Sabotaż

PREOI 2025 Kod zadania: **sab**
Dzień 2 – 26 stycznia 2025

Problem

Mamy dany ciąg n punktów znajdujących się w przestrzeni m-wymiarowej. Naszym zadaniem jest podzielić ten ciąg na takie spójne fragmenty, żeby zmaksymalizować sumę po największych odległościach w każdym fragmencie. Odległością $Odl(A, B)$ pomiędzy dwoma planetami $A = (x_1, x_2, \ldots, x_m)$ i $B = (y_1, y_2, \ldots, y_m)$ nazywamy:

$$Odl(A, B) = \sum_{k=1}^{m} |x_k - y_k|$$

Limity

- $n \leq 10^5$
- $m \leq 8$

Rozwiązanie brutalne

Pierwszą obserwacją w tym zadaniu jest to, że każdy fragment możemy skrócić tak, żeby najdalsza para była na końcach. Wówczas powstałe luki możemy uzupełnić nowymi fragmentami, potencjalnie nawet poprawiając wynik.
Narzucającym się pomysłem jest programowanie dynamiczne. I rzeczywiście niech $dp[i]$ oznacza największy możliwy wynik przy wykorzystaniu jedynie pierwszych i punktów. Wówczas $dp[i] = \max \begin{cases} dp[i-1] \\ dp[j-1] + Odl(punkt_j, punkt_i) \text{ dla } (j < i) \end{cases}$
Takie rozwiązanie ma złożoność $O(n^2)$ i uzyskuje 25 punktów. To rozwiązanie (lekko zmodyfikowane) jest też rozwiązaniem podzadania $m = 1$ lub *współrzędne planet są z przedziału* $[0, 1]$. W ten sposób można skompletować 61 punktów.

Rozwiązanie wzorcowe

Oznaczmy przez $A[i][j]$ j-tą współrzędną punktu i. Pierwszym krokiem do rozwiązania wzorcowego jest zadanie następującego pytania: licząc odległość między dwoma punktami a i b, które wpółrzędne dodam z punktu a i odejmę z punktu b, a które dodam z punktu b i odejmę z punktu a?

Możemy spróbować to wymusić. Niech $wartosci[i][maska]$ oznacza $\sum_{j=1}^{m} A[i][j] \cdot \begin{cases} 1, \text{ jeśli } j \in maska \\ -1, \text{w przeciwnym wypadku} \end{cases}$.
$maska$ oznacza zbiór liczb całkowitych od 1 do m, jest to podzbiór wymiarów. O dokładniejszej implementacji $maska$ można przeczytać na końcu tego omówienia.
Wówczas:

$$Odl(punkt_i, punkt_j) = \max_{maska}(wartosci[i][maska] - wartosci[j][dopelnienie\_maska])$$

Wzór ten znajduje zastosowanie w zaskakująco wielu zadaniach, dlatego zachęcam do przeanalizowania go.
Czyli dynamik z rozwiązania brutalnego wygląda następująco:

$$dp[i] = \max_{j<i, maska}(dp[j-1] + wartosci[i][maska] - wartosci[j][dopelnienie\_maska])$$

Oczywiście na końcu musimy jeszcze zmaksować obliczony wynik z $dp[i-1]$.
Gdybyśmy jednak przechodzili najpierw po $maska$ a dopiero potem po j uzyskamy coś następującego:

$$dp[i] = \max_{maska}(\max_{j}(dp[j-1] - wartosci[j][dopelnienie\_maska]) + wartosci[i][maska])$$

Czemu więc nie liczyć $\max_j(dp[j-1] - wartosci[j][dopelnienie\_maska])$ wcześniej i nie zapamiętywać wyników na jakiejś pomocniczej tablicy? Istotnie, niech

$$pom[maska] = \max_{j<i}(dp[j-1] - wartosci[j][maska])$$

Wówczas

$$dp[i] = \max_{maska}(pom[dopelnienie\_maska] + wartosci[i][maska])$$

możemy już policzyć w $O(2^m)$. Pytanie więc czy możemy szybko wyznaczać tablice pom? Chcemy jedynie zaktualizować $pom[maska]$ o $dp[i-1] - wartosci[i][maska]$, więc to również możemy robić w czasie $O(2^m)$.
W ten sposób uzyskujemy algorytm działający w złożoności $O(n2^m m)$ który przy dobrej implementacji dostaje 100 punktów, a przy gorszej co najmniej 82 punkty.
Dlaczego nasz algorytm ma złożoność $O(n2^m m)$, a nie $O(n2^m)$? Ponieważ musimy wyznaczać tablice $wartosci$. Obliczenie $wartosci[i][maska]$ zajmuje nam $O(m)$. Możemy jednak obliczać całe $wartosci[i]$ w czasie $O(2^m + m)$ przy pomocy programowania dynamicznego.
Na początku normalnie liczymy $wartosci[i][pusta\_maska]$. Potem aby policzyć $wartosci[i][maska]$ wystarczy wziąć dowolne $j \in maska$ i mamy:

$$wartosci[i][maska] = wartosci[i][maska \setminus \{j\}] + 2A[i][j]$$

Oczywiście, żeby móc to zrobić trzeba liczyć $wartosci[i]$ w dobrej kolejności oraz móc szybko wyznaczać j. Rozwiązanie które to umożliwia, to reprezentowanie $maska$ jako zapis binarny liczb od 0 do $2^m - 1$. Jeśli ktoś nie miał wcześniej doczynienia z takim rozwiązaniem to polecam poniższego bloga. Wówczas satysfakcjonuje nas zwykła kolejność rosnąca i liczenie j przy pomocy operatora `__builtin_clz`.
Po zastosowaniu tej optymalizacji otrzymujemy rozwiązanie działające w złożoności $O(n2^m)$, które powinno dostawać komplet punktów.