Algorytmy i struktury danych

Laboratorium - lista 2

Termin wysłania na SVN: 2025-04-06

Wykonać prezentację rozwiązań z wykorzystaniem programu `tmux`, jak opisano na Liście 0 w Zadaniu 1.
W prezentacji powinny się pojawić kolejno:

- Komentarz postaci `# imię nazwisko numer indeksu`
- polecenie postaci `svn export URL_katalogu_z_rozwiązaniami_listy`
- polecenie postaci `cd wyeksportowana_ścieżka`
- w podkatalogu z każdym zadaniem, kompilacja i uruchomienie programów dla wymaganych testów.

W przypadku testów dla dużych danych, umieścić w repozytorium jedynie obrazy wykresów z porównujące złożoności testowanych algorytmów.

Zadanie 1. [40 p.]

Celem zadania jest zaimplementowanie i przetestowanie następujących algorytmów sortowania:

- INSERTION SORT ,
- QUICK SORT ,
- *Algorytm hybrydowy*: QUICK SORT, który dla małych podtablic przełącza się na INSERTION SORT.

Zaimplementuj każdy z tych algorytmów w osobnym programie.
Elementy sortowanej tablicy nazywamy *kluczami*.
Wejściem (przez standardowy strumień wejściowy) dla programu są kolejno:

- liczba *n* — długość sortowanej tablicy,
- tablica *n kluczy* do posortowania.

Ponadto zaimplementuj programy generujące na standardowym wyjściu dane zgodne z powyższym opisem w postaci:

- ciąg losowych kluczy (zadbaj o dobry generator pseudolosowy),
- ciąg posortowany rosnąco,
- ciąg posortowany malejąco.

Każdy taki *generator danych,* jako argument z linii poleceń przyjmuje rozmiar tablicy *n.*
Testy uruchamiaj jako potok postaci: `generator_danych n | program_sortujący`
Na potrzeby testów przyjmijmy, że *klucze* są liczbami całkowitymi od zera do 2*n*-1.

Program sortujący powinien sortować tablicę wybranym algorytmem i wypisywać na standardowym wyjściu:

- Dla rozmiaru danych *n* < 40:
    - tablicę wejściową,

    - stany sortowanej tablicy w istotnych momentach (np. w MERGE SORT - po zakończeniu każdego scalania),
    - tablicę wejściową (ponownie dla porównania!),
    - tablicę po sortowaniu.
      (Dla czytelności drukujmy klucze jako liczby dwucyfrowe.)
- Dla dowolnego rozmiaru danych, na końcu:
    - łączną liczbę porównań między *kluczami*,
    - łączną liczbę przestawień *kluczy*.
      (Zaimplementować osobne funkcje/procedury do porównywania i przestawiania *kluczy*, które dodatkowo zwiększają swój globalny licznik odpowiednio porównań lub przestawień.)

Finalnie, program sam sprawdza, czy wynikowy ciąg jest posortowanym ciągiem wejściowym.

W prezentacji zademonstrować testy dla długości tablicy n ($n \in \{8, 32\}$) dla ciągów:

- losowego,
- posortowanego malejąco,
- posortowanego rosnąco.

Zadanie 2. [20 p.]

Wykorzystaj programy z zadania 1., aby porównać złożoności algorytmów.
Dla rozmiarów danych n, wykonaj po k niezależnych powtórzeń:

- sortowania ciągu każdym algorytmem,
- zapisania w pliku wykonanych liczb porównań i przestawień.

Eksperymentalnie wyznacz jak najkorzystniejszy próg przełączania między QUICK SORT a INSERTION SORT w algorytmie hybrydowym.
Wykorzystując zebrane wyniki, przedstaw na wykresach za pomocą wybranego narzędzia (np. numpy, Matlab, Mathematica):

- średnią liczbę wykonanych porównań (c) w zależności od n,
- średnią liczbę przestawień kluczy (s) w zależności od n,
- iloraz c/n w zależności od n,
- iloraz s/n w zależności od n.

Dane dotyczące różnych algorytmów sortujących przedstawić w różnych kolorach na jednym układzie współrzędnych, aby można je porównywać.

- Dla wszystkich algorytmów przeprowadź eksperymenty dla $n \in \{10, 20, 30, \ldots, 50\}$.
- Dla algorytmów różnych od INSERTION SORT, przeprowadź eksperymenty dla $n \in \{1000, 2000, 3000, \ldots, 50000\}$.

Przygotuj zestawy wykresów dla różnych wartości k (np. $k = 1$, $k = 10$, $k = 100$).

Zadanie 3. [30 p.]

Wykonaj eksperymenty analogiczne do zadań 1 i 2 dla wymyślonego przez siebie algorytmu sortowania, który:

1. Będzie charakteryzować się użyciem metodologii D&C (dziel i zwyciężaj).
2. Będzie wykorzystywać procedurę Merge z MergeSorta.
3. Do scalania będzie wykorzystywać aktualnie istniejące w danych wejściowych spójne podciągi rosnące. Taka idea jest użyta również w TimSorcie i PowerSorcie linki tutaj:
    a. https://en.wikipedia.org/wiki/Timsort
    b. https://youtu.be/exbuZQpWkQ0?feature=shared
    c. https://www.wild-inter.net/publications/munro-wild-2018.pdf
4. Twoim głównym zadaniem jest wymyślenie i zaimplementowanie zasad scalania znalezionych podciągów, które będą maksymalizować efektywności stworzonego algorytmu.
5. Porównaj na wykresie wyniki swojego algorytmu z klasycznym MergeSort'em.

Zadanie 4. [20 p.]

Uzupełnij Zadania 1. i 2. o algorytm DUAL-PIVOT QUICKSORT używając strategii COUNT :

- Mamy dwa pivoty *p* i *q* oraz załóżmy, że *p* < *q*.
- Załóżmy, że w procedurze PARTITION klasyfikując *i*-ty element tablicy mamy s_{i-1} elementów małych (mniejszych od *p*) oraz l_{i-1} elementów dużych (większych od *q*).
- Jeśli $l_{i-1} > s_{i-1}$, to porównuj *i*-ty element w pierwszej kolejności z q, a następnie, jeśli jest taka potrzeba, z *p*.
- Jeśli $l_{i-1} \leq s_{i-1}$, to porównuj *i*-ty element w pierwszej kolejności z *p*, a następnie, jeśli jest taka potrzeba, z *q*.

Dokonaj szczegółowych porównań otrzymanych statystyk dla algorytmu QUICKSORT (z zadania 2.) i DUAL-PIVOT QUICKSORT.
Przedstaw wykresy porównujące na jednym układzie współrzędnych.
Eksperymentalnie wyznacz stałą stojącą przy czynniku $n \log_2 n$ dla liczby porównań między kluczami.
W prezentacji wykonaj testy dla długości tablicy *n* ($n \in \{8, 32\}$) dla danych ciągów:

- losowego,
- posortowanego malejąco,
- posortowanego rosnąco.