Instytut Informatyki UWr

Wstęp do informatyki

Lista 6

Uwagi:

- Programy/funkcje stanowiące rozwiązania zadań na tej liście powinny być napisane w języku C lub Python i poprzedzone prezentacją **idei rozwiązania** (najlepiej z pomocą pseudokodu i/lub słownie). Należy również przeanalizować **złożoność** czasową i pamięciową. Staraj się, aby złożoność Twojego rozwiązania była jak najmniejsza!
- W rozwiązaniach zadań nie należy korzystać z funkcji/narzędzi/operatorów wspomagających ten proces, dostępnych w języku C/Python i/lub jego bibliotekach, które nie były stosowane na wykładzie. W szczególności, należy się stosować do zaleceń odnośnie dopuszczalnych operacji z języka Python!

1. [1] Poniższy fragment programu wyszukuje miejsce w podtablicy a[0], a[1], ..., a[i – 1], w które należy wstawić a[i] (zakładamy, że ciąg a[0], a[1],..., a[i – 1] jest uporządkowany niemalejąco). Fragment ten zawiera błąd sprawiający, że dla niektórych danych program wpadnie w nieskończoną pętlę. Opisz sytuacje, w których to nastąpi. Następnie usuń błąd tak, aby wynikowy kod wykonywał co najwyżej $O(\log n)$ porównań elementów z sortowanego ciągu i końcowa wartość `lewy` miała własność: po wstawieniu `a[i]` do ciągu a[0], a[1], ..., a[i – 1] przed element `a[lewy]` uzyskamy uporządkowany ciąg złożony z i+1 elementów.

```
{  x=a[i];
   lewy=0;
   prawy=i-1;
   while (lewy<prawy) {
     k=(lewy+prawy)/2; //dzielenie całkowite!
     if (a[k]<x) lewy=k;
     else prawy=k;
   }
}
```

2. [1] Przedstaw algorytm sortowania metodą **selekcji** (selection sort). Następnie:
   a. Zapisz ten algorytm jako funkcję w języku C/Python
   b. Pokaż, że jego złożoność czasowa jest $O(n^2)$
   c. Wskaż **instrukcje dominujące** w Twojej implementacji
   d. Wyznacz liczbę porównań i podstawień elementów ciągu wykonaną przez ten algorytm na ciągu uporządkowanym $a_1 \leq \ldots \leq a_n$ i ciągu odwrotnie uporządkowanym $a_1 \geq \ldots \geq a_n$
3. [1] Przedstaw algorytm sortowania **bąbelkowego** (bubble sort). Następnie:
   a. Zapisz ten algorytm jako funkcję w języku C/Python
   b. Pokaż, że jego złożoność czasowa jest $O(n^2)$
   c. Wskaż **instrukcje dominujące** w Twojej implementacji
   d. Wyznacz liczbę porównań i podstawień elementów ciągu wykonaną przez ten algorytm na ciągu uporządkowanym $a_1 \leq \ldots \leq a_n$ i ciągu odwrotnie uporządkowanym $a_1 \geq \ldots \geq a_n$

4. [1] Twoje zadanie:
   a. Uzasadnij, że wartość liczby o zapisie binarnym a[0] a[1] … a[k] jest równa wartości pewnego wielomianu (jakiego?) w punkcie $x = 2$.
   b. Uzupełnij wyrażenie w wierszu (4) poniższej funkcji tak, aby funkcja zwracała wartość liczby, której zapis binarny składa się z cyfr a[0] a[1] … a[k].

```
(1)   int  wartosc(int a[], int k)
(2)   { int w = 0, i;
(3)     for( i = 0; i<=k; i++)
(4)       w = ..... ..
(5)     return w;
(6)   }
```

```
(1) def wartosc(a,k):
(2)     w=0
(3)     for i in range(k+1):
(4)             w=.....
(5)     return w
```

   Uwaga. Zapis binarny a[0] a[1] … a[k] oznacza tutaj, że a[k] jest najmniej znaczącą cyfrą liczby, a a[0] jej najbardziej znaczącą cyfrą!

5. [2] Sito Eratostenesa jest efektywnym algorytmem wyznaczenia zbioru liczb pierwszych nie większych niż dana liczba naturalna *n*. Najpierw zbiór *S* składa się z liczb naturalnych większych niż 1 i nie większych niż *n*. Zbiór ten traktujemy jako kandydatów na *„pierwszość”*. Następnie usuwamy z *S* wielokrotności wszystkich liczb (pierwszych) większych niż 1 i nie większych niż pierwiastek z *n*. Końcowa zawartość zbioru *S* zawiera liczby pierwsze z zakresu [2; *n*].

   Zapisz sito Erastotenesa w języku C/Python przyjmując, że zbiór *S* jest reprezentowany przez tablicę *s*, gdzie *s*[*i*] równe 1 oznacza, że $i \in S$ oraz *s*[*i*] równe 0 oznacza, że $i \notin S$. Podaj też specyfikację, z którą zgodne jest Twoje rozwiązanie.

6. [**2**] Podaj rozwiązanie poprzedniego zadania tak, by dla liczb $m<n$ wyznaczane były wszystkie liczby pierwsze z przedziału [m, n]. Przy założeniu, że spełniony jest warunek

$$m < n < m+10\,000 < 100\,000\,000,$$

   Twoje rozwiązanie może korzystać ze stałej liczby tablic, gdzie każda z tablic ma nie więcej niż 10 000 elementów. Ponadto złożoność czasowa Twojego rozwiązania powinna zależeć od różnicy $m - n$ a nie liniowo (lub dla funkcji ponadliniowej) od samej wartości *n*.