NTIN060 | Homework 01/3

Milan Wikarski (milan@wikarski.sk)

Zadanie

Pro posloupnost čísel `x[1],…,x[n]` a číslo `S` najděte dvojici indexů `i, j` takovou, že `S=x[i]+x[j]`.

Jednoduché riešenie

Jednoduché riešenie spočíva vo vyskúšaní všetkých možných kombinácií indexov.

```
ALGORITMUS: jednoducheRiesenie
-----------------------------------------------------------------
 INPUT: pole čísel x[1], x[2], ..., x[n]; číslo S
OUTPUT: dvojica indexov i, j alebo null, ak také indexy neexistujú

for ∀ i ∈ {1, 2, ..., n}:
  for ∀ j ∈ {i+1, i+2, ..., n}:
    if (x[i] + x[j] = S):
      return i, j

return null
```

Časová zložitosť

Tento algoritmus vyskúša `O(n*(n-1)/2)` dvojíc indexov. Časová zložitesť teda je `O(n^2)`.

Priestorová zložitosť

Priestorová zložistť je `O(1)`, keďže používame iba konečné množstvo pomocných premenných.

Riešenie pomocou doplnkov

Toto riešenie spočíva v tom, že si najprv vytvoríme pole doplnkov, kde na i-tom indexe bude číslo, ktoré musíme prirátať ku `x[i]`, aby sme dostali `S`. Inak povedané: `∀ i ∈ {1, 2, ..., n}: D[i] + x[i] = S`. Ďalej si opäť prejdeme celé pole x a budeme sa pozerať, či pole D obsahuje prvok `x[i]`. Ak ho obsahuje, funkcia `indexOf` nám vráti jeho index. Ak pole D obsahuje prvok `x[i]`, znamená to, že existuje nejaký prvok `x[j]` taký, že `x[i] + x[j] = S`.

```
ALGORITMUS: doplnkoveRiesenie
-----------------------------------------------------------------
 INPUT: pole čísel x[1], x[2], ..., x[n]; číslo S
OUTPUT: dvojica indexov i, j alebo null, ak také indexy neexistujú

D <- []                           ◃ pole doplnkov
for ∀ i ∈ {1, 2, ..., n}:
  D[i] <- S-x[i]                  ◃ doplnok do čísla S
for ∀ i ∈ {1, 2, ..., n}:
  j <- indexOf(D, x[i])           ◃ index prvky v poli alebo -1, ak pole prvok neobsahuje
  if (j > -1):
    return i, j

return null
```

Časová zložitosť

Najprv si inicializujeme pole doplnkov - na to náš algortimus musí spraviť `n` krokov.

Ďalej si opäť prejdeme celé pole `x` a pre každý prvok z `x` budeme hľadať index tohto prvku v poli `D`. Hľadanie indexu v neusporiadanom poli nám zaberie čas lineárny s dĺžkou pola, pretože musíme prejsť všetky prvky. Dokopy teda máme `n^2` krokov.

Spolu pre inicializáciu a hľadanie indexu v poli `D` máme `n^2 + n = n(n + 1)` krokov. Časová zložitosť algoritmu teda je `O(n^2)`.

Priestorová zložitosť

Okrem pomocných premenných tentokrát potrebujeme aj pole `D` dĺžky `n`, v ktorom budeme držať doplnky prvkov pola `x`. Priestorová zložitosť algoritmu je `O(n)`.

Možné zlepšenia

Časová zložitosť algoritmu sa dá zlepšiť voľbou vhodnej datovej štruktúry pre uchovávanie doplnkov.

Binárne rozhodovacie stromy

Namiesto pola by sme si mohli zvoliť binárny rozhodovací strom, ktorý by sme hĺbkovo vyvažovali podľa potreby. Namiesto hodnôt by sme v ňom držali dvojice `(hodnota, index)`. Strom by bol zoradený podľa hodnôt. Vkladanie prvku v takom strome trvá `O(log(n))`. Implementácia funkcie `indexOf` v tomto strome by mala časovú zložitosť `O(log(n))`. Mali by sme `n` vložení a `n` volaní `indexOf`. Časová zložitosť by teda bola `O(2n*log(n)) = O(nlog(n))`.

Hashovacia tabuľka

Ešte lepšou voľbou by bola hashovacia tabuľka, ktorá pri voľbe vhodného hashovacieho algoritmu zvládne vkladanie aj hľadanie prvkov v konštantnom čase; ak počet 'priehradok' tabuľky bude rovný `n`. V takom prípade by sme inicializáciu pola `D`, ako aj následné hľadanie v ňom stihli v čase lineárnom podľa `n`. Namiesto hodnôt by sme (rovnako ako pri rozhodovacích stromoch) ukladali dvojice `(hodnota, index)`. Časová zložitosť by potom bola `O(2n) = O(n)`.