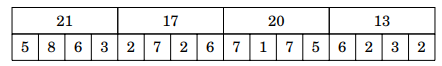
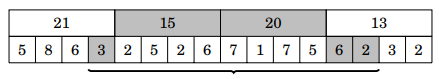
Pierwiastki  
[Square root decomposition]

# Podstawa

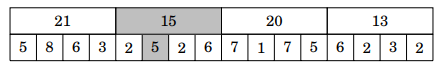
Mamy ciąg liczb . Mamy następujący problem: obsłużyć zapytań postaci 1) podaj sumę na przedziale , 2) zmień wartość na danym indeksie na inną, podaną. Problem można rozwiązać dzieląc ciąg na przedziały składowe: „pierwiastki”.



1)



2)

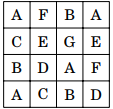


Mamy rozwiązanie w z bardzo małą stałą.

## Do kminienia

1. Dodawanie na przedziale; inne operacje
2. Implementacja

# Podział na algorytm [Combining algorithms]



Mamy daną prostokątną tabelę jakichś wartości (przypuśćmy, że liter, ale niech wielkość alfabetu ). Dla każdej z występujących wartości chcemy znaleźć jaka jest minimalna odległość manhattańska () między dwoma takimi samymi wartościami.

Dwa algorytmy aby rozwiązać ten problem:

1. Niech ilość wystąpień litery to . Wtedy można po prostu dla wszystkich wartości, dla każdej możliwe pary sprawdzić odległość. Złożoność na literę.
2. Dla każdej z liter puszczamy równoległego BFSa ze wszystkich wystąpień tej litery. Złożoność na literę.

Podzielmy litery na dwie grupy: wielką W (), i małą V (. Zauważmy, że . Użyjmy algorytmu b) aby rozpatrzyć grupę wielką, oraz a) aby rozpatrzyć grupę małą. Otrzymujemy złożoność:

Ta wartość jest minimalna dla Czyli złożoność to:

## Do kminienia

1. Rosyjscy oligarchowie. Dane jest oligarchów i firm. Każda firma ma swoją wartość . Oligarchowie mają sumaryczną wartość swojego majątku, która jest równa sumie wartości firm których są właścicielami (jedna firma może mieć wielu właścicieli, ale nic to nie zmienia). Obsłuż zapytań: 1) Oligarcha staje się właścicielem danej firmy (i zostaje nim do końca). 2) Wartość firmy zmienia się na inną podaną. 3) Podaj sumaryczny majątek danego oligarchy.
2. Kontenery z XXIV OI.

# Algorytm Mo

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 5 | 4 | 2 | 4 | 3 | 3 | 4 | …………🡪………… | 4 | 2 | 5 | 4 | 2 | 4 | 3 | 3 | 4 |

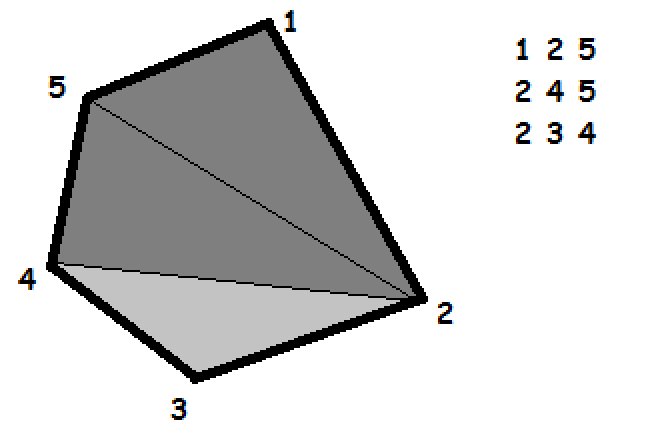
Masz dane zapytań offline o przedziały w ciągu, takich że różna kolejność zapytań na wejściu nie zmienia wyniku. Dodatkowo, mając dane rozwiązanie (i inne dane) dla danego przedziału, umiesz go przekształcić w inny usuwając/dodając po jednym elemencie z lewej/prawej strony (jedną taką operację wykonujesz w )

Algorytm Mo sortuje przedziały w taki sposób, że wykonasz co najwyżej operacji (sumaryczna złożoność ).

|  |
| --- |
| # query.left // Q -> pierwiastek do którego należy query  # `//` to dzielenie z zaokrągleniem w dół  Q = floor(sqrt(n))  def mo\_compare(lhs, rhs):   if lhs.left // Q != rhs.left // Q:  return lhs.left // Q < rhs.left // Q  else:  return lhs.right < rhs.right |

Tym razem dzielimy zapytania na pierwiastki, a nie ciąg. -ty pierwiastek zawiera zapytania których lewy koniec jest z przedziału .

## Do kminienia

1. Dowód złożoności (Hint: amortyzacja)
2. Triangulacja. Masz dane punkty wielokąta oraz jego triangulację – podział wielokąta na trójkąty, podana w formie indeksów wierzchołków wielokąta każdego z trójkątów. Każdy z trójkątów ma jakiś kolor. Ile co najwyżej cięć wielokąta możesz wykonać, takich że w żadnym momencie jakieś dwie części mają wspólne kolory? (Alternatywnie można rozwiązać trikiem na łączenie zbiorów. Hint: preorder/postorder z Megalopolis).

W tym wielokącie można wykonać tylko jedno cięcie   
(na odcinku 2 – 4).

Trójkąty to (1, 2, 5), (2, 4, 5), (2, 3, 4).