**[31] Złożoność obliczeniowa algorytmu**

**Złożoność obliczeniowa algorytmu** – ilość zasobów komputerowych potrzebnych do jego wykonania:

* **Złożoność czasowa** – to ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych.
* **Złożoność pamięciowa** – to ilość pamięci potrzebnej do wykonania zadania, wyrażona jako funkcja ilości danych.

Rozróżniamy kilka rodzajów złożoności:

* **złożoność pesymistyczną (duże O) O(f(n))** - określa ilość zasobów (czasu lub pamięci) potrzebnych do wykonania algorytmu przy założeniu wystąpienia „złośliwych” lub najgorszych danych.
* **złożoność oczekiwaną (THETA) Θ(f(n))** - określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia „typowych” lub oczekiwanych danych
* **złożoność optymistyczną (OMEGA)** **Ω(f(n))** - ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia "najlepszych" danych.

Złożoność wykonywania różnych algorytmów możemy szacować:

• od dołu stwierdzając, iż złożoność obliczeniowa jest nie mniejsza niż pewna klasa funkcji Ω(f(n))

• asymptotycznie, szacując dokładnie złożoność obliczeniową poprzez pewną klasę funkcji Θ(f(n))

• od góry ograniczając złożoność obliczeniową poprzez pewną klasę funkcji O(f(n))

Zasadniczo rozróżniamy następujące złożoności obliczeniowe:

* **stała - Θ(1)** - gdy czas wykonania algorytmu jest stały i niezależny od rozmiaru danych wejściowych,
* **logarytmiczna - Θ(log n)** - kiedy czas ten rośnie logarytmiczne wraz ze wzrostem wielkości danych. Logarytm jest niemal zawsze o podstawie 2 (w przypadku notacji asymptotycznych nie ma to aczkolwiek znaczenia, bowiem podstawa logarytmu może być zmieniona poprzez pomnożenie przez czynnik stały,
* **liniowa - Θ(n)** - czas działania jest proporcjonalny do rozmiaru danych wejściowych,
* **liniowo-logarytmiczna - Θ(n log n)** - złożoność jest iloczynem funkcji liniowej i logarytmicznej,
* **kwadratowa - Θ(n2)** - liczba instrukcji algorytmu rośnie proporcjonalnie do kwadratu rozmiaru danych wejściowych,
* **sześcienna - Θ(n3)** - liczba instrukcji algorytmu rośnie proporcjonalnie do sześcianu rozmiaru danych wejściowych,
* **wielomianowa - Θ(nr + nr-1 + ... + n1)** - liczba instrukcji algorytmu rośnie proporcjonalnie do pewnego wielomianu rozmiaru danych wejściowych,
* **wykładnicza - Θ(2n)** - czas wykonania rośnie wykładniczo względem rozmiaru danych,
* **silni - Θ(n!)** - czas wykonania rośnie z szybkością silni względem rozmiaru danych.

Jeśli jesteśmy w stanie rozwiązać postawione zadanie algorytmem o złożoności nie większej niż wielomianowej, wtedy mamy szansę na rozwiązanie zadania w realnym/rozsądnym czasie. Jeśli zaś algorytm posiada złożoność ponadwielomianową, wtedy w praktyce wykonywanie algorytmu dla większej ilości danych, nigdy się nie zakończy, gdyż często rozwiązanie trwałoby dłużej niż szacowany wiek Wszechświata. Warto więc rozważać złożoność obliczeniową algorytmów, gdyż bez niej nawet dokładny i poprawny algorytm może się dla nas realnie nigdy nie zakończyć.

Problemy złożoności obliczeniowej bezpośrednio przekładają się na możliwość wykonania algorytmu na współczesnych maszynach obliczeniowych opartych na modelu maszyny Turinga, a ponadto istotnie wpływają na czas obliczeń.

W praktyce dzielimy algorytmy wg ich złożoności na 2 klasy:

* **łatwe (P** - polynomial**)** - czyli problemy, które potrafimy rozwiązań w czasie co najwyżej wielomianowym
* **trudne (NP**– non-polynomial **)** - problemy, dla których nie znamy rozwiązań w czasie wielomianowym lub mniejszym, czyli zadania o złożoności co najmniej wykładniczej
* podklasa problemów trudnych **NP-zupełne (NPC**– NP complete **)** - nie udowodniono, iż nie posiadają rozwiązania wielomianowego

**Złożoność czasowa** powinna być własnością samego algorytmu jako metody rozwiązania problemu, więc powinna być niezależna od komputera, języka programowania czy sposobu jego kodowania. W tym celu wyróżnia się tzw. **operacje dominujące** – takie, których łączna liczba jest proporcjonalna do liczby wykonań wszystkich operacji jednostkowych w dowolnej komputerowej realizacji algorytmu. Za **jednostkę czasową** przyjmuje się wykonanie jednej operacji dominującej.

**Złożoność pamięciowa**

To ilość dodatkowych jednostek pamięci (najczęściej bajtów lub bitów) potrzebnych do rozwiązania problemu danym algorytmem. Złożoność pamięciową wyrażamy również jako funkcję rozmiaru danych.

**Badanie efektywności algorytmów**

Polega na oszacowaniu ich złożoności czasowej i pamięciowej. Poszczególne części algorytmu badane są pod kątem ilości operacji dominujących (porównań, operacji arytmetycznych, przestawień itp.) W zależności od zastosowania oraz rodzaju i rozmiaru przetwarzanych struktur danych można wyróżnić różne operacje dominujące.

Klasyfikacja algorytmów

– **Algorytmy dokładne**

Dla danej instancji problemu obliczają jego dokładne rozwiązanie

– **Algorytmy aproksymacyjne**

Dla danej instancji problemu obliczają jego przybliżone rozwiązanie z zadaną dokładnością

– **Algorytmy Heurystyczne**

Dla danej instancji problemu obliczają jego przybliżone rozwiązanie z nieznaną dokładnością, ale w akceptowalnym (kontrolowanym) czasie

Heurystyką nazywamy algorytm (metodę) zwracający rozwiązanie przybliżone. Zazwyczaj oczekuje się, że algorytm taki ma złożoność wielomianową i działa szybko w praktyce

Metoda heurystyczna dla problemów optymalizacyjnych nie gwarantuje znalezienia optymalnego rozwiązania. Powinna dążyć do wygenerowania rozwiązania dopuszczalnego o wartości funkcji celu jak najbliższej optymalnej

W przypadku dowolnych problemów przeszukiwania metoda heurystyczna może nie zwrócić żadnego rozwiązania, jeśli nie znajdzie dopuszczalnego. Może także (w zależności od potrzeby) naruszyć ograniczenie na dopuszczalność rozwiązania. Powinna wtedy dążyć do wygenerowania rozwiązania jak najbardziej podobnego do rozwiązania dopuszczalnego (np. jak najdłuższa prosta ścieżka w grafie zamiast ścieżki Hamiltona)