Dawid Pawliczek
Lista 6, Zadanie 1

Treść

Zaproponować *nierekurencyjną* wersję sortowania Quicksort, która

- działa *w miejscu* — poza tablicą $A[1 \dots n]$ używa tylko stałej liczby zmiennych typu `int`,
- wykonuje co najwyżej stały czynnik więcej operacji niż standardowa wersja rekurencyjna.

Idea

Klasyczny Quicksort zapisuje podzadania na stosie. Obserwacja:

> *Po każdej operacji* PARTITION *należy najpierw posortować* ***mniejszy*** *z dwóch powstałych przedziałów, a większy „odłożyć na później”.*

Zamiast stosu większy przedział zakodujemy *bezpośrednio w tablicy* — przestawiając element, który bezpośrednio *za* pivotem wyróżnia jego prawą granicę. Dwa wartowniki (wartości $+\infty$) na końcu tablicy gwarantują, że wyjście z jednego przedziału zawsze zostanie wykryte liniowym skanem.

Algorytm

Algorithm 1 ITERQUICKSORT

Require: $A[1 \dots n]$ (elementy różne);
$+\infty$ większe niż każdy $A[i]$ jest zapisywalne
Ensure: A — posortowana rosnąco
$A[n+1] \leftarrow +\infty$; $A[n+2] \leftarrow +\infty$
$l \leftarrow 1$; $r \leftarrow n$ ▷ aktualny przedział
$M \leftarrow 16$ ▷ granica dla Insertion Sort
while $l < r$ **do**
  while $r - l + 1 > M$ **do** ▷ duży przedział — dziel
    $(p) \leftarrow$ HOAREPARTITION(A, l, r) ▷ $A[p]$ = pivot
    if $p - l < r - p$ **then** ▷ lewa strona mniejsza
      swap$(A[p+1], A[r+1])$ ▷ zapisz większy przedział
      $r \leftarrow p - 1$ ▷ sortuj $_m niejszy_t eraz$ **else** ▷ prawa strona mniejsza
        swap$(A[p-1], A[l-1])$
        $l \leftarrow p + 1$
      end if
    end while
    INSERTIONSORT(A, l, r) ▷ krótki przedział
    $l \leftarrow r + 2$; $r \leftarrow l$ ▷ pomiń wartownik
    while $A[r+1] < A[l-1]$ **do** ▷ szukaj kolejnej granicy
      $r \leftarrow r + 1$
    end while
  end while

Używana pamięć. Stałe zmienne l, r, p, M, dwa wartowniki i pivot wewnątrz tablicy. Brak dynamicznego stosu — warunek „in place” spełniony.

Poprawność

1. **Działanie PARTITION.** Funkcja Hoare'a rozdziela elementy na $A[l..p-1] < A[p] < A[p+1..r]$.
2. **Wybór kolejności.** Zawsze sortujemy *mniejszy* fragment ($\min(p-l,\ r-p)$) – gwarantuje to, że rozmiar „bieżącego” przedziału maleje co najwyżej o połowę, a więc liczba aktywnych przedziałów w tym samym czasie nie przekracza 2.
3. **Kodowanie większego fragmentu.** Element tuż za pivotem zamieniamy z pierwszym wartownikiem ($+\infty$). Dzięki temu liniowy skan od pivot+1 napotyka pierwszy element >pivot dokładnie w miejscu, gdzie zaczyna się odłożony przedział i jego granice są odzyskiwane.
4. **Terminacja.** Każdy element może zostać pivotem najwyżej raz; dla podtablic nie większych niż M używamy sortowania przez wstawianie. Algorytm kończy, gdy $l \geq r$ (całość posortowana).

Złożoność

- **Czas:** identyczny jak klasycznego Quicksorta z tą samą procedurą wyboru pivota. W średnim przypadku $T(n) = \Theta(n \log n)$, w najgorszym $\Theta(n^2)$; stała ukryta w symbolu Θ nie rośnie — zamiana stosu na kodowanie granic kosztuje $O(1)$ na każdą wywołaną PARTITION.
- **Pamięć:** dokładnie $O(1)$ słów `int` poza tablicą A.

Ilustracja działania — dwa krótkie przykłady

Przykład 1 — jeden poziom rekurencji

$$A = \boxed{7}\ 2\ 9\ 4\ 3\ 6\ 1\ 5 \mid +\infty\ +\infty \qquad (l = 1,\ r = 8)$$

1. PARTITION z pivotem 4 daje $2\ 3\ 1\ \boxed{4}\ 7\ 6\ 9\ 5$. Pivot $p = 4$ dzieli tablicę na mały lewy blok $[1, 3]$ i duży prawy $[5, 8]$.
2. Blok większy kodujemy: zamieniamy element *za* pivotem z wartownikiem na $r+1$:

$$2\ 3\ 1\ \boxed{4}\ 7\ 6\ 9\ 5 \mid +\infty\ +\infty \xrightarrow{\text{swap}} 2\ 3\ 1\ \boxed{4}\ \boxed{+\infty}\ 6\ 9\ 5 \mid 7\ +\infty$$

3. Sortujemy lewy (krótki) odcinek $2, 3, 1$. Po INSERTIONSORT:

$$1\ 2\ 3\ \boxed{4}\ +\infty\ 6\ 9\ 5 \mid 7\ +\infty$$

4. Odtwarzanie granic: ustawiamy $l \leftarrow r+2 = 6$, $r \leftarrow l$ i skanujemy dopóki $A[r+1] < A[l-1]$. Warunek spełnia $9 < +\infty$, potem $5 < +\infty$, natomiast $7 \not< +\infty$ — zatrzymujemy się na

$$1\ 2\ 3\ 4\ +\infty\ \boxed{6}\ 9\ 5 \mid 7\ +\infty$$

Uzyskaliśmy zakodowany wcześniej podprzedział $[6, 8]$, który sortujemy analogicznie.

Przykład 2 — zagnieżdżenie dwóch dużych bloków

$$A = \boxed{9}\,7\,8\,1\,5\,3\,4\,2 \mid +\infty\ +\infty$$

1. Pierwsza PARTITION (pivot 5): $1\,2\,4\,\boxed{5}\,9\,3\,8\,7$. Prawy blok ($[5, 8]$) większy kodujemy go zamianą elementu 9 z wartownikiem:

$$1\,2\,4\,\boxed{5}\ +\infty\,3\,8\,7 \mid 9\ +\infty$$

2. Teraz pracujemy na $[1, 3]$. Pivot 2 rozdziela na lewy blok długości 1 i prawy długości 1 — oba poniżej progu M, więc po chwili mamy

$$1\,2\,4\,5\ +\infty\,3\,8\,7 \mid 9\ +\infty$$

3. Odtwarzanie: $l \leftarrow 5 + 2 = 7$, skan: $8 < +\infty \ \Rightarrow r = 7$, $7 < +\infty \ \Rightarrow r = 8$, $9 \not< +\infty$ podprzedział $[7, 8]$, pivotujemy i kończymy.

W obu przykładach zamiana elementu bezpośrednio za pivotem z wartownikiem skutecznie „notuje" granicę odłożonego bloku; później prosty skan pozwala ją odzyskać bez potrzeby dodatkowej pamięci.