Wrocław
University
of Science
and Technology

Struktury danych

Wykład 2

Struktury danych i złożoność obliczeniowa

dr inż. Jarosław Rudy

HR EXCELLENCE IN RESEARCH

Typ danych

Typ danych

Określony zbiór wartości wraz z określonymi operacjami, które można wykonywać na tych wartościach.

Proste przykłady:

- Przedział liczb całkowitych wraz z operacjami dodawania, odejmowania, mnożenia i dzielenia (całkowitoliczbowego).
- Przedział liczb rzeczywistych wraz z operacjami dodawania, odejmowania, mnożenia i dzielenia. Problem zaokrąglania.
- Przedział liczb naturalnych z operacjami dodawania, odejmowania, mnożenia i dzielenia (całkowitoliczbowego) w systemie z resztą.

Abstrakcyjny typ danych

Abstrakcyjny typ danych (ADT)

Opis typu danych uwzględniający własności wartości i wykonywanych na nich operacji bez uwzględnienia (określenia) szczegółów reprezentacji danych i implementacji operacji.

- ADT opisuje wymagane własności operacji, ale nie sposób ich zapewniania (definiuje interfejs typu danych).
- ADT opisuje logiczny aspekt danych.
- ADT powinien dokładnie opisywać wszystkie istotne przypadki.
- Przykładem ADT jest lista.

Struktura danych

Struktura danych (SD)

Opis (format) danych, określający sposób ich reprezentacji, przechowywania i zarządzania nimi.

- SD stanowi konkretną relizację pewnego ADT (definiuje implementację typu danych).
- SD opisuje fizyczny aspekt danych, zależny od konkretnej platformy (oprogramowania i sprzętu).
- Przykładami SD implementującymi listę (ADT) są tablica dynamiczna i lista wiązana.
- SD zasadniczo dotyczą danych przechowwanych w pamięci operacyjnej (RAM).

Struktura danych a ADT

- Czy strukura zbiorów rozłącznych jest strukturą danych czy ADT?
- Podział na struktury danych i ADT może być płynny.
- Lista lub słownik opisują tylko operacje i podstawowe własności, są więc ADT.
- Lista wiązana i tablica dynamiczna opisują konkretną implementację i strukturę w pamięci wykorzystując cechy języka programowania, są więc strukturami danych.
- Niektóre typy (np. kopiec, drzewo, graf, binarne drzewo poszukiwań) mogą być traktowane zarówno jako ADT jak i strukura danych, zależnie od kontekstu.
- Istotne jest co z przyjęcia danego ADT/SD wynika oraz które ADT/SD może być użyte do (wydajnej) implementacji których ADT/SD.

Typowe operacje na ADT/SD

- Stwierdzenie czy kolekcja jest pusta.
- Zwrócenie liczby elementów w kolekcji.
- Wyszukanie elementu:
  - po pozycji,
  - po kluczu,
  - po wartości.
- Dodanie elementu (też wg pozycji).
- Usunięcie elementu (wg pozycji, klucza, wartości).
- Przejrzenie (przejście) przez kolekcję.

Typy danych – przykłady (1)

Proste typy danych z języków programowania, niskopoziomowo zawsze są reprezentowane za pomocą bitów.

- Typy stałoprzecinkowe.
  - Typy liczbowe, z reguły całkowitoliczbowe.
  - Ze znakiem lub bez.
  - Reprezentacja w naturalnym kodzie binarnym lub kodzie uzupełnieniowym do 2.
  - Względnie wysoka precyzja i dokładność wyniku.
  - Względnie niski zakres wartości, problem nadmiaru.
  - Dzielenie „całkowitoliczbowe”.
  - Typy z C/C++: `int`, `long int`, `unsigned int`, `short int`, czasami też `char` i `boolean`.

Typy danych – przykłady (2)

- Typy zmiennoprzecinkowe.
  - Typy liczbowe, odzwierciedlenie liczb rzeczywistych.
  - Ze znakiem.
  - Reprezentacja z użyciem 3 liczb binarnych: znaku, ułamka z zakresu $[0, 1)$ i przesuniętego wykładnika.
  - Duży zakres wartości, dostateczna precyzja.
  - Dzielenie „rzeczywistoliczbowe".
  - Ograniczona dokładność (błąd reprezentacji, zaokrąglenia), nadmiar, niedomiar, NaN, wyjątki, brak łączności.
  - Typy z C/C++: `float`, `double`.

Typy danych – przykłady (3)

- Big numbers.
  - `python` domyślnie wspiera dowolnie duże liczby całkowite (np. zapis liczb w systemie o podstawie 2^{30} przy użyciu tablicy).
  - Biblioteka `BigNumbers` `pythona` zapewnia dowolnie duże liczby dowolnej precyzji.
- Typ znakowy (`char` w `C/C++`).
- Łańcuchy znaków (string).
  - Realizowane jako tablica.
  - Ograniczony rozmiar (przechowywany w zerowym elemencie tablicy), stosowane w `Pascalu`.
  - Null-terminated string (ASCIIZ, C string). Dowolna długość, łańcuch kończy pierwszy znak null. Powolne liczenie długości.

Typy danych – przykłady (4)

- Typ tablicowy (tablice statyczne).
  - Typ złożony, przechowuje wiele elementów (zwykle identycznego typu) zajmujących ciągły obszar pamięci.
  - Dostęp przez indeks (arytmetyka wskaźników).
  - Ustalony rozmiar.
- Typ strukturalny.
  - Typ złożony, przechowuje wiele elementów (zajmując ciągły obszar pamięci), elementy mogą być różnego typu.
  - Dostęp poprzez nazwę elementu.
  - Typy zawarte w strukturze często są niezmienne (w przeciwieństwie do wartości).
  - Definiowany przez użytkownika.

Typy danych – przykłady (5)

- Typ obiektowy.
  - Typ złożony rozszerzający typ strukturalny o definicje dozwolonych operacji.
  - Definiowany przez użytkownika.
  - Enkapsuluje własny kompletny typ danych (zbiór wartości powiązany z operacjami).
  - Jeden z paradygmatów programowania obiektowego w wielu językach (`C++`, `python`, `Java`, `Javascript`, `C#` itd.).
  - Stanowi podstawę do tworzenia ADT i SD niewspieranych natywnie przez język.

ADT – przykłady (1)

- Kolekcja (collection) i kontener (container) – ogólna grupa elementów ze sposobem przechowywania i dostępu.
- Lista (list) – zachowuje kolejność elementów (kolekcja liniowa). Elementy mogą się powtarzać. Dowolne wstawianie i usuwanie elementów.
- Zbiór (set) – brak kolejności, elementy nie mogą się powtarzać. Operacje z teorii zbiorów.
- Multizbiór (multiset) – odpowiednik zbioru, w którym elementy mogą się powtarzać.

ADT – przykłady (2)

- Kolejka (queue) – zachowuje kolejność, operacje możliwe tylko na końcach kolejki.
  - Kolejka First In-First Out (FIFO) – dodawanie na jednym końcu, usuwanie z drugiego końca.
  - Kolejka Last In-First Out (LIFO), stos – dodawanie i usuwanie elementów na tym samym końcu.
  - Kolejka dwustronna (double-ended queue, deque) – dodawanie i usuwanie na obu końcach.
  - Kolejka priorytetowa – kolejność definiowana przez priorytet elementu (elementy o wyższym priorytecie są usuwane pierwsze).

ADT – przykłady (3)

- Mapa (map), tablica asocjacyjna (associative table) lub słownik (dictionary) – przechowuje elementy w postaci pary klucz-wartość. Klucze nie mogą się powtarzać (klucz ma co najwyżej jedną wartość).
- Multimapa, multisłownik – odpowiednik mapy, w którym klucze mogą się powtarzać (klucz może mieć wiele wartości).
- Drzewo (tree) – każdy element ma jeden element rodzica (z wyjątkiem tzw. korzenia), każdy element może mieć potomków (dowolną liczbę).
- Graf (graph) – dowolne połączenia pomiędzy elementami, z lub bez określania kierunków.

Struktury danych – przykłady

- Tablica dynamiczna – tablica o zmiennej liczbie elementów zajmujących ciągły obszar w pamięci. Często stosowana do implementacji wielu ADT (lista, stos, kolejka, drzewo binarne). Dostęp indeksowy (swobodny).
- Lista wiązana – elementy nie muszą zajmować ciągłego obszaru pamięci. Narzut pamięci. Dostęp sekwencyjny. Używana do implementacji ADT takich jak lista, stos czy kolejka.
- Tablica mieszająca (hash table) – korzysta z funkcji mieszających. Używana do implementacji słownika.
- Binarne drzewa poszukiwań, drzewa czerwono-czarne, drzewa AVL – używane do implementacji słowników.
- Macierz sąsiedztwa, lista sąsiedztwa, lista krawędzi – struktury danych używane do reprezentacji grafów.

Złożoność obliczeniowa (1)

Złożoność obliczeniowa

Ilość zasobów potrzebnych algorytmowi do poprawnej pracy.

- Złożoność obliczeniową określa się dla konkretnego problemu oraz konkretnego algorytmu, programu lub operacji.
- Najczęściej ineresuje nas pojedyncza operacja ADT/struktury danych, ale możliwe jest rozważanie ciągu wielu operacji.
- Zwykle rozważa się albo konkretny rozmiar problemu (np. konkretny rozmiar listy, drzewa itd.), albo określa się złożoność jako funkcję, której argumentem jest rozmiar problemu.
  - Ile trwa wyszukanie elementu w liście N elementów?

Złożoność obliczeniowa (2)

Zasoby rozpatrywane w złożoności obliczeniowej:

- Złożoność czasowa.
- Złożoność pamięciowa.
- Złożoność komunikacji.
- Złożoność „równoległa” (liczba procesorów).

Złożoność czasowa

Złożoność (obliczeniowa) czasowa

Czas potrzebny do zakończenia algorytmu.

- Często szacowana liczbą elementarnych operacji.
- W praktyce występują nie tylko operacje elementarne.
- Różne operacje zajmują różny czas na różnych maszynach.
- W praktyce ilość czas rzeczywistego (wall time) potrzebnego do zakończenia algorytmu.
  - Także CPU time, problem pamięci podręcznej, narzut systemu operacyjnego itd.
- W niektórych sytuacjach – liczba operacji odczytu/zapisu pamięci operacyjnej.

Złożoność pamięciowa (1)

Złożoność (obliczeniowa) pamięciowa

Ilość pamięci (liczba komórek, bajtów itp.) potrzebnych do zakończenia algorytmu.

- Pamięć wejścia – pamięć potrzebna do zapisania danych wejściowych.
- Pamięć pomocnicza (dodatkowa) – pamięć potrzebna do zakończenia algorytmu z pominięciem pamięci wejścia.
- W praktyce często podaje się jedynie pamięć pomocniczą.

Złożoność pamięciowa (2)

Algorytm działający w miejscu (in-place)

Algorytm który nie potrzebuje pamięci pomocniczej proporcjonalnej do wielkości danych wejściowych.

Pojęcie nieścisłe, różne definicje praktyczne:

- Algorytm którego pamięć pomocnicza ma rozmiar co najwyżej stały (tj. $O(1)$).
- Algorytm którego pamięć pomocnicza ma rozmiar co najwyżej logarytmiczny (tj. $O(\log N)$).
- Czasami dopuszcza się dodatkową pamięć $O(N)$, o ile nie służy do przetwarzania wejścia.

Scenariusze złożoności (1)

Algorytmy mają różna złożoność dla różnych danych wejściowych. Dla uproszczenia analizy, często zakłada się konkretne scenariusze (przypadki).

- Przypadek pesymistyczny (worst-case) – dane dla których rozpatrywana złożoność jest największa.
  - Bardzo często wykorzystywany.
  - Gwarancja złożoności.
  - Ograniczona użyteczność (ekstremalne przypadki są zwykle rzadkie).
  - Jeśli nie można ustalić, to można zastąpić górnym ograniczeniem.

Scenariusze złożoności (2)

- Przypadek średni/typowy (average-case) – przypdek który jest wartością oczekiwaną dla całego rozkładu prawdopodobieństwa możliwych danych wejściowych.
  - Dość często wykorzystywany i użyteczny.
  - Często liczba zestawów danych wejściowych jest nieograniczona (nieskończona) – jak wyznaczyć?
- Przypadek optymistyczny (best-case) – dane dla których rozpatrywana złożoność jest najmniejsza.
  - Prawie nieużywny (niewielka użyteczność).
  - Jeśli nie można ustalić, to można zastąpić dolnym ograniczeniem.
- Inne – np. 95-ty percentyl.

Asymptotyczne tempo wzrostu

- Konieczność porównywania złożoności algorytmów dla różnego rozmiaru danych wejściowych.
- Często trudno jest ustalić dokładną złożoność (np. $15n^2 + 123n - 64$).
- Dla dużych n współczynniki (15, 123 i 64 powyżej) często nieistotne.
- Dla dużych n mniejsze miany ($123n$ lub 64) często nieistotne przy większych ($15n^2$).

Powyższe doprowadziło do pojęć asymptotycznego tempa wzrostu i rzędu funkcji złożoności obliczeniowej. Asymptotyczność oznacza, że rozpatruje się jak funkcja zachowuje się gdy $n \to \infty$.

Notacja dużego O (1)

Notacja dużego O

Dane są funkcje $f(x)$ oraz $g(x)$. Mówimy, że f jest rzędu co najwyżej g, co zapisujemy $f \in O(g)$, wtedy i tylko wtedy gdy:

$$\exists_{c \in \mathbb{R} \setminus \{0\}} \quad \exists_{x_0 \in \mathbb{R}} \quad \forall_{x \geqslant x_0} : \quad f(x) \leqslant c \cdot g(x). \tag{1}$$

- Czyli możemy znaleźć taki (wspólny) ogon obu funkcji (od x_0 „w prawo") i taką skalę funkcji g, że w tym ogonie zawsze $f(x)$ jest nie większa niż przeskalowane $g(x)$.
- Równoważnie można w definicji zapisać $c \cdot f(x) \leqslant g(x)$. Skala po prostu się odwraca (np. z $\frac{7}{2}$ na $\frac{2}{7}$).
- Często zamiast $f \in O(g)$ zapisuje się $f = O(g)$, ale jest to mylące i nie ma nic wspólnego z matematyczną równością! Z faktu że $f \in O(g)$ niekoniecznie wynika że $g \in O(f)$!

Notacja dużego O (2)

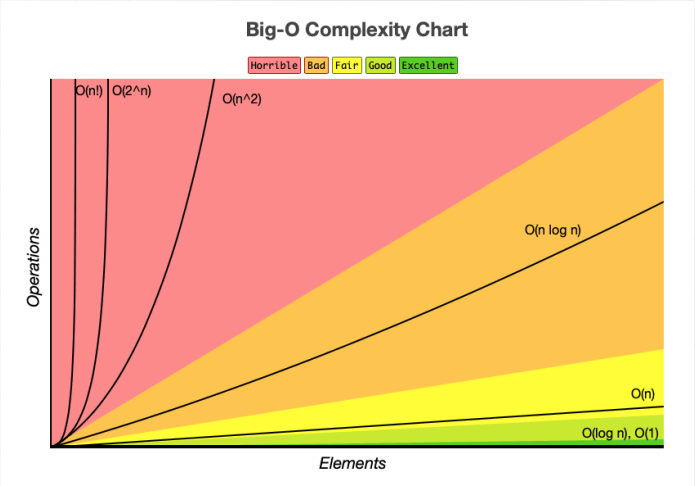
- $100n^2 \in O(n^2)$ – współczynnik bez znaczenia.
- $n^5 + n^3 + n - 5 \in O(n^5)$ – mniejsze miany nie mają znaczenia.
- $n^2 \in O(n^3)$ – kwadrat jest co najwyżej sześcianem.
- $n^2 \in O(2^n)$ – pomimo, że na odcinku $(2, 4)$ to n^2 jest większe.
- $n^3 \notin O(n^2)$ – jakikolwiek ogon i skalę dobierzemy, sześcian w końcu przegoni kwadrat.

Istotne rzędy złożoności (1)

Niektórze rzędy złożoności obliczeniowej (w kolejności rosnącej):

- $O(1)$ – złożoność stała (tj. ograniczona z góry przez stałą).
- $O(\log n)$ – złożoność logarytmiczna, dla dużego n traktowana jak $O(1)$.
- $O(n)$ – liniowa.
- $O(n^2)$ – kwadratowa. Dla niektórych problemów uznawana za szybką, dla niektórych (np. sortowanie) za powolną.
- $O(n^3)$ – sześcian.
- $O(2^n)$ – wykładnicza, zwykle nieakceptowalnie powolna.
- $O(n!)$ – silnia.

Istotne rzędy złożoności (2)



Source: [1]

Duże O – niuanse i praktyka (1)

- Fakt, że duże O oznacza „co najwyżej” (ograniczenie z góry), nie znaczy, że służy tylko i wyłącznie do opisu najgorszego przypadku!
  - Zadanie: w grupie n studentów znaleźć jednego nieobecnego.
  - Optymistycznie potrzeba 1 sprawdzenia, średnio $\frac{1+n}{2}$, pesymistycznie n.
  - Każdy z tych scenariuszy można ograniczyć od góry, odpowiednio $O(1)$, $O(n)$ oraz $O(n)$.
  - Każdy można też ograniczyć poprzez np. $O(n^2)$, ale zwykle interesuje nas możliwie dokładne ograniczenie.

Duże O – niuanse i praktyka (2)

- Czasami dla rzeczywistych rozmiarów danych algorytm o złożoności wyższego rzędu jest praktyczniejszy. Przykładowo:
  - Złożoność 2^n kontra $1\,000\,000n^2$.
  - Złożoność 2^n kontra n^{10}.
- Arytmetyka dużego O (przykłady):
  - $O(n) + O(n^2) = O(n + n^2) = O(n^2)$.
  - $10n^2 - O(n) + 15 = O(n^2) - O(n) + O(1) = O(n^2 - n + 1) = O(n^2)$
  - $10n^2 - O(n) + 15 = O(10n^2 - n + 15) = O(n^2)$.
  - $15n \cdot O(n^3) = O(15n^4) = O(n^4)$.
  - $O(\log n) \cdot O(n^2) = O(n^2 \log n)$.
  - $(3n + 1)^2 = 9n^2 + O(n)$.

Inne notacje

Wszystkie notacje zakładają, że rozważamy odpowiedni ogon funkcji i odpowiednie skale.

- $f \in O(g)$, duże O – f jest co najwyżej rzędu g.
- $f \in \Omega(g)$, duże omega – f jest co najmniej rzędu g.
- $f \in \Theta(g)$, teta – f jest rzędu (dokładnie) g. Implikuje $f \in O(g)$ oraz $f \in \Omega(g)$.
- $f \in o(g)$, małe o – f jest rzędu niższego niż g. Implikuje $f \in O(g)$.
- $f \in \omega(g)$, małe omega – f jest rzędu wyższego niż g. Implikuje $f \in \Omega(g)$.

Koszt zamortyzowany

- Różne definicje. Najczęściej wychodzi się od pesymistycznego sumarycznego kosztu ciągu k operacji (tych samych lub różnych).
- Z otrzymanej sumy wyciągamy średnią.
- Ponieważ zakładamy pesymizm każdej operacji, to wynik nie jest tym samym co analiza średniego przypadku (nie opiera się na probabilistyce).
- Koszt zamortyzowany pojedynczej operacji w ciągu może być różny (mniejszy) niż koszt tej samej operacji w tym samym ciągu obliczony z użyciem analizy najgorszego przypadku!

Koszt zamortyzowany – przykład

- Mamy zakład z n maszynami. Uruchamiane są razem jednym przyciskiem (jedna czynność), ale raz na n uruchomień trzeba je najpierw pojedynczo przeglądnąć ($n + 1$ czynności).
- Pesymistyczny czas pojedynczej operacji uruchomienia wynosi więc $O(n)$.
- Rozpatrzmy koszt pojedynczej operacji w ciągu n kolejnych uruchomień:
  - Klasyczna analiza najgorszego przypadku:

$$\frac{n \cdot O(n)}{n} = O\left(\frac{n^2}{n}\right) = O(n). \tag{2}$$

  - Koszt zamortyzowany:

$$\frac{\overbrace{1+1+\ldots+1}^{\text{n-1 razy}}+n+1}{n} = O\left(\frac{2n}{n}\right) = O(1). \tag{3}$$

Bibliografia

 https://blog.teclado.com/time-complexity-big-o-notation-python/