Algorytmy i Struktury Danych

Lista 3 Zadanie 10

1 Treść zadania

Niech $A = (a_1, \ldots, a_n)$ będzie posortowaną tablicą. Zbuduj algorytm sprawdzający czy w tablicy występuje element $a_i = i$ wykonujący $\mathcal{O}(\log n)$ porównań.

2 Rozwiązanie

2.1 Zbiór bez duplikatów

Na początku załóżmy, że w tablicy A nie występują dwa elementy takie, że $a_i = a_j, i \neq j$. Wtedy sprawdzenie warunku podanego w zadaniu sprowadza się do wykonania lekko zmodyfikowanego algorytmu przeszukiwania binarnego (pseudokod poniżej).

Algorytm 1 Przeszukiwanie binarne (zmodyfikowane)

```
1:  function binarySearchMod(A, start, end)
2:      if start ⩽ end then
3:          mid = (start + end) ÷ 2                    ▷ ÷ - dzielenie całkowitoliczbowe
4:          if mid = A[mid] then
5:              return true
6:          else if mid > A[mid] then
7:              return binarySearchMod(A, mid + 1, end)
8:          else
9:              return binarySearchMod(A, start, mid − 1)
10:         end if
11:     end if
12:     return false
13: end function
14: function magicIndex(A)
15:     return binarySearchMod(A, 0, len[A] − 1)
16: end function
```

Algorytm ten jest oparty o założenie, że elementy w tablicy są w relacji takiej, że $\forall i \in \{1, \ldots n-1\}, \quad a_i < a_{i+1}$, dlatego właśnie warunek z linii 6 algorytmu pozwala stwierdzić, że skoro indeks jest większy od wartości danego elementu, to oznacza, że musiało tak być również dla wszystkich poprzednich elementów w tablicy, dlatego musimy kontynuować poszukiwania w drugiej połowie zbioru. Analogicznie działa warunek w drugą stronę. W ten sposób dzielimy zbiór na połowy, aż znajdziemy interesujący nas tzw. "magiczny indeks". Zgodnie ze specyfikacją algorytmu przeszukiwania binarnego, ma on złożoność $\mathcal{O}(\log n)$.

2.2 Zbiór z duplikatami

W przypadku, gdy założymy, że elementy w tablicy mogą się powtarzać, warunek z poprzedniego zadania przestaje być wystarczający, aby stwierdzić, że "magiczny indeksńie wystąpił wcześniej ani później. W tym przypadku wiemy jedynie, że jeśli wartość elementu jest większa od jego indeksu to następna taka sytuacja, że $a_i = i$ może wystąpić dopiero na indeksie o tej wartości. Analogicznie, gdy wartość ta jest mniejsza. Dlatego tutaj wykorzystamy również algorytm przeszukiwania binarnego, lecz ze zmodyfikowanym sposobem wyboru podzbiorów.

Algorytm 2 Przeszukiwanie binarne (z duplikatami)

```
function BINARYSEARCHWITHDUPLICATES(A, start, end)
    if start ⩽ end then
        mid = (start + end) ÷ 2                          ▷ ÷ - dzielenie całkowitoliczbowe
        if mid = A[mid] then
            return mid
        end if
        leftIndex = min{mid − 1, A[mid]}
        left = BINARYSEARCHWITHDUPLICATES(A, start, leftIndex)
        if left ⩾ 0 then
            return left
        else
            rightIndex = max{mid + 1, A[mid]}
            return BINARYSEARCHWITHDUPLICATES(A, rightIndex, end)
        end if
    end if
    return −1
end function
function MAGICINDEXWITHDUPLICATES(A)
    return BINARYSEARCHWITHDUPLICATES(A, 0, len[A] − 1) ⩾ 0
end function
```

W ten sposób przeszukujemy zawsze najpierw lewy przedział, a w przypadku, gdy nie znajdziemy tam takiego indeksu, jesteśmy zmuszeni przeszukać również prawy. Złożoność tego algorytmu jest w najlepszym przypadku logarytmiczna, w najgorszym - liniowa.