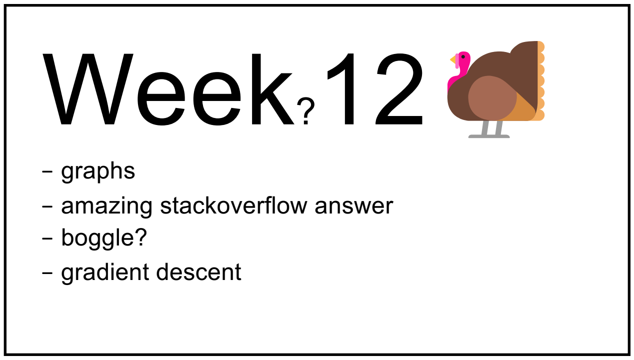
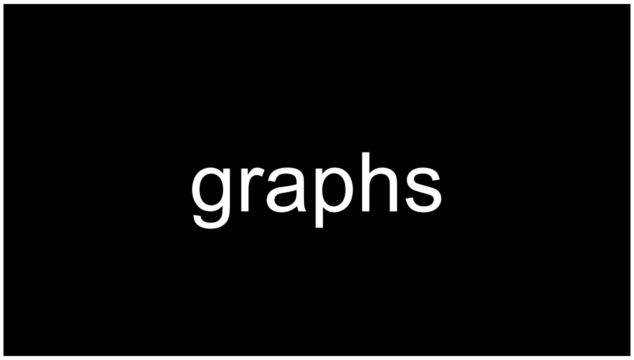
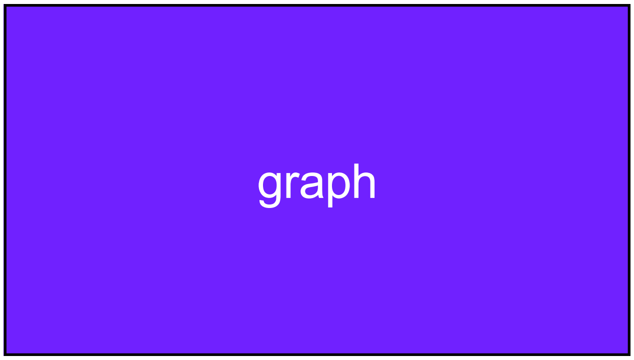


Week?12 🦃

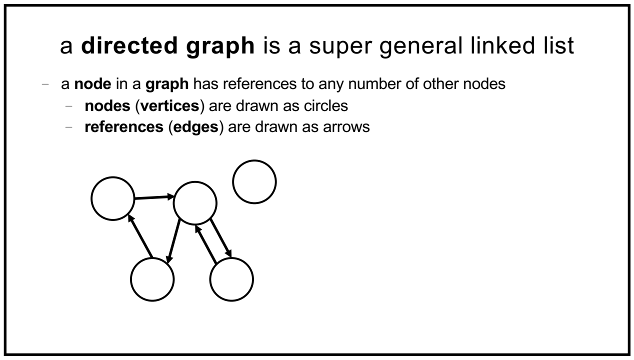
- graphs
- amazing stackoverflow answer
- boggle?
- gradient descent



graphs

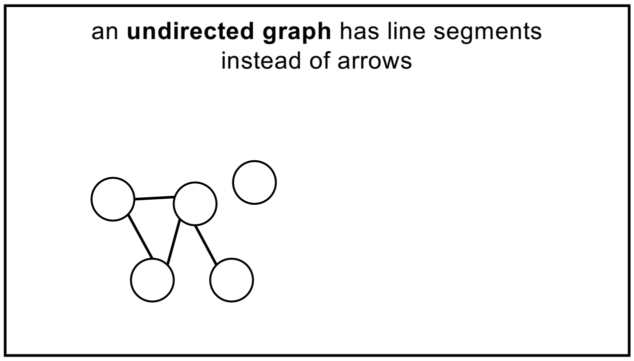


graph

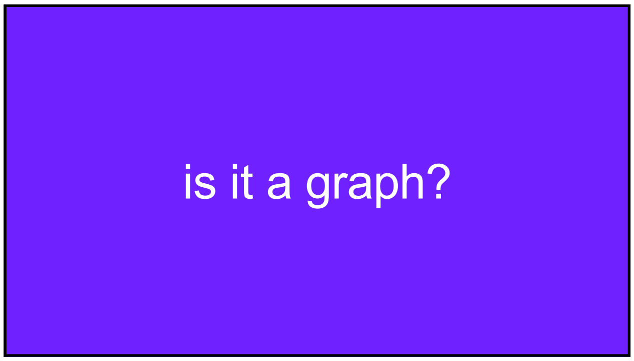


a **directed graph** is a super general linked list

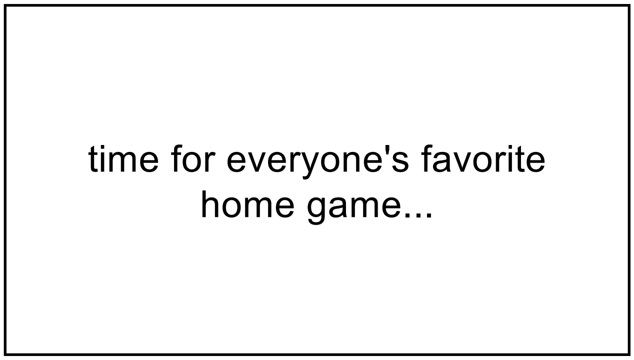
- a **node** in a **graph** has references to any number of other nodes
  - **nodes** (**vertices**) are drawn as circles
  - **references** (**edges**) are drawn as arrows



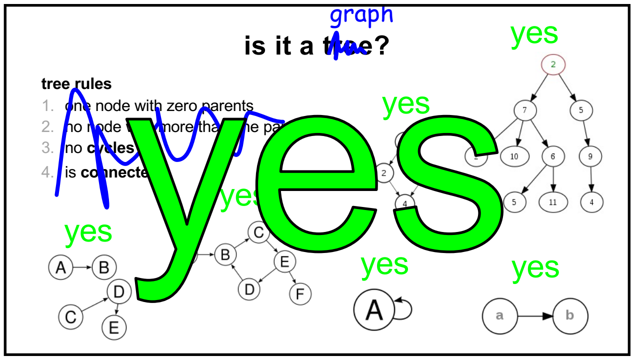
an **undirected graph** has line segments instead of arrows



is it a graph?



time for everyone's favorite home game...



is it a ~~tree~~ graph?

tree rules

1. one node with zero parents
2. no node w more tha ne pa
3. no **cycles**
4. is **co nnecte**

yes

yes

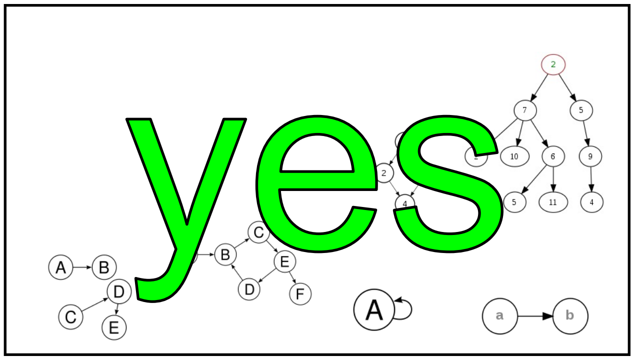
yes

yes

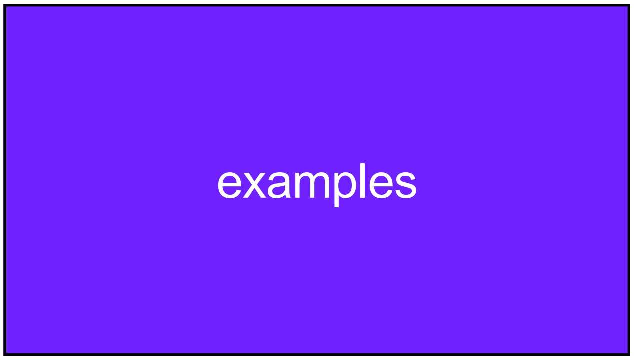
yes

yes

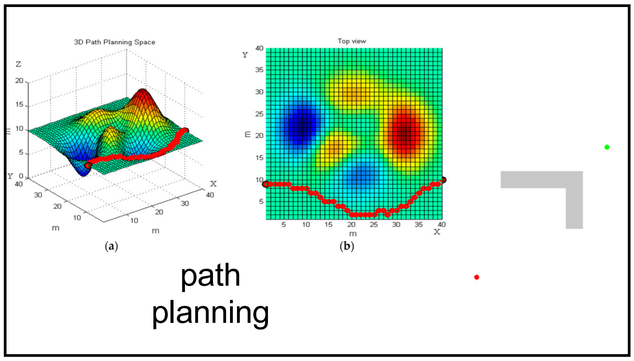
yes



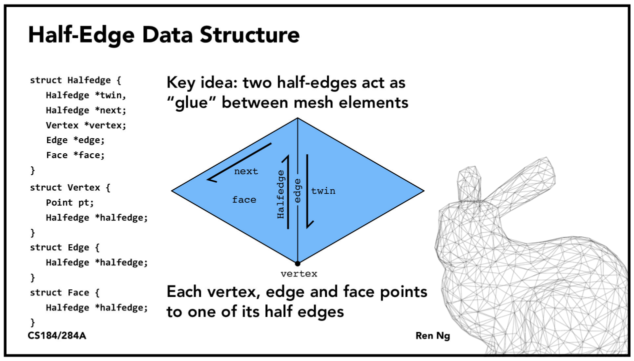
yes



examples



path planning



Half-Edge Data Structure

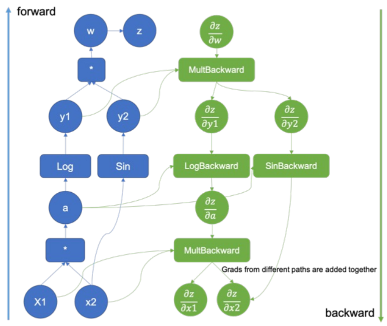
```
struct Halfedge {
   Halfedge *twin,
   Halfedge *next;
   Vertex *vertex;
   Edge *edge;
   Face *face;
}
struct Vertex {
   Point pt;
   Halfedge *halfedge;
}
struct Edge {
   Halfedge *halfedge;
}
struct Face {
   Halfedge *halfedge;
}
```

Key idea: two half-edges act as "glue" between mesh elements

Each vertex, edge and face points to one of its half edges

CS184/284A

Ren Ng


forward
w
z
*
y1
y2
Log
Sin
a
*
X1
x2
$\frac{\partial z}{\partial w}$
MultBackward
$\frac{\partial z}{\partial y1}$
$\frac{\partial z}{\partial y2}$
LogBackward
SinBackward
$\frac{\partial z}{\partial a}$
MultBackward
Grads from different paths are added together
$\frac{\partial z}{\partial x1}$
$\frac{\partial z}{\partial x2}$
backward

computational graph
(to take derivatives of deep nets)

PyTorch

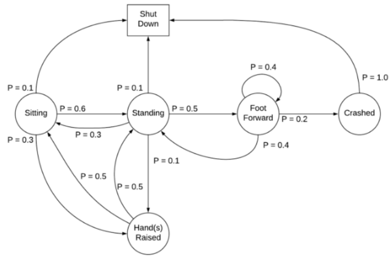
THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

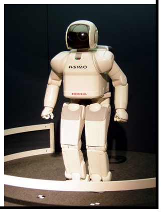
Yeah that's pretty much right
--Mark

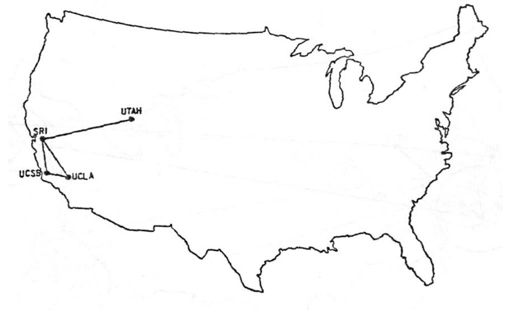
Is this an accurate quote?
--Jim

Yeah, sure
--Mark

Markov Decision Process

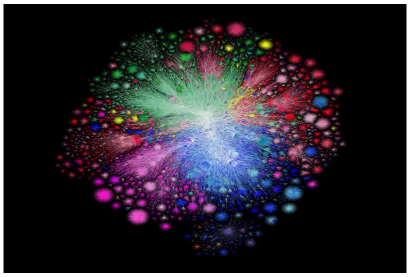




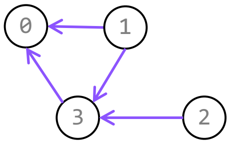


The ARPANET in December 1969

Present day, from Opte The Internet: 1997 - 2021.
VIDEO: BARRETT LYON/THE OPTE PROJECT



directed graph representations
(how do i store The Data?)

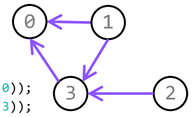


Object-Oriented list of nodes

```
class Graph {
    ArrayList<Node> nodes;
    Graph() { ... }
}
```

```
class Node {
    ArrayList<Node> neighbors;
    Node() { ... }
}
```

```
Graph graph = new Graph();
graph.nodes.add(new Node());
graph.nodes.add(new Node());
graph.nodes.add(new Node());
graph.nodes.add(new Node());
graph.nodes.get(1).neighbors.add(graph.nodes.get(0));
graph.nodes.get(1).neighbors.add(graph.nodes.get(3));
graph.nodes.get(2).neighbors.add(graph.nodes.get(3));
graph.nodes.get(3).neighbors.add(graph.nodes.get(0));
```

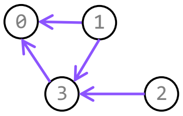


Object-Oriented list of nodes

```
class Graph {
    ArrayList<Node> nodes;
    Graph() { ... }
}
```

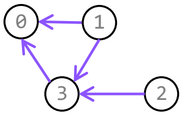
```
class Node {
    ArrayList<Node> neighbors;
    Node() { ... }
}
```

```
Graph graph = new Graph();
graph.addNode();
graph.addNode();
graph.addNode();
graph.addNode();
graph.addEdge(1, 0);
graph.addEdge(1, 3);
graph.addEdge(2, 3);
graph.addEdge(3, 0);
```



list of lists

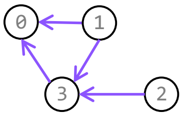
```
ArrayList<ArrayList<Integer>> graph = new ArrayList<>();
graph.add(new ArrayList<>());
graph.add(new ArrayList<>());
graph.add(new ArrayList<>());
graph.add(new ArrayList<>());
graph.get(1).add(0);
graph.get(1).add(3);
graph.get(2).add(3);
graph.get(3).add(0);
```



list of edges

```
class Edge {
    int i;
    int j;
    Edge(int i, int j) { ... }
}
```

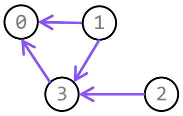
```
int numNodes = 4;
ArrayList<Edge> graph = new ArrayList<>();
graph.add(new Edge(1, 0));
graph.add(new Edge(1, 3));
graph.add(new Edge(2, 3));
graph.add(new Edge(3, 0));
```



math: adjacency matrix

- an adjacency matrix is a square matrix used to represent a graph
  - a graph with n nodes has corresponding $n{\times}n$ adjacency matrix $\boldsymbol{G}$
  - $G_{i,j} = \begin{cases} 1 & \text{if there is an edge from node i} \rightarrow \text{node j} \\ 0 & \text{otherwise} \end{cases}$

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 0 \end{array}$$

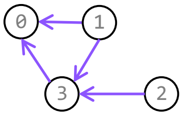


  - note: $G_{i,i}$ = is a "self edge"

dense matrix at 2D array

```java
int[][] graph = new int[4][4];
graph[1][0] = 1;
graph[1][3] = 1;
graph[2][3] = 1;
graph[3][0] = 1;
```

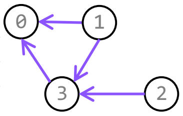
$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 0 \end{array}$$



dense matrix as 1D array

```java
int[] graph = new int[4 * 4];
graph[4 * 1 + 0] = 1;
graph[4 * 1 + 3] = 1;
graph[4 * 2 + 3] = 1;
graph[4 * 3 + 0] = 1;
```

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 0 \end{array}$$



0	0	0	0	1	0	0	1	0	0	0	1	1	0	0	0
(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(1,3)	(2,0)	(2,1)	(2,2)	(2,3)	(3,0)	(3,1)	(3,2)	(3,3)

sparse matrix

```java
class Entry {
    int row;
    int col;
    int val;
    Entry(...) { ... }
}
```

```java
ArrayList<Entry> graph = new ArrayList<>();
graph.add(new Entry(1, 0, 1));
graph.add(new Entry(1, 3, 1));
graph.add(new Entry(2, 3, 1));
graph.add(new Entry(3, 0, 1));
```

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 0 \end{array}$$