

Advanced Algorithms Simple (for real)



Gabriel Rovesti

Summary

[2 Course Introduction 3](#_Toc159870439)

[3 Graphs Algorithms 5](#_Toc159870440)

[3.1 Terminology and Concepts 5](#_Toc159870441)

[3.2 Basic Problems, Notations and Properties 7](#_Toc159870442)

[4 Minimum Spanning Tree (MST) 9](#_Toc159870443)

[5 Shortest Path 10](#_Toc159870444)

[6 Approximation Algorithms 11](#_Toc159870445)

[7 Randomized Algorithms 12](#_Toc159870446)

**Disclaimer**

# Course Introduction

(Usual general fluff, then comes the interesting part)

Algorithms have a general motivation: create a network of knowledge and allow, with pacing of times, different development and stories creation, while crafting new solutions. We might define them as a sequence of steps to solve the most mundane problems but also really complex problems.

There are different kinds of *applications*:

* network routing
* bioinformatics
* economics (e.g., game theory)
* fluid dynamics
* data mining
* cryptography
* machine learning

The point is this: even when making interviews, algorithms are both the logic and the solution to current problems, thinking *repeatedly and abstractly in a concrete (and fast) way*. Historically, there are still a lot of unsolved or still not found problems. That’s why the course is *mandatory*.

There are also different *goals*, wanting to introduce advanced principles of algorithm design and analysis. In particular, you’ll learn how to:

* Design algorithms for complex domains such as *graphs*
* Recognize “hard” problems and address them using *approximation algorithms*
* Use the power of *randomness* to design fast algorithms
  + and analyze them with appropriate mathematical tools

The *contents* of the course will be the following:

* (Basic) Graph algorithms
  + Graph search and its applications, minimum spanning trees, shortest paths, maximum flows 2 Approximation algorithms
* *Intractable* problems (not solvable in a reasonable amount of time)
  + NP-hardness and reductions between problems
  + Approximation algorithms for intractable problems
    - such as vertex cover, set cover, and the traveling salesperson problem
* Randomized algorithms
  + Main design techniques and analysis tools
    - with applications to problems such as sorting and minimum cuts

Although there are no formal prerequisites, an undergraduate course in algorithms and a good knowledge of (discrete) probability are assumed. Specifically, you should be familiar with:

* *Algorithm design techniques*: divide and conquer, greedy, dynamic programming
* *Data structures*: lists, stacks, queues, binary trees, search trees, heaps
* *Probability*: basic notions, discrete random variables

We want to discuss the *intuition* behind formulating algorithms and distill the core ideas making the algorithms work. Because we are computer scientists, we want to give *rigorousness*: algorithms without proofs are just conjecture and proofs give math logic to those.

We’ll follow, in part, an “active learning” approach:

* Will foster and encourage interaction during class
* Will frequently conclude class with 1-2 exercises
  + whose solution will be shown only at the beginning of next class
* Will frequently post on Moodle further readings
  + news/surveys/research articles/videos related to the topics covered in class
* There is no lab or coding assignments
  + but you are encouraged to code your favorite algorithms up and run them on real data

If you read until here, you sure wanna know: how is the exam?

* Written test, 2 hours. It consists of:
  + 3 questions
    - theory questions on the topics covered in class
    - aimed at verifying the student’s knowledge of the contents of the course
  + 2 problems
    - problems whose solution *requires some creativity*
    - aimed at verifying the student’s ability to use concepts
    - techniques learned during the course to solve new problems

# Graphs Algorithms

A graph is a repartition of the relationships between pairs of objects. In particular, we note:

* as the graph itself
  + = set of vertices (aka nodes)
  + (cartesian product = all) is a collection of edges
    - an edge is a pair of vertices
      * it indicates the connection between two nodes
      * a connection of vertices allows for repetition

In the following drawings, we find:

* directed graphs, which happens if
* undirected graphs, which happens if
* Immagine che contiene calligrafia, testo, inchiostro, Carattere

  Descrizione generata automaticamentearc = edge inside directed graphs (also called *directed edges*)

In this case, we’ll (mostly) use simple graphs, meaning:

* no parallel edges
* no self-loops

## Terminology and Concepts

We give some *terminology*:

* Given an edge
  + is incident on and (happens if vertex if one of endpoints in that edge)
  + and are adjacent (there is an edge between the two vertices)
* neighbors of a vertex: all vertices s.t.
  + all vertices directly connected to a given vertex by an edge
* degree of a vertex , denoted as or
  + the number of edges incident on

Examples of graphs: in many ways, graphs are the main modality of data we receive from nature.

* Road networks 🡪 (cities, roads)
* Computer networks 🡪 (computers, computers)
* World Wide Web (WWW) 🡪 (webpages, hyperlinks)
* Social networks (people, friendships relationships)
* Biological networks
  + e.g., molecules (atoms, chemical bonds)
  + e.g., brain (neurons, synapses)
* Finance 🡪 (accounts, transactions)

We give some concepts also:

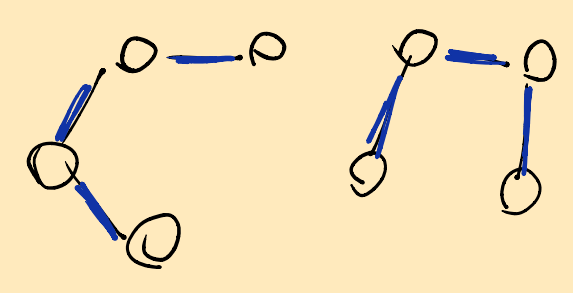
* path: and
  + finite/infinite sequence of edges which joins a sequence of vertices
* simple path: (all vertices) are all distinct
  + same definition as above and vertices are all distinct/so are the edges
* cycle: simple path s.t. (starts from a given vertex/ends at same vertex)
* subgraph:
  + the edges of are incident only on vertices of
  + in words: it is a subset of the larger original graph
* spanning subgraph: a subgraph with
  + a subgraph which “spans” the original graph (so there are all the vertices)
  + following other definitions
    - subgraph obtained by edge deletions only but retaining all vertices
    - so it’s a subgraph of with same vertex set as
* connected graph: if a path from to
* Immagine che contiene calligrafia, Carattere

  Descrizione generata automaticamenteconnected components: a partition of in subgraphs
  + is connected
  + there is no edge between and
* tree: connected graph without cycles
  + Immagine che contiene schizzo, disegno, clipart, Line art

    Descrizione generata automaticamenteany two vertices are connected by *exactly* one path
* Immagine che contiene schizzo, disegno, Arte bambini, arte

  Descrizione generata automaticamentespanning tree: a spanning subgraph connected and without cycles

(in figure: it exists only if is connected)

* spanning forest: a spanning subgraph without cycles
  + forest = undirected graph in which any two vertices are connected by *at most* one path

## Basic Problems, Notations and Properties

There are different *basic problems*:

* Traversal
* Connectivity (tell if the graph is connected or not)
* Computing connected components
* Spanning trees
* Minimum-weight spanning trees
* Shortest paths

Also consider some notations and properties:

* the size of this graph is
  + is not enough (normally online you would find it’s = count of edges)
  + consider scenario of graph with vertices and no edges
  + the size of graph would be but we don’t consider vertices

Exercise (Properties of graphs)

*Let be a simple, connected graph with vertices and edges. Then:*

1. *is a tree*
2. *is connected*
3. *is acyclic (i.e., is a forest)*

*Prove the previous properties*.

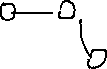
Solution

Consider an example for each one:

1. . The degree is the number of vertices incident in . The example clarifies it.



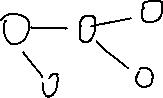
1. The number of edges in a complete graph with vertices in a complete graph is . This happens because in a complete graph, each vertex is connected to other vertices and we must divide by to avoid double-counting (since each edge is counted twice, once for each endpoint). So, the total number of edges in a complete graph with vertices is indeed
2. If is a tree, consider and, you would have:



1. If is connected we would have



1. Consider an acyclic graph, then



# Minimum Spanning Tree (MST)

# Shortest Path

# Approximation Algorithms

# Randomized Algorithms