Lecture 6: Introduction to Data Structures

BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman
Department of Biotechnology
Bhupat and Jyoti Mehta School of Biosciences
Indian Institute of Technology Madras

Introduction

What is a data structure?

- Data structures organise (*structure*) data on a computer, for efficient use by algorithms, e.g. Array
- Abstract data types (ADTs) are theoretical models of data structures defining both the **type of data** and the **operations that can be performed** on the data, e.g. Set
- Data structures are *physical* implementations of ADTs on a computer

What is a data structure?

- Data structures organise (*structure*) data on a computer, for efficient use by algorithms, e.g. Array
- Abstract data types (ADTs) are theoretical models of data structures defining both the **type of data** and the **operations that can be performed** on the data, e.g. Set
- Data structures are *physical* implementations of ADTs on a computer

What is a data structure?

- Data structures organise (*structure*) data on a computer, for efficient use by algorithms, e.g. Array
- Abstract data types (ADTs) are theoretical models of data structures defining both the **type of data** and the **operations that can be performed** on the data, e.g. Set
- Data structures are *physical* implementations of ADTs on a computer

What is a data structure?

- We already know the basic data types: integers, characters, floats, Booleans etc.
- Abstract data types (ADTs) organise and structure collections of such data types
- Many data structures can implement the same ADT
- Data structures organise data efficiently — so that we can find, update, add and delete parts of it efficiently …

What is a data structure?

- We already know the basic data types: integers, characters, floats, Booleans etc.
- Abstract data types (ADTs) organise and structure collections of such data types
- Many data structures can implement the same ADT
- Data structures organise data efficiently — so that we can find, update, add and delete parts of it efficiently …

What is a data structure?

- We already know the basic data types: integers, characters, floats, Booleans etc.
- Abstract data types (ADTs) organise and structure collections of such data types
- Many data structures can implement the same ADT
- Data structures organise data efficiently — so that we can find, update, add and delete parts of it efficiently ...

What is a data structure?

- We already know the basic data types: integers, characters, floats, Booleans etc.
- Abstract data types (ADTs) organise and structure collections of such data types
- Many data structures can implement the same ADT
- Data structures organise data efficiently — so that we can find, update, add and delete parts of it efficiently ...

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Philosophy of Data Structures

- Every data structure has costs and benefits
- Rarely is one data structure better than another in all situations
- Data structures require:
  - space for each data item it stores (data + overheads)
  - time to perform each basic operation,
  - programming effort (Some data structures/algorithms can be very complicated!)
- Each problem has constraints on available space and time
- Careful analysis of problem characteristics $\longrightarrow$ best data structure for the task

Common Goals of Data Structures [and Algorithms]

- Correctness
- Efficiency
- Robustness
- Reusability
- Adaptability

Selecting a Data Structure

- Analyse problem to determine resource constraints a solution must meet
- What are the basic operations that must be supported?
  - What are the resource constraints?
- Which data structure best meets these requirements?
  - Typically we want the "simplest" data structure that will meet requirements

Selecting a Data Structure

- Analyse problem to determine resource constraints a solution must meet
- What are the basic operations that must be supported?
  - What are the resource constraints?
- Which data structure best meets these requirements?
  - Typically we want the "simplest" data structure that will meet requirements

Selecting a Data Structure

- Analyse problem to determine resource constraints a solution must meet
- What are the basic operations that must be supported?
    - What are the resource constraints?
- Which data structure best meets these requirements?
    - Typically we want the “simplest” data struture that will meet requirements

Selecting a Data Structure

- Analyse problem to determine resource constraints a solution must meet
- What are the basic operations that must be supported?
  - What are the resource constraints?
- Which data structure best meets these requirements?
  - Typically we want the "simplest" data struture that will meet requirements

Selecting a Data Structure

- Analyse problem to determine resource constraints a solution must meet
- What are the basic operations that must be supported?
    - What are the resource constraints?
- Which data structure best meets these requirements?
    - Typically we want the "simplest" data struture that will meet requirements

Selecting a Data Structure

Important Questions

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
  - i.e. are the data static or dynamic?
- Can data be deleted?
  - This may often demand a more complex representation
- Are all data processed in some well-defined order, or is random access allowed?

Selecting a Data Structure
Important Questions

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
  - i.e. are the data static or dynamic?
- Can data be deleted?
  - This may often demand a more complex representation
- Are all data processed in some well-defined order, or is random access allowed?

Selecting a Data Structure
Important Questions

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
  - i.e. are the data static or dynamic?
- Can data be deleted?
  - This may often demand a more complex representation
- Are all data processed in some well-defined order, or is random access allowed?

Selecting a Data Structure
Important Questions

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
  - i.e. are the data static or dynamic?
- Can data be deleted?
  - This may often demand a more complex representation
- Are all data processed in some well-defined order, or is random access allowed?

Selecting a Data Structure
Important Questions

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
  - i.e. are the data static or dynamic?
- Can data be deleted?
  - This may often demand a more complex representation
- Are all data processed in some well-defined order, or is random access allowed?

Skiena on Data Structures

“Changing a data structure in a slow program can work the same way an organ transplant does in a sick patient. Important classes of abstract data types such as containers, dictionaries, and priority queues, have many different but functionally equivalent data structures that implement them. Changing the data structure does not change the correctness of the program, since we presumably replace a correct implementation with a different correct implementation. However, the new implementation of the data type realizes different tradeoffs in the time to execute various operations, so the total performance can improve dramatically.”

Properties of a Data Structure

- Efficient utilization of memory and disk space
- Efficient algorithms for:
  - manipulation (e.g. insertion / deletion)
  - data retrieval (e.g. find)
  - creation
- A well-designed data structure uses less resources
  - computational: execution time
  - spatial: memory space

Properties of a Data Structure

- Efficient utilization of memory and disk space
- Efficient algorithms for:
  - manipulation (e.g. insertion / deletion)
  - data retrieval (e.g. find)
  - creation
- A well-designed data structure uses less resources
  - computational: execution time
  - spatial: memory space

Properties of a Data Structure

- Efficient utilization of memory and disk space
- Efficient algorithms for:
  - manipulation (e.g. insertion / deletion)
  - data retrieval (e.g. find)
  - creation
- A well-designed data structure uses less resources
  - computational: execution time
  - spatial: memory space

Abstract Data Type: Linear List

- A list of items of a finite length *n*

Operations

- `create()`
- `delete()`
- `isEmpty()`
- `length()`
- `find()` *k*-th element
- `search()` for a given element
- `insert()` an element into the list
- `append()`, `join()`, `copy()`, ...

Abstract Data Type: Linear List

- A list of items of a finite length *n*

Operations

- `create()`
- `delete()`
- `isEmpty()`
- `length()`
- `find()` *k*-th element
- `search()` for a given element
- `insert()` an element into the list
- `append()`, `join()`, `copy()`, …

Think about the cost of each operation …

A Linear List can be implemented using a contiguous or linked data structure …

Contiguous vs. Linked Data Structures
Skiena

Data structures can be neatly classified as either contiguous or linked, depending upon whether they are based on arrays or pointers:

- Contiguously-allocated structures are composed of single slabs of memory, and include arrays, matrices, heaps, and hash tables
- Linked data structures are composed of distinct chunks of memory bound together by pointers, and include lists, trees, and graph adjacency lists

Contiguous vs. Linked Data Structures

Linked

- Extra storage required
- Better use of fragmented memory
- Insertion/deletion at middle is easier
- Joining lists easier
- 'Next' operation requires pointer dereference

Contiguous

- Next and Previous are implicit (less storage)
- Can take advantage of locality
- Random access
- 'Next' operation probably faster

Contiguous vs. Linked Data Structures

Linked

- Extra storage required
- Better use of fragmented memory
- Insertion/deletion at middle is easier
- Joining lists easier
- 'Next' operation requires pointer dereference

Contiguous

- Next and Previous are implicit (less storage)
- Can take advantage of locality
- Random access
- 'Next' operation probably faster

Contiguous vs. Linked Data Structures

Linked

- Extra storage required
- Better use of fragmented memory
- Insertion/deletion at middle is easier
- Joining lists easier
- 'Next' operation requires pointer dereference

Contiguous

- Next and Previous are implicit (less storage)
- Can take advantage of locality
- Random access
- 'Next' operation probably faster

Contiguous vs. Linked Data Structures

Linked

- Extra storage required
- Better use of fragmented memory
- Insertion/deletion at middle is easier
- Joining lists easier
- 'Next' operation requires pointer dereference

Contiguous

- Next and Previous are implicit (less storage)
- Can take advantage of locality
- Random access
- 'Next' operation probably faster

Contiguous vs. Linked Data Structures

Linked

- Extra storage required
- Better use of fragmented memory
- Insertion/deletion at middle is easier
- Joining lists easier
- ‘Next’ operation requires pointer dereference

Contiguous

- Next and Previous are implicit (less storage)
- Can take advantage of locality
- Random access
- ‘Next’ operation probably faster