Omówienie zadań z turnieju indywidualnego nr 28

Rafał Mańczyk, Wojciech Rybak

8 lutego 2025

1 Intensywny Trening

W rozwiązaniu zadania wykorzystamy algorytm zachłanny, który dla każdej kolejnej pary wyników wybierze najmniejszą wartość większą od poprzedniego elementu ciągu. Jeśli obie wartości są większe od ostatnio wybranej, stwierdzamy, że rozwiązanie nie istnieje. Złożoność wynosi $O(N)$.

2 Pilates

Zauważmy, że jeśli program zaakceptuje c pompek oraz d przysiadów, to zgodzi się także na wykonanie $c - 30$ pompek i d przysiadów lub c pompek oraz $d - 30$ przysiadów, o ile te liczby pozostaną dodatnie. Zmiany takie nie zmieniają żadnej z reszt dzielenia przez 2, 3 i 5. Dzięki tej obserwacji możemy stwierdzić, że optymalne rozwiązanie będzie miało wartości c i d nie większe niż 30. Możemy brutalnie sprawdzić 900 możliwości dla każdego przypadku testowego.

3 Prezent

Kluczową obserwacją w rozwiązaniu zadania jest informacja, że dla dowolnej liczby naturalnej a $NWD(a, a + 1) = 1$. Możemy więc dla każdego elementu x w permutacji p, jeśli $x \neq n$, umieścić na jego miejscu $x+1$, natomiast n zamienić na 1. Otrzymamy w ten sposób wynik n, który jest najmniejszym możliwym.

4 Racer

Możemy zauważyć, że chociaż długość planszy wynosi 10^9, to nie ma dla nas znaczenia, czy między kolejnymi blokadami na drodze znajduje się 5, 150 czy 10^5 pól. W takim razie możemy traktować trasę tak, jakby znajdowały się na niej skróty od pierwszego pola za k-tą blokadą do pierwszego pola przed $k + 1$ blokadą, o ile blokady nie sąsiadują ze sobą. Jednym z rozwiązań jest przenumerowanie wartości, w którym wykorzystamy tylko pola nr 1, 10^9 oraz dla każdej blokady na polu x pola $x - 1$, x, $x + 1$. Na takiej trasie o długości co najwyżej $3n + 2$ możemy bez problemu wykorzystać algorytm BFS lub DFS.

5 Zgłoszenia

Rozwiążemy zadanie z użyciem programowania dynamicznego. Niech $DP(i, x, y)$ oznacza maksymalny wynik Janka z pierwszych i podzadań zakładając, że pierwsze zgłoszenie otrzymuje z tych pierwszych i podzadań x punktów a drugie y punktów. Wyliczając $DP(i, x, y)$, mamy 4 opcje, albo żadne ze zgłoszeń nie rozwiązało i-tego podzadania, albo dokładnie jedno z nich je rozwiązało otrzymując za nie a_i punktów, albo oba je rozwiązały jednak punkty cały czas otrzymujemy tylko raz. Otrzymujemy wzór $DP(i, x, y) = max(DP(i-1, x, y), DP(i-1, x-a_i, y) + a_i, DP(i-1, x, y-a_i) + a_i, DP(i-1, x-a_i, y-a_i) + a_i)$. Możemy policzyć wartości DP na początku programu w $O(N^3)$, potem wynikiem dla każdego zapytania jest $DP(n, x, y)$.

6 Gra

Zastosujemy wyszukiwanie binarne po wyniku, Będziemy chcieli dla ustalonego k sprawdzić, czy Janek może otrzymać wynik równy przynajmniej k. Zauważmy, że możemy wyróżnić kubki z $k \leqslant a_i$, Janek chce otrzymać ściśle więcej kulek niż Karol w którymkolwiek z tych kubków, Karol chce doprowadzić do tego, żeby tak się nie stało. Zauważmy, że jeśli w pierwszym ruchu Janek wrzuci swoją kulkę do któregoś z wyróżnionych kubków, a Karol w następnym ruchu nie wrzuci swojej kulki do tego samego kubka, to Janek będzie mógł utrzymać przewagę dwoich kulek w tym kubku do końca, wybierając za każdym razem przedział zawierający ten kubek jeśli taki jeszcze istnieje. Załóżmy, że startegią Janka jest wybieranie zawsze przedziału zawierającego jak najwięcej wyróżnionych kulek, wtedy po pierwszym ruchu Janka, Karol musi wybrać przedział który zawiera co najmniej te kubki które zawierał przedział Janka i ponieważ przedział Janka był takim który zawierał ich najwięcej, to musi być to przedział który zawiera dokładnie te same wyróżnione Kubki co przedział Janka. Zauważmy, że jeśli Janek używa tej strategii do póki są jeszcze jakieś przedziały, za każdym razem Karol musi odpowiedzieć przedziałem, zawierającym dokładnie te same wyróżnione kubi co Janek. Znaczy to, że Janek nie osiągnie swojego celu tylko jeśli można połączyć w pary przedziały zawierające dokładnie te same wyróżnione kubki. W takim przypadku Karol, niezależnie od strategii Janka może zawsze odpowiadać 2 przedziałem z pary, więc Janek na pewno nie osiągnie swojego celu. Odpowiedź na pytanie dla ustalonego k sprowadza się, więc do sprawdznia czy można połączyć wszystkie przedziały w pary tak, że w każdej parze przedziały zawierają dokładnie te same wyróżnione kubki. Można to zrobić dla każdego przedziału, znajdując pierwsze i ostatnie wystąpienie wyróżnionego kubka wewnątrz tego przedziału(jeśli żadne nie istnieje, ten przedział nie wpłynie na rozgrywkę) i skrócić go tak, żeby jego końce pokrywały się z tym pierwszym i ostatnim wystąpieniem. Mając tak skrócone przedziały, można je posortować i spróbować sparować. Takie sprawdzenie będzie działać w $O(nlogN)$, więc cały algorytm będzie miał złożoność $O(nlog^2N)$.

7 Finał

Znajdźmy najpierw wynik dla ustalonego korzenia. Niech $s(v)$ będzie rozmiarem poddrzewa wierzchołka v, $sum(v)$ sumą wag w poddrzewie v, a $DP(v)$ będzie maksymalną wartością, $\sum_i i \cdot a_{q_i}$, gdzie $q_1, q_2, \ldots, q_{s(V)}$ oznacza dowolną kolejność DFS w poddrzewie v. Wartość $DP(v)$, będzie zależała od kolejności wchodzenia do synów v,Niech u oznacza syna v. Z pierwszego z synów do którego wejdziemy weźmiemy po prostu wartość $DP(u_1)$, jednak potem w kolejnych ich DP reprezentuję $\sum_i i \cdot a_{q_i}$, jednak chcemy przesuąć współczynniki i, o tyle ile wierzchołków odwiedziliśmy przed wejściem do tego syna u_i, nazwijmy tą wartość S. Chcemy, więc dodać $\sum_i (i+S) \cdot a_{q_i} = \sum_i i \cdot a_{q_i} + S * sum(u_i) = \sum_i DP(u_i) + S * sum(u_i)$. $\sum_i DP(u_i)$ jest wartością stała dla v, więc skupimy się na maksymalizacji drugiego składnika. Pomyślmy ile para synów u_x, u_y do niego dodaje. Jeśli u_x będzie w kolejności przed u_y to ta para doda do wyniku $s(u_x) * sum(u_y)$, oraz $s(u_y) * sum(u_x)$ w przeciwnym wypadku. Zauważmy teraz, że chcemy mieć u_x w kolejności przed u_y, wtedy i tylko wtedy jeśli $\frac{s(u_x)}{sum(u_x)} \geqslant \frac{s(u_y)}{sum(u_y)}$. Możemy więc posortować synów malejąco po ich wartości $\frac{s(u_i)}{sum(u_i)}$ i w takiej kolejności i w takiej kolejności ich odwiedzać. Przy policznonym w taki sposób $DP(v)$, możemy odczytać wynik dla ustalonego korzenia biorąc jego wartość DP. Żeby rozwiązać całe zadanie, czyli znaleźć maksimum po wszystkich korzeniach, możemy wybrać dowolny, policzyć wartości DP i z użyciem przekorzeniania, znaleźć wyniki dla innych korzeni. Żeby szybko wykonywać przekorzenianie, przydatne może być policzenie sum prefiksowych $s(u_i)$, oraz sum sufiksowych $sum(u_i)$, dla aktualnego korzenia. Złożoność całego rozwiązania to $(NlogN)$.