Metody Programowania (4) - Mergesort, problem plecakowy, problem wież Hanoi. Zadanie domowe.

Wojciech Szlosek

April 2020

1 Rekurencja ogonowa

Rekurencja ogonowa to rodzaj rekurencji, w której ostatnia operacja wykonywana przez funkcję to rekurencyjne wywołanie samej siebie lub zwrócenie końcowego wyniku. Porównajmy ją ze "zwykłą" rekurencją na bazie poniższych przykładów funkcji obliczającej silnię:

1.1 Kod rekurencyjny:

```
int silnia(int n){
  if (n==0){
    return 1;
  }
  return n*silnia(n-1);
}
```

1.2 Wersja z rekurencją ogonową:

```
int silnia(int n, int akumulator=1){
  if (n==0) {
    return 1;
  }
  return silnia(n-1, akumulator*n);
}
```

Zauważmy, że w tej wersji użyliśmy dodatkowej zmiennej "akumulator" - to taka dodatkowa opcjonalna zmienna o danej początkowej wartości (u nas 1).

2 "Dziel i zwyciężaj"

Metoda "dziel i zwyciężaj" (łac. divide et impera - dziel i rządź) to jedna z głównych metod rozwiązań wielu problemów informatyki, prowadząca do bardzo efektywnych rozwiązań. W strategii tej problem dzieli się rekurencyjnie na

dwa lub więcej mniejszych podproblemów, tak długo, aż fragmenty staną się wystarczająco proste do rozwiązania. Z kolei rozwiązania otrzymane dla podproblemów scala się, uzyskując rozwiązanie całego zadania.

Przykładami algorytmów korzystających z tej metody są między innymi: wyszukiwanie binarne i sortowanie przez scalanie (mergesort).

2.1 Mergesort

Dla ścisłości, kod:

```
public static void mergeSort(int[] array, int low, int high) {
                if (high <= low) return;
                int mid = (low+high)/2;
                mergeSort(array, low, mid);
                mergeSort(array, mid+1, high);
                merge(array, low, mid, high);
        }
        public static void merge(int[] array, int low, int mid, int high) {
                int leftArray[] = new int[mid - low + 1];
                int rightArray[] = new int[high - mid];
                for (int i = 0; i < leftArray.length; i++)
                        leftArray[i] = array[low + i];
                for (int i = 0; i < rightArray.length; i++)
                        rightArray[i] = array[mid + i + 1];
                int leftIndex = 0;
                int rightIndex = 0;
                for (int i = low; i < high + 1; i++) {
                        if (leftIndex < leftArray.length && rightIndex < rightAr
                                if (leftArray[leftIndex] < rightArray[rightIndex
                                        array[i] = leftArray[leftIndex];
                                        leftIndex++;
                                }
                                else {
                                        array[i] = rightArray[rightIndex];
                                        rightIndex++;
                                }
                        }
                        else if (leftIndex < leftArray.length) {
                                array[i] = leftArray[leftIndex];
                                leftIndex++;
                        }
                        else if (rightIndex < rightArray.length) {
                                array[i] = rightArray[rightIndex];
                                rightIndex++; } } }
```

Jednakże ten długi kod to tylko implementacja, rozważmy metodę "dziel i zwyciężaj" poprzez ten algorytm.
Wyróżnić można trzy podstawowe kroki:

- Podział zestawu danych na dwie równe części
-Zastosowanie sortowania przez scalanie dla każdej z nich oddzielnie, chyba że pozostał już tylko jeden element
-Połączenie posortowanych podciągów w jeden posortowany ciąg

Reasumując, dzięki podzieleniu zestawu danych mamy ogólne uproszczenie i zwiększenie wydajności.

Złożoność czasowa sortowania przez scalanie to $O(n \log n)$. Rozważmy to. Bez straty ogólności załóżmy, że długość ciągu, który mamy posortować jest potęgą liczby 2. Analiza złożoności daje, że: $T(n) = 0$, gdy n = 1; $T(n) = 2T(\frac{n}{2}) + n$, gdy $n > 1$. Ponieważ: mamy zawsze dwa podzadania wielkości $\frac{n}{2}$; koszt podziału minimalny (zero porównań). koszt scalania to liczba porównań podczas scalania ciągów posortowanych, równa sumie ich długości – tutaj n. Po rozwiązaniu tegoż rekurencyjnego równania otrzymujemy rzeczywiście: $T(n) = O(nlogn)$.

3 Problem plecakowy

W najprostszej postaci problem plecakowy polega na podejmowaniu prób umieszczenia w plecaku elementów o różnej wadze, tak aby sumaryczna waga plecaka osiągnęła pewną określoną wartość. Przy okazji nie ma wymogu, by w plecaku zostały umieszczone wszystkie elementy. Przykładowy algorytm postępowania został przedstawiony na wykładzie.
Rozważmy przykład: załóżmy, że sumaryczna waga plecaka ma wynosić 20 kilogramów, a do dyspozycji mamy elementy o wagach 9, 8, 5, 4 oraz 3 -kilogramów.

KROK 1:

9 waga docelowa: 20, 9 to za mało

KROK 2:

9, 8 waga docelowa: 11, 8 to za mało

KROK 3:

9, 8, 5 waga docelowa: 3, 5 to za dużo

KROK 4:

9, 8, 4 waga docelowa: 3, 3 to za dużo

KROK 5:

9, 8, 3 waga docelowa: 3, 3 pasuje - KONIEC

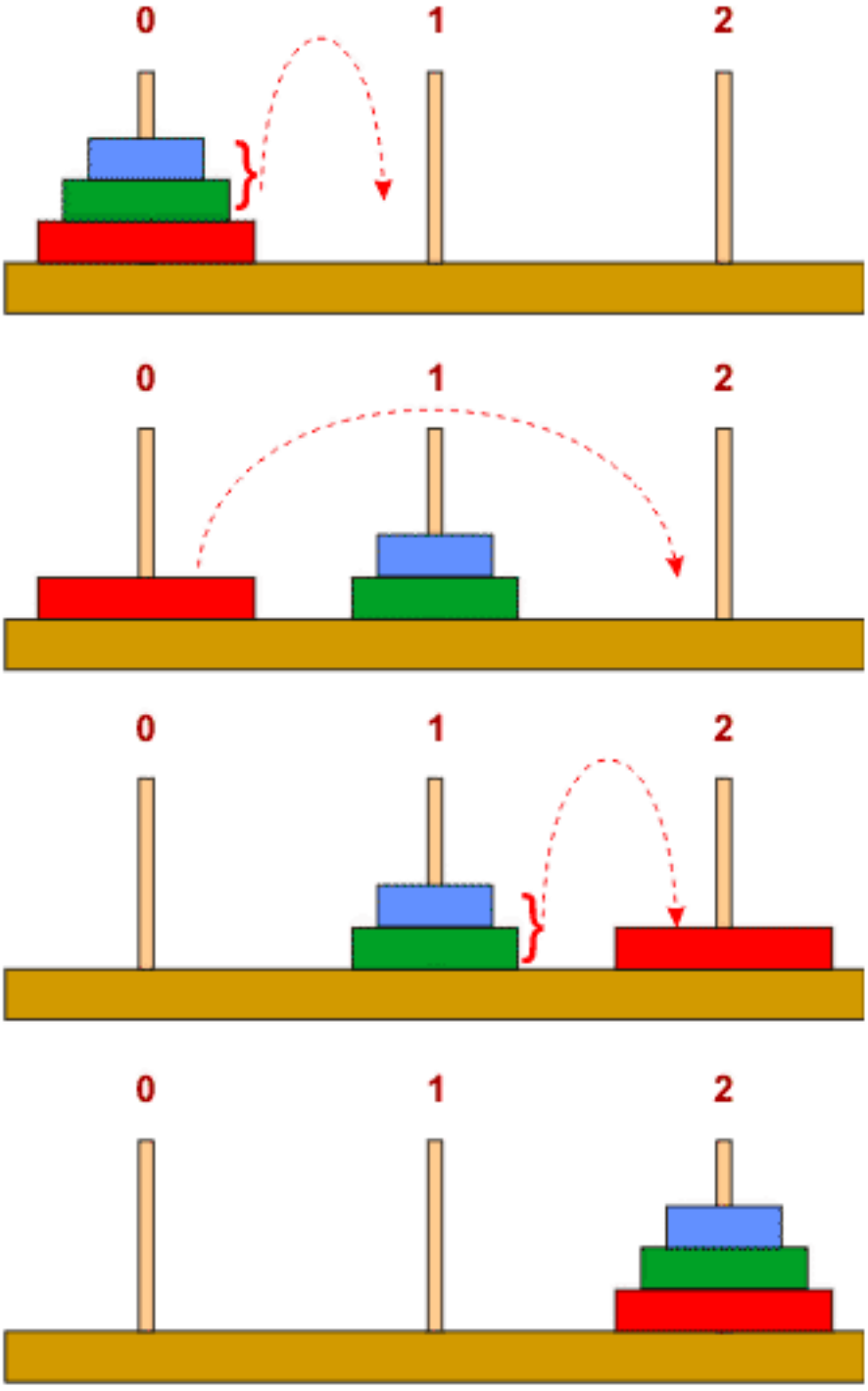
4 Problem wież Hanoi

Na wieży A jest n krążków o różnych średnicach, uporządkowanych rosnąco w dół wieży. Należy przenieść n krążków z wieży A na wieżę B, z pomocniczą wieżą C, przenosząc krążki pojedynczo, nie kładąc większego na mniejszym.

Przykładowy algorytm postępowania:

(1) Przenieś poddrzewo składające się z n-1 krążków z wieży A na wieżę B
(2) Przenieś ostatni (największy krążek) z A na wieżę docelową C
(3) Przenieś poddrzewo z wieży B na C

Rozważmy rozwiązanie przykładowego problemu tego typu za pomocą poniższej ilustracji:


0
1
2
0
1
2
0
1
2
0
1
2