Level UP! – Optymalizacja kodu (sheadovas/artykuly/programowanie/level-up-optymalizacja-kodu/)

Paź 07, 2015 / Programowanie (sheadovas/category/artykuly/programowanie/)

Kilka słów o algorytmach, optymalizacji i złożoności obliczeniowej.

W tym wpisie chciałbym Wam pokazać jak wygląda programowanie z punktu widzenia osoby, która umie już nieco więcej (aż chce się nawiązać do gier: „ma wyższy poziom umiejętności w programowaniu”), czyli nie tylko potrafi już rozwiązywać problemy, ale też chce rozwiązywać je najwydajniej jak jest tylko możliwe (zgodnie z mottem: *do it better*).

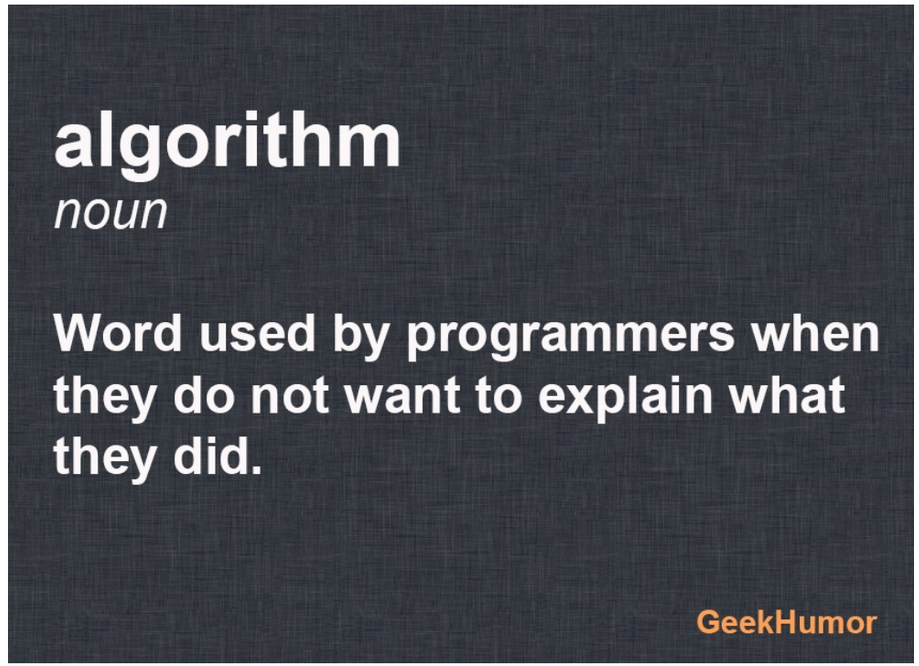
Tego artykułu nie należy traktować jako zbiór całej wiedzy na temat optymalizacji, a raczej jako krótkie wprowadzenie (żeby być dokładniejszym: to jest wstęp do wstępu).

Proszę o wytknięcie wszelkich błędów czy niejasności w komentarzach pod tym wpisem.

Algorytmy

Pseudo-Kod = Pseudo-Programista ~M. Adamczyk

Żeby móc w ogóle rozmawiać o optymalizacji kodu musimy wyjaśnić sobie kilka potencjalnych niejasności: czym jest algorytm?



Na poważniej:

(Wikipedia)

Algorytm – jednoznaczny przepis obliczenia w skończonym czasie pewnych danych wejściowych do pewnych danych wynikowych.

Zazwyczaj przy analizowaniu bądź projektowaniu algorytmu zakłada się, że dostarczane dane wejściowe są poprawne, czasem istotną częścią algorytmu jest nie tylko przetwarzanie, ale i weryfikacja danych.

Zgodnie z założeniem o jednoznaczności – dla identycznego zestawu danych początkowych, algorytm zdefiniowany klasycznie zawsze zwróci identyczny wynik.

Innymi słowy jest to zbiór instrukcji potrzebnych, których wykonanie jest potrzebne do rozwiązania określonego problemu, co ważne: algorytmy zazwyczaj zapisuje się przy użyciu pseudo-kodu, dzięki czemu zwiększa to uniwersalność algorytmu (przy okazji utrudniając pracę programistom).

Złożoność obliczeniowa

> The First Rule of Program Optimization: Don’t do it. The Second Rule of Program Optimization (for experts only!): Don’t do it yet. ~ Michael A. Jackson

Kolejnym dość ważnym pojęciem jest złożoność obliczeniowa, która pozwala nam określić jak zachowuje się algorytm w ściśle określonej sytuacji. Niestety nie zawsze da się powiedzieć o jakimś algorytmie, że „jest najlepszy w absolutnie każdej sytuacji”, bo są one najczęściej pisane pod określony zestaw danych (o czym jeszcze sobie powiemy później).

Wpływ na to jak dobry jest algorytm ma to jak się zachowuje przy różnych ilościach danych: ile czasu potrzebuje do wykonania zadania i ile pamięci potrzebuje do rozwiązania problemu, np niektóre algorytmy zużywają mało pamięci i szybko się wykonują dla małych zestawów, danych ale mimo małego zużycia pamięci czas potrzebny do ukończenia rośnie wraz z ilością danych; albo jeszcze inaczej: czas i pamięć rosną wraz z ilością danych.

Analizy algorytmu dokonuje się dla:

- najlepszego przypadku – najlepsza sytuacja dla algorytmu, gdzie dane są „podłożone” pod algorytm;
- średniego przypadku – przypadek, który zdarza się najczęściej, normalny sposób działania algorytmu;
- najgorszym przypadku – wydajność algorytmu dla tych danych jest najgorsza.

Do określania złożoności algorytmu informatycy upodobali sobie **notację dużego O.** Jest to wizualizacja zachowania algorytmu za pomocą funkcji, która pozwala zrozumieć jak zachowuje się algorytm, np zapis $O(n)=Cn$, oznacza że dla n danych wejściowych algorytm potrzebuje Cn kroków dla ustalonej liczby C.

Dla uproszczenia wizualizowania sobie tego jak działa algorytm to aproksymujemy go do powszechnie znanych funkcji matematycznych (w przykładzie powyżej złożoność była liniowa), poniżej przedstawiam zestawienie najpopularniejszych złożoności od najlepszych do najgorszych:

1. stała
2. logarytmiczna
3. liniowa
4. n log(n)
5. kwadratowa
6. wykładnicza

Optymalizacja

> Poznaj swojego wroga (…) ~Sun Tzu

Jak mówiłem wcześniej, w większości sytuacji (jeżeli nie wszystkich) nie istnieje uniwersalny algorytm na rozwiązanie problemu dla dowolnego zestawu danych, dlatego ważna jest wiedza o danych jakie otrzymamy.

Zastosujemy inny algorytm w przypadku gdy chcemy znaleźć liczbę w tablicy, która nie jest posortowana, a inny gdy już jest, ale znowu możemy wybrać inny algorytm (albo zastosować jego inną wersję) w przypadku gdy będziemy potrzebowali znaleźć ten element tylko raz, a inaczej się zachowamy gdy wyszukiwanie będzie wykonywane często.

Widzimy, że ważne jest dokładne poznanie kontekstu w jakim chcemy wykorzystać algorytm, bo optymalizacja to nic innego jak dostosowanie algorytmu pod potrzeby rozwiązania konkretnego problemu.

Jeżeli coś w naszym kodzie jest zbędne to nie robimy tego, np. jeżeli nie musimy używać tablic do wczytania danych to nie używamy ich (ile ja już widziałem programów, gdzie właśnie w ten sposób były używane).

Podsumowanie

Mam nadzieję, że ten artykuł okaże się dla kogoś przydatny. Jeżeli znaleźliście błąd lub macie pytania odnośnie przedstawionych tu treści, a może chcecie podzielić się swoją opinią to nie bójcie się skorzystać z komentarzy.

Code ON!