Lecture 08: Objects, Graphs, and DFS

COSC 225: Algorithms and Visualization

Spring, 2023

Annoucements

1. No new assignment next week
    - clean up and resubmit old assignments
2. Assignment 05 due date 03/06
3. Assignment 06 due 03/24 (after break!)
    - pair assignment!
    - posted next week

Outline

1. Graphs and DFS
2. Objects and Visualization
3. DFS Demo
4. Convex Hulls ← topic for assgt 06.

Last Time

- JavaScript Events
  - event listeners
  - responding to events
- Intro to JavaScript Objects
  - constructors, fields, methods
- Graphs
  - vertices and edges

associates clicks, etc. to DOM objects that are interacted w/

Today

More graph visualization!

- better `Graph`, `GraphVisualizer`

Visualizing algorithms!

- depth-first search

A geometric problem!

- convex hulls

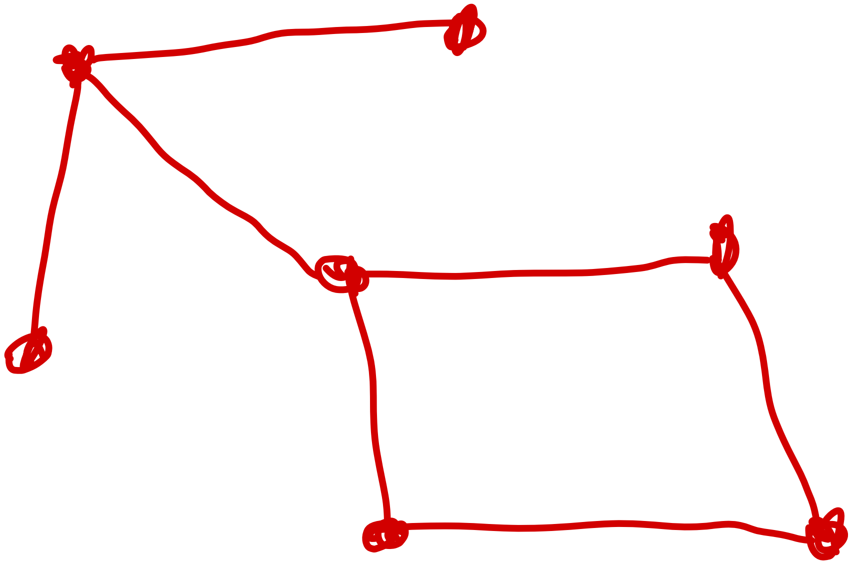
Graphs

Graphs

Mathematical abstraction of *networks*

- set V of **vertices** a.k.a. **nodes**
- set E of **edges**
  - each edge $e \in E$ is a *pair* of nodes

If $(u, v) \in E$, we say u and v are **neighbors**



Representing Graphs

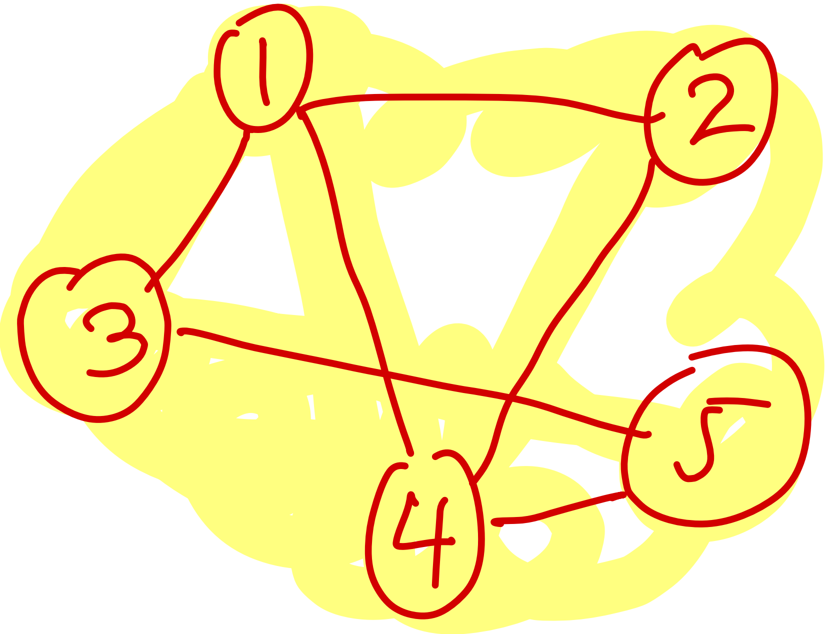
Adjacency list representation

- list (e.g., array) of vertices
- for each vertex, store a list of its neighbors

Example

- $V = \{1, 2, 3, 4, 5\}$
- $E = \{(1, 2), (1, 3), (1, 4), (2, 4), (3, 5), (4, 5)\}$

1: 2, 3, 4
2: 1, 4
3: 1, 5
4: 2, 5
5: 3, 4



Representing a Graph with Objects

JavaScript

Structure

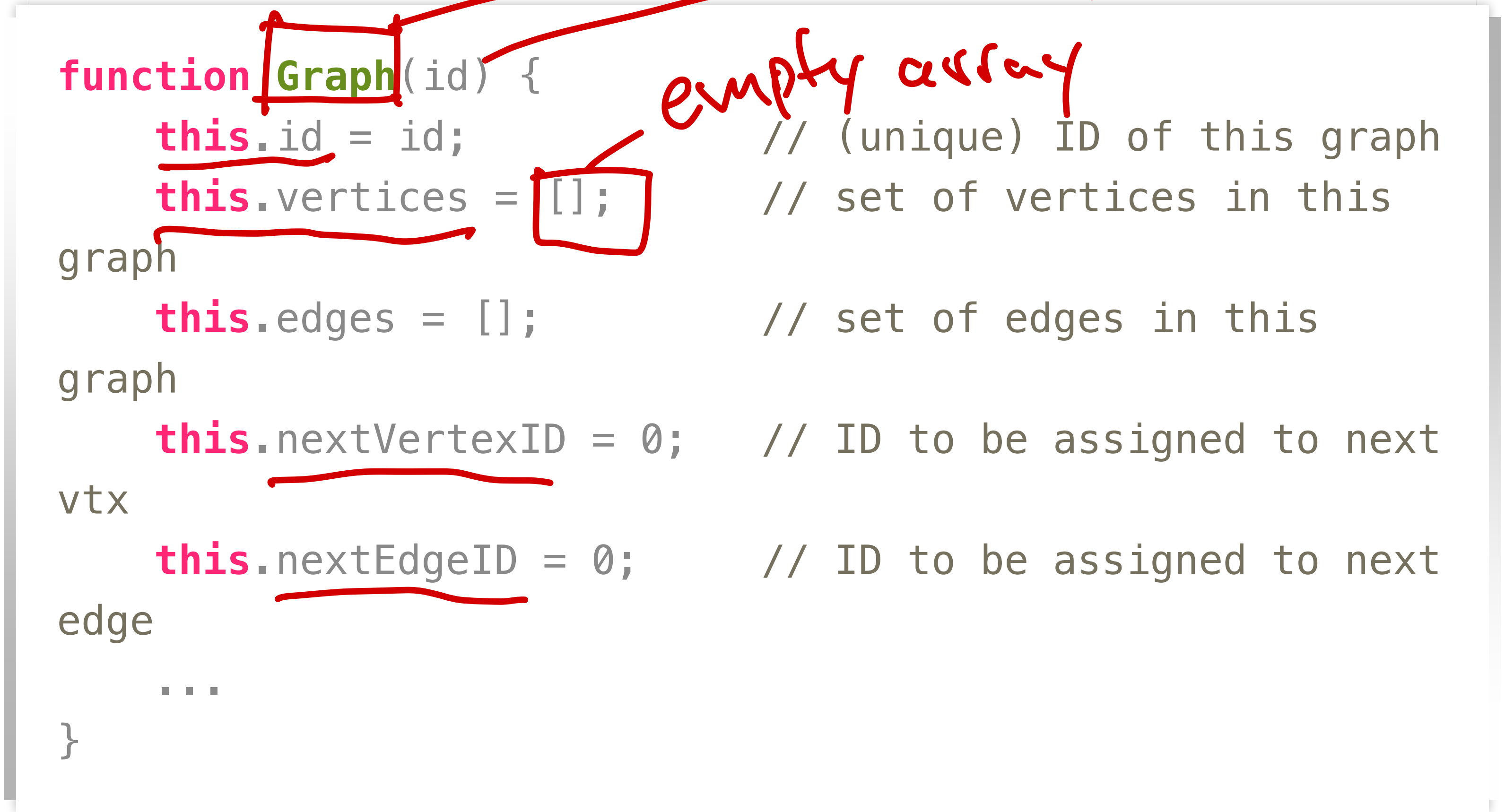
- `Graph`
  - stores sets of vertices, edges
- `Vertex`
  - stores ID, list of neighbors
- `Edge`
  - stores pair of endpoints

Graph in JavaScript

constructor name

argument

empty array



```
function Graph(id) {
    this.id = id;               // (unique) ID of this graph
    this.vertices = [];         // set of vertices in this
graph
    this.edges = [];            // set of edges in this
graph
    this.nextVertexID = 0;      // ID to be assigned to next
vtx
    this.nextEdgeID = 0;        // ID to be assigned to next
edge
    ...
}
```

Notes on JavaScript Arrays

- no specified datatypes
- self resizing
- support stack operations
  - `push(elt)` appends `elt` to end
  - `pop()` removes and returns last element
- *associative arrays* indices need not be numbers!

```
const a = [];          // make an array
a.push(1);
a.push(2);
a["name"] = "Alice";
let guess = a.pop(); // what does this do?
```

Graph Interactions

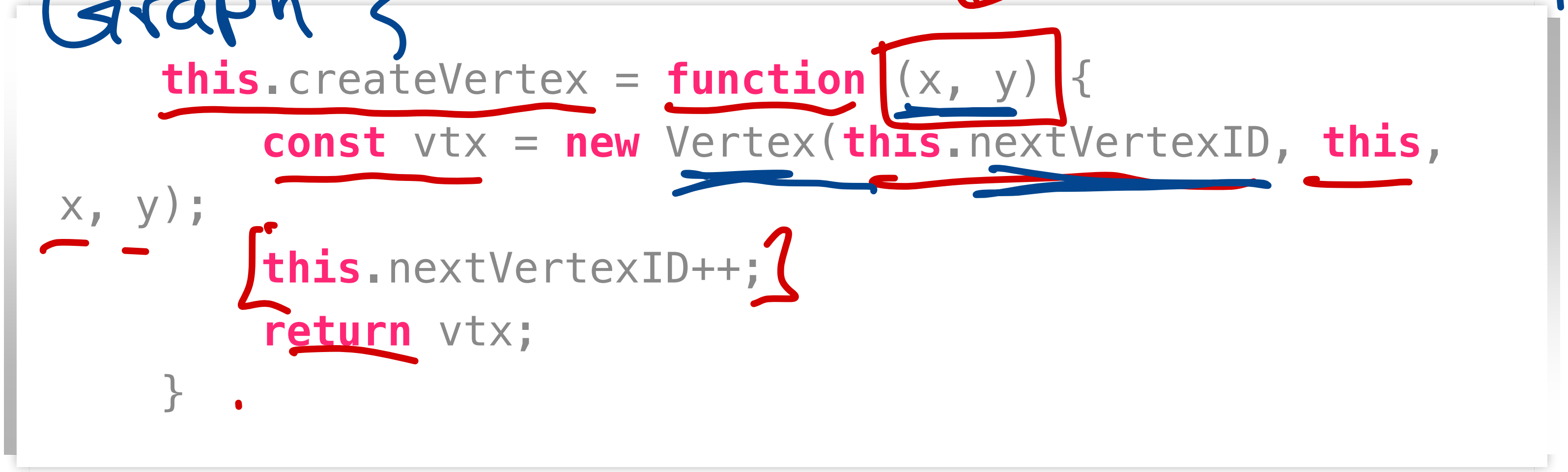
- add (remove?) vertices
- add (remove?) edges

Create/Add Vertices

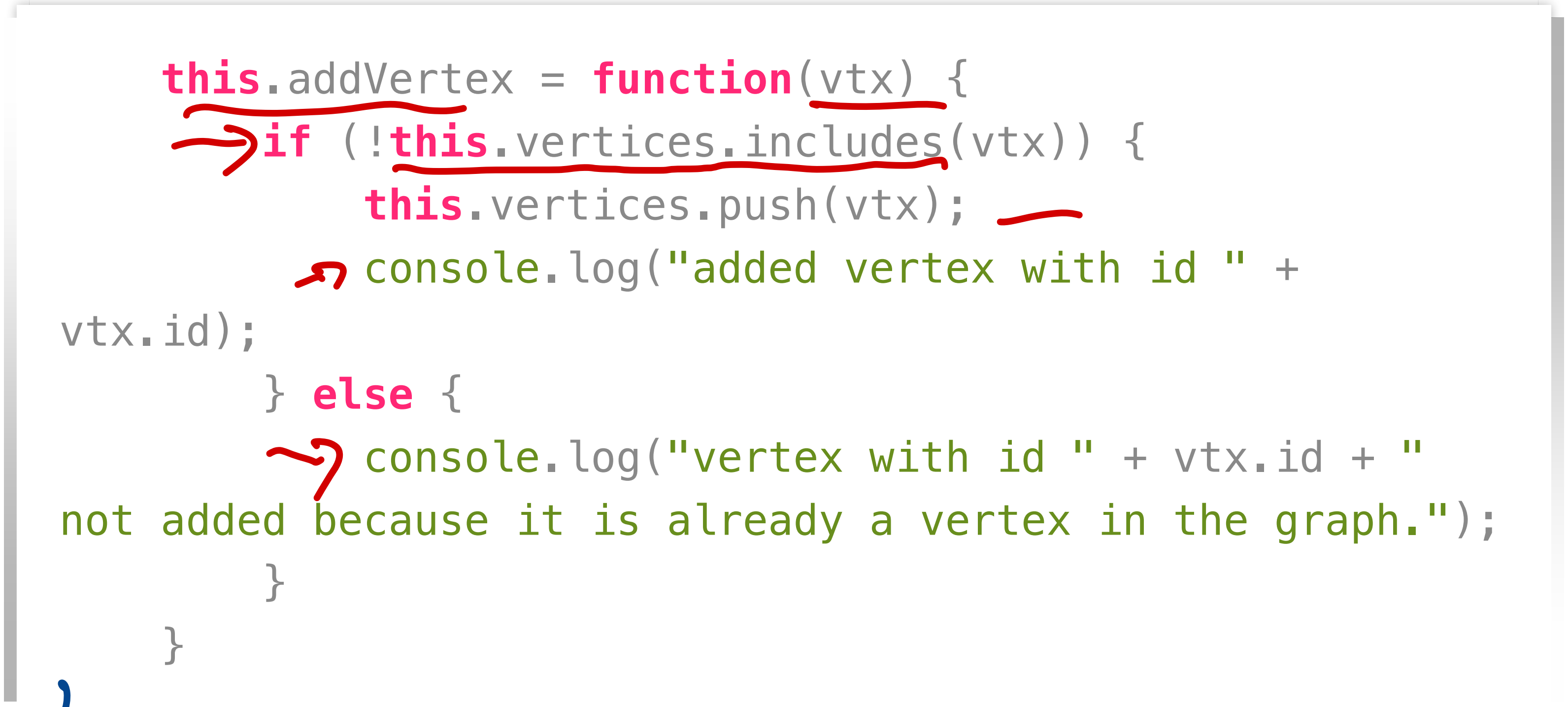
arguments

let g = new Graph

Graph {



```
    this.createVertex = function (x, y) {
        const vtx = new Vertex(this.nextVertexID, this,
x, y);
        this.nextVertexID++;
        return vtx;
    }
```



```
    this.addVertex = function(vtx) {
        if (!this.vertices.includes(vtx)) {
            this.vertices.push(vtx);
            console.log("added vertex with id " +
vtx.id);
        } else {
            console.log("vertex with id " + vtx.id + "
not added because it is already a vertex in the graph.");
        }
    }
```

}

Building Graphs Interactively

`GraphVisualizer` object

```
function GraphVisualizer (graph, svg, text) {
    this.graph = graph;         // the graph we are
visualizing
    this.svg = svg;             // the svg element we are
drawing on
    this.text = text;           // a text box

    ...
}
```

`GraphVisualizer`'s Role

`Graph` specifies *structure*
`GraphVisualizer` mediates *interactions* between user and `Graph`

- visualization/display
- interaction

Encapsulation:

- `Graph` does **not** reference any display attributes
- `GraphVisualizer` handles all
  - display (e.g., DOM elements)
  - interactions (e.g. clicks)
  - styling

`GraphVisualizer` behaviors

1. Respond to clicks
    - click to empty space adds a vertex
        - create/style DOM element, add to SVG image
        - create a `Vertex` and add to `Graph`
    - click to first vertex
        - highlights vertex
    - click to next vertex
        - adds `Edge` between `Vertexs` in `Graph`
        - draws line between corresponding vertices
2. Other visual modifications
    - highlight/mute vertices/edges

Graph Builder Demo

Future Work

- “import” an existing graph
- automated graph drawing
  - given just vertices/edges of a graph, determine how graph should be displayed
  - this is a major challenge!

Graph Search

Input

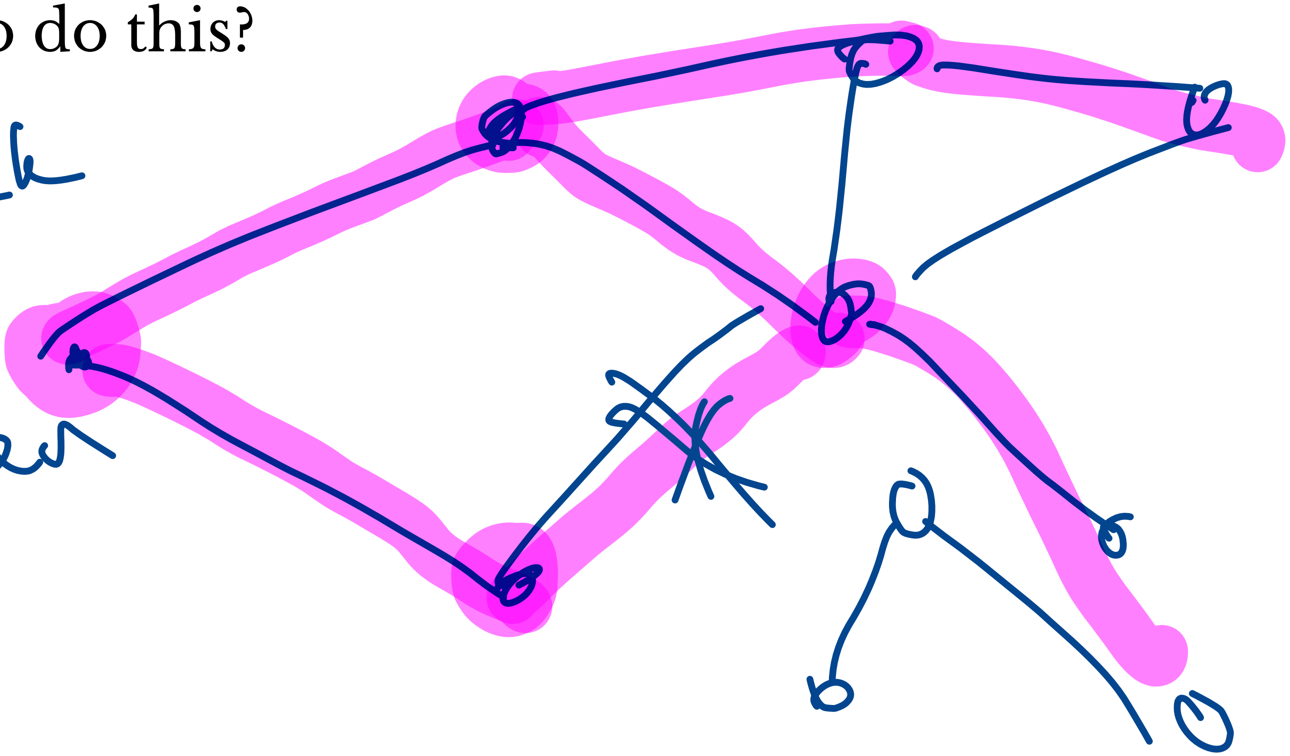
- `Graph` (adjacency list representation)
- starting `Vertex v`

Output

- Set of vertices reachable from v

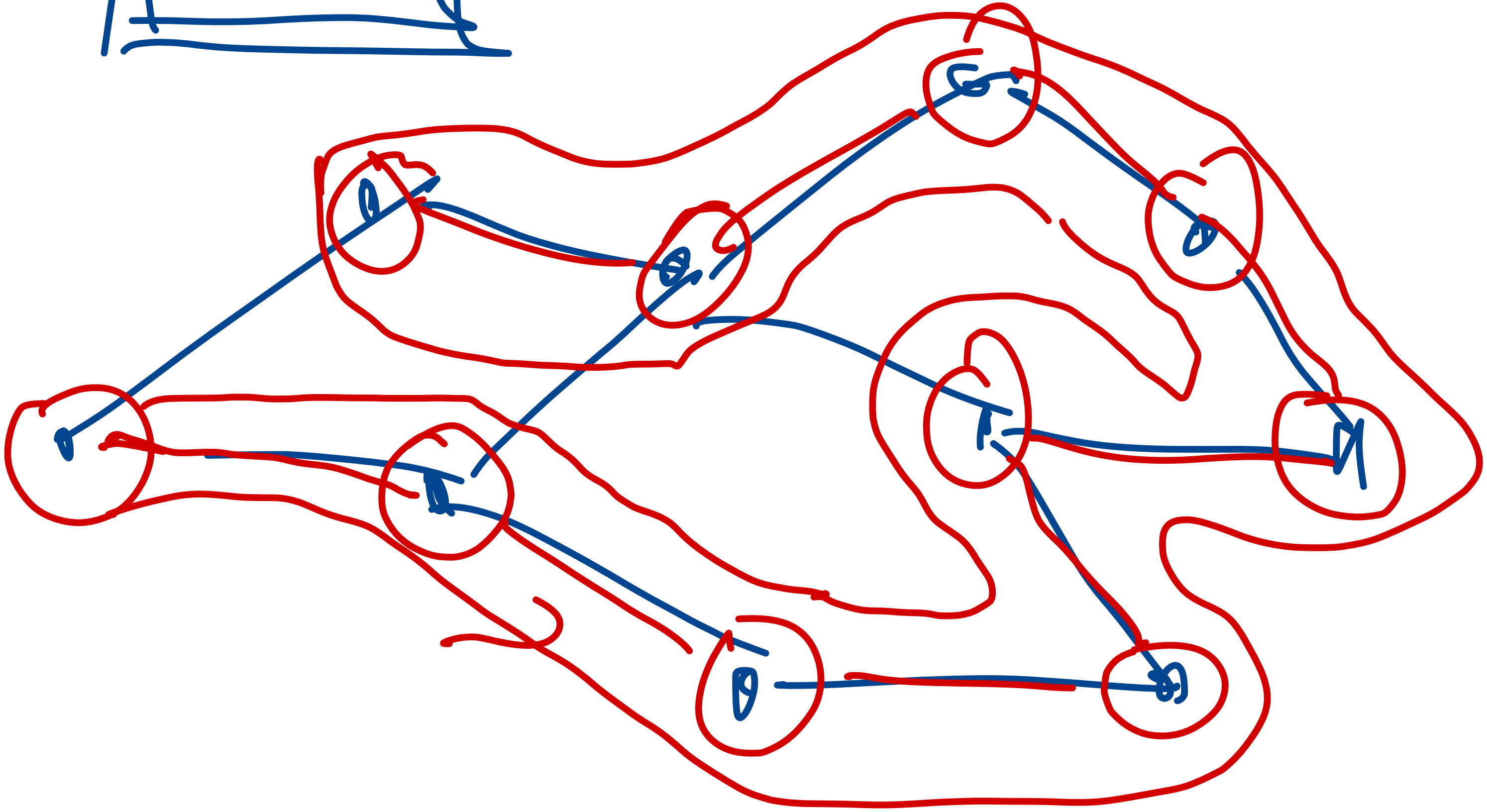
Question How to do this?

keep track of who's been visited



Depth-first Strategy

1. Start at starting vertex
2. Until stuck at starting vertex:
   - look for an unvisited neighbor
   - if found, move to unvisited neighbor
   - otherwise backtrack to vertex w/ unvisited neighbor



Implementing DFS

What do we need to keep track of throughout execution?

- visited nodes : set
- see neighbors
- stack of "active" vertices

↓

non- exhausted

DFS Pseudo-code

```
visited = [start]; // set
active = [start];  // stack

while active is not empty
   cur = top of active
   if cur has unvisited neighbor v
     push v to active
     add v to visited
   else
     pop cur off active ← backtrack
```

Visualizing DFS

What should we show user? How to illustrate behavior?

- color coding vertices by active/visited/cur/unvisited

  ⟶ add text rep too

- step button
  - each cur update

- arrows

Implementing DFS in JavaScript

1. Define a `Dfs` object type
   - what should it store?

2. Implement DFS procedure as **steps**
   - start procedure
   - individual actions to be visualized

   Question. What should count as a single step?

DFS Demo

Design Notes

`Dfs` stores

- `Graph` to explore
- `GraphVisualizer` to update
- local info for algorithm execution

`Dfs` tells `GraphVisualizer` what to highlight/mute, etc

- `GraphVisualizer` decides how to update display in response

Lab 06

Algorithm Visualization: Convex Hulls

Convex Hull Problem

Input:

- set of points in plane
  - (x, y)-coordinates of each point

Output:

- a sequence of points $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ that define the “boundary” of the set of points
  - path around $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ surrounds all points
  - the bounded region is **convex**

Which Points are on the convex hull?

Next Week

Algorithms for finding the convex hull!
Your Task (Assignment 06):

- implement a convex hull algorithm
- create an interactive visualization for the algorithm