Algorytmy i programowanie zajęcia 21 i 22

Marcin Żurowski

26 maja 2020

1 Zadania - sortowanie

Termin wykonania poniższych zadań (23:59 01 czerwca 2020). Mając dany ciąg liczb rzeczywistych a_n musimy utworzyć poprzez przestawienie elementów ciągu a_n ciąg a'_n o następującej własności: dla każdego elementu ciągu a'_n jeśli $i < j$ to $a'_i \leqslant a'_j$ dla sortowania w porządku niemalejącym ($a'_i \geqslant a'_j$ dla porządku nierosnącego).

Ponieważ algorytmy sortowane nierosnąco działają podobnie jak algorytmy sortowane niemalejąco, więc będziemy zajmować się sortowaniem w porządku niemalejącym.

1. sortowanie przez wybór

   Idea sortowania przez wybór jest następująca. Ponieważ wynikowy ciąg ma mieć porządek niemalejący to najmniejszy element ciągu musi się znaleźć na pierwszej pozycji. Algorytm sortowania przez wybór polega na znalezieniu najmniejszego elementu w tablicy, umieszczeniu go na pierwszej pozycji, a następnie uruchomienia algorytmu dla tablicy począwszy od drugiej pozycji. Do implementacji tego algorytmu może przydać się zmodyfikowana funkcja z zadania **APR_038_MIN_PRZEDZIAL**.

   Przykład:

   ```
   5 2 4 6 1 3 2 6
   ```

   1 jest szukanym minimum więc zamieniamy ten element z pierwszym elementem.

   ```
   1 2 4 6 5 3 2 6
   ```

   Dalej szukamy minimum od drugiej pozycji.
   adjule zadanie: APR_041_SELECT_SORT

2. sortowanie bąbelkowe

   Innym podejściem do sortowania jest skorzystanie z faktu, że między dwiema sąsiadującymi ze sobą wartościami w ciągu niemalejącym zachodzi następująca relacja $a_i \leqslant a_{i+1}$ dla $1 \leqslant i < n$. Możemy wykorzystując ten

fakt przejść przez tablicę zamieniając sąsiednie elementy w przypadku gdy powyższa relacja miedzy sąsiednimi elementami nie zachodzi. Po jednym przejściu takiej tablicy największy element będzie się znajdował pod indeksem n. Wystarczy czynność "przejścia przez tablicę" wykonać $n-1$ razy pomijając za każdym razem k ostatnich elementów, a otrzymamy posortowaną tablicę ($n-1$ ponieważ pierwszy element po $n-1$ przejściach jest elementem najmniejszym).
adjule zadanie: APR_042_BUBBLE_SORT

Przykład:

5 2 4 6 1 3 2 6
2 5 4 6 1 3 2 6
2 4 5 6 1 3 2 6
2 4 5 6 1 3 2 6
2 4 5 1 6 3 2 6
2 4 5 1 3 6 2 6
2 4 5 1 3 2 6 6

Na ostatniej pozycji mamy największy element w całej tablicy, po $n-1$ takich operacjach na tablicach mniejszych o 1 w każdym kroku mamy posortowaną tablicę.

2 4 5 1 3 2 6 6

3. sortowanie przez wstawianie

   Jeszcze innym podejściem do sortowania jest sposób, jakiego używamy, kiedy gramy w karty. Załóżmy, że już $k = 4$ z kart mamy na właściwej pozycji i rozpatrujemy kartę o numerze 1. **Przykład:**

   2 4 5 6 1 3 2 6

   Gdybyśmy grali w karty po prostu przełożyli byśmy kartę z numerem 1 na pierwsza pozycję w ciągu zielonych posortowanych kart. W algorytmie musimy znaleźć na 1 miejsce w tablicy. Zapamiętujemy więc 1 i liczby w zielonym ciągu przestawiamy dopóki nie znajdziemy miejsca na 1 **Przykład:**

   2 4 5 6 6 3 2 6
   2 4 5 5 6 3 2 6
   2 4 4 5 6 3 2 6
   2 2 4 5 6 3 2 6
   1 2 4 5 6 3 2 6
   1 2 4 5 6 3 2 6

   W ten sposób mamy posortowany ciąg o długości $k+1 = 5$.
   adjule zadanie: APR_043_INSERT_SORT

4. sortowanie przez scalanie

   Weźmy następująca tablicę:

```
5 2 4 6 1 3 2 6
```

Wywołajmy procedurę sortowania przez scalanie w następujący sposób:

```
MERGE-SORT(A, 1, 8)
```

gdzie 1 i 8 są numerami indeksów elementów tablicy, które mają zostać posortowane. Wyznaczmy środek tej tablicy za pomocą następującej instrukcji:

```
q = (p + r) DIV 2
```

gdzie p i r są odpowiednio drugim i trzecim argumentem i mają wartość odpowiednio 1 i 8. W ten sposób otrzymujemy, że $q = 4$. Mając podzieloną tablice na dwie podtablice:

```
5 2 4 6|1 3 2 6
```

wywołujemy na tych tablicach procedurę sortowania:

```
MERGE-SORT(A, p, q)
MERGE-SORT(A, q + 1, r)
```

czyli

```
MERGE-SORT(A, 1, 4)
MERGE-SORT(A, 5, 8)
```

otrzymujemy następująca tablicę:

```
2 4 5 6|1 2 3 6
```

Na tak powstałej tablicy wywołujemy procedurę`SCAL(A, 1, 4, 8)` z zadania **APR_039_SCAL_PRZEDZIAL** i otrzymujemy rozwiązanie:

```
1 2 2 3 4 5 6 6
```

Aby wykonać wszystkie operacje należy najpierw sprawdzić warunek czy `p < q` jest prawdziwy, w ten sposób wywołując funkcję `MERGE-SORT` w następujący sposób `MERGE-SORT(A, 1, 1)` nie wykonają się powyższe operacje z powodu tego, że tablica jednoelementowa jest już posortowana.

Złożoność algorytmu

Złożoność powyższego algorytmu jest następująca:

$$T(n) = a + b + T(\frac{n}{2}) + T(\frac{n}{2}) + n = 2 \cdot T(\frac{n}{2}) + n + a + b \qquad (1)$$

a i b są stałymi czasami odpowiednio za sprawdzenie warunku $p < q$ i wyliczenia q. $T(\frac{n}{2})$ to czas wykonania obydwu sortowań podtablic natomiast funkcja `SCAL` wykonuje się w czasie n. Ponieważ $n + a + b = O(n)$, więc powyższą funkcję czasu możemy zastąpić następującą funkcją:

$$T(n) = 2 \cdot T(\frac{n}{2}) + n \qquad (2)$$

Aby określić złożoność obliczeniową powyższej funkcji czasu należy użyć:

Twierdzenie o rekurencji uniwersalnej Niech $a \geqslant 1$ i $b > 1$ będą stałymi, niech $f(n)$ będzie pewną funkcją i niech $T(n)$ będzie zdefiniowane dla nieujemnych liczb całkowitych przez rekurencję

$$T(n) = aT(n/b) + f(n),$$

gdzie n/b interpretujemy jako $\lfloor n/b \rfloor$ lub $\lceil n/b \rceil$. Wtedy funkcja $T(n)$ może być ograniczona asymptotycznie w następujący sposób:

(a) Jeśli $f(n) = O(n^{\log_b a - \varepsilon})$ dla pewnej stałej $\varepsilon > 0$, to $T(n) = \Theta(n^{\log_b a})$.

(b) Jeśli $f(n) = \Theta(n^{\log_b a})$, to $T(n) = \Theta(n^{\log_b a} \lg n)$.

(c) Jeśli $f(n) = \Omega(n^{\log_b a + \varepsilon})$ dla pewnej stałej $\varepsilon > 0$ oraz $af(n/b) \leqslant cf(n)$ dla pewnej stałej $c < 1$ i wszystkich dostatecznie dużych n, to $T(n) = \Theta(f(n))$.

Rozwiązanie:

Dla naszego przypadku mamy $a = 2$, $b = 2$, $f(n) = n$, następnie obliczamy $\log_b a = \log_2 2 = 1$.

Następnie sprawdzamy czy (z dołu, z góry, a może z obu) jest ograniczona funkcja $f(n)$ przez wyrażenie $n^{\log_b a} = n$. $f(n) = n$ jest ograniczona przez n z dołu i z góry, więc prawdziwy jest drugi przypadek $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$, a skoro spełnione jest to założenie to $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n)$
adjule zadanie: APR_044_MERGE_SORT

5. sortowanie szybkie

Podobnym podejściem jest następujące sortowanie tablicy `A`
Przykład: Weźmy następująca tablicę:

```
5 2 4 6 1 4 2 3
```

Wywołajmy procedurę sortowania szybkiego w następujący sposób:

```
QUICK-SORT(A, 1, 8)
```

gdzie $p = 1$ i $r = 8$ są numerami indeksów końców przedziału tablicy, który ma zostać posortowany. Następnie, o ile $p < r$ wywołujemy funkcję `q = PODZIEL(A, n, p, r)`, gdzie n jest liczbą elementów tablicy, a $p = 1$ i $q = 8$ końcami przedziału dla którego ma zostać wykonane sortowanie. Czyli wywołujemy funkcję w następujący sposób `q = PODZIEL(A, 8, 1, 8)` funkcja ta działa identycznie jak funkcja z zadania **APR_040_PODZIEL** i otrzymujemy rozwiązanie:

```
2 1 2|3|5 4 4 6
q = 4
```

Następnie wywołujemy `QUICK-SORT` na obu nieposortowanych podtablicach w następujący sposób:

```
QUICK-SORT(A, 1, 3)
QUICK-SORT(A, 5, 8)
```

W ten sposób sortujemy całą tablicę.
adjule zadanie: APR_045_QUICK_SORT

6. sortowanie przez zliczanie

   Sortowanie przez zliczanie jest jednym z najszybszych sposobów sortowania, jednak za pomocą tego algorytmu można sortować jedynie specyficzne dane, a mianowicie liczby całkowite. Poza powyższym ograniczeniem musimy również znać minimalną i maksymalną wartość umieszczoną w sortowanej tablicy. Algorytm sortowania przez zliczanie przedstawię na następującym przykładzie:

   Przykład: Weźmy następująca tablicę, na czerwono zapisałem indeksy:

```
A 1 2 3 4 5 6 7 8 9 10
  3 0 2 3 1 5 3 2 3 5
```

   Następnie w tablicy `C` zliczamy wystąpienia każdej liczby

```
C 0 1 2 3 4 5
  1 1 2 4 0 2
```

   Następnie dla każdego elementu wykonujemy następującą instrukcje:

```
for i = 1 to k
   C[i] = C[i] + C[i - 1]
```

   otrzymujemy tablicę pozycji

```
C 0 1 2 3 4 5
  1 2 4 8 8 10
```

   teraz każdy element o wartości i z tablicy `A` przepisujemy do tablicy `B` pod indeksem takim jaki jest wpisany w tablice `C` pod indeksem i:

```
B 1|2|3|4|5|6|7|8|9|10
   | | | | | | |3| |
```

   i zmniejszamy o 1 wartość w tablicy `C`

```
C 0 1 2 3 4 5
  1 2 4 7 8 10
```

   Na koniec całą tablicę `B` przepisujemy do tablicy `A`.
   adjule zadanie: APR_046_COUNT_SORT

Literatura

[1] B. Kernighan, D. Ritchie (2007). Język ANSI C. Wydawnictwo Naukowo-Techniczne.

[2] E. Palka (2012). Elementy algorytmiki dla początkujących. Wydawnictwo Naukowe UAM. (http://lib.amu.edu.pl/ksiazki-elektroniczne/)

[3] K. A. Ross, C. R. B. Wright (1996). Matematyka dyskretna. Wydawnictwo Naukowe PWN.