Анализ

Подзадача №1

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

Подзадача №2

Нека се пробваме да характеризираме по някакъв начин всеки възможен текущ момент до който сме достигнали. Съответно ние разполагаме с $(2N)!$ възможни пермутации, до които да сме достигнали. За достигането на всяка една от тях има два варианта - използвали сме операция от вид 2), или не сме. Тази информация напълно характеризира стейта на "случилото се". Така, ако ние построим граф, в който свържем с насочени ребра "съседните" стейтове (т.е. от първия да се достига втория с 1 операция), ние ще може да направим BFS по "всичко"-графа и да определим минималният брой ребра (т.е. операции) от началният стейт до някой краен. В графа има до $O(N^2 \cdot (2N!))$ ребра, защото от всеки връх излизат до $N^2 + 2N$.

Постигната сложност: $O(N^2 \cdot (2N!))$

Имплементация: -

Подзадача №3

За да направим решение на тази подзадача вече ни трябва първо наблюдение по задачата.

Лема 1

Минималният брой нужен операции за сортиране на пермутация, само чрез размени на съседни, е равен на броят на инверсиите в пермутацията.

Първо, размяната на две съседни числа променя бройката на инверсиите с най-много 1. Наистина, единствената двойка стойности, чиито позиции биват сменени релативно, са тези на размяната. Същевременно, сортираната пермутация има точно 0 инверсии. Така всяка сортировка на пермутацията ще изисква брой размени $\geq$ броят на инверсиите.

Същевременно, ако допуснем, че една пермутация има 0 инверсии, то тя е сортирана. Действително, ако предишното допускане важи и разгледаме неравенствата само между съседни елементи, то ако пермутацията е $q_1, q_2, \cdots, q_M$, то $q_1 < q_2; q_2 < q_3; \cdots; q_{M-1} < q_M$, т.е. $q_1 < q_2 < q_3 < \cdots < q_M$, следователно пермуитацията е сортирана. Тоест, ако измислим стратегия, с която с всяка размяна намаляваме с 1 броят на инверсиите, ние ще сортираме пермутацията за именно бройка размени $=$ броят на инверсиите, която по горната оценка е оптимална.

Една лесна такава стратегия е `Selection sort`. Той работи по следния начин – има $N - 1$ итерации, като в i-тата премества елементът със стойност i към i-тата позиция (винаги този елемент ще е на позиция $\geq i$). Представете си как алгоритъмът протича – изначално ще преместим минималният елемент 1 наляво към първата позиция, след което ще хванем двойката и т.н. Така, ние на i-тата итерация извършваме размени, които съответстват точно на инверсиите на i-тия елемент, защото всяка размяна е задължително с по-голям елемент от посочения от текущата стъпка. Така този алгоритъм е оптимален.

Така задачата се превръща в това да измислим стратегия за използване на операциите от първи вид, за да постигнем пермутация с минимален брой инверсии. Нека разгледаме двойките елементи $(p_1, p_2); (p_3, p_4); \cdots (p_{2N-1}, p_{2N})$. Операцията 1) няма способността да "отдели" елементите от двойка едно от друго, но може да подреди тези "блокчета" от двойки елементи в избран от състезателя ред. Така всъщност търсим пермутиране на тези блокчета, което постига минимален брой инверсии.

Тъй като броят на тези блокчета е N, в текущата подзадача е предвидено да се провери всяко тяхно пермутиране с наивно смятане на броят инверсии.

Постигната сложност: $O(N! \cdot N^2)$

Имплементация: `swaps_18p.cpp`

Подзадача №4

Продължаваме с пълното изчерпване от предната подзадача. Единствено, вместо да разгледаме всяка една възможна пермутация поотделно, ние може да забележим, че може да направим ДП с битмаска. За целите на динамичното ще конструираме оптималното подреждане на блокчетата като първо фиксираме това на първата позиция, после това на втората и т.н. Стейта на динамичното ще поддържа чрез битмаска кои двойки са разположени на позиция, по-лява от текущата, като самото ДП ще пази минималният брой инверсии за довършване на текущото разположение (за целите на пресмятане на инверсиите не ни интересува подредбата на двойките преди нас). Така фиксираме следващата двойка в подредбата и проверяваме инверсиите рекурсивно.

Постигната сложност: $O(2^N \cdot N^2)$

Имплементация: `swaps_28p.cpp`

Подзадачи №5**,** 6**,** 7**,** 8

Подзадачите бяха предвидени със рандъм тестове. Авторова грешка е, че не проверих крайният вид на условието в системата, поради което на състезателите не беше ясно, че тестовете на тези подзадачи са рандъм. Същевременно избрах да са рандъм, защото нямах умението да докажа сложността на решенията (а и в момента го нямам) и избрах да са рандъм, за да емпирично да открия до какви N работят долните подходи. Интуитивно решенията приличат на потокови алгоритми, а там всички по-опитни състезатели знаем, че оценката на Диниц за $O(N^2M)$ много често няма връзка с реалността, та не мисля, че си заслужава човек да си блъска главата с това.

Всъщност, решенията за тези подзадачи са определен вид `Hill climbing`. От начална пермутация ще търсим тази двойка блокчета, които при размяна, се намалява броят инверсии с максимален брой. В по-долните решения става ясно защо такъв подход работи. Разликата в точките на решенията се обосновава в различните възможни подходи за търсене на тази двойка. Предвидените подходи почват от възможно най-наивният (броящ инверсиите с $O(N^2)$ обхождане) за произволно фиксиране на двойка, даващ $O(N^4)$ сложност на размяна, до доста оптимизиран вариант – броене на промяната на бройката на инверсии чрез Дърво на Фенуик за $(log_2 N)$ на двойка. За повече детайли може да разгледате имплементациите, споменати по-долу.

- **Имплементация за Подзадача** 5**:** `swapsSlow0_38p.cpp`
- **Имплементация за Подзадача** 6**:** `swapsSlow1_47p.cpp`
- **Имплементация за Подзадача** 7**:** `swapsSlow2_54p.cpp`
- **Имплементация за Подзадача** 8**:** `swapsSlow3_65p.cpp`

Подзадачи №9, №10

Вече трябва да си отговорим на въпроса *Какво по дяволите се случва в задачата?*.

Един добър въпрос е защо `Hill climbing` подходите изобщо са верни. Те лесно биха се счупили, ако има локални минимуми, тоест пермутации, които нямат "съседни" (по дефиницията от **Подзадача №2**), които са с по-малко инверсии, но и те самите не са се свели до минималния възможен брой инверсии. Изглежда, че такива няма. Това би ни довело до мисълта, че абстрактно погледнато, би трябвало да има много добре дефинирани "правила" за релативните позиции на блокчетата в оптималната подредба. Иначе казано, ако разгледаме която и да е пермутация, която не е оптимална, разглеждайки тези правила, ще и намерим "косур" заради неспазване на някое от "правилата". Казано по-практично, логично е да допуснем, че съществува добре дефинирана оптимална подредба на двойките, която ако не се следва, може с 1 размяна на двойки да се достигне пермутация с по-малко инверсии.

Нека разгледаме едно блокче (p_x, p_{x+1}). Каквото и да правим с операциите от първия вид, винаги p_x ще е преди p_{x+1}. Същевременно, ако $p_x > p_{x+1}$, ние винаги ще ги разменим като досортираме пермутацията с втората операция. Заради това може изначално при $p_x > p_{x+1}$ мисловно да ги разменим и да си добавим 1 операция към цената от бъдещите 2).

Друга интерпретация на "добре дефинирана оптимална подредба" е да се опитаме да дефинираме оператор $prec$ за сравенение между блокчетата. Иначе казано, ако $(A, B) \prec (C, D)$, то ще искаме винаги (A, B) да се намира по-рано от (C, D) в пермутацията. От предишното твърдение може да разглеждаме двойките при $A < B, C < D$.

Нека допуснем, че сме успяли да дефинираме някак си този оператор $\prec$. За да бъде "добре дефиниран", какво е логично да изпълнява? Според мен следните две твърдения стигат:

1. Ако $(A, B) \prec (C, D)$, то $(C, D) \not\prec (A, B)$, тоест, ако (A, B) е винаги преди (C, D) според оператора, то (C, D) не е преди (A, B).
2. Ако $(A, B) \prec (C, D)$ и $(C, D) \prec (E, F)$, то задължително $(A, B) \prec (E, F)$.
3. При размяна на съседни (A, B) и (C, D), където (A, B) е вляво от (C, D) и $(C, D) \prec (A, B)$, то общият брой инверсии ненараства. В противен случай оператора за сравнение не е валиден в контекста на задачата.

Всъщност, ако първите две от горните изисквания са изпълнени, ние ще може да сортираме блокчетата спрямо този оператор. Така за всяка двойка блокчета, правилното ще се намира вляво (спрямо оператора).

Ще продължа да поддържам напрежението като не разкрия какъв е точно оператора $(A, B) \prec (C, D)$. Той може да бъде какъвто и да е, да изчислява броя гълъби тежащи $AB - CD$ килограма, да се пробва да пресметне busy beaver за A състояния и азбука от B символа, както и за C състояния и азбука от D символа и да им сравнява отговорите, няма значение, стига да изпълнява горните три условия. Оказва се, че ако наистина изпълнява изискванията, той ше ни доведе до оптимално решение.

Нека допуснем, че съществува несортирана спрямо оператора последователност от блокчета, която е стриктно по-оптимална от сортираната, тоест броят инверсии е по-малък. Нека блокчетата в тази последователност ги означим с $B_1, B_2, \cdots, B_N$. Нека дефинираме $\prec$-инверсия, която ще бъде двойка блокчета B_x, B_y, такива че $B_y \prec B_x$ и $x < y$. Забележете, че тъй като подредбата на блокчетата не е сортирана, то съществува поне една такава $\prec$-инверсия. Същевременно, за всяка подредба с $\prec$-инверсии > 0, съществува двойка елементи $B_{x+1} \prec B_x$ (аналогично на твърденията от **Лема** 1). Така ние ще може да разменим B_x и B_{x+1}, от което от твърденията за вида на операцията, общият брой инверсии

не се намалява. Така ще продължим да прилагаме такива размени на съседни в несортираната оптимална подредба и ще получаваме не по-лоша подредба. Така неизбежно ще стигнем до момент, в който $\prec$-инверсиите са $= 0$ (броят $\prec$-инверсии е краен), откъдето се оказва, че сортираната подредба е не по-лоша от уж по-оптималната несортирана – противоречие.

Така остана единствено да измислим добра операция (броене на гълъби е непрактично и труднодоказуемо, че изпълнява условията). Барабани... Оказва се, че произволна смислена операция ни върши работа – и $\max(A, B) < \max(C, D)$, както и $A + B < C + D$, вършат работа. Помня, че имаше и още дефиниции на $\prec$, които вършиха работа. Очевидно двата варианта за операции изпълняват изискванията $1.$ и $2.$, единствено трябва да се разгледа дали има вариант, в който общия брой инверсии се вдига с размяна на съседни двойки блокчета. Това ще го оставя за упражнение на читателя, като използвайте фактът, че вече $A < B$ и $C < D$, помага за смаляване на броят случаи, които трябват да се разгледат.

Забележете, че `Hill climbing` алгоритмите работят именно, защото всяка неоптимална подредба на блокчетата е несортирана (по горните твърдения), откъдето поради съществуването на двойка съседни блокчета, които при размяна ще намалят броят инверсии в пермутацията, винаги тези алгоритми ще успеят да намерят по-добра пермутация. Така те не могат да забият в неоптимална подредба. Забележете също, че ако променим `Hill climbing` алгоритмите да разменят само съседни, те бързо се превръщат в $O(N^2)$ алгоритми за сортиране.

За довършване, ние трябва да сортираме редицата и да преброим инверсиите. Най-наивният вариант е с $O(N^2)$ алгоритъм за сортиране (например `Selection sort`, `Bubble sort`) и $O(N^2)$ наивно намиране на броя на инверсиите. За 100 точки може да се използва `std::sort` за сортиране и `Merge sort` или `Fenwick tree` за изчисление на броя на инверсиите.

- **Имплементация за Подзадача** 9: `swaps_75p.cpp`
- **Имплементация за Подзадача** 10: `swaps_100p.cpp`

Автор: Борис Михов