Omówienie zadań z Turnieju 39

Julia Kędziorska Adam Biel Mateusz Jurach
Karol Żeleźnik

21 listopada 2026

1 Podarek

W zadaniu mamy wypisać wzór, w którym w środku znajduje się cyfra N i oddalając się od środka cyfry maleją. Wzór dla $N = 2$ znajduje się poniżej.

```
    0
  0 1 0
0 1 2 1 0
  0 1 0
    0
```

W treści zadania mamy podane ograniczenie $2 \leqslant N \leqslant 9$. Oznacza to, że tak naprawdę mamy tylko 8 przypadków testowych.

Pierwszym i najprostszym sposobem rozwiazania zadania jest rozpatrzenie każdego z 8 przypadków: $N = 2, 3, \ldots, 9$. W kodzie możemy napisać 8 instrukcji `if`, dzięki którym sprawdzimy w jakim przypadku testowym się znaleźliśmy i wypisać wzór dla danego N. W każdym `if`-ie możemy po prostu wypisać wzór jedną lub kilkoma instrukcjami `cout`. Poniżej znajduje się fragment kodu, który odpowiadałby za wypisanie przypadku $N = 2$.

```
cout << "    0    " << endl;
cout << "  0 1 0  " << endl;
cout << "0 1 2 1 0" << endl;
cout << "  0 1 0  " << endl;
cout << "    0    ";
```

Pisanie 8 warunków może być jednak trochę czasochłonne i żmudne, więc możemy spróbować sprawniej wypisać wzór, bez rozpatrywania przypadków.

Na wyjście wypisujemy $2N + 1$ wierszy. Możemy wyobrazić sobie, że każdy wiersz składa się z $2N + 1$ pól, w których mogą znaleźć się liczby od 0 do N.

Pomiędzy sąsiednimi polami zawsze znajduje się pojedyncza spacja. Niektóre pola pozostaną puste, ale nadal musimy zachować odpowiednie odstępy. Pierwsze N i ostatnie N wierszy są takie same tylko w odwrotnej kolejności. Żeby wypisać pierwsze N wierszy możemy napisać pętlę, w której zmienna K będzie przyjmować kolejno wartości $0, 1, \ldots, N-1$. W każdym obrocie pętli wypisujemy jeden wiersz: pierwsze $N - K$ miejsc zajmą spacje (i pomiędzy nimi też będą spacje), potem wypisujemy pętlą cyfry od 0 do K, potem cyfry od $K - 1$ do 0 i na końcu $N - K$ pustych miejsc.

Po wypisaniu pierwszych N wierszy wypisujemy jeszcze środkowy wiersz, a następnie ponownie pierwsze N wierszy, tym razem w odwrotnej kolejności.

2 Wybieranie Składów

W zadaniu dostajemy trzy liczby A, B, i C oznaczające kolejno liczności składów Jasia i Małgosi oraz liczbę uczniów jeszcze nie wybranych do żadnego składu. Mamy określić jaka będzie róznica w liczności składów, po procesie wybierania opisanym w treści.

W tym zadaniu dobrą drogą do rozwiązania było przeanalizowanie kilku przypadków i nabranie intuicji jakie ruchy są optymalne dla obu graczy.

Można na przykład zauważyć, że jeżeli $A \geqslant B$, to w pierwszym ruchu Jasia opłaca mu się wziąć wszystkich C uczniów do swojego składu i zapewnić sobie dużą różnicę. W takim wypadku wynik wynosi więc $A + C - B$.

Jeżeli $A \leqslant B$ to dalej w niektórych przypadkach najlepszym wyborem Jasia będzie wzięcie wszystkich, na przykład jeżeli na wejściu dostajemy `3 5 7`. Są też jednak przypadki, że B jest o tyle większe od A, że najlepszym wyborem Jasia jest nie wziąć nikogo do swojego składu, na co Małgosia odpowiedziałaby również nie wybierając nikogo (nie chce ona powiększać różnicy). Jest tak na przykład, jeżeli na wejściu dostaniemy `3 5 3`. Widzimy więc, że któryś z tych dwóch skrajnych ruchów przeważnie działa dla Jasia. Czy może się okazać, że wzięcie innej liczby zawodników w pierwszej turze jest najlepszym wyborem? Nie, ponieważ jeżeli Jaś wybierze liczbę osób pomiędzy 0 i C, to Małgosia może odpowiadać biorąc 0 osób i wynik Jasia jest gorszy od wyniku, w przypadku gdyby wybrał któryś ze skrajnych ruchów.

Jaś ma więc do wyboru tylko dwa ruchy – wziąć 0 lub C osób. Jeżeli Jaś nie wybierze żadnego ucznia, Małgosia również nie będzie musiała nikogo wybierać, więc końcowa różnica wyniesie $B - A$. Jeżeli natomiast Jaś wybierze wszystkich uczniów, końcowa różnica wyniesie $A + C - B$. Możemy porównać te wartości i okazuje się, że wybranie 0 osób jest najlepszym wyborem, jeżeli $A+C-B < B-A$, czyli jeżeli $C < 2(B - A)$ i wtedy wynik to $B - A$. W przeciwnym wypadku najlepszym pierwszym ruchem jest wzięcie C osób i wynik to $A + C - B$.

Możemy rozpatrzyć omówione wyżej przypadki kilkoma warunkami `if`.

3 Odwracanie Znaków

Obserwacje wstępne

Rozważmy jeden zestaw testowy. Niech $A_1, A_2, \ldots, A_N$ będzie daną tablicą. W jednej operacji wybieramy indeks i i zmieniamy znaki elementów A_i oraz A_{i+1}. Naszym celem jest uzyskanie największej możliwej sumy elementów tablicy.

Zauważmy najpierw, że żadna wartość A_i nie może wnieść do wyniku więcej niż $|A_i|$. Jeśli oznaczymy

$$S = \sum_{i=1}^{N} |A_i|,$$

to S jest górnym ograniczeniem na odpowiedź. Pozostaje ustalić, kiedy można doprowadzić wszystkie elementy do wartości nieujemnych.

Kluczowa jest następująca własność operacji:

1. **Każda operacja zmienia znaki na dwóch pozycjach.** Zatem łączna liczba pozycji, których znak został zmieniony nieparzystą liczbę razy, jest parzysta.

2. **Możemy zmienić znaki na dowolnym zbiorze pozycji o parzystej liczności.** Aby zmienić znaki tylko na pozycjach p i q, gdzie $p < q$, wykonujemy operacje kolejno na parach

   $$(p, p+1), (p+1, p+2), \ldots, (q-1, q).$$

   Elementy na końcach tego przedziału zmienią znak raz, a każdy element wewnętrzny dwa razy, więc ostatecznie zmienią się wyłącznie znaki A_p i A_q. Dowolny parzysty zbiór pozycji możemy podzielić na pary i zastosować tę konstrukcję do każdej pary.

Niech K oznacza liczbę elementów ściśle ujemnych, a

$$M = \min_{1 \le i \le N} |A_i|.$$

Krok 1: Obliczenie potrzebnych wartości

Podczas jednego przejścia po tablicy obliczamy:

- sumę wartości bezwzględnych S,

- najmniejszą wartość bezwzględną M,
- liczbę elementów ujemnych K,
- informację, czy w tablicy występuje zero.

Wartości S i M przechowujemy w 64-bitowym typie całkowitym. Sama odpowiedź może być znacznie większa niż pojedynczy element tablicy.

Krok 2: Przypadek parzystej liczby elementów ujemnych

Jeśli K jest parzyste, zbiór pozycji zawierających elementy ujemne ma parzystą liczność. Z obserwacji wstępnej wynika, że możemy zmienić znaki dokładnie na tych pozycjach. Wszystkie elementy staną się wtedy nieujemne, więc otrzymana suma będzie równa S.

Ponieważ S jest górnym ograniczeniem na dowolny wynik, w tym przypadku odpowiedzią jest

$$S.$$

Krok 3: Przypadek nieparzystej liczby elementów ujemnych

Załóżmy teraz, że K jest nieparzyste.

Jeśli w tablicy występuje zero, do zbioru pozycji ujemnych możemy dołączyć pozycję tego zera. Otrzymamy wtedy zbiór o parzystej liczności, więc możemy zmienić znaki na wszystkich jego pozycjach. Wszystkie elementy ujemne staną się dodatnie, a zmiana znaku zera nie wpłynie na tablicę, ponieważ $-0 = 0$. Ponownie osiągamy sumę S.

Jeśli w tablicy nie ma zera, parzystość liczby elementów ujemnych nie może się zmienić. Każda operacja zmienia znaki dwóch niezerowych elementów, więc liczba elementów ujemnych zmienia się o -2, 0 albo 2. Skoro początkowo jest nieparzysta, po wykonaniu dowolnych operacji również musi pozostać nieparzysta. Co najmniej jeden element musi zatem pozostać ujemny.

Aby stracić jak najmniej względem górnego ograniczenia S, pozostawiamy ujemny element o najmniejszej wartości bezwzględnej M, a wszystkie pozostałe elementy doprowadzamy do wartości dodatnich. Jest to możliwe, ponieważ zmieniamy znaki na $K-1$ początkowo ujemnych pozycjach, a liczba $K-1$ jest parzysta. Zamiast składnika M otrzymujemy w sumie składnik $-M$, więc wynik zmniejsza się o $2M$. Odpowiedź wynosi wtedy

$$S - 2M.$$

Złożoność obliczeniowa

Dla pojedynczego zestawu testowego wykonujemy jedno przejście po tablicy, więc złożoność czasowa wynosi $\mathcal{O}(N)$, a złożoność pamięciowa $\mathcal{O}(1)$ poza pamięcią potrzebną na dane wejściowe. Ponieważ elementy można przetwarzać podczas wczytywania, nie trzeba przechowywać całej tablicy.

Dla wszystkich zestawów testowych łączna złożoność czasowa wynosi $\mathcal{O}(\sum N)$, gdzie suma długości tablic nie przekracza 200 000.

4 Kiedy Dostawa?

Obserwacje wstępne

Mamy dany napis S długości N, opisujący cykliczny harmonogram dostaw. Wiemy, że dzisiejsza dostawa zawiera koszulki w rozmiarze D, a Bajtek czeka na rozmiar C.

Nie znamy aktualnego dnia cyklu, dlatego musimy rozważyć wszystkie pozycje i, dla których $S_i = D$, i znaleźć największą możliwą liczbę dni oczekiwania na najbliższą dostawę rozmiaru C.

Zauważmy dwie istotne własności:

1. **Cykliczność harmonogramu:** Po ostatnim dniu harmonogram rozpoczyna się ponownie od pierwszego. Oznacza to, że jeśli za pozycją i nie występuje już żaden znak C, najbliższej dostawy tego rozmiaru należy szukać na początku kolejnego cyklu.

2. **Najbliższa dostawa:** Jeśli znamy pozycję j najbliższego wystąpienia znaku C na pozycji i lub za nią, to czas oczekiwania wynosi $j - i$. Przeglądając harmonogram od prawej do lewej, możemy na bieżąco zapamiętywać pozycję najbliższej dostawy rozmiaru C.

Dzięki tym obserwacjom możemy wyznaczyć odpowiedź podczas jednego przejścia po harmonogramie, odpowiednio uwzględniając jego cykliczność.

Krok 1: Uwzględnienie cykliczności

Będziemy przeglądać harmonogram od prawej do lewej, zapamiętując w zmiennej *next* pozycję najbliższej dostawy rozmiaru C.

Musimy jednak uwzględnić sytuację, w której za rozpatrywaną pozycją nie ma już żadnej dostawy rozmiaru C w bieżącym cyklu. Wówczas najbliższa taka dostawa nastąpi dopiero w kolejnym cyklu harmonogramu.

Dlatego najpierw znajdujemy pozycję p pierwszego wystąpienia znaku C w napisie S i ustawiamy

$$next = N + p.$$

W ten sposób, rozpoczynając przeglądanie napisu od prawej strony, mamy już zapamiętaną pozycję najbliższej dostawy rozmiaru C w kolejnym cyklu. W miarę przesuwania się w lewo będziemy aktualizować $next$, gdy napotkamy kolejne wystąpienia znaku C.

Krok 2: Wyznaczenie najdłuższego czasu oczekiwania

Przeglądamy napis S od prawej do lewej, czyli dla kolejnych pozycji i od N do 1.

Jeśli $S_i = C$, aktualizujemy $next = i$, ponieważ znaleźliśmy bliższą dostawę poszukiwanego rozmiaru.

Jeśli natomiast $S_i = D$, obliczamy czas oczekiwania jako $next - i$ i aktualizujemy maksymalną znalezioną wartość.

W ten sposób rozważymy wszystkie możliwe dni rozpoczęcia oczekiwania i dla każdego z nich wyznaczymy czas do najbliższej dostawy rozmiaru C.

Złożoność obliczeniowa

Najpierw jednokrotnie przeglądamy napis w celu znalezienia pierwszego wystąpienia znaku C. Następnie wykonujemy kolejne przejście, tym razem od prawej do lewej. Dla każdej pozycji wykonujemy stałą liczbę operacji, zatem całkowita złożoność czasowa wynosi $\mathcal{O}(N)$.

Poza napisem wejściowym przechowujemy jedynie kilka zmiennych pomocniczych, więc dodatkowa złożoność pamięciowa wynosi $\mathcal{O}(1)$, a uwzględniając przechowywanie napisu, $\mathcal{O}(N)$.

5 Początkujący Wspinacz

Obserwacje wstępne

Mamy dany ciąg N szczytów o wysokościach $A_1, A_2, \ldots, A_N$. Bajtek może rozpocząć i zakończyć wyprawę na dowolnym szczycie, ale musi odwiedzać kolejne szczyty bez pomijania żadnego z nich. Oznacza to, że każda wyprawa odpowiada pewnemu spójnemu przedziałowi $[l, r]$ naszej tablicy.

Wyprawa jest poprawna, jeśli zawiera co najwyżej D szczytów o wysokości większej lub równej H. Naszym zadaniem jest znalezienie maksymalnej długości takiego przedziału oraz wszystkich pozycji, od których może się on zaczynać.

Zauważmy kilka istotnych własności:

1. **Poprawność przedziału:** Niech $cnt(l, r)$ oznacza liczbę elementów $A_i \geq H$ na przedziale $[l, r]$. Przedział jest poprawny wtedy i tylko wtedy, gdy

$$cnt(l, r) \leq D.$$

2. **Rozszerzanie przedziału:** Jeśli rozszerzymy przedział $[l, r]$ o jeden element, wartość *cnt* pozostanie bez zmian lub zwiększy się o 1, zależnie od tego, czy nowy szczyt ma wysokość co najmniej H. Analogicznie, jeśli skrócimy przedział o jeden element, wartość *cnt* pozostanie bez zmian lub zmniejszy się o 1. Dzięki temu możemy aktualizować *cnt* w czasie stałym, bez konieczności ponownego przeglądania całego przedziału.

3. **Monotoniczność:** Jeśli pewien przedział zawiera więcej niż D wysokich szczytów, to rozszerzanie go o kolejne elementy nie sprawi, że stanie się poprawny. Dodawanie nowych szczytów nie może bowiem zmniejszyć liczby wysokich szczytów w przedziale.

Powyższe własności pozwalają nam sprawnie kontrolować poprawność przedziału podczas zmieniania jego granic. Możemy więc wykorzystać algorytm **dwóch wskaźników**, nazywany również **gąsienicą**, aby znaleźć rozwiązanie w czasie liniowym.

Krok 1: Najdłuższa wyprawa dla każdego początku

Zastosujemy algorytm dwóch wskaźników l oraz r, oznaczających odpowiednio początek i koniec aktualnie rozpatrywanej wyprawy. Będziemy również utrzymywać zmienną *cnt*, określającą liczbę szczytów o wysokości co najmniej H w tym przedziale.

Naszym celem jest wyznaczenie dla każdej pozycji początkowej l najdalej wysuniętego końca r, do którego Bajtek może dotrzeć, nie przekraczając limitu D wysokich szczytów.

Początkowo ustawiamy $l = 1$, $r = 0$ oraz $cnt = 0$, co odpowiada pustemu przedziałowi. Następnie dla kolejnych pozycji l wykonujemy następujące operacje:

1. Przesuwamy prawy wskaźnik r tak daleko, jak to możliwe, dopóki dołączenie kolejnego szczytu nie spowoduje przekroczenia limitu D. Przy każdym przesunięciu odpowiednio aktualizujemy *cnt*.

2. W ten sposób otrzymujemy najdłuższą poprawną wyprawę rozpoczynającą się na pozycji l, o długości $r - l + 1$.

3. Następnie przesuwamy lewy wskaźnik l o jeden w prawo, usuwając z *cnt* wkład opuszczanego szczytu, jeśli należał on do przedziału. Jeśli przedział

był pusty, ustawiamy $r = l - 1$. Dla nowego początku ponownie próbujemy przesunąć prawy wskaźnik jak najdalej.

Powtarzając ten proces, rozpatrzymy wszystkie możliwe początki wyprawy i dla każdego z nich znajdziemy maksymalną osiągalną długość. Co istotne, żaden ze wskaźników nie musi się cofać, dzięki czemu nie przeglądamy tych samych elementów wielokrotnie.

Krok 2: Wyznaczenie wyniku

Dla każdej pozycji początkowej l znamy już długość najdłuższej poprawnej wyprawy. Na bieżąco aktualizujemy maksymalną znalezioną długość L oraz przechowujemy wszystkie indeksy początkowe, dla których uzyskano tę długość.

Ponieważ rozpatrujemy pozycje l w kolejności rosnącej, zapisane indeksy będą już uporządkowane i nie wymagają dodatkowego sortowania.

Złożoność obliczeniowa

Oba wskaźniki przesuwają się wyłącznie do przodu, więc każdy z nich wykona łącznie co najwyżej N przesunięć. Dodatkowo dla każdej pozycji l możemy wykonać co najwyżej jedno nieudane sprawdzenie możliwości rozszerzenia przedziału. Wszystkie te operacje zajmują czas stały, zatem całkowita złożoność czasowa wynosi $\mathcal{O}(N)$.

Przechowujemy tablicę wysokości oraz listę optymalnych pozycji początkowych, które mogą zawierać po $\mathcal{O}(N)$ elementów. Złożoność pamięciowa wynosi więc $\mathcal{O}(N)$.

6 Pielęgnacja Trawnika

Obserwacje wstępne

Po wybraniu skupiska k o wartości A_k zmieniane są dokładnie skupiska leżące w jego wierszu lub kolumnie. Oznaczmy przez $\text{rowMax}[R]$ i $\text{colMax}[C]$ największe początkowe wartości w danym wierszu i kolumnie. Największa wartość wśród skupisk zmienionych przez wybór k jest równa

$$Q_k = \max\big(\text{rowMax}[R_k], \text{colMax}[C_k]\big) - A_k.$$

Wartość ta jest nieujemna, ponieważ wybrane skupisko należy do obu rozważanych zbiorów i po operacji ma wartość 0. Skupiska spoza wybranego wiersza i kolumny nie zmieniają się. Jeżeli ich największą wartość oznaczymy przez U_k, to wynik wyboru k jest równy $\max(Q_k, U_k)$, przyjmując $U_k = 0$, gdy taki zbiór jest pusty.

Podzadanie 1

Współrzędne nie przekraczają 50. Rozpatrujemy kolejno każdy niepusty wiersz R. Dla każdej kolumny wyznaczamy największą wartość skupiska leżącego w tej kolumnie, ale poza wierszem R. Maksima prefiksowe i sufiksowe po kolumnach pozwalają dla punktu (R, C) w czasie $O(1)$ znaleźć największą wartość poza wierszem R i kolumną C, czyli U_k. Wartość Q_k obliczamy z wcześniej wyznaczonych maksimów wierszy i kolumn. Sprawdzamy w ten sposób każde skupisko i wybieramy najmniejszy wynik. Jeśli D oznacza liczbę niepustych wierszy, algorytm działa w czasie $O(D(N + 50))$, gdzie $D \leq 50$, i używa $O(N + 50)$ pamięci.

Podzadanie 2

Współrzędne nie przekraczają 1000. Ustalamy dowolne skupisko o globalnie największej wartości G, leżące w (R, C). Jeśli wybrane skupisko k nie leży ani w wierszu R, ani w kolumnie C, to G pozostaje niezmienione i $U_k = G$. Dla wyborów z wiersza R wyznaczamy największą wartość w każdej kolumnie poza tym wierszem, a następnie zapamiętujemy dwie największe spośród tych wartości. Pozwala to w $O(1)$ pominąć kolumnę C_k. Symetrycznie obsługujemy wybory z kolumny C. Po jednorazowym przygotowaniu tablic każdy kandydat jest sprawdzany w czasie stałym. Złożoność tego rozwiązania wynosi $O(N + 1000)$ czasu i $O(N + 1000)$ pamięci.

Rozwiązanie pełne

Duże numery współrzędnych uniemożliwiają użycie tablic indeksowanych bezpośrednio. Kompresujemy więc osobno wszystkie współrzędne wierszy i kolumn, a następnie obliczamy rowMax i colMax. Za pomocą wyszukiwania binarnego znajdujemy najmniejszy możliwy wynik M. Początkowy przedział wyszukiwania to $[0, \max_{1 \leq i \leq N} A_i]$.

Dla ustalonej wartości M nazwiemy skupisko złym, jeśli $A_i > M$. Każde złe skupisko musi zostać objęte operacją, ponieważ skupisko niezmienione zachowałoby wartość większą niż M. Zliczamy złe skupiska w każdym wierszu i każdej kolumnie. Niech K oznacza ich łączną liczbę. Kandydat k obejmuje

$$\text{badRow}[R_k] + \text{badCol}[C_k] - [A_k > M]$$

złych skupisk. Ostatni składnik usuwa podwójne policzenie wybranego skupiska. Pierwszy wymagany warunek ma zatem postać

$$\text{badRow}[R_k] + \text{badCol}[C_k] - [A_k > M] = K.$$

Ten warunek gwarantuje, że każde skupisko nieobjęte operacją ma wartość najwyżej M. Dla skupisk objętych operacją korzystamy z obserwacji wstępnej i sprawdzamy drugi warunek

$$\max\big(\text{rowMax}[R_k], \text{colMax}[C_k]\big) - A_k \leq M.$$

Oba warunki są konieczne i wystarczające, więc kandydat spełniający je osiąga wynik nie większy niż M.

Jeśli istnieje złe skupisko, wybieramy dowolne z nich. Każdy poprawny kandydat musi dzielić z nim wiersz albo kolumnę, bo w przeciwnym razie to skupisko nie zostałoby zmienione. Wystarczy więc sprawdzić tylko kandydatów spełniających jeden z tych dwóch warunków. Gdy nie ma złych skupisk, dowolna operacja nie zwiększa wartości, zatem M jest od razu osiągalne. Osiągalność jest monotoniczna: jeśli można uzyskać maksimum nie większe niż M, można też uzyskać każde większe ograniczenie. Wyszukiwanie binarne zwraca więc najmniejszy możliwy wynik.

Złożoność obliczeniowa

Kompresja współrzędnych zajmuje $O(N \log N)$ czasu. Jedno sprawdzenie wartości M działa w $O(N)$, a liczba sprawdzeń wynosi $O(\log A_{\max})$. Złożoność czasowa wynosi $O(N \log N + N \log A_{\max})$, a złożoność pamięciowa wynosi $O(N)$.

7 Pociąg Do Sum

Obserwacje wstępne

Niech $a_1, a_2, \ldots, a_d$ będzie szukaną permutacją liczb od 1 do d (gdzie $d \geq 3$ jest nieparzyste), a $P = \frac{d(d+1)}{2}$ oznacza zarówno łączną liczbę wagonów w pociągu, jak i masę pojedynczego ciężkiego wagonu.

Z treści zadania wynikają dwie kluczowe własności:

1. **Struktura bloków:** Pociąg składa się z d naprzemiennych bloków wagonów lekkich (o masie 1) oraz ciężkich (o masie P). Ponieważ d jest nieparzyste, pierwszy i ostatni blok (a_1 oraz a_d) składają się z wagonów lekkich. Oznacza to, że każdy blok wagonów ciężkich ($a_2, a_4, \ldots, a_{d-1}$) sąsiaduje z obu stron z blokami wagonów lekkich. Zauważmy też, że długość każdego bloku jest $\leq d$, czyli w szczególności $< P$.

2. **Klasyfikacja zmierzonych mas:** Każdy ważony spójny fragment zawiera co najwyżej jeden blok ciężkich wagonów. Jeśli fragment składa się z y ciężkich wagonów ($0 \leq y < P$) oraz x lekkich wagonów ($0 \leq x < P$), to jego

masa wynosi $s = yP + x$. Dzieląc s z resztą przez P, otrzymujemy jednoznacznie liczbę ciężkich wagonów $y = \lfloor s/P \rfloor$ oraz liczbę lekkich wagonów $x = s \bmod P$. Wyróżniamy zatem trzy rodzaje fragmentów:

- *wyłącznie lekkie* ($y = 0$, $1 \leq x < P$): ich masa spełnia $s < P$,
- *wyłącznie ciężkie* ($y \geq 1$, $x = 0$): ich masa spełnia $P \mid s$,
- *mieszane* ($y \geq 1$, $1 \leq x < P$): ich masa spełnia $s > P$ oraz $P \nmid s$.

Krok 1: Oddzielenie fragmentów mieszanych i odtworzenie długości bloków

Wszystkie m podanych na wejściu mas umieszczamy w multizbiorze $\mathcal{A}$. Dopóki największy element $t = \max \mathcal{A}$ spełnia warunki $t > P$ oraz $P \nmid t$, pochodzi on z fragmentu mieszanego - przenosimy go zatem do osobnego multizbioru $\mathcal{M}$.

Gdy największy element $t = \max \mathcal{A}$ nie spełnia powyższego warunku, musi odpowiadać fragmentowi jednorodnemu (wyłącznie lekkiemu, gdy $t < P$, lub wyłącznie ciężkiemu, gdy $P \mid t$). Każdy fragment jednorodny jest zawarty w pewnym pojedynczym bloku wagonów tego samego typu, więc jego długość nie przekracza długości tego bloku. Oznacza to, że najdłuższy pozostały w $\mathcal{A}$ fragment jednorodny musi stanowić cały najdłuższy nierozpatrzony jeszcze blok danego typu:

- jeśli $P \mid t$, to istnieje blok ciężkich wagonów o długości $l = t/P$ (masa pojedynczego wagonu to $w = P$),
- jeśli $t < P$, to istnieje blok lekkich wagonów o długości $l = t$ (masa pojedynczego wagonu to $w = 1$).

Blok złożony z l wagonów o masie w generuje dla każdego $i \in \{1, 2, \ldots, l\}$ dokładnie $l - i + 1$ spójnych fragmentów o masie $i \cdot w$. Usuwamy z multizbioru $\mathcal{A}$ wszystkie te wartości wraz z ich krotnościami i powtarzamy procedurę aż do opróżnienia $\mathcal{A}$. Po zakończeniu tego etapu znamy już długości wszystkich ciężkich i lekkich bloków, a ponadto w multizbiorze $\mathcal{M}$ znajdują się dokładnie wszystkie masy fragmentów mieszanych.

Krok 2: Wyznaczenie sąsiadów bloków ciężkich

Teraz będziemy przetwarzać bloki ciężkich wagonów w kolejności malejących długości, korzystając wyłącznie z multizbioru $\mathcal{M}$. Niech $t = \max \mathcal{M}$. Zauważmy, że $b = \lfloor t/P \rfloor$ to długość najdłuższego jeszcze nierozważanego bloku ciężkiego. Jak wynika z obserwacji wstępnych, ten blok ma dwa sąsiadujące z nim bloki lekkie. Oznaczmy ich długości jako a i c.

Z maksymalności t wynika, że $t = bP + a + c$. Po przekształceniu:

$$a + c = t - bP.$$

Zauważmy teraz, że dowolna suma $r \in \mathcal{M}$ taka, że $\lfloor t/P \rfloor = \lfloor r/P \rfloor$, odpowiada fragmentowi mieszanemu, który zawiera aktualnie rozważany blok ciężki oraz fragmenty jego lekkich sąsiadów. Liczbę takich sum r (oznaczmy ją przez I) można łatwo wyznaczyć, ponieważ będą to kolejne największe elementy z $\mathcal{M}$.

Z drugiej strony liczbę fragmentów zawierających blok ciężki o długości b i fragmenty jego lekkich sąsiadów o długościach a i c można wyrazić jako $(a+1)(c+1) - 1$. Daje nam to równość:

$$I = ac + a + c.$$

Teraz można iteracyjnie sprawdzić, jakie a i c spełniają równości $I = ac + a + c$ oraz $a + c = t - bP$. Istnienie i jedyność takich a i c (z dokładnością do kolejności) wynika z rozwiązania tego układu równań, które sprowadza się do równania kwadratowego:

$$x^2 - (t - bP)x + I - (t - bP) = 0.$$

Po wyznaczeniu a i c zapamiętujemy, że blok o długości b ma sąsiadów o długościach a i c (można przechowywać tę informację jako dwie krawędzie w grafie nieskierowanym).

Krok 3: Usunięcie pozostałych mas związanych z blokiem b

Aby utrzymać niezmiennik dla kolejnych (krótszych) bloków ciężkich wagonów, musimy usunąć z multizbioru $\mathcal{M}$ masy wszystkich pozostałych fragmentów mieszanych korzystających z bloku b - czyli takich, które zawierają od 1 do b jego ciężkich wagonów oraz część bloku a lub część bloku c.

Znając wartości a, b, c, tworzymy lokalny ciąg wag postaci $\underbrace{1, \ldots, 1}_{a}, \underbrace{P, \ldots, P}_{b}, \underbrace{1, \ldots, 1}_{c}$ długości $a + b + c$. Następnie przeglądamy wszystkie jego spójne fragmenty. Dla każdego fragmentu o masie s, który jest mieszany ($s > P$ oraz $P \nmid s$), usuwamy jedno wystąpienie s z multizbioru $\mathcal{M}$. Fragmenty niemieszane zostały już usunięte w Kroku 1. Po usunięciu wszystkich odpowiednich sum przechodzimy do analizowania kolejnego bloku ciężkich wagonów (Krok 2). Powtarzamy ten proces tak długo, aż multizbiór $\mathcal{M}$ stanie się pusty.

Krok 4: Odtworzenie permutacji

Ponieważ długości wszystkich bloków w ciągu $a_1, a_2, \ldots, a_d$ są parami różne, każdy blok odpowiada dokładnie jednemu wierzchołkowi grafu. Każdy blok ciężkich wa-

gonów a_{2k} został połączony krawędziami ze swoimi dwoma sąsiadami a_{2k-1} oraz a_{2k+1}, natomiast skrajne bloki lekkie a_1 i a_d mają stopień równy 1.

Otrzymany graf jest zatem prostą ścieżką $a_1 - a_2 - \cdots - a_d$. Znajdujemy dowolny wierzchołek o stopniu 1 i przechodzimy ścieżkę w głąb, wypisując kolejne odwiedzane wierzchołki. Otrzymujemy w ten sposób permutację $(a_1, a_2, \ldots, a_d)$ lub jej odbicie $(a_d, \ldots, a_2, a_1)$. Odwrócenie permutacji odpowiada odwróceniu kolejności wagonów w całym pociągu (ponieważ d jest nieparzyste, bloki lekkie i ciężkie pozostają na swoich typach pozycji), co daje dokładnie ten sam multizbiór zmierzonych mas - obie odpowiedzi są więc poprawne.

Złożoność obliczeniowa

Zauważmy, że każdą sumę dodamy i usuniemy z $\mathcal{A}$ oraz z $\mathcal{M}$ co najwyżej po razie. Zatem, ponieważ pojedyncza operacja na multizbiorze działa w czasie $\mathcal{O}(\log m)$, to cały algorytm działa w czasie $\mathcal{O}(m \log m)$.

Ciekawostka: Da się pokazać, że ograniczenie górne na m w zależności od d to $\mathcal{O}(d^3)$. Zatem ograniczenia na m podane w treści miały jedynie charakter pomocniczy.