Zadanie projektowe nr 1

Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych.

Należy zaimplementować oraz dokonać pomiaru czasu działania operacji takich jak dodawanie elementu, usunięcie elementu i wyszukanie elementu w następujących strukturach danych:

a) Tablica,
b) Lista dwukierunkowa,
c) Kopiec binarny (typu maksimum – element maksymalny w korzeniu) ,
d) Drzewo przeszukiwań binarnych BST (Binary Search Tree),
e) Drzewo czerwono-czarne (należy wcześniej zapoznać się z drzewem BST),
f) Drzewo AVL (należy wcześniej zapoznać się z drzewem BST).

Należy przyjąć następujące założenia:

- podstawowym elementem struktur jest 4 bajtowa liczba całkowita,
- wszystkie struktury danych powinny być alokowane dynamicznie (w przypadku tablic powinny zajmować jak najmniej miejsca – powinny być relokowane przy dodawaniu/usuwaniu elementów),
  - tablica, która służy do przechowywania kopca nie będzie relokowana po każdej operacji dodawania lub usuwania – przyjąć jakąś stałą wartość nadmiarową, aby można było wykonać na niej operacje dodawania.
- w przypadku tablicy i listy rozpatrzeć osobno operacje dodawania i usuwania elementu na następujących pozycjach:
  a. początek tablicy/listy,
  b. koniec tablicy/listy,
  c. losowe miejsce tablicy/listy (losujemy indeks)
- w przypadku usuwania z drzewa - usuwamy zawsze korzeń,
- w przypadku wyszukiwania - generujemy liczbę, której szukamy w strukturze,
- lista powinna być implementowana z ogonem,
- w przypadku drzewa BST implementujemy algorytm równoważenia drzewa (algorytm DSW), który wywołujemy po automatycznie w przypadku usuwania lub dodania elementu (czas tych operacji mierzymy razem z operacją równoważenia),
- należy pomierzyć czasy wykonywania poszczególnych operacji w funkcji rozmiaru danej struktury (ilości elementów w niej przechowywanych). W przypadku zbyt krótkich czasów należy zwiększyć ilość danych pomiarowych. Ponieważ wyniki zależą także od rozkładu danych, to pomiary dla

konkretnego rozmiaru struktury należy wykonać wielokrotnie (np. 100 razy – za każdym razem generując nową populację i dla każdej nowej populacji dodać/usunąć np. *k* liczb – oczywiście k<<n, np. gdy n=2000, to k=100) a wynik uśrednić. Ilość przechowywanych elementów należy dobrać eksperymentalnie (np. może to być 5000, 8000, 10000, 16000, 20000,40000,60000,100000) w zależności od wydajności sprzętu. Nie włączać do czasu pomiaru czasu generacji danych. Ilość różnych punktów pomiarowych (rozmiarów problemu) musi wynosić co najmniej 8. Podczas pomiaru czasu wykonywania wyłączyć zbędne aplikacje,

- należy mieć na uwadze, że czas wykonywania operacji może zależeć od wartości przechowywanych elementów, co należy uwzględnić w pomiarach i wnioskach,
- dodatkową funkcją programu musi być możliwość sprawdzenia poprawności zaimplementowanych operacji i zbudowanej struktury (szerzej w na ten temat w dalszej części dokumentu),
- do dokładnego pomiaru czasu w systemie Windows w c++ można skorzystać z funkcji *QueryPerformanceCounter* lub `std::chrono::high_resolution_clock` (opis na stronie http://cpp0x.pl/forum/temat/?id=21331),
- dopuszczalnymi językami programowanie są języki kompilowane do kodu natywnego (np. C, C++), a nie interpretowane lub uruchamiane na maszynach wirtualnych (np. JAVA, .NET, Phyton),
- używanie okienek nie jest konieczne i nie wpływa na ocenę (wystarczy wersja konsolowa),
- nie wolno korzystać z gotowych bibliotek np. STL, Boost lub innych – wszystkie algorytmy i struktury muszą być zaimplementowane przez studenta (nie kopiować gotowych rozwiązań),
- realizacja zadania powinna być wykonana w formie jednego programu,
- kod źródłowy powinien być komentowany,
- program musi skompilowany do wersji exe (i w takiej wersji zostanie poddany testom),
- program musi być napisany w wersji obiektowej (tzn. pojedyncza struktura i operacje na niej
- będą tworzyły obiekt).

Sprawdzenie poprawności zbudowanej struktury/operacji obejmuje:

- utworzenie struktury z liczb zapisanych w pliku tekstowym .Każda liczba będzie w osobnej linii, natomiast pierwsza liczba określa ilość zapisanych liczb w pliku. Nazwa pliku nie może być na sztywno wpisana do programu – należy o nią zapytać przy wczytywaniu danych z pliku. Przy tworzeniu nowej struktury z pliku poprzednie dane muszą zostać usunięte. W przypadku tablicy i listy kolejność danych musi być taka sama jak w pliku. W przypadku drzew tworzymy je poprzez kolejne dodawanie elementów z pliku w kolejności jak są zapisane w pliku (w przypadku kopca, można użyć także algorytmu Floyda tzw. Floyd's Heapify),

- wyświetlenie tej struktury na ekranie (w przypadku drzew wymyślić jakąś przejrzystą formę – kopiec też jest drzewem – można korzystać z gotowych rozwiązań – kopiec ma być wyświetlamy dodatkowo w postaci tablicy),
- możliwość wykonania operacji na strukturze z tym, że w przypadku:
    a. Usuwania z tablicy i kopca - zostanie podana pozycja liczby(indeks), którą należy usunąć. Indeksujemy od zera,
    b. Usuwania z listy - zostanie podana tylko wartość , którą należy usunąć (jeżeli wartości się powtarzają usuwamy tylko pierwszą wartość) ,
    c. Usuwania (dodawania) dla drzew – zostanie podana liczba (klucz), którą należy usunąć (dodać). Z kopca usuwamy dowolny element poprzez podanie indeksu,
    d. Dodawanie do tablicy - zostanie podana pozycja i wartość, którą należy wstawić na podana pozycję (tablice w razie konieczności rozsunąć)-nie robić w menu dodatkowej pozycji dodawania na początek i koniec,
    e. Dodawania do listy - podajemy wartość i numer pozycji (indeks) na który mamy wstawić wartość -nie robić w menu dodatkowej pozycji dodawania na początek i koniec,
    f. Wyszukiwania (dla wszystkich struktur) – zostanie podana liczba, którą należy znaleźć – należy tylko wyświetlić, czy liczba jest w strukturze, czy nie.

Menu programu

Opisane operacje najlepiej zrealizować w formie dwupoziomowego menu, w którym najpierw wybieramy strukturę a później wyświetla się menu dla każdej struktury (prawie identyczne), gdzie będzie możliwość wykonania tych operacji np. (poziom 2 menu):

1.Zbuduj z pliku (elementy powinny się pojawić w tablicy i liście w takiej kolejności jak w pliku, natomiast w przypadku drzew budowanie ma polegać na dodawaniu kolejnego wczytanego elementu do drzewa. Opcja „Zbuduj z pliku” powinna usunąć poprzednie dane).

2.Usuń

3.Dodaj

4.Znajdź

5. Utwórz losowo (losowe wygenerowanie struktury) – należy spytać o wielkość struktury.

6.Wyświetl (tablicę, listę, kopiec, drzewo). Tablicę oraz listę wyświetlamy w jednej linii (elementy rozdzielamy spacją). W przypadku listy lista jest wyświetlana dwukrotnie - od przodu w jednej linii i od tyłu w drugiej. Do wyświetlania drzew można skorzystać z gotowych rozwiązań.

7.Równoważenie drzewa (tylko dla drzewa BST – operacja: zbuduj z pliku, dodaj, usuń powinna być rozdzielona od operacji równoważenia drzewa dla celów sprawdzania poprawności programu) – ale do pomiaru czasu powinna być połączona z dodawaniem lub usuwaniem.

Student może wbudować także własne opcje w menu, ale nie będą one testowane przez prowadzącego.

UWAGA ! Po każdej operacji (oprócz wyszukiwania) należy z automatu wyświetlić strukturę

Sprawozdanie powinno zawierać:

- krótki wstęp w którym zostaną przedstawione złożoności obliczeniowe operacji w implementowanych strukturach na podstawie literatury,
- plan eksperymentu czyli założenia co do wielkości struktur, sposobu generowania elementów tych struktur, sposobie pomiaru czasu, itp.,
- zestawienie wyników w formie tabelarycznej i graficznej (czas wykonania danej operacji w funkcji ilości elementów – użyć wykresu linowego po osi X). Dla każdej operacji konstruujemy osobny wykres na którym razem umieszczamy wyniki czasowe dla danej operacji dla wszystkich badanych struktur w formie krzywych (wskazane jest połączenie punktów z danej struktury zwykłymi liniami lub lepiej utworzyć krzywą aproksymowaną złożonością obliczeniową). Ponieważ liczba operacji wynosi 3 (dodawanie, usuwanie, wyszukiwanie) zatem będą 3 osobne wykresy,
- dla tablicy i listy ilość krzywych będzie potrojona dla operacji dodawania/usuwania, bo musi uwzględniać pozycję (początek, koniec, losowy),
- wnioski dotyczące efektywności poszczególnych struktur w zależności od zastosowań, wielkości struktury itp., wskazać (jeśli są) przyczyny rozbieżności pomiędzy uzyskanymi eksperymentalnie złożonościami, a teoretycznymi
- jednostki w sprawozdaniu powinny być mianowane i mieć zachowaną odpowiednią precyzję.

Maksymalna ocena za zadanie:

3.5 - eksperymenty na tablicy, liście i kopcu binarnym (program w wersji obiektowej) – po 0,5 za każdy z tych 3 elementów

3.75 – jak na 3.5 + 0,25 za wykonanie pomiaru czasu tworzenia kopca metodą Floyda (umieścić w sprawozdaniu odpowiedni osobny wykres)

4.0 - eksperymenty na tablicy, liście i kopcu binarnym i drzewie BST bez równoważenia drzewa - po 0,5 za każdy z tych 4 elementów

4.3 - eksperymenty na tablicy, liście i kopcu binarnym i drzewie BST z równoważenia drzewa algorytmem DSW - po 0,5 za tablicę, listę, kopiec, drzewo BST oraz równoważenie BST

4.7 - eksperymenty na tablicy, liście i kopcu binarnym i drzewie AVL – po 0,5 za tablicę, listę i kopiec oraz 1,2 za drzewo AVL

5.0 - eksperymenty na tablicy, liście i kopcu binarnym i drzewie czerwono- czarnym – po 0,5 za tablicę, listę i kopiec oraz 1,5 za drzewo czerwono-czarne

Tzw. „wysypywanie się” programu skutkuje obniżeniem oceny oraz koniecznością jego poprawienia.

Na stronie http://jaroslaw.mierzwa.staff.iiar.pwr.edu.pl/sdizo/menu.cpp podano przykładowe menu, które można wykorzystać przy pisaniu programu. Osoby, które piszą w innym języku też zachęcam do zapoznania się z wartością wskazywanego pliku.

W przypadku korzystania z innych języków programowania zabrania korzystania się z gotowych kontenerów implementujących w/w struktury.

Pozostałe uwagi

- nie pisać w sprawozdaniu jak wyglądają poszczególne operacje teoretycznie
- wykresy powinny być czytelne
- programy najczęściej wysypują się przy operacjach na skrajnych pozycjach struktury (np. w liście po usunięciu ostatniego elementu dodanie nowego)
- całkowita odbudowa (tworzenie od początku) kopca do dodaniu lub usunięciu jest błędem - przy tych operacjach stosujemy odpowiednio naprawę w górę lub w dół odpowiednio (nie używać funkcji tworzącej kopiec)
- przy usuwaniu elementu z kopca z dowolnego miejsca robimy naprawę w dół lub w górę albo wcale
- na ocenę wpływa także zgodność programu i sprawozdania z wytycznymi
- proszę zwrócić uwagę, że niektóre typy operacji podczas badań różnią się od operacji podczas sprawdzania poprawności algorytmu (np. dla kopca procedura usuwania polega na usuwaniu korzenia podczas pomiaru czasu, a podczas sprawdzania przez prowadzącego polega na zalezieniu elementu w drzewie i jego usunięciu)

Wizualizacja operacji na drzewie czerwono-czarnym

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Wizualizacja operacji na drzewie AVL

https://www.cs.usfca.edu/~galles/visualization/AVLtree.html