ALGORITHMS

Basic Concepts

Juan Mendivelso
Assistant Professor

Universidad Nacional de Colombia
Facultad de Ciencias
Departamento de Matemáticas

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Program

- Sequence of **instructions** written to perform a specified task with a **computer**.
- A computer requires programs to function.
- Computer **source code** is written by computer **programmers**.
- Source code is written in a **programming language**.
- Source code may be converted into an **executable file** by a **compiler**.

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Computational Problem

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Sorting Problem

- **Input**: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
- **Output**:A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Algorithm

- Well-defined computational procedure that takes some set of **instances** and produces some set of **solutions**.
- Sequence of computational steps that transform the **input** into the **output**.
- Tool for solving a well-specified computational problem.

Example

For the instance $\langle 31, 41, 59, 26, 41, 58 \rangle$, a sorting algorithm must produce $\langle 26, 31, 41, 41, 58, 59 \rangle$.

Correctness

Correct Algorithm

- An algorithm is **correct** iff, for every input instance, it halts with the correct output.
- A **correct** algorithm **solves** the given computational problem.

Incorrect Algorithm

An algorithm is **incorrect** if

- It does not halt at all on some input instances.
- It halts with an answer other than the desired one.

An **incorrect** algorithm may be useful if the error rate can be controlled.

Origin of the word Algorithm

- The word **algorithm** stems from **Al-Khwarizmi**.
- Al-Khwarizmi was a Persian mathematician, astronomer y geographer who lived between 780 and 850.
- He worked on written processes to achieve a goal.

Algorithms & Computers

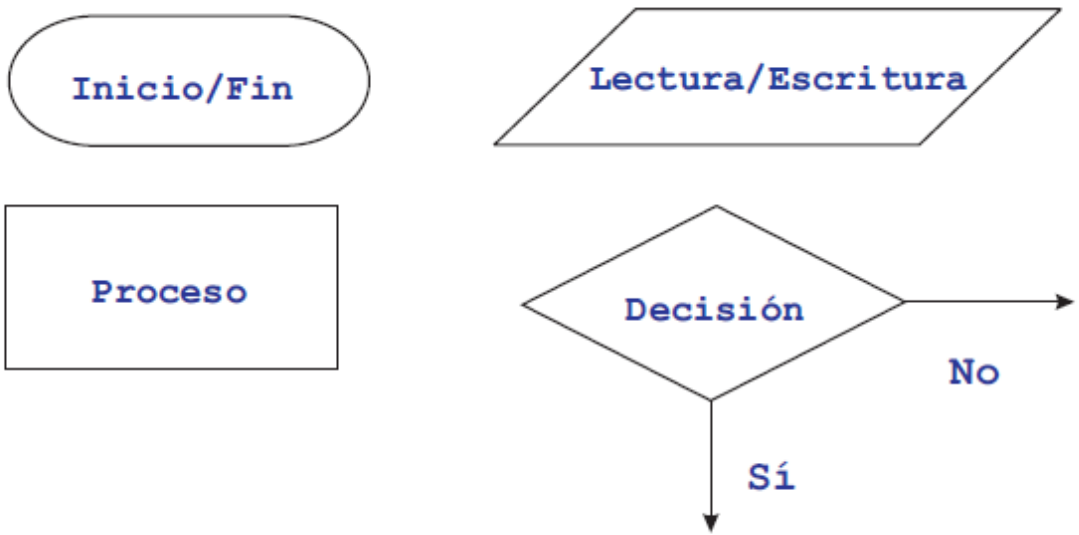
- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Specification

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

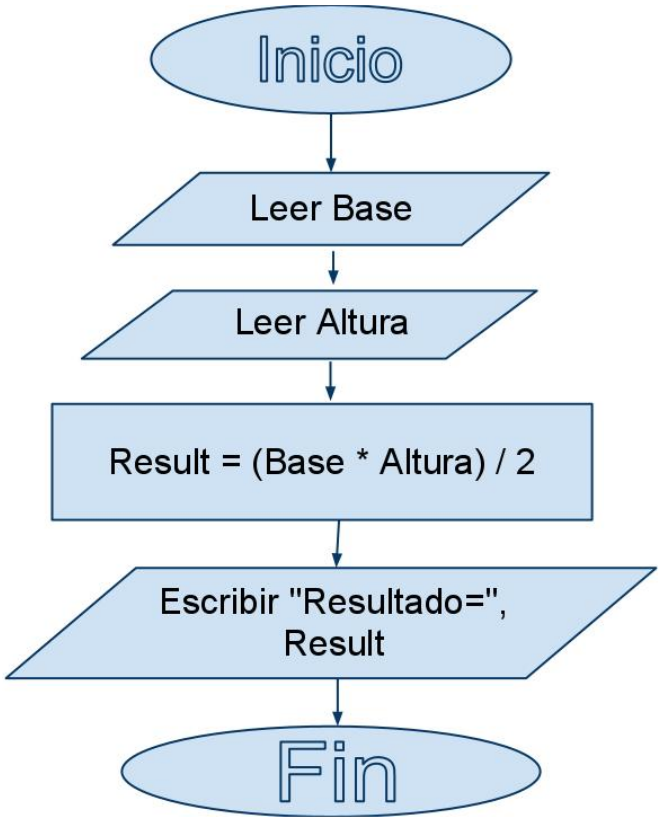
Data Flow Diagram (DFD)

- Graphic representation of a step sequence.
- Each operation is represented by a given figure established by the ANSI.
- Useful for small algorithms or the tricky parts.



Data Flow Diagram (DFD)

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.
	Desicion	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/Function	Used to represent a group of statements performing one processing task.



Pseudocode

- Similar to code, but not as strict.
- Similar to natural language.
- More compact than a DFD. It can represent more complex instructions.
- It is easier to translate into a computer program.
- It does not follow any standard.

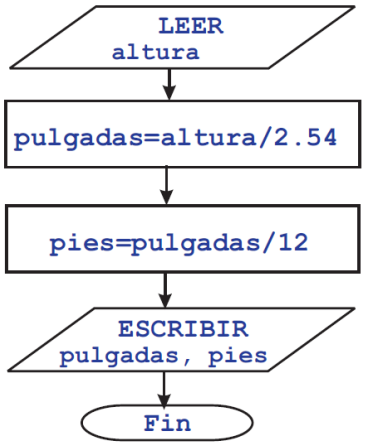
Algoritmo 5.3 Cálculo del valor de la función $f(x) = 0$ si $x \leq 0$, $f(x) = x^2$ si $x > 0$.

```
Inicio
   1- LEER x
   2- HACER f=0
   3- Si x>0
        HACER f=x^2
      Fin Si
   4- IMPRIMIR 'El valor de la funcion es: ', f
Fin
```

Pseudocode Data Flow Diagrams

Algoritmo 5.2 Calcular una altura en pulgadas (1 pulgada=2.54 cm) y pies (1 pie=12 pulgadas), a partir de la altura en centímetros, que se introduce por el teclado.

```
Inicio
   1- IMPRIMIR 'Introduce la altura en centimetros: '
   2- LEER: altura
   3- CALCULAR pulgadas=altura/2.54
   4- CALCULAR pies=pulgadas/12
   5- IMPRIMIR 'La altura en pulgadas es: ', pulgadas
   6- IMPRIMIR 'La altura en pies es : ', pies
Fin
```



Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example,
  - `Heap-Sort`
  - `Insertion-Sort`
  - `Merge-Sort`
  - `Quick-Sort`
- Which algorithm is better for a given application? It depends upon:
  - The number of items to be sorted
  - The extent to which the items are already somewhat sorted
  - Possible restrictions on the item values

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Data Structure

- A data structure is a way to store and organize data in order to facilitate access and modifications.
- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them in order to solve computational problems.

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Efficiency (I)

- Efficiency is the extent to which time or effort is well used for the intended task or purpose.
- In the study of algorithm, we try to reduce the time and the memory required.

If computers were infinitely fast and computer memory was free... Would you still need to study algorithms?

Efficiency (II)

- It is not the reality!
- Computer time and space in memory are bounded resources.
- Algorithms devised to solve the same problem often differ dramatically in their efficiency.
- These differences can be much more significant than differences due to hardware and software.

Example

- `Insert-Sort` takes $c_1 n^2$ instructions; `Merge-Sort` takes $c_2 n \log n$ instructions. The number of instructions is proportional to the time required.
- Generally, $c_1 < c_2$.

Efficiency (III)

Example

- Suppose the time required for `Insertion-Sort` is $2n^2$ instructions and it is run on computer A.
- Suppose the time required for `Merge-Sort` is $50n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time is required by each one of them, when $n = 1.000.000$?.

Efficiency (IV)

Example

- Suppose the time required for `Insertion-Sort` is $2n^2$ instructions and it is run on computer A.
- Suppose the time required for `Merge-Sort` is $50n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time is required by each one of them, when $n = 10.000.000$?.

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Analysis of Algorithms

- Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.
- Resources such as memory, communication bandwidth, or computer hardware are of primary concern, but most often it is **computational time** that we want to measure.
- Generally, by analyzing several candidate algorithms for a problem, a most efficient one (or some) can be easily identified.
- **Requirements:** combinatorics, probability theory and algebra.

Implementation Technology (I)

- Before we can analyze an algorithm, we must have a model of the **implementation technology** that will be used.
- We will assume a generic one processor, **random-access machine (RAM)** model of computation (instructions are executed one after another).
- Strictly speaking, one should precisely define the instructions of the RAM model and their costs. Tedious!

Implementation Technology (II)

- We must be careful not to abuse the RAM model.
- Realistic operations are arithmetic, data movement (load, store, copy), and control (conditionals, subroutine call and return).
- Is exponentiation a realistic instruction?
- We will not model the memory hierarchy that is common in contemporary computers (caches, virtual memory).

More about Analyzing Algorithms

- The behavior of an algorithm may be different for each possible input.
- We need a means for summarizing that behavior in simple, easily understood formulas.
- There are many choices in deciding how to express our analysis. Choose the simplest that shows important characteristics!
- The time taken by an algorithm grows with the size of the input.

Input Size

- For many problems, such as sorting the most natural measure is the number of items in the input.
- For multiplying two integers, the best measure of input size is the total number of bits needed to represent the input in ordinary binary notation.
- Sometimes, it is more appropriate to describe the size of the input with two numbers rather than one. For example, graphs ($|V|$ and $|E|$)

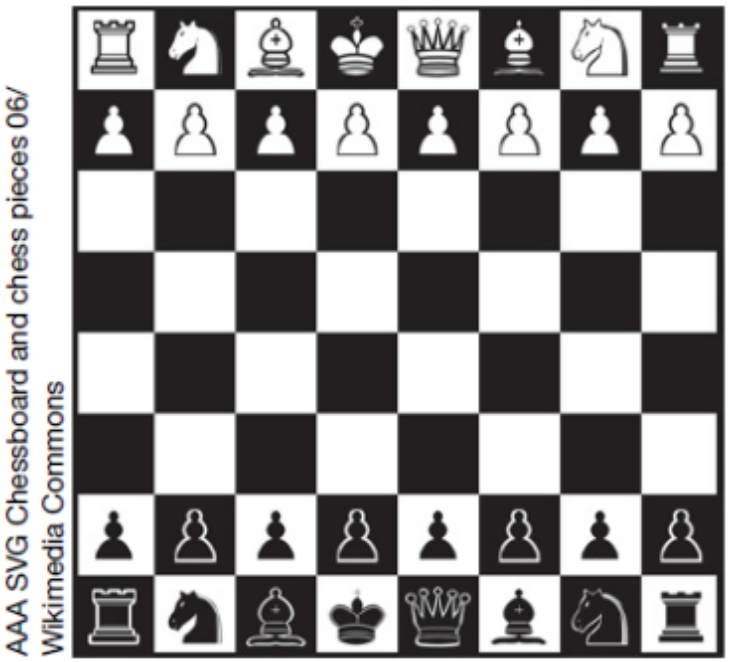
Running time

- The number of **primitive operations** or **steps** executed.
- It is convenient to define the notion of step so that it is as **machine-independent** as possible.
- Each execution of the i-th line takes time c_i where c_i is a constant. Some operations like calling a subroutine may take more than constant time, though.
- Then, we count the number of times each line is executed.
- This allows us to estimate the number of primitive operations performed.

Why should we analyze an algorithm?

- We want to know about its **scalability**: Does it support large input sizes?
- We want to know about its behavior on the best, worst and average-case.
- What mathematical functions describe its behavior?
- We use **asymptotic notation**.
- It allows to establish what is feasible and what is not feasible.

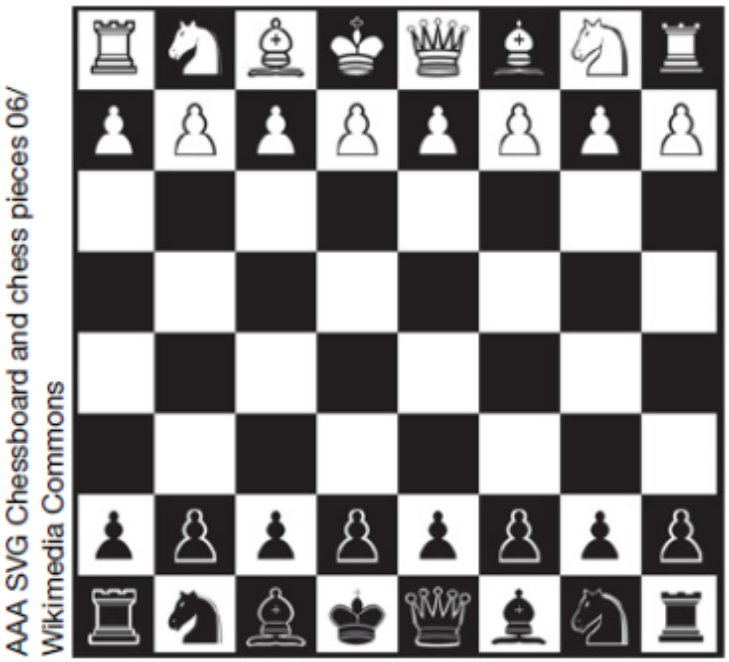
How to succeed at chess?



Given a chessboard, what is the best move I can make?

A brute-force algorithm would have to consider all the possible moves after the current move.

How to succeed at chess?



There are 10^{120} possible chess games.

There are 10^{80} atoms in the observable universe.

It is required 3×10^{90} grains of sand to fill the solid universe.

But we have to solve the problem in reasonable time.

There are **approximation algorithms**.

Outline

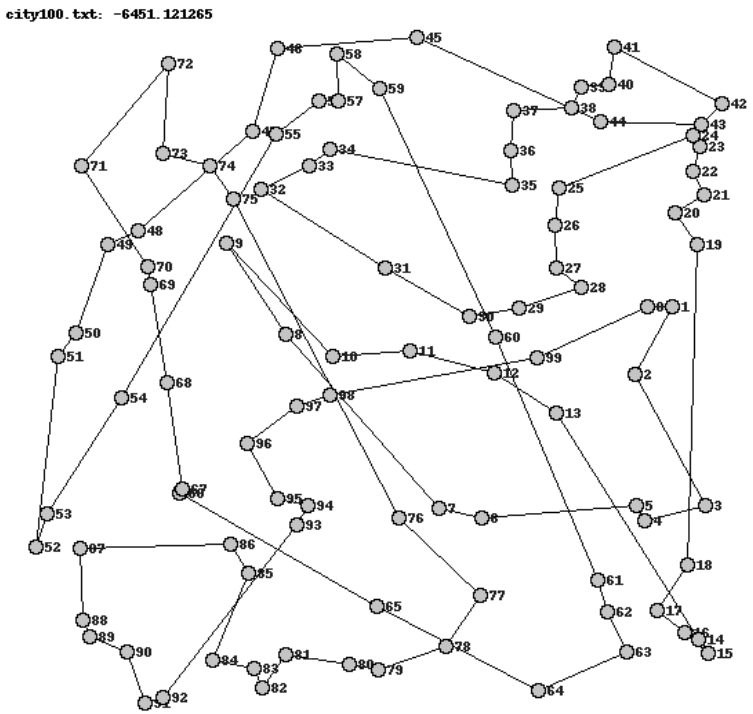
1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

NP-Complete Problems

- A NP-complete problem is a problem for which no efficient solution is known.
- Nobody has ever proven that an efficient algorithm for one cannot exist.
- The set of NP-complete problems has the remarkable property that if an efficient algorithm exists for any one of them, then efficient algorithms exist for all of them.
- Several NP-complete problems are similar, but not identical, to problems for which we do know of efficient algorithms.

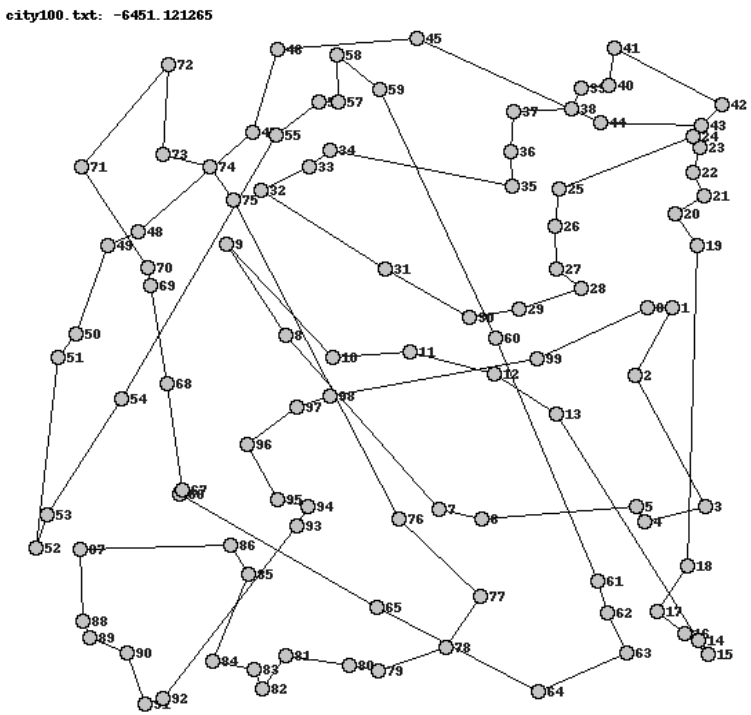
The Travelling-Salesman Problem (TSP)

- Some of them arise surprisingly often in real applications.
- An example is the **Travelling-Salesman Problem**. He must visit all the cities traversing the minimum distance.
- Good solutions through approximation algorithms.
- How many options should a brute force algorithm consider?



The Travelling-Salesman Problem (TSP)

- $10! = 3.628.800.$
- $20! = 2.432.902.008.640.000.$
- $30!$ is greater than 10^{64}.
- There are approximation algorithms.

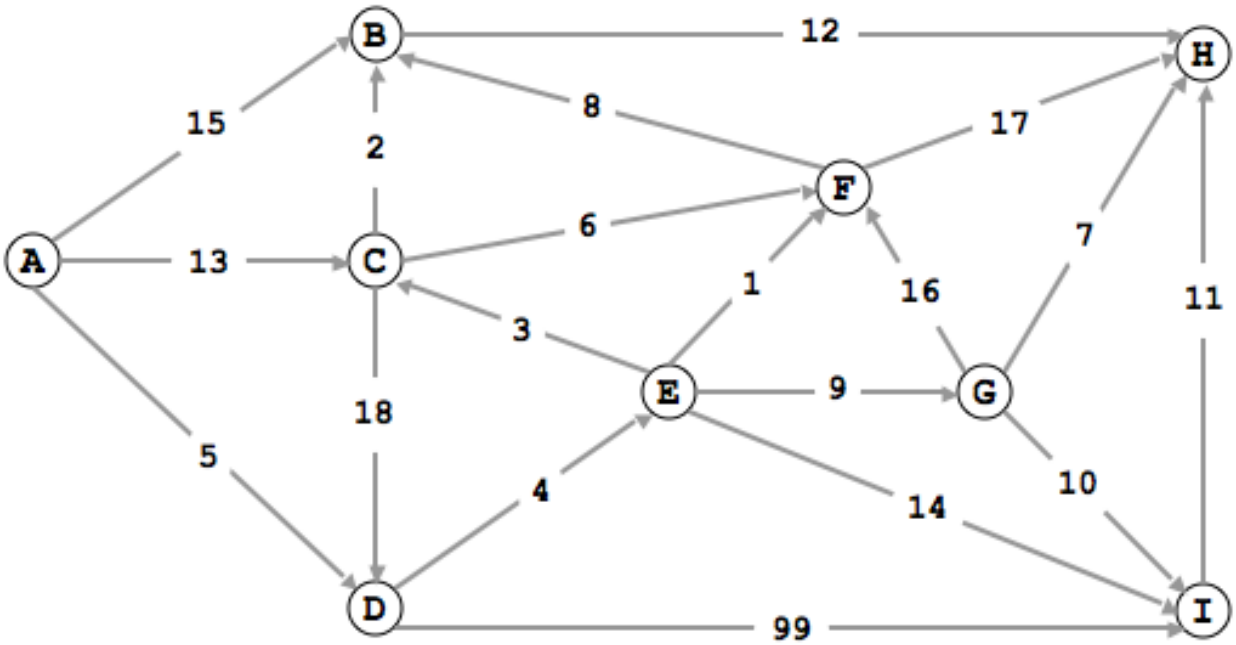


Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

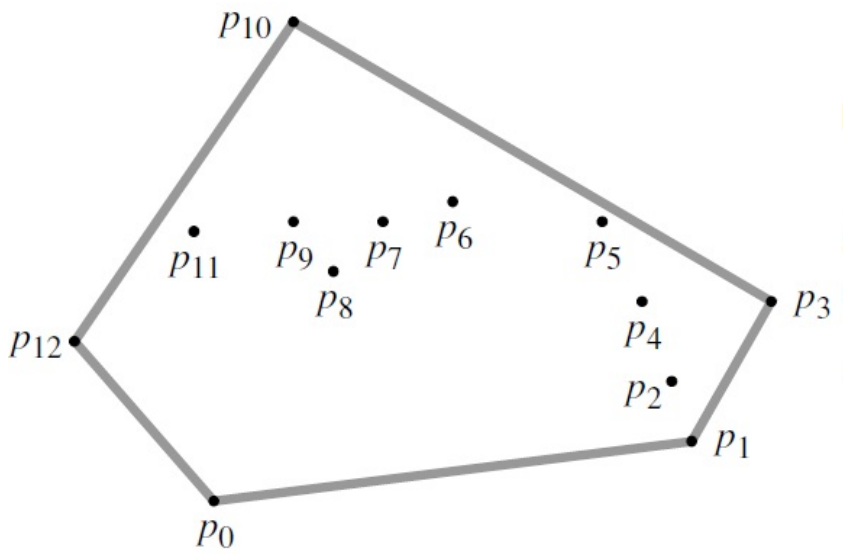
Graphs

Finding the shortest path between two vertices in a graph.



Computational Geometry

Finding the convex hull of a set of points p_1, p_2, ..., p_n through Computational Geometry. The convex hull is the smallest convex polygon containing the points.



Dynamic Programming and Number Theory

Multiplying matrices

Given a set of matrices A_1, A_2, ..., A_n, find the order with the lowest number of calculations to calculate $A_1 \times A_2 \times \ldots A_n$.
Technique: Dynamic Programming

Equations with modulo

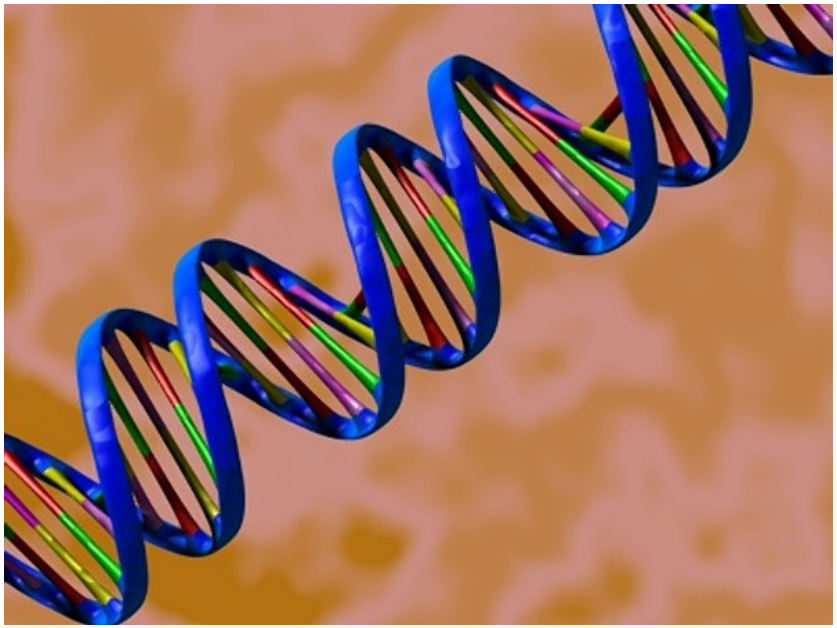
Solve $ax \equiv b(modn)$ through Number Theoretic Algorithms.

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

The Human Genome Project

- Identify genes in chemical base pairs.
- Storing information and data analysis.
- Savings in money and time.



Internet

- Access and retrieve large amounts of information.
- Good routes for the data to travel.
- String Matching and Hash Tables.

Electronic Commerce

- Online negotiation of goods and services.
- It is required to keep privacy.
- Number theoretic algorithms: public key cryptography and digital signatures.

Manufacturing

- Allocate scarce resources in the most beneficial way.
- Linear Programming
- Examples:

Outline

1. Definitions
   - Computer Program
   - Computational Problem
   - Algorithm
   - Data Structure
   - Efficiency
   - Analysis of Algorithms
   - NP-Complete Problems
2. Specific Algorithmic Problems
3. Some Applications of Algorithms
4. Exercise

Exercise

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Bibliography

1. Cormen, TH, Leiserson CE, Rivest, RL, Stein C. **Introduction to Algorithms, 3rd Edition**. The MIT Press. 2009.
2. Dierbach, C. **Introduction to Computer Science using Python**. A computational problem-solving focus. Wiley. 2013.