Akademia Górniczo-Hutnicza

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Adrian Horzyk

WSTĘP DO INFORMATYKI

Struktury liniowe

STRUKTURY LINIOWE – SEKWENCJE

Struktury liniowe (sekwencje) to takie struktury danych, których każdy element poza pierwszym i ostatnim posiada swojego poprzednika i następnika. Ostatni element nie posiada następnika, a pierwszy nie posiada poprzednika.

Struktury liniowe różnią się:

- możliwością dynamicznego zwiększania lub zmniejszania swojej struktury,
- sposobem i szybkością dostępu do elementów struktury,
- zajętością pamięci niezbędnej do przechowywania elementów struktury.

Każda struktura liniowa w Pythonie składa się z **elementów**, a każdy element w Pythonie jest obiektem, a każdy obiekt jest klasą, która może dziedziczyć z dowolnego typu.

Do podstawowych struktur liniowych (kolekcji) zaliczamy:

- tablice
- stosy
- kolejki
- listy
- krotki
- słowniki
- zbiory

[4]-[5]-[1]-[9]-[6]-[3]-[6]-[8]-[3]-[1]-[2]-[5]-[7]

Wszystkie struktury liniowe w Pythonie są wysoce zoptymalizowane pod kątem szybkości sortowania i przeszukiwania. Implementacja tych struktur oparta jest o tablice.

STRUKTURY LINIOWE w PYTHONIE

Struktury liniowe (sekwencyjne) w Pythonie:

- Listy (list) – to tablice o zmiennej liczbie elementów z możliwością ich dodawania, usuwania i podmiany. [typ zmienny – mutable]
- Krotki (tuple) – to tablice o stałej liczbie elementów bez możliwości ich dodawania, usuwania i podmiany. [typ niezmienny – immutable]
- Łańcuchy i napisy (string) – to tablice o stałej długości zawierające dowolne znaki, po których można iterować, lecz nie można ich zmieniać. [typ niezmienny – immutable]
- Słowniki (dictionary) – to struktury nieuporządkowane w postaci tablic asocjacyjnych, tzw. map, przyporządkowujące kluczom pewne wartości, gdzie klucze muszą być typami niezmiennymi (obiektami haszowalnymi posiadającymi metodę __hash__()), umożliwiającymi zdefiniowanie funkcji haszującej w celu ich szybkiego odnajdywania. [typ zmienny – mutable]
- Zbiory (set lub frozenset) – to tablice asocjacyjne, których elementy muszą być typami niezmiennymi (obiektami haszowalnymi posiadającymi metodę __hash__()), umożliwiającymi zdefiniowanie funkcji haszującej w celu ich szybkiego odnajdywania. [typ zmienny – mutable albo typ niezmienny – immutable w zależności od typu zbioru]
- Stosy (stack, FILO – first in last out) i kolejki (queue, FIFO – first in fistout) w Pythonie implementowane są z wykorzystaniem list, których są szczególnym przypadkiem.

W Pythonie wszystkie te kolekcje są implementowane z wykorzystaniem tablic (bez wskaźników), wykorzystując naturalną strukturę sekwencyjnych pamięci RAM, które są tablicami komórek, zwykle tzw. słów, składających się z bajtów.

TABLICE I TABELE

Tablica to jednorodna, zwarta struktura danych o niezmiennej długości składająca się z komórek zawierających dane tego samego typu.

Tablice mogą być:

- jednowymiarowe (wektory),
- dwuwymiarowe (macierze)
- lub wielowymiarowe.

4	5	1	9	6	3	6	8	3	1	2	5	7

4	5	1	9
6	3	6	8
3	1	2	5

Tabele to tablice rekordów, mogące zawierać dane różnych typów, lecz wszystkie rekordy tabel zawierają dane tych samych typów, które nazywamy atrybutami, cechami lub parametrami obiektów. Ponadto dane rekordów w tablicach są uporządkowane względem tych atrybutów tworząc kolumny danych tego samego typu.

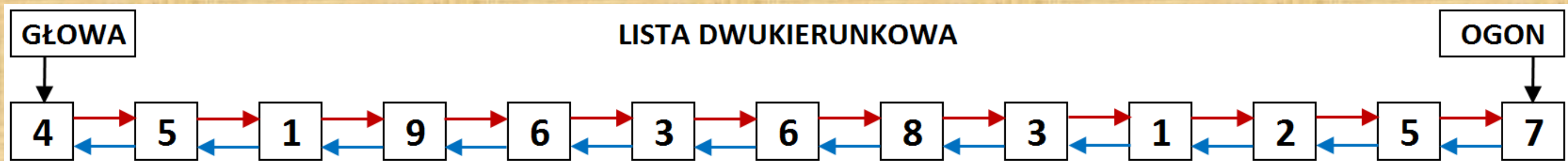
KLUCZ GŁÓWNY
KOLUMNA
KATEGORIA
ATRYBUT
POLE
TABELA
WIERSZ
REKORD
KROTKA
OBIEKT

Customers

CustomerID	FirstName	LastName	Address	City	Region	PostalCode	Country	PhoneNumb
ACKPI	Pilar	Ackerman	8808 Backbay S	Bellevue	WA	88004	USA	(425) 555-0194
ADATE	Terry	Adams	1932 52nd Ave.	Vancouver	BC	V4T 1Y9	Canada	(604) 555-0193
ALLMI	Michael	Allen	130 17th St.	Vancouver	BC	V4T 1Y9	Canada	(604) 555-0192
ASHCH	Chris	Ashton	89 Cedar Way	Redmond	WA	88052	USA	(425) 555-0191
BANMA	Martin	Bankov	78 Riverside Dr.	Woodinville	WA	88072	USA	(425) 555-0190
BENPA	Paula	Bento	6778 Cypress Pl	Oak Harbor	WA	88277	USA	(360) 555-0189
BERJO	Jo	Berry	407 Sunny Way	Kirkland	WA	88053	USA	(425) 555-0187
BERKA	Karen	Berg	PO Box 69	Yakima	WA	88902	USA	(509) 555-0188
BOSRA	Randall	Boseman	55 Grizzly Peak	Butte	MT	49707	USA	(406) 555-0186
BRETE	Ted	Bremer	311 87th Pl.	Beaverton	OR	87008	USA	(503) 555-0185
BROKE	Kevin F.	Browne	666 Fords Landi	Seattle	WA	88121	USA	(206) 555-0184
CAMDA	David	Campbell	22 Market St.	San Francisco	CA	84112	USA	(415) 555-0183

LISTY

Lista to dynamiczna, liniowa struktura danych składająca się z elementów powiązanych ze sobą w taki sposób, iż można z łatwością dodawać, usuwać, modyfikować lub podmieniać poszczególne elementy. Listy są wobec tego zwykle implementowane z wykorzystaniem wskaźników, które wiążą ze sobą rekordy (obiekty, klasy), zawierające w sobie dane oraz wskaźniki lub indeksy wskazujące poprzedni oraz następny rekord w liście, za wyjątkiem elementu pierwszego i ostatniego, których odpowiednie wskaźniki nie są ustawione.



Lista posiada dwa wyróżnione wskaźniki, zwane **głową (head)** i **ogonem (tail)**, które wskazują odpowiednio na początek i koniec listy, czyli pierwszy i ostatni element listy.

Wszystkie operacje na liście, tj. dodawanie, usuwanie, modyfikowanie i podmienianie elementów, mogą być wykonywane **w dowolnym miejscu listy**.

Powiązanie poszczególnych elementów w listach może być:

- jednokierunkowe
- lub dwukierunkowe,

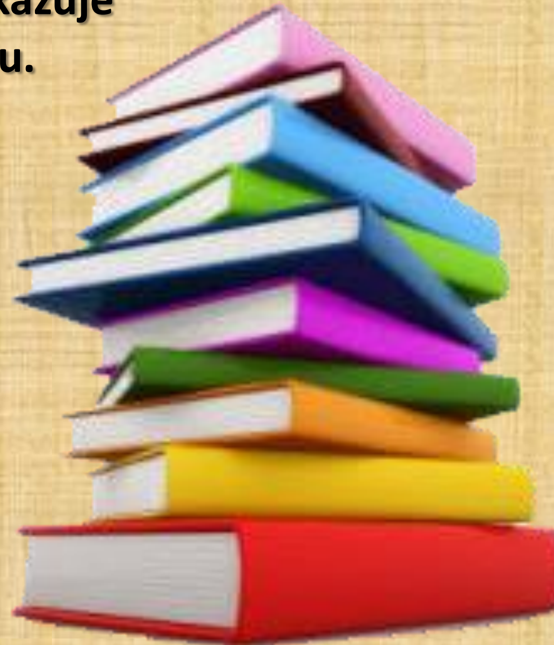
aczkolwiek zwykle przyjmuje się powiązania dwukierunkowe, które umożliwiają swobodę w poruszaniu się po elementach listy w obu kierunkach, co niestety odbywa się kosztem dodatkowej pamięci związanej z przechowywaniem dwóch wskaźników lub indeksów w zależności od sposobu implementacji listy w pamięci.

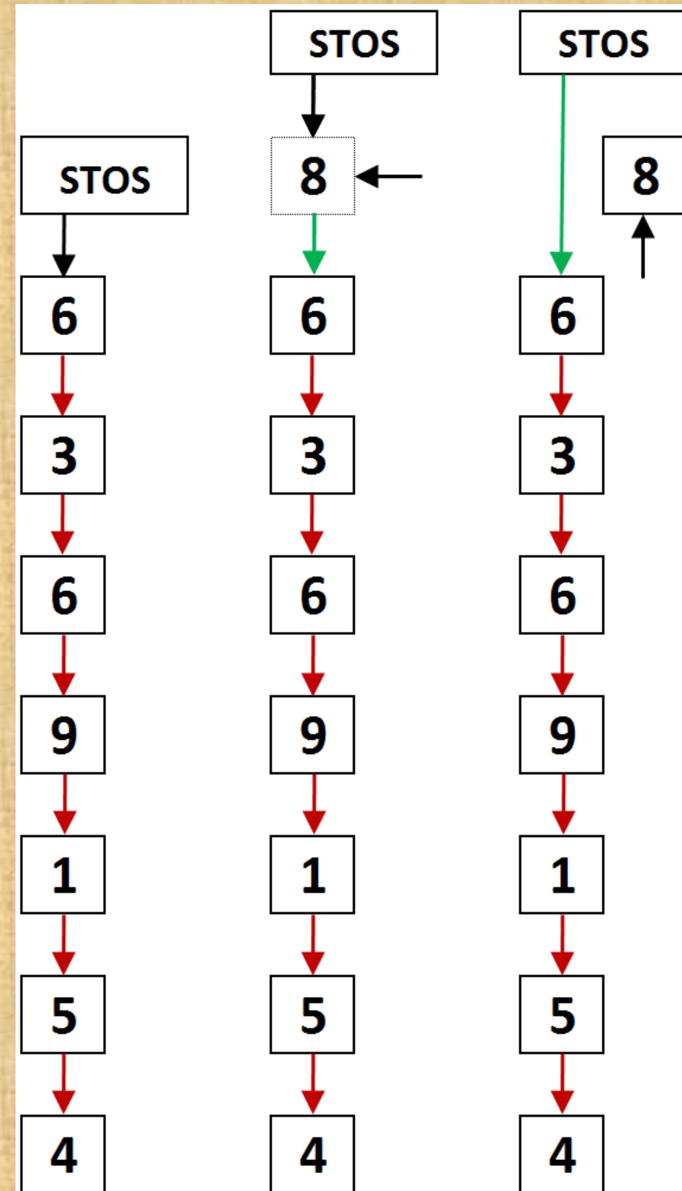
STOSY

Stos to dynamiczna, liniowa struktura danych składająca się z elementów powiązanych ze sobą w taki sposób, iż elementy można dodawać lub usuwać tylko z góry stosu.
Stos to szczególny przypadek listy, ograniczający operacje na niej.

Powiązanie poszczególnych elementów w stosie jest jednokierunkowe.

Stos posiada jeden wyróżniony wskaźnik, zwany górą stosu, która wskazuje na element na jego wierzchu. Tylko na tym elemencie można położyć kolejny albo go zdjąć ze stosu.



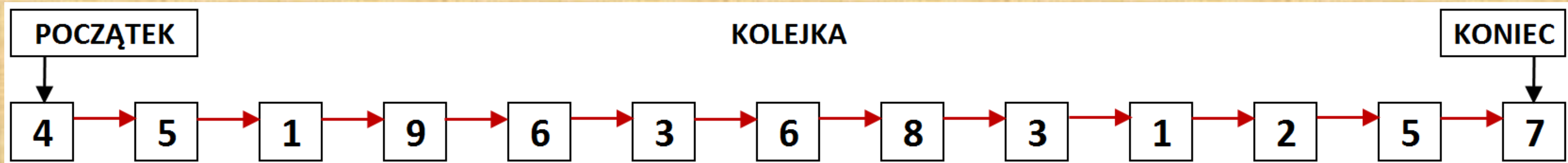


W Pythonie stos implementujemy z wykorzystaniem listy.

KOLEJKI

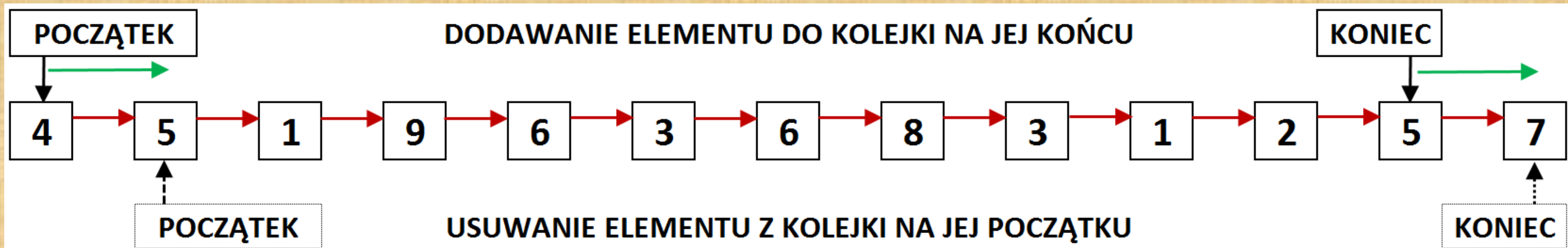
Kolejka to dynamiczna, liniowa struktura danych składająca się z elementów powiązanych ze sobą w taki sposób, iż elementy można z dodawać tylko na końcu kolejki, a usuwać je tylko z jej początku. Kolejka to szczególny przypadek listy, ograniczający operacje na niej.

Powiązanie poszczególnych elementów w kolejce jest jednokierunkowe.



Dodawanie nowego elementu na końcu kolejki wiąże się ze zmianą wskaźnika końca kolejki i dotychczas ostatniego elementu kolejki, ustawiając je na nowo dodanym elemencie.

Usuwanie elementu z przodu kolejki powoduje przesunięcie wskaźnika początku kolejki na następny element.

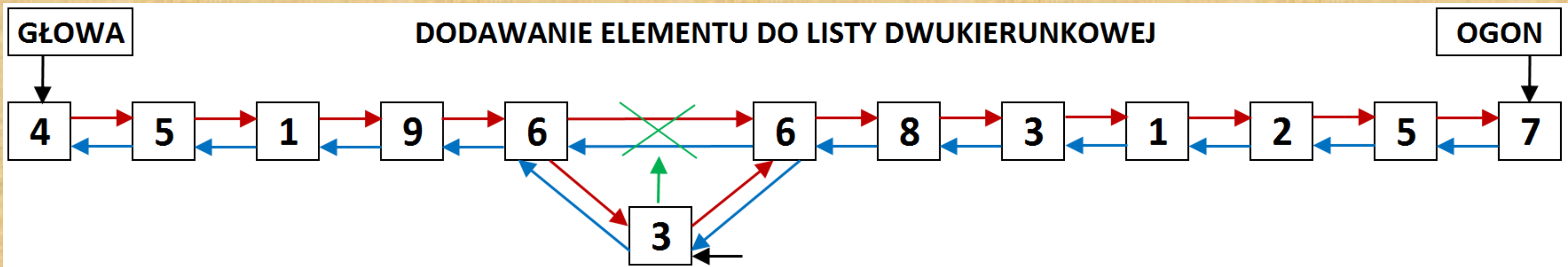


Kolejki mogą być priorytetowe, wtedy elementy dodawane są na końcu kolejki, a usuwane są według największego priorytetu, więc są często implementowane za pomocą stogu, czyli kopca zupełnego, który w czasie logarytmicznym pozwala wykonać te operacje.

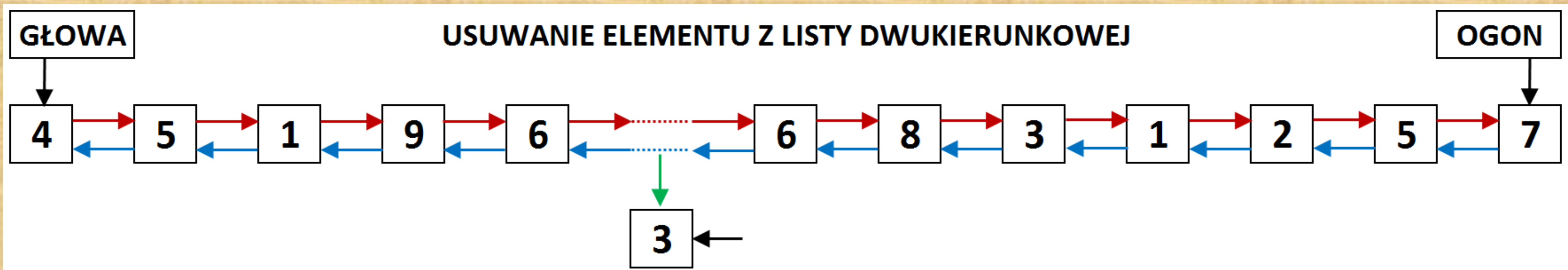
W Pythonie kolejkę implementujemy z wykorzystaniem listy.

OPERACJE NA LISTACH

Dodawanie elementu do listy w dowolnym jej miejscu:

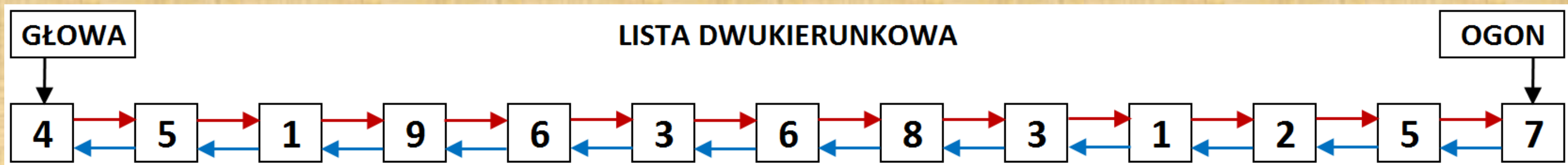


Usuwanie elementu z listy w dowolnym wskazanym miejscu:



POZYTYWNE I NEGATYWNE CECHY LIST I TABLIC

Klasyczna lista jest strukturą w pełni dynamiczną z punktu widzenia każdego jej elementu. A więc umożliwia łatwe dodawanie, usuwanie, podmianę i przemieszczanie elementów. Nie można jednak w szybki sposób jej przeglądać (np. w czasie logarytmicznym ani stałym), lecz niezbędne jest **przeszukiwanie elementów po kolei**, idąc po kolejnych wskaźnikach. Niestety również taka implementacja listy zajmuje **sporo więcej miejsca w pamięci**, gdyż każdy jej element wymaga dwóch wskaźników oraz struktury, która zawiera te wskaźniki razem z danymi właściwymi przechowywanymi wewnątrz każdego jej elementu.



Tablice pod względem **szybkości dostępu do dowolnego jej elementu** są bezkonkurencyjne, gdyż indeks elementu jednoznacznie wskazuje miejsce elementu w pamięci, które może zostać bardzo szybko i w czasie stałym wyznaczone.
Z punktu widzenia **efektywności wykorzystania pamięci** są również bezkonkurencyjne, gdyż nie wymagają ekstra pamięci na powiązanie poszczególnych elementów tablicy ze sobą.
Niestety w odniesieniu do operacji dodawania, usuwania i przemieszczania jej elementów z sekwencji tablice są bardzo kosztownym rozwiązaniem, szczególnie jeśli trzeba dokonywać tych operacji blisko jej początku.
Podobnie sprawa ma się z **możliwością zwiększanie lub zmniejszania** jej rozmiaru, wymagając przekopiowania wszystkich jej elementów do nowej tablicy.

4	5	1	9	6	3	6	8	3	1	2	5	7

WYKORZYSTANIE POZYTYWNYCH CECH

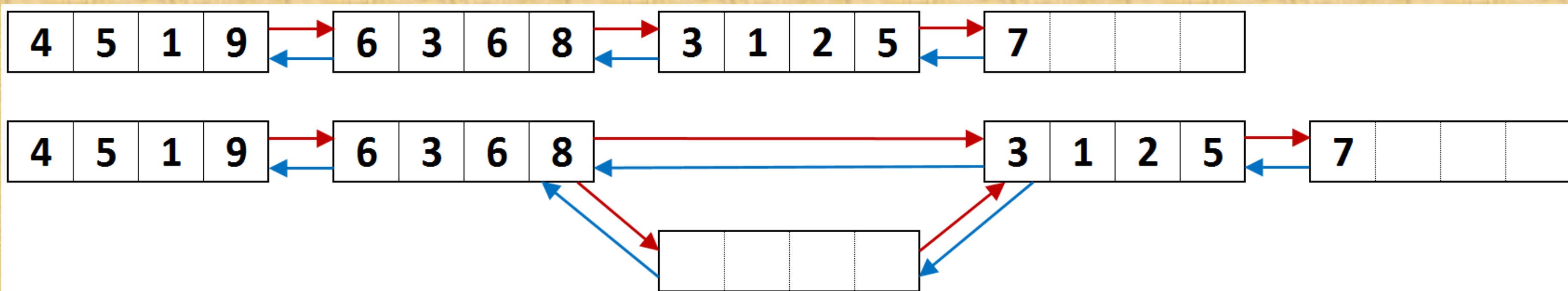
Powstaje więc pytanie, czy można wykorzystać pozytywne i wyeliminować negatywne cechy list i tablic, tworząc inny, nowy rodzaj struktury danych, bez ograniczeń i kosztu wykonywania nieefektywnych operacji na klasycznej tablicy i liście?

Odpowiedź na te pytania muszą zadawać sobie twórcy nowych języków programowania oraz kompilatorów – w tym również Pythona, którzy wyraźnie postawili na szybkość.

Żeby osiągnąć szybkość dostępu i oszczędność pamięci należałoby wykorzystać tablice, jeśli zaś łatwość dodawania, usuwania i przemieszczenia elementów właściwości list.

Pewnym kompromisem w stosunku dla braku możliwości rozszerzania tablic jest wykorzystanie tablicy nieco większej niż szacowana jej wielkość, a jeśli powiększenie to nie będzie przekraczało kosztu pamięciowego wskaźników listy, będzie rozwiązaniem dobrym.

To jednak nie rozwiązuje problemu nieefektywności wstawiania, usuwania i przemieszczania elementów w tablicy. Poszukano więc pewnych rozwiązań kompromisowych łączących pozytywne cechy obu struktur danych, tworząc tablico-listę (ArrayList), która jest w zasadzie listą tablic o znanej wielkości, przyspieszającą operacje przeszukiwania listy stosującą indeksy, które umożliwiają wyznaczenie, w której tablicy znajduje się poszukiwany element:



WYKORZYSTANIE POZYTYWNYCH CECH

Tablico-listy nie muszą jednak operować na tablicach tej samej wielkości, co w przeciwnym przypadku nie eliminowałoby problemu przesuwania elementów w tablicach przy próbie ich dodania, usunięcia lub przemieszczenia, lecz jeśli zastosujemy zmienną ilość elementów w tablicy, wtedy podmianie będą podlegały tylko pewne części całej struktury.
Takie rozwiązanie jednak komplikuje kwestię indeksowania i wyznaczania tablicy, w której znajduje się poszukiwany element.

4 5 1 9 | 6 3 6 8 | 3 1 2 5 | 7
1

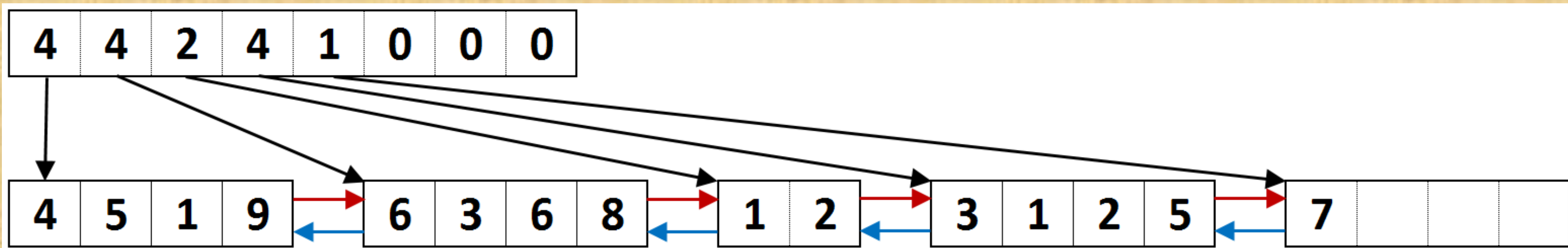
4 5 1 9 | 6 3 6 8 | 3 1 2 5 | 7
1 2

4 5 1 9 | 6 3 6 8 | 3 1 2 5 | 7
1 2 3

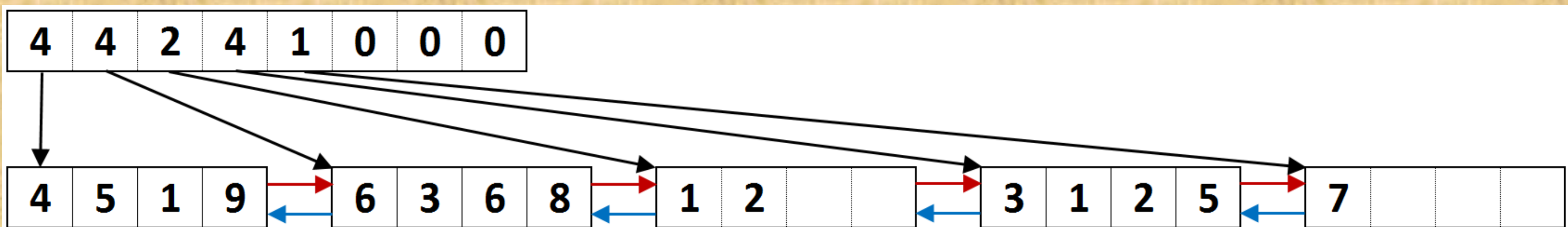
4 5 1 9 | 6 3 6 8 | 3 1 2 5 | 7
1 2 3 4

WYKORZYSTANIE POZYTYWNYCH CECH

Tablico-listy nie muszą jednak operować na tablicach tej samej wielkości, co w przeciwnym przypadku nie eliminowałoby problemu przesuwania elementów w tablicach przy próbie ich dodania, usunięcia lub przemieszczenia, lecz jeśli zastosujemy zmienną ilość elementów w tablicy, wtedy podmianie będą podlegały tylko pewne części całej struktury.
Takie rozwiązanie jednak komplikuje kwestię indeksowania i wyznaczania tablicy, w której znajduje się poszukiwany element.
Można jednak zastosować dodatkową tablicę zawierającą zarówno ilości przechowywanych elementów w poszczególnych tablicach, jak również wskaźniki do nich, co umożliwia w miarę szybkie wyznaczenie tablicy, w której znajduje się element o podanym indeksie, jak również wykorzystanie wskaźnika, żeby bezpośrednio przejść do tej tablicy:



albo



WYKORZYSTANIE POZYTYWNYCH CECH

Przeszukiwanie dodatkowej tablicy ilości indeksów ze wskaźnikami również można przyspieszyć stosując w miejsce takiej tablicy wyważone drzewo binarne zawierające ilości elementów w tablicach danych oraz sumy elementów w poszczególnych poddrzewach tych tablic, co przyspiesza do czasu logarytmicznego względem ilości sum elementów w tablicach indeksów wyznaczenie tablicy docelowej, indeksu w niej poszukiwanego elementu oraz odnalezienie wskaźnika prowadzącego do takiej tablicy, szczególnie jeśli operujemy na dużych kolekcjach danych.

Powstają więc drzewa list tablic, które przy doborze odpowiedniej wielkości tablic danych umożliwiają bardzo szybkie wykonywanie wszystkich operacji na nich, stanowiąc pewien kompromis pomiędzy tablicami, listami i drzewami:

15

14 1

8 6 1 0

4 4 2 4 1 0 0 0

4	5	1	9

6	3	6	8

1	2		

3	1	2	5

7			

EFEKTYWNE WYSZUKIWANIE W SŁOWNIKU

Słownik jest standardowo strukturą liniową, ale czasami można skorzystać ze szczególnych właściwości kluczy stosowanych w słownikach w celu optymalizacji dostępu do wartości.

Wyobraźmy sobie, iż kluczami słownika są słowa języka polskiego czy angielskiego.

Zależy nam na szybkim dostępie do opisu tego słowa lub jego tłumaczenia na inny język.

Możemy więc skorzystać z standardowej struktury słownika opartej o wyszukiwanie za pośrednictwem funkcji haszujące w Pythonie lub często stosowane wyszukiwanie połówkowe w innych językach programowania albo wykorzystać tą właściwość kluczy, zoptymalizować sposób ich przechowywania i dodatkowo zapewnić możliwość odnajdywania dowolnych wartości w czasie stałym:

DZRZEWO JĘZYKOWE - SŁOWNIK

A B C D E F G H I J K L M N O P R S T U V W X Y Z

B C D F G H J K L M N P R S T U V W X Z

BIBLIOGRAFIA I LITERATURA UZUPEŁNIAJĄCA

1. L. Banachowski, K. Diks, W. Rytter: „Algorytmy i struktury danych”, WNT, Warszawa, 2001.
2. M. Sysło: „Elementy Informatyki”.
3. A. Szepietowski: „Podstawy Informatyki”.
4. R. Tadeusiewicz, P. Moszner, A. Szydełko: „Teoretyczne podstawy informatyki”.
5. W. M. Turski: „Propedeutyka informatyki”.
6. N. Wirth: „Wstęp do programowania systematycznego”.
7. N. Wirth: „ALGORYTMY + STRUKTURY DANYCH = PROGRAMY”.