Cukierki

Całkowita liczba zjedzonych w k dni cukierków wynosi

$$x + (x+1) + \cdots + (x+k-1) \;=\; kx + \frac{k(k-1)}{2}.$$

Warunek z zadania

$$kx + \frac{k(k-1)}{2} \;=\; N$$

można przekształcić do postaci

$$x \;=\; \frac{N - \frac{k(k-1)}{2}}{k}.$$

Aby rozwiązanie było poprawne, licznik po prawej stronie musi być *nieujemny* i *podzielny przez* k. Dlatego dla każdej długości ciągu k (gdzie $k(k-1)/2 \leqslant N$) wystarczy sprawdzić, czy

$$N - \frac{k(k-1)}{2} \equiv 0 \pmod{k}.$$

Jeśli tak, istnieje dokładnie jedna wartość x, czyli jedno poprawne ustalenie liczby cukierków na pierwszy dzień.

Złożoność Maksymalne k spełnia przybliżenie $k^2 \approx 2N$, więc pętla wykona się $O(\sqrt{N})$ razy, a cały algorytm działa w czasie $O(\sqrt{N})$.

Ranking Jasia

Podane w zadaniu relacje między uczestnikami tworzą skierowany graf $G = (V, E)$ o wierzchołkach $V = \{1, \ldots, N\}$ i krawędziach $E = \{a \to b\}$.

Wykrywanie sprzeczności Jeśli w grafie istnieje cykl skierowany, porównania są sprzeczne (bo w cyklu ktoś musiałby być jednocześnie lepszy i gorszy od samego siebie). Wystarczy więc sprawdzić, czy G jest acykliczny, np. algorytmem DFS.

Ile osób *musi* być za Jasiem? W oryginalnym grafie wykonujemy DFS/BFS z 1. Liczba odwiedzonych wierzchołków (bez 1) to *niżej* – zawodnicy pewni, że będą za Jasiem.

Ile osób *musi* być przed Jasiem? Odwróćmy wszystkie krawędzie i uruchommy DFS/BFS z wierzchołka 1. Liczba osiągalnych wierzchołków (bez 1) to *wyżej* – zawodnicy, którzy *muszą* być przed Jasiem.

Zakres możliwych miejsc

$$najwyższa\_pozycja = 1 + wyżej, \qquad najniższa\_pozycja = N - niżej.$$

Dowolna klasyfikacja spełniająca ograniczenia może wstawić pozostałych $N - wyżej - niżej - 1$ nieokreślonych zawodników zarówno przed, jak i za Jasiem.

Złożoność $O(N + M)$ (DFS).

Porządek musi być!

Mamy dynamiczny zestaw słów powstających w trakcie N zapytań („dopisz literę", „skopiuj słowo", „porównaj dwa słowa"). Potrzebujemy obsłużyć wszystkie operacje na bieżąco i na każde zapytanie typu 3 wypisać wynik porównania leksykograficznego.

Odpowiednią strukturą danych do tego zadania jest *drzewo trie*. Wówczas każdy prefiks wszystkich dotychczasowych słów będzie odpowiadał określonemu wierzchołkowi w drzewie.

Porównanie dwóch słów W celu porównania dwóch słów leksykograficznie, będziemy chcieli znaleźć *najdłuższy wspólny prefiks* dwóch słów. Tuż za nim znajduje się pierwsza litera rozróżniająca te słowa.

Niech u, v będą wierzchołkami reprezentującymi słowa w drzewie trie. Aby znaleźć ich najdłuższy wspólny prefiks, należy obliczyć $\mathrm{LCA}(u, v)$. Rozpatrzmy możliwe przypadki:

- $\mathrm{LCA}(u, v) = u = v$: słowa się nie różnią.
- $\mathrm{LCA}(u, v) = v$: jedno ze słów jest prefiksem drugiego.
- W pozostałych przypadkach: chcemy porównać literki na pozycjach za najdłuższym wspólnym prefiksem. W tym celu należy porównać odpowiednie dzieci wierzchołka $\mathrm{LCA}(u, v)$ (te, które znajdują się na ścieżkach do u oraz v).

Aby szybko obliczać *LCA* w dynamicznie tworzonym drzewie trie, można użyć standardowego algorytmu używającego skoków o potęgi dwójki i wyrównującego głębokości obu wierzchołków. Tablicę skoków dla wierzchołka obliczamy przy tworzeniu węzła. Całe drzewo można też zbudować zanim zacznie się odpowiadać na pytania (offline), wtedy można użyć dowolnego algorytmu obliczającego LCA.

Złożoność $O(N \log N)$ (LCA)

Bonus: Istnieje algorytm offline, który rozwiązuje ten problem w $O(N)$. Korzysta on z obserwacji, że aby porównać dwa słowa wystarczy porównać indeksy preorder odpowiadającym im wierzchołkom w drzewie trie.

Trening

Jasio ma do rozwiązania N zadań, a każdego dnia potrafi uporać się z co najwyżej K zadaniami. Szukamy minimalnej liczby dni D, takich że

$$D \cdot K \geqslant N.$$

Wzór Najmniejsza całkowita liczba spełniająca nierówność to

$$D = \left\lceil \frac{N}{K} \right\rceil = \frac{N+K-1}{K} \quad \text{(dzielenie całkowite)}.$$

Tuje

Oznaczmy początkowe wysokości tui przez $T_1, T_2, \ldots, T_n$. Garść nawozu może być rozsypana tylko na pozycjach $2 \ldots n-1$:

tuja	$i-1$	i	$i+1$
przyrost	$+1$	$+2$	$+1$

Niech $a_i \geqslant 0$ oznacza liczbę garści pod i-tą tuję, a X – docelową wysokość tui. Warunek wyrównania prowadzi do liniowego układu równań

$$\begin{aligned}
T_1 + a_2 &= X, \\
T_2 + 2a_2 + a_3 &= X, \\
T_i + a_{i-1} + 2a_i + a_{i+1} &= X, \quad i = 3, \ldots, n-2, \\
T_{n-1} + a_{n-2} + 2a_{n-1} &= X. \\
T_n + a_{n-1} &= X.
\end{aligned}$$

Wyraźmy kolejne zmienne a_i w postaci $b_i X + c_i$:

$$\begin{aligned}
a_2 &= X - T_1, \\
a_3 &= X - T_2 - 2 * (a_2) \\
a_{i+1} &= X - T_i - (a_{i-1}) - 2(a_i), \qquad i = 2, \ldots, n-2.
\end{aligned}$$

Podstawiając wartość pod a_{n-1} równaniu $T_n + a_{n-1} = X$, jednoznacznie wyznaczamy wartość X.

Jeśli X spełnia wszystkie równania, a zmienne a_i mają nieujemne wartości, to znaleźliśmy rozwiązanie. W przeciwnym wypadku otrzymujemy sprzeczność i należy wypisać "NIE".

Złożoność $O(n)$.

Wielka szachownica

Jeśli utożsamimy kolor czarny z liczbą 0, a kolor biały z liczbą 1, to kolor pola znajdującego się na pozycji (x, y) jest równy $(x+y)\%2$. Fakt ten można udowodnić następująco: Szukamy takiej funkcji, która punktowi $(1, 1)$ przypisuje 0, oraz każdym dwóm sąsiednim punktom przypisuje różne wartości. Nietrudno się przekonać, że warunki te określają szukaną funkcję jednoznacznie (tzn istnieje co najwyżej jedna taka funkcja). Jest tak, bo warunki te to po prostu reguły kolorowania szachownicy, przy założeniu, że pole $(1, 1)$ jest czarne, a takie kolorowanie jest tylko jedno. Formalnie można to pokazać przez indukcję (choć tak naprawdę każdy, kto wierzy, że szachownicę można pokolorować na tylko jeden sposób, z pewnością ma tę indukcję w głowie, nawet jeśli nie jest tego świadom). Skoro istnieje tylko jedna funkcja spełniająca te warunki, to wystarczy pokazać, że $f(x, y) = (x + y)\%2$ też je spełnia (więc musi być równa funkcji kolorującej szachownicę).

Ostatecznie zadanie sprowadza się do wczytania dwóch liczb i sprawdzenia wartości wyrażenia $(x + y)\%2$.

Łączony WF

W tym zadaniu należy znaleźć największy podzbiór, pewnego zbioru liczb naturalnych, o parzystej sumie elementów. Jeśli suma elementów podanego zbioru jest parzysta, to szukanym podzbiorem jest cały zbiór. W przeciwnym wypadku ze zbioru trzeba będzie wyrzucić przynajmniej jedną liczbę nieparzystą. Co więcej, wyrzucenie dokładnie jednej wystarcza, by suma pozostałych elementów była parzysta. Ponadto, w zbiorze takim musi istnieć przynajmniej jedna liczba nieparzysta. Zadanie sprowadza się więc do znalezienia najmniejszej, spośród podanych liczb nieparzystych, oraz wypisania różnicy sumy wszystkich elementów, oraz znalezionej liczby nieparzystej. Całość można zrobić w czasie i pamięci $\mathcal{O}(N)$

Trójkąty równoramienne

Zadanie to można rozwiązać naiwnie, sprawdzając wszystkie trzyelementowe podzbiory w czasie $\mathcal{O}\left(\binom{N}{3}\right) = \mathcal{O}(N^3)$. Rozwiązanie takie dostanie werdykt TLE i 0 punktów, autorzy przypominają jednak, że na zawodach typu olimpiada informatyczna warto pisać bruty.

Oczekiwane rozwiązanie ma złożoność $\mathcal{O}(n^2 \log n)$ i opiera się na następującej obserwacji: Niech p będzie punktem. Wówczas wszystkie punkty, które mogą utworzyć trójkąt równoramienny z p o długości ramienia równej d, w którym p jest wierzchołkiem przy obu ramionach, leżą na okręgu o środku w punkcie p i promieniu równym d. Policzmy więc trójkąty równoramienne z wyróżnionym wierzchołkiem przy obu ramionach:

- Procedurę rozważamy dla każego punktu p
- Najpierw sortujemy punkty po rosnącej odległości od p
- W ustalonym porządku wszystkie punkty leżące na jakimś okręgu zajmują spójny segment w tablicy. Można więc jednym przejściem pętli podzielić punkty według okręgów, na których leżą
- Dla każdego okręgu, jeśli leży na nim k punktów, to do wyniku dodajemy $\binom{k}{2}$

Złożoność tej procedury dla jednego punktu jest zdominowana przez sortowanie i wynosi $\mathcal{O}(N \log N)$. Ponieważ procedurę powtarzamy dla każdego punktu, to sumarycznie kosztuje nas ona $\mathcal{O}(N^2 \log N)$.

Mamy więc policzone trójkąty z wyróżnionym jednym wierzchołkiem. Jak stąd otrzymać wynik? Zauważamy, że w trójkącie, który nie jest równoboczny, tylko jeden wierzchołek może leżeć przy obu ramionach (bo tylko dwa boki tego trójkąta mogą być ramionami). Każdy taki trójkąt został więc policzony dokładnie raz. Sprawa ma się inaczej w przypadku trójkątów równobocznych, każdy z nich policzyliśmy 3 razy. Problemem jest też fakt, że policzyliśmy również trójkąty zdegenerowane.

Jest jednak lepiej niż mogłoby się wydawać. Okazuje się bowiem, że nie istnieją trójkąty równoboczne o trzech wierzchołkach w punktach kratowych. Fakt ten jest konsekwencją twierdzenia Picka, [1] z którego wynika, że trójkąt taki musiałby mieć wymierne pole. Jednak, jak wiemy, pole trójkąta równobocznego o boku a jest równe $\frac{a^2}{4}\sqrt{3}$, a wartość a^2 dla trójkąta o wierzchołkach w punktach kratowych musi być całkowita. Dochodzimy więc do sprzeczności, bo iloczyn liczby wymiernej i niewymiernej jest niewymierny.

Ta część jest już prosta. Niech $\{a, b\}$ będą dwoma wierzchołkami. Chcemy policzyć ile jest zdegenerowanych trójkątów $\{a, b, c\}$, w których b jest wierzchołkiem leżącym przy obu ramionach. Taki trójkąt może być jednak tylko jeden. Jeśli v jest wektorem od b do a, tzn $b+v = a$, to jedyny punkt c mogący utworzyć z punktami $\{a, b\}$ równoramienny trójkąt zdegenerowany jest równy $b-v$. Wystarczy sprawdzić, czy punkt ten należy do zbioru punktów, co można zrobić np trzymając zbiór punktów na secie. Całą tą część można więc obliczyć również w złożoności $\mathcal{O}(N^2 \log N)$.

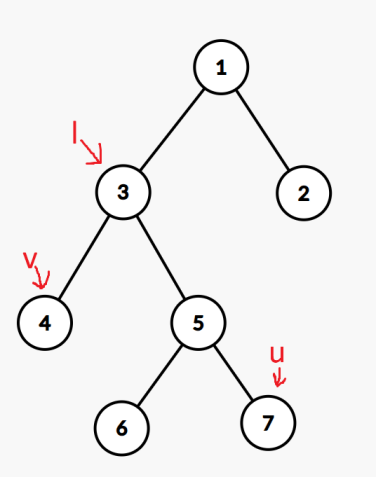
[1] `https://pl.wikipedia.org/wiki/Wz%C3%B3r_Picka`

Andrzej Hood

Symbol $a \oplus b$ będzie oznaczał operację xor zapisów binarnych liczb a i b.[2] Zdefiniuję *xor ścieżki* p, jako wartość xora wszystkich wag stojących przy krawędziach należących do p. W zadaniu dostajemy drzewo i jesteśmy proszeni o wypisanie liczby wszystkich ścieżek, których xor wynosi 0.

Załóżmy, że drzewo jest ukorzenione w wierzchołku o indeksie 1. Niech $XOR(v, u)$ oznacza xora ścieżki między wierzchołkami v i u. Ustalmy teraz konkretne wierzchołki v i u i niech l będzie ich najniższym wspólnym przodkiem w tym ukorzenieniu. Zachodzi związek:

$$XOR(1, v) \oplus XOR(1, u) = XOR(1, l) \oplus XOR(l, v) \oplus XOR(1, l) \oplus XOR(l, u) \tag{1}$$



Rysunek 1: Wyjaśnienie: Część wspólna ścieżek $1 \rightsquigarrow v$ i $1 \rightsquigarrow u$ jest ścieżką $1 \rightsquigarrow l$

Teraz, korzystając z przemienności xora, oraz z zależności $a \oplus a = 0$ i $a \oplus 0 = a$ dostajemy

$$XOR(1, v) \oplus XOR(1, u) = XOR(l, v) \oplus XOR(l, u) = XOR(v, u) \tag{2}$$

Jeśli

$$XOR(1, v) \oplus XOR(1, u) = 0 \tag{3}$$

To możemy nałożyć xor z liczbą $XOR(1, u)$ do obu stron równania, skąd dostaniemy równanie równoważne, jako że nakładanie xora przez dowolną liczbę jest funkcją odwracalną

$$XOR(1, v) = XOR(1, u) \tag{4}$$

Wystarczy więc zliczyć pary wierzchołków (v, u) spełniających równanie (4). Wartość $XOR(1, v)$ można policzyć algorytmem DFS dla każdego wierzchołka v w sumarycznym czasie $\mathcal{O}(N)$. Zebrane wartości należy posortować, podzielić na bloki, w których wartości te są takie same, a następnie dla każdego takiego bloku dodać do wyniku $\binom{d}{2}$, gdzie d jest rozmiarem bloku. Końcowa złożoność jest zdominowana przez sortowanie i wynosi $\mathcal{O}(N \log N)$.

[2] `https://pl.wikipedia.org/wiki/Alternatywa_roz%C5%82%C4%85czna`