EECS 498 System Design of a Search Engine

Winter 2026

Lecture 1: Introduction

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving



Nicole Hamilton

nham@umich.edu
https://web.eecs.umich.edu/~nham/
Office: Beyster 4820
C: 425-765-9574

Office hours:
TTh 3:00 pm to 4:00 pm

Education

BS & MS EE, Stanford, 1973.

MBA, Boston University, 1987.

Background

Started my career doing hardware design at IBM but quickly moved into software.

Spent most of my career as an entrepreneur selling a C shell I wrote.

When the dot-com collapse hit, I went to Microsoft, where I worked on the first release of Bing.

Thought I was retired when I fell into a new career teaching.

Hamilton C shell - Wikipe

Nicole

Secure | https://en.wikipedia.org/wiki/Hamilton_C_shell

Msnicki Talk Sandbox Preferences Beta Watchlist Contributions Log out

Article Talk Read Edit View history More TW Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item

Hamilton C shell

From Wikipedia, the free encyclopedia

Hamilton C shell is a clone of the Unix C shell and utilities[1][2] for Microsoft Windows created by Nicole Hamilton[3] at Hamilton Laboratories as a completely original work, not based on any prior code. It was first released on OS/2 on December 12, 1988[4][5][6][7][8][9] and on Windows NT in July 1992.[10][11][12] The OS/2 version was discontinued in 2003 but the Windows version continues to be actively supported.

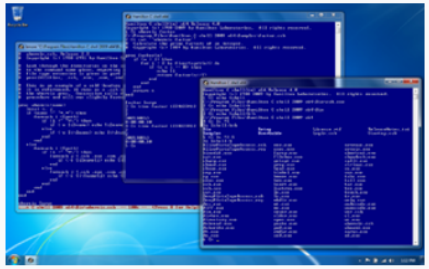
Contents [hide]

1 Design
  1.1 Parser
  1.2 Threads
  1.3 Windows conventions
2 References
3 External links

Design [edit]

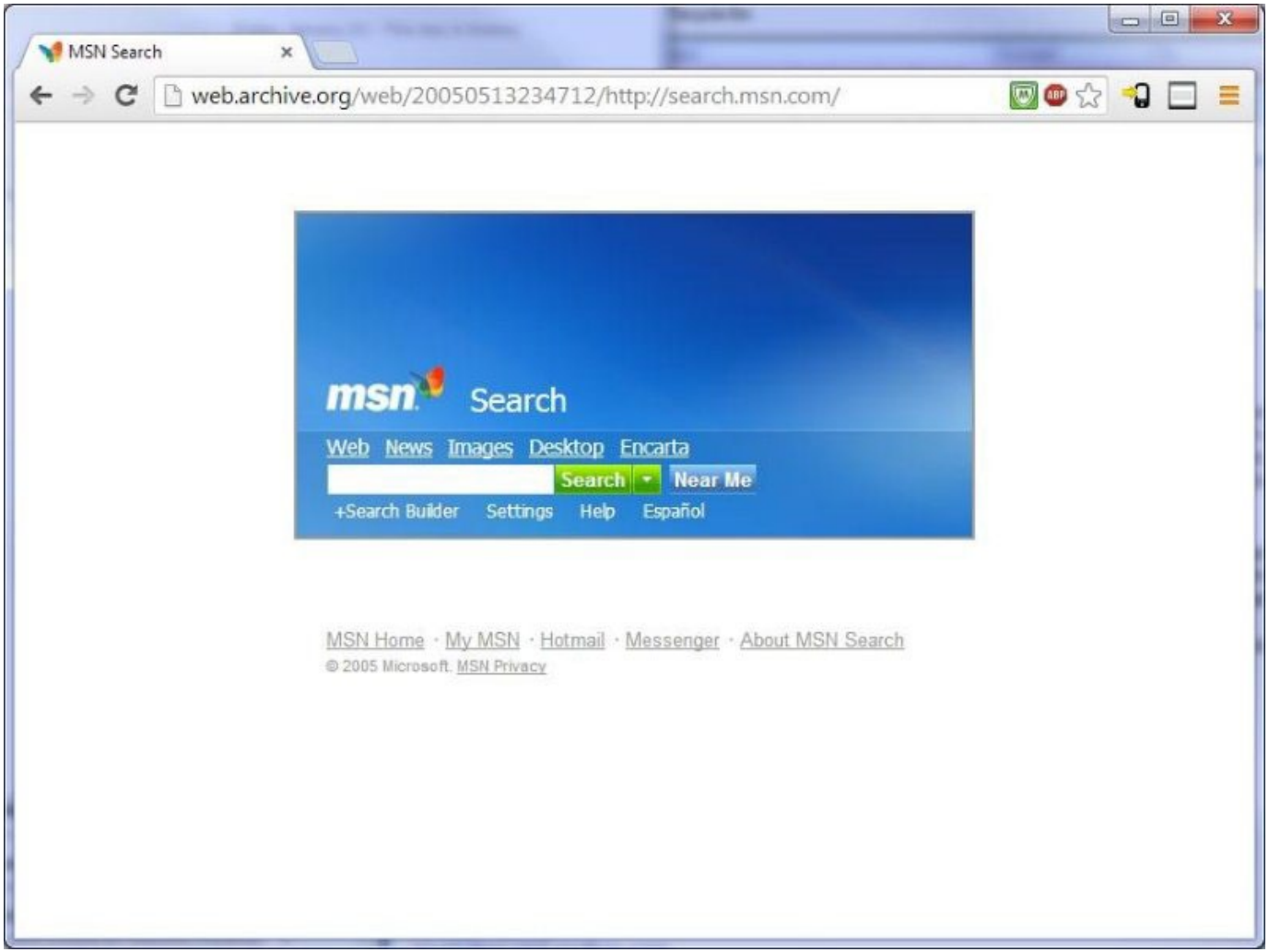
Hamilton C shell differs from the Unix C shell in several respects, its compiler architecture, its use of threads, and the decision to follow

Hamilton C shell



64-bit Hamilton C shell on a Windows 7 desktop.

Original author(s)	Nicole Hamilton
Initial release	December 12, 1988; 28 years ago
Stable release	5.2.g / March 5, 2017; 5 months ago
Written in	C
Operating system	Windows
Type	Unix Shell on Windows
License	Commercial proprietary



MSN Search in early 2005.



Joined the team in July 2003 as the ninth member.

The ranker was the last major piece no one had taken.

Wrote the ranker and the query language for the first release in January 2005.

Almost 30 KLOC in C++.

This is the sixth time I’m teaching this class.

I'm still working to improve it.

staff



Christopher Davis
chcdavis@umich.edu



Rickey Dong
rickeyd@umich.edu

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

Course organization

1. You don’t need to know anything more than 281.
2. I will teach you everything else you need to know.
3. First half will focus on operating system topics needed to build a search engine.
4. Second half will focus building the engine.
5. You’ll have homework every week to help you learn the material. You’ll do most of it in groups.
6. I’m more interested in your completing an assignments than the deadline. If you need an extension, talk to me.

This semester

1. The schedule and almost all the assignments for the entire semester are posted on Canvas.
2. Most of the homework assignments are coding exercises you'll submit to the autograder.
3. On group assignments, you are free to break up the work any way you like.
4. Your first assignments are this weekend.

EECS 498 W26 System Design of a Search Engine Schedule

Week	Sunday	Tuesday Lecture	Thursday Lecture	Lab	Assignment (due next Sunday 11:59 pm)
1	1/4		Introduction to the course	Introduction Speed dating Group activity	Introduce Yourself 01 Most Positive Subsequence
2	1/11	Search engine basics	Project planning Advice on the project	Planning the project What's most important Speed dating	Gantt chart for graduate school Join a team
3	1/18	HTML Utf8 HTTP/HTTPS Redirects	Intro to the filesystem Handles stdin, stdout Opening files Seeking	Group activity HTML Utf8 filesystem	Group photo 02 Vector and string 03 Utf8
4	1/25	Mapped files stat'ing a file Directories	Hashing Relocatable hash tables Memory-mapped hash tables	Project plans Memory-mapped wc Hash tables	Project plan 04 HTML Parser 05 Hashtable and Top 10 Rate the group photos
5	2/1	Processes and threads	Locks Producer-consumer relationships RAII	Team meetings with instructor	06 HashBlob and HashFile 07 Multithreaded file operations
6	2/8	TCP/IP, DNS, sockets	Secure Socket Layer	Sockets SSL Reading a web page	08 LinuxGetUrl
7	2/15	LinuxTinyServer Simple web form	The index and constraint solver		09 LinuxGetSsl 10 LinuxTinyServer
8	2/22	The constraint solver	**Midterm**	No lab	No assignment
9	3/1	Spring break			
10	3/8	The frontier Distributing your engine	robots.txt Bloom filter		No assignment
11	3/15	Parsing and compiling	Parsing and compiling		Tee shirt design 11 Bloom filter 12 Robots.txt
12	3/22	The query compiler	Ranking		Rate the tee shirts 13 Expression parser
13	3/29	Shingles	Other system design problems		
14	4/5	The digital world we live in	Ethics		Presentation slides
15	4/12	Team presentations	Team presentations		Final report Rate the team presentations Rate your teammates
16	4/19	Last lecture course debrief	Final exam (optional)	Team meetings with instructor Project demo	Project archive
17	4/26	Team meetings with instructor Project demo			**Final exam 4/29 4:00 pm**

Grading

Group project	
Group performance	25%
Individual contribution	25%
Homework	
Group	10%
Individual	10%
Midterm	15%
Optional final	15%

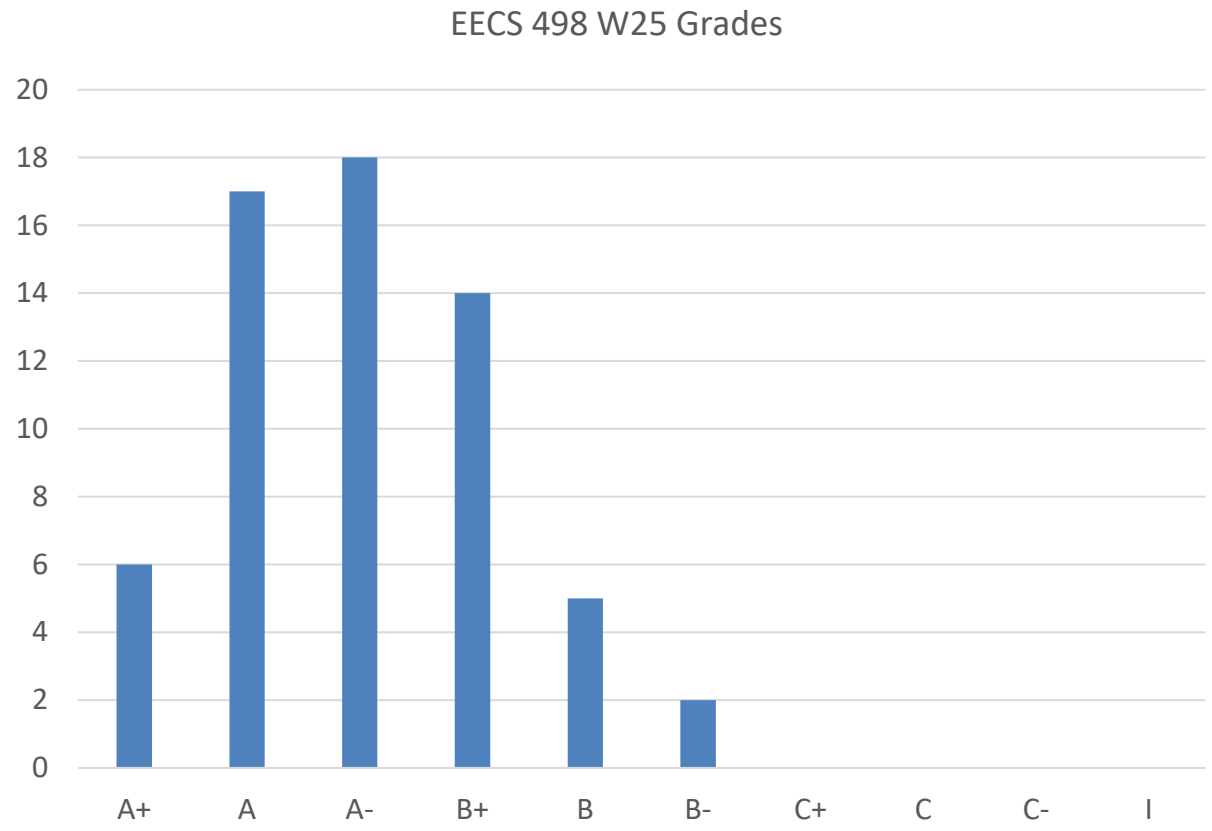
The midterm is mandatory.
If you skip the final, I'll use your midterm score.

Some of the grading will be done by surveys.

Past curves

Here was the grade distribution last time.

Having realized you have easier alternatives, I've eased my grading.



Your team grade on the project

Slightly competitive, based on ranking your engine's functionality, *performance*, e.g., time and size to *crawl, create or search* a *multi-terabyte index*, its *features* and *result quality* against the engines created by the *other teams*.

Your individual grade on the project

Your individual grade on the project will be based on:

1. The number of lines of code you contribute relative to others on your team and in the class.
2. A survey of your teammates asking them to rate your contribution and performance.

Your first homework

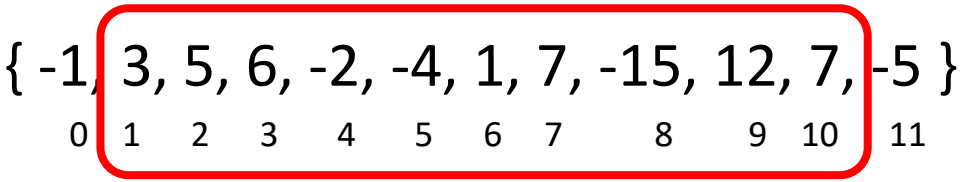
This weekend:

1. Write a short introduction of yourself with a recent picture.
2. A coding problem, MostPositivedSubsequence( ). You'll submit your code to the autograder and upload a screenshot of your debugger.
3. This is the only coding homework you'll have to do alone.

HW1 MostPositiveSubsequence( )

Write a C++ function that can scan a sequence of N integers in an array A, returning the sum and the left and right indices of the most positive subsequence.

For example, for the sequence



the best sum = 20, left = 1, right = 10.

HW1 MostPositiveSubsequence( )

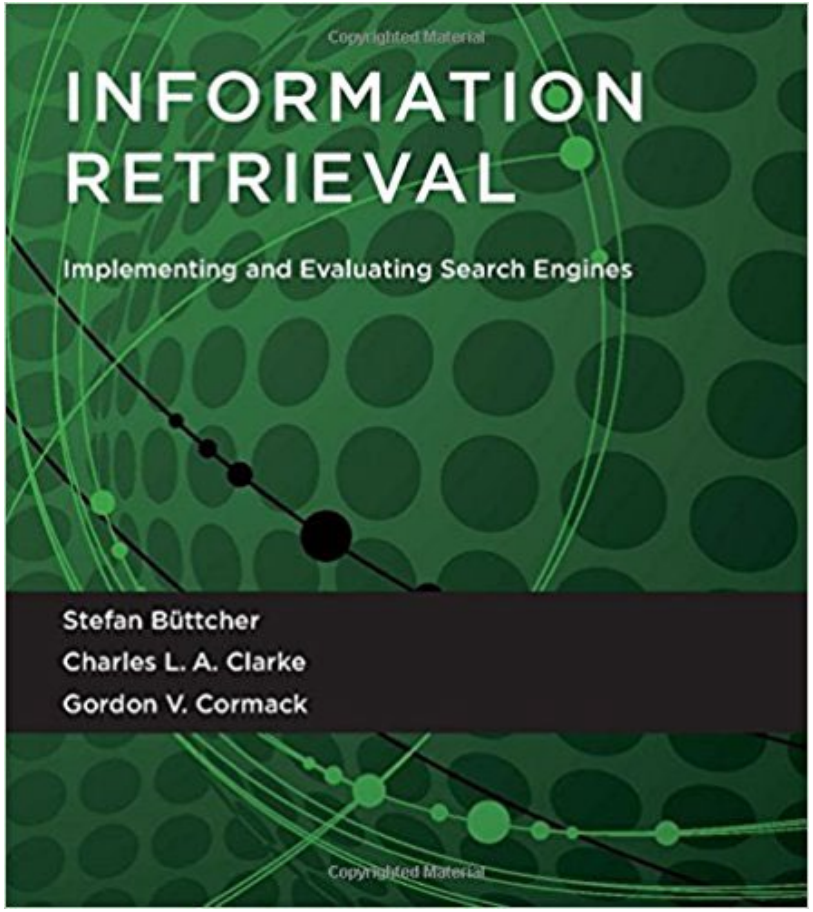
1. Due Sunday, January 12, 2025.
2. Write and submit two C++ files to the autograder.
    a. MostPositiveSubsequence.cpp, containing your implementation of the MostPositiveSubSequence( ) function.
    b. BestSubsequence.cpp, containing a simple main( ) routine that takes a sequence on the command line an argv and reports the results.
3. You must also demonstrate that you can build and debug it *with a graphical debugger* by uploading a screenshot showing you stepping through your code.

General rules for homework

Unless otherwise specified:

1. You must solve it on your own or as a registered group.
2. Do not use AI to do your work for you.
3. You are to make good choices anywhere you find the spec ambiguous and you will be judged on them.
4. You may not discuss it with others or use the internet or other resources to help.

Textbooks



Information Retrieval: Implementing and Evaluating Search Engines
Reprint edition (February 12, 2016)
Stefan Büttcher, Charles L.A. Clarke, Gordon V. Cormack
The MIT Press
ISBN 978-0262528870

There's a PDF posted on Canvas.
Read the first 6 chapters.

All the work must be your own

1. Submitting work that's not your own or attempting to deceive me will be reported for academic misconduct.
2. I'm good at spotting misconduct and very good at reporting it.
3. I do not give warnings. I report everything.

Where you'll find stuff

Canvas for announcements, lecture recordings, files, individual and group homework assignments and grades.

The autograder to test code submissions.

EdStem for questions and discussion.

agenda

1. Course staffCourse organization
2. Learning objectives
3. System design
4. The project
5. Basic parts of a search engine
6. Planning your engine
7. Teamwork
8. The world around us
9. Beautiful software
10. Problem solving

Learning objectives

1. Ability to work on a team to design a large software project.
2. Learn how a search engine works.
3. Learn how to use low-level OS facilities.
4. Ability to create elegant, reliable code in C++.
5. How to distribute a system across multiple machines.
6. An appreciation of software as art.
7. The ability to communicate your results.
8. Exposure to other topics related to the practice of engineering.

Interwoven themes

1. What is system design?
2. What is a beautiful design?
3. How do you tackle a large project?
4. How do you work as a team?
5. How does a search engine work?
6. How do you use the operating system?

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

systems

What makes something a system?

What's special about designing systems?

system design projects

1. There’s an important domain-specific part: You need to learn something new about an interesting problem you’ve never seen before.
2. You need to invent a solution with lots of moving parts.
3. They’re usually “close to the metal”.
4. They're usually team efforts because they're too big to do any other way.
5. You get to build your part from scratch and it feels good.

A search engine hits on every bit of what a system design project is to me.
I also intend for it to relatable to family and friends and recruiters.
I want you to get jobs.

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

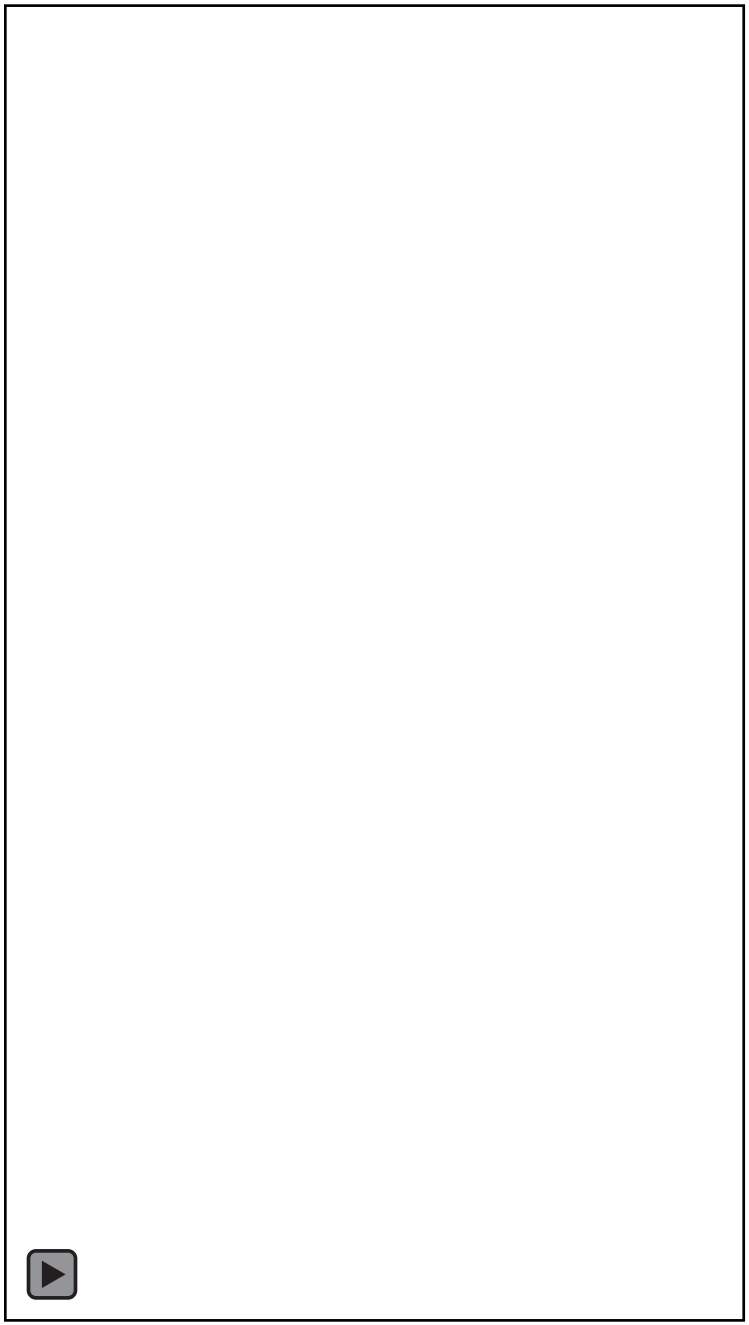
the project

You are to self-select into teams of 6 to design and build an end-to-end search engine completely from scratch in C++, assigning your own roles.

It’s supposed to be fun.

The Seekers captured the moment in W19 when their engine finally worked.

It was 5:00 am.



past engines in LOC

	Project total					Individual				
	w18	w19	f19	w21	w25	w18	w19	f19	w21	w25
High	14,271	26,887	19,264	20,523	48,027	4,096	8,646	7,826	8,127	13,506
Median	5,300	13,000	11,973	11,647	16,581	1,006	1,750	1,572	1,800	1,843
Low	4,170	9,414	4,575	5,887	6,132	0	444	120	200	400

past index sizes in documents

	w18	w19	f19	w21	w25
High	13.4M	150M	586M	536M	100M
Median	8,000	4.65M	397M	84M	45M
Low	4,816	1.9M	207M	3M	10M

agenda

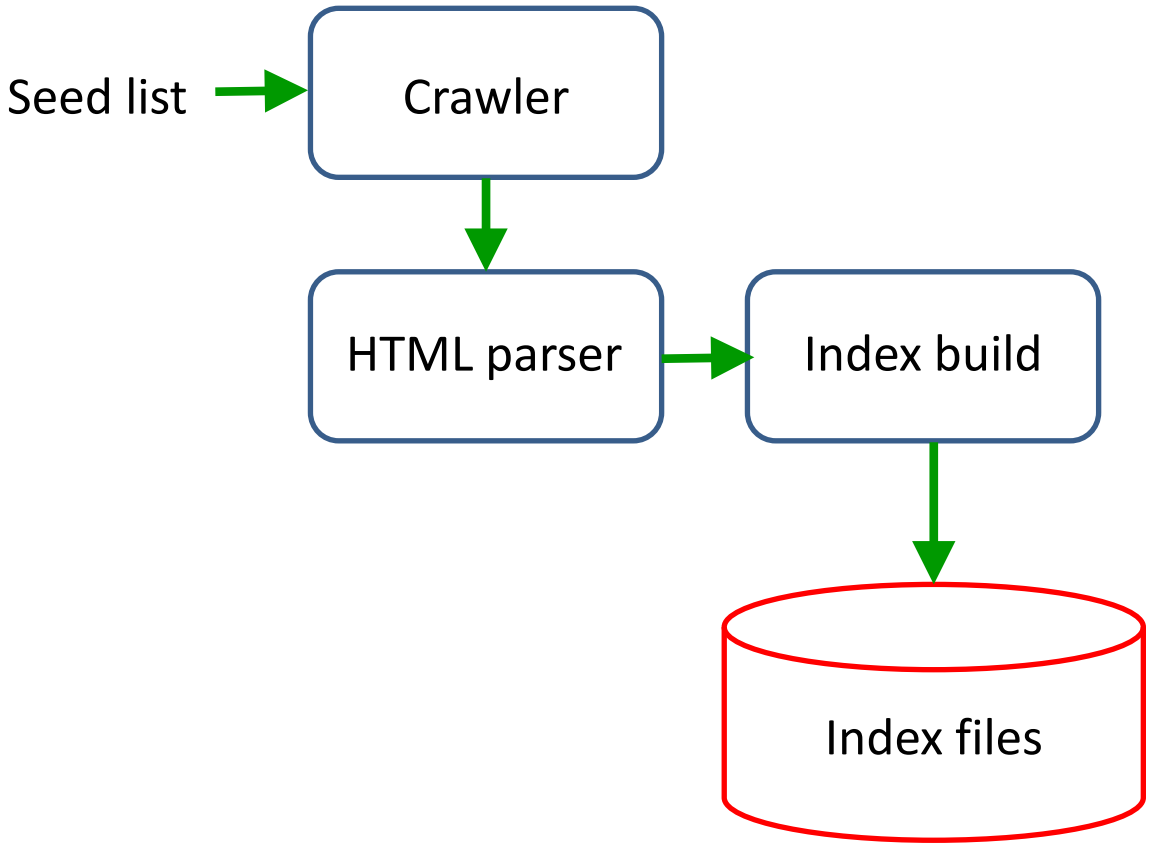
1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

basic parts of a search engine

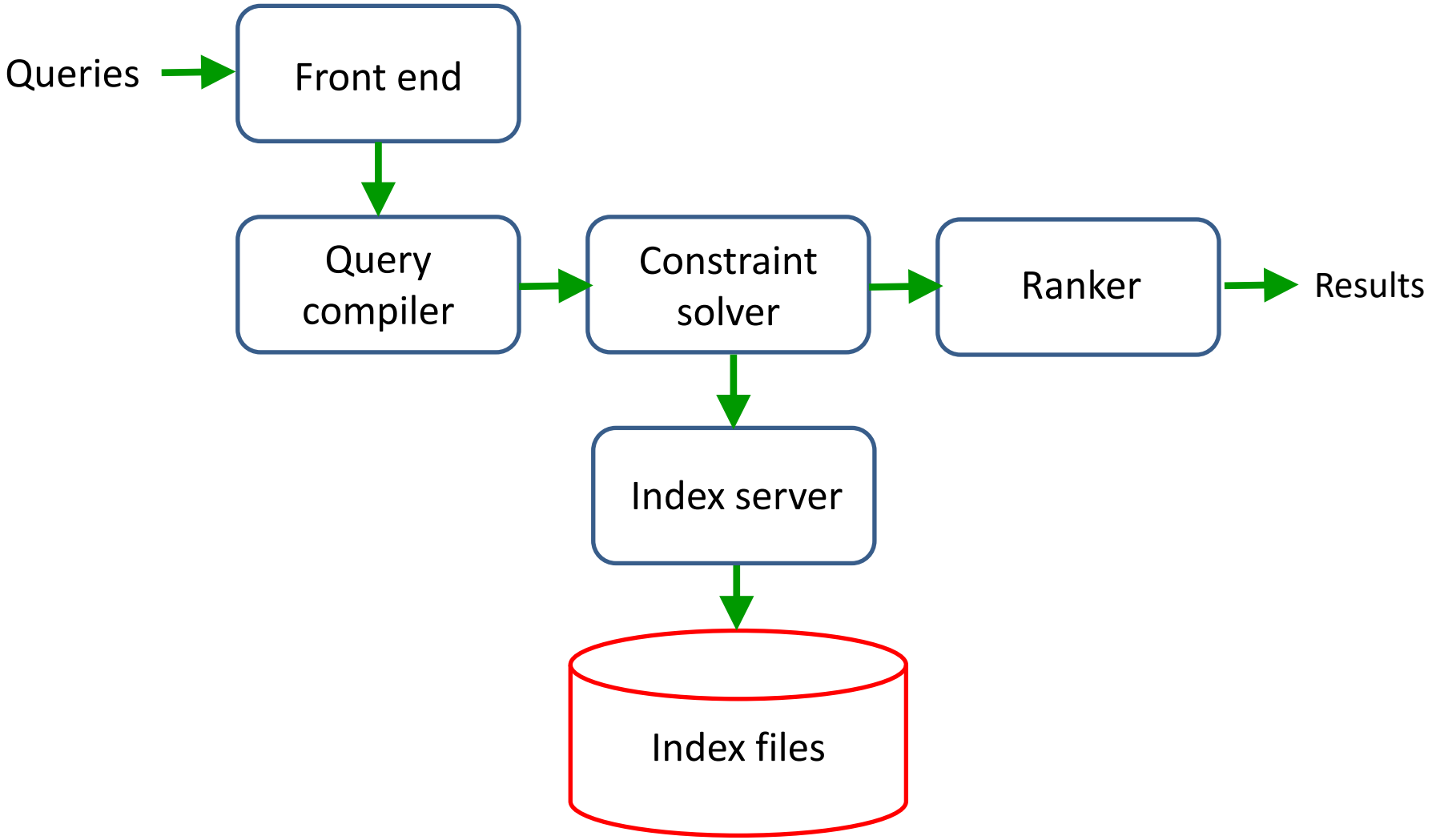
These are the basic pieces you will need.

1. HTML parser.
2. Crawler.
3. Index.
4. Constraint solver.
5. Query compiler.
6. Ranker.
7. Front end.

the index build side



the query serve side



agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

your search engine

1. You may deliver your engine on any platform but everyone always chooses Linux.
2. I will arrange $50 Google Cloud accounts for all of you.
3. Almost all of your work will be in C++ and all of it must be yours.
4. I've posted my stylesheet but no one ever follow it. (Your loss.)
5. I care less about test cases than I do about getting things working.
6. You are discouraged from using STL unless you write your own versions.
7. You will meet with me periodically as a team to discuss your plans and progress.
8. You will give a short presentation of your results to the class.
9. You will have a final review, demo, submit a paper and an archive of your code.

levels of functionality

0. Create a plan.
1. Parse text files into a hash table.
2. Build a crawler.
3. Build a reverse word index.
4. Create a user interface.
5. Build a constraint solver and query parser.
6. Build a ranker.
7. Advanced functionality, e.g., distributed processing.

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

What are the most important determinants of all your outcomes in life?

I suggest it’s all the other people in your life. Sometimes, you get to choose.

observations

On the best-performing teams:

1. They like each other.
2. They buy into the rules and set out to win at them.
3. Their plans are filled with a lot of brain-storming about the problems they need to solve and how they'll do it.
4. There's a lot of brain-storming in their execution as well.
5. They come to my office frequently.

observations

On teams having difficulties:

1. Their plans are often thin on detail, treating much of the design as TBD once we got to the appropriate lecture.
2. They check off some “all of the above” boxes, e.g., “all of level 7”, without much analysis of what that would entail.
3. There are questions about who is in charge. Decisions don’t get made. No one is available to meet at the same time.
4. They don’t do a lot of brainstorming and they don’t resolve technical questions with technical arguments.
5. Interfaces between the components and what each component does are unclear.
6. They complain about the rules.

observations

What makes for a great teammate:

1. They step up and get stuff done.
2. It's less about what they know and more about what they're willing to do.
3. They have lots of ideas but they don't insist on their own.
4. They respect boundaries.
5. They're available and supportive.

Your first group activity is to form a group and submit a group photo.

We'll do "speed dating" breakouts in the labs this week to help you get to know others in the class.

You are welcome to invite friends into the class to round out your group. Talk to me.

group photo

1. You are to form groups of 6, choose a name for your group and submit a photo of yourselves.
2. You must all appear in the photo. Faces have to be clearly visible and easily recognized.
3. Each person must be named in the photo.
4. Your group name must appear in the photo.
5. You may not spend any money on this.
6. You can edit the image but, please, no pasting your faces on other people's heads or bodies.
7. You may not steal copyrighted artwork.
8. You may use public domain images.

1. This assignment is slightly competitive based on creativity and execution.
2. Your grades will be determined by a vote.
3. Please submit only one copy for the entire team.

Here were some past submissions.


MAXIMAL MUNCH
BRANDON KAYES
RYAN WUNDERLY
DANIEL HOEKWATER
ALEX RAISTRICK
AUSTIN KIEKINTVELD
ADOLFO APOLLONI


Jack Bowman
"The Debugger"
Alex Jalkanen
"The Destructor"
Jason Kathawa
"The Compiler"
Alex Erf
"The Assembler"
Noah Tutt
"The Constructor"
Zhe Ye
"The Linker"
The Toolchain


WRITE BEAUTIFUL CODE →
WHILE (M
{
}
LET'S REDO STL!
MEAN GIRLS
MEAN BOIZ
(+MEL)
AM I IN THE RIGHT CLASS??
398
WAT??
C++ WITH GANTT CHART
Actual High School Mathlete
Uses one letter variable names
CODY MORTERUD
Also doesn't write comments
MELISSA GEORGE
CHAD SHARP
Uses nested tenary operators
plays hype music when submitting to AG
COLTEN WILLIAMS
Goes into a CS research not knowing struct
RAHUL AGRAWAL BEJARANO
GABRIEL LUO
Indents with 3 spaces non-ironically


KEYBOARD SURFERS
SHESANTH RAMAKRISHNAN
THOMAS BARTLETT
ERAY MITRANI
ENGIN AKDEMIR
CAROLYN BUSCH

C++LUE;



Anvi Arora
Mrs. Peacock
Ben Bergkamp
Wadsworth
Veronica Day
Miss Scarlet
Zane Dunnings
Professor Plum
Nicholas Yang
Mr. Green
Jake Close
Colonel Mustard


Internet Explorers
Pranav Tadikonda, Anthony Jonikas, Andrew Shea, Federico Kulyckyj, Julia Atayde, Chris Bahr


CodeKeepers
Divyansh Sharma
Anton Yang
Nick Bui
Tadeo Yelos
Will Wendorf
Paulo Warren

THE EXCEPTIONS




DEVESH MODI
CHRIS HOANG
SACH VAIDYA
OSAMA SAEED
SAMIUR KHAN
MINH NGUYEN
MIHIR BALA
THE SEEKERS
OUT OF 14,000,605 DOCUMENTS, ONLY 1 WAS RELEVANT


C·O·L·L·E·A·G·U·E·S
Ali
Maddy
Jacque
Will
Hannah
Jed


EECS 398
SYSTEM DESIGN
Please Please Compile * The C-Tles
POSIX
THE C-TLES
PLEASE
PLEASE COMPILE
with Love Me Do-While
and 12 other errors
Produced By: Graham Eger, Max Guthmann,
Bradley Huang, Dennis Li, Jason Setting, and Jacob Crouch


Injured in a browser crash?
Derek Lau
Barbara Zhong
Allison Easton
Jonson Jin
Michael Zhong
Vincent Nagel
We've literally never lost a case.
(I mean look at us)
really attractive lawyers
really attractive lawyers near me
why are lawyers so hot
Bing
lawyer plz
Aol.
totally legit real lawyers
Your search ends here.
Call: 1-800-GETBREAD
Office of Easton&Jin&Lau&Nagel&Zhong&Zhong


THE FELLOWSHIP OF THE RING BUFFER
DANIEL SAGE
CHRISTOPHER DAVIS
KY PARK
IHSAAN YASIN
ARNAV SHAH
MADHAV SHARMA
ANUBHAV AGARWAL


BING
TEA
冰茶
Kevin Li
Rickey Dong
Jimmy Dai
Menu
Drinks
- Bubble-Sorted Tea
- Merge-Sorted Mocha
- OOPlong Tea
- Green Tea MatChar
- Java Coffee
Copy and PASTEries
- Query Quiche
- Cookies
> Firefox-flavored
> Chrome-flavored
> Edge-flavored
Roee Tsimhoni
Matthew Wang
Danny Kim
Wonbin Jin

my usual experience

A team's performance on the simple task of submitting a creative photo of themselves is often predictive of how they'll do on everything else.

Please don't do something lame.

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

But none of these statements are true for your parents.

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

There was never a time before credit cards.

DirecTV begins direct digital satellite broadcasting June 17, 1994. (31 years)

HD becomes available in the US in 1998. (28 years)

FCC mandates that all TV in the US is HD on June 12, 2009. (16 years)

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

iPod released October 23, 2001. (24 years)

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

iPhone released June 29, 2007. (19 years)

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

Consumer digital cameras become available in the 1990s and overtake film in 2003. (23 years)

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

eBay goes live September 3, 1995. (30 years)

Amazon goes live July 16, 1995. (30 years)

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never has hiss.

Apple II goes on sale June 10, 1977. (48 years)

IBM PC released August 12, 1981. (44 years)

if you are 20 in 2026

TV has never had snow.

You could listen to any song you want anytime you want.

Phones have always had apps.

Cameras have never used film.

You can buy anything online.

Everyone has always had computers.

Music has never had hiss.

Compact discs introduced in Japan in November 1981. (44 years)

what made this possible

Positional numbers

Digital circuits

Microelectronics

Three key technologies made this possible.

what made this possible

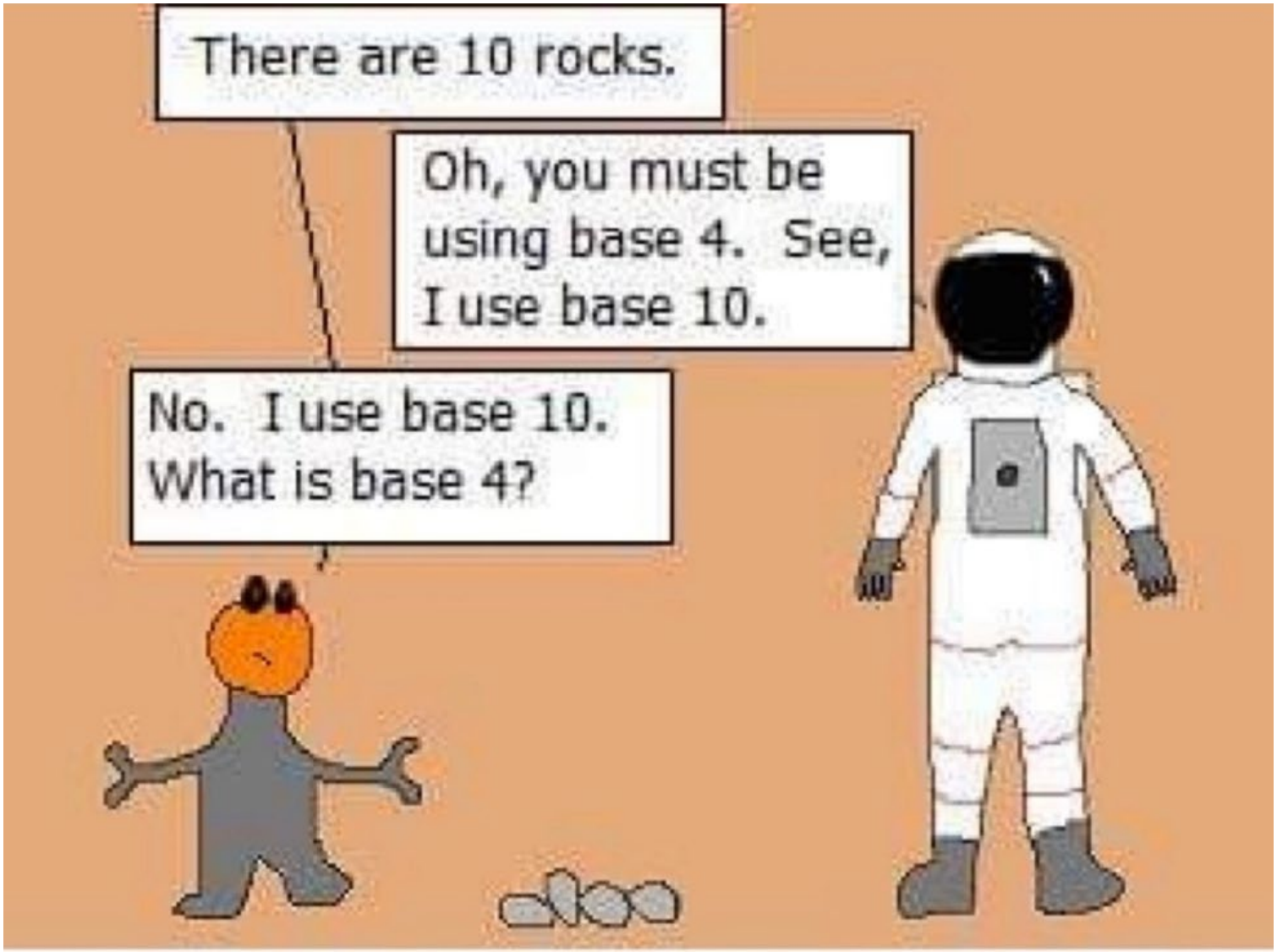
Positional numbers

Digital circuits

Microelectronics

Positional number systems, where each position is a power of the base, were invented by the Babylonians, making arithmetic simple.

(Consider the alternative of doing addition in Roman numerals.)



Every base is base 10.

what made this possible

Positional numbers

Digital circuits

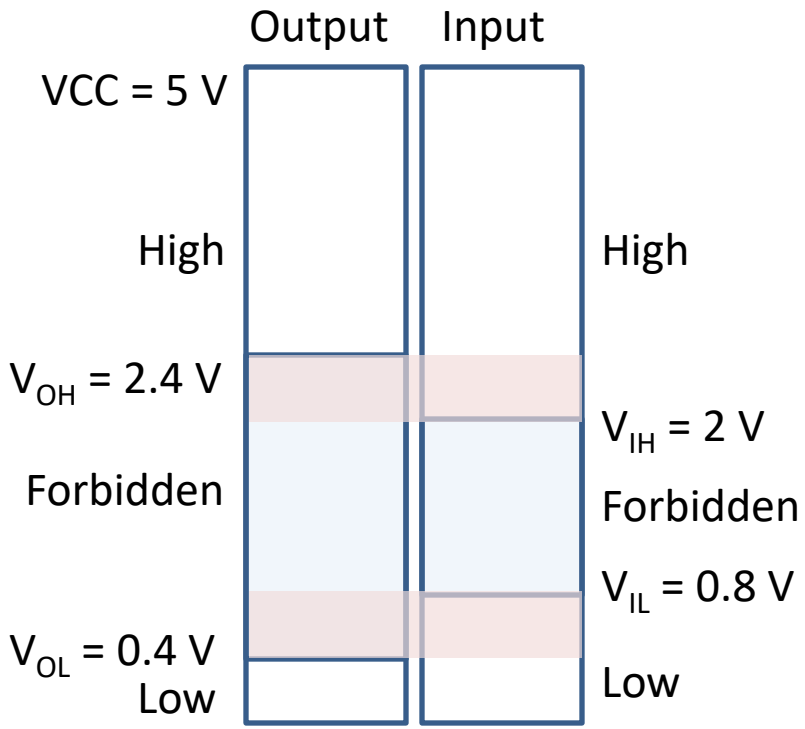
Microelectronics

Digital circuits built with working parts in a correctly designed circuit will always give the right output, even if there's some noise on the input.

the noise margin

Manufacturers guarantee that a working part will have a minimum output voltage of 2.4V for a 1 and a maximum output voltage of 0.4V for a 0.

The part is forbidden to have an output voltage in the forbidden range in a properly designed circuit except when transitioning between states.



TTL voltage levels

what made this possible

Positional numbers

Digital circuits

Microelectronics

Engineers have known how to build digital circuits since the 1940s, but they only had tubes to work with.

Early computers, like the ENIAC, built in 1945, required 18,000 vacuum tubes and consumed 150KW. A tube failed every 2 days.

what made this possible

Positional numbers

Digital circuits

Microelectronics

Microelectronics made error-free digital circuits practical.

If we'd been stuck with tubes, the digital transformation of the modern world would not have happened.

what made this possible

Positional numbers

Digital circuits

Microelectronics

We can now throw billions of digital circuits at a problem.

You can have as many as you like, they take almost no space, and you can power them with a small battery.

what made this possible

Positional numbers

Digital circuits

Microelectronics

Transistor invented 1947 by John Bardeen, Walter Brattain and William Shockley at Bell Labs.

what made this possible

Positional numbers

Digital circuits

Microelectronics

Integrated circuit invented by Jack Kirby at Texas Instruments on September 12, 1958.

what made this possible

Positional numbers

Digital circuits

Microelectronics

Moore's Law, 1965: The number of transistors in an integrated circuit doubles about every two years

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving



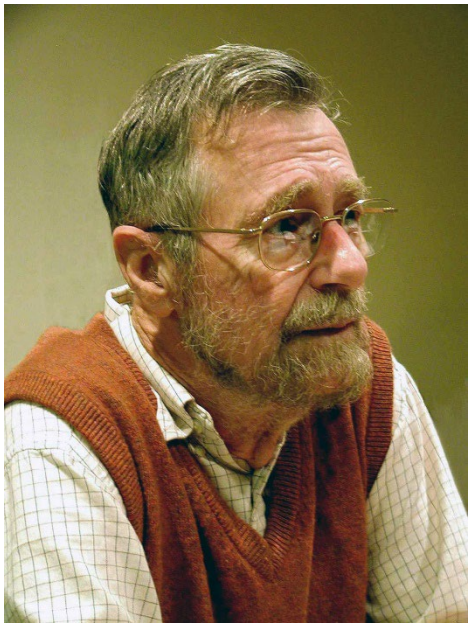
Who is this?



Edsger Dijkstra
(1930 – 2002)

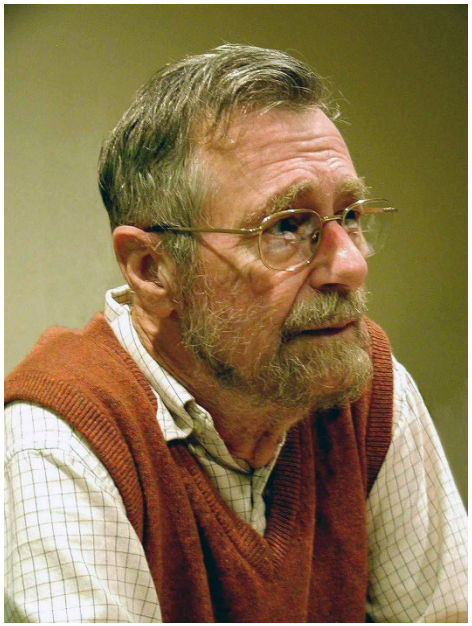
Dutch computer scientist, coined "structured programming", known for his work on mutual exclusion, winner of the first ACM Turing Award in 1972.

Edsger Dijkstra



Big believer in the importance of simplicity and highly critical of baroque programming languages with lots of features, especially PL/I, the C++ of the day.

Edsger Dijkstra



"I absolutely fail to see how we can keep our growing programs firmly within our intellectual grip when by its sheer baroqueness the programming language – our basic tool, mind you! – already escapes our intellectual control."

"The Humble Programmer", CACM, October 1972.

why I'm not a fan of STL

Algorithmic and tuning choices, performance and size tradeoffs and other issues are much *harder to spot* when hidden inside an opaque template, especially *where the whole point* of the template is that *you're not supposed to care how it works* inside.

How big is an STL string?

How big is an STL string?

It depends on the compiler and OS.

With the MS compiler, an empty string takes 36 bytes = 28 bytes on the stack + 8 bytes on the heap. New'ing a string, it's the same two chunks but both go on the heap. Adding "hello" does not require additional memory. Adding another 28 characters requires another 48 bytes.

With Cygwin g++, an empty string takes 8 bytes whether it's on the heap or the stack. But the moment you add "hello" to it, it jumps to 38 bytes. Adding another 28 characters requires another 58 bytes allocated and 30 freed.

With g++ under WSL, an empty string takes 32 bytes whether it's on the heap or the stack. Adding "hello" does not require additional memory. Adding another 28 characters requires another 34 bytes.



Who is this?



Don Knuth

Professor Emeritus at Stanford

Famous for his *Art of Computer Programming* textbooks.
(Seven planned, only 3 written.)

He was my professor 46 years ago for CS 144A and B, introduction to algorithms and data structures.

Image source: https://en.wikipedia.org/wiki/File:KnuthAtOpenContentAlliance.jpg

Don Knuth on programming as art



"When I speak about computer programming as an art, I am thinking primarily of it as an *art form*, in an aesthetic sense. The chief goal of my work as educator and author is to help people learn how to write *beautiful programs*."

-- Don Knuth, CACM, December 1974

Source: http://www.paulgraham.com/knuth.html



"My feeling is that when we prepare a program, it can be *like composing poetry or music*; … programming can give us both *intellectual and emotional satisfaction*, because it is a *real achievement* to master complexity and to establish a system of consistent rules."

So, what is beautiful code?

beautiful code

“I may not know art but I know what I like.”

“If they can’t tell me how they would improve it if they could do it over, I won’t hire them.”

“Artists are never done with a piece, they just stop working on it.”

beautiful code

To me, beautiful code should be:

1. Elegant, easy to read, sparse, clear.
2. Stylish.
3. Fast and efficient.

It should make a hard problem seem easy.

To get better, you must be able to criticize your work.

beautiful code

Consider the problem:

Write a function that determines if a C-string contains doubled letters.

Here is one sample solution.

```
bool has_doubled_letters(const char *str) {
    const char *ptr = str + 1;
    while (*str != '\0' && *ptr != '\0') {
        if (*str == *ptr) {
            return true;
        }
        ++str, ++ptr;
    }
    return false;
}
```

Here is another.

```
bool HasDoubledLetters( const char *s )
   {
   while ( *s && s[ 0 ] != s[ 1 ] )
      s++;
   return *s;
   }
```

Can we say if we like one better than another?

```
bool has_doubled_letters(const char *str) {
    const char *ptr = str + 1;
    while (*str != '\0' && *ptr != '\0') {
        if (*str == *ptr) {
            return true;
        }
        ++str, ++ptr;
    }
    return false;
}
```

```
bool HasDoubledLetters( const char *s )
   {
   while ( *s && s[ 0 ] != s[ 1 ] )
      s++;
   return *s;
   }
```

agenda

1. Course staff
2. Course organization
3. Learning objectives
4. System design
5. The project
6. Basic parts of a search engine
7. Planning your engine
8. Teamwork
9. The world around us
10. Beautiful software
11. Problem solving

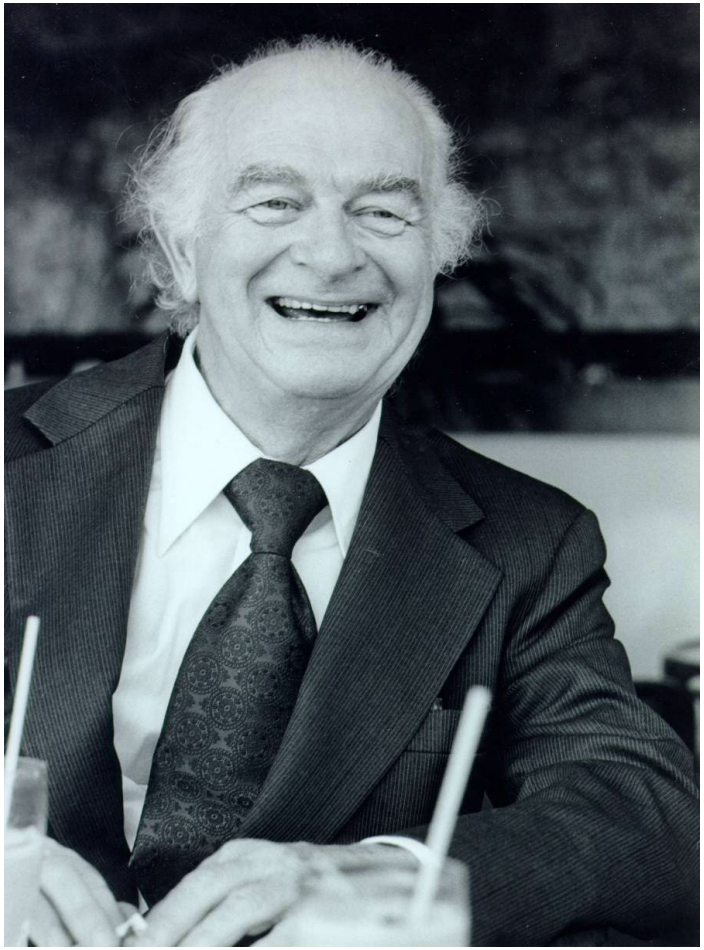
A lot of problem-solving is coming up with ideas of how to do something.

A lot of problem-solving is coming up with ideas of how to do something.

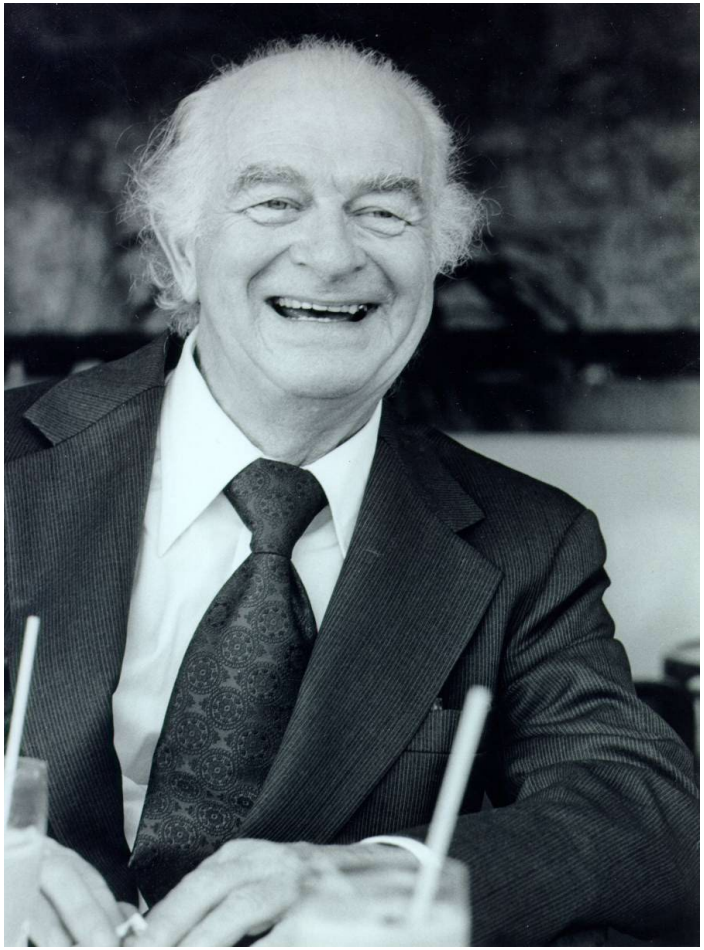
For example, a lot of debugging is coming up with ideas of how you got it wrong and how you'll find your mistake.

The number 1 problem many people make is stopping with their first idea.

They refuse to ask what's wrong with it and how it could be made better or if a different idea would be even better.



Who is this?

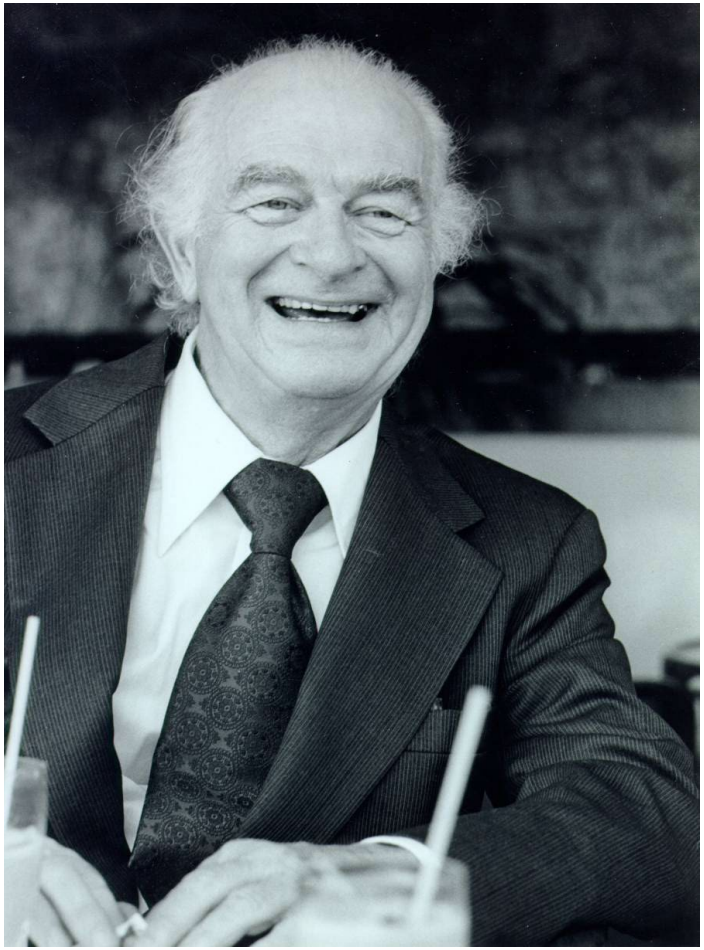


Linus Pauling (1901-1994)

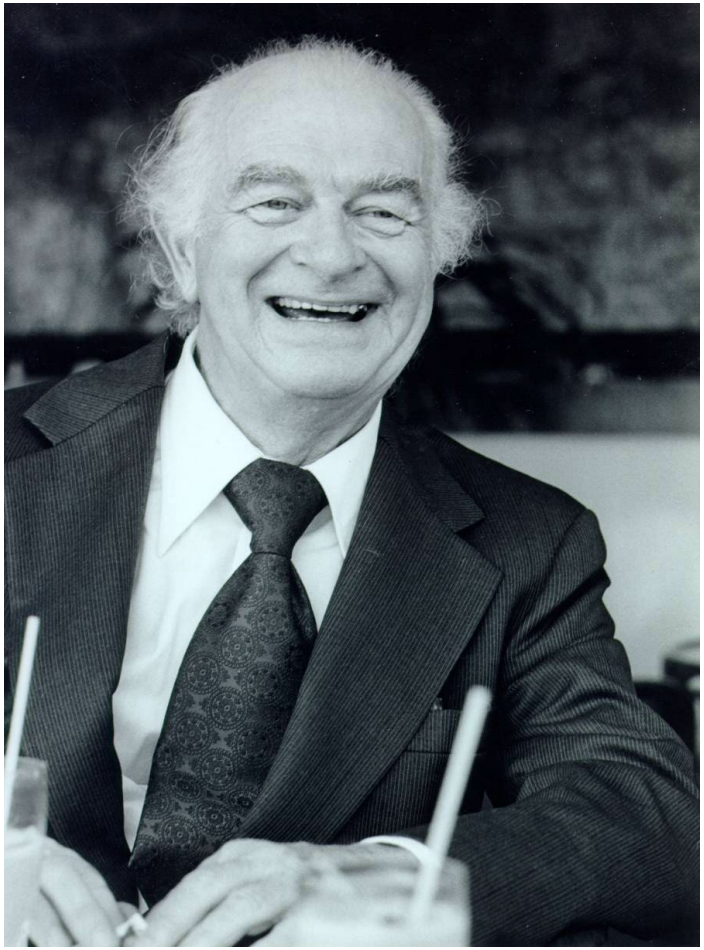
American chemist, biochemist, peace activist.

Nobel Prizes in Chemistry and Peace.

Image source: http://www.villages-news.com/linus-pauling-and-prostate-cancer/



He was asked, *"How do you get so many good ideas?"*



"If you want to have good ideas *you must have many ideas*. Most of them will be wrong, and what you have to *learn* is *which ones to throw away*."

-- Linus Pauling

As quoted by Francis Crick in his presentation, "The Impact of Linus Pauling on Molecular Biology" (1995).

Do you think people will steal your new idea?



https://www.youtube.com/watch?v=eywi0h_Y5_U

new ideas

If an idea is *really new*, it's surprisingly hard to give it away.

If it's *really good*, most people will think you're crazy.

Most people have trouble imagining how their life would be different and why they would want that.

Problem-solving versus
problem-choosing

A fabulous solution to a problem nobody cares about is still a solution nobody cares about.

But even a mediocre solution to a really important problem can be important.

Opportunities are created when the world changes.

Something becomes possible that wasn’t before.



https://www.youtube.com/watch?v=_uXtWIg_A7M

what made search engines possible

The insight was that you could do it.

You could crawl the entire web and make and index your own private copy of the whole thing.

Processors, bandwidth and storage were so cheap you could do it.

When MSN Search went live in Jan 2005, we had 10 rows of 500 machines, each with a copy of 1/500-th of the entire web = 10 complete copies.