Lecture 01: Introduction

COSC 225: Algorithms and Visualization

Spring, 2023

Outline

1. Course Motivation and Aims
2. Course Policies (highlights)
3. Overview of the Tools
4. Introduction to HTML

Course Motivation and Aims

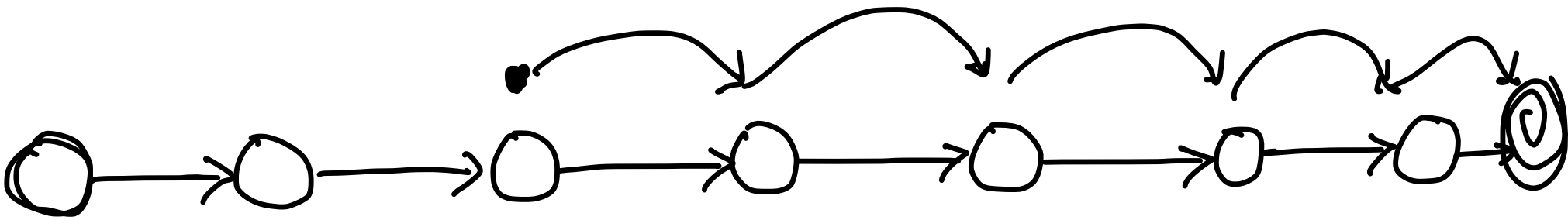
Example (from Will's research)

General Question: How do we efficiently move stuff from one place to another?

Example (from Will's research)

General Question: How do we efficiently move stuff from one place to another?

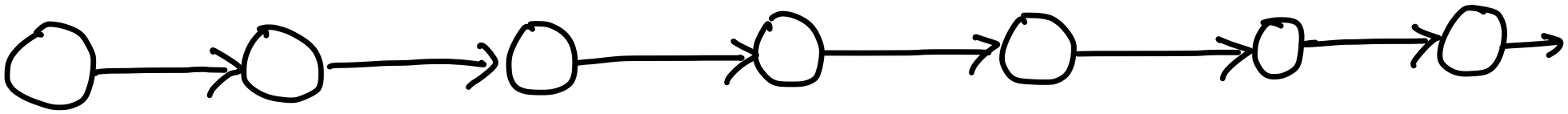
- Network is a directed path of n vertices
  - each vertex has **buffer** that can store **packets**



Example (from Will's research)

General Question: How do we efficiently move stuff from one place to another?

- Network is a directed path of n vertices
  - each vertex has **buffer** that can store **packets**
- Motion in synchronous rounds:
  - adversary injects a single packet (arbitatry location)
  - each node may forward a single packet to next buffer

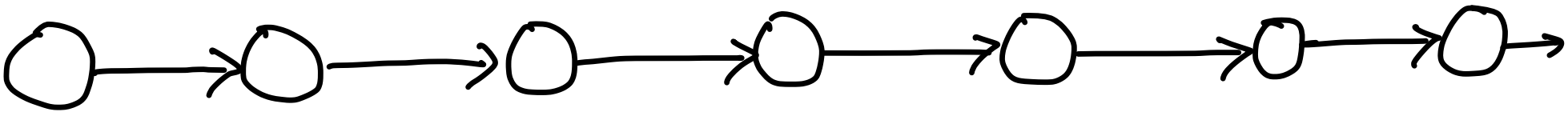


Example (from Will's research)

General Question: How do we efficiently move stuff from one place to another?

- Network is a directed path of n vertices
  - each vertex has **buffer** that can store **packets**
- Motion in synchronous rounds:
  - adversary injects a single packet (arbitatry location)
  - each node may forward a single packet to next buffer
- Packet is **delivered** when it is forwarded by right-most node

Goal: Minimize the maximum load of any buffer in the network.



Forwarding Protocols

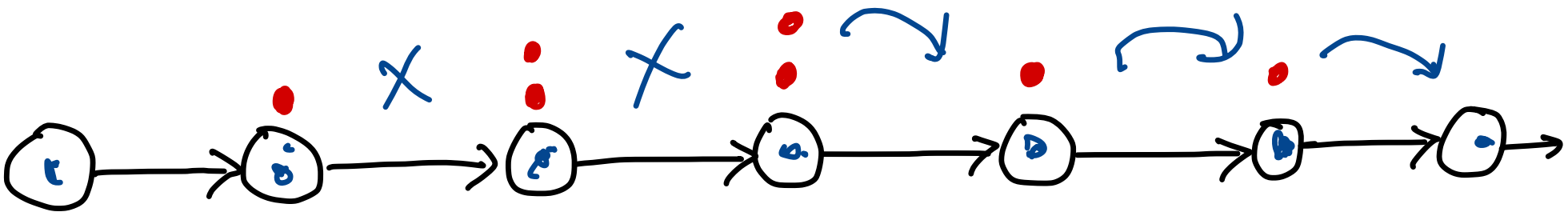
Buffer i stores $L(i)$ packets—$L(i)$ is **load** of i

Greedy Forwarding:

- If $L(i) > 0$, buffer i forwards a packet

OED Forwarding (Patt-Shamir & Rosenbaum):

- Buffer i forwards a packet if:
  - $L(i) > L(i+1)$, or
  - $L(i) = L(i+1)$ and $L(i)$ is *odd*



Confusion

1. What do these protocols do?
2. Why would I expect one to be better than the other?
3. How could I get some intuition about the behavior of these protocols?

Protocols Illustrated

- Greedy Forwarding
- OED Forwarding

Takeaways

1. Visualization can be a powerful method to help us understand and reason about algorithmic processes.
2. Interactivity gives a means of **exploring** behavior beyond pre-defined examples.
3. Web programming gives us a robust toolkit to visualize processes and to disseminate our work.

Course Aims

1. Achieve competency with web programming tools: HTML/CSS/JavaScript
2. Apply visualization techniques to illustrate algorithms
3. Apply algorithmic techniques to produce appealing visualizations

Course Structure

Meetings

- 2 lectures/week
  - guided discussion
  - small group discussion
  - mixture of lecture/discussion/activities
  - small-group activities will require your laptop
- Readings posted to course website
  - do readings **before** class

Coursework

- Coding Assignments (bi-weekly)
  - some individual
  - some in pairs
- Quizzes/in class activities
- Final Project (small group)

Attendance & Illness

Attendance

- Regular attendance is expected
- No penalty for a few missed classes
  - lectures will be recorded and posted to Moodle

Illness & Masking

- do **not** attend class if you are sick (e.g., with fever)
- if mild symptoms:
  - take a Covid test before coming to class
  - wear a mask
- otherwise come to class, masks optional

Office Hours

Will's office: SCCE C216

Drop-in (in person):

- Monday 3:30–4:30
- Friday 2:00–3:00

By appointment (in person or on Zoom):

- Thursday (time tbd)

please wear a mask to in-person office hours

Overview of the Tools

Modern Web Programming

Three basic tools:

- HTML (hypertext markup language)

  specifies document *content*, *structure*, *semantics*

  - content = text, images, etc
  - structure = logical organization of content
  - semantics = associate meaning to content items

Modern Web Programming

Three basic tools:

- HTML (hypertext markup language)

  specifies document *content*, *structure*, *semantics*

  - content = text, images, etc
  - structure = logical organization of content
  - semantics = associate meaning to content items

- CSS (cascading style sheets)

  specifies how content is displayed based on structure/semantics

Modern Web Programming

Three basic tools:

- HTML (hypertext markup language)

  specifies document *content*, *structure*, *semantics*

  - content = text, images, etc
  - structure = logical organization of content
  - semantics = associate meaning to content items

- CSS (cascading style sheets)

  specifies how content is displayed based on structure/semantics

- JavaScript

  manipulation of and interaction with content

This Course

Week 1:

- HTML structure, syntax, semantics

Week 2:

- CSS color, style, and taste

Week 3:

- Controlling HTML with JavaScript
- Adding user interactions

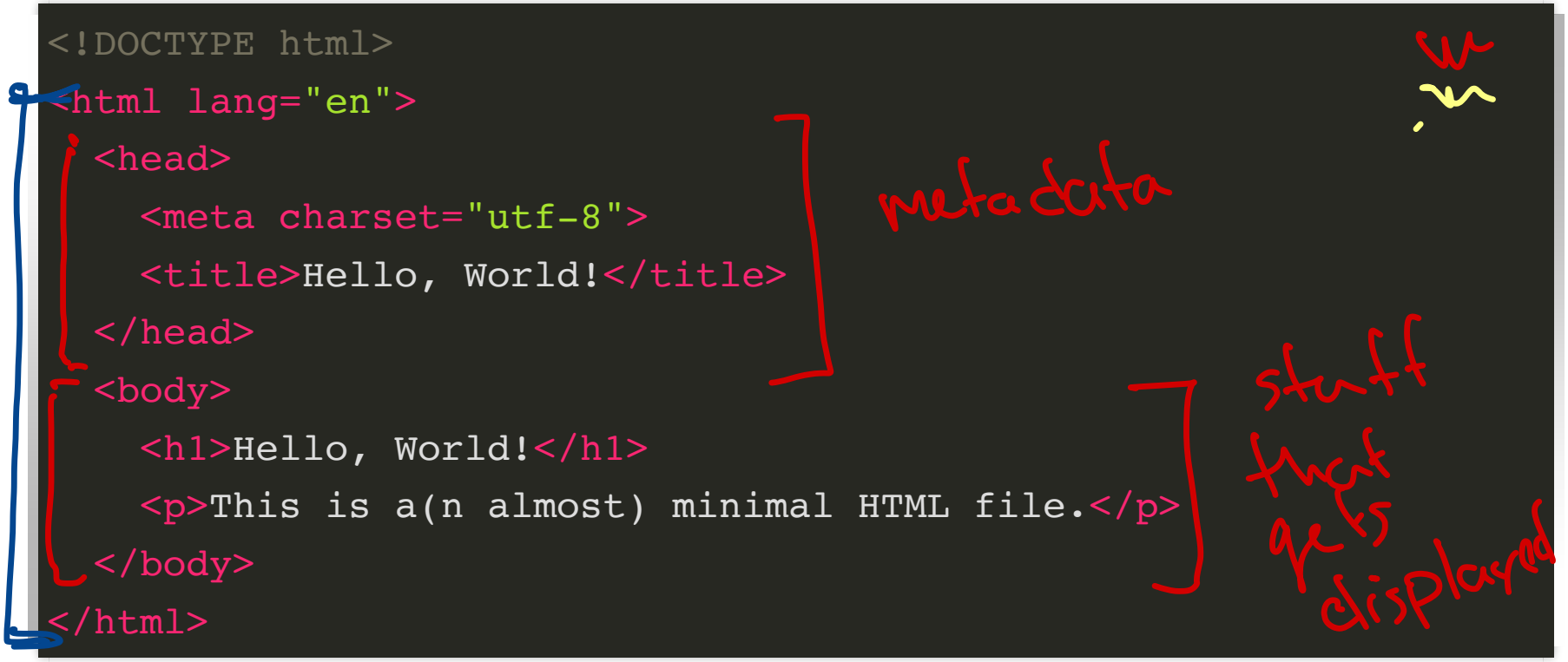
Weeks 4+:

- applying web programming tools to interesting *algorithmic* applications

Introduction to HTML

<h1>Hello, World!</h1>

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Hello, World!</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a(n almost) minimal HTML file.</p>
  </body>
</html>
```



view the site

View Page Source

- inspect elements
- modify elements

HTML Terminology

```
<p>This is a(n almost) minimal HTML file.</p>
```

tag text surrounded by < and >

- opening tag, e.g.,
- closing tag, e.g.,
- some tags are self-closing: e.g.,
 breaks up a line

HTML Terminology

```
<p>This is a(n almost) minimal HTML file.</p>
```

tag text surrounded by < and >

- opening tag, e.g.,
- closing tag, e.g.,
- some tags are self-closing: e.g.,
 breaks up a line

element logical item demarcated by a tag

- [open tag] [contents] [closing tag]
- [self-closing tag]
- tag specifies the type of element

HTML Terminology

```
<p>This is a(n almost) minimal HTML file.</p>
```

tag text surrounded by < and >

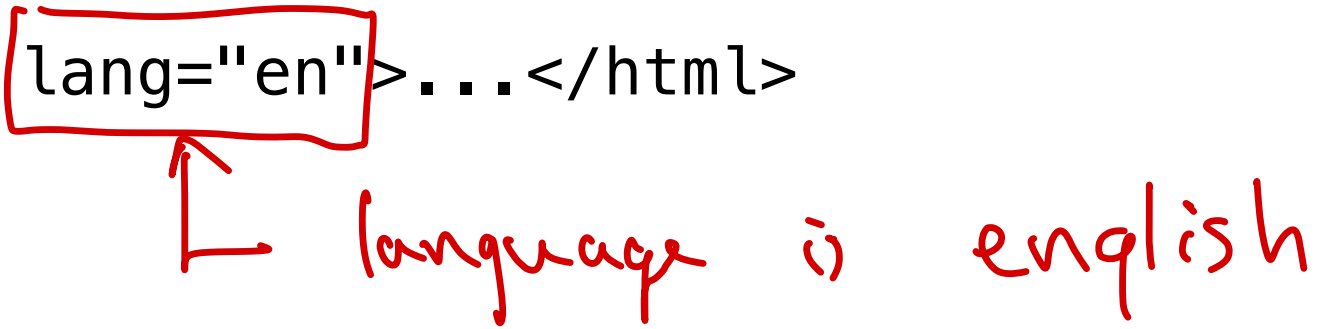
- opening tag, e.g.,
- closing tag, e.g.,
- some tags are self-closing: e.g.,
 breaks up a line

element logical item demarcated by a tag

- [open tag] [contents] [closing tag]
- [self-closing tag]
- tag specifies the type of element

attributes specify other element properties/values

- <html lang="en">...</html>



Nesting $\Longrightarrow$ Trees

Opening and closing tags can be *nested*

- Yes: `<foo><bar>contents</bar></foo>`
- No: `<foo><bar>contents</foo></bar>`

Nested tags give **tree** structure to document's elements:

- `<foo>...</foo>` is `<bar>...</bar>`'s **parent**
- `<bar>...</bar>` is `<foo>...</foo>`'s **child**

Nesting $\Longrightarrow$ Trees

Opening and closing tags can be *nested*

- Yes: `<foo><bar>contents</bar></foo>`
- No: `<foo><bar>contents</foo></bar>`

Nested tags give **tree** structure to document's elements:

- `<foo>...</foo>` is `<bar>...</bar>`'s **parent**
- `<bar>...</bar>` is `<foo>...</foo>`'s **child**

Elements can also have siblings:

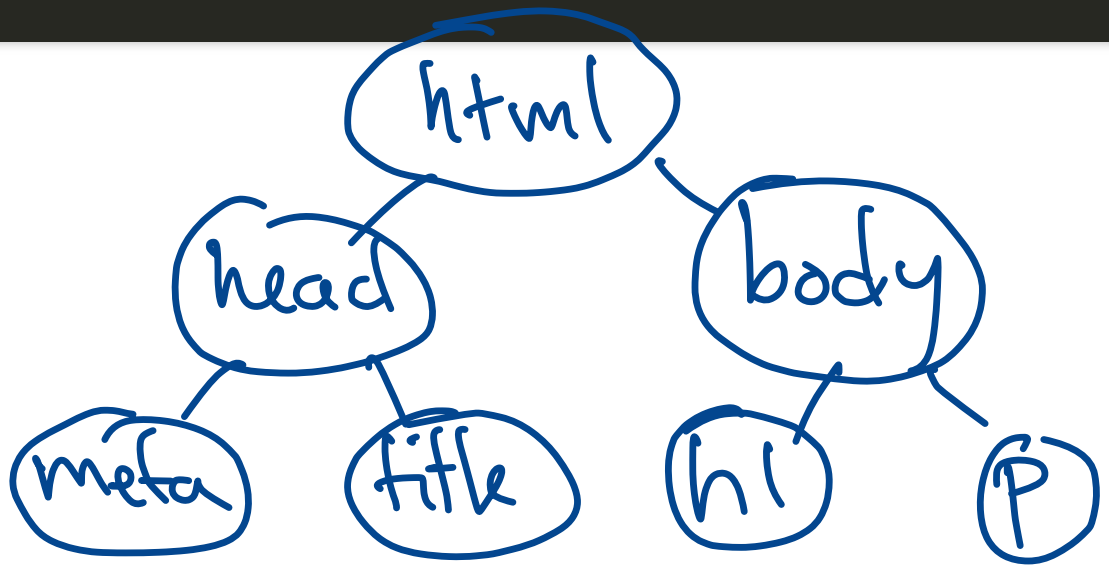
```
<p>some <em>italic</em> and <strong>bold</strong> text</p>
```

- the `em` and `strong` elements are siblings

Draw a Tree!

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Hello, World!</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a(n almost) minimal HTML file.</p>
  </body>
</html>
```

attributes



Pair Activity

Open A Basic Website

1. List all the elements/tags you see. What do you think they mean?
2. Draw the tree structure of the website

Tags: Syntax and Semantics

Tree Structure?

Consider the following page:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Block vs. Inline</title>
  </head>
  <body>
    <h1>Block vs. Inline</h1>
    <p>A paragraph with <em>italic</em> and <strong>bold</strong>
    <p>Another paragraph with <em>italic</em> and <strong>bold</s
  </body>
</html>
```

Rendered Page

How is Content Displayed?

Positions are specified left to right and top to bottom of screen

Elements laid out in order they appear in the HTML file

inline elements placed

- to right of previous element (if space available)
- below, otherwise

Inline elements: `a`, `em`, `strong`, …

block elements placed below previous element

Block elements: `h1`, `h2`, `p`, …

Who Chooses Display Details?

- browser defaults
- programmer specification: element attributes and style (CSS)

Assignment 01

1. Make a personal website: spash page & about me page
2. Only use plain HTML
3. Type everything by hand

Rest of class

Get set up with git!

- Git is version control software
  - git repositories (repos) track changes to a project
- GitHub hosts repositories
  - remote version of your repository
- GitHub can also make your repository a website!
  - GitHub pages