Projekt 4

Temat: Binarne drzewa wyszukiwań

Celem tego projektu jest implementacja binarnego drzewa wyszukiwań (BST).

Wymagania organizacyjne

- Węzły drzewa zaimplementowane są jako klasa `BSTNode`, samo drzewo zaimplementowane jest jako klasa `BST`. Klasa `BSTNode` moze być klasa zagnieżdżoną w klasie `BST`. Obie klasy umieszczone są w przestrzeni nazw `AiSD`.
- Kod źródłowy umieszczony jest w plikach `BST.h/.cpp`
- Do implementacji musi być dołączone:
  - minimum dziesięć zbiorów testowych umieszczonych w plikach o nazwach `set_XX.txt`. Mile widziane jest:
    - zróżnicowanie wielkości zbiorów testowych
    - zestawy danych, które powodują powstanie drzewa o szczególnym kształcie.
    - demonstracja przypadku pesymistycznego/optymistycznego
  - wyniki działania programu dla zbiorów testowych umieszczonych w plikach o nazwach `set_XX_output.txt`
  - sprawozdanie zawierajace:
    - wyliczenie spełnionych wymagań zgodnie z poniższą numeracją
    - do każdego spełnionego wymagania należy opisać gdzie i jak zostało ono spełnione, np.: wskazując miejsce w kodzie + opis
  - program demonstrujący każde spełnione wymaganie:
    - w sprawozdaniu należy jasno wskazać w jaki sposób program testujący wykonuje demonstrację;
    - jeżeli jest to wygodne, można (trzeba) łączyć demonstracje dla różnych wymagań
    - w sprawozdaniu należy podać sposób uruchomienia, ewentualnej interakcji z użytkownikiem, tworzone pliki wyjściowe etc.
    - patrz też wymagania podstawowe
  - kod programu testujacego powinien być w pliku o nazwie: `projekt_4_<inicjaly_TL>_<inicjaly_>_demo.cpp`,

Wymagania podstawowe (muszą być spełnione)

1. Klucze drzewa są typu `key_t = unsigned short`. Klucze w drzewie nie mogą się powtarzać.

2. Węzły zawierają następujące pola:

   - key: klucz wyszukiwania typu `key_t`
   - `parent`, `left`, `right`: wskaźniki na rodzica, lewego i prawego syna

```
// BST.h
namespace AiSD{
class BSTNode{
...
  key_t key;
  BSTNode* parent;
  BSTNode* left;
  BSTNode* right;
...
};

class BST{
    // details
};
} //namespace AiSD
```

3. Dostępne są następujace operacje:

   - `void Insert(const key_t k)`: wstawia węzeł z kluczem k
   - `void Delete(const key_t k)`: usuwa węzeł z kluczem k
   - `BSTNode* Search(const key_t k)`: wyszukuje węzeł o kluczu k, zwraca wskaźnik do tego węzła albo `nullptr`, gdy nie znaleziono
   - `void Clear()`: usuwa wszystkie węzły z drzewa
   - `BSTNode* Min()`, `BSTNode* Max()`: zwraca wskaźnik do węzła zawierającego minimalny/maksymalny klucz drzewa albo `nullptr`, gdy takiego węzła nie ma
   - `BSTNode* Predecessor(const key_t k)`, `BSTNode* Successor(const key_t k)`: zwraca wskaźnik do węzła zawierającego poprzednik/nastepnik klucza k albo `nullptr`, gdy takiego węzła nie ma

4. W przypadku próby wstawienia istniejącego klucza funkcja nie zmienia drzewa i nie zgłasza błędu. W przypadku próby usunięcia nieistniejącego węzła funkcja nie zmienia drzewa i nie zgłasza błędu.
5. Dostępne są funkcje:
   - wypisująca klucze w kolejności rosnącej
   - wyświetlająca na ekranie drzewo w sposób umożliwiający odczytanie jego struktury
   - zapisująca do pliku drzewo w sposób umożliwiający odczytanie jego struktury

6. Dostepna jest funkcja wczytująca z pliku ciąg elementów typu `key_t` i budująca na jego podstawie drzewo

Wymagania dodatkowe (mogą być spełnione)

1. Węzeł drzewa zawiera dodatkowo pole `data` reprezentujace dane towarzyszące typu `data_t`, który jest złożonym typem danych (strukturą, klasą).
   - (wariant a) pole key może być wyznaczone podstawie jednego z pól `data` albo tożsame z jednym z pól. Przykładem może być klasa reprezentująca samochody z polami `nr_rejestracjny` i `marka`
   - (wariant b) typ `data_t` dostarcza własną funkcję porównującą i jest używany jako klucz, w tym wypadku może być konieczna modyfikacja BSTNode oraz interfejsów funkcji. Przykładem typu `data_t` może być klasa reprezentująca punkt na płaszczyźnie wraz z porządkiem leksykograficznym
2. Drzewo pozwala na przechowywanie zdublowanych kluczy.
3. Dostępne są lokalne wersje funkcji `Search`, `Min`, `Max`, `Predecessor`, `Successor` itd., które operują na wskazanym poddrzewie, np.:
   - `BSTNode* Min(BSTNode* subtree_root)` wyszukuje element minimalny w poddrzewie o korzeniu `subtree_root`
4. Dostępne są funkcje:

   a) raportująca węzły na ustalonym poziomie od lewej do prawej, np.: poprzez zwrócenie listy lub tablicy zawierająca wskaźniki do tych węzłów

   b) wyznaczająca wysokość poddrzewa

   c) zliczająca ilość liści w poddrzewie

   d) zliczająca ilość węzłów w poddrzewie

   e) zliczająca ilość węzłów na ustalonym poziomie w poddrzewie
5. Dostepna jest funkcja wyświetlająca na ekranie i zapisująca do pliku drzewo w sposób atrakcyjny wizualnie, np.: dodane krawędzie, drzewo jest narysowane tak wąsko jak to tylko możliwe itp.