Data Structures Augmentation

Choosing a Data Structure

SoftUni Team

Technical Trainers

Software University

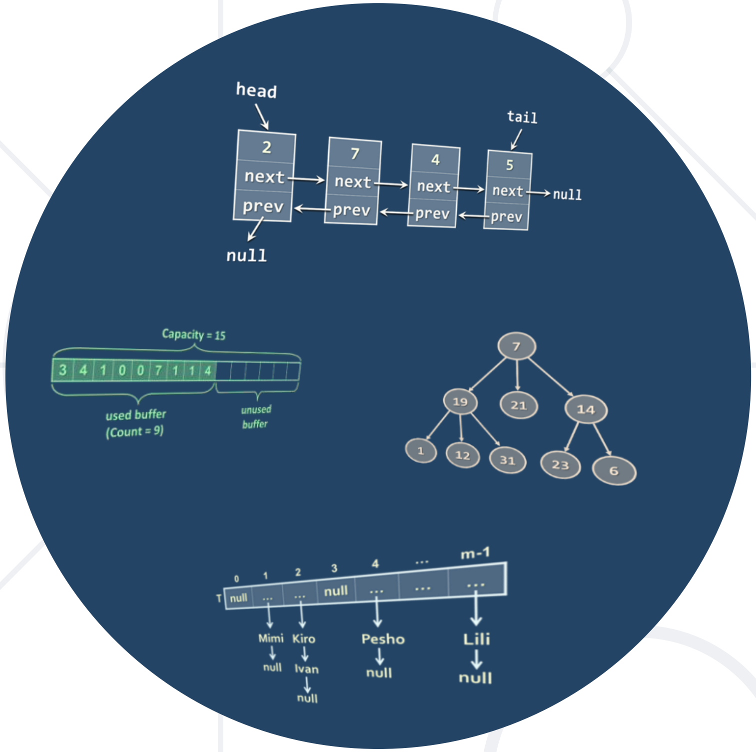
Software University

https://softuni.bg

SoftUni

Table of Contents

1. Classical Collection Data Structures – Summary
   - Linear Data Structures
   - Balanced Binary Search Trees
   - Hash Tables
2. Choosing a Collection Data Structure



Choosing the Right DS

Lists vs. Hash Tables vs. Balanced Trees

Choosing a Collection – Array

- Array (**T[]**)
  - Use when **fixed number of elements** need processing **by index**
    - No resize for fixed number of elements only
    - **Add** / **delete** needs creating a new array + move **O(n)** elements
    - Compact and lightweight

Data Structure	Add	Find	Delete	Get-by-index
`T[]`	`O(n)`	`O(n)`	`O(n)`	`O(1)`

Choosing a Collection – Array Based List

- Array-based list (**List<T>**)
  - Use when elements should be **added fast** and processes by **index**
  - Add (append to the end) has **O(1)** amortized complexity
  - The most-often used collection in programming

Data Structure	Add	Find	Delete	Get-by-index
`List<T>`	O(1)	O(n)	O(n)	O(1)

Choosing a Collection – Linked List

- Singly/Doubly linked list (**LinkedList<T>**)
  - Use when elements should be **added at the both sides** of the list
  - Use when you need to **remove by a node reference**
  - Otherwise use resizable array-based list (**List<T>**)

Data Structure	Add	Find	Delete	Get-by-index
`LinkedList<T>`	`O(1)`	`O(n)`	`O(n)`	`O(n)`

Choosing a Collection – Stack

- Stack (**Stack<T>**)
  - Use to implement **LIFO** (last-in-first-out) behavior
  - **List<T>** could also work well

Data Structure	Add	Find	Delete	Get-by-index
`Stack<T>`	O(1)	-	O(1)	-

Choosing a Collection – Queue

- Queue (**Queue<T>**)
  - Use to implement **FIFO** (first-in-first-out) behavior
  - **LinkedList<T>** could also work well

Data Structure	Add	Find	Delete	Get-by-index
Queue<T>	O(1)	-	O(1)	-

Choosing a Collection – Dictionary

- Hash-table based map (**Dictionary<K, V>**)
  - Fast **add key-value pairs** + fast **search by key** – O(1)
  - Keys have **no particular order**
  - Keys should implement **GetHashCode(…)** and **Equals(…)**

Data Structure	Add	Find	Delete	Get-by-index
`Dictionary<K, V>`	O(1)	O(1)	O(1)	-

Choosing a Collection – Sorted Dictionary

- Tree based map (**SortedDictionary<K, V>**)
  - Elements are **ordered** by key
  - Fast **add key-value pairs** + fast **search by key** + fast **sub-range**
  - Keys should be **IComparable<K>**
  - Balanced trees slower than hash-tables: **O(log n)** vs. **O(1)**

Data Structure	Add	Find	Delete	Get-by -index
`SortedDictionary<K, V>`	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Multi Dictionary

- Hash-table based multi-dictionary (**MultiDictionary<K, V>**)
  - Fast **add key-value** + fast **search by key** + **multiple values** by key
  - Add by existing key **appends a new value** for the same key
  - Keys have **no particular order**

Data Structure	Add	Find	Delete	Get-by-index
MultiDictionary<K, V>	O(1)	O(1)	O(1)	-

Choosing a Collection – Ordered Multi Dictionary

- Tree based multi-dictionary (**OrderedMultiDictionary<K, V>**)
  - Keys are **ordered** by key
  - Fast **add key-value** + fast **search by key** + fast **sub-range**
  - Add by existing key appends a new value for the same key

Data Structure	Add	Find	Delete	Get-by-index
OrderedMultiDictionary<K, V>	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Hash Set

- Hash-table based set (**HashSet<T>**)
  - **Unique** values + fast **add** + fast **contains**
  - Elements have **no particular order**
  - Elements should implement **GetHashCode(…)** and **Equals(…)**

Data Structure	Add	Find	Delete	Get-by-index
`HashSet<T>`	O(1)	O(1)	O(1)	-

Choosing a Collection – Sorted Set

- Tree based set (**SortedSet<T>**)
  - **Unique** values + **sorted order**
  - Fast **add** + fast **contains** + fast **sub-range**
  - Elements should be **IComparable<T>**

Data Structure	Add	Find	Delete	Get-by-index
`SortedSet<T>`	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Bag

- Hash-table based bag (**Bag<T>**)
  - Bags allow **duplicates**
  - Fast **add** + fast **find** + fast **contains**
  - Elements have **no particular order**

Data Structure	Add	Find	Delete	Get-by-index
Bag<T>	O(1)	O(1)	O(1)	-

Choosing a Collection – Ordered Bag

- Tree based bag (**OrderedBag<T>**)
  - Allow **duplicates**, **sorted order**
  - Fast **add** + fast **find** + fast **contains**
  - Access by **sorted index** + extract **sub-range**

Data Structure	Add	Find	Delete	Get-by-index
OrderedBag<T>	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Special DS

- Priority Queue (**Heap**) – **fast max**/**min** element
- **Rope** – fast **add**/**remove** by index
- **Prefix** tree (Trie) – fast **prefix search**
- **Suffix** tree – fast **suffix search**
- **Interval** tree – fast **interval search**
- **K-d** trees, **Quad** trees – fast **geometric distance search**

Data Structure Efficiency – Comparison

Data Structure	Add	Find	Delete	Get-by-index
`T[]`	O(n)	O(n)	O(n)	O(1)
`LinkedList<T>`	O(1)	O(n)	O(n)	O(n)
`List<T>`	O(1)	O(n)	O(n)	O(1)
`Stack<T>`	O(1)	-	O(1)	-
`Queue<T>`	O(1)	-	O(1)	-

Data Structure Efficiency – Comparison (2)

Data Structure	Add	Find	Delete	Get-by-index
Hash-table: `Dictionary<K, V>`	O(1)	O(1)	O(1)	-
Tree: `SortedDictionary<K, V>`	O(log n)	O(log n)	O(log n)	-
Hash-table: `HashSet<T>`	O(1)	O(1)	O(1)	-
Tree: `SortedSet<T>`	O(log n)	O(log n)	O(log n)	-

Data Structure Efficiency – Comparison (3)

Data Structure	Add	Find	Delete	Get-by-index
Hash-table: `MultiDictionary<K, V>`	O(1)	O(1)	O(1)	-
Tree: `OrderedMultiDictionary<K, V>`	O(log n)	O(log n)	O(log n)	-
Hash-table: `Bag<T>`	O(1)	O(1)	O(1)	-
Tree: `OrderedBag<T>`	O(log n)	O(log n)	O(log n)	-

Combining Data Structures

- Many scenarios combine several DS
- For example, we can combine:
  - A **hash-table** for fast **search by key_1** (e.g., name)
  - A **hash-table** for fast **search by {key_2 + key_3}** (e.g., name + town)
  - A **balanced search tree** for fast **range(startKey … endKey)**

Summary

- Different data structures have different efficiency for their operations
  - **List-based collections** provide **fast append** and access-by-index, but **slow find** and delete
  - The **fastest** add / find / delete structure is the **hash table** – O(1) for all operations
  - **Balanced trees** are **ordered** – O(log n) for add / find / delete + range(start, end)
- **Data structures Augmentation** is often essential
  - E.g., combine multiple hash-tables to find by different keys

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter International

INDEAVR Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

Questions?

SoftUni

- Software University
- SoftUni Creative
- SoftUni Digital
- SoftUni Foundation
- SoftUni Kids
- Finance Academy

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License