Laboratorium 3

Zastosowania drzew binarnych

Cel ćwiczenia

Celem ćwiczenia jest praktyczne zapoznanie się z wybranymi algorytmami operującymi na strukturach drzewiastych dla różnych typów danych, złożonością obliczeniową, wydajnością i możliwością jej poprawy w wybranych przypadkach.

UWAGA! Przypomnienie: do porównania kilku metod powinny być wykorzystywane zawsze **te same dane wejściowe**. Wszystkie pomiary czasu powinny być wykonywane na tej samej maszynie a przynajmniej na maszynach o identycznych parametrach, (np. na dwóch różnych komputerach laboratoryjnych). Należy wziąć to pod uwagę planując wykonanie części ćwiczenia w domu.

Struktura programu *aisd23*

Program *aisd23* jest wykorzystywany w dwóch ćwiczeniach: drugim (sortowanie metodami „standardowymi") i trzecim (sortowanie z użyciem drzew binarnych oraz inne zastosowania takich drzew). Podstawowy opis programu znajduje się zatem w instrukcji do ćwiczenia nr 2. W niniejszej instrukcji przedstawiona zostanie tylko ta część programu, której używa się w ćwiczeniu nr 3.

Kod programu jest rozbity na wiele plików lecz wykonanie ćwiczenia wymaga wprowadzenia do nielicznych z nich zaledwie kilku modyfikacji. **Po wprowadzeniu dowolnej zmiany w kodzie należy usunąć poprzednie produkty kompilacji i ponownie skompilować cały projekt wpisując kolejno *make clean* i *make*.** Poniżej wymienione są najważniejsze pliki, które mogą (lub powinny) być modyfikowane w ćwiczeniu.

1. **Podstawowe parametry eksperymentu (niezależne od badanego algorytmu)** – plik *control.h.* Patrz instrukcja do ćwiczenia 2.
2. **Wybór rodzaju drzewa** – plik *control_drzewa.h*. Zdefiniowane są tu makrodefinicje–flagi służące do wyboru badanego rodzaju drzewa oraz pozwalające sterować przebiegiem niektórych algorytmów. (**Uwaga:** pewne algorytmy sortowania – lub ich fragmenty – badane w ćwiczeniu nr 2 są wykorzystywane również jako części składowe algorytmów badanych w ćwiczeniu nr 3. Ustawienie w pliku *control_sort-std.h* flag dla algorytmów sortowania może mieć zatem wpływ na przebieg działania algorytmów „drzewiastych" z ćwiczenia nr 3).

   Poniżej znajduje się lista makrodefinicji dostępnych w pliku *control_drzewa.h*:

   - BADANIE_KOPCA, BADANIE_TURNIEJU, BADANIE_HUFFMANA, BADANIE_BST – wybór rodzaju badanego drzewa. Tylko jedna z tych flag może mieć wartość „tak".
   - KOPCOWANIE_ZSTEPUJACE – flaga stosowana przy badaniu algorytmów kopcowych służąca do wyboru rodzaju zastosowanego algorytmu ukopcowania.

- KOPCOWANIE_ODWROCENIE – flaga stosowana przy badaniu algorytmów kopcowych. Ustawienie jej na „tak” powoduje odwrócenie kolejności rekordów w tablicy po zakończeniu sortowania metodą kopcową.
- KOPCOWANIE_POPRAWKA – flaga stosowana przy badaniu algorytmów kopcowych. Służy do przyspieszenia sortowania przez kopcowanie z odcinaniem korzenia. Odcinany jest nie tylko korzeń ale i jego potomkowie jeżeli wartości ich kluczy są równe wartości klucza korzenia.
- ROZMIAR_BAJTU – makrodefinicja stosowana przy badaniu algorytmu Huffmana. Określa rozmiar (w bitach) jednostkowej porcji informacji (dane wejściowe dzielone są właśnie na takie porcje).
- NAZWA_PLIKU – makrodefinicja stosowana przy badaniu algorytmu Huffmana. Zawiera nazwę pliku z danymi wejściowymi. Standardowo jest to string „hamlet.txt” będący nazwą dostarczonego pliku z pełnym tekstem „Hamleta” Williama Szekspira.

3. **Wybór eksperymentu** – plik *badanie_alg_sort-drzewa.hpp* (analogiczny do stosowanego w ćwiczeniu nr 2 pliku *badanie_alg_sort-std.hpp*). Służy do przeprowadzania eksperymentów z sortowaniem używającym drzew binarnych).
4. **Wybór eksperymentu** – modyfikacje sterowania eksperymentami drzewem Huffmana (np. po dodaniu własnych metod) można implementować w pliku *main.cpp*
5. **Wybór eksperymentu** – plik *badanie_bst.hpp* służy do sterowania eksperymentami z drzewem BST. Przykładowo w pliku tym pokazano wywołanie dwóch rotacji.

Uwaga: podstawowe zasady tworzenia struktury klas stosowane dla drzewa Huffmana i dla drzewa BST są inne niż zasada tworzenia struktury klas stosowana do badania wszystkich metod sortowania z ćwiczeń 2 i 3 (różnią się także nieco między sobą). Przed przystąpieniem do ewentualnych modyfikacji kodu należy się dokładnie zapoznać z tymi strukturami.

Uwaga: wydruk kontrolny zawartości drzewa BST uzyskiwany na konsoli po wywołaniu procedury *wypiszNaKonsole* jest „obrócony o 90°”, tzn. zorientowany horyzontalnie, a nie wertykalnie – jak to się zwykle rysuje. Wywołanie *wypiszNaKonsole()* wyprowadza tylko istniejące węzły drzewa BST. Natomiast wywołanie *wypiszNaKonsole(true)* wyprowadza te węzły oraz kreski w miejscu węzłów nieistniejących na całej wysokości drzewa.

Przebieg ćwiczenia

Dla wskazanych przez prowadzącego algorytmów sortowania należy przystosować kod przykładowego programu do zaplanowanych pomiarów i symulacji. W zadaniach wymagających modyfikacji algorytmów **konieczna jest weryfikacja poprawności działania własnych metod i zamieszczenie w sprawozdaniu zmodyfikowanego kodu.**

Przykładowe zadania do wykonania na ćwiczeniu (prowadzący może podać inne):

1. Zbadać złożoność obliczeniową wskazanych algorytmów sortowania wziąwszy pod uwagę ewentualne początkowe uporządkowanie sortowanego zbioru danych.

2. Zbadać stabilność wybranych algorytmów sortowania.
3. Wyznaczyć złożoność obliczeniową sortowania przez kopcowanie z odcięciem liścia, z odcięciem korzenia oraz z odcięciem korzenia z poprawką. Aby zbadać wpływ zakresu wartości kluczy, należy wziąć pod uwagę przypadki kluczy bez powtórzeń oraz kluczy losowanych z możliwością powtórzeń z różnych zakresów (argument funkcji *fillWithRandomRecords,* wywoływanej w pliku *main.cpp*).
4. Zaimplementować dla kopca zmodyfikowaną procedurę spływu korzenia w kierunku liścia (metoda *fixDownHeap*) – zamiast za każdym razem zamieniać węzeł z jego potomkiem o mniejszym kluczu należy zapamiętać korzeń w buforze i „przesuwać w górę" kolejne następniki tak długo, jak długo ich klucze będą mniejsze lub równe od klucza swojego poprzednika. Na końcu procedury znajdujący się w buforze węzeł wstawić na ostatnią „opróżnioną" pozycję. Należy użyć zmodyfikowanej metody do sortowania przez kopcowanie (w wersji „zamiana korzenia z liściem") i porównać czas wykonania z oryginalną implementacją dla różnych liczności sortowanych zbiorów.
5. Zmodyfikować operację spływu węzła do liścia, tak by składała się z dwóch etapów: 1) dany węzeł jest *bezwarunkowo* zamieniany z mniejszym ze swoich potomków tak długo, aż stanie się liściem (czyli w stosunku do „standardowej" procedury spływu eliminowane jest jedno porównanie na każdym poziomie drzewa); 2) spływ tego węzła w kierunku przeciwnym – do korzenia. Można oczekiwać, że „powrót" węzła w kierunku korzenia zakończy się bardzo szybko – w kopcu bowiem około połowy elementów znajduje się na najniższym poziomie, około trzech czwartych – na najniższych dwóch poziomach itp. Sumarycznie zatem liczba porównań w zmodyfikowanym spływie do liścia może być znacząco mniejsza niż w procedurze oryginalnej. Zbadać prawdziwość tej hipotezy: należy użyć zmodyfikowanej metody do sortowania przez kopcowanie (w wersji „zamiana korzenia z liściem") i porównać czas wykonania z oryginalną implementacją dla różnych liczności sortowanych zbiorów.
6. Zbadać wydajność i porównać różne wersje algorytmu sortowania turniejowego.
7. Zbadać zależności występujące dla stopnia kompresji pliku tekstowego z użyciem drzewa Huffmana, np. dla różnych tekstów (ale o podobnej objętości) w tym samym języku, dla tekstów o różnej objętości w tym samym języku, dla tekstów o tej samej objętości w różnych językach, itp
8. Określić stopień kompresji pliku tekstowego z użyciem drzewa Huffmana dla różnych wartości rozmiaru elementarnej porcji danych („bajtu") – znaleźć rozmiar, przy którym kompresja będzie największa. Zbadać zależność wartości tego parametru od języka i objętości tekstu.
9. Zaimplementować wybrane (podane przez prowadzącego) metody równoważenia drzewa BST i przetestować ich stosowalność i skuteczność.

Analiza wyników, wnioski, sprawozdanie

Ocena z ćwiczenia bezpośrednio zależy od zawartości oddanego w terminie sprawozdania (wyłącznie w postaci elektronicznej w formacie PDF). Sprawozdanie powinno zawierać następujące elementy:

1. Cel ćwiczenia (własnymi słowami).
2. Stosowne wykresy itp.,
3. W przypadku zadań wymagających modyfikacji kodu:
   - listing zmienionej funkcji (**z wyraźnym wyróżnieniem własnych modyfikacji**),
   - opis zmian i ich uzasadnienie,
   - wynik **testowania poprawności** własnej implementacji.
4. Podsumowanie ćwiczenia czyli wnioski: **konkretne**, **na temat**, **samodzielne**, **twórcze**.

Sprawozdanie powinno zostać nadesłane pocztą elektroniczną w terminie podanym przez prowadzącego zajęcia (zazwyczaj 7–14 dni). **Pod rygorem nie sprawdzania sprawozdania dodatkowo niezbędne jest wypełnienie następujących wymogów formalnych**:

- jedyny przyjmowany format to **PDF**, przy czym tekst **nie może** być wklejony jako rysunek,
- nazwa pliku zawierającego sprawozdanie musi mieć formę **AISDE_3_Nazwisko_Imię.pdf**.
- temat maila ze sprawozdaniem musi pasować do wzorca: **AISDE_3_Nazwisko_Imię.**

Wymagania do kolokwium wstępnego:

- znajomość definicji, struktury i zastosowań drzew binarnych omawianych na wykładzie,
- znajomość badanych algorytmów używających drzew binarnych,

Literatura

1. R. Sedgewick *Algorytmy w C++*, Wydawnictwo RM 1999
2. A. Drozdek *C++ algorytmy i struktury danych*, Wydawnictwo Helion 2004
3. T. Cormen et al. *Wprowadzenie do algorytmów*, Wyd. Naukowe PWN 2013
4. J. Grębosz *Symfonia C++* Wydawnictwo Oficyna Kallimach 1997
5. J. Grębosz *Pasja C++* Wydawnictwo Oficyna Kallimach 1997

Opracowanie: dr inż. Dominik Kasprowicz, *dkasprow@imio.pw.edu.pl*
dr inż. Adam Wojtasik, *aw@imio.pw.edu.pl*

NOTATNIK

AISDE LAB 3

Imię i nazwisko, Grupa dziekańska	Nr Indeksu	Data wykonania ćwiczenia	Data oddania protokołu

Punkty do wykonania	1	2	3	4	5	6	7	8	9	Wskazania prowadzącego:

Analizowany algorytm	Parametry	Uwagi / Zauważone błędy

NOTATNIK NALEŻY ODDAĆ PROWADZĄCEMU ZAJĘCIA W CHWILI ZAKOŃCZENIA ĆWICZENIA