

DSA-CL III – WINTER TERM 2018-19

DATA STRUCTURES AND ALGORITHMS FOR COMPUTATIONAL LINGUISTICS III

CLAUS ZINN

Çağrı Çöltekin



<https://dsacl3-2018.github.io>

DSA-CL III course overview

What is DSA-CL III?

- Intermediate-level survey course.
- Programming and problem solving, with applications.
 - **Algorithm:** method for solving a problem.
 - **Data structure:** method to store information.
- Second part focused on Computational Linguistics

Prerequisites:

- Data Structures and Algorithms for CL I
- Data Structures and Algorithms for CL II

Lecturers:

- Çağrı Cöltekin
- Claus Zinn

Tutors:

- Marko Lozajic
- Michael Watkins

Slots:

- Mon 12:15 & 18:00
- Wed 14:15 — 18:00 (lab)

Course Materials: <https://dsacl3-2018.github.io>

Why study algorithms?

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

Security. Cell phones, e-commerce, voting machines, ...

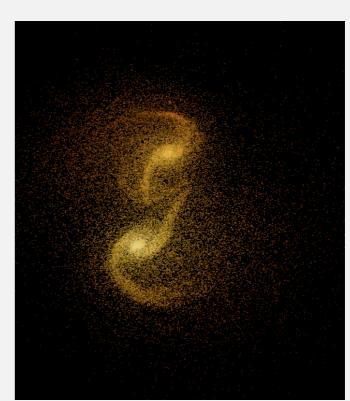
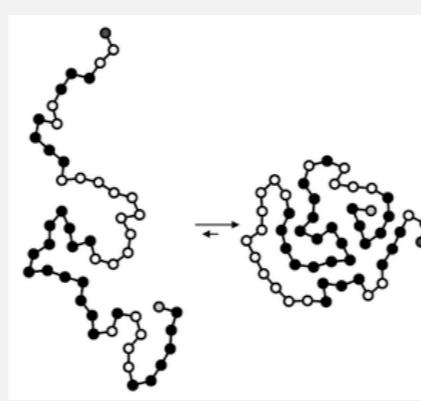
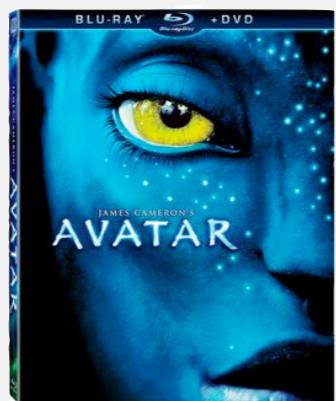
Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮

Google™
YAHOO![®]
bing™



Why study algorithms?

Their impact is broad and far-reaching.

Mysterious algorithm was 4% of trading activity last week

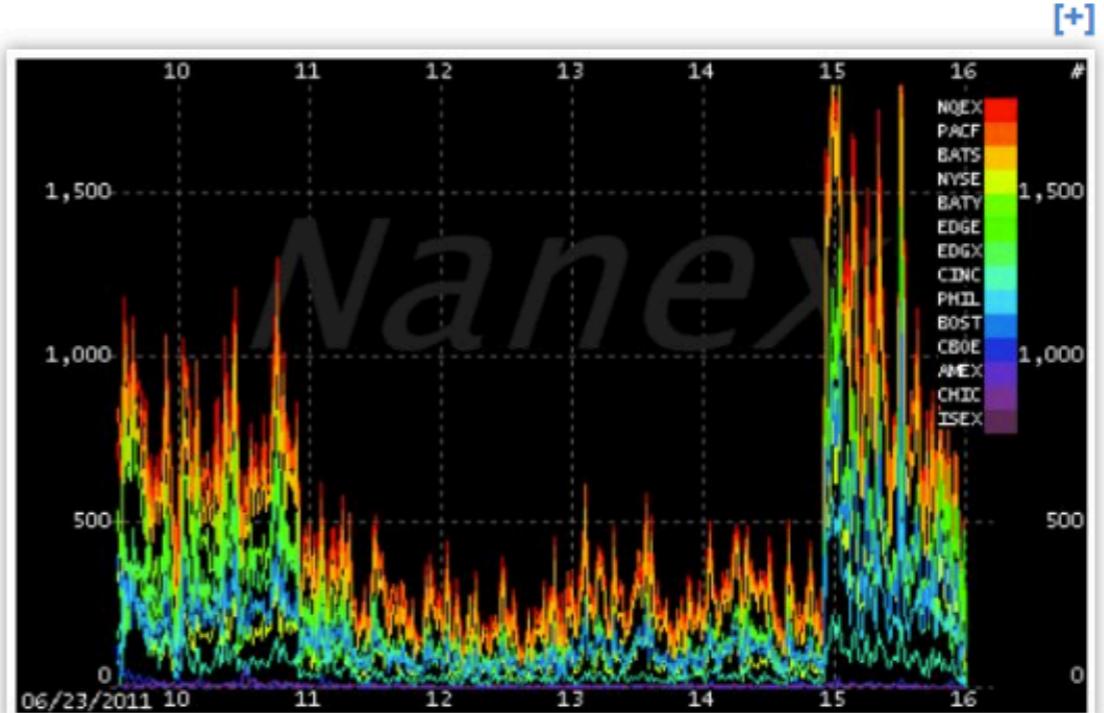
October 11, 2012

A single mysterious computer program that placed orders — and then subsequently canceled them — made up 4 percent of all quote traffic in the U.S. stock market last week, according to the top tracker of [high-frequency trading](#) activity.

The motive of the algorithm is still unclear, [CNBC](#) reports.

The program placed orders in 25-millisecond bursts involving about 500 stocks, according to Nanex, a market data firm. The algorithm never executed a single trade, and it abruptly ended at about 10:30 a.m. ET Friday.

"My guess is that the algo was testing the market, as high-frequency frequently does," says Jon Najarian, co-founder of TradeMonster.com. "As soon as they add bandwidth, the HFT crowd sees how quickly they can top out to create latency." ([Read More: Unclear What Caused Kraft Spike: Nanex Founder.](#))



Generic high frequency trading chart (credit: Nanex)

Why study algorithms?

For intellectual stimulation.

“For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.” — Francis Sullivan

FROM THE
EDITORS

THE JOY OF ALGORITHMS

Francis Sullivan, Associate Editor-In-Chief



THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF COMPUTING IN SCIENCE & ENGINEERING IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—to call the 10 examples we've selected “THE TOP 10 ALGORITHMS OF THE CENTURY.”

Computational algorithms are probably as old as civilization. Sumerian cuneiform, one of the most ancient written records, contains what may be the first algorithm, for calculating the area of a trapezoid. It was probably used for surveying land plots. And I suppose we could claim that the Druid algorithm for estimating the start of summer is enshrined in Stonehenge? That's probably not true, but it's a nice thought.

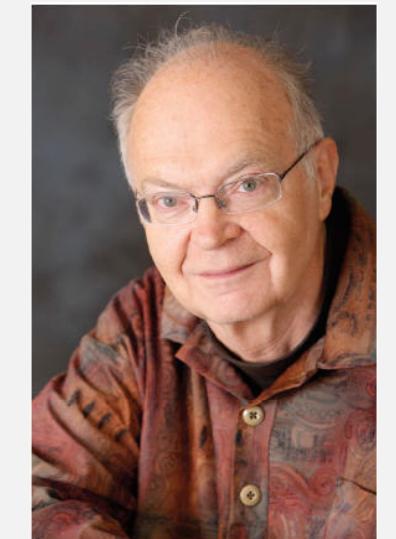
Like so many other things that technology affects, algorithms have advanced in startling and unexpected ways in the past few decades. In fact, the 10 algorithms of the century we chose for this issue have been essential for progress in communications, health care, manufacturing, economics, and more. The search for better algorithms, and the variety of progress in these areas has stimulated the search for even more algorithms. Recall one late-night session on the March 1998 cover: “What’s the algorithm?” “What’s the error?” After all, they don’t look very appealing. After the usual speculations about the behavior of six gulls, someone gave what must be the right answer: “A very large person first ate a crab.”

The search for better algorithms is the search for “better invention creates its own increasing.” Our need for powerful machines always exceeds their availability. Each significant computational advance creates a demand for more powerful and larger, competition to be done. New algorithms are an attempt to bridge the gap between the demand for cycles and the available data. In spite of a lot of good work, the question of how much data is still almost untouched. There are still very big challenges coming from more “traditional” tasks, too. For example, progress in speech recognition has been slow. After a decade of Moore’s Law factor of two every 18 months, he effect, Moore’s Law changes the constant in front of the estimate of running time. If you’re running at 100 cycles per second, and you do not come along every 1.5 years, but when they do, they can change the exponent of the complexity?

For now, the search for better invention in the theory of computation is still small, but the added confidence in its power is large. Is there an analog for things such as large, multidisciplinary teams that have cracked many hard problems since January 1990, but we are posing some even harder ones on the next century. In spite of a lot of good work, the question of how much data is still almost untouched. There are still very big challenges coming from more “traditional” tasks, too. For example, progress in speech recognition has been slow. After a decade of Moore’s Law factor of two every 18 months, he effect, Moore’s Law changes the constant in front of the estimate of running time. If you’re running at 100 cycles per second, and you do not come along every 1.5 years, but when they do, they can change the exponent of the complexity?

The next century is not going to be very fruitful for us, but it is not going to be dull either. ■

“An algorithm must be seen to be believed.” — Donald Knuth

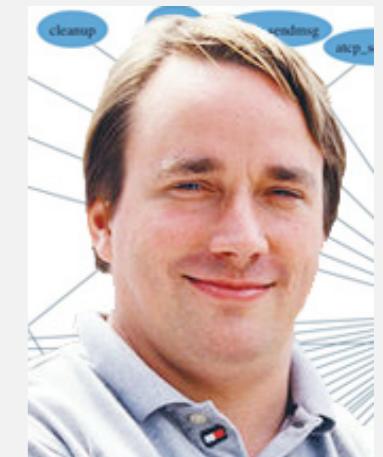


Why study algorithms?

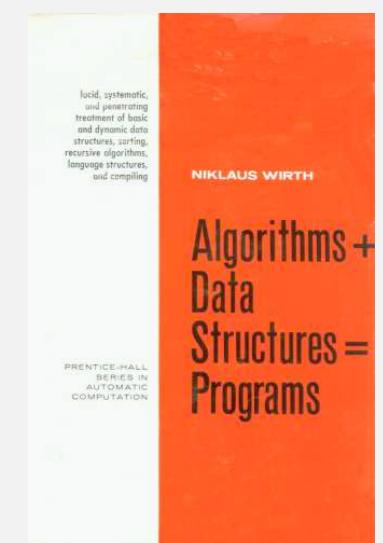
To become a proficient programmer.

“I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. ”

— Linus Torvalds (creator of Linux)



“Algorithms + Data Structures = Programs. ” — Niklaus Wirth



Why study algorithms?

They may unlock the secrets of life and of the universe.

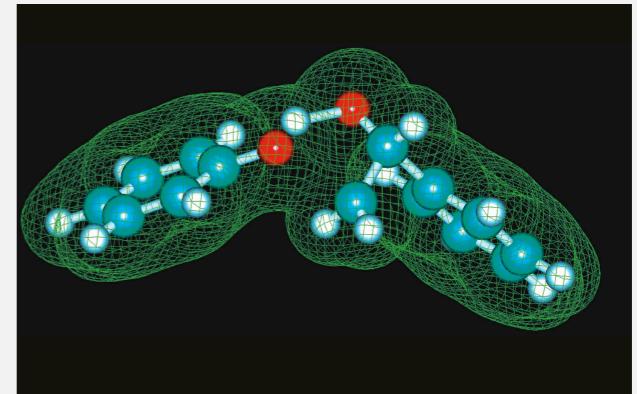
“ Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube. ”

— Royal Swedish Academy of Sciences

(Nobel Prize in Chemistry 2013)

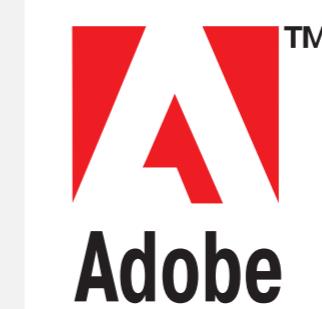
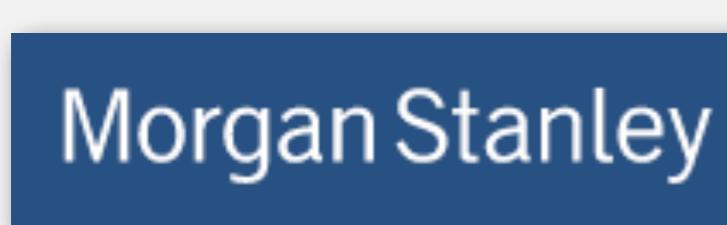
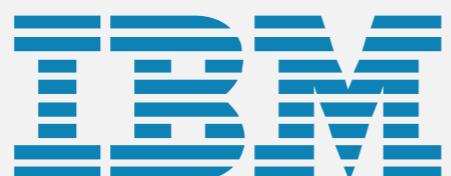


Martin Karplus, Michael Levitt, and Arieh Warshel



Why study algorithms?

For fun and profit.



Why study algorithms?

- Their impact is broad and far-reaching.
- Old roots, new opportunities.
- For intellectual stimulation.
- To become a proficient programmer.
- They may unlock the secrets of life and of the universe.
- To solve problems that could not otherwise be addressed.
- Everybody else is doing it.
- For fun and profit.

Why study anything else?



Coursework and grading

Reading material for most lectures

Weekly programming assignments

Four graded assignments. 60%

- Due on Tuesdays at 11pm via electronic submission (Github Classroom)
- Collaboration/lateness policies: see web.

Written exam. 40%

- Midterm practice exam 0%
- Final exam 40%

Honesty Statement

Honesty statement:

- Feel free to cooperate on assignments that are not graded.
- Assignments that are graded must be your own work. Do **not**:
 - Copy a program (in whole or in part).
 - Give your solution to a classmate (in whole or in part).
 - Get so much help that you cannot honestly call it your own work.
 - Receive or use outside help.
- Sign your work with the honesty statement (provided on the website).
- Above all: You are here for yourself, practice makes perfection.

Organisational issues

Presence:

- A presence sheet is circulated **purely** for statistics.
- Experience: those who do not attend lectures or do not make the assignments usually fail the course.
- Do not expect us to answer your questions if you were not at the lectures.

Office hours:

- Office hour: **Monday, 14:00-15:00**, please make an **appointment!**
- Please ask questions about the material presented in the lectures during the lectures — Everyone benefits
- We will discuss each assignment that is not graded during the next lab.

Registration:

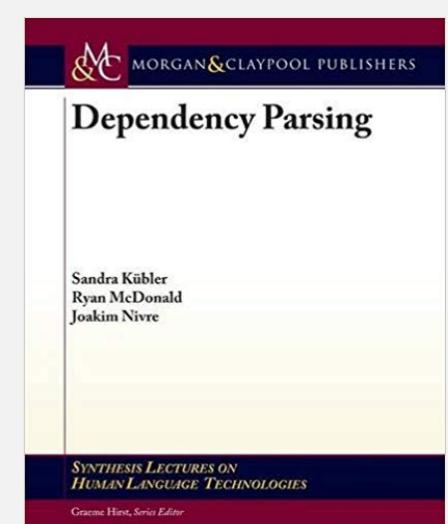
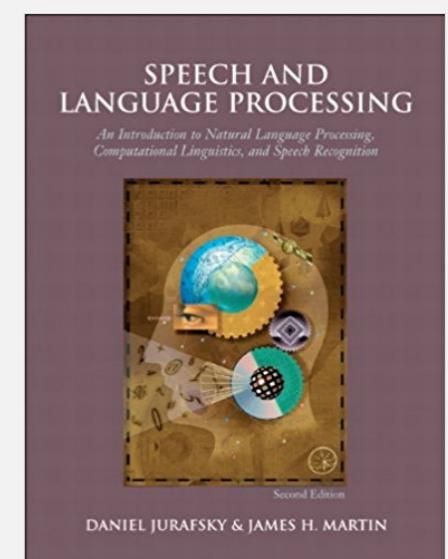
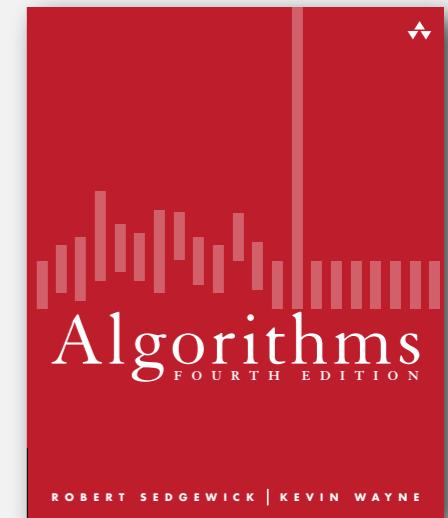
- Do the first assignment, A0.

Walk-Through GIT Classroom

Resources (textbook)

Required reading.

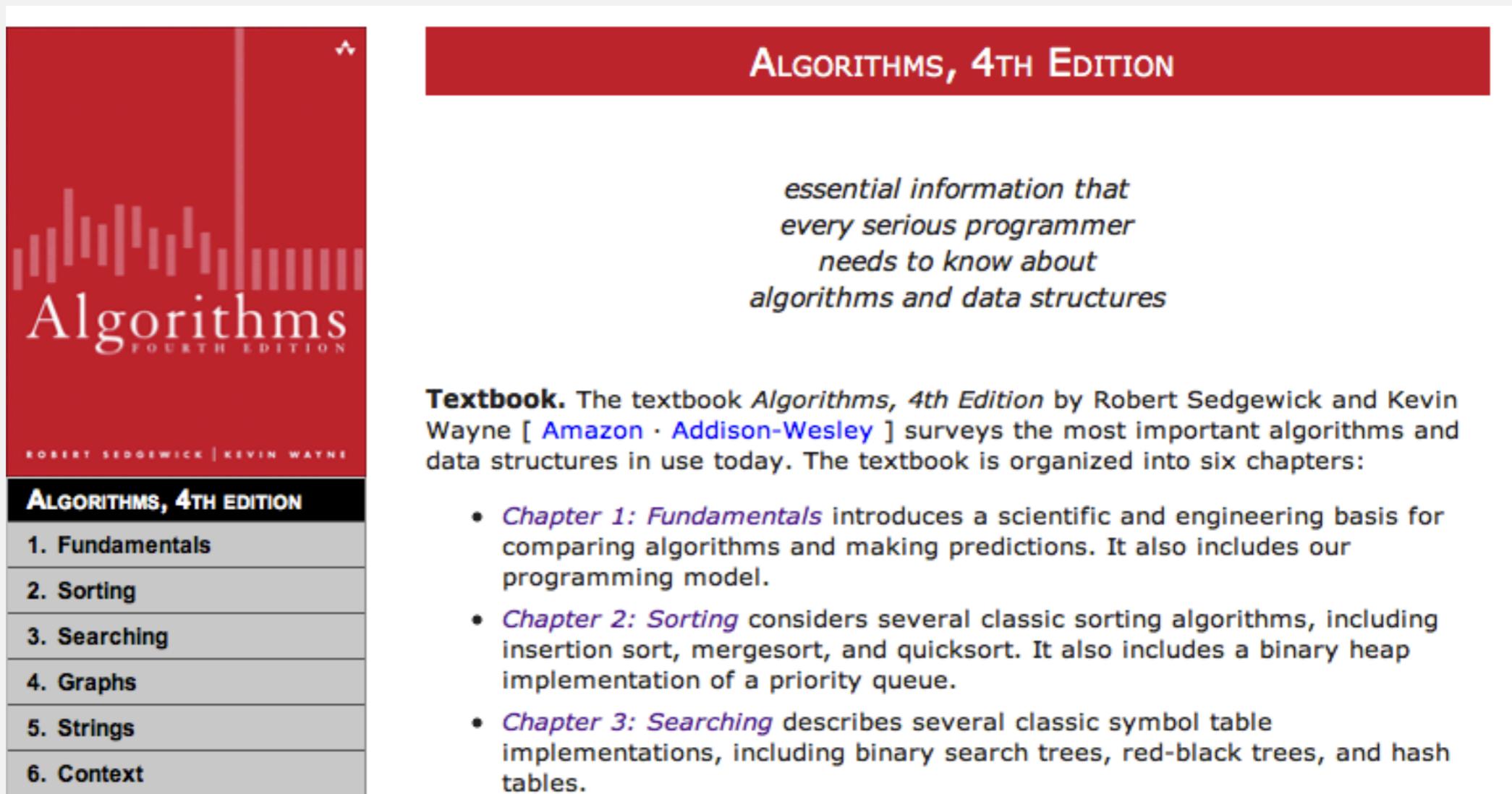
- Algorithms 4th edition by R. Sedgewick and K. Wayne,
Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.
 - Readable from university network thru Safari books:
 - see [proquest.tech.safaribooksonline.de/
9780132762571](http://proquest.tech.safaribooksonline.de/9780132762571)
- Speech and Language Processing, Jurafsky & Martin, 2nd Edition, Prentice Hall
 - Draft chapters of 3rd. edition available
 - see web.stanford.edu/~jurafsky/slp3/
- Dependency Parsing, Kübler, McDonald & Nivre, Morgan & Claypool



Resources (web)

Book site for first part of class

- Brief summary of content.
- Download code from book.
- APIs and Javadoc.



The screenshot shows the homepage of the *Algorithms, 4th Edition* website. On the left is a red sidebar featuring the book's cover image, which is a red rectangle with white vertical bars of varying heights, resembling a bar chart or a音波图. Below the cover, the word "Algorithms" is written in a large serif font, with "FOURTH EDITION" in smaller letters underneath. At the bottom of the sidebar, it says "ROBERT SEDGEWICK | KEVIN WAYNE". To the right of the sidebar is a white main content area. At the top of this area is a red header bar with the text "ALGORITHMS, 4TH EDITION" in white. Below the header, there is a large, italicized quote: "essential information that every serious programmer needs to know about algorithms and data structures". Further down, under the heading "Textbook.", there is a paragraph describing the book as a survey of important algorithms and data structures, organized into six chapters. To the right of this paragraph is a vertical sidebar containing a table of contents with six items: 1. Fundamentals, 2. Sorting, 3. Searching, 4. Graphs, 5. Strings, and 6. Context.

Textbook. The textbook *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne [[Amazon](#) · [Addison-Wesley](#)] surveys the most important algorithms and data structures in use today. The textbook is organized into six chapters:

- *Chapter 1: Fundamentals* introduces a scientific and engineering basis for comparing algorithms and making predictions. It also includes our programming model.
- *Chapter 2: Sorting* considers several classic sorting algorithms, including insertion sort, mergesort, and quicksort. It also includes a binary heap implementation of a priority queue.
- *Chapter 3: Searching* describes several classic symbol table implementations, including binary search trees, red-black trees, and hash tables.

What's ahead

Week	Monday (lectures)	Wednesday (lab)	
01	Oct 22 A: <i>Introduction</i> B: <i>Complexity theory</i>	Oct 24 lab: <i>Language Guessing</i>	
02	Oct 29 A: <i>Elementary sorts</i> B: <i>Quicksort</i>	Nov 1 <i>No class</i>	
03	Nov 05 A: <i>Undirected graphs</i> B: <i>Undirected graphs</i>	Nov 07 lab: <i>Sorting</i>	
04	Nov 12 A: <i>Directed graphs</i> B: <i>Directed graphs</i>	Nov 14 lab: <i>Undirected graphs</i>	
05	Nov 19 A: <i>Distance measures</i> B: <i>Binary heaps & heapsort</i>	Nov 21 lab: <i>Directed graphs</i>	
06	Nov 26 A: <i>Binary heaps & heapsort</i> B: <i>Exam practice</i>	Nov 28 lab: <i>Burkhard-Keller trees</i>	
07	Dec 03 A: <i>Formal languages and automata</i> B: <i>Formal languages and automata</i>	Dec 05 lab: <i>TBA</i>	
08	Dec 10 A: <i>Regular grammars and finite state automata</i> B: <i>Regular grammars and finite state automata</i>	Dec 12 lab: <i>TBA</i>	

What's Ahead

	Dec 17	Dec 19
09	A: <i>Finite-state transducers</i> B: <i>Finite-state transducers and computational morphology</i>	lab: TBA
Sem. break	<i>No class</i>	<i>No class</i>
10	Jan 7 A: <i>Context-free languages and constituency parsing</i> B: <i>Context-free languages and constituency parsing</i>	Jan 9 lab: TBA
11	Jan 14 A: <i>Dependency grammars and treebanks</i> B: <i>Dependency grammars and treebanks</i>	Jan 16 lab: TBA
12	Jan 21 A: <i>Dependency parsing</i> B: <i>Dependency parsing</i>	Jan 23 lab: TBA
13	Jan 28 A: <i>A gentle introduction to classification</i> B: <i>Transition-based dependency parsing</i>	Jan 30 lab: TBA
14	Feb 04 A: <i>Exam review / practice</i> B: <i>Exam review / practice</i>	Feb 06 lab: TBA
15	Feb 11 A: <i>Exam</i> B:	Feb 13 lab: TBA

Sorting

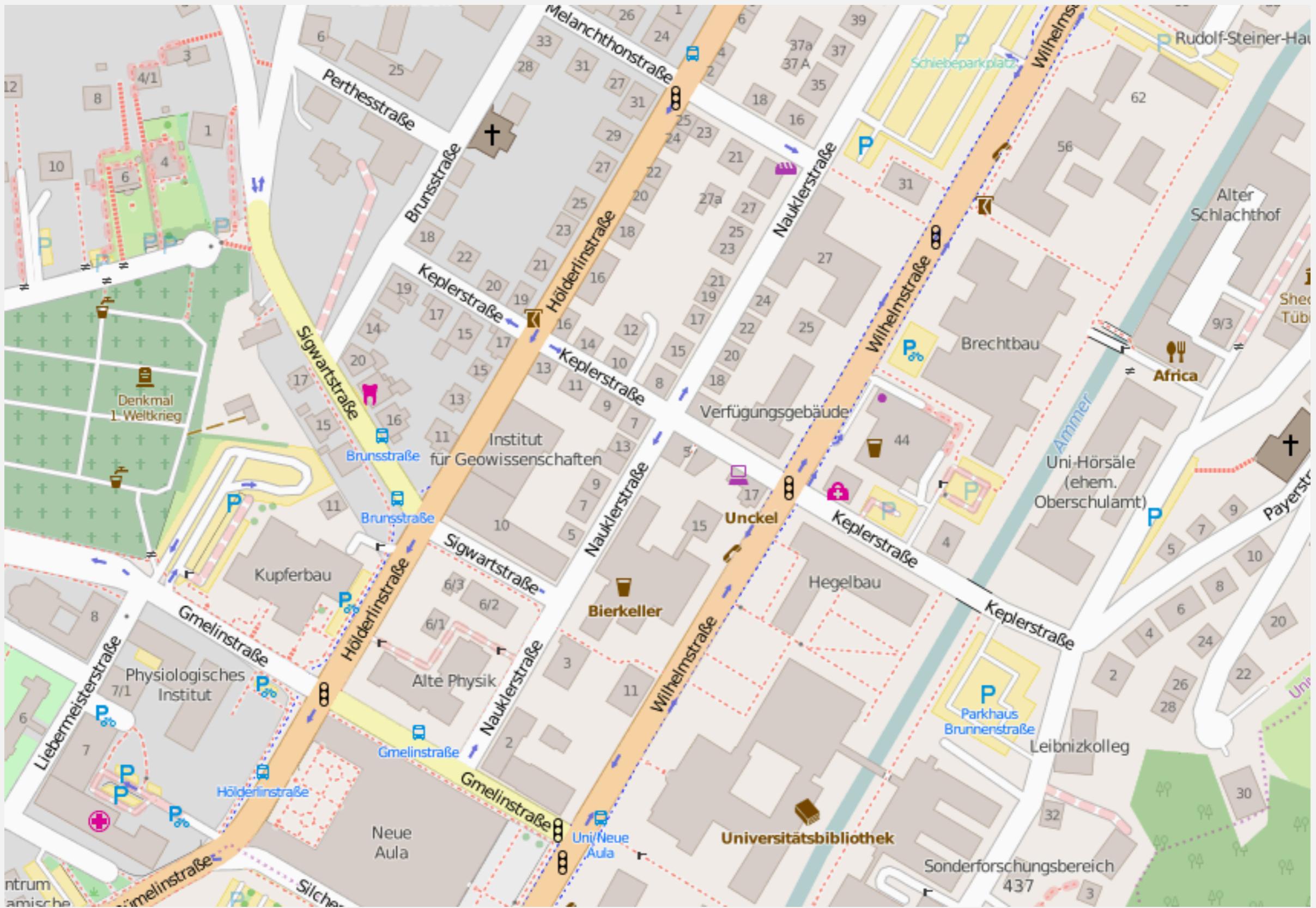


Bundesarchiv, Bild 183-22350-0001
Foto: Junge, Peter Heinz | 20. November 1953

Undirected Graphs



Directed Graphs



String Distance

The screenshot shows a search results page from a search engine. At the top left is a red logo consisting of two curved lines forming a stylized 'e'. To its right, the search term 'malorca' is displayed in a large, dark font. Below the search bar, there is a horizontal navigation menu with five items: 'All' (highlighted in blue), 'Images', 'Maps', 'News', and 'Shopp'. A thin blue horizontal bar is positioned below the menu line. The main content area displays the search results summary: 'About 114.000.000 results (0,64 seconds)'. Below this, a large bold text block says 'Showing results for **mallorca**' followed by 'Search instead for **malorca**'.

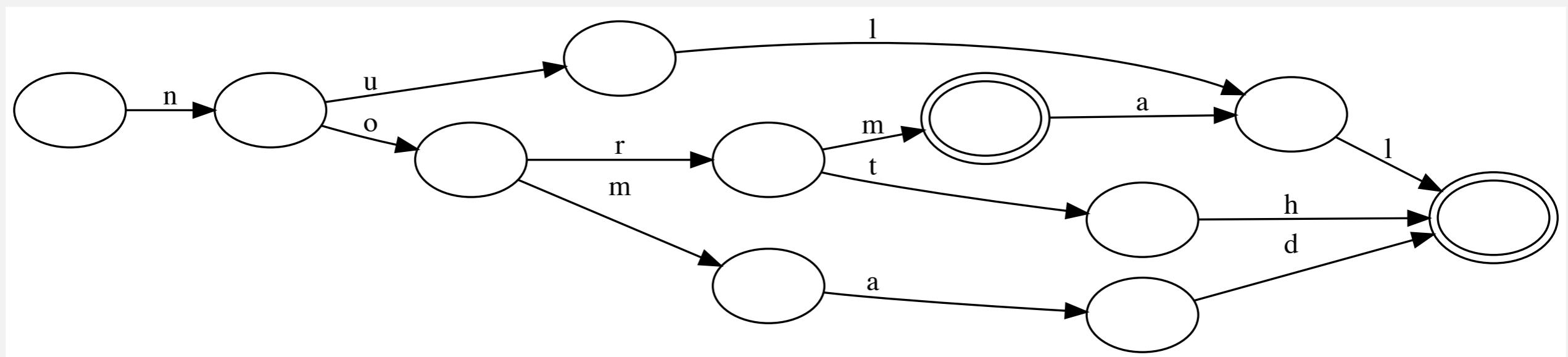
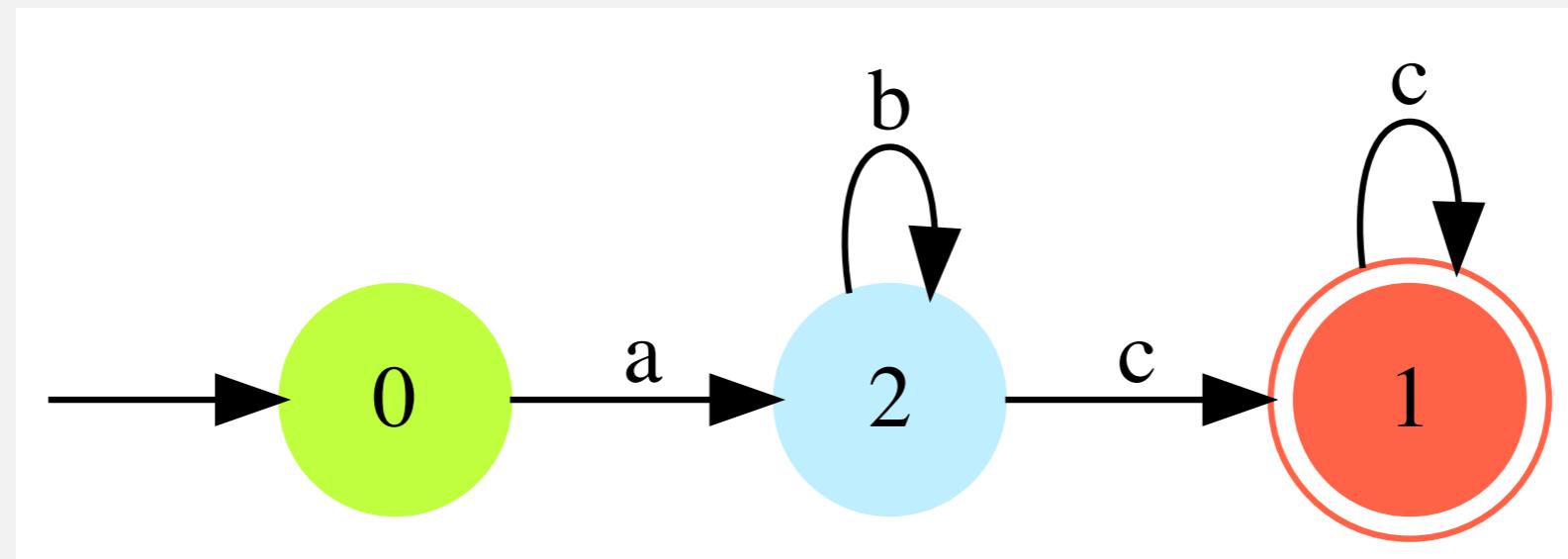
malorca

All Images Maps News Shopp

About 114.000.000 results (0,64 seconds)

Showing results for **mallorca**
Search instead for **malorca**

Finite State Automata



Parsing

← → ⌂ https://weblicht.sfs.uni-tuebingen.de/Tundra/?tcfurl=http://ws1-clarind.esc.rzg.mpg.de/drop-off/storage/result-1538660565119.tcf.xml&nomsg&tbsent=1

TCF-Dep | File ▾ Navigate ▾ Help ▾

Search

Enter Query Here. Use double quotes around strings or use TüNDRA query syntax.

Run Stats

Tree

Browse Treebank

#1: Die Besetzer werden mit Gewalt von Polizei und Militär vertrieben.

ROOT

AUX

PP

DET

SUBJ

PN

PP

PN

KON

CJ

-PUNCT-

Die

Besetzer

werden

mit

Gewalt

von

Polizei

und

Militär

vertrieben

.

PoS ART

PoS NN

PoS VAFIN

PoS APPR

PoS NN

PoS APPR

PoS NN

PoS KON

PoS NN

PoS VVPP

PoS \$.

Treebank Browser

Language Guessing

Language Guessing

Applications:

- Spamassassin uses the guessed language as a feature in spam identification.
- Web browsers language identification to offer you to translate a page when it is not in your native language.
- Google Translate uses language identification to determine the source language of a text to be translated.
- The CLARIN Language Resource Switchboard uses language identification (together with the identification of the resource's media type) to determine tools that can process the resource.

De **lambdacalculus**, soms ook als **λ -calculus** geschreven, is een formeel systeem dat in de wiskunde en [theoretische informatica](#) wordt gebruikt om het definiëren en uitvoeren van berekenbare functies te onderzoeken. Hij werd in [1936](#) door [Alonzo Church](#) en [Stephen Kleene](#) geïntroduceerd als onderdeel van hun onderzoek naar de [grondbeginselen](#) van de wiskunde, maar wordt tegenwoordig vooral gebruikt bij het onderzoeken van [berekenbaarheid](#). De lambdacalculus kan worden gezien als een soort minimale programmeertaal die [in staat](#) is elk [algoritme](#) te beschrijven. De lambdacalculus is [Turing-volledig](#) en vormt de basis van het paradigma voor [functionele programmeertalen](#).

De rest van dit artikel gaat over de oorspronkelijke, ongetypeerde lambdacalculus. De meeste toepassingen gebruiken varianten daarvan met een type-aanduiding.

Language II

Лáмбда-исчисление (λ -исчисление) — формальная система, разработанная американским математиком Алонзо Чёрчем, для формализации и анализа понятия вычислимости.

λ -исчисление может рассматриваться как семейство [прототипных языков программирования](#). Их основная особенность состоит в том, что они являются языками высших порядков. Тем самым обеспечивается систематический подход к исследованию операторов, аргументами которых могут быть другие операторы, а значением также может быть оператор. Языки в этом семействе являются функциональными, поскольку они основаны на представлении о [функции](#) или [операторе](#), включая функциональную аппликацию и функциональную абстракцию. λ -исчисление реализовано Джоном Маккарти в языке [Лисп](#). Вначале реализация идеи λ -исчисления была весьма громоздкой. Но по мере развития Лисп-технологии (прошёдшей этап аппаратной реализации в виде [Лисп-машины](#)) идеи получили ясную и четкую реализацию.

Language III

A **lambda-kalkulus** (vagy λ -kalkulus) egy **formális rendszer**, amit eredetileg matematikai **függvények** tulajdonságainak (definiálhatóság, rekurzió, egyenlőség) vizsgálatára vezettek be. Az elmélet kidolgozói **Alonzo Church** és **Stephen Cole Kleene** voltak az 1930-as években. Church, 1936-ban, a λ -kalkulus segítségével bizonyította, hogy nem létezik algoritmus a híres **Entscheidungsproblem** (döntési probléma) megoldására. A λ -kalkulus (akárcsak a **Turing-gép**) lehetővé teszi, hogy pontosan (formálisan) definiáljuk, mit is értünk **kiszámítható függvény** alatt.

A λ -kalkulust nyugodtan nevezhetjük a legegyszerűbb **általános célú programozási nyelvnek**. Csak egyfajta értéket ismer: a függvényt (absztrakciót), és csak egyfajta művelet van benne: a függvény alkalmazás (változó-behelyettesítés). Ezen látszólagos egyszerűsége ellenére minden algoritmus, ami Turing-gépen megvalósítható, az megvalósítható tisztán a λ -kalkulusban is. Ez az azonosság a λ -kalkulus és a Turing-gép **kifejező ereje** (expressive power) között adja egyébként a **Church–Turing-tézis** alapját.

Míg korábban a λ -kalkulus elsősorban a **kiszámíthatóságelmélet** (Theory of Computation) miatt volt érdekes, napjainkban ez már kevésbé hangsúlyos, és sokkal inkább a **funkcionális programozási nyelvek** elméleti és gyakorlati megalapozásában játszott jelentős, mondhatni központi szerepe került előtérbe.

λ 演算（英语：lambda calculus, λ -calculus）是一套用于研究函数定义、函数应用和递归的形式系统。它由阿隆佐·邱奇和他的学生斯蒂芬·科尔·克莱尼在20世纪30年代引入。邱奇运用 λ 演算在1936年给出判定性问题（Entscheidungsproblem）的一个否定的答案。这种演算可以用来清晰地定义什么是一个可计算函数。关于两个lambda演算表达式是否等价的命题无法通过一个“通用的算法”来解决，这是不可判定性能够证明的第一个问题，甚至还在停机问题之先。Lambda演算对函数式编程语言有巨大的影响，比如Lisp语言、ML语言和Haskell语言。

Lambda演算可以被称为最小的通用程序设计语言。它包括一条变换规则（变量替换）和一条函数定义方式，Lambda演算之通用在于，任何一个可计算函数都能用这种形式来表达和求值。因而，它是等价于图灵机的。尽管如此，Lambda演算强调的是变换规则的运用，而非实现它们的具体机器。可以认为这是一种更接近软件而非硬件的方式。

本文讨论的是邱奇的“无类型lambda演算”，此后，已经研究出来了一些有类型lambda演算。

関数を表現する式に文字ラムダ (λ) を使うという慣習からその名がある。アロンゾ・チャーチとスティーヴン・コール・クリーネによって1930年代に考案された。1936年にチャーチはラムダ計算を用いて一階述語論理の決定可能性問題を（否定的に）解いた。ラムダ計算は「計算可能な関数」とはなにかを定義するために用いられることがある。計算の意味論や型理論など、計算機科学のいろいろなところで使われており、特にLISP、ML、Haskellといった関数型プログラミング言語の理論的基盤として、その誕生に大きな役割を果たした。

ラムダ計算は1つの変換規則（変数置換）と1つの関数定義規則のみを持つ、最小の（ユニバーサルな）プログラミング言語であるということもできる。ここでいう「ユニバーサルな」とは、全ての計算可能な関数が表現でき正しく評価されるという意味である。これは、ラムダ計算がチューリングマシンと等価な数理モデルであることを意味している。チューリングマシンがハードウェア的なモデル化であるのに対し、ラムダ計算はよりソフトウェア的なアプローチをとっている。

この記事ではチャーチが提唱した元来のいわゆる「型無しラムダ計算」について述べている。その後これを元にして「型付きラムダ計算」という体系も提唱されている。

Any Ideas

Language Guessing

- Any ideas
- (Brainstorming)

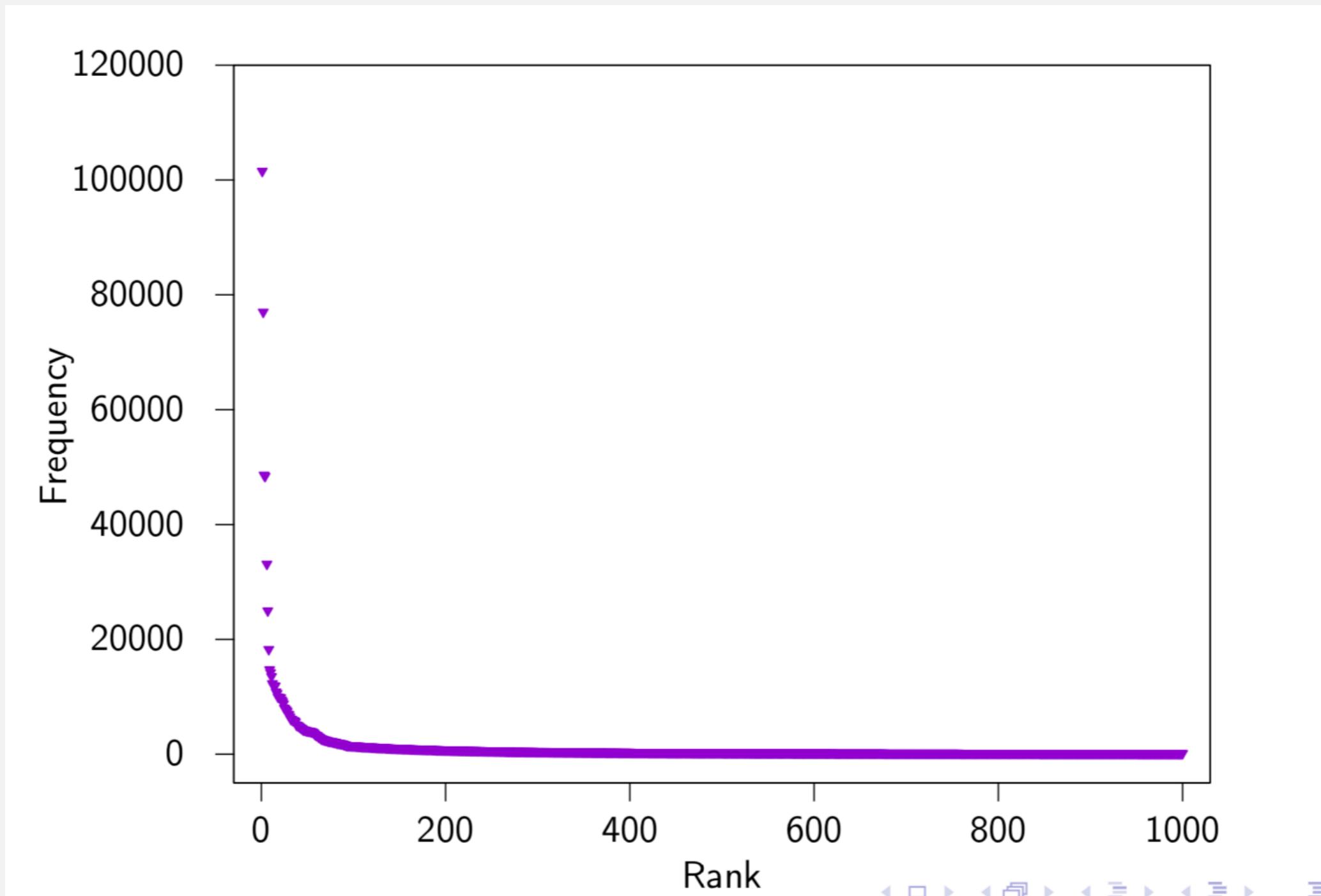
Method

- We can make a computer guess the language:
 - Using simple **n-gram** statistics
 - Using a small amount of training data
 - With high accuracy
- Here we will discuss the method of Cavnar and Trenkle, 1994
- We can usually identify a language using only a very short fragment.
E.g.:
 - German: *plötzlichen Ausbruch des Vulkans Ontake in Japan*
 - English: *cross-country navigational exercise and made a banking*
 - Spanish: *provenientes del idioma japonés que describen una*
- Some examples of n-grams that frequently occur these languages:
 - German: *ung, chen, der, die, ö*
 - English: *th, y_, ed_, wh*
 - Spanish: *la, que, ió, los_*

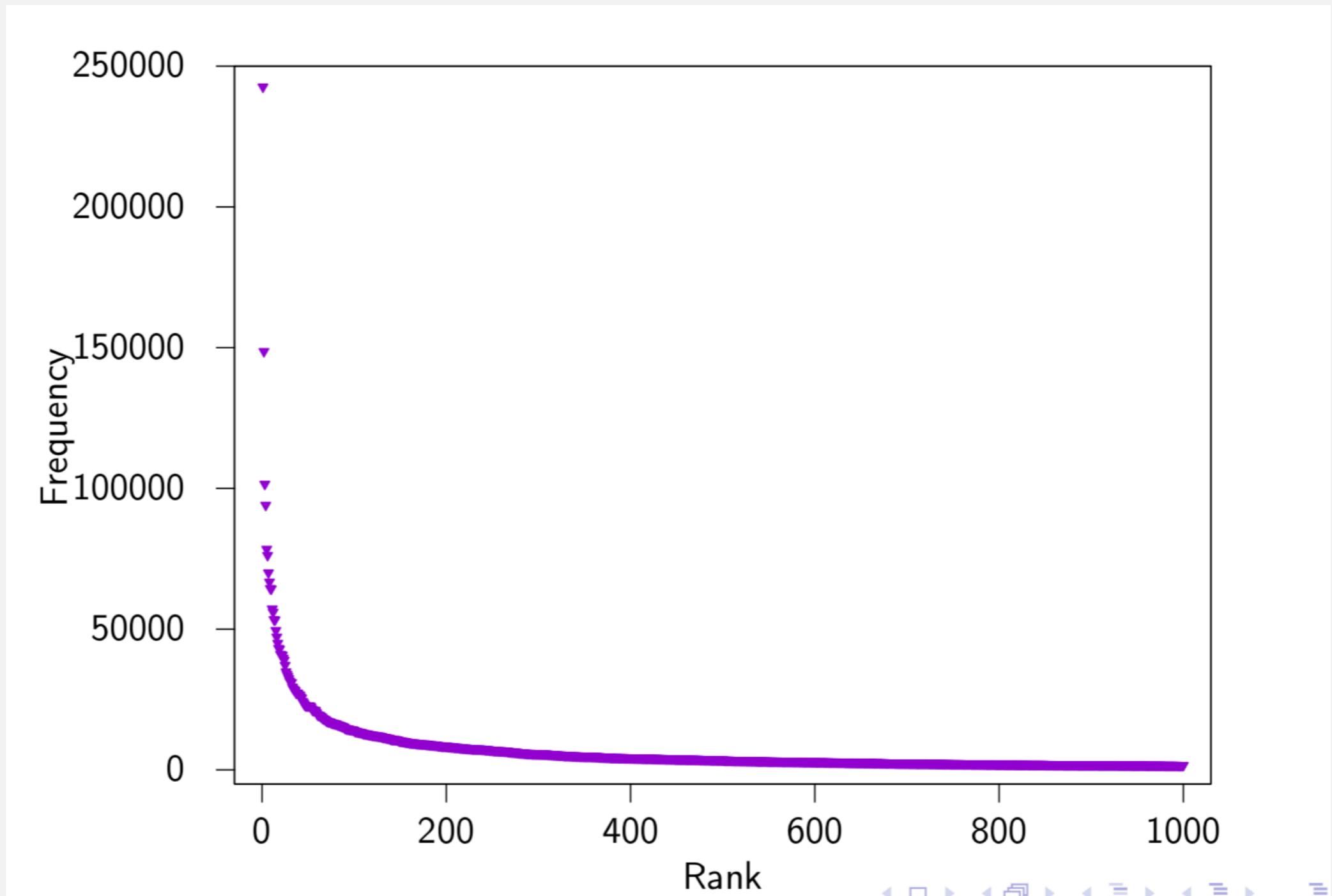
How much information is needed

- If we were to build a model of a couple of languages, how much information do we need per language to classify most texts correctly?
- To find an answer to this question, we look at Zipf's law:
 - *The frequency of a word is inversely proportional to its frequency-based rank*
- That is,
 - the most frequent word will occur approximately twice as many times as the second most frequent word,
 - thrice as many times as the third most frequent word *etc.*

Distributions of tokens in TüBa-D/Z



Distributions of character trigrams in TüBa-D/Z



Lessons Learned

- A small number of n-grams pop up 'all over the place';
- consequently, only a small number of n-grams are effective indicators;
- documents from a language should have similar n-gram frequency distributions.
- Cavnar and Trenkle create a profile of a language using a small amount of text in the following manner:
 - Count each 1..5-gram in the text
 - Sort the n-grams by frequency (most frequent first)
 - Retain the 300 most frequent n-grams
- Note: Cavnar and Trenkle discard all characters that are not letters or quotes.

Example: bananas

bi-grams: _T, TE, EX, XT, T_

tri-grams: _TE, TEX, EXT, XT_, T__

quad-grams: _TEX, TEXT, EXT_, XT__, T___

In general, a string of length k , padded with blanks, will have $k+1$ bi-grams, $k+1$ tri-grams, $k+1$ quad-grams, and so on.

n-gram	freq	n-gram	freq
a	3	ba	1
an	2	ban	1
ana	2	bana	1
n	2	banan	1
na	2	nan	1
anan	1	nana	1
anana	1	nanas	1
anas	1	nas	1
as	1	s	1
b	1		

Example: bananas

n-gram	rank	n-gram	rank
a	1	ba	11
an	2	ban	12
ana	3	bana	13
n	4	banan	14
na	5	nan	15
anan	6	nana	16
anana	7	nanas	17
anas	8	nas	18
as	9	s	19
b	10		

Language identification

- Generate a profile for each language, based on a longer text. When classifying the language of a document:
 - Create a profile of the document.
 - Compare the profile of the document with the profile of each language.
 - Choose the language with the most similar profile.
- How do we compare two profiles?

Given the document profile p and language profile q , the distance function is defined as:

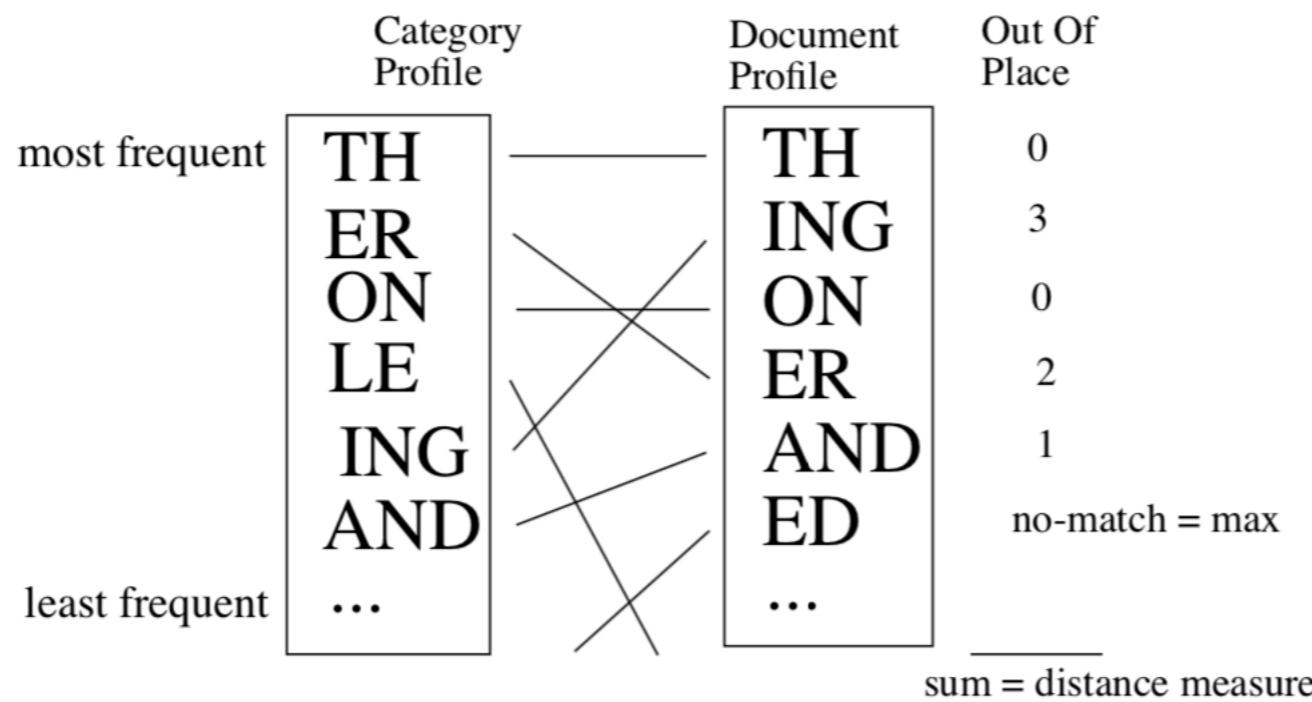
$$d(p, q) = \sum_{\text{ngram} \in p} |rank(\text{ngram}, p) - rank(\text{ngram}, q)|$$

The *rank* function gives the rank of an n-gram in the profile, or the size of the profile if it does not have the n-gram.

Example & Algorithm

p	q	$ \Delta $
th	th	0
ing	er	3
on	on	0
er	le	2
and	ing	1
ed	and	max
...

FIGURE 3. Calculating The Out-Of-Place Measure Between Two Profiles



```
 $L \leftarrow \{\langle \text{language}, \text{profile} \rangle\}$ 
 $p \leftarrow \text{document profile}$ 
 $\text{guess} \leftarrow \epsilon$ 
 $\text{guess\_dist} \leftarrow \infty$ 
for all  $(l, q) \in L$  do
     $\text{dist} \leftarrow d(p, q)$ 
    if  $\text{dist} < \text{guess\_dist}$  then
         $\text{guess\_dist} \leftarrow \text{dist}$ 
         $\text{guess} \leftarrow l$ 
    end if
end for
```

Method Evaluation

Article Length	≤ 300	≤ 300	≤ 300	≤ 300	> 300	> 300	> 300	> 300
Profile Length	100	200	300	400	100	200	300	400
Newsgroup								
australia	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
brazil	70.0	80.0	90.0	90.0	91.3	91.3	95.6	95.7
britain	96.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
canada	100.0	100.0	100.0	100.0	100.0	*99.6	100.0	100.0
celtic	100.0	100.0	100.0	100.0	99.7	100.0	100.0	100.0
france	90.0	95.0	100.0	*95.0	99.6	99.6	*99.2	99.6
germany	100.0	100.0	100.0	100.0	98.9	100.0	100.0	100.0
italy	88.2	100.0	100.0	100.0	91.6	99.3	99.6	100.0
latinamerica	91.3	95.7	*91.3	95.7	97.5	100.0	*99.5	*99.0
mexico	90.6	100.0	100.0	100.0	94.8	99.1	100.0	*99.5
netherlands	92.3	96.2	96.2	96.2	96.2	99.0	100.0	100.0
poland	93.3	93.3	100.0	100.0	100.0	100.0	100.0	100.0
portugal	100.0	100.0	100.0	100.0	86.8	97.6	100.0	100.0
span	81.5	96.3	100.0	100.0	90.7	98.9	98.9	99.45
Overall	92.9	97.6	98.6	98.3	97.2	99.5	99.8	99.8

Note: Asterisks indicate combinations of test variables that did worse than similar combinations using shorter profiles.

Complications

The classification problem can be made more complicated by:

- Adding more languages
 - Adding languages that are very similar
 - Adding dialects
 - Identification of very short fragments
 - Documents with multiple languages
-
- Apache OpenNLP includes char n-gram based statistical detector and comes with a model that can distinguish 103 languages
 - Apache Tika contains a language detector for 18 languages
-
- There are newer methods that use more sophisticated statistical modeling and/or machine learning to identify languages.

Maven

Maven

Apache Maven is a tool for building and managing Java projects.

Advantages of Maven are:

- Declarative: you do not have to specify the steps to build a project.
- Dependency management: you specify the dependencies of your project and Maven will automatically download them and make them available in the classpath.
- IDE agnostic: all major IDEs (including IntelliJ, NetBeans, and Eclipse) have plugins for Maven, meaning that you can open a Maven project in any IDE.
- Plugins: the functionality of Maven can easily be extended using plugins.

Maven Project Layout

- **pom.xml** Maven project description
- **src/main** Main project sources
 - **java** Main Java sources
 - **resources** Resources
- **src/test** Test sources
 - **java** Java sources for tests
 - **resources** Resources for tests

Dependencies:

- Many Java libraries are in the Maven Central Repository
 - search.maven.org
- Usually, you will find a fragment on the website of a project.

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>18.0</version>
</dependency>
```

Basic Maven Commands

- # Clean up a project (remove compiled Java code)
 - **mvn clean**
- # Compile a project
 - **mvn compile**
- # Run unit tests
 - **mvn test**