Section 7:
Dijkstra's & Midterm Postmortem

Slides adapted from Alex Mariakakis,
with material Kellen Donohue, David Mailhot, and Dan Grossman

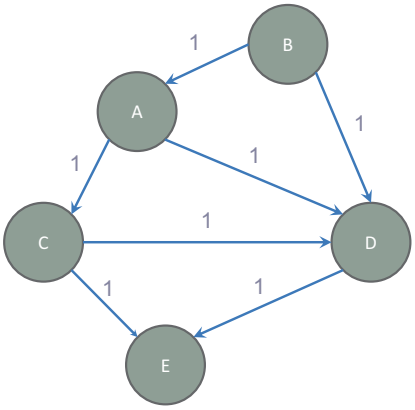
Agenda

- How to weight your edges
- Dijkstra's algorithm
- Midterm Q&A

Homework 7

- Modify your graph to use generics
  - Will have to update HW #5 and HW #6 tests
- Implement Dijkstra's algorithm
  - Search algorithm that accounts for edge weights
  - Note: This should not change your implementation of Graph. Dijkstra's is performed on a Graph, not within a Graph.
- The more well-connected two characters are, the lower the weight and the more likely that a path is taken through them
  - The weight of an edge is equal to the inverse of how many comic books the two characters share
  - Ex: If Carol Danvers and Luke Cage appeared in 5 comic books together, the weight of their edge would be 1/5
  - No duplicate edges

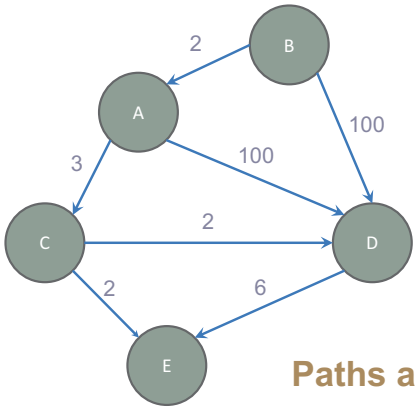
Review: Shortest Paths with BFS



From Node B

Destination	Path	Cost
A	<B,A>	1
B	<B>	0
C	<B,A,C>	2
D	<B,D>	1
E	<B,D,E>	2

Shortest Paths with Weights

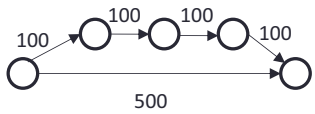


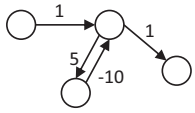
From Node B

Destination	Path	Cost
A	<B,A>	2
B	<B>	0
C	<B,A,C>	5
D	<B,A,C,D>	7
E	<B,A,C,E>	7

Paths are not the same!

BFS vs. Dijkstra's





- BFS doesn't work because path with minimal cost ≠ path with fewest edges
- Dijkstra's works if the weights are non-negative
- What happens if there is a negative edge?
  - Minimize cost by repeating the cycle forever

Dijkstra's Algorithm

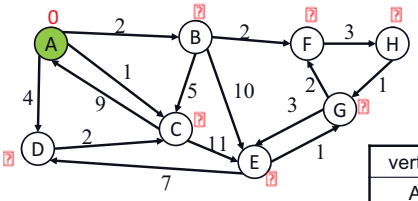
- Named after its inventor Edsger Dijkstra (1930-2002)
  - Turing Award winner and all-around amazing computer scientist
  - Other work includes Banker's algorithm, semaphores
- The idea: reminiscent of BFS, but adapted to handle weights
  - Grow the set of nodes whose shortest distance has been computed
  - Nodes not in the set will have a "best distance so far"
  - A priority queue will turn out to be useful for efficiency

Dijkstra's Algorithm

1. For each node `v`, set `v.cost = ∞` and `v.known = false`
2. Set `source.cost = 0`
3. While there are unknown nodes in the graph
   a) Select the unknown node `v` with lowest cost
   b) Mark `v` as known
   c) For each edge (`v`,`u`) with weight `w`,

```
c1 = v.cost + w      // cost of best path through v to u
c2 = u.cost          // cost of best path to u previously known
if(c1 < c2)          // if the new path through v is better, update
     u.cost = c1
     u.path = v
```

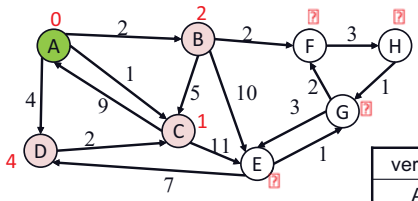
Example #1



Order Added to Known Set:

vertex	known?	cost	path
A	Y	0	
B		∞	
C		∞	
D		∞	
E		∞	
F		∞	
G		∞	
H		∞	

Example #1

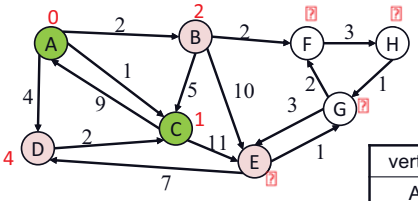


Order Added to Known Set:

A

vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Example #1

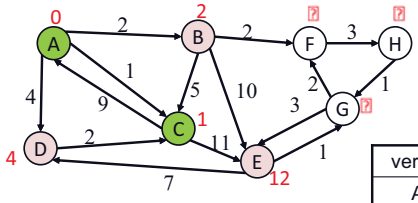


Order Added to Known Set:

A, C

vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Example #1

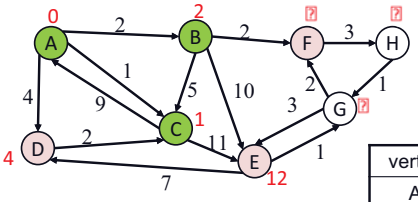


Order Added to Known Set:

A, C

vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Example #1

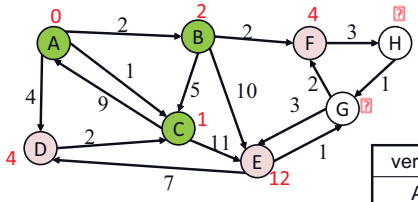


Order Added to Known Set:

A, C, B

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Example #1

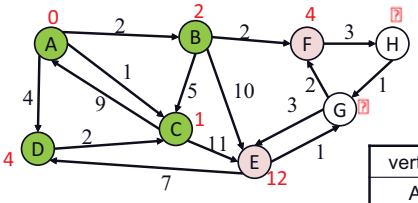


Order Added to Known Set:

A, C, B

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Example #1

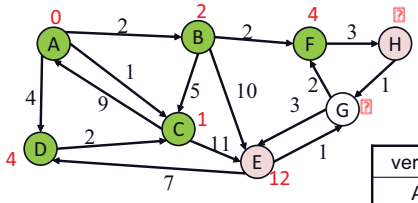


Order Added to Known Set:

A, C, B, D

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Example #1

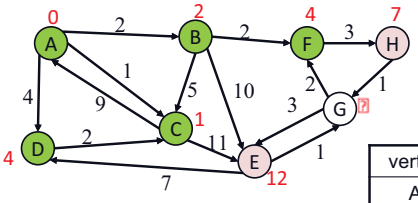


Order Added to Known Set:

A, C, B, D, F

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		∞	

Example #1

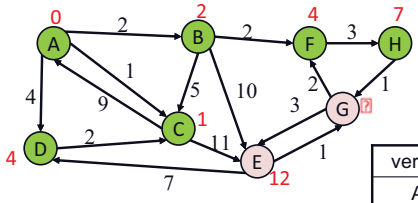


Order Added to Known Set:

A, C, B, D, F

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		≤ 7	F

Example #1

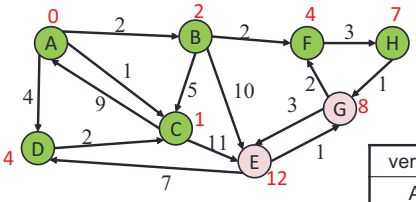


Order Added to Known Set:

A, C, B, D, F, H

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H	Y	7	F

Example #1

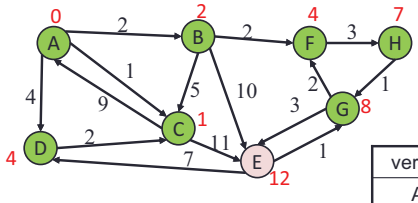


Order Added to Known Set:

A, C, B, D, F, H

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

Example #1

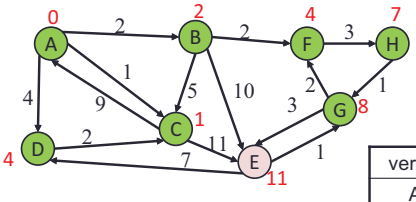


Order Added to Known Set:

A, C, B, D, F, H, G

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G	Y	8	H
H	Y	7	F

Example #1

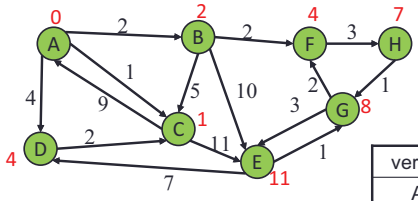


Order Added to Known Set:

A, C, B, D, F, H, G

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Example #1

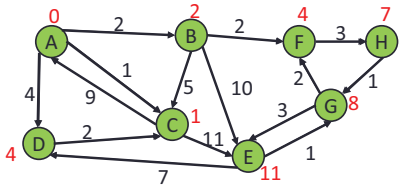


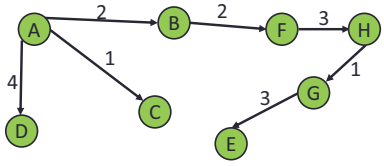
Order Added to Known Set:

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

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

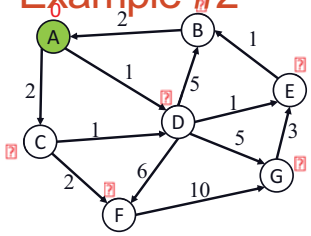
Interpreting the Results





vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

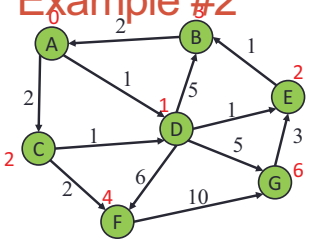
Example #2



Order Added to Known Set:

vertex	known?	cost	path
A	Y	0	
B		∞	
C		∞	
D		∞	
E		∞	
F		∞	
G		∞	

Example #2



Order Added to Known Set:

A, D, C, E, B, F, G

vertex	known?	cost	path
A	Y	0	
B	Y	3	E
C	Y	2	A
D	Y	1	A
E	Y	2	D
F	Y	4	C
G	Y	6	D

Pseudocode Attempt #1

```
dijkstra(Graph G, Node start) {
  for each node: x.cost=infinity, x.known=false
  start.cost = 0
  while(not all nodes are known) {
    b = dequeue
    b.known = true
    for each edge (b,a) in G {
      if(!a.known) {
        if(b.cost + weight((b,a)) < a.cost){
          a.cost = b.cost + weight((b,a))
          a.path = b
        }
      }
  brackets…
```

O(|V|)

O(|V|²)

O(|E|)

O(|V|²)

Can We Do Better?

- Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices
- PriorityQueue is like a queue, but returns elements by lowest value instead of FIFO

Priority Queue

- Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices
- PriorityQueue is like a queue, but returns elements by lowest value instead of FIFO
- Two ways to implement:
  1. Comparable
     a) class Node implements Comparable<Node>
     b) public int compareTo(other)
  2. Comparator
     a) class NodeComparator extends Comparator<Node>
     b) new PriorityQueue(new NodeComparator())

Pseudocode Attempt #2

```
dijkstra(Graph G, Node start) {
  for each node: x.cost=infinity, x.known=false
  start.cost = 0
  build-heap with all nodes
  while(heap is not empty) {
    b = deleteMin()
    if (b.known) continue;
    b.known = true
    for each edge (b,a) in G {
     if(!a.known) {
       add(b.cost + weight((b,a)) )
     }
brackets…
```

O(|V|)

O(|V|log|V|)

O(|E|log|V|)

O(|E|log|V|)

Proof of Correctness

- All the "known" vertices have the correct shortest path through induction
  - Initially, shortest path to start node has cost 0
  - If it stays true every time we mark a node "known", then by induction this holds and eventually everything is "known" with shortes path
- Key fact: When we mark a vertex "known" we won't discover a shorter path later
  - Remember, we pick the node with the min cost each round
  - Once a node is marked as "known", going through another path will only add weight
  - Only true when node weights are positive

MIDTERM QUESTIONS!

Midterm Question 6

A. @returns some number between x − 10 and x + 10

B. @returns some number between x − 5 and x + 5

C. @requires x > 0
@returns some number between x − 5 and x + 5

D. @requires x > 0 or x < −5
@returns some number between x − 5 and x + 5

E. @requires x > 0
@throws IllegalArgument if x > 100
@returns some number between x − 10 and x + 10