Algorytmy i struktury danych

Zadanie 9, lista 2

Dawid Żywczak

24 kwietnia 2020

Do rozwiązania problemu postawionego w zadaniu skorzystamy z kodowania Huffmana, gdzie symbolami będą nasze liście, natomiast częstości występowania to wartości $w_1, w_2, ..., w_n$. Przedstawię teraz algorytm, a następnie lematy, które pomogą w udowodnieniu poprawności rozwiązania. Algorytm wygląda następująco:

$Q \leftarrow$ kolejka priorytetowa liści
for $i = 1$ **to** $i = n - 1$ **do**
  stwórz nowy węzeł v
  ustaw lewe dziecko v na deletemin(Q)
  ustaw prawe dziecko v na deletemin(Q)
  przypisz wagę v jako suma wag jego dzieci
  dodaj v do Q
end for

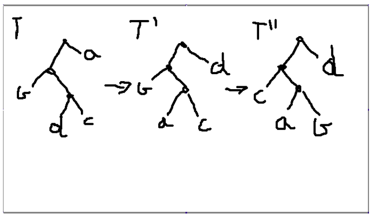
(niżej indeks 1 oznacza drzewo T, indeks 2 oznacza drzewo T')

Lemat 1. *Dowolny wierzchołek w optymalnym drzewie, rozwiązującym problem z zadania, posiada dwóch synów lub jest liściem.*

Dowód. Załóżmy niewprost, że istnieje drzewo optymalne T, które posiada wierzchołek mający tylko jednego syna, nazwijmy ten wierzchołek v. Rozważmy drzewo T' powstałe przez usunięcie wierzchołka v i "podłączenie" jego jedynego poddrzewa w jego miejsce. Teraz zauważmy, że w wyniku usunięcia wierzchołka v, $EL(T) > EL(T')$, a zatem T nie jest optymalnym rozwiązaniem, sprzeczność. □

Lemat 2. *Jeśli wierzchołki a i b są liśćmi o najmniejszej wadze, to istnieje drzewo optymalne T, w którym a i b są braćmi.*

Dowód. Weźmy optymalne drzewo T, załóżmy że liście c i d są braćmi. Niech L będzie zbiorem liści drzewa T. Pokażemy, że możemy zamienić drzewo T na inne, lepsze drzewo T' w którym a i b są braćmi.



Bez starty ogólności załóżmy, że $c(d) \leq c(c)$ i $c(a) \leq c(b)$. Oczywiście też $c(a) \leq c(d)$ i $c(b) \leq c(c)$. Prześledźmy jak działa przejście z T do T'. Zamieniami wierzchołek d z wierzchołkiem c miejscami, w wyniku czego otrzymujemy drzewo o mniejszej wadze EL. Zobaczmy dlaczego.

$$EL(T) - EL(T') = \sum_{v \in L} c(v) \cdot d_1(v) - \sum_{v \in L'} c(v) - d_2(v) =$$
$$c(a)d_1(a) + c(d)d_1(d) - c(a)d_2(a) - c(d)d_2(d) =$$
$$c(a)d_1(a) + c(d)d_1(d) - c(a)d_1(d) - c(d)d_1(a) = (c(d) - c(a))(d_1(d) - d_1(a)) \geq 0$$

Analogicznie możemy pokazać dla przejścia T' w T'', a skoro T jest optymalne to T' też. □

Lemat 3. *Niech a i b będą liśćmi o najmniejszej wadze. Niech d będzie nowym wierzchołkiem o $c(d) = c(a) + c(b)$ i niech $L' = L \setminus \{a, b\} \cup \{d\}$ (d wstawiamy w miejsce ojca a i b). Wtedy jeśli T' jest optymalnym drzewem dla L' to T jest optymalne dla L.*

Dowód. Zauważmy najpierw, że wagi wierzchołków których nie zmieniami są takie same w drzewie T i T'. Zauważmy też, że $d_1(a) = d_1(b) = d_2(d) + 1$. Wtedy

$$c(a)d_1(a) + c(b)d_1(b) = (c(a) + c(b))(d_2(d) + 1) = c(a) + c(b) + (c(a) + c(b))d_2(d) =$$
$$c(a) + c(b) + c(d)d_2(d).$$

Czyli $EL(T) = EL(T') + c(a) + c(b)$. Załóżmy, że T' jest optymalne, ale T nie jest. Istnieje zatem drzewo G t.że $EL(G) < EL(T)$ (T i G mają ten sam zbiór liści). Zamieńmy teraz parę liści a i b o tym samym ojcu (wiem, że taka para istnieje z lematu 1.) na wierzchołek d, taki że $c(d) = c(a) + c(b)$. Korzystając z równania znajdującego się wyżej, wiem że otrzymujemy drzewo o wadze $EL(G) - c(a) - c(b) < EL(T) - c(a) - c(b) = EL(T')$ co jest sprzeczne z optymalnością T'. □

Uzbrojeni w te wszystkie lematy, możemy przejść do dowodu poprawności algorytmu. Zrobimy to indukcyjnie po mocy zbioru liści.

Dowód. Baza: 1 liść - z lematu 1. drzewo jednoelementowe 2 liście - oczywiste Krok: Załóżmy, że nasz algorytm znajduje optymalne rozwiązanie dla $\|L\| \leq n$. Weźmy zbiór L' o mocy $n + 1$. Wybierzmy dwa wierzchołki o najmniejszej wadze, będące braćmi oraz nazwijmy je a i b. Zastąpmy je wierzchołkiem d tak jak w lemacie 3. Otrzymujemy zbiór liści o mocy n więc z założenia potrafimy znaleźć dla niego rozwiązanie optymalne T'. Teraz niech a i b będą synami wierzchołka d. Wtedy z lematu drugiego wiemy, że drzewo otrzymane w ten sposób jest optymalnym rozwiązaniem dla zbioru L'. □

Złożoność czasowa: Budujemy na początku kolejkę, co zajmuje nam O(n), później w każdym przebiegu pętli usuwamy dwa razy min oraz wrzucamy jeden element na kopiec. Otrzymujemy zatem O(nlogn).
Złożoność pamięciowa: O(n) - musimy trzymać kolejkę w pamięci