PAMSI – Sortowanie (na dwa kolejne zajęcia)

Zagadnienia: Algorytmy sortowania, analiza efektywności

Na pierwszych zajęciach kartkówka z sortowania przez kopcowanie i scalanie (nie będzie to polegało na pisaniu programu).

Należy wybrać **trzy** algorytmy sortowania (według tabeli). Zaimplementować (**samodzielnie**) wybrane algorytmy oraz przeprowadzić analizę ich efektywności. Wskazane jest użycie szablonów (*templates*) aby łatwo można było wykorzystać zaimplementowane algorytmy do sortowania tablic zawierających elementy różnych typów dla uzyskania oceny 5.

Ocena	Algorytmy
3	bąbelkowe, przez wstawianie, przez kopcowanie, przez scalanie
4	przez kopcowanie, przez scalanie, Shella, quicksort
5	przez scalanie, quicksort, introspektywne

Uwaga! Na pierwszych zajęciach powinien być zaimplementowany co najmniej jeden algorytm.

Wymaganie dotyczące aplikacji:

Aplikacja powinna umożliwiać załadowanie tablicy z pliku tekstowego, posortowanie jej oraz zapis wyników do innego pliku tekstowego. Tak powinno być dla każdego wybranego przez studenta algorytmu sortowania.

Aplikację można podzielić na dwie części. W jednej wykonujemy testy z plikiem, w drugiej przeprowadzamy testy efektywności. Przy niewielkim nakładzie pracy można zrobić tak, aby aplikacja wygenerowała od razu potrzebne wyniki i zapisywała je do pliku tekstowego, z którego można wczytać dane do arkusza kalkulacyjnego i tam zrobić wykresy.

Testy efektywności

Dla 100 tablic (elementy typu całkowitoliczbowego) o następujących rozmiarach: 10 000, 50 000, 100 000, 500 000 i 1 000 000 wykonać eksperymenty z sortowaniem (zmierzyć czas sortowania) w następujących przypadkach:

- wszystkie elementy tablicy losowe,
- 25%, 50%, 75%, 95%, 99% początkowych elementów tablicy jest już posortowanych,
- wszystkie elementy tablicy już posortowane, ale w odwrotnej kolejności.

W celu uzyskania wiarygodnych wyników zaleca się powtórzenie eksperymentu wielokrotnie. Aby wyniki były porównywalne każdy algorytm powinien sortować tą samą tablicę, co pozostałe. Należy pamiętać aby mierzyć sam czas sortowania, a nie także czas dodatkowych operacji (np. generowania tablicy).

Pomiar czasu wykonujemy za pomocą funkcji `QueryPerformanceCounter` dla systemu Windows

Zawartość sprawozdania

- krótkie wprowadzenie,
- dla testowanych algorytmów krótkie omówienie ich złożoności obliczeniowej (dla przypadku średniego i najgorszego),
- omówienie przebiegu eksperymentów i przedstawienie uzyskanych wyników (w postaci tabel i wykresów – do każdego wykresu tabela),
    - dla uporządkowania losowego na jednym wykresie przedstawić wyniki dla różnych algorytmów sortowania. Powtórzyć to samo dla wstępnego uporządkowania tablicy 25% oraz 99%
    - dla każdego algorytmu przedstawić na jednym wykresie wyniki sortowania dla różnych sposobów ułożenia elementów w tablicy sortowanej (tyle wykresów, ile algorytmów)
- krótkie podsumowanie i wnioski (w przypadku niezgodności uzyskanych wyników z przewidywanymi spróbować wyjaśnić przyczyny),
- w sprawozdaniu umieścić parametry komputera (typ procesora, pamięć) na którym wykonywano eksperymenty,
- literatura (materiały wykorzystane do wykonania ćwiczenia, również strony internetowe).

Krótki opis algorytmów

Sortowanie bąbelkowe (*buble sort*)

Porównywane są dwa sąsiednie elementy tablicy, na początku ostatni i przedostatni. Jeśli ich kolejność jest niewłaściwa, to zamieniane są one miejscami. Porównanie (i ewentualna zamiana) powtarzane jest dla pozostałych elementów. Po pierwszym przebiegu element najmniejszy znajdzie się na początku tablicy (w części uporządkowanej). W kolejnych krokach procedura jest powtarzana, ale tylko dla nieuporządkowanej części tablicy. W ulepszonej wersji algorytmu sortowanie kończy się w sytuacji, gdy po wykonaniu całego przebiegu nie zostanie wykonana ani jedna zamiana.

http://pl.wikipedia.org/wiki/Sortowanie_b%C4%85belkowe

Sortowanie przez wstawianie (*insertion sort*)

Zakładamy, że początkowa część tablicy jest już posortowana (na początku będzie to pierwszy element tablicy). Pierwszy element z drugiej (nieposortowanej) części tablicy przenoszony jest do zmiennej pomocniczej. Porównywana jest wartość tej zmiennej z kolejnymi elementami pierwszej (posortowanej) części tablicy (zaczynając od ostatniego). Dopóki elementy te są większe, to przesuwane są o jedną pozycję w prawo. Po zakończeniu przesuwania element ze zmiennej pomocniczej wstawiany jest w wolne miejsce. Procedura jest powtarzana dopóki pozostają jeszcze jakieś elementy w drugiej, nieuporządkowanej części tablicy.

http://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie

Sortowanie przez kopcowanie (*heap sort*)

Z sortowanej tablicy tworzony jest kopiec binarny. W korzeniu kopca znajdzie się element największy (jeśli przyjmiemy sortowanie w porządku niemalejącym). Element ten zamieniany jest z ostatnim elementem kopca. Elementy przenoszone na koniec tworzą część posortowaną i w kolejnych krokach nie wchodzą już w skład kopca. Pozostały kopiec (bez elementów już posortowanych) jest naprawiany (przywracanie własności kopca). Procedura jest powtarzana dopóki wszystkie elementy nie zostaną posortowane.

http://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie

Sortowanie przez scalanie (*merge sort*)

Sortowana tablica dzielona jest rekurencyjnie na dwie podtablice aż do uzyskania tablic jednoelementowych. Następnie podtablice te są scalane w odpowiedni sposób, dający w rezultacie tablicę posortowaną. Wykorzystana jest tu metoda podziału problemu na mniejsze, łatwiejsze do rozwiązania zadania („dziel i rządź”).

http://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie

Sortowanie Shella (*Shellsort*)

Sortowana tablica dzielona jest na kilka podtablic, utworzonych z elementów oddalonych o ustaloną liczbę pozycji. Tablice te są oddzielnie sortowane (zwykle metodą sortowania przez wstawianie lub bąbelkowego – algorytmy te dobrze sprawdzają się dla danych częściowo posortowanych). W kolejnych krokach wykonywane są podziały z coraz mniejszym odstępem (dające coraz mniejszą liczbę podtablic) aż do uzyskania tylko jednej tablicy. Kluczowym problemem jest dobór odpowiedniej sekwencji odstępów przy podziałach tablicy (można użyć zasady zaproponowanej przez D. Knutha).

http://pl.wikipedia.org/wiki/Sortowanie_Shella

Sortowanie szybkie (*quicksort*)

Na początku wybierany jest tzw. element osiowy. Następnie tablica dzielona jest na dwie podtablice. Pierwsza z nich zawiera elementy mniejsze od elementu osiowego, druga elementy większe lub równe, element osiowy znajdzie się między nimi. Proces dzielenia powtarzany jest aż do uzyskania tablic jednoelementowych. Właściwe sortowanie jest tu jakby ukryte w procesie przygotowania do sortowania. Wybór elementu osiowego wpływa na równomierność podziału na podtablice (najprostszy wariant – wybór pierwszego elementu tablicy – nie sprawdza się w przypadku, gdy tablica jest już prawie uporządkowana).

http://pl.wikipedia.org/wiki/Sortowanie_szybkie

Sortowanie introspektywne (*introspective sort* lub *introsort*)

Jest to metoda hybrydowa, będąca połączeniem sortowania szybkiego i sortowania przez kopcowanie. Sortowanie introspektywne pozwala uniknąć najgorszego przypadku dla sortowania szybkiego (nierównomierny podział tablicy w przypadku, gdy jako element osiowy zostanie wybrany element najmniejszy lub największy).

http://pl.wikipedia.org/wiki/Sortowanie_szybkie

Dla wygody podane zostały linki do *Wikipedii*. Nie należy ich jednak traktować jako jedynego źródła informacji.

Literatura:

[1] Cormen T., Leiserson C.E., Rivest R.L., Stein C., *Wprowadzenie do algorytmów*, WNT
[2] Drozdek A., *C++. Algorytmy i struktury danych*, Helion