Minikurs języka C

Lista zadań nr 4

Na zajęcia 22 kwietnia 2020
grupa AKu

Za każde zadanie można otrzymać od 0 do 10 punktów.

Zadanie 1. Drzewo poszukiwań binarnych (Binary Search Tree) jest strukturą która pozwala przechowywać elementy z uporządkowanego zbioru, obsługuje następujące operacje:

- dodanie elementu do zbioru,
- usunięcie elementu ze zbioru,
- wyszukanie elementu w zbiorze.

W skrócie, w drzewie binarnym każdy węzeł drzewa zawiera jakiś element ze zbioru. Dodatkowo drzewo spełnia własność BST — jeśli w jakimś węźle v jest element x, to potomkowie v zawierający el. mniejsze lub równe są w lewym poddrzewie, a większe w prawym. Przykład możesz znaleźć tutaj: `https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84`. Twoim zadaniem jest zaimplementowanie drzewa BST o potencjalnie nieograniczonym rozmiarze. Węzeł takiego drzewa powinien być następującą strukturą:

```
struct node {
    int value;
    struct node* left;
    struct node* right;
};
```

Ponadto napisz funkcje:

- zwracającą wysokość drzewa,
- wypisującą wszystkie elementy w drzewie w porządku inorder (a więc posortowane)

Powyższą strukturę danych zamknij w module, tzn. dostarcz plik nagłówkowy `bst.h` z deklaracjami odpowiednich funkcji, oraz plik `bst.c` z implementacją. Następnie

- przetestuj swoje drzewo BST dostarczając program `main.c` korzystający ze zdefiniowanego przez Ciebie modułu. Zadbaj o poprawną obsługę przypadków brzegowych, np. próbę usunięcia z- oraz wyszukania elementu w- pustym drzewie. Drzewo po opróżnieniu musi nadawać się do ponownego użycia, np. sekwencja operacji: dodanie elementu do początkowo pustego drzewa, usunięcie tego elementu, dodanie innego musi dać poprawne jednoelementowe drzewo.
- w osobnym pliku źródłowym zakoduj eksperymenty: a) dodaj do drzewa 106 losowych wartości i zwróć jego głębokość, b) posortuj 106 losowych wartości (używając wstawiania do drzewa i przejścia metodą inorder).
- dostarcz skrypt `compile.sh` kompilujący powyższe programy wywołaniem `gcc`.

Uwaga: Zadanie bez operacji usuwania warte jest 7 punktów.

Zadanie 2. Przypomnij sobie Zadanie 2 z Listy 1 (to z kolejką FIFO). Tym razem w kolejce będziemy chcieli przetrzymywać następujące dane:

- napisy(tablice typu `char`),
- liczby całkowite,
- liczby zmiennoprzecinkowe,
- tablice liczb całkowitych.

Celem zadania jest stworzenie kolejki FIFO takiej jak w Zad. 2 z Listy 1, jednak tym razem nie chcielibyśmy marnować za dużo pamięci, bardziej precyzyjnie to na każdy dodany element chcemy zużywać co najwyżej stałą liczbę dodatkowych bitów.

W tym celu kolejka ma być zrealizowana jako lista łączona, a każdy element listy ma być następującą strukturą:

```
struct node{
    int data_type;
    size_t allocated_size;
    void* data;
    struct node* next_node;
}
```

Gdzie `data_type` oznacza stałą odpowiadającą typowi danych przechowywany w danym miejscu (np. dla napisów może być to 1, dla liczb całkowitych 2 itd.), a `allocated_size` oznacza ile bajtów zarezerwowaliśmy (tzn. rozmiar pamięci wskazywanej przez `data`). Zauważ, że to są wszystkie informacje niezbędne do sprawdzenia jaki typ danych jest przechowywany i, w przypadku tablicy, ile elementów jest w niej przechowywanych.

Samodzielnie doprecyzuj szczegóły, kod dostarcz w postaci modułu (plik nagłówkowy + plik z implementacją) obsługującego kolejkę FIFO oraz programu testującego ten moduł. Pamiętaj, by podczas wstawiania elementu do kolejki dokonywać faktycznego kopiowania (a zatem również alokacji pamięci przy pomocy `malloc` i przypisania jej do zmiennej `data`) a nie tylko przypisania do data wskaźnika na argument. Podobnie, przy usuwaniu elementu z kolejki należy zwolnić pamięć przypisaną do wskaźnika `data`.

Uwaga: Jeśli chcesz to zamiast kolejki FIFO możesz zaimplementować kolejkę LIFO, czyli stos.

Zadanie 3. Pojawi się w systemie SKOS.