Algorithms and Data Structures for Data Science
Shortest Path Algorithms

CS 277
Brad Solomon

April 24, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Exam 3 Signups Available

April 24 — April 27

Very limited window for makeup exams (since end of semester is near)

Covers content from week 10 — 14

Mini-Project 3

Average: 88%   Median: 99%

An optional final reflection form: [Mini-Project Reflection Feedback Form]

Please fill out ICES Evaluations

Feedback is important for the development of the class

This is especially important for this class as it is very new and growing

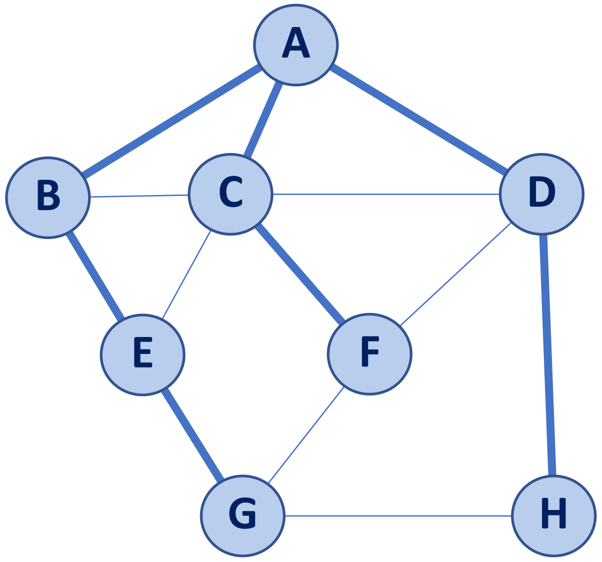
Learning Objectives

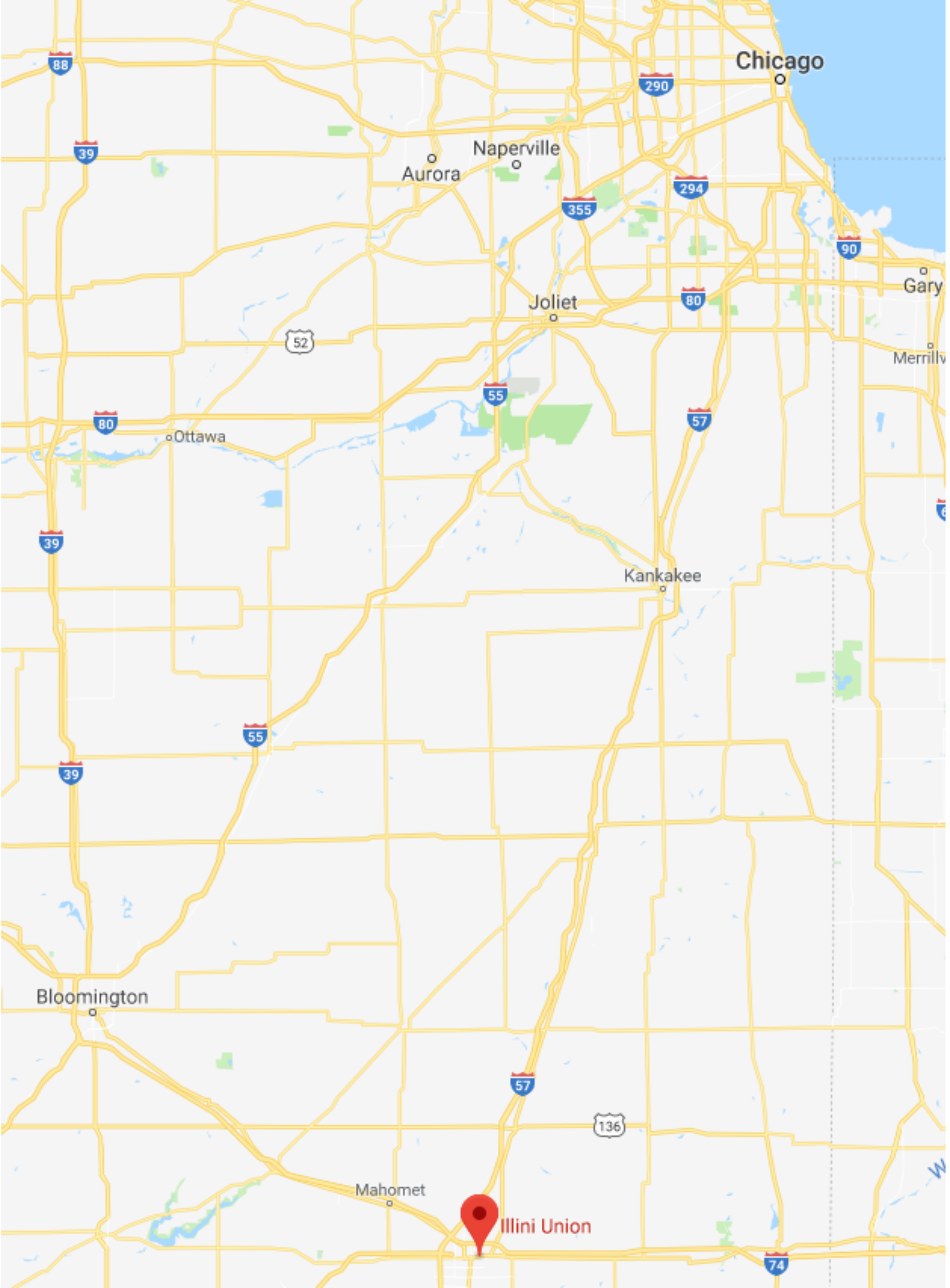
Introduce complexity of weights to shortest path / MST problem

Conceptualize Dijkstras Shortest Path Algorithm

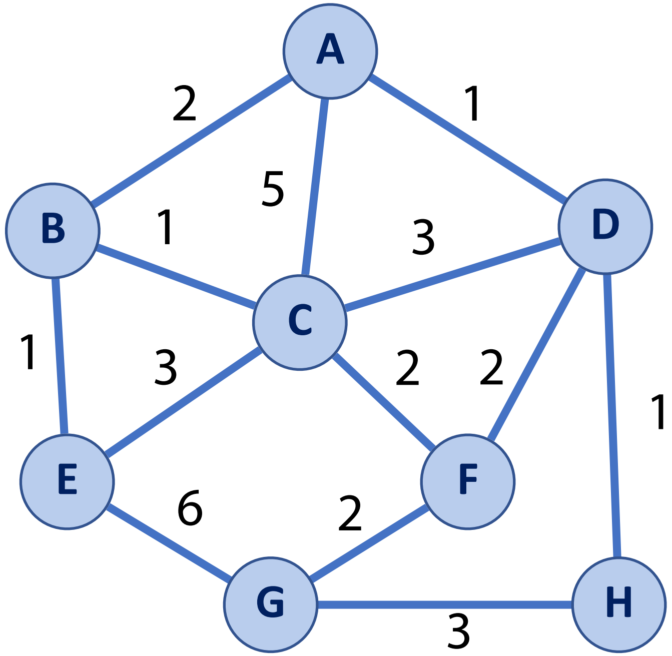
Code Dijkstras in Python and see how to use in Python

Shortest Path





Dijkstras Shortest Path (Distances)



1) Create data structs to store distances

   Create data structs to store visited

2) Initialize all distances to ∞ (and source to 0)

3) While there exists an unvisited vertex:

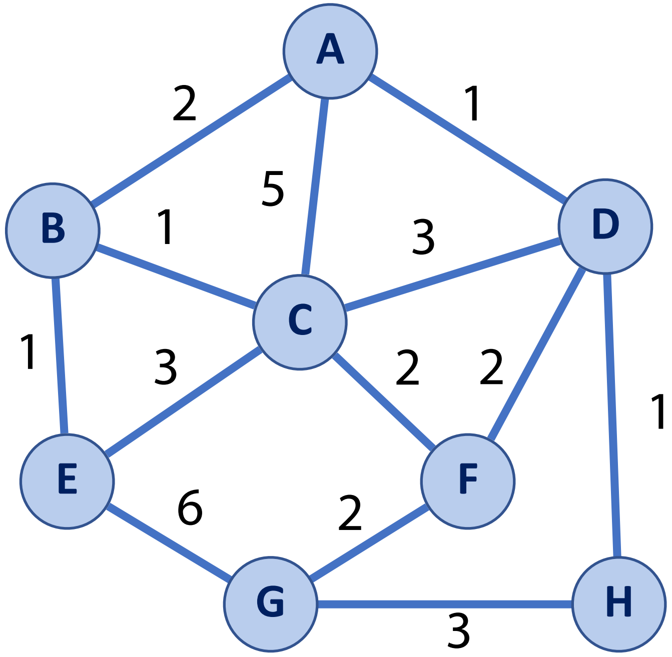
   Visit the current nearest vertex

   Update distances based on current edges

A: B: C: D: E: F: G: H:

Visited

Dijkstras Shortest Path (Distances)



1) Create data structs to store distances

   Create data structs to store visited

2) Initialize all distances to ∞ (and source to 0)

3) While there exists an unvisited vertex:

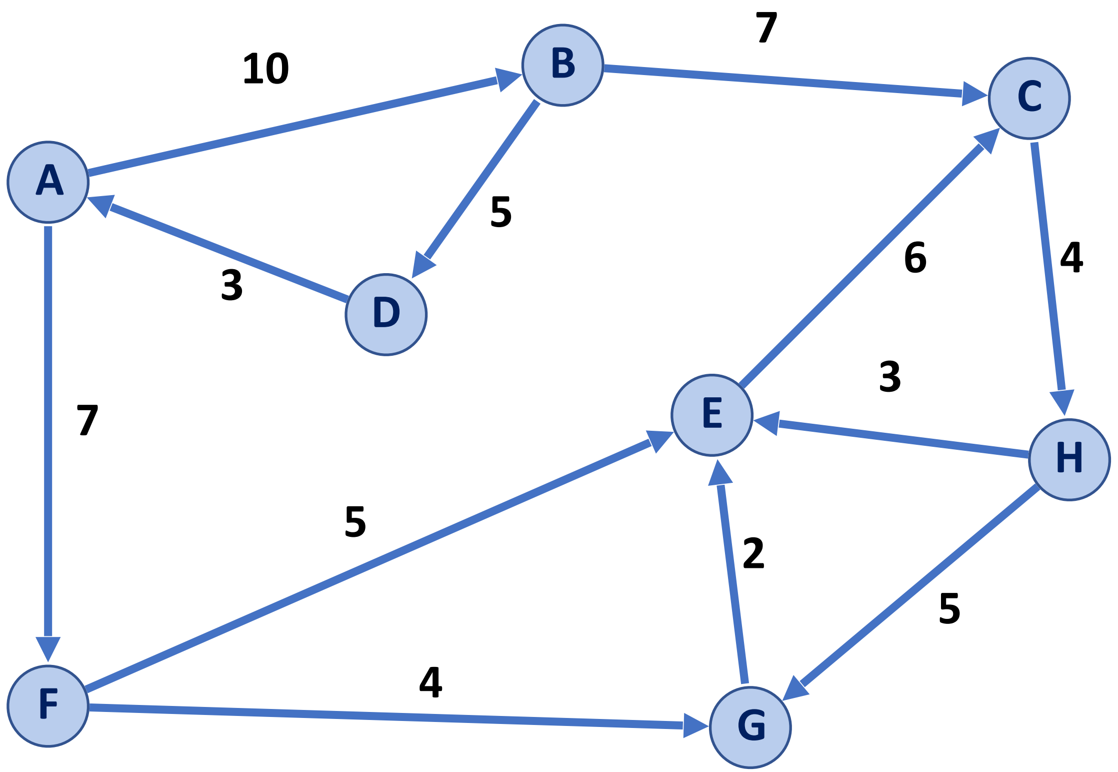
   Visit the current nearest vertex

   Update distances based on current edges

A: 0	B: 2	C: 3	D: 1	E: 3	F: 3	G: 5	H: 2

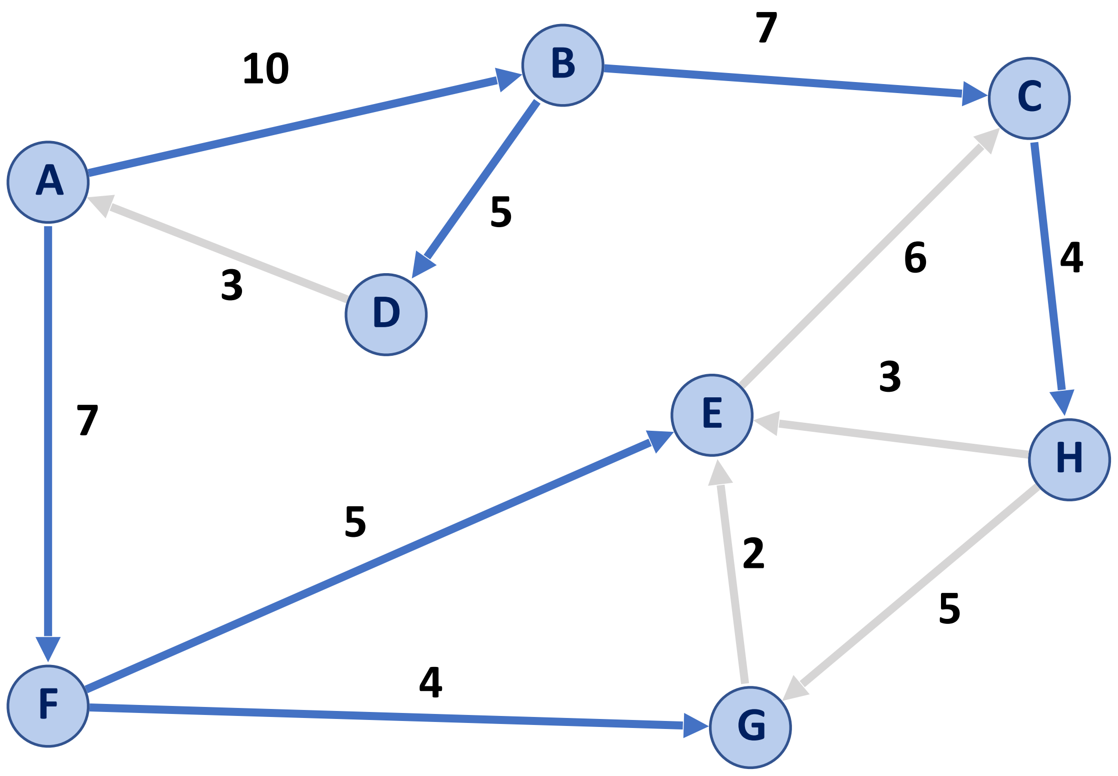
Visited D, B, H, C, E, F, G

Dijkstras Shortest Path (Full Paths)



Vertex	Distance	Previous
A		
B		
C		
D		
E		
F		
G		
H		

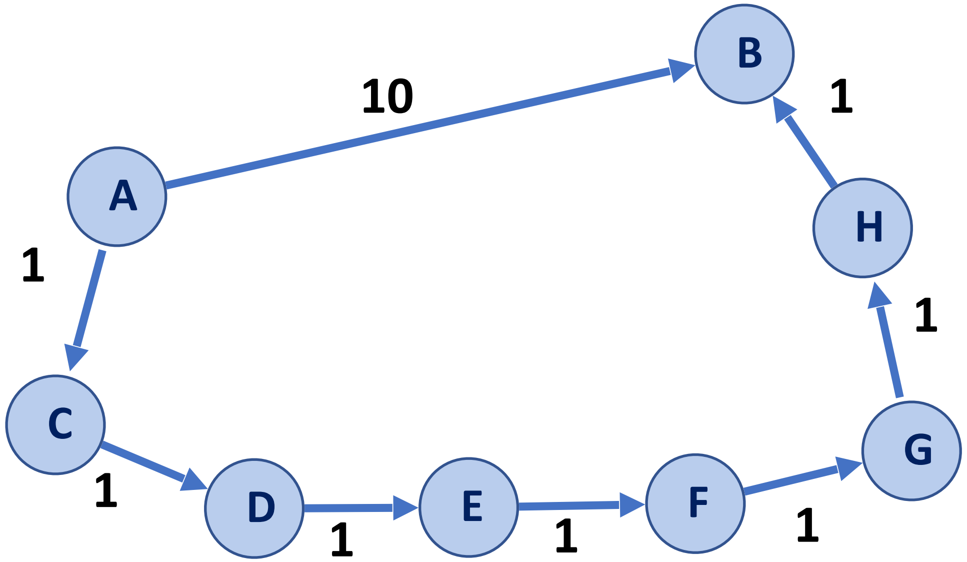
Dijkstras Shortest Path (Full Paths)



Vertex	Distance	Previous
A	**0**	**None**
B	**10**	**A**
C	**17**	**B**
D	**15**	**B**
E	**12**	**F**
F	**7**	**A**
G	**11**	**F**
H	**21**	**C**

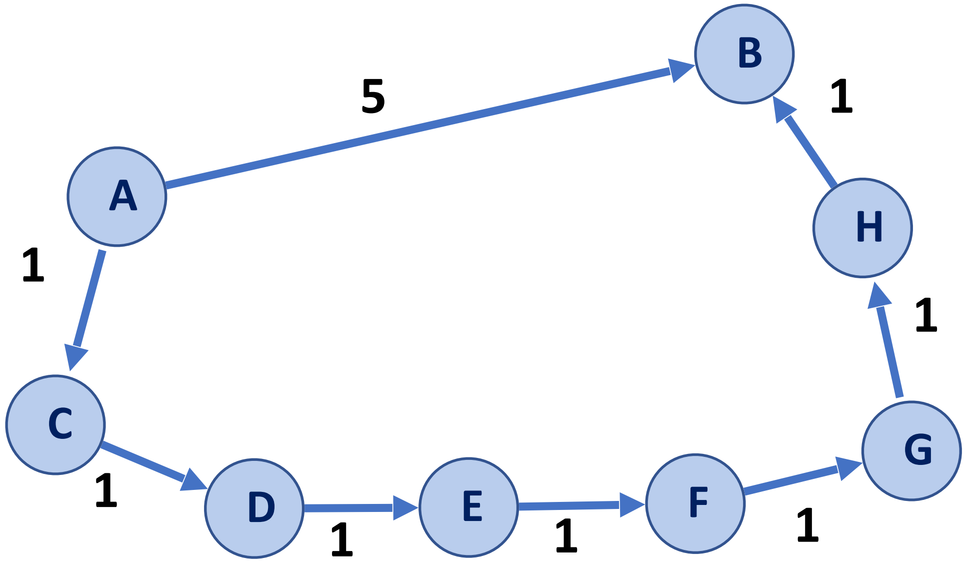
Dijkstra's Algorithm (SSSP)

When we will visit B in the following graph?



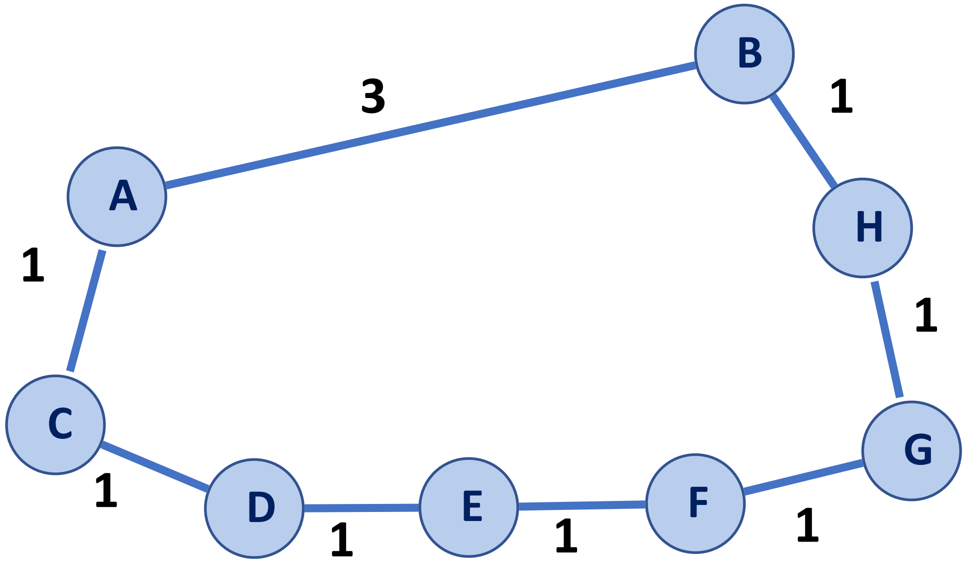
Dijkstra’s Algorithm (SSSP)

When we will visit B in the following graph?



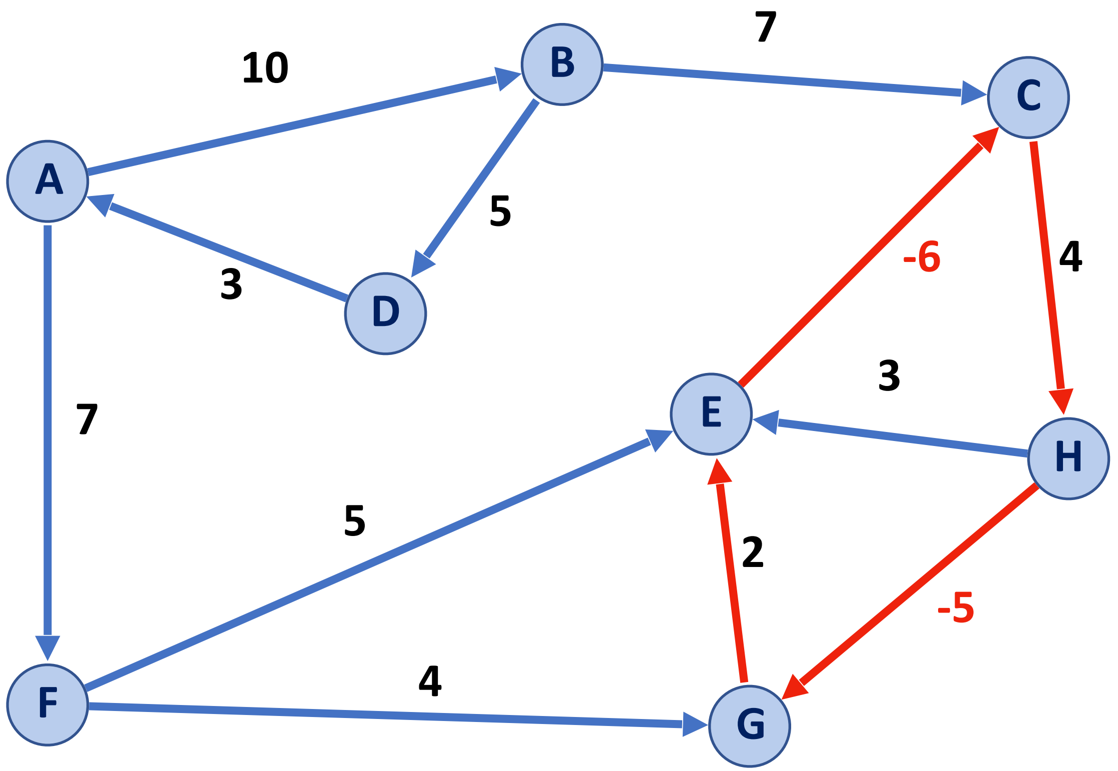
Dijkstra's Algorithm (SSSP)

When we will visit B in the following graph?



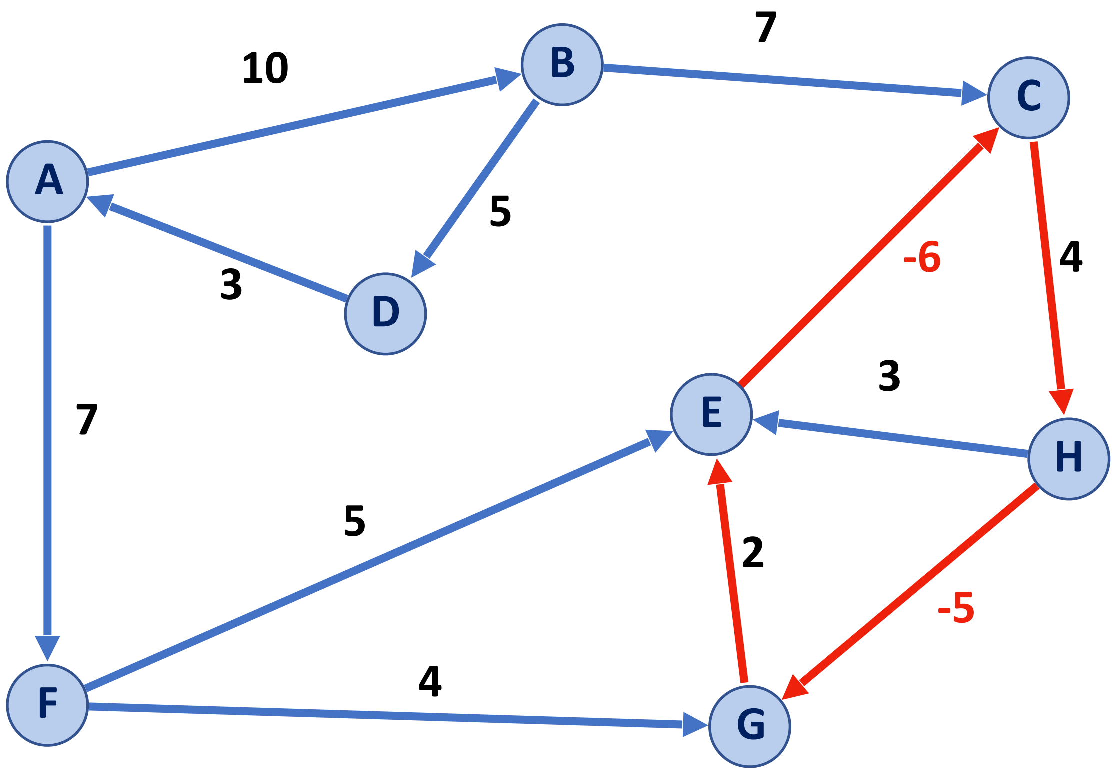
Dijkstra’s Algorithm (SSSP)

How does Dijkstras handle a negative weight cycle?



Dijkstra’s Algorithm (SSSP)

How does Dijkstras handle a negative weight cycle?



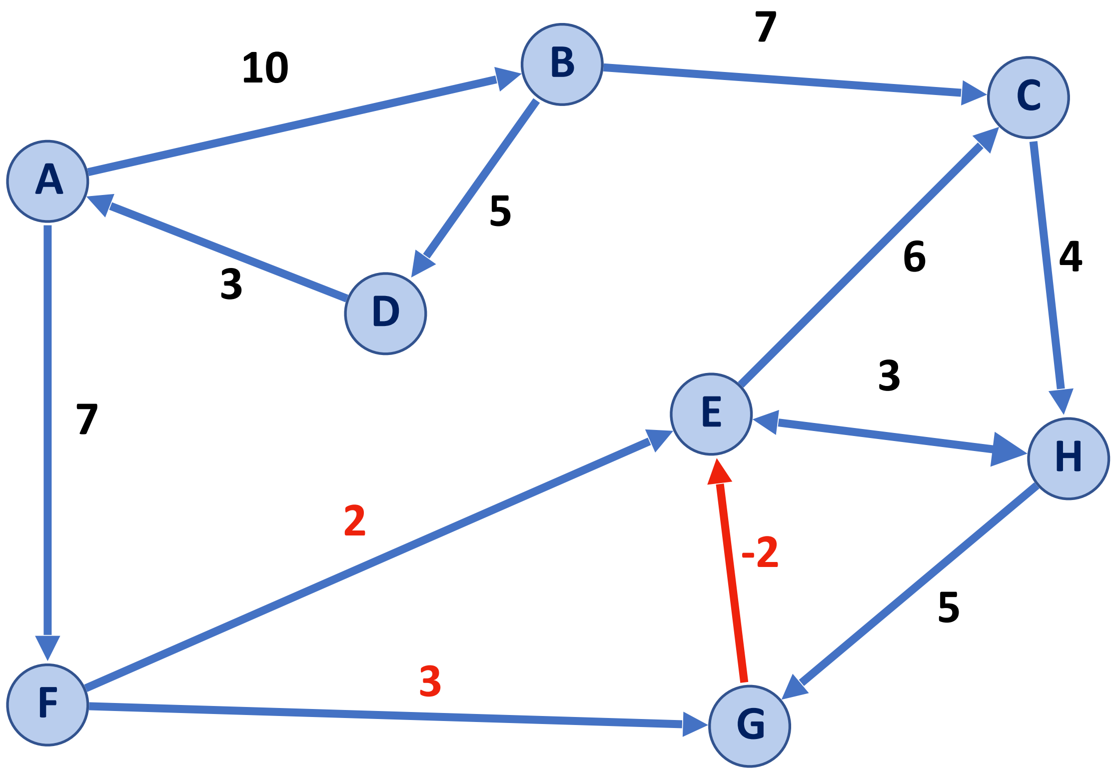
Shortest Path (A → E): A → F → E → (C → H → G → E)*

Length: 12

Length: -5 (repeatable)

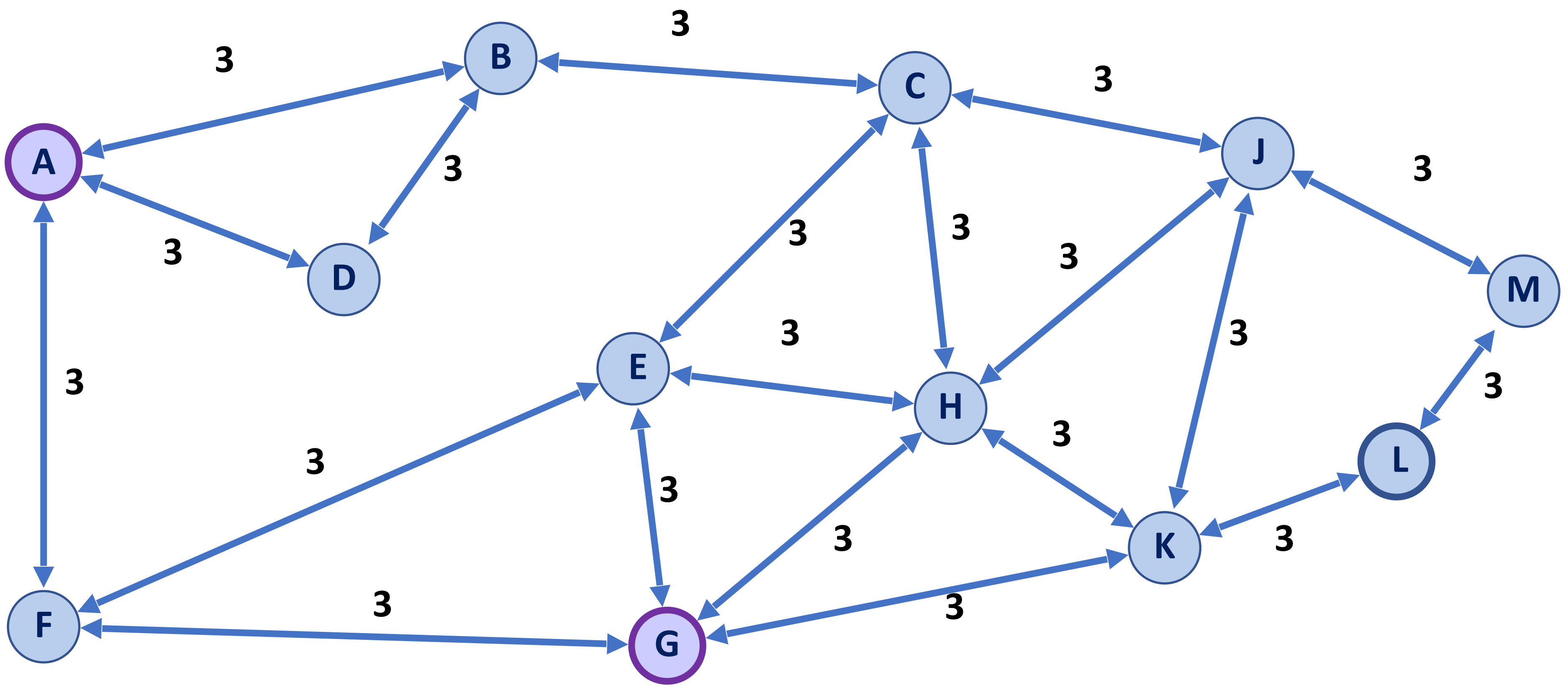
Dijkstra's Algorithm (SSSP)

How does Dijkstras handle a single negative edge?



Landmark Path Problem

What if I wanted to get the shortest path from A to G but stopping at L along the way?



Floyd-Warshall Algorithm

Floyd-Warshall's Algorithm is an alternative 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

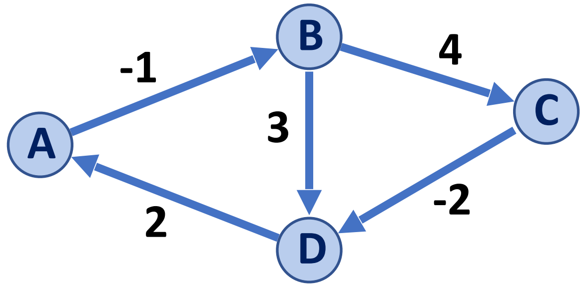
```
FloydWarshall(G):
  Let d be a adj. matrix initialized to +inf
  foreach (Vertex v : G):
    d[v][v] = 0
  foreach (Edge (u, v) : G):
    d[u][v] = cost(u, v)

  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]
```

```
def floydwarshall(inGraph):
    dist = {}

    for u in inGraph.edges.keys():
        dist[u]={}

    for u in inGraph.edges.keys():
        for v in inGraph.edges.keys():
            dist[u][v]=float('inf')

    for u in inGraph.edges.keys():
        dist[u][u]=0

    for u in inGraph.edges.keys():
        for e in inGraph.edges[u]:
            v = e[0]
            w = e[1]
            dist[u][v] = w

    for u in inGraph.edges.keys():
        for v in inGraph.edges.keys():
            for w in inGraph.edges.keys():
                if dist[u][v] > dist[u][w] + dist[w][v]:
                    dist[u][v]=dist[u][w] + dist[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)
```

	A	B	C	D
A				
B				
C				
D				

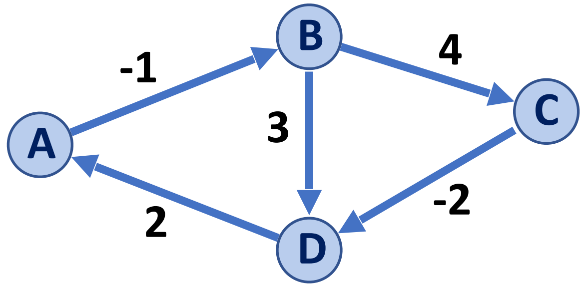


Floyd-Warshall Algorithm

```
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]
```

Let us consider w = A:

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



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

Let us consider w = A:

B → C **4** **vs.** B → A → C **+∞**

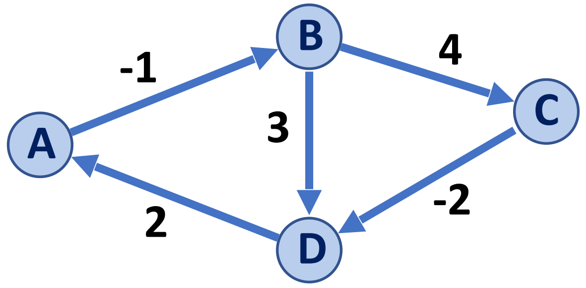
B → D **3** **vs.** B → A → D **+∞**

C → B **+∞** **vs.** C → A → B **+∞**

C → D **-2** **vs.** C → A → D **+∞**

D → B **+∞** **vs.** D → A → B

D → C **+∞** **vs.** D → A → C



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	1	∞	0

Let us consider k=A:

B → C 4 vs. B → A → C +∞

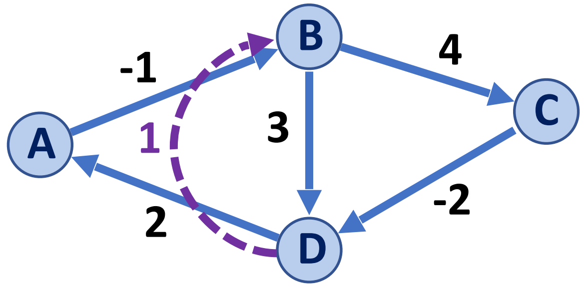
B → D 3 vs. B → A → D +∞

C → B +∞ vs. C → A → B +∞

C → D -2 vs. C → A → D +∞

D → B +∞ vs. D → A → B 1

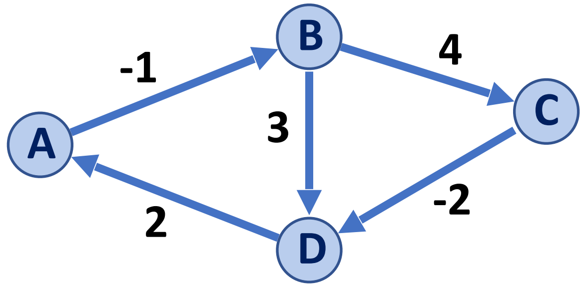
D → C +∞ vs. D → A → C +∞

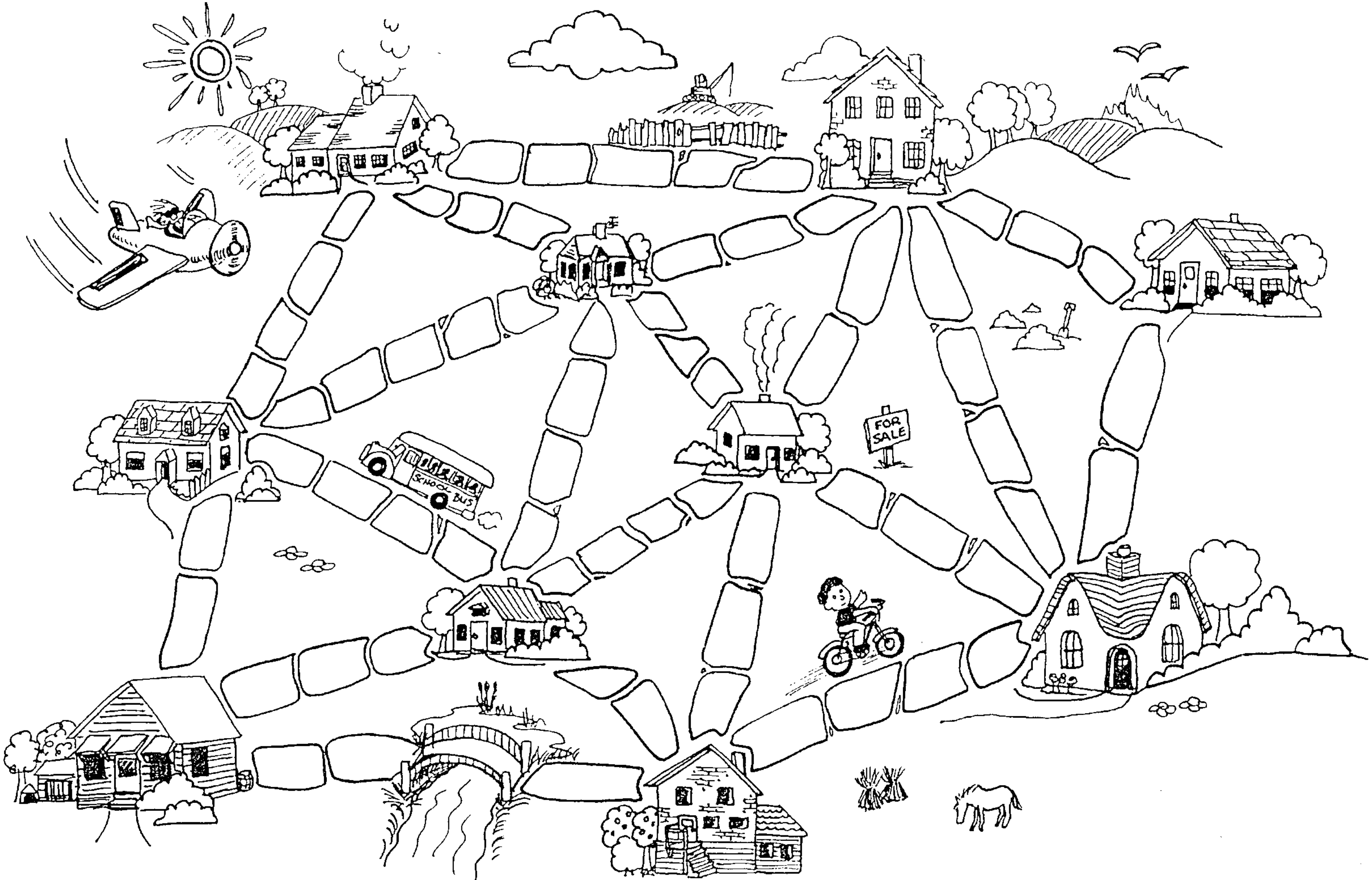


Floyd-Warshall Algorithm

```
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	0	-1	3	1
B	5	0	4	2
C	0	-1	0	-2
D	2	1	5	0




FOR SALE
SCHOOL BUS