Анализ

Под турнир се има предвид мелето.

Подзадача №0

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

Подзадача №1

Тази подзадача ни кара да се питаме първият логичен въпрос, а именно – какво става с елементът с най-голямо A? В случая той ще има и най-голямо B, откъдето всеки елемент който има различно A или B от него не може да го бие директно. Така може да видим, че всеки един елемент с A равно на максималното може да бие всеки друг в директен двубой, а всички останали не могат да спечелят турнира, защото нито един от тях не може да победи максималните елементи. Така отговора е равен на броят елементи с максимално A.

Постигната сложност: (N)
AI имплементация: `chatgpt_first_10p.cpp`

Конструктивна интерпретация

В този анализ ще разгледаме две решения, едното което работи изцяло на база на свойствата на турнира, което ще нарека *Конструктивна интерпретация* на задачата, и друго, което въвежда граф и работи главно с него, което ще бъде ”Графовата интерпретация”.

Подзадачи №4,6

Следващия въпрос, който следва е защо максимумите (съответно в A и в B) не печелят само те винаги, а отговорът се намира в тестовия пример – може някой *късметлия* да намери поредица от боеве с които побеждава по-силните от него, което разчита на фактът, че ще има *достатъчно* участници, които са слаби в единия показател и силни в другия.

Едно от най-естествените неща, които правим в задача в която се интересуваме под една или друга форма от сравнения между елементи е да ги сортираме. Тъй като тук не може да сортираме двойките спрямо и двата им критерия (т.е. $(A_1, B_1) \leq (A_2, B_2) \leq \cdots \leq (A_n, B_n)$), ние бихме могли да го направим само по единия. Заради това от тук насетне в анализа ще приемем, че сме сортирали двойките така че $A_1 \geq A_2 \geq \cdots \geq A_n$.

Наблюдение: ако участник номер i може да спечели турнира, то всички с номера $\leq i$ могат. Това е така, защото за всеки $j < i$ може променим начинът по който се водят боевете до победа на i-тия участник по такъв начин, че j-тия участник да спечели. Това ще го направим по следния начин – в сценария за победа на i-тия човек ще има единствен бой при който j-тия ще отпадне. Тогава ние няма да проведем този двубой, като ще проведем останалите битки от турнира както е по сценария. Тогава, след тяхното провеждане ще останат само i-тия и j-тия участник, където ще проведем битка между тях, от която j може да излезне победител, защото $A_j \geq A_i$.

Като следствие, ние можем вместо да разгледаме 2^N възможни множества от победители, ние ще трябва да разгледаме само N, защото като сортираме винаги отговорът ще бъде всички с номера $\leq M$, и губещи всички с $i > M$, за някое $1 \leq M \leq N$.

Следва натуралното продължение на нашите мисли – да опитаме да преценим за всяка една позиция е *разделителя* между печеливши и губещим, иначе казано да намерим горното M.

Една добра стратегия за да намерим M е да се питаме следния въпрос – *какви свойства има тази позиция?*. Едно от тях го открихме в първата подзадача (където M беше просто първата позиция след максималните елементи) – няма елемент след M, който да може да бие елемент преди M, иначе и той ще бъде победител. Така, ние сме сигурни, че ако M е разделителят, то няма елемент след него, който да бие елемент преди него.

Така вече имаме два вида позициии – една, която разделя печеливши от губещи, и други, такива, че за всяка една от тях K, всички елементи преди K не могат да бъдат победени в двубой от елемент след K. Вторият вид позиции ще ги наричам *шлюзове*, защото ако начертаете двойките на декартова координатна система ще забележите, че при такива позиции може да преместите O в такава точка, че всички в III квадрант са в позиция $> K$, а всички в квадрант sa в позиция $\leq K$. Генерално е добра идея да се пробваме да си представяме информацията геометрично.

Естествено е да пробваме да намерим разделителят по позициите на *шлюзовете*, защото това да проверим дали една позиция е *шлюз* изглежда по-лесно. Естествено, ако една позиция е *шлюз* не е задължително да бъде *разделител*, най-малкото N е *шлюз*. От там може да се питаме следното – възможно ли е позиция преди разделителя да бъде *шлюз*? Ако това е вярно, то ще има печеливш елемент, за който няма поредица от боеве, с които да успява да победи някой преди шлюза, откъдето няма да е печеливш – противоречие. Така може да видим, че разделителят е именно този *шлюз*, който се намира най-рано в позицията (няма нито един *шлюз* преди него, а той самият се пада *шлюз*). Така сведохме задачата да намерим най-малката позиция, която е *шлюз*.

За да намерим това, трябва да се върнем от *Страната на чудесата*, в която мислехме за абстрактни идеи, и да се върнем в постановката на задачата. Една позиция K е щлюз, ако за всяка двойка елементи $(i, j)|1 \leq i \leq K < j \leq N$ е изпълнено:

- $A_i > A_j$
- $B_i > B_j$

Тоест, j не може да бие i. Тъй като $A_i \geq A_{i+1} \geq \cdots \geq A_j$, ние на практика трябва да сме сигурни, че не може да изберем $A_i = A_j$. Това би било възможно тогава и само тогава, когато $A_K = A_{K+1}$. Така една позиция за да е *шлюз* задължително трябва $A_K > A_{K+1}$.

Другото условие е еквивалентно на $min(B_1, B_2, \cdots, B_K) > max(B_{K+1}, B_{K+2}, \cdots, B_N)$. За линейно време, може с две обхождания на намерим префиксните минимуми $prefMIN_X = min(B_1, B_2, \cdots, B_X) = min(prefMIN_{X-1}, B_X)$ и суфиксните максове $suffMAX_X = max(B_X, B_{X+1}, \cdots, B_N) = max(B_X, suffMAX_{X+1})$, като впоследствие проверим дали условието е вярно през $prefMIN_K > suffMAX_{K+1}$.

С това намираме първата позиция, която е *шлюз* и сме готови.

Постигната сложност: $(Nlog_2N)$
Имплементация: `author_100p.cpp`

Графова интерпретация

Нека построим насочен граф, в който има $O(N^2)$ ребра, като има ребро от u към v, ако u може да бие v.

Тогава, един връх може да е победител в турнира тогава и само тогава, когато от него има път към всички останали върхове.

Доказателството на това е по-тежко отколкото е самата идея. Това условие е необходимо, защото в противен случай ще има връх който не може да отпадне от турнира. Това условие е достатъчно, защото ако е изпълнено има начин, по който да организираме боевете, така че този връх да спечели. Първото твърдение може да се докаже като се разгледат битките отзад напред и се покаже, че условието е изпълнено по индукция. Второто – като битките се проведат в обратен DFS ред. Детайлите по доказателството са оставени за упражнение на читателя.

От тук следват няколко решения

Подзадача №3

Директна проверка с DFS за всеки отделен връх дали достига всички останали.

Постигната сложност: $O(N^3)$
AI имплементация: `chatgpt_dfs_20p.cpp`,

Подзадача №4

Когато се разглежда свързаност в насочен граф е много логично да се провери дали силно свързаните компоненти на графа са релевантни.

Нека намерим силно свързаните компоненти за $O(N+M) = O(N^2)$. Тогава, за да може един връх да стигне всички останали, то той задължително трябва да бъде в in връх (т.е. в такъв в който не влизат ребра), иначе самият връх няма да може да достигне върхове от in връх. Може да забележим, че това е достатъчно, защото в графа има само един in връх, защото между всяка двойка върхове има поне едно ребро (П.С. ако има ≥ 2 in върха, то ще може да построим ребро между два от тях, противоречие) (П.П.С. генерално за да може да достигнем до всички останали върхове, то трябва да има точно 1 in връх) (П.П.П.С. в този граф важи по-силното условие, че кондензираният DAG е пръчка). Така, намираме кой е върхът в кондензирания граф, към който не влизат ребра (а такъв има защото е DAG), с което сме готови.

Постигната сложност: $O(N^2)$
AI имплементация: `chatgpt_scc_explitic_50p.cpp`, `chatgpt_scc_implicit_50p.cpp`

Подзадача №6

Може да забележим, че горният алгоритъм е неефективен. Това е така, защото във всеки връх се влиза точно веднъж от едно DFS, с което ако успеем по добър начин да *пропускаме* ребрата към върховете, които вече сме посетили с DFS-а, ние бихме имали $O(N)$ алгоритъм.

Нека за момент се върнем към началната задача и видим към кои върхове има ребра връх, съответстващ на човек с коефициенти (a, b). То ребрата биха били към всички върхове, които (a, b) бие, т.е. за (c, d), $c \leq a$ или $d \leq b$.

Тогава, на практика, ние правим следното със всеки един от върховте – за текущ връх (x, y), гледаме кои са необходените върхове с $x \leq a$ и ги обхождаме, след това гледаме всички върхове с

$y \leq b$, и съответно и тях също ги обхождаме. Така може да видим следното свойство – всички върхове, които чакат да бъдат извикани за DFS от техен съсед са с $a \leq X$ или $b \leq Y$ за някакви X и Y.

Така, във всеки един момент, върховете, които бихме искали да обходим с DFS-а, щяха да са всички с $a \leq A$ и $b \leq B$. От тук може да ни хрумне идеята да правим следното – построяваме два сортирани масива на играчите в турнира, единия по координатата A, другия по координатата B, като поддържаме две показалки, една за масивът A, една за B, с които отчитаме докъде сме обходили върховете, като за да обходим всички с $a \leq A, b \leq B$, бутаме показалките докато не стигнем на върхове, които текущия не побеждава по съответния критерии.

Единствено трябва да сме внимателни с показалките, защото извикване на DFS на връх, към който имаме ребро може да им премести позицията в двата масива. *Така в крайна сметка ще имаме алгоритъм, който има подобна имплементация на Алгоритъма на Диниц за намиране на максимален поток.* Алтернативно, може да имплементираме алгоритъма за силно свързани компоненти чрез BFS.

Постигната сложност: $O(Nlog_2N)$

AI имплементация: `chatgpt_scc_dfs_100p.cpp`, `chatgpt_scc_bfs_100p.cpp`

За любопитните

Идеите в тази задача се разширяват и към K измерен вид на задачата (вместо 2) – конструктивната идея пак може да проверява дали някой вляво пада от някой вдясно, ако сме ги сортирали по която и да е координата, като отново ще пазим масив $minPref$ и $maxSuff$ за всяка координата, различна от тази по която сме ги сортирали, а в графовата ще пазим K показалки вместо 2.

Автор: Борис Михов