Paradygmaty Programowania

Sprawy organizacyjne
Struktury danych
Typy danych

dr inż. Marcin Bączyk (prowadzący)
dr inż. Adam Wojtasik, dr inż. Dominik Kasprowicz (współautorzy prezentacji)

Wykład 1

28 lutego 2022

Treść dzisiejszego wykładu

- Omówienie regulaminu przedmiotu
- Struktury danych
  - struktury tablicowe
  - struktury łączowe
  - struktury tablicowo-łączowe
- Typy danych
  - typy proste
  - typy złożone
  - abstrakcyjne typy danych
  - kolejki uogólnione

Wykład i egzamin

Wykład

- 10 tygodni – poniedziałek 8.15–10.00
- obecność nie jest obowiązkowa (ale znacznie ułatwia zaliczenie przedmiotu)

Kolokwia

- 2 x 15 minut w godzinach wykładu

Egzamin

- trzy terminy

Kolokwia i egzamin – korzystanie z materiałów pomocniczych

- dozwolone jest wykorzystywanie **własnych notatek i książek w formie "papierowej"**,
- dozwolone jest korzystanie z urządzeń elektronicznych (teści wykładów i inne w formie elektronicznej),
- niedopuszczalne jest komunikowanie się z kimkolwiek.

Laboratoria

- 5 ćwiczeń co ok. 2 tygodnie w środy i czwartki
  - instrukcje wstępne w materiałach wykładowych na serwerze Studia,
  - polecenia – bezpośrednio na laboratoriach,
  - przed każdym ćwiczeniem – wejściówka.
- Wynik ćwiczenia – sprawozdanie PDF
- Nieobecność na zajęciach należy usprawiedliwić w przeciągu 7 dni od ustania przyczyny nieobecności na zajęciach.
- Nieusprawiedliwiona nieobecność na laboratorium skutkuje **wyzerowaniem punktów z ćwiczenia**.
- Niesamodzielne wykonanie **dowolnej części** sprawozdania skutkuje **wyzerowaniem punktów z całego ćwiczenia**; dotyczy zwłaszcza treści przepisanych:
  - z literatury, "internetu" i innych ogólnie dostępnych źródeł
  - od kolegi lub koleżanki z bieżącego semestru (**punkty są zerowane obu osobom** bez analizowania kierunku przepływu informacji)
  - z cudzego sprawozdania z PAPRO z ubiegłych lat

Zasady oceniania (propozycja)

- **Punktacja**
  - laboratoria: 5 x 6 pkt.
    - wejściówka: 1,5 pkt.,
    - wykonanie: 4,5 pkt.,
    - zaliczenie laboratorium: od **2,5 pkt**., w tym min. **0,5 pkt. za wejściówkę**.
  - kolokwia: 2 x 10 pkt.
  - egzamin: 50 pkt.
- Do zaliczenia przedmiotu konieczne jest:
  - zaliczenie przynajmniej **3** laboratoriów,
  - zdobycie przynajmniej **51** pkt. z całego przedmiotu.
- **Oceny**
  - **5** – od 91 pkt.
  - **4,5** – od 81 pkt.
  - **4** – od 71 pkt.
  - **3,5** – od 61 pkt.
  - **3** – od 51 pkt.
- Zdobycie co najmniej **45 pkt**. w trakcie semestru zwalnia z egzaminu z oceną 5.
- Reszta zasad na stronie przedmiotu

Kontakt z prowadzącym(i)

Marcin Bączyk

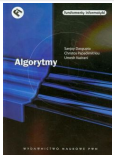
- pokój **449 GE** – prawie codziennie g. 10–16,
- **marcin.baczyk@pw.edu.pl** – sprawdzam raz dziennie
- Teams:
  - kanał ogólny przedmiotu – w sprawach, które mogą dotyczyć całej grupy
  - czat – w sprawach indywidualnych

Zagadnienia poruszane na wykładzie i laboratorium

- Problemy algorytmiczne, algorytmy, struktury danych, typy danych
- Metodyki programowania: funkcyjne, proceduralne, obiektowe
- Zasady SOLID
- Notacja UML
- Wzorce projektowe
- Programowanie współbieżne
- Programowanie zdarzeniowe
- Architektura oprogramowania

Literatura

S. Dasgupta, C. Papadimitriou, U. Vazirani *Algorytmy*, PWN, 2010



R. Sedgewick, K. Wayne *Algorytmy. Wydanie IV*, Helion, 2012

T. Cormen et al. *Wprowadzenie do algorytmów*, PWN, 2012

Literatura cd.

W. Malina, P. Mironowicz *Programowanie strukturalne*, PWN, 2018

S. Lott *Programowanie funkcyjne*, Helion, 2019

M. Weisfeld *Myślenie obiektowe w programowaniu*, Helion, 2010

Literatura cd.

E. Gamma, et al. (“Banda Czterech”) *Wzorce projektowe*, Helion, 2010

R. Martin (“Wujek Bob”) *Czysta architektura*, Helion, 2018

M. Richards *Software architecture patterns*, O’Reilly, 2015

Materiały do przedmiotu

https://studia2.elka.pw.edu.pl/pl/22Z/103A-ELxxx-ISP-PAPRO/lim/

Struktura danych

Mechanizm organizacji informacji (danych), który implikuje sposób dostępu do tych informacji i przeprowadzania na nich operacji

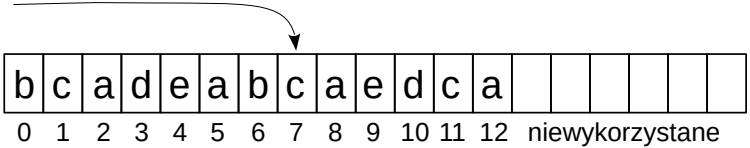
---

Jedyne dwie elementarne zasady budowania struktury danych

- Struktura tablicowa – elementy umieszczone są kolejno obok siebie i mają kolejne numery (indeksy).
- Struktura z łączami – elementy mogą być rozrzucone w pamięci, ale każdy z nich wyposażony jest w dodatkową informację mówiącą o tym, gdzie leży następny element.

Wszystkie stosowane struktury danych opierają się na stosowaniu jednej lub obu tych zasad

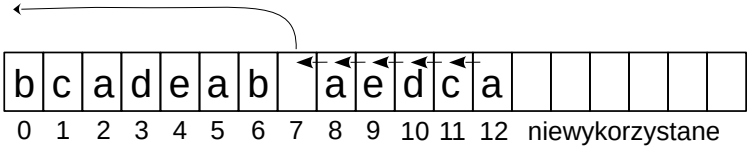
Tablicowa struktura danych



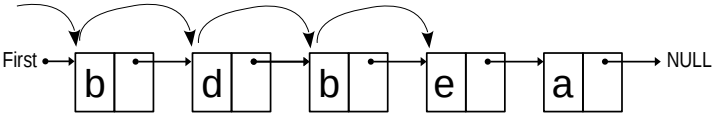
- Łatwy i szybki dostęp (równoległy),

Ale:

- Często część zarezerwowanej pamięci jest niewykorzystana.
- Budowa jest sztywna, trudna do modyfikacji czy reorganizacji.



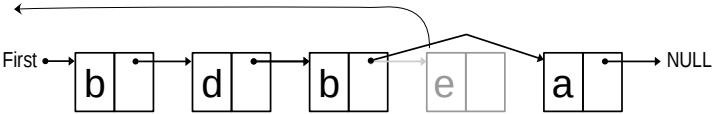
Łączowa struktura danych



- Niezbyt szybki dostęp (sekwencyjny, szeregowy).
- Każdy element musi zajmować dodatkową pamięć na łącza.

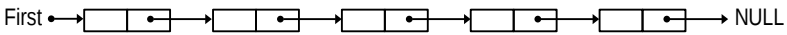
Ale:

- Nie ma „pustej” pamięci.
- Budowa jest elastyczna, łatwa do do reorganizacji czy modyfikacji.

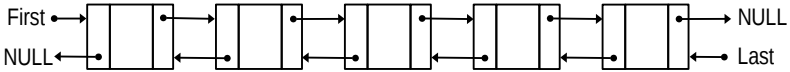


Łączowe struktury danych – przykłady

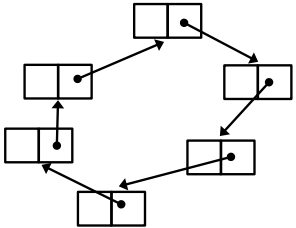
Lista:



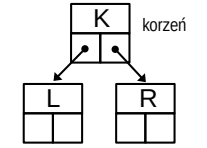
Lista dwukierunkowa:



Lista cykliczna:



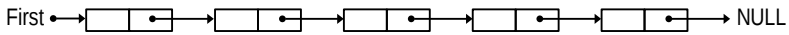
Drzewo binarne:



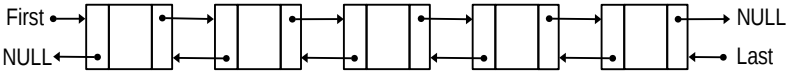
Potomkowie leżący na danym poziomie drzewa są korzeniami dla następnego poziomu

Łączowe struktury danych – przykłady

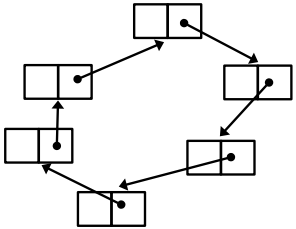
Lista:



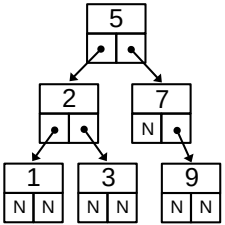
Lista dwukierunkowa:



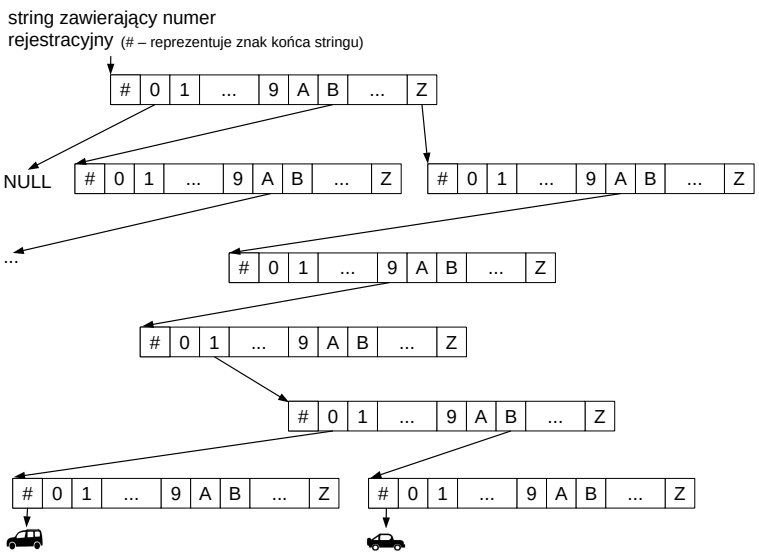
Lista cykliczna:



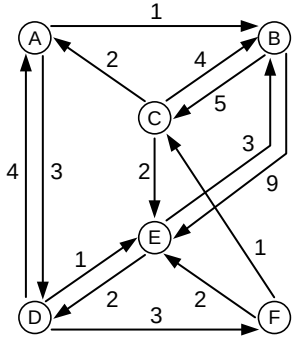
Drzewo poszukiwań binarnych (BST):



Tablicowo-łączowe struktury danych – przykład: B-drzewo

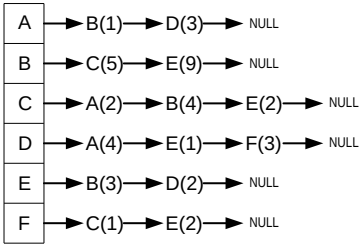


Alternatywy w wyborze struktury danych



	A	B	C	D	E	F
A	–	1	–	3	–	–
B	–	–	5	–	9	–
C	2	4	–	–	2	–
D	4	–	–	–	1	3
E	–	3	–	2	–	–
F	–	–	1	–	2	–

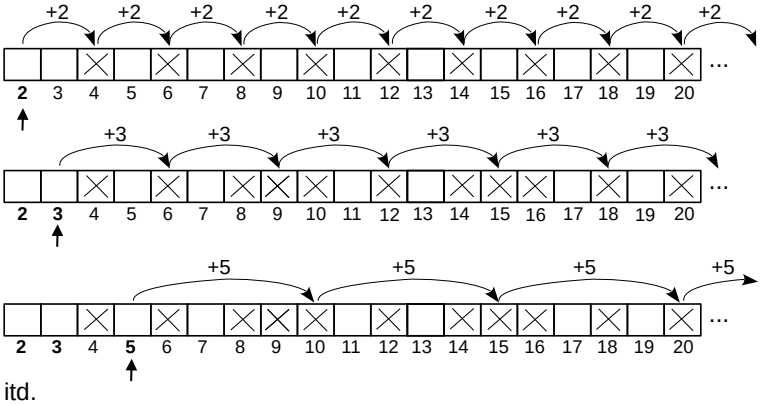
Różne struktury
przechowujące taki
sam graf skierowany



Algorytm a struktura danych

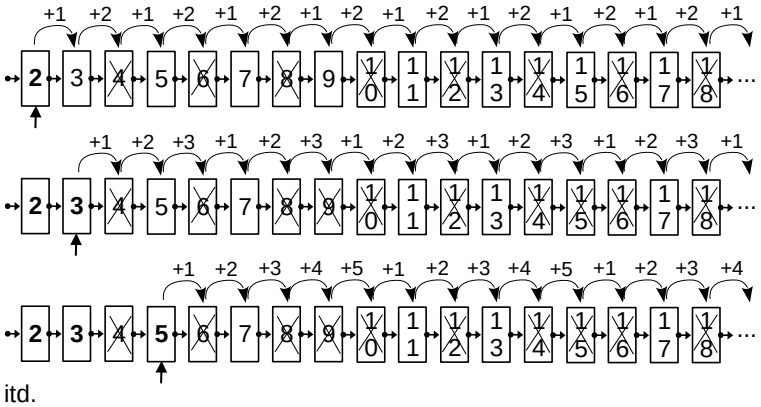
Algorytm i struktura danych są ze sobą mocno związane: wybór algorytmu implikuje wybór struktury danych i na odwrót

Algorytm a struktura danych – sito Eratostenesa



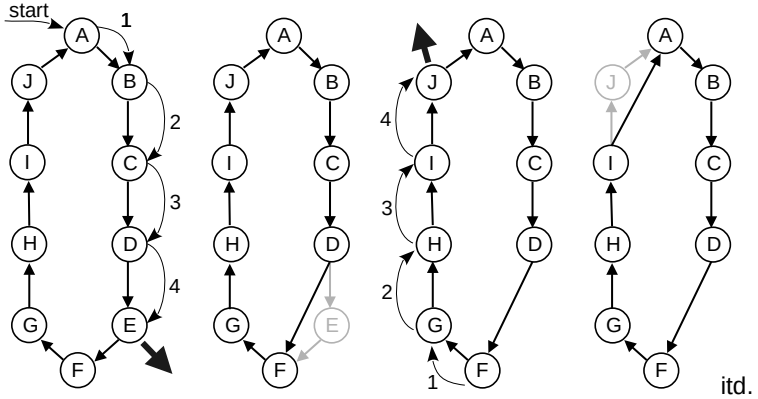
Użycie tablicy jest naturalne: implementacja jest łatwa (kolejne liczby to po prostu indeksy tablicy), działanie przebiega sprawnie.

Algorytm a struktura danych – sito Eratostenesa



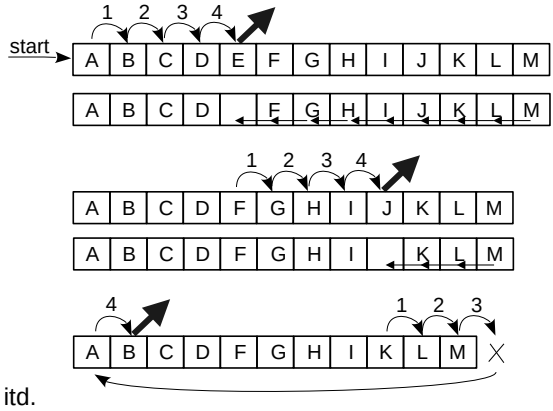
Użycie listy jest możliwe, ale implementacja mniej wygodna (trzeba stworzyć listę i wypełnić jej elementy kolejnymi liczbami naturalnymi), działanie jest mało efektywne szybkościowo.

Algorytm a struktura danych – eliminacja Józefa Flawiusza (Josephusa)



Użycie listy cyklicznej pozwala na łatwość implementacji i dużą sprawność działania.

Algorytm a struktura danych – eliminacja Józefa Flawiusza (Josephusa)



Użycie tablicy jest możliwe, ale niewygodne – wymuszające konieczność wykonywania dodatkowych czasochłonnych czynności (lub przy innej implementacji – alokacji dodatkowych informacji).

Typ danych

Zbiór wartości (informacji) oraz zbiór operacji, jakie można na tych wartościach wykonać

Typ danych

- **prosty**
  Integer 32-bitowy: –32 768 ÷ 32 767 oraz dodawanie, odejmowanie, mnożenie itd.
- **złożony (agregacyjny, zenkapsulowany)**
  np. tzw. krotka {dana1, dana2, dana3}: wszystkie kombinacje dopuszczalnych wartości danych 1, 2, 3 oraz jakieś operacje na tych danych.

Instancja typu danych (instancja danych)

Pojedynczy egzemplarz informacji należący do danego typu danych

Implementacja typu danych

Typ danych
jest pojęciem na wysokim poziomie abstrakcji (teoretycznym).
Jest to ogólnie pojęta informacja + ogólnie pojęte operacje.
Praktyczne użycie typu danych wymaga jego implementacji.

- Implementacja prostych typów danych wbudowana jest zazwyczaj w język programowania[1].
- Niektóre „standardowe" agregacyjne typy danych mogą być wbudowane w języki programowania[1] i/lub dostarczane przez biblioteki i zręby (frameworki).
- Programista może sam tworzyć potrzebne mu agregacyjne typy danych.

[1] W wielu językach używa się więc pojęcia „typ wbudowany".

Implementacja typu danych

- Do wewnętrznej organizacji informacji w implementacji typu danych stosowane są jakieś struktury danych (mniej lub bardziej złożone).
- Programista korzystający z wbudowanego lub bibliotecznego typu danych nie musi znać zastosowanej struktury (i na ogół jej nie zna – nie jest to potrzebne).

Abstrakcyjny typ danych

Interfejs

Operacje (procedury) stowarzyszone z typem danych, dzięki którym program ma dostęp do informacji zawartych w danej instancji i może na nich operować[a]

[a] Zatem definicja typu danych mogłaby brzmieć: Jest to zbiór wartości i jego interfejs.

Abstrakcyjny typ danych

Typ danych, dla którego informacje zawarte w instancjach są dostępne wyłącznie za pośrednictwem interfejsu (występuje tu tzw. hermetyzacja)[a]

[a] Programista nie bezpośredniego dostępu do wewnętrznej struktury danych.

Hermetyzacja

W pełni **zhermetyzowany** typ danych oznacza, że instancja tych danych jest **czarną skrzynką** o nieznanej strukturze wewnętrznej, więc „świat zewnętrzny" nie może bez „zgody" typu czegokolwiek tam zmienić

Konsekwencja hermetyzacji

Wewnętrzna budowa abstrakcyjnego typu danych jest niezależna od reszty programu, Można zupełnie przeorganizować tę budowę (czyli użytą strukturę danych) i zmienić zastosowane wewnętrzne algorytmy nie zmieniając interfejsu. Wtedy, mimo tak ogromnych zmian, cały „zewnętrzny" program stosujący ten typ danych nie musi być modyfikowany – będzie działał tak samo.

Podstawowy abstrakcyjny typ danych

Kolejka uogólniona

Składa się z danych oraz co najmniej dwóch operacji interfejsu: wstaw (włóż, push, insert, put) i usuń (wyjmij, pull, remove, get)

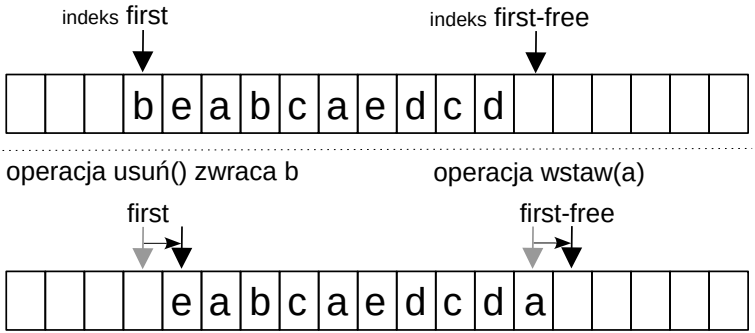
Najważniejsze rodzaje kolejek uogólnionych

- **stos (kolejka LIFO)**
- **kolejka (kolejka FIFO)**
- **kolejka priorytetowa**
- **słownik**
- **kolejka losowa**

Inne przykładowe typowe operacje obecne w interfejsach kolejek: wyszukaj (search), wybierz (select), uporządkuj (sort), połącz (join)

Kolejki – przykłady implementacji

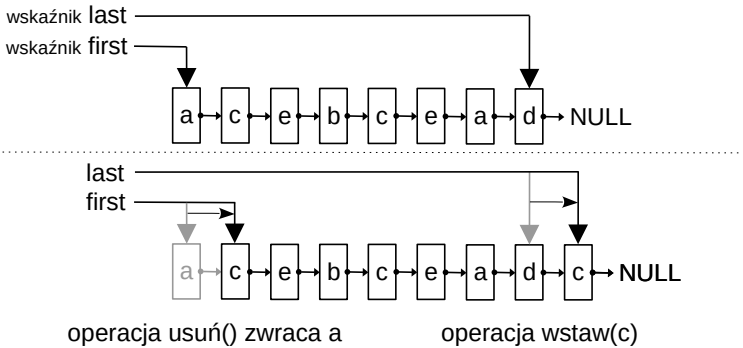
- **Kolejka FIFO – implementacja z użyciem tablicy:**



Kiedy indeks first-free wyjdzie poza rozmiar tablicy należy go „zresetować" na pierwszy element tablicy.
Kolejka działa poprawnie dla liczby elementów mniejszej o jeden od rozmiaru tablicy.
Można tego uniknąć implementując dodatkowo procedurę powiększenia (realokacji) tablicy w przypadku przekroczenia jej rozmiaru.

Kolejki – przykłady implementacji

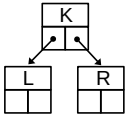
- **Kolejka FIFO – implementacja z użyciem listy:**



Liczba elementów w kolejce jest (teoretycznie) nieograniczona.

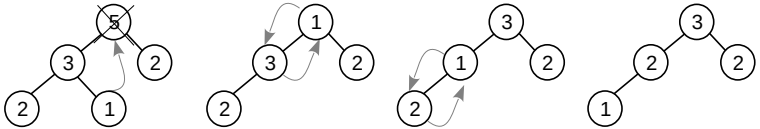
Kolejki – przykłady implementacji

- **Kolejka priorytetowa – implementacja z użyciem kopca:**



Drzewo binarne, w którym zawsze klucze węzłow spełniają warunki: K >= L oraz K >= P.
Zatem w korzeniu całego drzewa (na najwyższym poziomie) jest klucz o największej wartości.

Pu usunięciu korzenia trzeba przywrócić własność kopca:



Podobna procedura pozwala przywrócić własność kopca po wstawieniu nowego węzła (na pierwsze wolne miejsce).

Uwaga: Mimo, że jest to struktura łączowa kopiec wygodnie jest implementować w tablicy! Umieszcza się w niej kolejne węzły z kolejnych poziomów – wtedy dla korzenia o indeksie i lewy potomek ma indeks $2i$, a prawy $2i + 1$.

Kolekcja standardowa (kontener standardowy)

Narzędzie programistyczne (będące elementem języka programowania lub dostarczane jako biblioteczne), które jest kolejką uogólnioną do przechowywania instancji danych dowolnego rodzaju, mające cechy funkcjonalne związane z jakąś strukturą danych i mające jakąś charakterystyczną funkcjonalność

Uwaga

W ogólności kontener (kolekcja) to każdy typ zagregowany mający cechy kolejki uogólnionej (zarówno wbudowany w język lub biblioteczny, jak i zdefiniowany przez programistę na potrzeby jego programu)

Podstawowe rodzaje kolekcji – 1

Uwaga: W poszczególnych językach programowania niektóre typy kolekcji nie muszą być dostępne.
Uwaga: Dla różnych języków programowania kolekcje danego rodzaju mogą różnić się funkcjonalnością.

Tablica (Array)

Interfejs gwarantuje, że zawarte w niej elementy są uporządkowane jeden za drugim. Dostęp do nich uzyskuje się podając ich numer kolejny (wartość indeksu).

Zbiór (Set)

Interfejs nie gwarantuje żadnego uporządkowania elementów. Kolejność dostępu do poszczególnych instancji jest przypadkowa

Podstawowe rodzaje kolekcji – 2

Lista (List)

Interfejs gwarantuje, że zawarte w niej elementy są uporządkowane jeden za drugim. Kolejność dostępu do poszczególnych instancji jest sekwencyjna – przeszukiwanie zaczyna się zawsze od pierwszego elementu

Słownik (Dictionary)

Elementy mają przyporządkowane unikatowe słowa kluczowe (etykiety). Interfejs gwarantuje dostęp do poszukiwanej instancji po podaniu takiego słowa.

Uwaga: W różnych językach programowania takie same kolekcje mogą mieć różne nazwy. Także takie same funkcjonalnie procedury interfejsu mogą być inaczej nazywane.

Uwaga: W poszczególnych językach programowania dane rodzaje kolekcji mogą występować w kilku odmianach różniących się nieco funkcjonalnością i/lub właściwościami.