Algorytmy i Struktury Danych, 7. ćwiczenia

2008-11-24

1 Plan zajęć

- Ukryte słowniki (search, insert, delete w czasie $\sqrt{n}$; alternatywnie wyszukiwanie w $O(\log n)$ wstawianie, usuwanie $O(\sqrt{n} * \log n)$)
- 2-3 drzewa, dodatkowe operacje join i split.
- Pokazać, że przy pomocy rotacji można zawsze przejść z jednego BST do drugiego.

2 Ukryty słownik

(BKR) *Cyklicznym blokiem* nazywamy dowolną tablicę $S[i, \ldots, j]$, której wartości spełniają następujący warunek:
istnieje indeks k, $i \leq k \leq j$, taki, że:

$$S[k] < S[k+1] < \ldots < S[j] < S[i] < \ldots < S[k-1]$$

Tablicę $S[1..n]$ o wartościach ze zbioru L nazywamy *tablicą cyklicznych bloków*, jeśli spełnione są następujące dwa warunki:

- każdy blok jest cyklicznym blokiem,
- wszystkie elementy bloku $i-1$ są mniejsze niż wszystkie elementy bloku i dla $2 \leq i \leq h$, gdzie h jest maksymalnym numerem bloku.

Wykaż, że używając struktury danych tablic cyklicznych bloków operacje *access* można wykonać za pomocą $2 \log n$ porównań, a operacje *insert* i *delete* za pomocą $O(\sqrt{n} \log n)$ instrukcji (n oznacza liczbę elementów w zbiorze S).

- organizujemy tak, tablicę, żeby i–ty blok miał długość i (ewentualnie oprócz ostatniego).
- *access* — wyszukaj binarnie odpowiedni blok, w bloku wyszukujemy przesuniecie k, i binarnie wyszukujemy wewnątrz bloku,
- *insert* — wyszukaj odpowiedni blok ($\log n$), jeśli to nie jest ostatni blok, to trzeba zrobić miejsce na nowy elementy — wyrzucając największy, największy element trzeba przesunąć do następnego bloku ($\sqrt{n}$), z którego wypada kolejny element, itd. ($\sqrt{n}$ razy, ponieważ nie trzeba już nic przesuwać jedna faza kosztuje $\log n$).
- *delete* — podobnie.

3 2-3 drzewa

2-3 drzewa to drzewa zrównoważone o następujących własnościach:

- każdy węzeł przechowuje 1 lub 2 klucze,
- każdy węzeł wewnętrzny (oprócz korzenia) ma 2 lub 3 synów,
- wszystkie liście mają tą samą głębokość,
- zachowany jest porządek kluczy w poddrzewach (mniej więcej jak w drzewach BST),

Wstawienia do drzewa:

- znajdujemy liść w którym powinien znaleźć się nowy klucz, dodajemy go do węzła,
- może się okazać, że węzeł posiada chwilowo 3 klucze, w takim wypadku dzielimy go na dwie części (w każdej z nich zostawiamy jeden klucz) a trzeci klucz przekazujemy poziom wyżej,
- tak długo jak drzewo zawiera węzeł z 3 kluczami powtarzamy tą procedurę.

Usuwanie z drzewa:

- jeśli klucz leży w węźle wewnętrznym, to zamieniamy go z następnikiem (lub poprzednikiem), i redukujemy problem do usuwania z liścia,
- jeśli klucz leży w liściu, to usuwamy go z węzła,
- jeśli okazuje się, że jakiś węzeł zawiera 0 kluczy, to poprawiamy drzewo idąc od tego węzła do korzenia i próbując zastąpić puste miejsce kluczem pożyczonym od sąsiadów,

Scalanie (zakładamy, że klucze są przetrzymywane w węzłach):

- dane są drzewa T_1, T_2,
- usuwamy z T_2 najmniejszy klucz x,
- znajdujemy w T_2 węzeł v którego poddrzewo ma wysokość $|T_2|$,
- dodajemy do v klucz x z prawym poddrzewem T_2,
- jeśli v zawiera 3 klucze, to poprawiamy warunku 2-3 drzewa idąc od v do korzenia.

Złożoność $O(\log(|T_1| + |T_2|))$.

Scalanie (zakładamy, że klucze są przetrzymywane w liściach):

- jeśli $h_1 = h_2$, to tworzymy trzewo T z wartością węzła $max(T_1.root)$, lewym poddrzewem T_1, prawym poddrzewem T_2,
- jeśli $h_1 > h_2$, to znajdujemy na skrajnie prawej ścieżce T_1, węzeł v o wysokości $h_2 + 1$, dodajemy do v nową wartość z węzła $min(T_2.root)$ i podłączamy T_2 jako skrajnie prawe poddrzewo, jeśli w wyniku tej operacji węzeł v ma 3 wartości, to poprawiamy drzewo idąć od v do korzenia,

- jeśli $h_1 < h_2$, to postępujemy analogicznie jak w poprzednim przypadku (ale dodajemy T_1 do T_2 i szukamy węzła na skrajnie lewej ścieżce).

Złożoność $O(1 + |h_1 - h_2|) = O(\log(|T_1| + |T_2|))$.

Split (zakładamy, że klucze są przetrzymywane w liściach) – podział drzewa T na dwa poddrzewa T_1 (z kluczamy $\leq x$) i T_2 (z kluczamy $> x$):

- szukamy ścieżki od korzenia do liścia w którym należałoby wstawić klucz x, węzły na tej ścieżce oznaczamy przez $v_0 = root, v_1, \ldots, v_k$,
- likwidujemy węzły $v_0, \ldots, v_{k-1}$, a węzeł v_k rozbijamy na dwa węzły (jeden z wartościami $\leq x$, drugi z wartościami $> x$),
- drzewo rozpada się na dwa lasy $F_{\leq}$ (drzewa z wartościami $\leq x$), i $F_{>}$ (drzewa z wartościami $> x$),
- scalamy drzewa $F_{\leq}$ wg. rosnących wysokości otrzymując T_1,
- scalamy drzewa $F_{>}$ wg. rosnących wysokości otrzymując T_2,

4 Pokazać, że przy pomocy rotacji można zawsze przejść z jednego BST do drugiego

Za pomocą rotacji możemy dowolne drzewo BST zamienić na listę (i na odwrót). Wstarczy tak długo jak drzewo zawiera węzeł z lewym synem, wykonujemy na nim (i lewym synie) prawą rotację.