с нея. Например, при заявка за червената и синята стойност, медианата е клетка №6. Първия момент, в който стъпим в червена клетка и ще сме подминали медианата ще е като стъпим в клетка №9. Нека сме подминали медианата с k клетки. Ако $k = 0$, то очевидно текущата клетка е медианата. В противен случай сме сигурни, че медианата ще е $k - 1$-вата предишна синя клетка. Единствено трябва да се внимава със случаи като заявката с розово, където няма да има розова клетка, за която да сме подминали медианата. Тези заявки може да се отговорят по аналогичен начин след като сме обходили цялата редица.

Постигната сложност: $O((N + Q)\sqrt{N})$.

Имплементация: `coloring_70p.cpp`

Решения, използващи двоично търсене

Подзадача №5

Нека наблегнем на една по-простичка идея. Ние лесно може да отговаряме на заявки от вида „колко числа на позиция $\leq pos$ са със стойност x или y“. Пазим вектори за всяка стойност, съдържащи позициите, на които се срещат и правим двоично по векторите за x и y. Всъщност, какво по-точно е заявката за намиране на медиана? Ние желаем да намерим k-тата стойност отляво-надясно, равна на x или y, за $k = \left\lfloor \frac{cnt_x + cnt_y + 1}{2} \right\rfloor$. За целта може да правим двоично по отгвора, като търсим първата позиция pos, за която има поне k числа, равни на x или y, намиращи се преди нея.

Постигната сложност: $O(N + Q \log_2^2 N)$.

Имплементация: `coloringBin_70p.cpp`

Подзадача №6

В задачи, в които има много заявки за двоично търсене, които самостоятелно се отговарят бавно, може да се приложи подхода *Паралелно двоично търсене*. Една прекрасна статия за техниката може да намерите тук, но накратко е следното:

- Нека построим сегментно дърво върху редицата. Тогава всеки връх описва възможен интервал, който едно двоично търсене може да разгледа. Така, нека си представим показалка на двоичното търсене, която се намира първоначално в корена на сегментното дърво, съответно с интервал $[1, N]$. Показалката постепенно се спуска по дървото към отговора. Във всеки един момент, когато двоичното търсене знае, че отговора е в интервал $[l, r]$ и провери дали отговора е в интервала $[l, mid]$, то реално проверява, ако се намира във върха на сегментното с интервал $[l, r]$, дали отговора е в поддървото на лявото му дете.

- В това сегментно дърво във всеки връх си поддържаме заявките, за които знаем, че отговорът им се намира в някое листо в поддървото на върха. Първоначално всички заявки се намират в корена на дървото.
- Нека сме намерили колко пъти се среща всяка стойност в интервала $[1, mid]$, където mid е средата на интервала $[1, N]$. Нека i-тата стойност се среща $currCnt_i$ пъти. Разглеждаме всяка заявка и определяме дали отговорът е в интервала $[1, mid]$ или $[mid + 1, N]$. Това го постигаме, като за заявка (x, y) проверяваме дали $currCnt_x + currCnt_y \geq \frac{cnt_x + cnt_y + 1}{2}$. Така продължаме същия процес за заявките в интервалите $[1, mid]$ и $[mid + 1, N]$, след това – за интервалите $[1, mid_2], [mid_2 + 1, mid], [mid + 1, mid_3], [mid_3 + 1, N]$, където mid_2 и mid_3 са съответно средите при интервалите $[1, mid]$ и $[mid + 1, N]$ и така нататък.
- Когато достигнем листо в това сегментно дърво, всички заявки попаднали в него са с медиана тази клетка в редицата.
- Всяка заявка ще е минала най-много $\log_2 N$ върха докато стигне до листо.

Може да се питате как този подход ни подпомага за намирането на $currCnt_i$ по-бързо. Забележете, че височината на дървото е $\log_2 N$. Така, ако обходим редицата веднъж за всяко ниво на дървото, ние ще направим $O(N \log_2 N)$ стъпки. Нека разгледаме второто ниво на дървото, тоест това с върхове с интервали $[1, mid]$ и $[mid + 1, N]$. Нека средите на тези интервали са mid_2 и mid_3. Така при левия връх ще ни интересува $currCnt_i$ за позиции $\leq mid_2$, а за десния връх – за позиции $\leq mid_3$. Ние може първо да обработим заявките за левия връх, като сме изчислили $currCnt_i$ за позиции $\leq mid_2$, а след това да добавим и позиции $[mid_2 + 1, mid_3]$ към $currCnt_i$. Така може да обходим всяко ниво отляво-надясно и да поддържаме $currCnt_i$ за $O(N)$ на ниво. Обхождането на нивата в сегментното може да се имплементира много удобно с BFS, тъй като всички върхове на едно ниво имат равни разстояния до корена. С това задачата ни е решена.

Постигната сложност: $O((N + Q)\log_2 N)$

Имплементация: `coloring_100p.cpp`

Автор: Борис Михов

П.П: Лемите може да изглеждат тромави за доказване, но на състезания по информатика се изисква само интуицията, че са верни, не самите доказателства ☺. Лемите, ако приемат по общ вид, може да ги намерите често из задачи (например CK11. wojnici 2018, B2. postman от Летен турнир 2021, Niceset от InfO(1)cup 2022 и много, много други), тъй че състезателите с повече опит в задачи в областта не би трябвало да им бъде от голяма трудност да преминат през тях.