Zadanie 9 z listy 2 AiSD
(Kodowanie Huffmana)

Bartosz Brzoza

Marzec 2020

1 Sformułowanie zadania

Dla danych wag W naleźć takie drzewo binarne T z tymi wagami na liściach, minimalizujące wielkość

$$EL_W(T) = \sum_{w \in W} w * d_T(w),$$

gdzie $d_T(w)$ jest głębokością w w drzewie T.

2 Algorytm

W jest multizbiorem wag.

Algorithm 1 $Huffman(W)$

if $|W| = 1$ **then**
  return w t.że $w \in W$
else
  $a \leftarrow min(W)$
  $b \leftarrow min(W - \{a\})$
  $W' \leftarrow W \cup \{a + b\} - \{a, b\}$
  $T' \leftarrow Huffman(W')$
  $T \leftarrow T'$ z dołączonymi a i b jako synowie $a + b$
  return T
end if

W dalszej części zostanie zaprezentowany pseudokod do efektywnej implementacji.

3 Dowód optymalności

3.1 Lemat 1.

Nie istnieje drzewo optymalne, w którym jakiś liść x nie ma brata. Dowód nie wprost: Załóżmy, że takie drzewo T istnieje. Niech T' będzie takim samym drzewem, tylko że z x w miejscu swojego ojca.

$$EL_W(T) - EL_W(T') = xd_T(x) - xd_{T'}(x) = x$$

Zatem drzewo T nie jest optymalne - sprzeczność.

3.2 Lemat 2.

Załóżmy, że T i T' są drzewami binarnymi takimi, że T' otrzymujemy przez zamienienie miejscami liści x, y w drzewie T. Wtedy $EL_W(T') - EL_W(T) = (x - y)(d_T(y) - d_T(x))$
Dowód:

$$EL_W(T') - EL_W(T) = \sum_{w \in W} w * d_{T'}(w) - \sum_{w \in W} w * d_T(w) =$$

$$xd_T(y) + yd_T(x) - xd_T(x) - yd_T(y) = (x - y)(d_T(y) - d_T(x))$$

3.3 Lemat 3.

Dla każdego W istnieje takie drzewo T, że jest optymalne, oraz dwie najmniejsze wartości a i b są w nim braćmi.
Dowód: Weźmy dowolne drzewo optymalne T^* oraz dowolne rodzeństwo x, y z najgłębszego poziomu. Zamieniając a z x oraz b z y miejscami otrzymujemy drzewo T.

$$EL_W(T) - EL_W(T^*) = (x - a)(d_{T^*}(a) - d_{T^*}(x)) + (y - b)(d_{T^*}(b) - d_{T^*}(y)) \leqslant 0,$$

gdyż $x \geqslant a$, $d_{T^*}(a) \leqslant d_{T^*}(x)$, $y \geqslant b$ oraz $d_{T^*}(b) \leqslant d_{T^*}(y)$. Zatem drzewo T jest optymalne oraz a i b są w nim braćmi.

3.4 Właściwy dowód

Dowód przez indukcję po rozmiarze W.
Baza: Algorytm dla $|W| = 1$ zwraca optymalne drzewo - ok
Krok: Załóżmy, że $Huffman(W)$ zwraca optymalne drzewo dla każdego W t.że $|W| < n$
Weźmy dowolne W o mocy n. Rozważamy co zrobi algorytm Huffmana (notacja zgodna z algorytmem zaprezentowanym w drugim punkcie).

$$EL_W(T) = \sum_{w \in W - \{a,b\}} w * d_T(w) + a * d_T(a) + b * d_T(b) =$$

$$\sum_{w \in W - \{a,b\}} w * d_T(w) + (a+b)(d_{T'}(a+b)+1) =$$

$$\sum_{w \in W'} w * d_{T'} + a + b = EL_{W'}(T') + a + b$$

Załóżmy niewprost, że T nie jest optymalnym drzewem, za to Z jest optymalne i ma a, b jako bracia. Niech Z' jest takim drzewem jak Z ze złączonymi liśćmi a, b. Analogicznie do powyższego argumentu $EL_W(Z) = EL_{W'}(Z') + a + b$. Wtedy $EL_{W'}(T') = EL_W(T) - a - b > EL_W(Z) - a - b = EL_{W'}(Z')$ co przeczy optymalności T'. Zatem T jest optymalnym drzewem.
■

4 Efektywna implementacja

Wykorzystuje kopiec.

Algorithm 2 $Huffman(W)$

$H \leftarrow make\_heap(W)$ {w kopcu liczby $1, ..., n$, porządek na podstawie W}
for $p = n+1, ..., 2n-1$ **do**
  $a \leftarrow H.pop\_min()$
  $b \leftarrow H.pop\_min()$
  $H.push(p)$
  $W[p] \leftarrow W[a] + W[b]$
  $lchild[p] \leftarrow a$
  $rchild[p] \leftarrow b$
end for
return $2n-1$ {to jest korzeń drzewa}

4.1 Złożoność

Tworzymy kopiec w $O(n)$.
Następnie $(n-1)$-krotnie wykonujemy $2 \times H.pop\_min() + H.push()$ (w czasie $logn$). Zatem złożoność to $O(nlogn)$.

5 Alternatywna implementacja - zadanie 12

Sortuje wagi i wykorzystuje dwie kolejki.

Algorithm 3 $Huffman(W)$

> $QA \leftarrow queue(sort(W))$ {sortowanie na podstawie W}
> $QB \leftarrow queue()$
> **for** $p = n+1, ..., 2n-1$ **do**
> $a \leftarrow pop\_min(QA.first(), QB.first())$ {popujemy z tej kolejki, gdzie jest mniejsza wartość}
> $b \leftarrow pop\_min(QA.first(), QB.first())$ {popujemy z tej kolejki, gdzie jest mniejsza wartość}
>   **if** $QA.empty()$ **then**
> $swap(QA, QB)$ {jeżeli QA jest puste - zamieniamy QA z QB - tylko wskaźniki oczywiście}
>   **end if**
> $QB.push(p)$
> $W[p] \leftarrow W[a] + W[b]$
> $lchild[p] \leftarrow a$
> $rchild[p] \leftarrow b$
> **end for**
> **return** $2n-1$ {to jest korzeń drzewa}

5.1 Dowód poprawności

Elementy w kolejkach QA i QB są zawsze w kolejności rosnącej.
Dowód indukcyjny po ilości kroków:
Baza: Na początku kolejka QA jest posortowana, a QB jest pusta - ok.
Krok: Załóżmy że w poprzednim kroku elementy kolejek były w kolejności rosnącej oraz usunęliśmy a, b i dołożyliśmy $a+b$. Wybieramy 2 najmniejsze elementy c, d z tych kolejek - skoro kolejki były posortowane, to $a \leqslant b \leqslant c \leqslant d$, czyli też $a+b \leqslant c+d$.

a) dodajemy $c+d$ na tą samą kolejkę co $a+b$, ale $a+b \leqslant c+d$, więc porządek jest nadal utrzymany.

b) dodajemy $c+d$ na pustą kolejkę, czyli porządek jest nadal utrzymany

Zatem ta implementacja realizuje ten sam algorytm co w punkcie 2.

5.2 Złożoność alternatywnej implementacji

Sortujemy w czasie $nlogn$. Następnie $(n-1)$ krotnie wykonujemy stałe obliczenia. Zatem złożoność to $O(nlogn)$.
Jeżeli dane na wejściu są już posortowane - pomijamy sortowanie i wtedy złożoność to $O(n)$.