Omówienie zadań z turnieju drużynowego nr 32

Marcel Szelwiga

26.04.2025

Misja: Wykopać bazę

W zadaniu dana była liczba D, należało odszukać takie liczby naturalne A i B, których iloczyn $A \cdot B$ był maksymalny oraz $2 \cdot (A+B) = D$. Zauważmy, że jeśli D jest nieparzyste lub N jest równe 2 to odpowiedź nie istnieje. W przeciwnym wypadku wiemy, że $A + B = \frac{D}{2}$. Intuicyjnie wydaje się, że iloczyn będzie maksymalny gdy A i B będą możliwie blisko siebie (a więc w pobliżu liczby $\frac{D}{4}$), spróbujmy sformalizować tą intuicję.

Wprowadźmy zmienną X, oznaczającą odległość liczby A od $\frac{D}{4}$ i zauważmy, że będzie ona również odległością B od $\frac{D}{4}$. Niech $X = A - \frac{D}{4}$, mamy zatem $X = \left(\frac{D}{2} - B\right) - \frac{D}{4} = \frac{D}{2} - B$. Po przekształceniu dostaniemy $A = \frac{D}{4} + X$ oraz $B = \frac{D}{4} - X$. Zauważmy, że iloczyn $A \cdot B = \left(\frac{D}{4} - X\right) \cdot \left(\frac{D}{4} + X\right) = \frac{D^2}{16} - X^2$ osiąga maksymalną wartość gdy X^2 jest najmniejsze możliwe, a zatem gdy liczby A i B są równe lub ich różnica jest równa 1, gdy $\frac{D}{2}$ jest liczbą nieparzystą. Odpowiedzią jest więc zatem para liczb: $\lfloor \frac{D}{4} \rfloor$ i $\lceil \frac{D}{4} \rceil$.

Misja: Zdobyć chrupki

W zadaniu dany był napis S, w którym należało obrócić wszystkie samogłoski. Możliwym rozwiązaniem byłoby utworzenie drugiego napisu X, który zawierałby wszystkie samogłoski z pierwszego napisu (w ich oryginalnej kolejności); taki napis można utworzyć iterując się po napisie S i dodając na koniec napisu X tylko samogłoski z napisu S. Następnie można ponownie przejść po kolejnych literkach napisu S i jeśli napotykamy na samogłoskę to wstawić w jej miejsce ostatnią literę napisu X i usunąć ją. Proponowane rozwiązanie działa w złożoności czasowej $\mathcal{O}(|S|)$.

Misja: Pomóc Lemurom

W zadaniu mieliśmy daną listę pozycji N niebieskich punktów na osi liczbowej. Należało zaznaczyć na niej minimalną liczbę żółtych punktów, w taki sposób, aby każdy niebieski punkt oddalony był o co najwyżej 5 od dowolnej żółtej kropki.

W tym zadaniu powinno nasuwać się zachłanne rozwiązanie polegające na umieszczeniu pierwszej żółtej kropki skrajnie na prawo, w taki sposób, aby pokrywała skrajnie lewą, niepokrytą jeszcze niebieską kropkę. A następnie powtarzanie tego procesu. Takie rozwiązanie można łatwo zaimplementować zwyczajnie iterując się po niebieskich kropkach i utrzymując pozycje ostatniej postawionej żółtej kropki oraz liczby postawionych już żółtych kropek. Proponowany algorytm działa w złożoności czasowej $\mathcal{O}(N)$.

Dowód poprawności skonstruowanego przez nasz algorytm rozwiązania opiera się na argumencie wymiany, który sprowadza dowolne optymalne rozwiązanie do naszego.

Rozważmy dowolne optymalne rozwiązanie, w którym pozycje żółtych kropek różnią się od tych wybranych przez nasz algorytm. Prześledźmy pierwszą różnicę między tym rozwiązaniem a naszym. Załóżmy, że nasz algorytm umieścił żółtą kropkę w punkcie y, a rozwiązanie optymalne w punkcie y', gdzie $y' < y$. Ponieważ nasz algorytm zawsze umieszcza żółtą kropkę możliwie najbardziej na prawo, tak aby nadal pokryć pierwszą niepokrytą niebieską kropkę, to oznacza to, że wszystkie niebieskie kropki pokryte przez y' są również pokryte przez y (korzystamy tutaj z założenia, że y jest pierwszym różniącym się elementem obu rozwiązań). Zatem możemy bez straty ogólności zastąpić y' przez y, nie pogarszając jakości rozwiązania. Powtarzając ten proces dla każdej różnicy między optymalnym rozwiązaniem a naszym, możemy przekształcić dowolne optymalne rozwiązanie w to skonstruowane przez nasz algorytm.

Misja: Dla niepoznaki

W zadaniu mieliśmy dany ciąg N zmian licznika, jego maksymalną wartość B oraz minimalną wartość A, należało wyznaczyć maksymalną początkową wartość licznika, dla której jego wartość po każdej zmianie mieściła się pomiędzy minimalną a maksymalną wartością. Dodatkowo licznik po osiągnięciu wartości negatywnej dopełniany jest do zera.

Spróbujmy na początku ustalić wynik $W = B + K$ (maksymalna możliwa wartość wyniku, niezależnie od zmian) oraz stan licznika na T, a następnie przetwarzać kolejne zmiany. Jeśli stan licznika T przekroczy maksymalną, to możemy zmniejszyć początkowy wynik W o $T - B$ czyli o ile stan licznika przekroczył B oraz ustawić stan licznika na B i udawać, że początkowy stan licznika zawsze taki był. Zauważmy jednak, że operacja ta nie zawsze doprowadzi do "prawdziwej" sytuacji. Być może potencjalne obniżenie początkowego stanu nie wpłynie na aktualną pozycję ze względu na możliwe wcześniejsze wystąpienie dopełnienie do zera. Zauważmy jednak, że nie zmienia to faktu, że nasze ograniczenie na wynik W dalej będzie poprawne.

Następnie na podstawie otrzymanego wyniku W możemy zasymulować wszystkie zmiany i sprawdzić czy wartość licznika zawsze będzie w odpowiednim przedziale, jeśli nie, to oznacza to, że wynik nie istnieje, w przeciwnym przypadku udało nam się wyznaczyć odpowiedź. Przedstawiony algorytm działa w czasie $\mathcal{O}(N)$.

To zadanie ma alternatywne (choć podobne w działaniu) rozwiązanie. Zauważmy, że dolne i górne ograniczenie wyniku wynikają odpowiednio z dolnego i górnego limitu licznika. Zauważmy, że gdyby nie istniało dolne ograniczenie, to wynik można byłoby obliczyć podobną co w poprzednim rozwiązaniu metodą lub wyszukiwaniem binarnym, analogicznie możemy obliczyć ograniczenie wyniku gdyby nie istniało górne ograniczenie wyniku. Na koniec wystarczyłoby się upewnić, że dolne ograniczenie jest mniejsze niż górne.

Misja: Nie dać się

W zadaniu dana była zagadka z zapałek znana z pudełek od zapałek czy łapaczy kliknięć na stronach internetowych. Polega ona na przełożeniu jednej zapałki w prostym wyrażeniu arytmetycznym, w taki sposób, aby wyrażenie z zagadki stało się poprawne. Zagadka ma postać "`DoD=D`" gdzie "`D`" oznacza cyfrę a "`o`" oznacza znak "`+`" lub "`-`".

Zadanie było czysto implementacyjne. Można zacząć od rozpisania dla każdej cyfry i znaku z jakich segmentów składają się (i ponumerować segmenty liczbami), wtedy każda pozycja będzie zbiorem zapalonych segmentów, a wyrażenie ciągiem pięciu zbiorów segmentów. Następnie można wygenerować listę wszystkich niekoniecznie poprawnych wyrażeń (ciągów zbiorów segmentów), powstałych w skutek usunięcia jednego segmentu, a następnie dla każdego z nich utworzyć kolejną listę wyrażeń dokładając jeden segment w każdą możliwą pozycję. Na koniec wystarczy sprawdzić czy którekolwiek z powstałych wyrażeń jest poprawne (składa się ze zbiorów reprezentujących cyfry oraz czy zachodzi równość).

Zadanie można zaimplementować naiwnie (wykorzystując listy zbiorów) lub nieco efektywniej wykorzystując maski bitowe do reprezentacji zbioru.

Misja: Przetrwać

W zadaniu dana była plansza o wymiarach $N \times M$, na której zaznaczone były niektóre pola oraz ciąg Q ruchów, góra, lewo, dół, prawo; w każdym ruchu wszystkie zaznaczone pola zaznaczają jedno z sąsiednich zgodnie z kierunkiem zadanym przez ten ruch; celem zadania jest wyznaczenie ostatnich zajętych pozycji oraz czasu, po którym cała plansza zostanie pokryta.

Rozwiązanie symuluje cały proces, oczywiście w nie najbardziej naiwny sposób. Przykładowo możemy utrzymywać dla każdego z kierunków zbiór pozycji, które zostaną pokryte, jeśli ruch zostanie wykonany w tym kierunku. Wykonanie ruchu będzie sprowadzało się do iteracji po pozycjach oraz zapaleniu niepokrytych jeszcze pól, z każdym zapaleniem pola wiązać się będzie dopisanie wszystkich sąsiadów do zbiorów pozycji dla konkretnych kierunków.

Zauważmy, że w powyższym rozwiązaniu każde pole może zostać zapalone tylko raz, a do każdego ze zbiorów każde pole może trafić co najwyżej raz. Algorytm ma zatem złożoność czasową $\mathcal{O}(N \cdot M + Q)$.

Misja: Drzewo

W zadaniu dane jest drzewo, należy podać minimalną średnicę drzewa jakie może powstać po usunięciu i dodaniu jednej krawędzi.

Do zadania można podejść klasycznie i rozważyć usunięcie każdej krawędzi. Zauważmy, że po usunięciu krawędzi od u do v, drzewo rozbije się na dwie części: T_u zawierające wierzchołek u i T_v zawierające wierzchołek v, oczywistym jest, że najbardziej opłacalne będzie połączenie wierzchołków ze środka średnic drzew T_u i T_v. Średnica nowo powstałego drzewa będzie wtedy odpowiadała maksimum poniższych wartości:

- $S(T_u)$ – średnica drzewa T_u,
- $S(T_v)$ – średnica drzewa T_v,
- $\left\lceil \frac{S(T_u)}{2} \right\rceil + \left\lceil \frac{S(T_v)}{2} \right\rceil + 1$.

Musimy w takim razie dla każdego wierzchołka znać średnicę w jego poddrzewie oraz średnicę w dopełnieniu jego poddrzewa. Średnicę w poddrzewie każdego wierzchołka możemy wyznaczyć standardowym programowaniem dynamicznym, które dla każdego wierzchołka utrzymuje długość najdłuższej ścieżki kończącej się w nim oraz rozmiar średnicy w jego poddrzewie. W celu obliczenia analogicznych wartości w dopełnieniu drzewa możemy uruchomić dfs-a ponownie i wyznaczać wyniki od korzenia korzystając przy tym z wartości obliczonych w poprzednim uruchomieniu dfs-a. Całe rozwiązanie ma złożoność czasową $\mathcal{O}(N)$ wynikającą z algorytmu dfs.

Misja: Ogradzanie

W zadaniu dana była lista współczynników B_i oraz zbiór N wartości A, wartości ze zbioru A należało ustawić w takiej kolejności aby ciąg współczynników B_i opisywał nierówności pomiędzy kolejnymi wartościami w ułożeniu zbioru A oraz $\sum_{i=1}^{N-1} |A_{i+1} - A_i|$ było maksymalne.

Zauważmy, że ciąg B_i wyznacza dla każdej pozycji w ułożeniu elementów ciągu A współczynnik z jakim będzie on stał przy sumie po zdjęciu wartości bezwzględnej. Elementy ułożenia większe od dwóch swoich sąsiadów będą miały współczynniki 2, elementy mniejsze od obu swoich sąsiadów będą miały współczynniki -2, a elementy mniejsze od jednego sąsiada i większe od drugiego będą miały współczynniki 0. Wyjątek stanowią elementy o jednym sąsiedzie, które będą miały współczynniki -1 lub $+1$.

Osiągnięcie maksymalnej sumy jest już teraz oczywiste. Wystarczy przypisać największe wartości w posortowaniu pozycjom o współczynniku 2, następną wartość w posortowaniu pozycji o współczynniku 1 (jeśli istnieje), następne wartości w posortowaniu pozycjom o współczynniku 0, i tak dalej. Rozwiązanie można zaimplementować z użyciem sortowania w złożoności czasowej $\mathcal{O}(N \log N)$ lub z użyciem algorytmów selekcji w czasie $\mathcal{O}(N)$.

Misja: Lot

W zadaniu dana była tablica N wysokości oraz Q zapytań dwóch typów, na które należy wyznaczyć odpowiedź:

- ile najwięcej zszybowań można wykonać z wierzchołka x,
- jaka jest najdłuższa trasa zszybowań od wierzchołka x do y;

Możemy skonstruować na elementach tablicy acykliczny graf skierowany w taki sposób, aby zawierał on wszystkie najdłuższe trasy, w tym wszystkie odpowiedzi na zapytania. Rozważmy do jakich elementów tablicy opłaca się poruszać z elementu na pozycji x. Zauważmy, że osiągalne są wszystkie elementy na przedziale od a do b, gdzie a to skrajnie prawa pozycja elementu o wartości większej równej wartości elementu x na lewo od x, a b to skrajnie lewa pozycja elementu o wartości większej równej wartości elementu x na prawo od x.

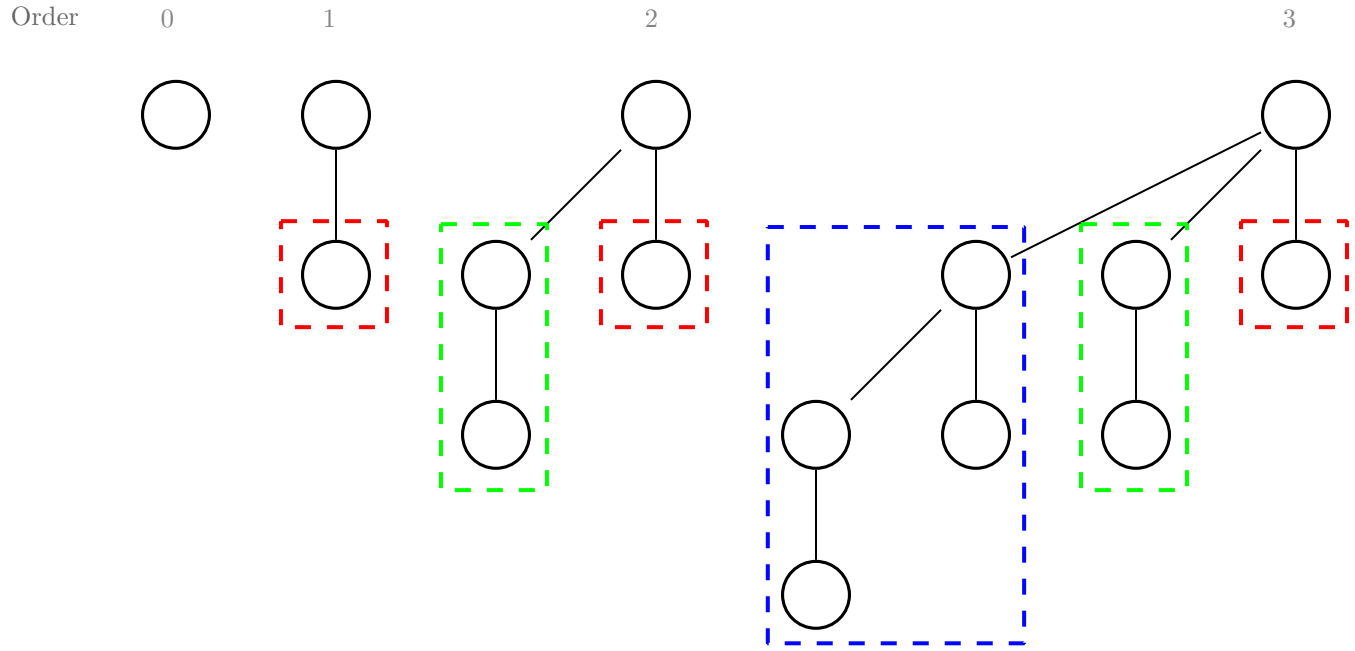
Na przedziale od $a+1$ do $x-1$, będzie opłacało się nam szybować tylko do elementów maksymalnych na tym przedziale. Analogicznie na przedziale od $x+1$ do $b-1$. Zgodnie z tą regułą możemy utworzyć krawędzie grafu, można to zrobić efektywnie z użyciem drzewa przedziałowego udostępniającego operacje znajdź następny element większy od określonej wartości na lewo i na prawo oraz maksimum na przedziale. Alternatywnie można użyć drzewa, które udostępnia samą operację znajdź maksimum na przedziale oraz rekurencji, która będzie przechodziła po kolejnych maksimach i rozbijała ciąg na części.

Zauważmy, że w skonstruowanym grafie możemy obliczyć głębokości (odległości od korzenia, maksymalnej wartości w tablicy) oraz najdłuższe ścieżki zaczynające się od danego wierzchołka.

Zauważmy, że odpowiedzią na pytanie ile zszybowań można wykonać z danego elementu tablicy jest długość najdłuższa ścieżki z odpowiadającego mu wierzchołka, a odpowiedzią na pytanie o najdłuższą trasę zszybowań pomiędzy dwoma elementami tablicy jest różnica głębokości ich wierzchołków. Rozwiązanie ma złożoność $\mathcal{O}(N \log N + Q \log N)$.

Misja: Potwór

W zadaniu mieliśmy skonstruować przykład, który udowadnia, że algorytm Union Find bez kompresji ścieżki rzeczywiście działa w czasie logarytmicznym.



https://upload.wikimedia.org/wikipedia/commons/c/cf/Binomial_Trees.svg

Przypomnijmy definicję drzewa dwumianowego: drzewo dwumianowe T_0 składa się z jednego wierzchołka; drzewo dwumianowe T_k rzędu składa się z korzenia oraz jego dzieci, którymi są drzewa dwumianowe rzędów mniejszych niż k.

Zauważmy, że po wykonaniu operacji `Union` na korzeniu drzewa i nowym wierzchołku, tak aby nowy wierzchołek został korzeniem, oraz wykonaniu operacji `Find` na najniższym wierzchołku drzewa dwumianowego otrzymamy drzewo dwumianowe tego samego stopnia z dodatkowym wierzchołkiem oraz wykonamy przy tym logarytmicznie, względem rozmiaru drzewa operacji.

Możemy zatem na początku skonstruować drzewo dwumianowe określonego rzędu, a następnie wykonywać wymienioną powyżej parę operacji, każda z nich będzie zmieniała wartość licznika o wysokość drzewa dwumianowego.