Ćwiczenia 10

7.1 W tym zadaniu badamy implementację słownika w tablicy $a[1..N]$, gdzie N jest dodatnią liczbą całkowitą i wiadomo, że rozmiar słownika n nigdy jej nie przekroczy. Przyjmijmy, że elementami słownika są liczby całkowite.

(a) Załóżmy, że elementy słownika umieszczamy na przekątnych dwuwymiarowej nieskończonej tablicy $A[1..,1..]$ w kolejności $A[1,1]$, $A[2,1]$, $A[1,2]$, $A[3,1]$, $A[2,2]$ itd. Dodatkowo przyjmijmy, że elementy w tablicy są uporządkowane rosnąco w wierszach (z lewa na prawo) i rosnąco w kolumnach (z góry do dołu).
Zaproponuj implementację operacji słownikowych Search, Insert, Delete, z których każda działa w czasie $O(\sqrt{n})$. Pokaż, w jaki sposób ukryć implementację dwuwymiarową w tablicy jednowymiarowej, bez strat na czasach wykonywania poszczególnych operacji słownikowych.

(b) Załóżmy teraz, że elementy słownika zapisano w tablicy $a[1..n]$ (prefiks tablicy $a[1..N]$) w blokach B_1, B_2, ..., B_i, ... (aż do wyczerpania wszystkich n elementów) o następujących własnościach:
- blok B_i zajmuje spójny fragment tablicy a od miejsca $i(i-1)/2+1$ do miejsca $\min(i(i-1)/2+i, n)$,
- elementy w bloku są uporządkowane rosnąco i (być może) przesunięte cyklicznie,
- dla każdego $i > 1$, każdy element w bloku B_i jest większy od każdego elementu w bloku B_{i-1}.

Zaprojektuj wydajne operacje słownikowe przy tej organizacji danych.

7.2 Danych jest n par liczb całkowitych, które się różnią na każdej pozycji. Pierwsze elementy par to klucze, zaś drugie to priorytety. Innymi słowy, mamy n różnych kluczy i n różnych priorytetów.

(a) Wykaż, że istnieje dokładnie jedno drzewo binarne, które jest drzewem BST ze względu na klucze i jednocześnie kopcem typu MAX ze względu na priorytety.

(b) Niech T będzie drzewem BST ze względu na klucze. Opisz konstrukcję ciągu rotacji o długości $O(n)$, które należy wykonać, żeby przekształcić drzewo T w drzewo BST T', które będzie jednocześnie kopcem binarnym typu MAX ze względu na priorytety.

(c) Zaproponuj wydajny algorytm, który znajdzie ciąg rotacji, o którym mowa w poprzednim punkcie.

7.4 Zaprojektuj strukturę danych dla dynamicznego zbioru domkniętych przedziałów liczbowych S, umożliwiającą wydajne wykonywanie operacji:

- Search$(S, [a, b])$:: podaj wskaźnik do wystąpienia $[a, b]$ w S; jeśli $[a, b]$ nie ma w S, odpowiedzią jest NULL,
- Insert$(S, [a, b])$:: $S := S \cup \{[a, b]\}$,
- Delete$(S, [a, b])$:: $S := S \setminus \{[a, b]\}$,
- Intersect$(S, [a, b])$:: sprawdź, czy S ma niepuste przecięcie z $[a, b]$.

Do samodzielnej pracy

7.5 Zaprojektuj strukturę danych dla dynamicznego ciągu liczbowego x_1, x_2, $\ldots$, x_n umożliwiającą wydajne wykonywanie następujących operacji:

- Ini():: zainicjuj ciąg jako pusty,
- Delete(i):: usuń i-ty element ciągu,
- Insert(i, a):: wstaw liczbę a jako i-ty element ciągu,
- Find(i):: wskaż i-ty element ciągu,
- ParitySum():: podaj sumę wszystkich elementów na pozycjach parzystych w ciągu.