Zadanie 8 lista 6

Jakub Kuciński, prowadzący Szymon Dudycz

26 czerwca 2020

1 Treść

Napisz procedurę $Split(T, k)$ rozdzielającą drzewo AVL T na dwa drzewa AVL: jedno zawierające klucze mniejsze od k i drugie zawierające pozostałe klucze. Jaka jest złożoność Twojej procedury?

2 Idea rozwiązania

Żeby podzielić T na dwa drzewa AVL jedno T_1 zawierające klucze mniejsze od k i drugie T_2 zawierające pozostałe klucze musimy wyznaczyć, które elementy będą znajdowały się w którym drzewie. Drzewo AVL posiada porządek BST. Możemy przejść T w poszukiwaniu klucza k. Wtedy za każdym razem jak skręcamy w lewo możemy dodać wszystkie klucze z prawego poddrzewa do T_2 (wszystkie są większe od k), jeśli skręcamy w prawo to klucze z lewego poddrzewa do T_1 (wszystkie są mniejsze od k). Zauważmy, że jeśli będziemy chcieli dodać klucze $T.left$ do T_1 i T_1 jest drzewem AVL o kluczach mniejszych niż k pochodzących z prawego syna (z $T.right$), to dodawane klucze tworzą drzewo AVL, w którym wszystkie klucze są mniejsze niż w już utworzonym T_1. Analogicznie dla T_2. Dzięki temu spostrzeżeniu można utworzyć funkcję $Join(T_1, T_2)$ łączącą dwa drzewa AVL, takie że wszystkie klucze pierwszego drzewa są mniejsze od wszystkich kluczy drugiego drzewa, działającą w czasie $\mathcal{O}(\log n_1 + \log n_2)$.

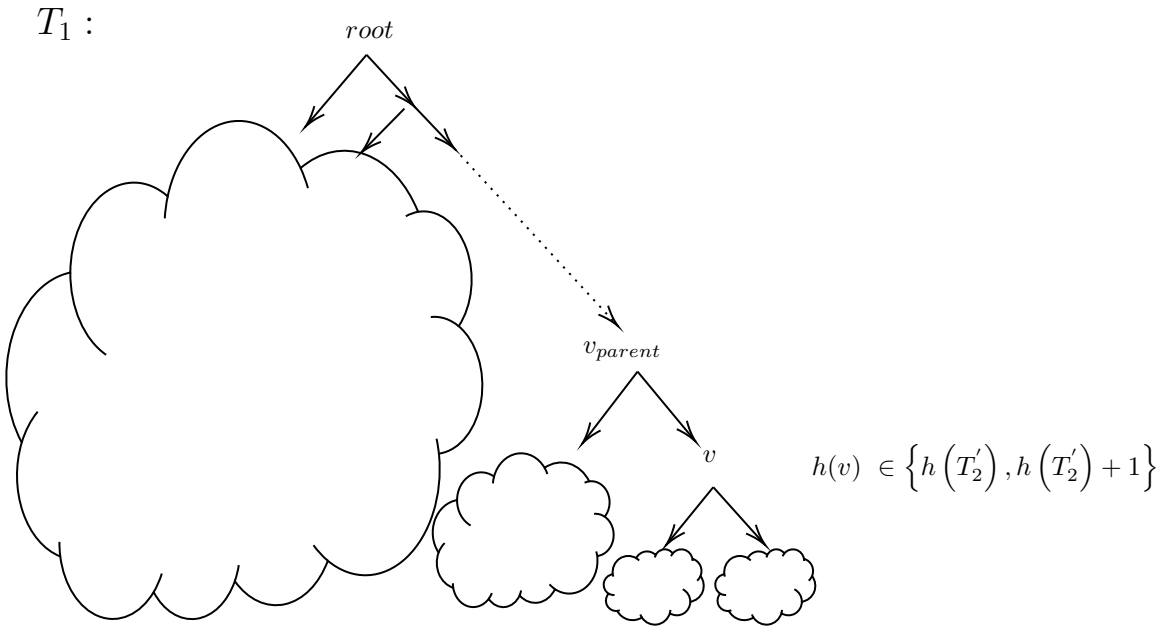
3 Funkcja Split

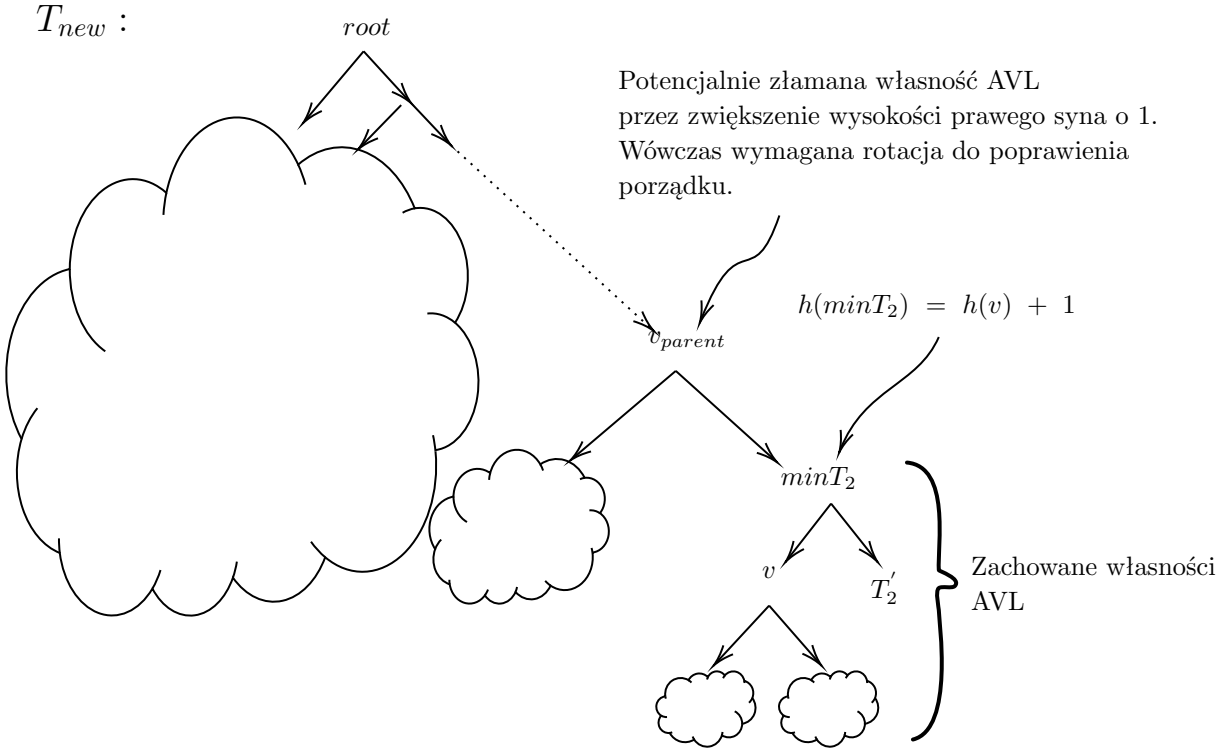
Algorithm 1 $Split(T, k)$

```
1: if T == null then
2:     return (null, null)
3: if T.value == k then
4:     return (T.left, Join(makeAVL(T.value), T.right))
5: if T.value < k then
6:     (T_1, T_2) = Split(T.right, k)
7:     return (Join(T.left, T_1), T_2))
8: (T_1, T_2) = Split(T.left, k)
9: return (T_1, Join(T_2, T.right))
```

4 Funkcja Join i jej złożoność

Funkcja *Join* przyjmuje za argumenty dwa drzewa AVL T_1 i T_2 takie, że wszystkie klucze T_1 są mniejsze od wszystkich kluczy T_2. Na początek potrzebujemy wyznaczyć wysokości T_1 i T_2. Oba sprawdzenia można wykonać w czasie $\mathcal{O}(\log n_1 + \log n_2)$ korzystając z informacji o balansie wierzchołka i idąc za każdym razem do cięższego syna. Rozważmy przypadek, gdy $h(T_1) \geqslant h(T_2)$. Drugi przypadek jest analogiczny. Musimy znaleźć wierzchołek $minT_2$ w T_2, który jest mniejszy od każdego z pozostałych T_2. Niech T_2' będzie drzewem T_2 po usunięciu $minT_2$. Wierzchołek z $minT_2$ będzie mógł być korzeniem poddrzewa, którego prawym synem będzie T_2', a lewym pewne poddrzewo z T_1 o korzeniu w v. Będziemy chcieli znaleźć taki wierzchołek v w T_1, że $h(v) \in \{h(T_2'), h(T_2') + 1\}$ oraz v jest na skrajnie prawej ścieżce. Zapewnimy sobie w ten sposób, że nowe drzewo o korzeniu w $minT_2$, lewym synie v i prawym synie T_2' zachowuje własności AVL (różnica wysokości synów $\leqslant 1$ oraz zachowany jest porządek BST). Tak utworzne drzewo podpinamy w dotychczasowe miejsce v. Zachowana zostaje własność BST dla całego drzewa, ale potencjalnie mogło dojść do złamania własności AVL w nowym ojcu $minT_2$, bo wysokość jego prawego syna zwiększyła się o 1. Można to naprawić jedną pojedynczą lub podwójną rotacją. Poniższe rysunki ilustrują działanie funkcji *Join*.





Wyznaczenie wysokości T_1 i T_2 $\mathcal{O}(\log n_1 + \log n_2)$ operacji. Znalezienie i usunięcie najmniejszego elementu w T_2 sprowadza się do przejścia maksymalnie lewą ścieżką po T_2 i jednej operacji *delete* - złożoność $\mathcal{O}(\log n_2)$. Znalezienie odpowiedniego v sprowadza się z kolei do przechodzenia T_1 maksymalnie prawą ścieżką licząc przy tym wysokość jaką ma poddrzewo (znamy wysokość T_1 i balans każdego wierzchołka, więc wiemy czy przechodząc w prawo wysokość zmniejsza się o 2 czy o 1). Zatrzymujemy się gdy wysokość będzie mniejsza równa $h(T_2) + 1$, czyli $\mathcal{O}(\log n_1 - \log n_2)$ operacji. Utworzenie nowego poddrzewa i ewentualna rotacja do przywrócenia porządku AVL wymaga stałej liczby operacji - $\mathcal{O}(1)$. Zatem sumaryczna złożoność funkcji $Join$ wynosi $\mathcal{O}(\log n_1 + \log n_2)$.

5 Złożoność Split

Funkcja *Split* wykona się $\mathcal{O}(log(n))$-krotnie, bo za każdym razem schodzi o jeden poziom drzewa T niżej. Przy każdym wywołaniu *Split* funkcja $Join$ wykona się co najwyżej raz. Stąd mamy łącznie $\mathcal{O}(log(n)) \cdot \mathcal{O}(log(n)) = \mathcal{O}(log^2(n))$ operacji.