Lista 0

tags: `AiSD`, `lista0`

Zadanie 3

```
Procedure BubbleSort(T[1..n])
    for i <- 1 to n do
        for j <- i+1 to n do
            if T[i] > T[j]:
                t = T[i]
                T[i] = T[j]
                T[j] = t
```

Koszt procedury nie zależy od ułożenia elementów w wejściowej tablicy. Dla każdego i wykonujemy $n-i$ przejść, czyli mamy $n + \sum_i^n n-i = n+ n(n-1)/2$, zatem pesymistyczna złożoność BubbleSorta to $O(n^2)$. Warto tutaj zauważyć, że BubbleSort wykonuje bardzo dużo porównań (po jednym na każde j, czyli aż $n(n-1)/2$). Algorytm jest algorytmem stabilnym tj. zachowujemy ustawienie między elementami na tym samym miejscu w porządku. Ostatecznie jednak wypada gorzej od Selection i/lub Insertion sorta w każdej kategorii (liczba porównań, złożoność, liczba zamian)

Zadanie 4

Algorytm mnożenia liczb po rosyjsku (Dzielimy z podłogą lewą liczbę przez 2 aż otrzymamy jeden, sjednocześnie mnożąc prwą stronę. Następnie wycinamy wiersze, w których lewa strona jest parzysta)

przykład dla a = 134, b = 95:

a	b
13	238
~~6~~	~~476~~
3	952
1	1904

Sumując prawą stronę otrzymujemy 3094 = a*b

Dowód Zacznijmy od przedstawienia tych liczb w systemie binarnym a = 1101, b = 11101110

a	b
1101	11101110
~~110~~	~~111011100~~
11	1110111000
1	11101110000

Zauważmy, że dzielenie z podłogą to tak naprawdę przesunięcie bitowe w prawo. Natomiast sumowanie prawej strony odpowiada mnożeniu pisemnemu:

```
        1 1 1 0 1 1 1 0   (238)
×               1 1 0 1   (13)
-------------------------
        1 1 1 0 1 1 1 0   (238)
      0 0 0 0 0 0 0 0 0   (0)
    1 1 1 0 1 1 1 0 0 0   (952)
+ 1 1 1 0 1 1 1 0 0 0 0   (1904)
-------------------------
1 1 0 0 0 0 0 1 0 1 1 0   (3094)
```

Pseudokod

```
Procedure russian_multiplication(a,b)
    pow <- 1
    result <- 0
    while a > 0 do
        if a mod 2 == 0 do
            result <- result + b * pow
        pow <- pow * 2
        a <- a / 2
```

Złożoność czasowa

Kryterium jednorodne kosztów Instrukcje wewnątrz pętli kosztują nas stałą liczbę operacji w maszynie ram, a sama pętla obróci się $O(log_2{a})$ razy, bo w każdym obrocie dzielimy a przez 2.

Kryterium logarytmiczne kosztów Złożoność operacji wykonywanych na a jest rzędu $O(log_2a)$. Co więcej pow < a, czyli złożoność $b * pow$ to $log_2b + log_2a$, czyli $O(log_2ab)$. Jest to najrdoższa operacja w pętli która obróci się $O(log_2a)$. Razem mamy $O(log_2a * log_2ab)$

Złożoność pamięciowa **Kryterium jednorodne kosztów** Potrzebujemy stałej liczby zmiennych, czyli O(1).

Kryterium logarytmiczne kosztów Wciąż potrzebujemy stałej liczby zmiennych, ale musimy oszacować najdłuższą w zapisie binarnym z nich. Będzie to wynik po wszyskich iteracjach. Skoro w każdej iteracji $result < b * pow$ oraz pow jest mniejsze od początkowego a to będzie miała ona długość $O(log_2{ab})$.

Zadanie 6

```
 x = 0
while |A| > 0 do
    a <- losowy element z A:
    A <- A \ {a}
    x <- (x + (a mod 2)) mod 2
return x
```

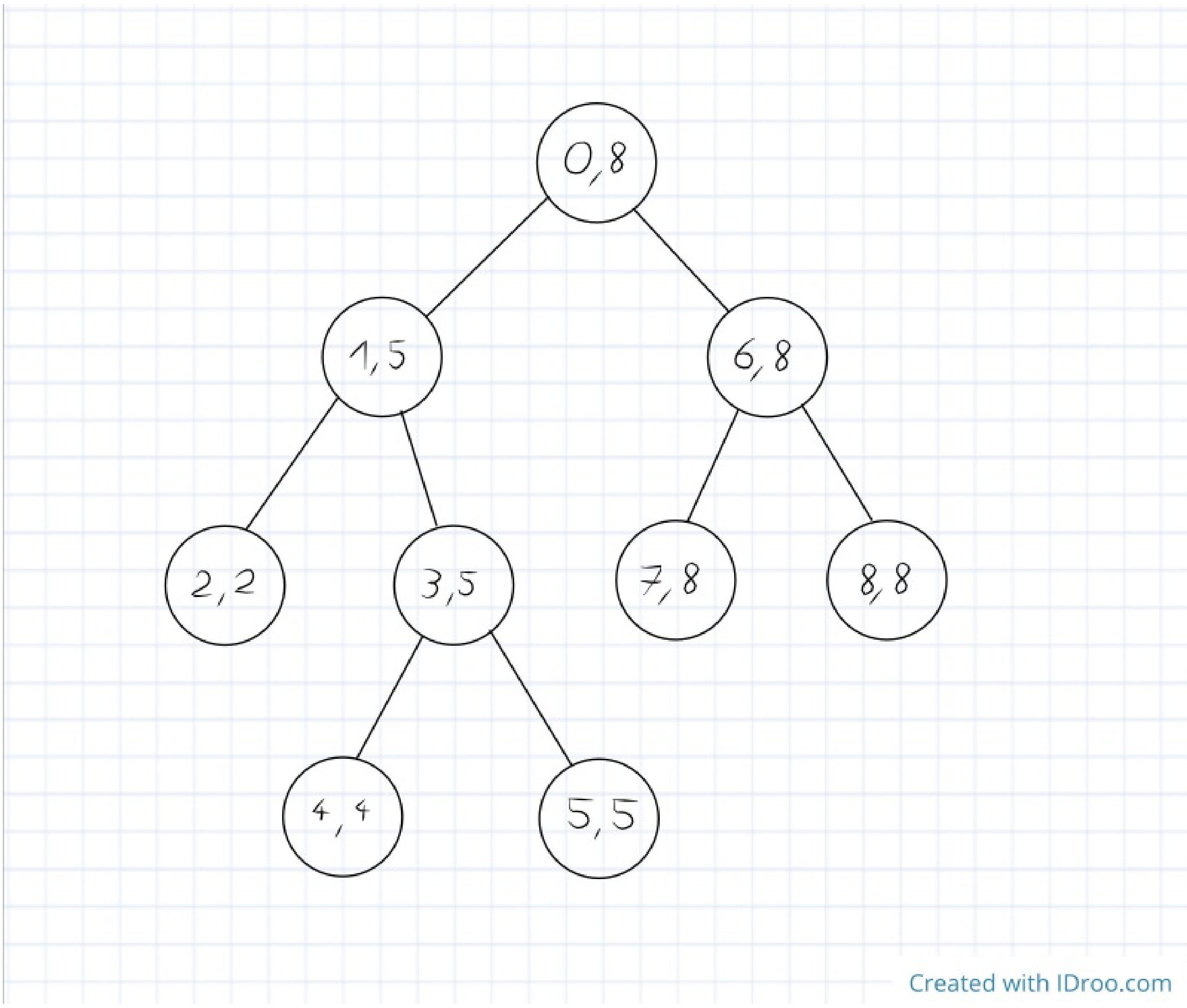
(a + b) mod 2 = (a mod 2 + b mod 2) mod 2

jeśli A = {a, b, c}, to x = (+-a +- b +- c) mod 2 = ((+-a +- b) +- c) mod 2 = ((+-a +- b) mod 2 +- c mod 2) mod 2 = ((+-a mod 2 +- b mod 2) mod 2 +- c mod 2) mod 2

Zadanie 7

Zakładam, że wierzchołki są ponumerowane i nie trzeba ich mapować

```
def solve(G, pairs):

    def build_tree():
        tree[n]
        for (v, p) in G:
            tree[p].append(v)
        return tree

    def findRoot():
        vertices[n]           //pusta tablica o rozmiarze n
        for (v, u) in G:
            vertices[u] = 1
        for i from 0 to n:
            if vertices[i] == 0:
                return i

    tree = build_tree();
    time = 0; timers[n][2] //pusta tablica o rozmiarze nx2
    def DFS(v):
        time += 1
        timers[v][0] = time
        for u in tree[v]:
            DFS(u)
        timers[v][1] = time

    DFS(findRoot())
    def onPath(v, u):
        return timers[u][0] >= timers[v][0] and timers[u][1] <= timers[v][1]

    output = list()
    for p in pairs:
        output.append(onPath(p[0], p[1]))
    return output
```


0,8
1,5
6,8
2,2
3,5
7,8
8,8
4,4
5,5
Created with IDroo.com