CM0889 Analysis of Algorithms
Introduction to Algorithms

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2020-2

Administrative Information

Course web page

http://www1.eafit.edu.co/asr/courses/cm0889-analysis-of-algorithms/

Textbook, evaluation, programming labs, course's repository, etc.

See course web page.

Preliminaries

Conventions

- The number assigned to chapters, examples, exercises, figures, sections, or theorems on these slides correspond to the numbers assigned in the textbook [Skiena 2012].
- The source code examples are in course's repository.

From the Textbook

Designing correct, efficient, and implementable algorithms for real-world problems requires access to two distinct bodies of knowledge:

- *Techniques*
  Good algorithm designers understand several fundamental algorithm design techniques.
- *Resources*
  Good algorithm designers stand on the shoulders of giants.

[Preface, p. v]

From Problems to Programs via Algorithms

Question

Can be any problem solved by a program?

From Problems to Programs via Algorithms

Question

Can be any problem solved by a program?

No!

- Limitations when specifying the problem (no precise specification)
- Computation limitations (theoretical or practical)
- Ethical considerations and regulations

From Problems to Programs via Algorithms

About solving problems

Half the battle is knowing what problem to solve. [Aho, Hopcroft and Ullman 1985, p. 1]

From Problems to Programs via Algorithms

About solving problems

Half the battle is knowing what problem to solve. [Aho, Hopcroft and Ullman 1985, p. 1]

Perhaps the single most important design technique is modeling, the art of abstracting a messy real-world application into a clean problem suitable for algorithmic attack. [p. v]

From Problems to Programs via Algorithms

Steps when writing a computer program to solve a problem

- Problem formulation and specification
- Design of the solution (algorithm)
- Implementation
- Testing
- Documentation
- Evaluation
- Maintenance

In software engineering the above steps are part of the software development life cycle.

From Problems to Programs via Algorithms

Definition (common and informal)

*An **algorithm**, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

From Problems to Programs via Algorithms

Definition (common and informal)

> *An **algorithm**, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

Question

Are missing the computers on the above definition of algorithm?

From Problems to Programs via Algorithms

Definition (common and informal)

> *An **algorithm**, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

Question

Are missing the computers on the above definition of algorithm? No!

From Problems to Programs via Algorithms

Definition (common and informal)

*An **algorithm**, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

Question

Are missing the computers on the above definition of algorithm? No!

Question

What is an instruction?

From Problems to Programs via Algorithms

Definition (common and informal)

An ***algorithm****, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

Question

Are missing the computers on the above definition of algorithm? No!

Question

What is an instruction?

Remark

Any informal definition of algorithm necessary will be imprecise (but the above definition is enough for our course).

From Problems to Programs via Algorithms

Definition (common and informal)

> *An **algorithm**, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

From Problems to Programs via Algorithms

Definition (common and informal)

> *An **algorithm**, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. [Aho, Hopcroft and Ullman 1985, p. 1]*

Discussion

Is any computer program the implementation of some algorithm?

From Problems to Programs via Algorithms

Paradigms of programming

Imperative/object-oriented: Describe computation in terms of state-transforming operations such as assignment. Programming is done with statements.

Logic: Predicate calculus as a programming language. Programming is done with sentences.

Functional: Describe computation in terms of (mathematical) functions. Programming is done with expressions.

Examples

Imperative/OO:	C, C++, Java, Python
Logic:	CLP(R), Prolog
Functional:	Erlang, Haskell, ML

From Problems to Programs via Algorithms

Discussion

Does the algorithm for solving a problem depend of the programming language used for implementing it?

Example: Sorting

Introduction

A sorting algorithm is an algorithm that puts elements of list according to some linear (total) order. Sorting algorithms are fundamental in Computer Science.

Example: Sorting

Introduction

A sorting algorithm is an algorithm that puts elements of list according to some linear (total) order. Sorting algorithms are fundamental in Computer Science.

Sorting specification

Problem:	Sorting
Input:	A sequence of n keys $(a_1, a_2, \ldots, a_n)$.
Output:	A permutation (reordering) $(a'_1, a'_2, \ldots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example: Sorting

Example

Input:	$(154, 245, 568, 324, 654, 324)$
Output:	$(154, 245, 324, 324, 568, 654)$

Example: Sorting

Example

Input:	$(154, 245, 568, 324, 654, 324)$
Output:	$(154, 245, 324, 324, 568, 654)$

Example

Input:	(Mike, Bob, Sam, Jill, Jan)
Output (lexicographic order):	(Bob, Jan, Jill, Mike, Sam)
Output (shortlex order):	(Bob, Jan, Sam, Jill, Mike)

Example: Sorting

An algorithm: Insertion sort

> *Insertion sort is a method for sorting that starts with a single element (thus forming a trivially sorted list) and then incrementally inserts the remaining elements so that the list stays sorted. [p. 3]*

Example: Sorting

An algorithm: Insertion sort

> *Insertion sort is a method for sorting that starts with a single element (thus forming a trivially sorted list) and then incrementally inserts the remaining elements so that the list stays sorted. [p. 3]*

See simulation at `https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/visualize/`.

Example: Sorting

An algorithm: Insertion sort

> *Insertion sort is a method for sorting that starts with a single element (thus forming a trivially sorted list) and then incrementally inserts the remaining elements so that the list stays sorted. [p. 3]*

See simulation at `https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/visualize/`.

See implementation and tests in `sorting-test.c`.

Example: Robot Tour Optimisation

Specification

Problem:	Robot Tour Optimisation
Input:	A set P of n points in the plane.
Output:	The shortest cycle tour that visits each point in the set P.

See Skiena's lecture slides: Introduction to Algorithms

Example: Robot Tour Optimisation

Remarks about the closest-pair heuristic

In a vertex chain $(v_1, v_2, ..., v_n)$, the vertices v_1 and v_n are the endpoints.

A single vertex chain is a vertex chain (v) with only a vertex.

In the nearest-neighbour heuristic, the algorithm only looks at the neighbours of the current vertex. In the closest-pair heuristic, the algorithm looks in all the neighbours of the partial vertex chain.

'The description states that every vertex always belongs either to a "single-vertex chain" (i.e., it's alone) or it belongs to one other chain; a vertex can only belong to one chain. The algorithm says at each step you select every possible pair of two vertices which are each an endpoint of the respective chain they belong to, and don't already belong to the same chain. Sometimes they'll be singletons; sometimes one or both will already belong to a non-trivial chain, so you'll join two chains.'*

*From https://stackoverflow.com/a/7216814/1709190.

Example: Selecting the Right Jobs

Specification

Problem:	Movie Scheduling Problem
Input:	A set I of n intervals on the line.
Output:	The largest subset of mutually non-overlapping intervals which can be selected from I.

See Skiena's lecture slides: Introduction to Algorithms

From the Previous Examples

Take-Home Lesson

There is a fundamental difference between algorithms, which always produce a correct result, and heuristics, which may usually do a good job but without providing any guarantee. [p. 9]

From the Previous Examples

Take-Home Lesson

There is a fundamental difference between algorithms, which always produce a correct result, and heuristics, which may usually do a good job but without providing any guarantee. [p. 9]

Take-Home Lesson

Reasonable-looking algorithms can easily be incorrect. Algorithm correctness is a property that must be carefully demonstrated. [p. 11]

Reasoning about the Correctness of Algorithms

Prerequisites

A well-specified problem (input/output correspondence):

(i) The set of all allowed input instances.

(ii) The required properties of the problem's output.

Reasoning about the Correctness of Algorithms

Prerequisites

A well-specified problem (input/output correspondence):

(i) The set of all allowed input instances.

(ii) The required properties of the problem's output.

Correctness of an algorithm

Partial correctness: If an answer is returned it will be correct

Total correctness: Partial correctness + termination

Reasoning about the Correctness of Algorithms

Prerequisites

A well-specified problem (input/output correspondence):

(i) The set of all allowed input instances.
(ii) The required properties of the problem's output.

Correctness of an algorithm

Partial correctness: If an answer is returned it will be correct

Total correctness: Partial correctness + termination

Remark

The (mathematical) proof of the (partial or total) correctness of an algorithm can be a non-easy task.

Reasoning about the Correctness of Algorithms

Remark

Reasoning about the correctness/verification of algorithms is different to the correctness/verification of programs.

Reasoning about the Correctness of Algorithms

Remark

Reasoning about the correctness/verification of algorithms is different to the correctness/verification of programs.

Remark

We talk about formal verification when the underlying proof is machine-checked. For a current survey, see [Nipkow, Eberl and Haslbeck 2020].

Reasoning about the Correctness of Algorithms

Demonstrating incorrectness

A counter-example is a direct way to prove the incorrectness of a heuristic. Think about small/extreme/related-to-the-decision-criteria examples...

Reasoning about the Correctness of Algorithms

Induction and recursion

To every recursively defined set there is a correspond induction principle.

Example

Whiteboard.

Reasoning about the Correctness of Algorithms

Example

Prove the correctness of the following recursive algorithm for adding one to a natural number:

```
INCREMENT($n : \mathbb{N}$)
if $n == 0$
    return $1$
else
    if $(n \bmod 2 == 1)$
        return $(2 \cdot \text{INCREMENT}(\lfloor n/2 \rfloor))$
    else return $(n + 1)$
```

Reasoning about the Correctness of Algorithms

Proof by strong induction

Let $P(n)$: $\textsc{increment}(n) = n + 1$.

We need to prove that $(\forall n \in \mathbb{N})P(n)$.

(i) Basis case $P(0)$.

Since $\textsc{increment}(0) = 1$ by line 2, then $P(0)$ holds.

(ii) Inductive step $(P(1) \wedge P(2) \wedge \cdots \wedge P(k-1)) \rightarrow P(k)$.

a) Case: $k - 1$ is odd

Since k is even and $\textsc{increment}(k) = k + 1$ by line 6, then $P(k)$ holds.

Reasoning about the Correctness of Algorithms

Proof by strong induction (continuation)

ii) Inductive step $(P(1) \wedge P(2) \wedge \cdots \wedge P(k-1)) \rightarrow P(k)$.

b) Case: $k-1$ is even

The statement $(P(1) \wedge P(2) \wedge \cdots \wedge P(k-1)) \rightarrow P(k)$ holds because

$$\begin{aligned}
&\textsc{Increment}(k) & \\
&= \textsc{Increment}(2m+1) && \text{(k is odd)} \\
&= 2 \cdot \textsc{Increment}(\lfloor (2m+1)/2 \rfloor) && \text{(line 5)} \\
&= 2 \cdot \textsc{Increment}(\lfloor m + 1/2 \rfloor) && \text{(algebra)} \\
&= 2 \cdot \textsc{Increment}(m) && \text{(algebra)} \\
&= 2 \cdot (m+1) && \text{($m < k$ and IH)} \\
&= k+1 && \text{(algebra)}
\end{aligned}$$

References

- Aho, Alfred V., Hopcroft, John E. and Ullman, Jeffrey D. [1983] (1985). Data Structures and Algorithms. Reprinted with corrections. Addison-Wesley (cit. on pp. 7, 8, 10–16).
- Nipkow, Tobias, Eberl, Manuel and Haslbeck, Maximilian P. L. (2020). Verified Textbook Algorithms. A Biased Survey. In: Automated Technology for Verification and Analysis (ATVA 2020). Ed. by Hung, Dang Van and Sokolsky, Oleg. Lecture Notes in Computer Science. To appear. Springer (cit. on pp. 34, 35).
- Skiena, Steven S. [1997] (2012). The Algorithm Design Manual. 2nd ed. Corrected printing. Springer. DOI: 10.1007/978-1-84800-070-4 (cit. on p. 3).