Omówienie zadania ”Liczba Fibonacciego”

Adam Furtak

2018-11-12

1 Wprowadzenie

Liczby fibonacciego definujemy nastepujaco:

$$fib(x) = \begin{cases} fib(x-1) + fib(x-2) & x \geq 2 \\ 1 & x = 1 \\ 0 & x = 0 \end{cases}$$

W zadaniu jesteśmy pytani o n-ta liczbe fibonacciego.

2 Rozwiazanie naiwne

Po zobaczeniu definicji liczb fibonacciego pierwszym rozwiazaniem jakie przychodzi do głowy jest zapewne rozwiazanie za pomoca rekurencji.

```
int fib(int x) {
        if (x == 0)
                return 0;
        else if (x == 1)
                return 1;
        else
                return fib(x - 1) + fib(x - 2);
}
```

Zastanówmy sie natomiast ile operacji potrzebuje wykonać nasz algorytm. Wywołanie funkcji $fib(n)$ powoduje powstanie binarnego drzewka wywołań (każda funkcja wywołuje sama siebie 2 razy) o głebokości około n. Zatem wnioskujemy, że złożoność $fib(n)$ to około $O(2^n)$. Da sie zrobić to szybciej.

3 Rekurencja z zapamietywaniem

Głównym problemem poprzedniego rozwiazania był fakt, że nasza funkcja liczyła po kilka razy ta sama wartość $fib(n)$. Spróbujmy pozbyć sie tego problemu.

```
int fib(int x) {
        if (F[x] != -1)
                return F[x];
        else {
                if (x == 0)
                        return 0;
                else if (x == 1)
                        return 1;
                else {
                        F[x] = fib(x - 1) + fib(x - 2);
                        return F[x];
                }
```

```
        }
}
```

W tym rozwiazaniu, przy każdym wywołaniu funkcji sprawdzamy czy znamy już wartość x-tej liczby fibonacciego ($F[x] \neq 0$). Jeśli tak, to zwracamy ja, bez dodatkowego wywołania rekursji. Jeśli nie, wyliczamy ja, zapamietujemy i zwracamy jej wartość. Zapobiega nam to tworzenia drzewa binarnego wywołań i zbija złożoność z $O(n^2)$ do $O(n)$, kosztem dołożenia liniowej pamieci.

4 A gdyby tak iteracja?

Spróbujmy teraz zapisać algorytm nie używajac rekurencji.

```
int fib(int x) {
    F[0] = 0;
    F[1] = 1;
    for (int i = 0; i <= x; i++)
        F[i] = F[i - 1] + F[i - 2];
    return F[x];
}
```

Co nam to dało? Na razie niewiele. Ale przyjżyjmy sie naszej funkcji i zastanówmy sie co można jeszcze poprawić. Łatwo zauważyć, że nie musimy zapamietywać wszystkich poprzednich liczb ciagu. Wystarczy, że bedziemy pamietać tylko 2 elementy.

```
int fib(int x) {
    int a = 0;
    int b = 1;

    if (x == 0)
        return 0;
    if (x == 1)
        return 1;

    for (int i = 2; i <= x; i++) {
        int temp = a + b;
        b = a;
        a = temp;
    }
    return a;
}
```

I tak oto otrzymujemy rozwiazanie w złożoności czasowej $O(n)$) i pamieciowej $O(1)$.