Lista 8

Wednesday, 7 December 2022 02:41

Wdi-Lista08

2.3

$T(1) = 1$

$T(n) = 2T(n/2) + 1 \quad (n = 2m)$

$T(n) = T((n-1)/2) + T((n+1)/2) + 1 \quad (n = 2m+1)$

$T(16) = 31$ $\quad T(16) = 31$

$\hookrightarrow 2T(8) + 1$

$\hookrightarrow 2T(4) + 1$

$\hookrightarrow 2T(2) + 1$

$\hookrightarrow 2T(1) + 1$

$\hookrightarrow 1$

n postaci 2^k $k \in \mathbb{N}$

$n = 2^k$

$T(n) = 2T(n/2) + 1 = 2[2T(n/4) + 1] + 1 =$

$= 2[2[2T(n/8) + 1] + 1] + 1 = 2[4T(n/8) + 2 + 1] + 1 =$

$= 8T(n/8) + 4 + 2 + 1 = \ldots = 2^k \; T(n/2^k) + 2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0 =$

$= \underbrace{2^k + 2^{k-1} + 2^{k-2} + \ldots}_{} = n + \frac{2^k - n}{1} = 2n - 1$

PORTRYOWANIE

Instytut Informatyki UWr

Wstęp do informatyki

Lista 8

Dla zdefiniowanych rekurencyjnych podanych w zadaniach 1, 2 i 3 wykonaj polecenia:

- Wyznacz wartość $T(16)$.
- Rozwiąż zdefiniowaną metodą podstawienia. W zadaniach 2, 3 wystarczy, że uzyskasz rozwiązanie dla $n = 2^k$ i naturalnego k.
- Podaj przykład algorytmu przetwarzającego n danych wejściowych na takich (lub na liście rozmiarowej), którego złożoność czasowa jest opisana zadanym w zadaniu równaniem rekurencyjnym.

(1) $T(1) = 1$ — oczywiste. Czyli szybkość wykonywania (potrzebna ilość kroków)

$T(n) = T(n-1) + n$ dla $n > 1$

(1) $T(1) = 0$

$T(n) = T(n/2) + 1$ dla parzystego $n > 1$ — wyjątkowe kiedy trzeba wykonać 2 kroki na n

$T(n) = T(n-1/2) + 1$ dla nieparzystego $n > 1$ (własność logarytmiczna)

$T(n) = 2T(n/2) + 1$ dla parzystego $n > 1$ — oczywiste: wykonamy wykonywanie, potem wykonamy wykonywanie

$T(n) = T(n-1/2) + T(n+1/2) + 1$ dla nieparzystego $n > 1$ — potem wykonywanie

4. [1] Uzasadnij, że asymptotycznie (czyli w notacji dużego-O) funkcje podane w zadaniach 1 i 2 ograniczone przez funkcje wymienione w tych zadaniach metodą podstawienia dla każdego naturalnego argumentu n, nie tylko dla potęg dwójki.
Dodatkowo, Można się zadowolić, że podane funkcje są monotoniczne i wykorzystać ten fakt.
5. [2] Rozwiązny następujący algorytm wyznaczania największej liczby w ciągu n-elementowym $a_1, a_2, \ldots, a_n$ liczb całkowitych. Algorytm dzieli ciąg na dwa podciągi (przeciwstawiamy, że n jest potęgą dwójki) ciąg na dwa podciągi (przeciwstawiamy, że n jest potęgą dwójki), wyznaczamy minima tych dwóch ciągów m_1 i m_2 a następnie wybieramy mniejszą z liczb m_1, m_2.
   a. Napisz funkcję rekurencyjną realizującą ten algorytm.
   b. Podaj zależność rekurencyjną opisującą liczbę porównań elementów z ciągu wykonywanych przez zaproponowany algorytm. Wskazówka: zależność dla n postaci $n = 2^k$ i naturalnego k. Wynik powinny z liczb pierwszych jedynie tego samego elementu.
   Przyjmij, że procedura `podział` wybiera pierwszy element rozważanego podciągu jako pierwszy element do podziału (pivot).
   (1) Sortujemy ciąg złożone z 10, 100, 1000 elementów. Dla każdej z tych wartości n oraz
   a. głębokość drzewa wywołań rekurencyjnych, gdy stosujemy sortowanie przez scalanie;
   b. wysokość (maksymalną) możliwą głębokość drzewa wywołań rekurencyjnych, gdy sortujemy stosując sortowanie szybkie (Quicksort).
   Podaj zależność rekurencyjną, którą wartością dla danego n jest liczba poziomów drzewa wywołań rekurencyjnych sortowania przez scalanie ciągu n-elementowego.
   (1) Zajmowaną się z problemem wież z Hanoi. Sformułuj jego specyfikację i przedstaw jego rekurencyjne rozwiązanie w pseudokodzie oraz w postaci funkcji w języku C/Python.

Zadania dodatkowe, niedotyczące (nie wliczają się do puli punktów do zdobycia na ćwiczeniach, punktacja została podana tylko jako informacja o trudności zadań wg wykładowcy)

9. [1] Podaj nierekurencyjny algorytm znajdowania najmniejszego i największego elementu w ciągu, w którym wykonywane jest co najwyżej $3/2$ n porównań elementów.
10. [R] Sortujemy metodą Quicksort ciąg złożony z 10 elementów. Przedstaw:
   a. drzewo wywołań rekurencyjnych dla przypadku, gdy algorytm będzie działał najdłużej (gdy mamy „najgorsze" wybór funkcji podziału).
   b. drzewo wywołań rekurencyjnych dla przypadku, gdy algorytm będzie działał najkrócej (gdy mamy „najlepsze" wybór funkcji podziału).
11. [2] Zaimplementuj algorytm Quicksort bez użycia rekurencji.
   Wskazówka: Musisz zaimplementować własną strukturę danych (np. stos), pozwalającą na zapamiętywanie i wykorzystanie w tym zadaniu.
   Uwaga: Jeśli to zadanie jest dla Ciebie zbyt łatwe, spróbuj zaimplementować Quicksort bez rekurencji, używając jedynie pamięci pomocniczej stałego rozmiaru. Takie rozwiązania już były w grupie.
12. [2] Zaimplementuj sortowanie przez scalanie bez użycia rekurencji. Poza funkcją scalającą dwa ciągi. Twoje rozwiązania powinno używać jedynie pamięci stałego rozmiaru.
13. [2] Podaj iteracyjne (nierekurencyjne) rozwiązanie problemu wież z Hanoi, wymagające jedynie pamięci pomocniczej stałego rozmiaru.
14. [N,5] Uzasadnij, że Quicksort (funkcja sort podana na wykładzie) nie spełnia własności stabilności. Stabilność oznacza, że jeśli w ciągu wejściowym są co najmniej dwa elementy o takich samych wartościach x, y takich, że $x \leq y$, sprawdź też, czy sytuacja taka może mieć miejsce w przypadku funkcji `podział` podanej na wykładzie.

2.1

$T(1) = 1 \quad n > 1$

$T(n) = T(n-1) + n$

$T(n-1) \Rightarrow T(n-2) + n-1$

$\hookrightarrow T(n-2) + n-2$

$\hookrightarrow T(n-3) + n-3$

$\hookrightarrow T(n-4) + n-4$

METODA PODSTAWIANIA

$T(n) = T(n-1) + n = [T(n-2) + n-1] + n = T(n-3) + n-2 + n-1 + n + T(n-4) + n-3 + n-2 + n-1 + n =$

$= T(n-4) + 4n - (3+2+1) = \ldots = T(n-(n-1)) + n + n-1 + n-2 + \ldots + n-(n-2) = n + (n-1) + (n-2) + \ldots + 2 + 1 = \frac{(n-1)}{2} \cdot n$

$T(1) = 1$

$T(n) = T(n-1) + n = [T(n-2) + n] + n = [T(n-3) + n-1] + n = \ldots = T(n-(n-1)) + (n-1) + (n-2) + \ldots + (n-(n-2)) = 1 + 2 + 3 + \ldots + (n-2) + (n-1) + n = \sum_{i=1}^{n} i$

2.2

$T(1) = 0$

- $T(n) = T(n/2) + 1$ dla parzystego $n > 1$
- $T(n) = T((n+1)/2) + 1$ dla nieparzystego $n > 1$

$T(16)$

$\hookrightarrow T(8) + 1$

$\hookrightarrow T(4) + 1$

$\hookrightarrow T(2) + 1$

$\hookrightarrow T(1) + 1$

$\hookrightarrow 0$

$T(16) = 4$

— wynikiem kiedy wykonane kilka kroków potęgą 2 i tzn. $2^k \geq n$

Kiedy po pewnym czasie kilka wykonań, zawsze dostaje do rozmiaru dodaje 1 $n = 2^k$

$T(n) = T(n/2) + 1 = T(n/4) + 1 + 1 = T(n/8) + 1 + 1 + 1 = \ldots =$

$= T(n/2^{k-1}) + (k-1) = T(n/2^k) + (k-1) = 1 + (k-1) = k = \log n = O(\log n)$

2.5

```
def minimum(tab):
  if len(tab)==1:
    return tab[0]
  tab1 = tab[:len(tab)//2]
  tab2 = tab[len(tab)//2:len(tab)]
  if minimum(tab1) < minimum(tab2):
    return minimum(tab1)
  else:
    return minimum(tab2)
```

$T(1) = 1$

$\hookrightarrow T(2) + 1$

$\hookrightarrow T(4) + 1$

$\hookrightarrow T(8) + 1$

$\hookrightarrow T(16) + 1$

$\hookrightarrow T(32) + 1$

$n/2 + n/4 + 4 + 5 + 6 + \ldots + n/8 = \frac{n+6}{8} \cdot \frac{n}{2} = n/16$

2.6

a) $[2, 3, 3, 3, 1]$

$x \quad y$ — zamienić to 0

Jest to zawsze możliwość

$\Rightarrow$ wystarczy ze będzie mnie o wystarczy w TABLICE i $(j \vee)$

funkcja podział ze sobie

$\Rightarrow$ i j y w sport

wszystkie i j y x sport

kiedy x kiedy inne

PIVOTEM, typ x nie x sport

$\Rightarrow$ sport

$\Rightarrow$ $s[l,r]$

def sport (l, r):

if $[l < r]$:

$s = \text{podział}(l, r) \Rightarrow s$ — miejsce od którego wszystkie są większe od pivota

sport $(l, s-1)$

sport (s, r)

```
podział(int l, int p, int a[])
{ int i,j,y,x;
  x=a[l]; i=l; j=p;
  while (i<j)
  { while (a[j]>x) j--;
    while (a[i]<x) i++;
    if (i<j)
    { y=a[j]; a[j]=a[i]; a[i]=y;
      i++; j--; }
  }
  return j;
}
```

b) $[2, 1, 3, 1, 2]$

podział (pierwszy wystąpienia)

$[1, 1, 2, 3, 2]$

— możemy tylko mamy wymienić 2 typu (na typu wszystkie liczby, by kiedy wystąpienia liczby, i te przyjść y, zamieniać z jest większe od liczby, tablica na stałe jest przyjść)

c) $[2, 2, 2, 2, 2]$ — liczy jest wszędzie to pivot, i możemy w nim

— możemy wszystkie funkcji podział zamienić te same elementy w tablicy a ze ostrzę i zwiększ o i ciągle odczyt „zamiast" zapełnie wszystkiego ile podmiana i wynik $\Rightarrow$ obserwacje

2.7

```
def mergesort(a, l, r):
#sortuje a[l..r]
 if r-l>0:
   s=(l+r)/2
   mergesort(a,l,s)
   mergesort(a,s+1,r)
   merge(a,l,s+1,r)
```

$\log n$ określa głębokość drzewa (ile razy dzielimy na dwa)

$n \log n$ ✓ $\quad n^2$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

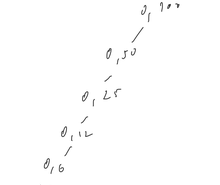
$2^7 =$

a) $MS(0, 10)$

$\Rightarrow$ GŁĘBOKOŚĆ – 4

$\Rightarrow$ 100 GŁĘBOKOŚĆ – 7

$\Rightarrow$ 1000 GŁĘBOKOŚĆ – 10



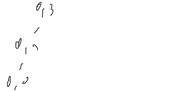
b) $\Rightarrow 10$ 9

$\Rightarrow 100$ 99

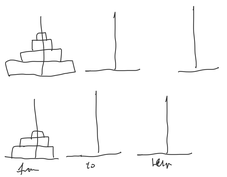
$\Rightarrow 1000$ 999

$D(1) = 0$

$D(n) = D(\frac{n}{2})$



2.8.




```
def hanoi (from, to, help, disk):
    if disk == 1:
        print("Move", disk, from: from[-1], to: to[-1])
    hanoi (from, help, to, disk-1)
    print("Move", disk, from: from[-1], to: to[-1])
    hanoi (help, to, from, disk-1)
hanoi ('A','B','C', 3)
```