Metody Programowania - wykład 1, 2, 3, 4 (Kadane, szukanie binarne, proste sortowanie). Krótkie opracowanie. Zadanie domowe.

Wojciech Szlosek

March 2020

1 Algorytm Kadane - prezentacja

To algorytm obliczajacy maksymalna podtablice w danej tablicy. Metoda ta oblicza maksymalna podtablice kończaca sie w i, majac obliczona podtablice kończaca sie w $i-1$.

Złożoność: $O(n)$.

1.1 Kod

```
int beginMax = 0;
int endMax = 0;
int sumMax = 0;
int beginB = 0;
int curr = 0;

for(int i = 0; i<n; i++){
   curr += tab[i];

   if(curr<0){
       curr = 0;
       beginB = i+1;
   }

   else if(curr>sumMax){
       sumMax = curr;
       beginMax = beginB;
       endMax = i;
   }
}
```

1.2 Przykład:

Dany jest ciag: -8, 3, -1, 2, -6, 7.

sumMax	curr	beginMax	endMax	i	tab[i]	beginB
0	0	0	0	0	-8	0
-	-8/0	-	-	1	3	1
3	3	2	2	2	-1	1
-	2	-	-	3	2	1
4	4	2	3	4	-6	1
4	-2/0	-	-	5	7	5
7	7	5	5	-	-	5

$(odp.) max = 7$

2 Wyszukiwanie binarne - prezentacja

Algorytm szuka danego elementu w tablicy uporzadkowanej (posortowanej). Algorytm jest realizowany metoda "dziel i zwyciężaj". Dzieli on tablice na mniejsze podtablice do momentu wyszukania pozycji (lub nie w przypadku gdy taki element nie istnieje) elementu szukanego. Przyjmujemy, że u nas tablica nie ma duplikatów.

Złożoność: $O(logn)$.

2.1 Przykład:

Mamy odnaleźć liczbe $x = 6$ w ciagu: 1, 2, 5, 6, 7, 44, 98, 99, 101, 103. Przyjmijmy, że "wybieramy" liczbe po lewej stronie (w przypadku gdy trafimy na ich parzysta ilość).

$$1, 2, 3, 5, 6, 7, 44, 98, 101, 103$$

$$1, 2, 3, 5, 6, \underline{7}, 44, 98, 99, 101, 103$$

$$1, 2, \underline{3}, 5, 6, 7, 44, 98, 99, 101, 103$$

$$1, 2, 3, \underline{5}, 6, 7, 44, 98, 99, 101, 103$$

$$1, 2, 3, 5, \underline{6}, 7, 44, 98, 99, 101, 103$$

(KONIEC)

2.2 Posłowie

Złożoność tego algorytmu (pesymistycznie) to $O(n)$, jednakże podajac przykład, gdzie mamy odnaleźć liczbe 5 w ciagu: 1, 2, 5, 7, 11, zauważamy, że już za pierwszym razem te liczbe odnajdziemy (bo jest na środku) - to przypadek optymistyczny. Przykład pesymistyczny wystepuje miedzy innymi w przypadku, gdy nie szukamy liczby, która wystepuje w ciagu. Np.: szukamy liczby x = 3, w ciagu: 1, 7, 10, 15, 26.

3 Wyszukiwanie interpolacyjne - prezentacja

Algorytm szuka danego elementu w tablicy uporzadkowanej (posortowanej). Można powiedzieć, że jest to "ulepszona" wersja wyszukiwania binarnego. Jeśli założymy liniowy rozkład wartości elementów tablicy to możemy prezycyjniej wytypować pozycje poszukiwanego "klucza". W porównaniu do wyszukiwania binarnego, "wzór" na pozycje to nie średnia arytmetyczna (wzór na wykładzie). Wyszukiwanie interpolacyjne posiada klase czasowej złożoności obliczeniowej równa $O(loglogn)$ (średnia), zatem wyszukuje znaczaco szybciej w zbiorach o liniowym rozkładzie elementów niż wyszukiwanie binarne o klasie $O(logn)$.

3.1 Przykłady:

Przypadkiem optymistycznym jest np. wyszukanie elementu x = 15, w ciagu uporzadkowanym:

10, 11, 13, 13, 15, 16, 18, 19, 20, 22, 22, 23, 25, 27, 27, 28, 29, 30, 32.
10, 11, 13, 13, <u>15</u>, 16, 18, 19, 20, 22, 22, 23, 25, 27, 27, 28, 29, 30, 32.
(KONIEC! - za pierwszym razem)
Pesymistyczna złożoność to $O(n)$, przykład: wyszukać liczbe x = 3, w ciagu: 1, 2, 2, 7, 9. Nie znajdzie jej, a "zahaczy" o wszystkie!

4 Bubble Sort

Sortowanie babelkowe jest bardzo łatwe - sprawdzamy czy nastepny element tablicy jest wiekszy od aktualnego, jeżeli tak, to zamieniamy te elementy miejscami.

Złożoność: $O(n^2)$

4.1 Kod

```
void bubbleSort(int arr[])
   {
       int n = arr.length;
       for (int i = 0; i < n-1; i++)
           for (int j = 0; j < n-i-1; j++)
               if (arr[j] > arr[j+1])
               {
                   swap(arr[j],arr[j+1]);
               }
   }
```

Idea jest prosta, zapisujemy wartość danego elementu z tablicy i sprawdzamy aż do jej końca czy znajdzie sie element od niej wiekszy, jeśli to to ”swapujemy” te wartości (zamieniamy miejscami).

4.2 Przykład:

Opieramy sie na funkcji swap (zamiana wartości).

$$2, 4, 1, 3$$

$$2, 4, \underline{1}, \underline{3}$$

$$2, \underline{4}, \underline{1}, 3$$

$$\underline{2}, \underline{1}, 4, 3$$

$$1, 2, 4, 3$$

$$1, 2, \underline{4}, \underline{3}$$

$$1, \underline{2}, \underline{3}, 4$$

$$\underline{1}, \underline{2}, 3, 4$$

$$1, 2, 3, 4$$

KONIEC, wynik: uporzadkowany ciag.

5 Selection Sort

Kolejne sortowanie proste. Polega ono na wyszukaniu elementu majacego sie znaleźć na żadanej pozycji i zamianie miejscami z tym, który jest tam obecnie (ponownie swap). Operacja jest wykonywana dla wszystkich indeksów sortowanej tablicy. Uwaga: algorytm jest niestabilny.

Złożoność: $O(n^2)$

5.1 Kod

```
void selectionsort(int arr[])
    {
        int n = arr.length;

        for (int i = 0; i < n-1; i++)
        {
            //szukamy max w nieposortowanej tablicy
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            swap(arr[i],arr[min_idx]);
        }
    }
```

5.2 Przykład:

$Klamra - ozn. : gotowe$

$$6,4,8,1$$

$$6,4,\underline{8},\underline{1}$$

$$6,4,1,\underbrace{8}$$

$$\underline{6},4,\underbrace{1},\underbrace{8}$$

$$1,\underline{4},\underbrace{6},\underbrace{8}$$

$$\underline{1},\underbrace{4},\underbrace{6},\underbrace{8}$$

$$1,4,6,8$$

KONIEC, wynik: uporzadkowany ciag.