# CS 225

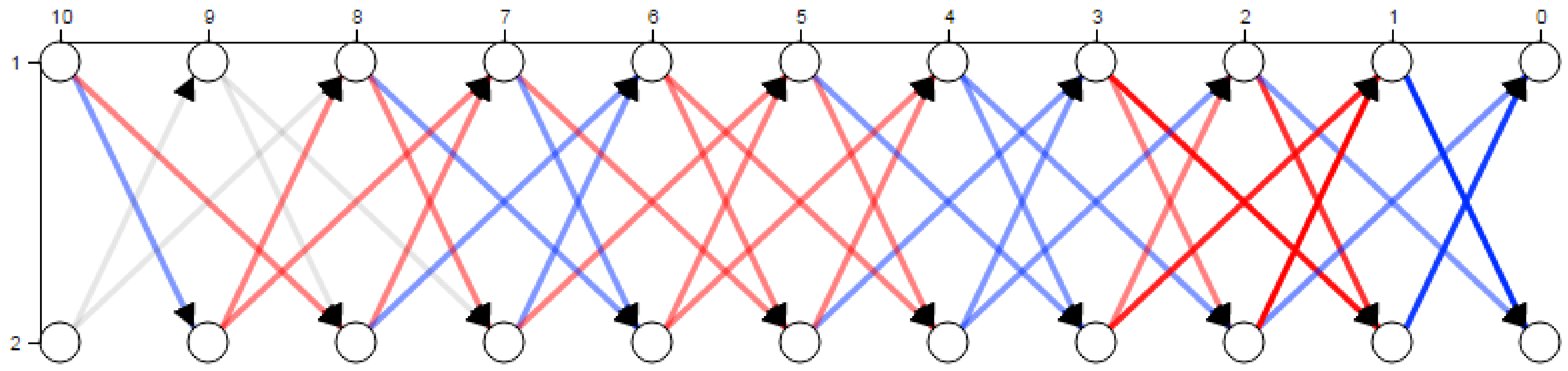
**Data Structures**

## *Dec. 11 – Floyd-Warshall's Algorithm*

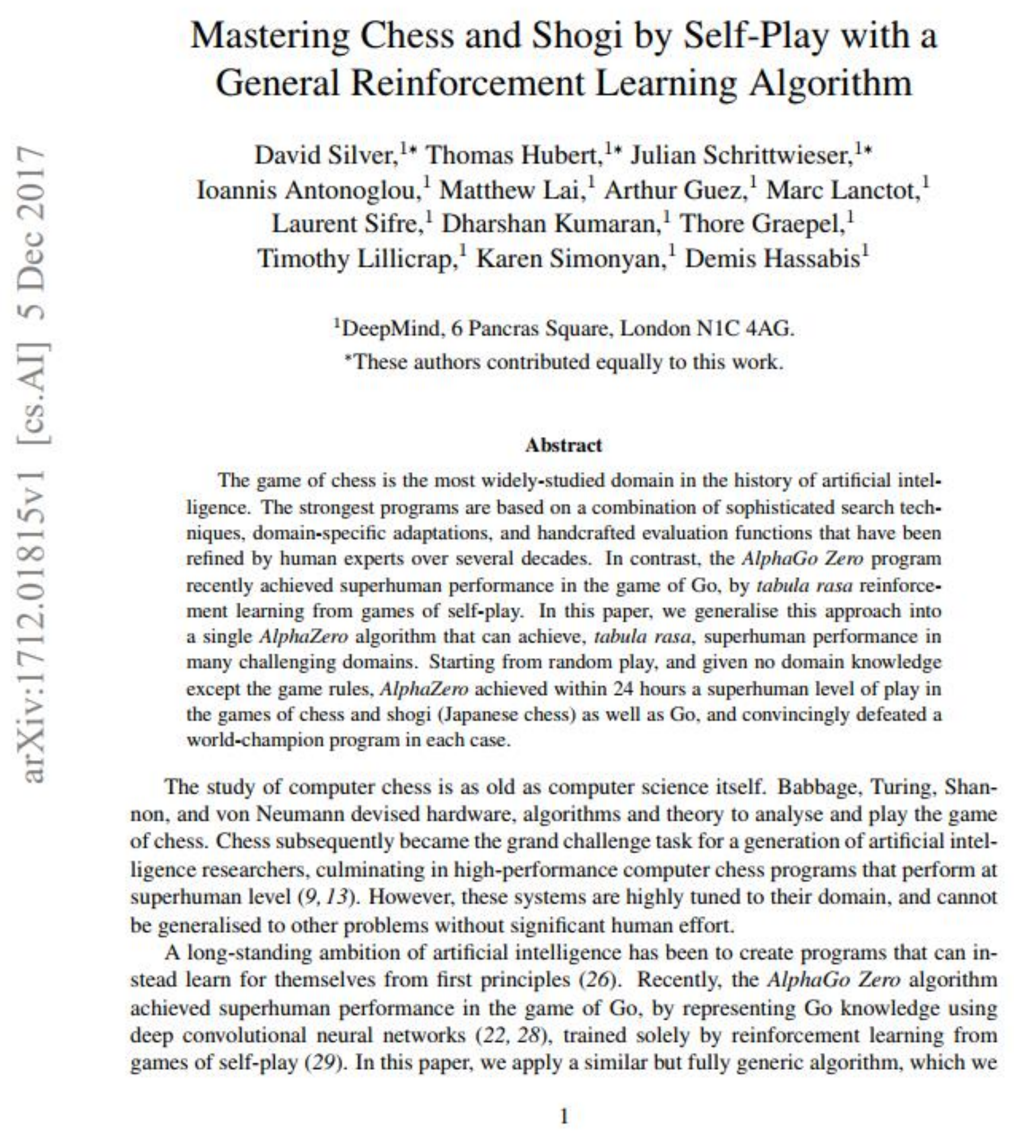
*Wade Fagen-Ulmschneider*

# Reinforcement Learning

| Available Tokens | Learned Move |
|---|---|
| 10 | Take 1 token ➔ 9 |
| 9 | Take 2 tokens ➔ 7 |
| 8 | Take 2 tokens ➔ 6 |
| 7 | Take 1 token ➔ 6 |
| 6 | Take 1 token ➔ 5 |
| 5 | Take 2 tokens ➔ 3 |
| 4 | Take 1 token ➔ 3 |
| 3 | Take 1 token ➔ 2 |
| 2 | Take 2 tokens ➔ 0 (win) |
| 1 | Take 1 token ➔ 0 (win) |



# Reinforcement Learning



arXiv:1712.01815v1 [cs.AI] 5 Dec 2017

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver,[1*] Thomas Hubert,[1*] Julian Schrittwieser,[1*]
Ioannis Antonoglou,[1] Matthew Lai,[1] Arthur Guez,[1] Marc Lanctot,[1]
Laurent Sifre,[1] Dharshan Kumaran,[1] Thore Graepel,[1]
Timothy Lillicrap,[1] Karen Simonyan,[1] Demis Hassabis[1]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.
*These authors contributed equally to this work.

**Abstract**

The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the *AlphaGo Zero* program recently achieved superhuman performance in the game of Go, by *tabula rasa* reinforcement learning from games of self-play. In this paper, we generalise this approach into a single *AlphaZero* algorithm that can achieve, *tabula rasa*, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, *AlphaZero* achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

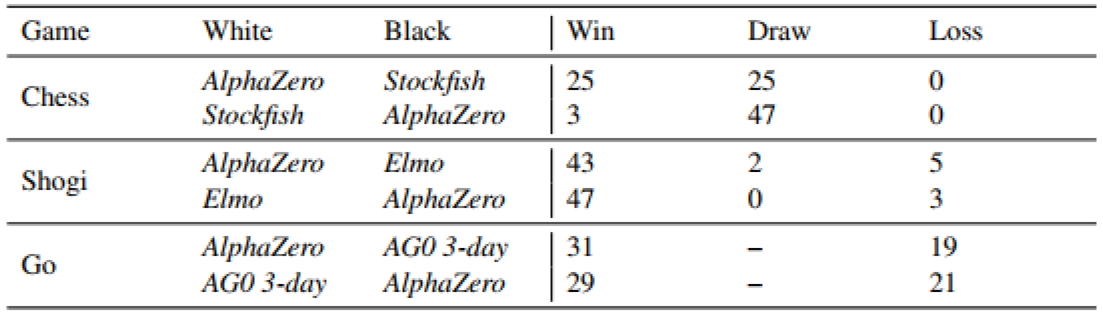
The study of computer chess is as old as computer science itself. Babbage, Turing, Shannon, and von Neumann devised hardware, algorithms and theory to analyse and play the game of chess. Chess subsequently became the grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that perform at superhuman level (*9, 13*). However, these systems are highly tuned to their domain, and cannot be generalised to other problems without significant human effort.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*26*). Recently, the *AlphaGo Zero* algorithm achieved superhuman performance in the game of Go, by representing Go knowledge using deep convolutional neural networks (*22, 28*), trained solely by reinforcement learning from games of self-play (*29*). In this paper, we apply a similar but fully generic algorithm, which we

1

Last week, Google's DeepMind AI team released a new research paper:

- Using reinforcement learning, an algorithm knowing only the rules of chess trained for 4 hours.
- After training, it destroyed the best chess program (Stockfish):



| Game | White | Black | Win | Draw | Loss |
|---|---|---|---|---|---|
| Chess | *AlphaZero* | *Stockfish* | 25 | 25 | 0 |
| | *Stockfish* | *AlphaZero* | 3 | 47 | 0 |
| Shogi | *AlphaZero* | *Elmo* | 43 | 2 | 5 |
| | *Elmo* | *AlphaZero* | 47 | 0 | 3 |
| Go | *AlphaZero* | *AG0 3-day* | 31 | – | 19 |
| | *AG0 3-day* | *AlphaZero* | 29 | – | 21 |

Table 1: Tournament evaluation of *AlphaZero* in chess, shogi, and Go, as games won, drawn or lost from *AlphaZero*'s perspective, in 100 game matches against *Stockfish*, *Elmo*, and the previously published *AlphaGo Zero* after 3 days of training. Each program was given 1 minute of thinking time per move.

**"Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm",** https://arxiv.org/abs/1712.01815

## Final Exam Information

**Multiple Choice:**

- 22 total multiple choice questions
    - 8 questions on graphs
    - 14 questions on pre-graph conten
    - No questions specifically about C++, pointers, etc

**Programming:**

- One “easy” question
    - Heaps, hash tables, disjoint sets, tree encoding, etc. are fair game
- One “hard” question
    - Graph algorithm: be able to implement Prim, Kruskal, Dijkstra, BFS, DFS, etc
- We will likely not tell you which algorithm to use!
- We will post the .h files on Wednesday.

## End of Semester Logistics

**Regrades on Exams:**

- Most of these have been posted.
- Any corrections needed, send Mattox an email, *not* Piazza.

## Next Semester (and every Spring!)

**CS 421: “Programming Languages”**

- Learn what goes into a language!
- Be able to write an interpreter for the language of your choice!
- Learn functional programming in Haskell!

## End of Semester Logistics

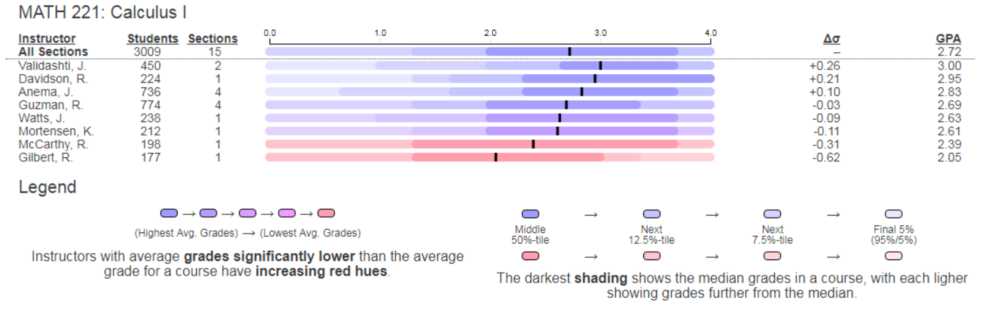
**Regrades on MPs/Labs:**

- Regrades are being processed today/tomorrow.
- I will make a Piazza update once grade updates are complete; will follow-up via Piazza.

http://waf.cs.illinois.edu/discovery/

# My Passion: Data Discovery

## GPAs at Illinois:



MATH 221: Calculus I

| Instructor | Students | Sections | Δσ | GPA |
|---|---|---|---|---|
| All Sections | 3009 | 15 | – | 2.72 |
| Validashti, J. | 450 | 2 | +0.26 | 3.00 |
| Davidson, R. | 224 | 1 | +0.21 | 2.95 |
| Anema, J. | 736 | 4 | +0.10 | 2.83 |
| Guzman, R. | 774 | 4 | -0.03 | 2.69 |
| Watts, J. | 238 | 1 | -0.09 | 2.63 |
| Mortensen, K. | 212 | 1 | -0.11 | 2.61 |
| McCarthy, R. | 198 | 1 | -0.31 | 2.39 |
| Gilbert, R. | 177 | 1 | -0.62 | 2.05 |

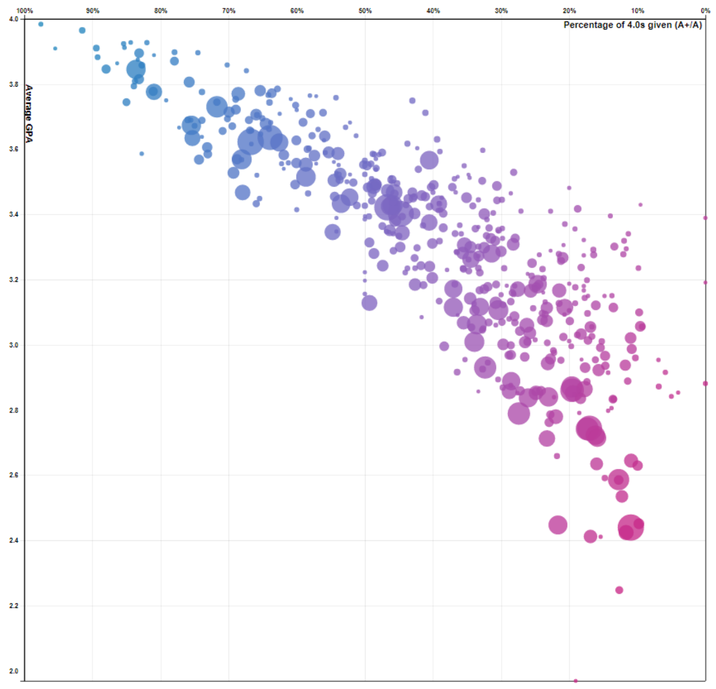
Legend

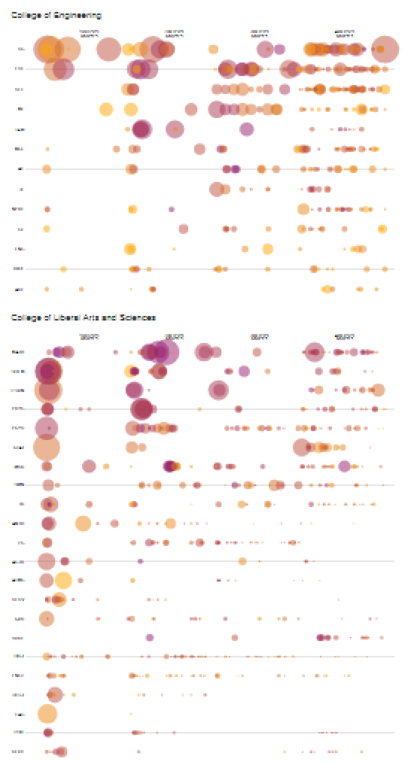
(Highest Avg. Grades) ⟶ (Lowest Avg. Grades)

Instructors with average **grades significantly lower** than the average grade for a course have **increasing red hues**.

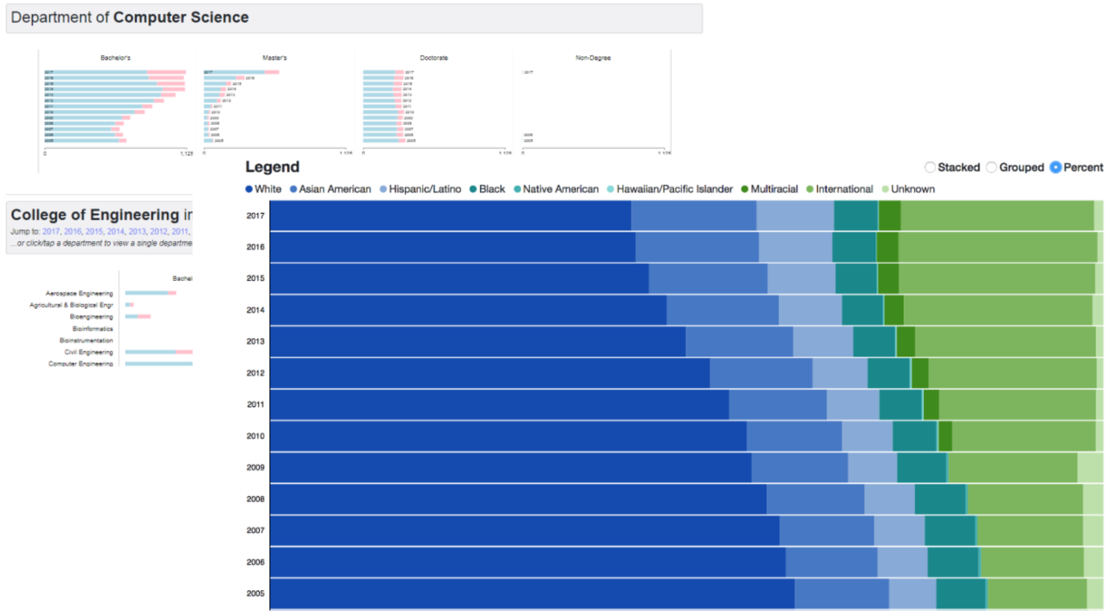
Middle 50%-tile → Next 12.5%-tile → Next 7.5%-tile → Final 5% (95%/5%)

The darkest **shading** shows the median grades in a course, with each ligher showing grades further from the median.

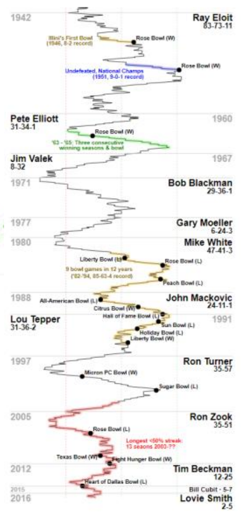




## Diversity at Illinois:



## And others:



## CS 305: Data Driven Discovery (Fall 2018)

- Non-majors (no CS, no ECE)
  *(Sorry, not my decision! Department feels data visualization in Python is too simple for CS credit.)*
  - *Benefit: Everyone is nearly on the same playing field – passion of data with core programming tools*

- Next offering: Fall 2018!

## Floyd-Warshall Algorithm

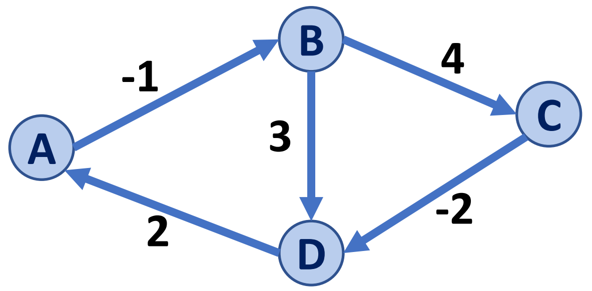
Floyd-Warshall's Algorithm is an alterative to Dijkstra in the presence of negative-weight edges (not negative weight cycles).

```
FloydWarshall(G):
  Let d be a adj. matrix initialized to +inf
  foreach (Vertex v : G):
    d[v][v] = 0
  foreach (Edge (u, v) : G):
    d[u][v] = cost(u, v)

  foreach (Vertex u : G):
    foreach (Vertex v : G):
      foreach (Vertex w : G):
        if d[u, v] > d[u, w] + d[w, v]:
          d[u, v] = d[u, w] + d[w, v]
```

# Floyd-Warshall Algorithm

```
FloydWarshall(G):
  Let d be a adj. matrix initialized to +inf
  foreach (Vertex v : G):
    d[v][v] = 0
  foreach (Edge (u, v) : G):
    d[u][v] = cost(u, v)

  foreach (Vertex u : G):
    foreach (Vertex v : G):
      foreach (Vertex w : G):
        if d[u, v] > d[u, w] + d[w, v]:
          d[u, v] = d[u, w] + d[w, v]
```

|   | A | B | C | D |
|---|---|---|---|---|
| A |   |   |   |   |
| B |   |   |   |   |
| C |   |   |   |   |
| D |   |   |   |   |

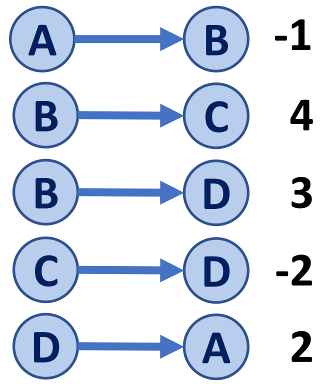


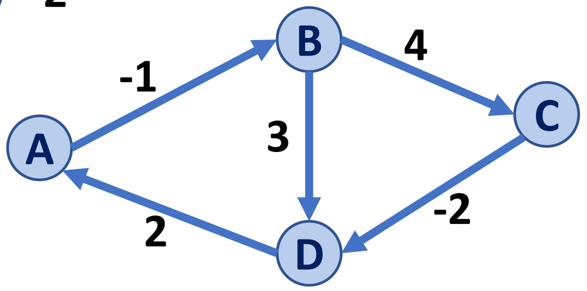
# Floyd-Warshall Algorithm

```
foreach (Vertex u : G):
  foreach (Vertex v : G):
    foreach (Vertex w : G):
      if d[u, v] > d[u, w] + d[w, v]:
        d[u, v] = d[u, w] + d[w, v]
```

|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | -1 |  |  |
| B |  | 0 | 4 | 3 |
| C |  |  | 0 | -2 |
| D | 2 |  |  | 0 |

**Initially:**



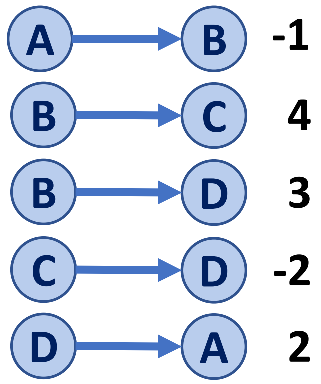


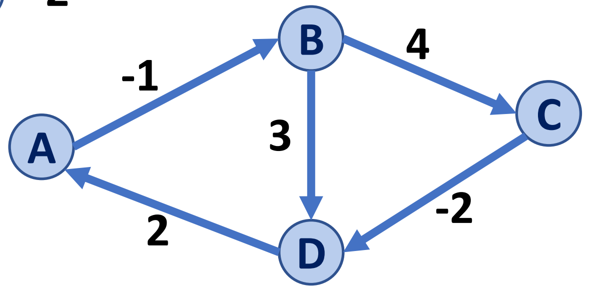
# Floyd-Warshall Algorithm

|   | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | -1 |   |   |
| **B** |   | 0 | 4 | 3 |
| **C** |   |   | 0 | -2 |
| **D** | 2 |   |   | 0 |

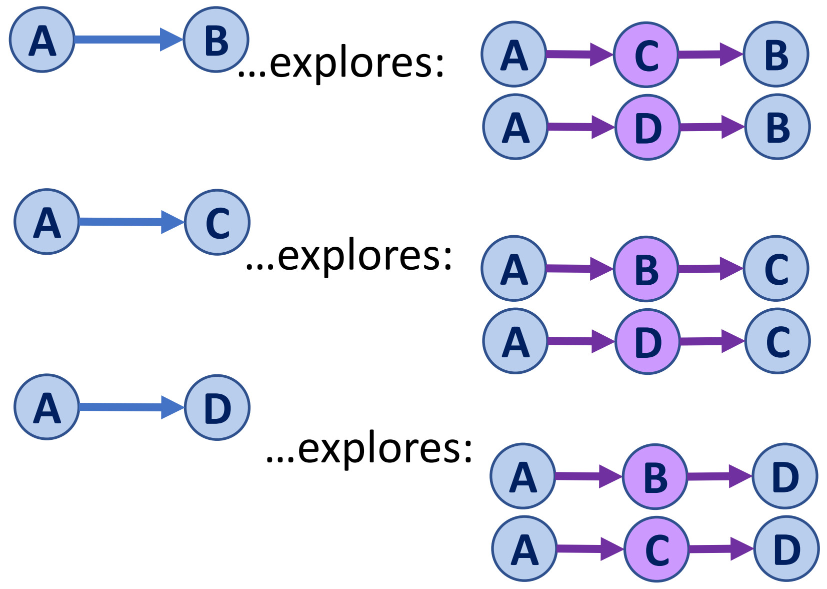
```
  foreach (Vertex u : G):
    foreach (Vertex v : G):
      foreach (Vertex w : G):
        if d[u, v] > d[u, w] + d[w, v]:
          d[u, v] = d[u, w] + d[w, v]
```

**Initially:**





**Let u = A; v and w explores for better paths:**



# Floyd-Warshall Algorithm

```
foreach (Vertex u : G):
  foreach (Vertex v : G):
    foreach (Vertex w : G):
      if d[u, v] > d[u, w] + d[w, v]:
        d[u, v] = d[u, w] + d[w, v]
```

|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | -1 | 2 | 1 |
| B |  | 0 | 4 | 3 |
| C |  |  | 0 | -2 |
| D | 2 |  |  | 0 |

**Initially:**

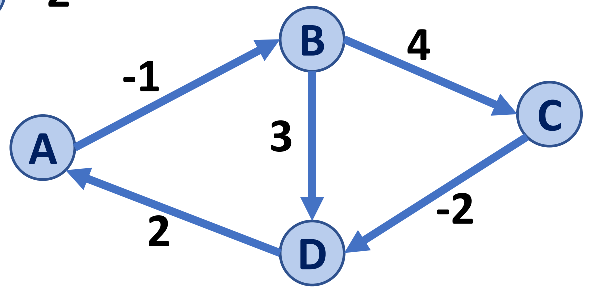
$$\begin{tikzcd} A \arrow[r, dotted] & B & -1 \end{tikzcd}$$

$$\begin{tikzcd} B \arrow[r, dotted] & C & 4 \end{tikzcd}$$

$$\begin{tikzcd} B \arrow[r, dotted] & D & 3 \end{tikzcd}$$

$$\begin{tikzcd} C \arrow[r, dotted] & D & -2 \end{tikzcd}$$

$$\begin{tikzcd} D \arrow[r, dotted] & A & 2 \end{tikzcd}$$



**Let u = A; v and w explores for better paths:**

$$\begin{tikzcd} A \arrow[r, dotted] & B \end{tikzcd}$$ …explores:

$$\begin{tikzcd} A \arrow[r] & C \arrow[r] & B \end{tikzcd}$$ $+\infty$

$$\begin{tikzcd} A \arrow[r] & D \arrow[r] & B \end{tikzcd}$$ $+\infty$

$$\begin{tikzcd} A \arrow[r, dotted] & C \end{tikzcd}$$ …explores:

$$\begin{tikzcd} A \arrow[r] & B \arrow[r] & C \end{tikzcd}$$ $-1 + 3 = \mathbf{2}$ UPDATE!

$$\begin{tikzcd} A \arrow[r] & D \arrow[r] & C \end{tikzcd}$$ $+\infty$

$$\begin{tikzcd} A \arrow[r, dotted] & D \end{tikzcd}$$ …explores:

$$\begin{tikzcd} A \arrow[r] & B \arrow[r] & D \end{tikzcd}$$ $-1 + 3 = \mathbf{2}$ UPDATE!

$$\begin{tikzcd} A \arrow[r] & C \arrow[r] & D \end{tikzcd}$$ $3 + (-2) = \mathbf{1}$ UPDATE!

# Floyd-Warshall Algorithm

```
foreach (Vertex u : G):
  foreach (Vertex v : G):
    foreach (Vertex w : G):
      if d[u, v] > d[u, w] + d[w, v]:
        d[u, v] = d[u, w] + d[w, v]
```

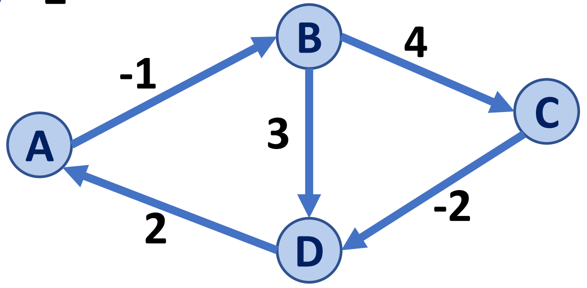
| | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | -1 | 2 | 1 |
| **B** | | 0 | 4 | 3 |
| **C** | | | 0 | -2 |
| **D** | 2 | | | 0 |

**Initially:**

$$\begin{tikzcd} A \arrow[r, dotted] & B & -1 \\ B \arrow[r, dotted] & C & 4 \\ B \arrow[r, dotted] & D & 3 \\ C \arrow[r, dotted] & D & -2 \\ D \arrow[r, dotted] & A & 2 \end{tikzcd}$$

**Let u = A; v and w explores for better paths:**

$$\begin{tikzcd} A \arrow[r, dotted] & C & 2 \\ A \arrow[r, dotted] & D & 1 \end{tikzcd}$$



# Floyd-Warshall Algorithm

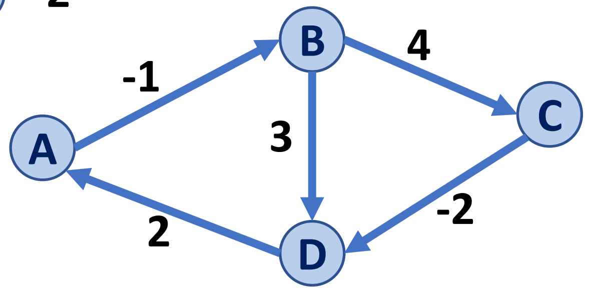
```
foreach (Vertex u : G):
  foreach (Vertex v : G):
    foreach (Vertex w : G):
      if d[u, v] > d[u, w] + d[w, v]:
        d[u, v] = d[u, w] + d[w, v]
```

|   | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | -1 | 2 | 1 |
| **B** | 5 | 0 | 4 | 2 |
| **C** |  |  | 0 | -2 |
| **D** | 2 |  |  | 0 |

**Initially:**

$$\begin{tikzcd}
A \arrow[r, dotted] & B & -1 \\
B \arrow[r, dotted] & C & 4 \\
B \arrow[r, dotted] & D & 3 \\
C \arrow[r, dotted] & D & -2 \\
D \arrow[r, dotted] & A & 2
\end{tikzcd}$$

$$\begin{tikzcd}
A \arrow[r, dotted] & C & 2 \\
A \arrow[r, dotted] & D & 1
\end{tikzcd}$$



**Let u = B; v and w explores for better paths:**

$$\begin{tikzcd}
B \arrow[r, dotted] & A
\end{tikzcd}$$

…explores:

$$\begin{tikzcd}
B \arrow[r] & C \arrow[r] & A \\
B \arrow[r] & D \arrow[r] & A
\end{tikzcd}$$

$$\begin{tikzcd}
B \arrow[r, dotted] & C
\end{tikzcd}$$

…explores:

$$\begin{tikzcd}
B \arrow[r] & A \arrow[r] & C \\
B \arrow[r] & D \arrow[r] & C
\end{tikzcd}$$

$$\begin{tikzcd}
B \arrow[r, dotted] & D
\end{tikzcd}$$

…explores:

$$\begin{tikzcd}
B \arrow[r] & A \arrow[r] & D \\
B \arrow[r] & C \arrow[r] & D
\end{tikzcd}$$

# Floyd-Warshall Algorithm

```
  foreach (Vertex u : G):
    foreach (Vertex v : G):
      foreach (Vertex w : G):
        if d[u, v] > d[u, w] + d[w, v]:
          d[u, v] = d[u, w] + d[w, v]
```

|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | -1 | 2 | 1 |
| B | 5 | 0 | 4 | 2 |
| C |   |   | 0 | -2 |
| D | 2 |   |   | 0 |

**Initially:**

$A \dashrightarrow B$ **-1**

$B \dashrightarrow C$ **4**

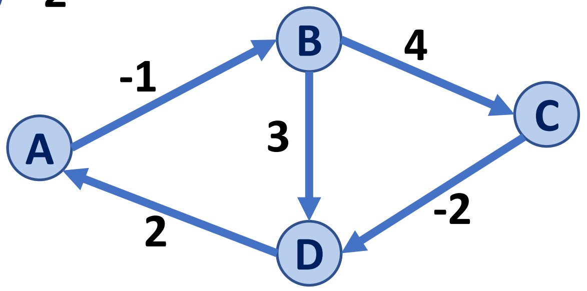
$B \dashrightarrow D$ **3**

$C \dashrightarrow D$ **-2**

$D \dashrightarrow A$ **2**

$A \dashrightarrow C$ **2**

$A \dashrightarrow D$ **1**



**Let u = B; v and w explores for better paths:**

$B \dashrightarrow A$ …explores:

$B \rightarrow C \rightarrow A$ $+\infty$

$B \rightarrow D \rightarrow A$ 3 + 2 = **5** **UPDATE!**

$B \dashrightarrow C$ …explores:

$B \rightarrow A \rightarrow C$ 5 + 2 = **7** **> 4, no update**

$B \rightarrow D \rightarrow C$ $+\infty$

$B \dashrightarrow D$ …explores:

$B \rightarrow A \rightarrow D$ 5 + 1 = **6** **> 3, no update**

$B \rightarrow C \rightarrow D$ 4 + (-2) = **2** **UPDATE!**

## Shortest Path Algorithms Runtime:

- Dijkstra's Algorithm:

  **$O(m + n \lg(n))$**

- Floyd-Warshall:

  **$O(n^3)$**

All Pairs Shortest Path:

Dense Graphs:

Sparse Graphs:

## Graphs

**Graph Implementations:**
- Edge List
- Adjacency Matrix
- Adjacency List

**Graph Traversals:**
- Breadth First
- Depth First

**Minimum Spanning Trees:**
- Kruskal’s Algorithm
- Prim’s Algorithm

**Shortest Path:**
- Dijkstra’s Algorithm
- Floyd-Warshall’s Algorithm

## CS 225 – Things To Be Doing

**Exam 13: Makeup Exam starts today**

**More Info:** https://courses.engr.illinois.edu/cs225/fa2017/exams/

**MP7: The final MP!**

*Due: Monday, Dec. 11 at 11:59pm*

**Final Exam starts Thursday!**

*Worth 250 points, the largest assessment all semester!*