Lista 5

Zadanie 3

---

1 Rozwiązanie w czasie liniowym

Wystarczy wziąć strukturę RB-Tree, w której operacje `Search`, `Insert` oraz `Delete` mają złożoność równą $O(\log n)$.

Wówczas operacja `Min-Luka` działa poprzez generowanie listy elementów w drzewie w kolejności `inorder` i liniowo przegląda każdą parę elementów, znajdując najmniejszą różnicę. Jeśli znaleziono różnicę równą 0 zwraca 0 jako, że to najmniejsza możliwa różnica pomiędzy elementami. `Min-Luka` ma wtedy złożoność obliczeniową równą $O(n)$.

2 Rozwiązanie w stałym czasie

Weźmy strukturę RB-Tree, i dodajmy do definicji węzła pola:

1. `min` — określające min z elementów w danym pod-drzewie tworzonym przez dany węzeł
2. `max` — określające max z elementów w danym pod-drzewie tworzonym przez dany węzeł
3. `local-min-diff` — określające lokalny żądany wynik operacji `Min-Luka`

Należy zauważyć, że nowe pola można zdefiniować w następujący sposób:

$\forall_{x \in T}$:

$$x.\texttt{min} = \begin{cases} x.\texttt{left}.\texttt{min} & \text{jeśli } x.\texttt{left} \neq \text{NIL} \\ x.\texttt{value} & \text{otherwise} \end{cases}$$

$$x.\texttt{max} = \begin{cases} x.\texttt{right}.\texttt{max} & \text{jeśli } x.\texttt{right} \neq \text{NIL} \\ x.\texttt{value} & \text{otherwise} \end{cases}$$

$\texttt{tmp} = \{\infty\}$

$\texttt{if } x.\texttt{left} \neq \text{NIL}$:

  $\texttt{tmp} = \{x.\texttt{left}.\texttt{local-min-diff},\ x.\texttt{value} - x.\texttt{left}.\texttt{max}\}$

$\texttt{if } x.\texttt{right} \neq \text{NIL}$:

$\texttt{tmp} = \texttt{tmp} \cup \{x.\texttt{right.local-min-diff},\ x.\texttt{right.min} - x.\texttt{value}\}$
$x.\texttt{local-min-diff} = \min(\texttt{tmp})$

Dzięki czemu można łatwo zauważyć, że nowo dodane pola zależą jedynie od wartości węzła lub jego bezpośrednich potomków. Wówczas korzystając z twierdzenia 14.1 z książki „Wprowadzenie do algorytmów” mamy pewność, że złożoność czasowa operacji `Insert`, `Delete` oraz `Search` nie zmieni się.

Oczywiście musimy zmodyfikować standardowe operacje na RB-Tree.

2.1 `Insert`

Jedyne co musimy dodać do standardowej operacji `Insert` to zapamiętywanie przez węzły wartości minimalnej i maksymalnej dla pod-drzew które tworzą.

Każdy z węzłów startuje z polami `min` oraz `max` przyrównanymi do wartości tego węzła. Pole `local-min-diff` aktualizujemy na podstawie wzoru podanego wyżej.

Kiedy dodajemy nowy węzeł x to dla każdego węzła który napotka węzeł x aktualizujemy wartości pól `min` oraz `max` dla tego napotkanego węzła (jeśli wartość x jest większa niż pole `max` węzła napotkanego to zmieniamy wartość tego pola, `min` analogicznie).

Przy ewentualnej rotacji aktualizujemy węzły których dotyczy dana rotacja w czasie $O(\log n)$ dlatego, że aktualizujemy wartości pól przodków węzłów rotowanych. Zważając na wysokość drzewa $\leqslant 2\log(n+1)$ nasza złożoność pozostanie $O(\log n)$.

2.2 `Delete`

Przy usuwaniu musimy zaktualizować wartości pól `min` oraz `max` przodków węzła usuwanego. Robimy to w czasie $O(\log n)$ dlatego, że RB-Tree ma wysokość $\leqslant 2\log(n+1)$.

2.3 `Search`

Używamy tutaj standardowej operacji `RB-Search` bez żadnych modyfikacji.

2.4 `Min-Luka`

Zwracamy wartość pola `local-min-diff` węzła `root` drzewa. Jest to pojedyncza operacja przetwarzająca jeden węzeł przez co złożoność obliczeniowa tej operacji wynosi $O(1)$.