

EECS 498 System Design of a Search Engine

Winter 2026

Lecture 15: The frontier and robots.txt

Nicole Hamilton

[https://web.eecs.umich.edu/~nham/
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

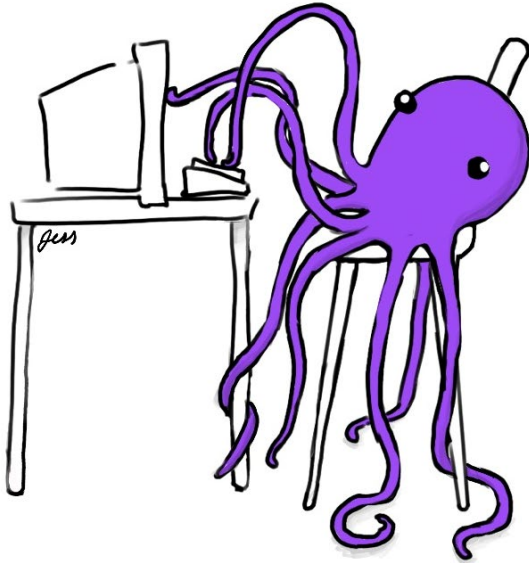
details

1. The regrade window for the midterm is open until 11:59 pm.
2. Surprisingly, about 1/4 of you have not looked at your exams on Gradescope to see how they were graded.
3. Your scores may go up silently, never down.
4. We have a cartoon mascot and a tee shirt design!
5. No HW this weekend.
6. I have finished my robots.txt solution, now working to turn that into an assignment and adjust the due date.

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

We have a design, thanks to Jess Morton.



EECS 498 WINTER 2026
SEARCH ENGINE DEV TEAM

Front



I SURVIVED NICOLE HAMILTON'S
SEARCH ENGINE DESIGN CLASS

Back

Our mascot needs a names

Let's get suggestions
and do a survey.

Octothorpe

Jeff

Zork

tf/idf

Purple

Sami

Carlos

Rahul

Jess

So Cute

Searchy

Rose

Gilbert

George

Octavious

Hamilton

Purelle

Sally

Nicky

Squishy

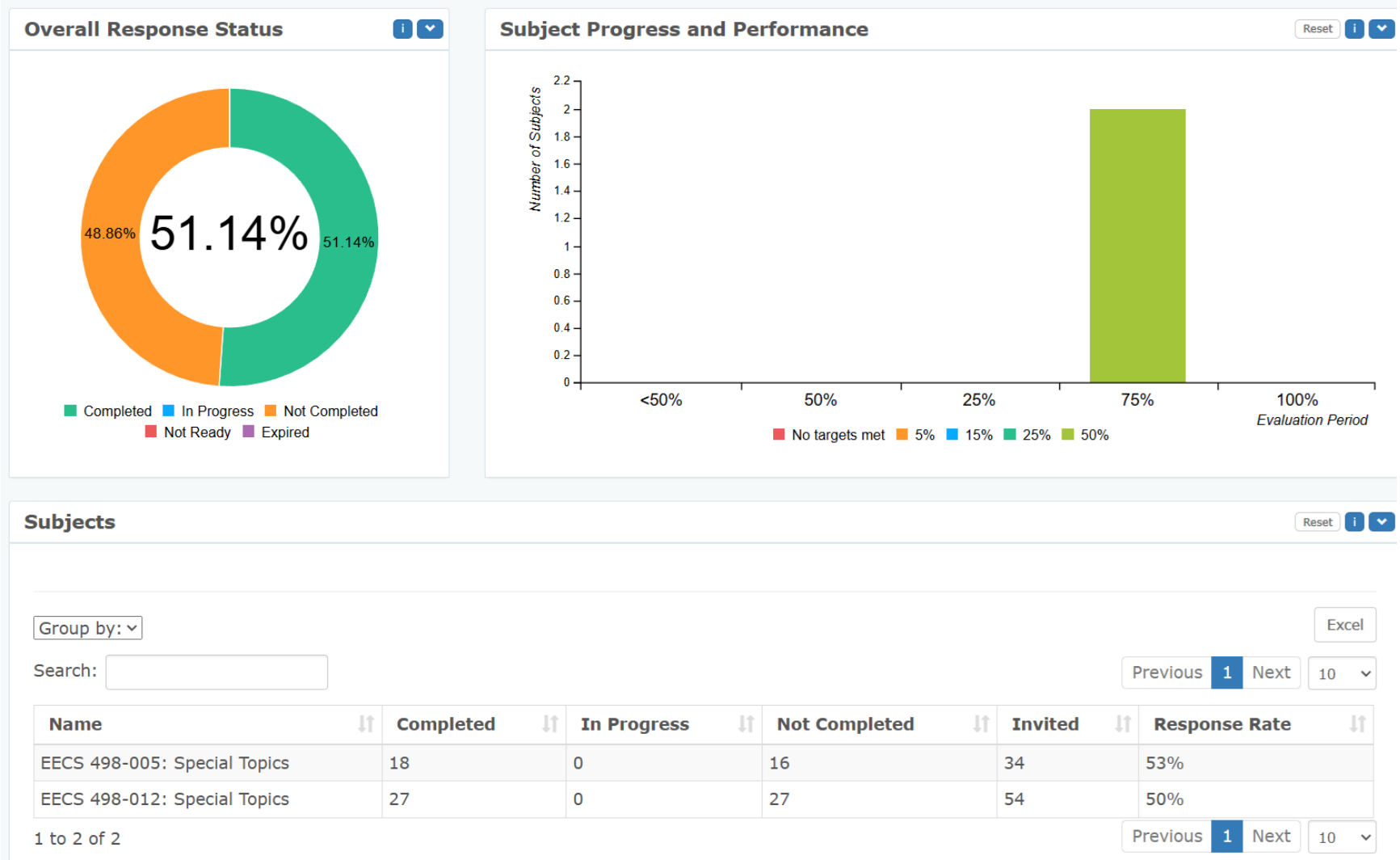
Seashell

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

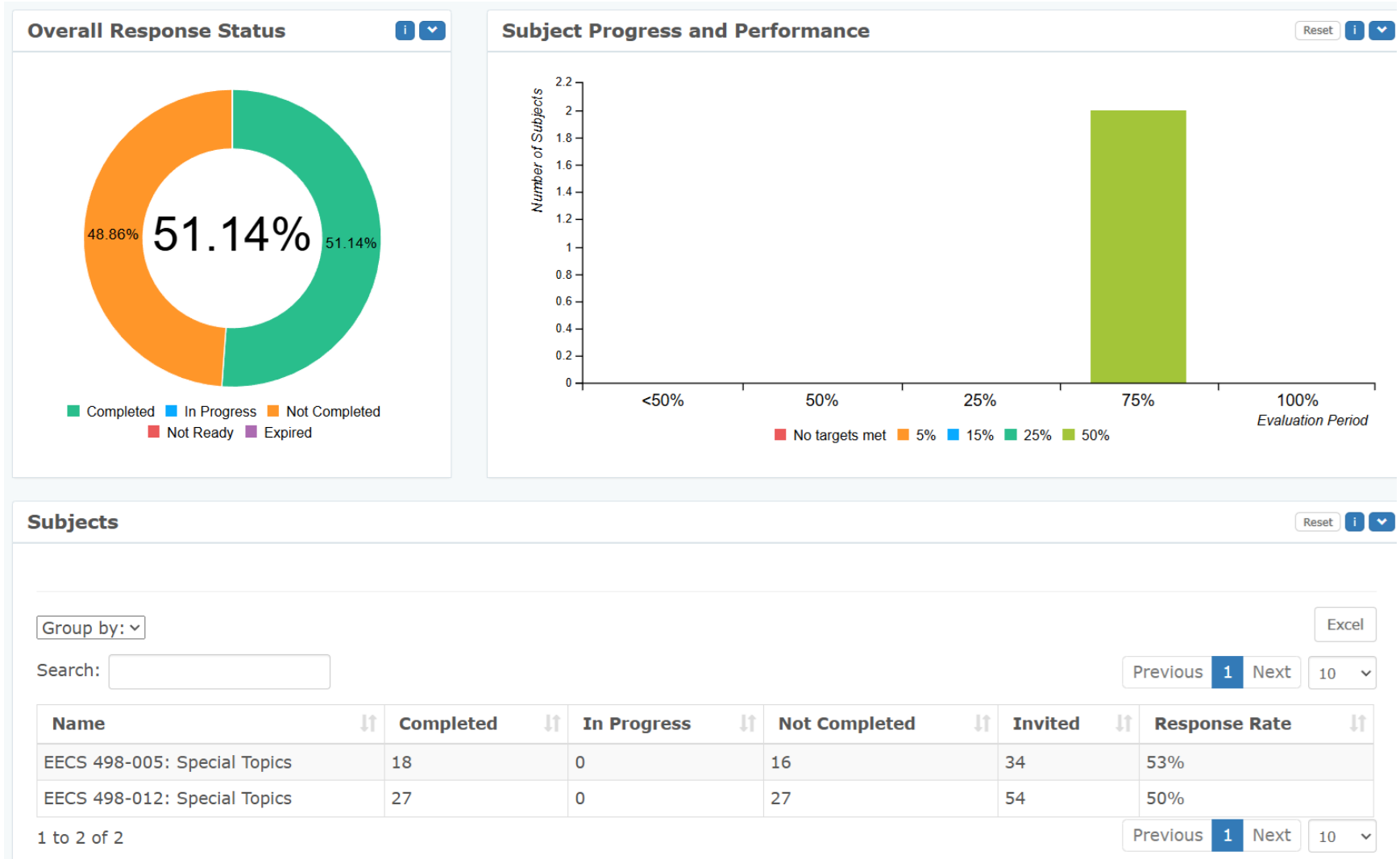
midterm course eval

Closes today at 11:59 pm. As of 8:00 am:



midterm course eval

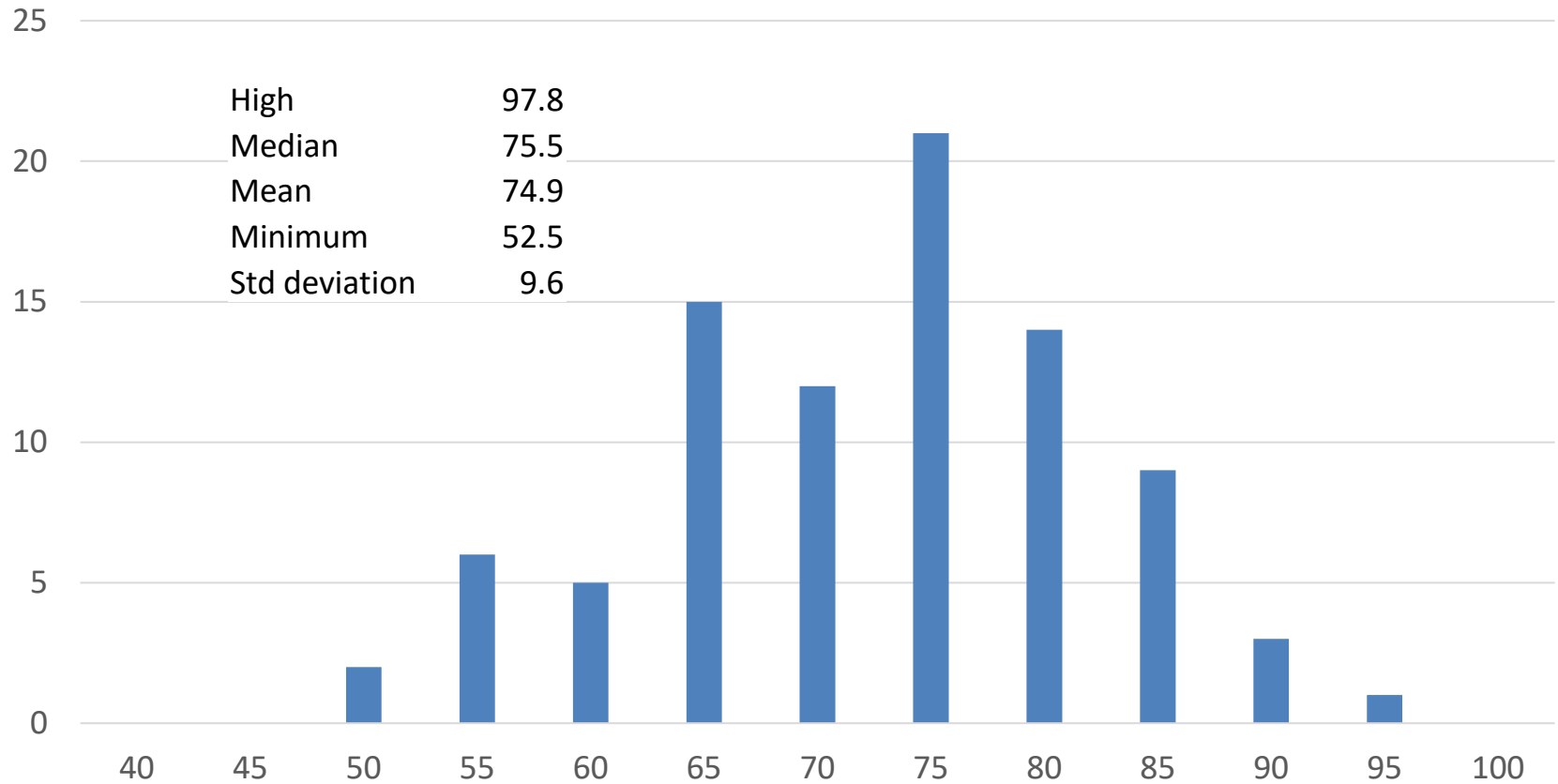
If you haven't done it, please do it now: <https://umich.bluera.com/umich/>



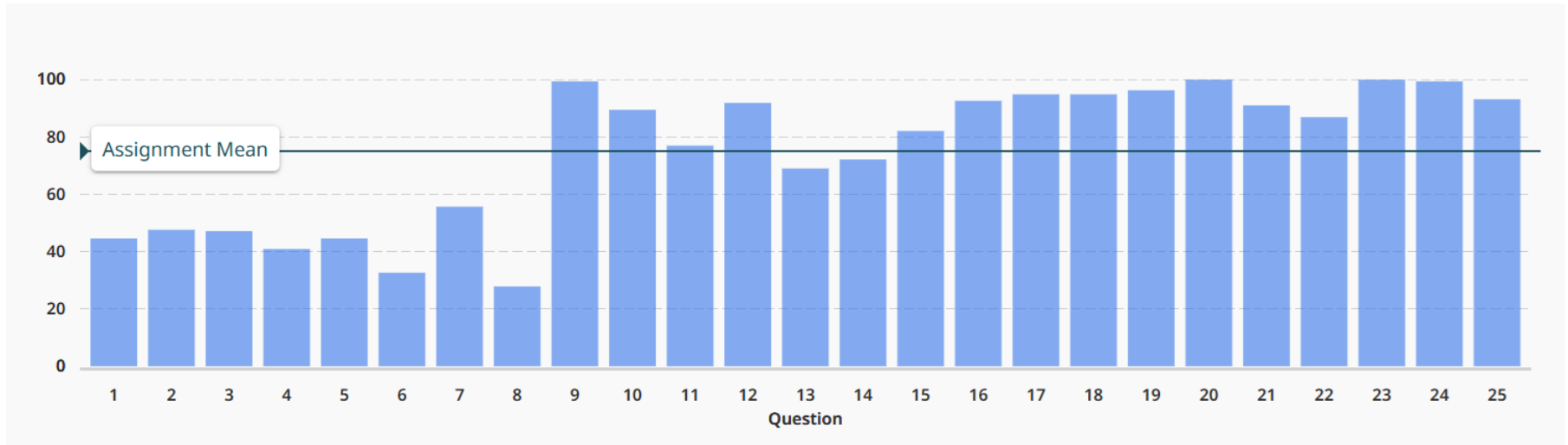
agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

midterm scores



The first 8 questions were new. The rest were taken from the W25 midterm.



Midterm 100.0 points

Minimum

52.5%

Median

75.5%

Maximum

97.8%

Mean

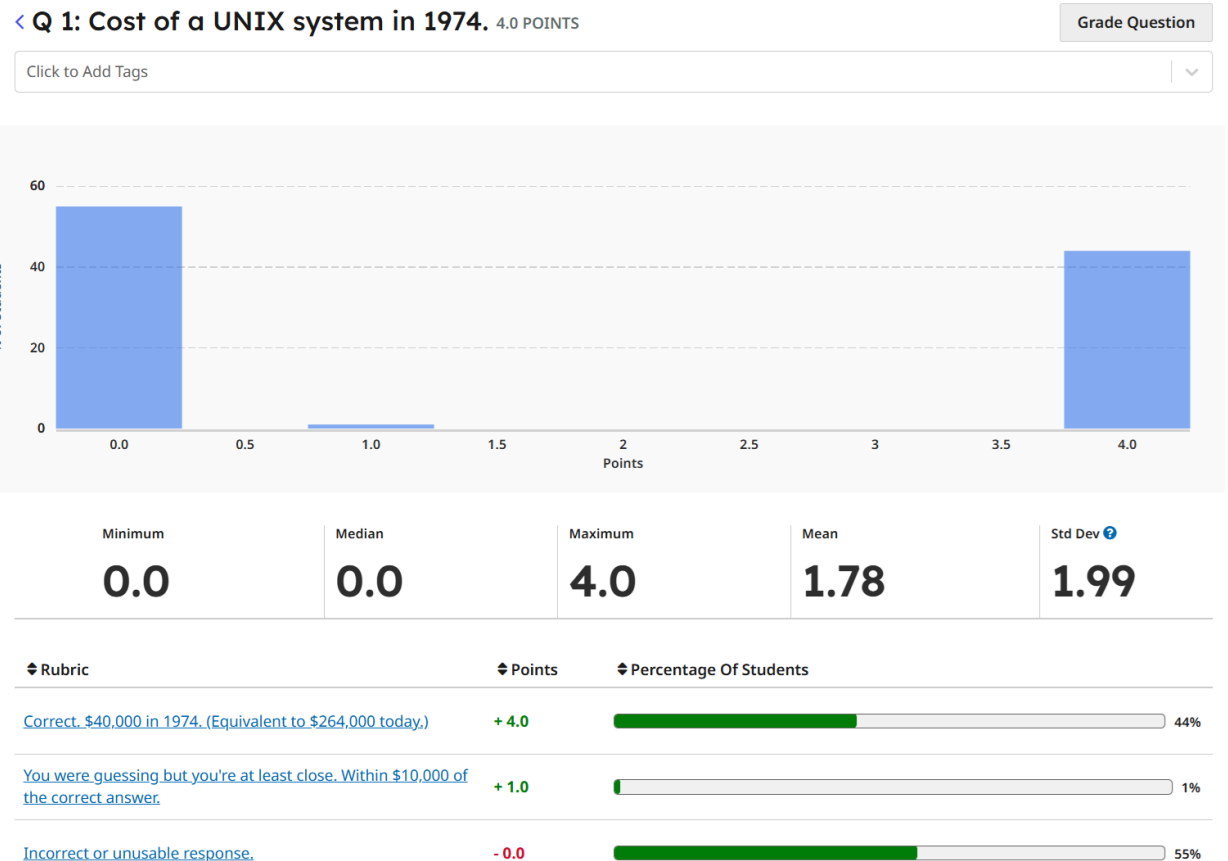
74.86%

Std Dev [?](#)

9.61%

In July 1974, in “The UNIX Time-sharing System”, Dennis Ritchie and Ken Thompson wrote, “Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive.” UNIX could run on hardware costing as little as how much? (Adjusting for inflation, \$100 then had the same buying power as \$660 today.)

\$40,000 in 1974. (Equivalent to \$264,000 today.)



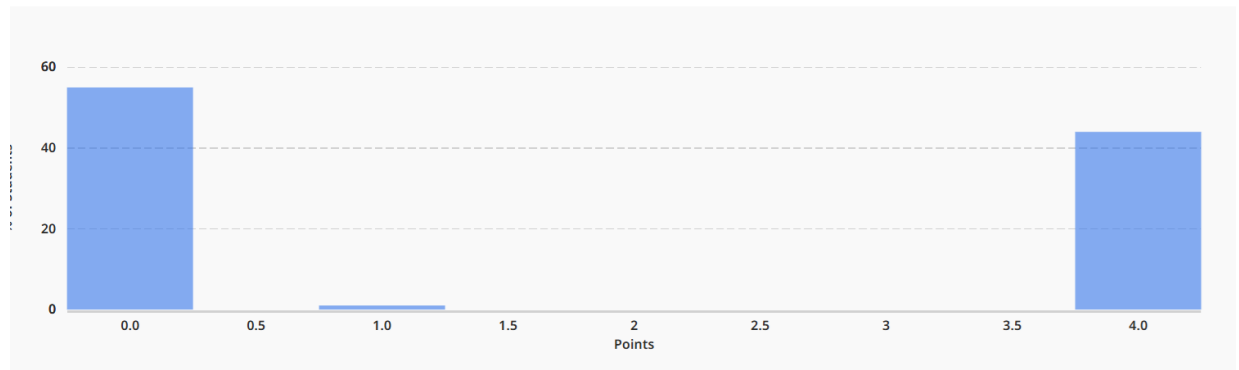
In July 1974, in “The UNIX Time-sharing System”, Dennis Ritchie and Ken Thompson wrote, “Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive.” UNIX could run on hardware costing as little as how much? (Adjusting for inflation, \$100 then had the same buying power as \$660 today.)

\$40,000 in 1974. (Equivalent to \$264,000 today.)

< Q 1: Cost of a UNIX system in 1974. 4.0 POINTS

Grade Question

Click to Add Tags



Minimum

0.0

Median

0.0

Maximum

4.0

Mean

1.78

Std Dev

1.99

◆ Rubric

◆ Points

◆ Percentage Of Students

[Correct. \\$40,000 in 1974. \(Equivalent to \\$264,000 today.\)](#)

+ 4.0

44%

[You were guessing but you're at least close. Within \\$10,000 of the correct answer.](#)

+ 1.0

1%

[Incorrect or unusable response.](#)

- 0.0

55%

Unix ran on a PDP-11/40. This is what some of you thought you could buy for \$5 (or less!) in 1974.

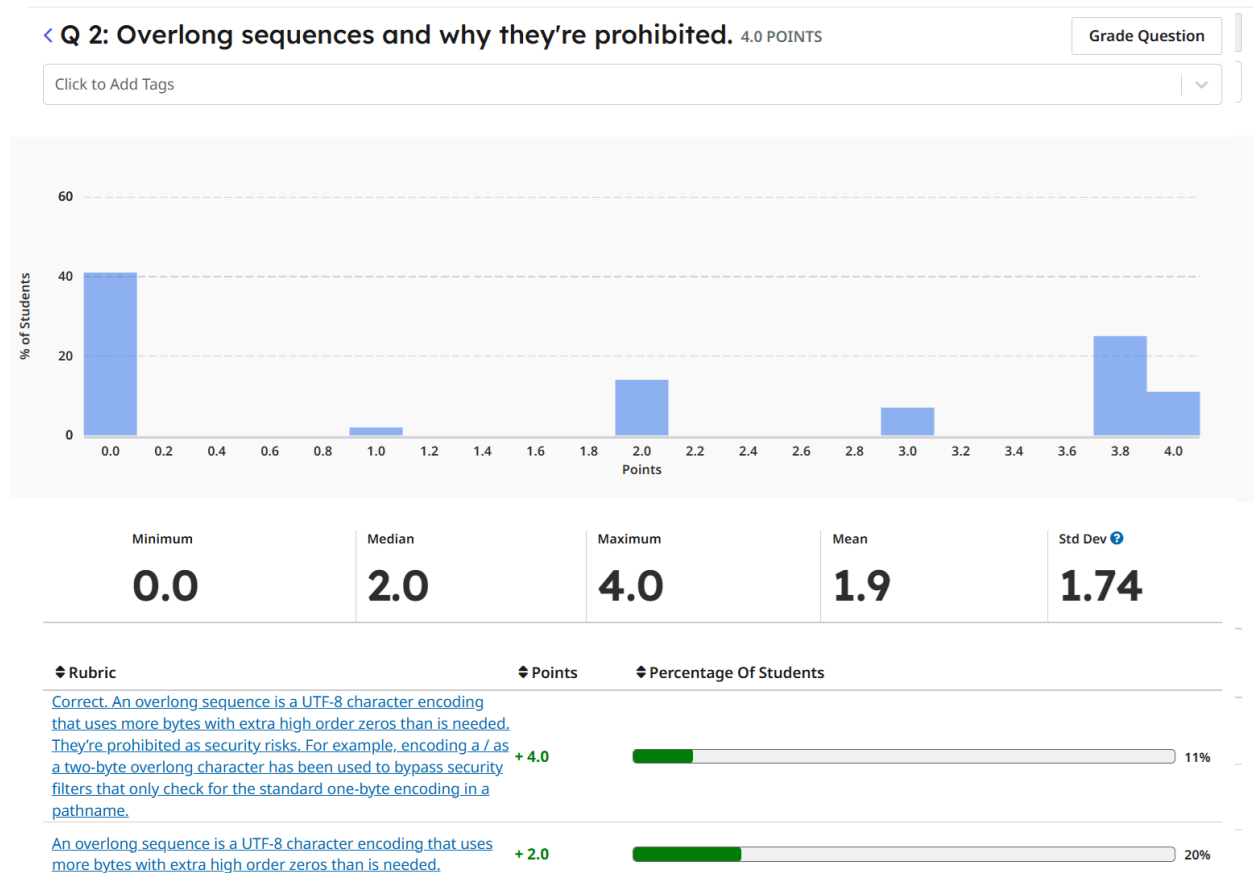


This is what most computers looked like in 1974. That's why the PDP-11 seemed inexpensive.



What are overlong sequences and why are they prohibited?
Can you give an example and why it might be a problem?

An overlong sequence is a UTF-8 character encoding that uses more bytes with extra high order zeros than is needed. They're prohibited as security risks. For example, encoding a / as a two-byte overlong character has been used to bypass security filters that only check for the standard one-byte encoding.



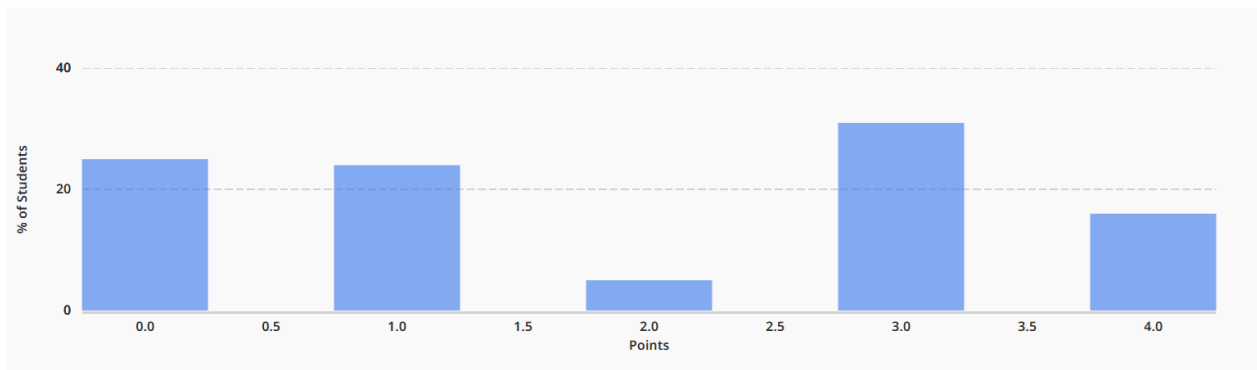
Word locations can be numbered from (a) the start of the document or (b) the start of the index. Which is better and why?

It's better to go with start of index because the deltas when crossing document boundaries will be smaller and will need shorter UTF-8 encodings. (If you go with start of document, the high order document bits changing causes a large delta.) This is important because search time is linear with index size.

< Q 3: Number locations from start of document or start of index. 4.0 POINTS

Grade Question

Click to Add Tags



Minimum

0.0

Median

2.0

Maximum

4.0

Mean

1.89

Std Dev

1.48

◆ Rubric

◆ Points

◆ Percentage Of Students

Correct. It's better to go with start of index because the deltas when crossing document boundaries will be smaller and will need shorter UTF-8 encodings. (If you go with start of document, the high order document bits changing causes a large delta.) This is important because search time is linear with index size.

Advocates start of index because it takes less space, e.g., mentioning need to keep doc ID otherwise, but offering a poor explanation of why, e.g., omitting mention of crossing document boundaries, and/or a poor connection to search time.

+ 4.0

16%

+ 3.0

31%

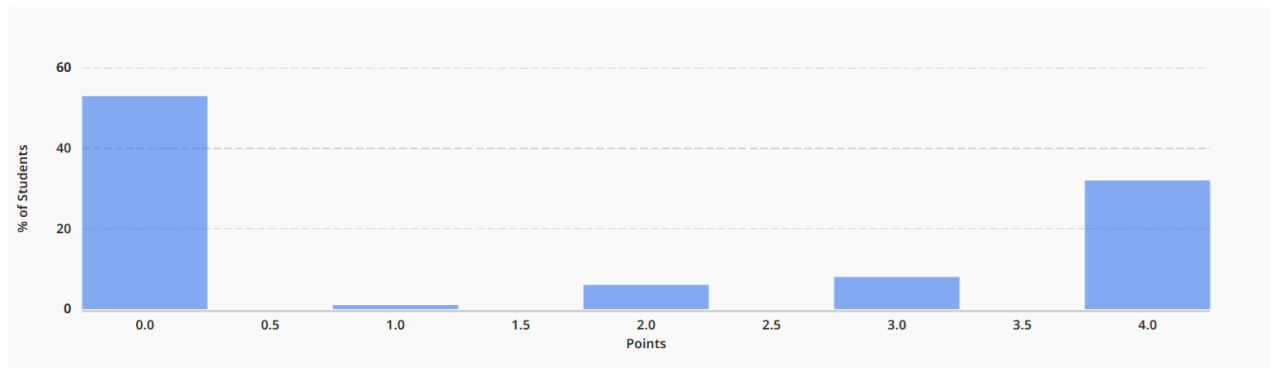
You read an HTTP page with `LinuxGetUrl` and it returns a redirect to an HTTPS page but it's the wrong one. But if you type the same HTTP URL into your browser, it works. What's going on and why? How does this relate to your project?

Your browser isn't bothering to try the HTTP version. It's automatically trying the HTTPS version first and only falling back to HTTP if HTTPS doesn't work. They probably assume most sites now support HTTPS but that there are a lot old links out there that still say HTTP. It might not be a bad idea for your crawler to do this as well.

< Q 4: Wrong HTTP redirect but it works. 4.0 POINTS

Grade Question

Click to Add Tags



Minimum

0.0

Median

0.0

Maximum

4.0

Mean

1.64

Std Dev ⓘ

1.85

◆ Rubric

▼ Points

◆ Percentage Of Students

[Correct. Your browser isn't bothering to try the HTTP version. It's automatically trying the HTTPS version first and only falling back to HTTP if HTTPS doesn't work. They probably assume most sites now support HTTPS but that there are a lot old links out there that still say HTTP. It might not be a bad idea for your crawler to do this as well.](#)

+ 4.0

32%

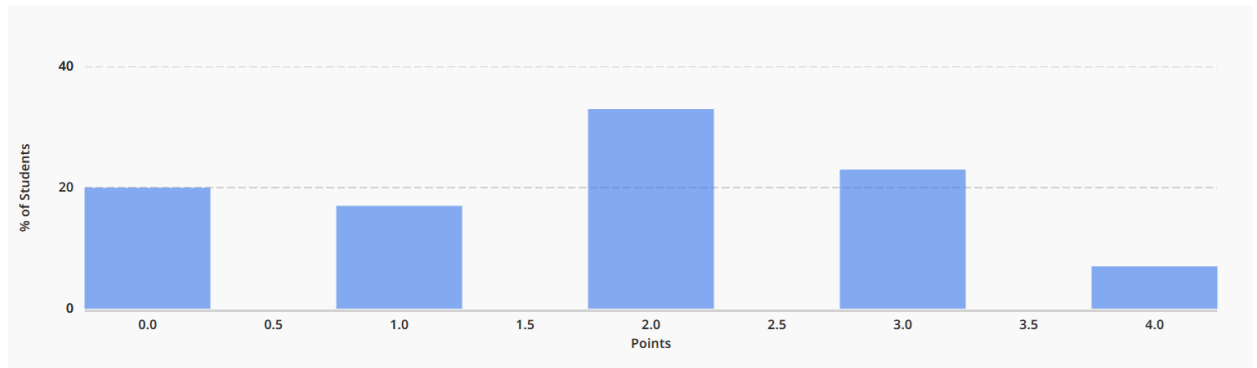
You observe that
“LinuxTinyServer 5000 .” works.
But “LinuxTinyServer 80 .” fails.
Why is that?

Under Linux, ports 0 to 1023 are privileged. Only processes running with root (superuser) privileges can bind to these ports. This is to prevent unprivileged users from creating a malicious HTTP port 80, HTTPS port 443, or other server masquerading as a legitimate system application.

< Q 5: LinuxTinyServer 80 . fails. 4.0 POINTS

Grade Question

Click to Add Tags



Minimum

0.0

Median

2.0

Maximum

4.0

Mean

1.78

Std Dev

1.21

◆ Rubric

◆ Points

◆ Percentage Of Students

[Correct. Under Linux., ports 0 to 1023 are privileged. Only processes running with root \(superuser\) privileges can bind to these ports. This is to prevent unprivileged users from creating a malicious HTTP port 80, HTTPS port 443, or other server masquerading as a legitimate system application.](#) + 4.0

7%

[Says port 80 is privileged or reserved but not why or what's required to use that port.](#) + 2.0

33%

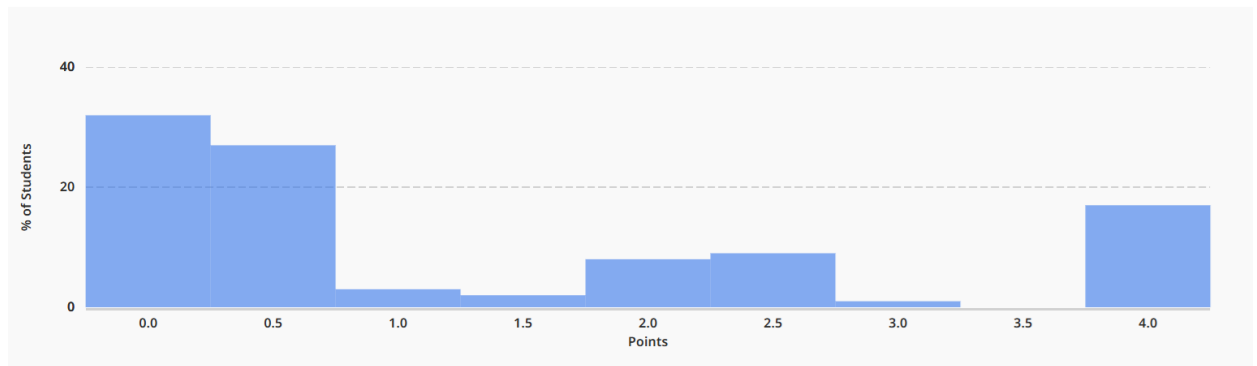
An HTML form might use GET or POST to return what's been typed by the user. What's the difference, is one better than the other for a search engine webpage, and why?

GET returns whatever's been typed as a query string in the URL. Everything after the "?" is the query string, giving name=value pairs separated by ampersands. POST returns the same information but as content in the HTTP message. GET is better for a search engine because you can give the URL to someone else and they can get the same result without having to retype the query into the form.

< Q 6: GET vs POST for a search engine form. 4.0 POINTS

Grade Question

Click to Add Tags



Minimum

0.0

Median

0.5

Maximum

4.0

Mean

1.31

Std Dev

1.48

◆ Rubric

▼ Points

◆ Percentage Of Students

Correct. GET returns whatever's been typed as a query string in the URL. Everything after the "?" is the query string, giving name=value pairs separated by ampersands. POST returns the same information but as content in the HTTP message. GET is better for a search engine because you can give the URL to someone else and they can get the same result without having to retype the query into the form.

+ 4.0

17%

If POST is used with a search engine, you couldn't just give someone the URL to get the same result. They would have to

+ 1.0

1%

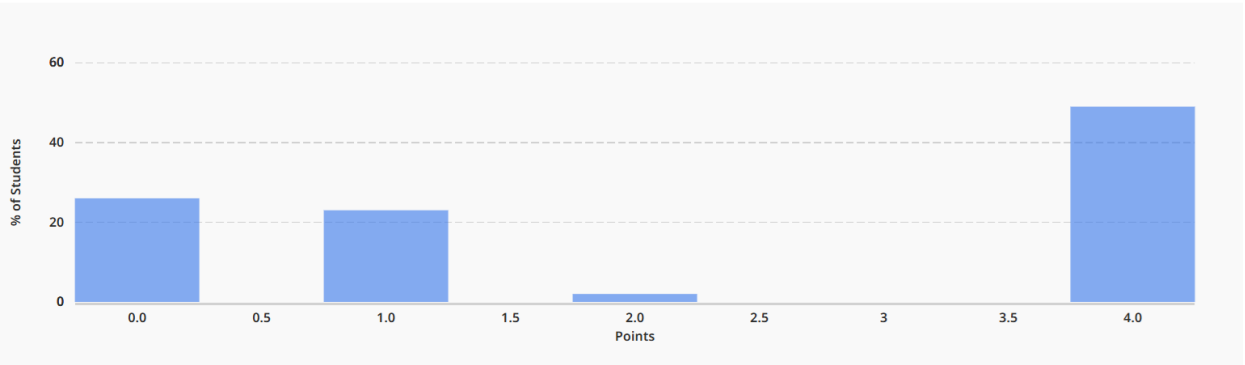
Suppose you have developed a formula for calculating static rank as a linear combination, $S = \sum a_i x_i$, where the x_i represent various heuristics, e.g., short URL, and the a_i represent the relative weights. Is it worth writing the S values into the index to save time recalculating them when you need them later?

No, this is not helpful. You should store only raw data, e.g., URL length, not a heuristic test like short URL, and definitely not the S values. You'll want to experiment and revise your weightings and your heuristics to tune your ranker once everything is running, instantly making any S values stored in your index out of date. And anyway, calculating a simple sum of products is fast.

< Q 7: Static rank is a sum of products. Store the result? 4.0 POINTS

Grade Question

Click to Add Tags



Minimum	Median	Maximum	Mean	Std Dev
0.0	2.0	4.0	2.23	1.79

Rubric	Points	Percentage Of Students
Correct. No, this is not helpful. You should store only raw data, e.g., URL length, not a heuristic test like short URL, and definitely not the S values. You'll want to experiment and revise your weightings and your heuristics to tune your ranker once everything is running, instantly making any S values stored in your index out of date. And anyway, calculating a simple sum of products is fast.	+ 4.0	49%
Says no because the formula will change but misses the point that you should store only raw data, not heuristics or weights.	+ 2.0	2%

Once you have your engine working on a single machine, what changes will you need to make to distribute it across multiple machines?

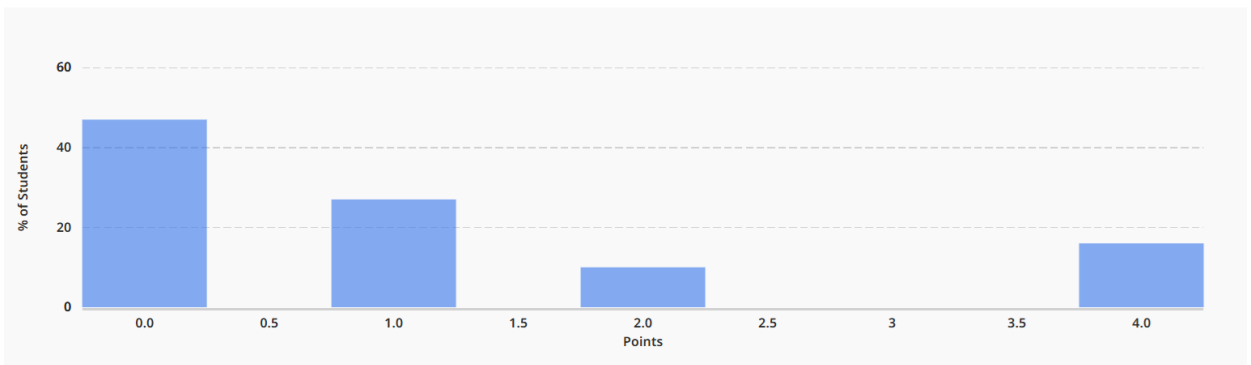
When the front end gets a query, it should broadcast it to the sibling machines and then merge what they send back with the local results.

When the crawler finds a URL, it should decide which machine gets it, e.g., by hashing the URL modulo number of machines. It should crawl only its own slice of the web and send URLs in the other slices to the appropriate machines.

< Q 8: Changes required to distribute your engine. 4.0 POINTS

Grade Question

Click to Add Tags



Minimum

0.0

Median

1.0

Maximum

4.0

Mean

1.11

Std Dev

1.42

◆ Rubric

▼ Points

◆ Percentage Of Students

Correct. When the front end gets a query, it should broadcast it to the sibling machines and then merge what they send back with the local results. When the crawler finds a URL, it should decide which machine gets it, e.g., by hashing the URL modulo number of machines. It should crawl only its own slice of the web and send URLs in the other slices to the appropriate machines.

+ 4.0

16%

Crawler should crawl only its own slice of the web and send URLs for the other slices to the appropriate machines.

+ 2.0

6%

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

robots.txt

File at root of domain which specifies whether robot crawlers are allowed or disallowed from accessing various parts of a website.

Applies only to paths within the protocol, host and port where it's posted.

robots.txt

Example: <https://umich.edu/robots.txt>

User-agent: *

Disallow: /admin/

Disallow: /announcements/

Disallow: /pingtest/

Disallow: /search

robots.txt

Expected to be encoded as Utf-8.

Any byte order mark (BOM) is ignored.

Line-oriented.

Comments beginning with #, and User-agent, Allow and Disallow directives must start at the beginning of a line.

robots.txt

A series of rules.

Each rule may have one or more User-agent statements followed by one or more Allow, Disallow, or Crawl-delay statements.

Comments and blank or invalid lines are ignored.

User-agent, Allow, Disallow, and Sitemap directives and robot names are case-insensitive.

Paths are case-sensitive.

User-agent statement specifies a single robot name that matches it appears anywhere in the crawler's name.

A Sitemap directive specifies an XML file containing lists of paths for robots to crawl.

User-agent

The robot must obey the highest priority rule that contains a User-agent line whose value contains the name of the robot as a substring.

If no such record exists, it should obey the highest priority User-agent specifying “*”.

If no record exists specifying either “*” or the robot, or no records are present at all, access is unlimited.

paths

Allow and Disallow directives specify a path, which must start with / or *, is compared against the start of the path portion of the URL being considered.

If the first part of the URL matches the path, the rest of the path doesn't matter.

The path may contain * and \$ wildcard characters.

* matches any number of characters.

\$ matches the end of the URL.

Paths are URL-decoded (except for %2f = "/") before being compared.

paths

If more than one directive applies based on the path, choose the one which is:

1. More specific, i.e., matches more literal characters.
2. Ends in \$.
3. Less restrictive, i.e., choose allow over disallow.

```
//      This table illustrates some examples:
//
//      Record Path          URL path          Matches
//      /tmp                  /tmp            yes
//      /tmp                  /tmp.html       yes
//      /tmp                  /tmp/a.html     yes
//      /tmp/                 /tmp            no
//      /tmp/                 /tmp/           yes
//      /tmp/                 /tmp/a.html     yes
//
//      /a%3cd.html           /a%3cd.html     yes
//      /a%3Cd.html           /a%3cd.html     yes
//      /a%3cd.html           /a%3Cd.html     yes
//      /a%3Cd.html           /a%3Cd.html     yes
//
//      /a%2fb.html           /a%2fb.html     yes
//      /a%2fb.html           /a/b.html       no
//      /a/b.html             /a%2fb.html     no
//      /a/b.html             /a/b.html       yes
//
//      /%7Ejoe/index.html    /~joe/index.html yes
//      /~joe/index.html      /%7Ejoe/index.html yes
```

```
//  
// Robot.txt syntax  
//  
  
// <robots.txt> ::= <Line> { <Line> }  
// <Line> ::= { <blank line> | '#' <arbitrary text> | <Rule> } <LineEnd>  
// <Rule> ::= <UserAgent> { <UserAgent> } <Directive> { <Directive> }  
// <UserAgent> ::= "User-agent:" <name>  
// <Directive> ::= <Permission> <path> | <CrawlDelay>  
// <Permission> = "Allow:" | "Disallow:"  
// <wildcard> ::= <a string that may contain *>  
// <Crawl-delay> ::= "Crawl-delay:" <integer>
```

```

class RobotsTxt
{
private:

    std::vector< class Rule * > rules;

    Directive *FindDirective( const Utf8 *user, const Utf8 *url,
        int *crawlDelay = nullptr, class Rule **rule = nullptr );

public:

    // Constructor to parse the robots.txt file contents into a
    // set of rules.

    RobotsTxt( const Utf8 *robotsTxt, const size_t length );

    // Destructor

    ~RobotsTxt( );

    // Check a full URL against the rules in this robots.txt.
    // Return true if access is allowed. Optionally report
    // any crawl delay.

    bool UrlAllowed( const Utf8 *user, const Utf8 *url,
        int *crawlDelay = nullptr );

    // Check the path portion of a URL against the rules in this
    // robots.txt. Return true if user is allowed to crawl the path.
    // Optionally report any crawl delay.

    bool PathAllowed( const Utf8 *user, const Utf8 *path,
        int *crawlDelay = nullptr );
};

```

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

crawler

1. Manage a frontier of new links to be crawled.
2. Decide what will or will not be crawled and in what order.
3. Keep track of what's already been crawled.
4. Read pages over HTTP and HTTPS.
5. Obey robots.txt files.
6. Deal with redirects.

All of this has to be highly multithreaded so you don't wait on slow sites, instead overlapping them.

You will also want to spread it across multiple machines.

crawler

Typically maintains pool of worker processes or threads to read and parse webpages.

Each worker retrieves the file and queues it for the HTML parser which creates an object.

The resulting stream of objects are then passed to the Index builder.

the frontier

1. Basic problem: A very large list of URLs that you have not yet crawled. What should you crawl next?
2. Some links are obviously better than others, e.g., .edu vs. .biz.
3. You don't want to DOS anyone.
4. Some pages are prohibited by robots.txt, meaning you will need to cache this for each domain.
5. If you are crawling on multiple machines, you will need to decide how to split up the work to avoid duplicate crawling but combine the results.
6. Just because you crawled a page doesn't mean you want to index it or follow any of its links. You may want to blacklist it.

questions

1. Should crawling be a depth or breadth-first search of the web?
2. How would you decide whether to add a link to frontier in the first place?
3. Will you capture anchor text or number of links?
4. How would you decide which links already on the frontier should be crawled next?
5. What do you need to cache per domain?
6. Will you have a blacklist?
7. Will you do language or porn detection? How?

the frontier

Often described as a priority queue problem.

But you have a choice when to do the priority calculations.

You can do the calculation as you add it your queue or as you pull things off.

But what happens if you go in strict priority order?

Mercator

The Mercator system in 2001 by Marc Najork and Allan Heydon used a complex system of queues.

<http://www.cs.cornell.edu/courses/cs685/2002fa/mercator.pdf>

(Please read this article.)

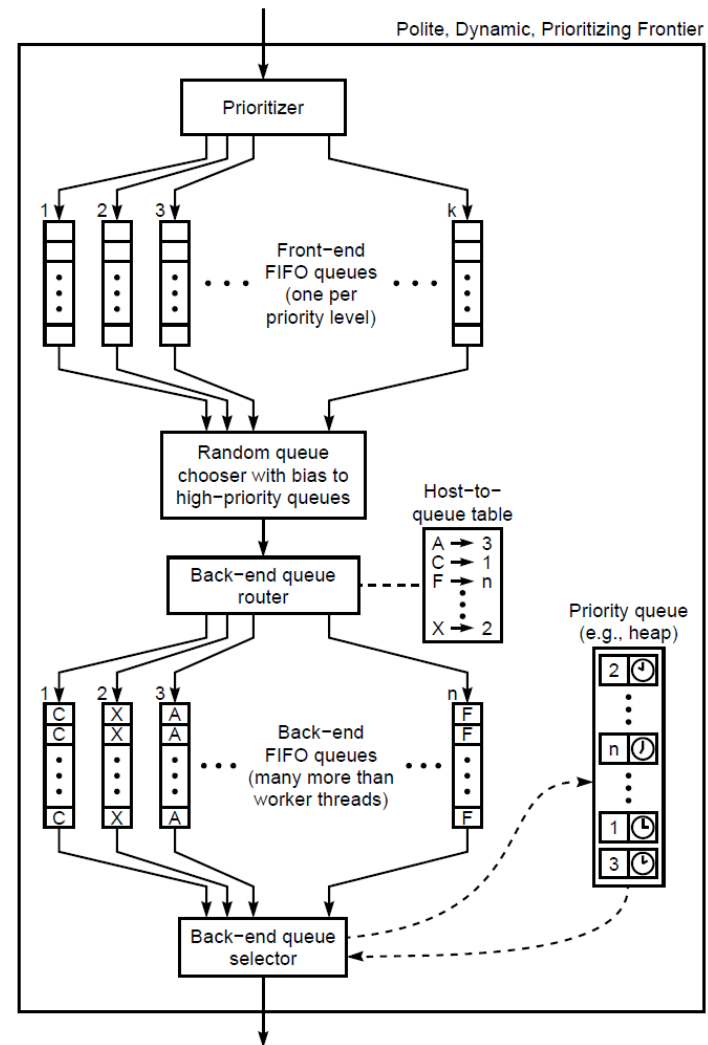


Figure 3: Our best URL frontier implementation

A simpler strategy

Assume there are a very large number of links on your frontier.

1. Randomly choose n links and rank them using whatever information you have available.
2. Crawl the top k links.
3. Repeat.

agenda

1. details
2. tee shirts
3. midterm course eval
4. exam review
5. robots.txt
6. the frontier
7. distributing your engine

basic problem

Distribute the functionality across multiple machines.

Minimize losses due to communication.

basic question

Will all your machines be identical?

distributing your index

Avoid having a master.

If you have n crawlers, use a $\text{hash}(\text{URL or domain}) \% n$ to decide what machine should crawl it.

If a given machine crawls a URL, it should be the one to index it.

URLs belonging to another machine can be batched.

When a query is presented, it should be broadcast to all its sibling machines, which respond with results, that are merged before reporting.