Data Structures and Algorithms

Géza Horváth

Third Lecture

Lecture topics

1. Introductory examples of the design and analysis of algorithms.
2. Asymptotic behavior of functions, asymptotic notation, further examples of simple algorithms.
3. Basic notions concerning data structures: modeling, abstraction. Set, multiset, array, matrix.
4. Elementary data structures: stacks, queues, lists.
5. Tables, hash tables, hash functions, collisions.
6. Trees, binary trees, binary search trees, traversal, searching, insertion, deletion.
7. Balanced binary search trees, AVL trees.
8. Red-black trees.
9. B-trees.
10. Graphs, traversal, shortest path.
11. Parallel algorithms: matrix multiplication, merge sort.
12. Decision problems, the complexity classes P and NP.
13. Summary.

Data structures

Data Structure: elements + relationships + operations + representation.

More precisely, a data structure is a collection of data values, the relationships among them, the operations that can be applied to the data and the representation which describes how a computer – together with its peripheral devices – handles data in its memory, on magnetic media (and in optical devices).

Data structures serve as the basis for abstract data types.

- The **abstract data type** defines the **logical** form of the data type.
- The **data structure** implements the **physical** form of the data type.

Operations over data structures

Operations:

- **create** data structure
- **modify** data structure
  - add elements
  - change elements
  - delete elements
- **access** elements

Representation of data structures

Modern computers use RAM (random-access memory). Random-access (more precisely and more generally called **direct access**) is the ability to access an arbitrary element of a sequence in equal time or any item of data from a population of addressable elements roughly as easily and efficiently as any other, no matter how many elements may be in the set.

Representation of data structure can be:

- continuous (or vector) – today
- linked – next week

Classification of data structures

See the separate paper about this topic.

Example – binary search

For the **binary search** operation, the following properties must be satisfied:

- solid data structure (same kind of elements with same size)
- sorted data
- continuous representation (with direct access)

The set – abstract data type

A set is a collection of distinguishable objects, called its members or elements. If an object x is a member of a set S, we write $x \in S$ (read "x is a member of S" or, more briefly, "x is in S"). If x is not a member of S, we write $x \notin S$. We can describe a set by explicitly listing its members as a list inside braces. For example, we can define a set S to contain precisely the numbers 1, 2, and 3 by writing $S = \{1,2,3\}$.

The set – abstract data type

Let A and B be two sets. One can perform the following set operations.

Set operations:

- The ***intersection*** of sets A and B is the set

  $$A \cap B = \{x : x \in A \text{ and } x \in B\} .$$

- The ***union*** of sets A and B is the set

  $$A \cup B = \{x : x \in A \text{ or } x \in B\} .$$

- The ***difference*** between two sets A and B is the set

  $$A - B = \{x : x \in A \text{ and } x \notin B\} .$$

The set – abstract data type

Set operations obey the following laws:

Empty set laws:

$$A \cap \emptyset = \emptyset ,$$
$$A \cup \emptyset = A .$$

Idempotency laws:

$$A \cap A = A ,$$
$$A \cup A = A .$$

Commutative laws:

$$A \cap B = B \cap A ,$$
$$A \cup B = B \cup A .$$

The set – abstract data type

Associative laws:

$$A \cap (B \cap C) = (A \cap B) \cap C ,$$
$$A \cup (B \cup C) = (A \cup B) \cup C .$$

Distributive laws:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) ,$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) .$$

Absorption laws:

$$A \cap (A \cup B) = A ,$$
$$A \cup (A \cap B) = A .$$

DeMorgan's laws:

$$A - (B \cap C) = (A - B) \cup (A - C) ,$$
$$A - (B \cup C) = (A - B) \cap (A - C) .$$

The set – data structure

Set properties:

- solid
- without structure
- dynamic
- continuously represented

The set – representation

Let A be a set containing numbers between 0 and 9.

For example A=$\{0,3,4,5,6,8\}$.

A good representation of the set A have only 10 bits:

1	0	0	1	1	1	1	0	1	0
0	1	2	3	4	5	6	7	8	9

The set – set operations

Let A and B be sets containing numbers between 0 and 9.

For example A={0,3,4,5,6,8}, B={1,3,5,9}.

The representation of the set A:

1	0	0	1	1	1	1	0	1	0
0	1	2	3	4	5	6	7	8	9

The representation of the set B:

0	1	0	1	0	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9

The set – union

A={0,3,4,5,6,8}, B={1,3,5,9}.

The representation of the set A:

1	0	0	1	1	1	1	0	1	0
0	1	2	3	4	5	6	7	8	9

The representation of the set B:

0	1	0	1	0	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9

To calculate the **union** of the set A and B we can use the logical (bitwise) **OR** operation:

1	1	0	1	1	1	1	0	1	1
0	1	2	3	4	5	6	7	8	9

The set – intersection

A={0,3,4,5,6,8}, B={1,3,5,9}.

The representation of the set A:

1	0	0	1	1	1	1	0	1	0
0	1	2	3	4	5	6	7	8	9

The representation of the set B:

0	1	0	1	0	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9

To calculate the **intersection** of the set A and B we can use the logical (bitwise) **AND** operation:

0	0	0	1	0	1	0	0	0	0
0	1	2	3	4	5	6	7	8	9

The set – difference

A={0,3,4,5,6,8}, B={1,3,5,9}, $\bar{B}$={0,2,4,6,7,8}.

The representation of the set A:

1	0	0	1	1	1	1	0	1	0
0	1	2	3	4	5	6	7	8	9

The representation of the set B:

0	1	0	1	0	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9

To calculate the **difference** of the set A and B we can calculate the logical expression **A AND NOT B**. First calculate **NOT B**:

1	0	1	0	1	0	1	1	1	0
0	1	2	3	4	5	6	7	8	9

Then calculate **A AND NOT B**:

1	0	0	0	1	0	1	0	1	0
0	1	2	3	4	5	6	7	8	9

The set – data structure operations

Implemented data structure operations:

- **create** data structure – continuous (vector) representation
- **modify** data structure
  - add elements – union (create singleton then union)
  - delete elements – difference (create singleton then difference)
  - change elements – difference and union
- **access** elements – $\in$

The multiset – abstract data type

In mathematics, a multiset is a modification of the concept of a set that, unlike a set, allows for multiple instances for each of its elements.

The positive integer number of instances, given for each element is called the multiplicity of this element in the multiset.

For example, we can define a multiset S to contain precisely the numbers 1 twice, 2 once, and 3 twice, by writing S=$\{1,1,2,3,3\}$.

Multiset operations and representation will be explained in practical class.

Array – abstract data type

In computer science, an array data type is a data type consisting of fix number of elements, each identified by a sequence of integers.

The one dimensional array is called **vector**, and its elements are identified by one integer called "index", which is the position of the element in the vector.

Example: Let A={2,11,21,8,6,5,9,3,15,4} be a vector.

2	11	21	8	6	5	9	3	15	4
1	2	3	4	5	6	7	8	9	10

Here the element 11 is identified by 2, – its position in the vector, because A[2]=11, – and the index of 3 is 8, because A[8]=3.

Array – abstract data type

The two dimensional array is called **matrix**, and its elements are identified by two integers, these numbers show the position of the element in the matrix.

Example: Let M be the following matrix.

	1	2	3	4	5	6
1	6	11	8	5	22	3
2	12	9	2	21	6	9
3	1	5	24	2	9	17

Here the element 21 is identified by (2,4), – its position in M, because M$[2, 4]$=21, – and the indexes of 17 is (3,6), because A$[3, 6]$=17.

The array – abstract data type

There are higher dimensional arrays too, for example, the elements of a 4 dimensional array is identified by a sequence of 4 integers.

How can we "imagine" higher dimensional arrays?

The array – data structure

Array properties:

- solid
- associative
- **static**
- continuously represented

Vector representation

This is the most "natural" continuous representation.

int A[5];

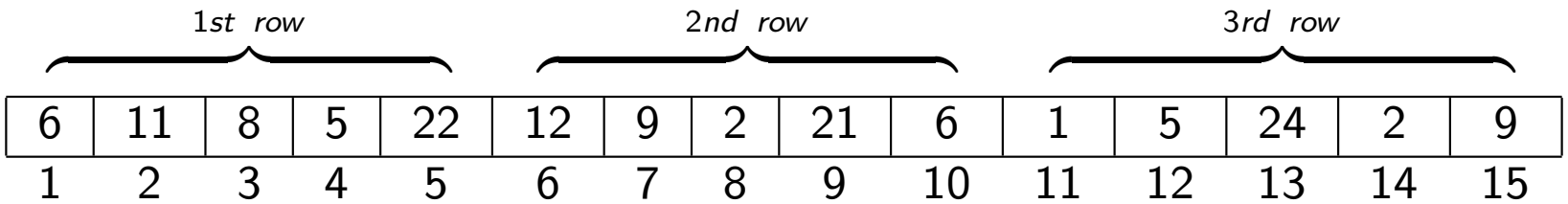
It reserves memory for 5 integers consecutively.

Matrix representation – row major order

Let M be the following matrix.

	1	2	3	4	5
1	6	11	8	5	22
2	12	9	2	21	6
3	1	5	24	2	9

The row major order representation is:



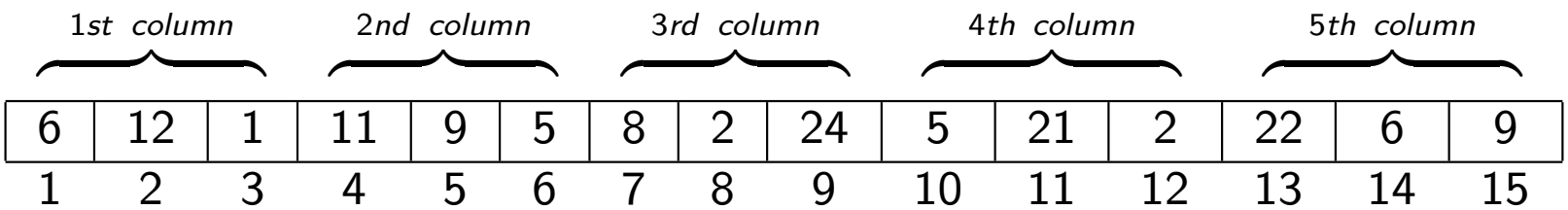
Row major order representation supports direct access.

Matrix representation – column major order

Let M be the following matrix.

	1	2	3	4	5
1	6	11	8	5	22
2	12	9	2	21	6
3	1	5	24	2	9

The column major order representation is:



Column major order representation supports direct access.

Sparse matrix

Sparse matrix is a matrix in which most of the elements are zero.

For example, let M be the following sparse matrix.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	3	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	11	0	0	0	0	0	9	0
3	0	5	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	25	0	0	0

Representation?

Sparse matrix – 3 row representation

Let M be the following sparse matrix.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	3	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	11	0	0	0	0	0	9	0
3	0	5	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	25	0	0	0

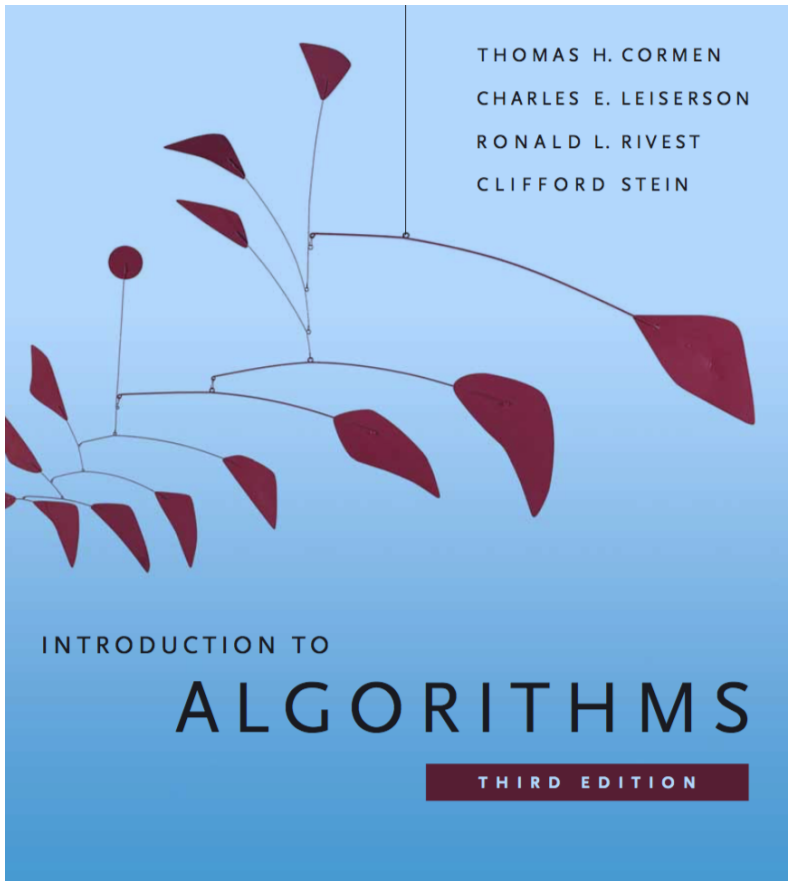
The 3-row representation is:

row	1	2	2	3	4
column	4	7	13	2	11
value	3	11	9	5	25

Plus the size of the original matrix: (4,14).

3-row representation destroys direct access.

References



This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.