Algorytmy i struktury danych

Laboratorium 4

Termin wysłania (MS Teams): **10 maja 2021 godz. 07:29**

Zadanie 1. [75%]

Napisz program, który symuluje działanie wybranych struktur danych przechowujących ciągi znaków (przyjmujemy porządek leksykograficzny). Program powinien przyjmować jako parametr wejściowy typ struktury:

[15%] --type bst drzewo BST;

[30%] --type rbt drzewo czerwono-czarne, patrz np. rozdział 13 w [1];

[30%] --type splay samoorganizujące drzewo binarne (*splay tree*), patrz np. [2] lub rozdział 4.3 w [3].

Każda ze struktur powinna udostępniać przynajmniej poniższe funkcjonalności podawane na standardowym wejściu:

- **insert** s – wstaw do struktury ciąg s (jeśli na początku lub końcu ciągu znajduje się znak spoza klasy `[a-zA-Z]`, to znak ten jest usuwany);
- **load** f – dla każdego oddzielonego białym znakiem lub znakiem interpunkcyjnym wyrazu z pliku f wykonaj operację insert lub zwróć informację o nieistniejącym pliku;
- **delete** s – jeśli struktura nie jest pusta i dana wartość s istnieje, to usuń element s;
- **find** s – sprawdź, czy w strukturze przechowywana jest wartość s; jeśli tak to wypisz 1, w p. p. wypisz 0;
- **min** – wypisz najmniejszy element znajdujący się w strukturze lub – dla struktur pustych – pustą linię;
- **max** – wypisz największy element znajdujący się w strukturze lub – dla struktur pustych – pustą linię;
- **successor** k – wypisz następnik elementu k lub – jeśli on nie istnieje (np. struktura nie zawiera k, k nie ma następników) – pustą linię;
- **inorder** – wypisz elementy drzewa w posortowanej kolejności (od elementu najmniejszego do największego) lub – dla struktur pustych – pustą linię.

Wynik powinien być wypisywany na standardowe wyjście, a na standardowym wyjściu błędów powinny być wypisywane w kolejności: czas działania całego programu, liczba operacji każdego typu, maksymalna liczba elementów (maksymalne zapełnienie struktury w czasie działania programu), końcowa liczba elementów w strukturze oraz całkowita liczba porównań między elementami.

Zaplanuj i wykonaj eksperymenty pozwalające oszacować średni czas działania każdej z operacji. Prezentując rozwiązanie, omów uzyskane wyniki przeprowadzonych eksperymentów.

Wejście Wejście składa się z $n + 1$ linii. W pierwszej znajduje się liczba n określająca liczbę wykonywanych operacji, w liniach od 2 do $n + 1$ znajdują się kolejne operacje zgodnie z ich specyfikacją. Program może wykorzystywać więcej niż jeden wątek, jednak operacje muszą być wykonane w zadanej kolejności. Długość pojedynczego ciągu znaków nie przekracza 100, natomiast $n + 1$ nie przekracza zakresu Integera.

Wyjście Wyjście składa się z $k \leqslant n$ linii będących wynikami kolejnych operacji podanych na wejściu.

Przykład Przykładowe wywołanie

```
./trees --type rbt < ./input > out.res
```

input	**out.res**
17	
max	a aaa ab b
insert aaa	ab
insert a	1
insert b	1
insert ab	1
inorder	0
delete a	aaa
delete b	
max	
load sample.txt	
find three	
delete three	
find three	
find Three	
delete Three	
find Three	
min	

Zadanie 2. [25%]

Wykonaj eksperymenty, które pozwolą postawić tezę na temat ograniczenia dolnego i górnego oraz średniej dla liczby porównań między elementami wykonywanych przez procedurę **find** w każdej ze struktur. Testy wykonaj na posortowanej liście unikatowych ciągów (np. `aspell_wordlist.txt`) oraz na jej losowych permutacjach, a także na takiej, gdzie możliwe są powtórzenia (np. `lotr.txt`, `KJB.txt`).[1]

Podczas prezentacji rozwiązania przedstaw wyniki eksperymentów oraz wyciągnięte wnioski. Wskaż przykłady danych, które pozwalają każdej z tych struktur być najlepszą oraz uzasadnij, dlaczego takie dane powodują ten efekt. Krótko omów słabości i zalety tych trzech typów binarnych drzew poszukiwań.

Zadanie 3.* [dodatkowe – 30%] (termin wysyłania: 17 maja 2021 godz. 07:29)

Uzupełnij **Zadanie 1.** oraz **Zadanie 2.** o tablice hashujące (**--type hmap**; patrz np. rozdział 11 w [1]) z metodą łańcuchową dla przechowywanych w jednej komórce danych długości mniejszej niż n_t oraz z wykorzystaniem zbalansowanych drzew binarnych (np. drzew czerwono-czarnych) dla przechowywanych w jednej komórce danych o długości większej niż n_t. Zaimplementuj metody **insert**, **load**, **delete** oraz **find**. Przeprowadź testy mające na celu oszacowanie n_t, dla którego zysk w czasie dostępu do elementu uzasadnia nadkład wykonywanych operacji balansujących. Dobierz liczbę komórek m odpowiednio do wybranej funkcji hashującej.

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[2] Daniel D. Sleator and Robert E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.

[3] Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, USA, 1983.

[1] Na stronie Project Gutenberg możesz znaleźć wiele innych książek w formacie `.txt`.