

# Advanced Shortest Paths: Bidirectional Dijkstra

Michael Levin

Higher School of Economics

Graph Algorithms  
Data Structures and Algorithms

# Outline

- 1 Bidirectional Search
- 2 Bidirectional Dijkstra

## Shortest Path

**Input:** A graph  $G$  with *non-negative* edge weights, a source vertex  $s$  and a target vertex  $t$ .

**Output:** The shortest path between  $s$  and  $t$  in  $G$ .

# Why not just Dijkstra?

- $O((|E| + |V|) \log |V|)$  is pretty fast, right?

# Why not just Dijkstra?

- $O((|E| + |V|) \log |V|)$  is pretty fast, right?
- For a graph of USA with 20M vertices and 50M edges it will work for several seconds on average

# Why not just Dijkstra?

- $O((|E| + |V|) \log |V|)$  is pretty fast, right?
- For a graph of USA with 20M vertices and 50M edges it will work for several seconds on average
- Millions of users of Google Maps want the result in a blink of an eye, all at the same time

# Why not just Dijkstra?

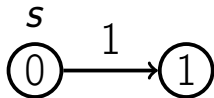
- $O((|E| + |V|) \log |V|)$  is pretty fast, right?
- For a graph of USA with 20M vertices and 50M edges it will work for several seconds on average
- Millions of users of Google Maps want the result in a blink of an eye, all at the same time
- Need something significantly faster

# Dijkstra Progression

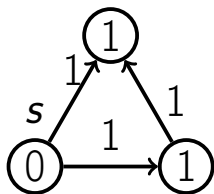
$s$   
①



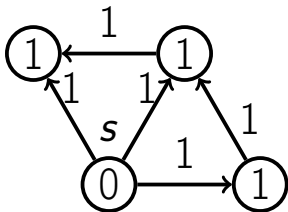
# Dijkstra Progression



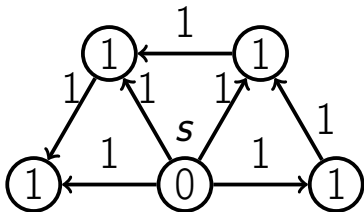
# Dijkstra Progression



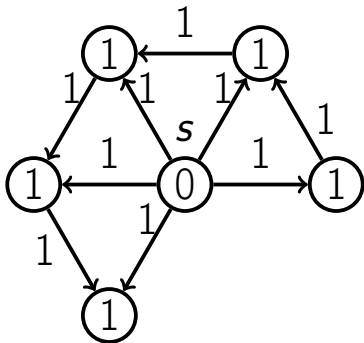
# Dijkstra Progression



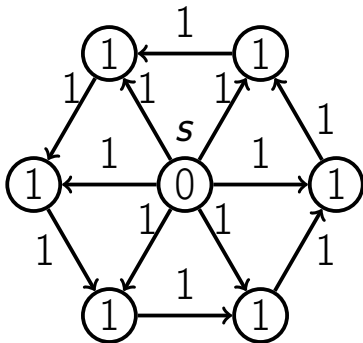
# Dijkstra Progression



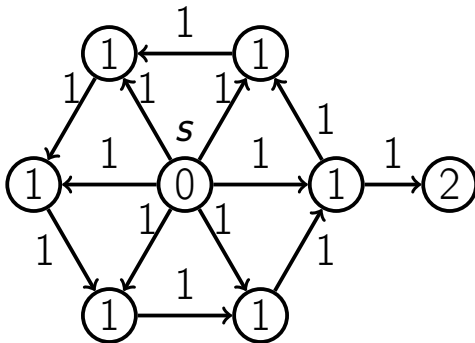
# Dijkstra Progression



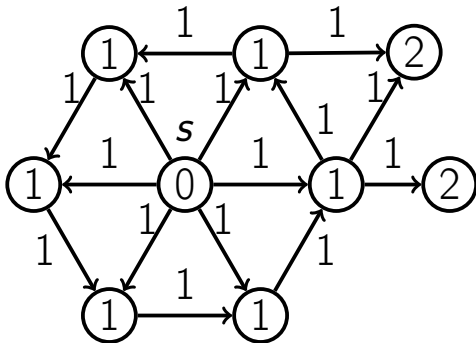
# Dijkstra Progression



# Dijkstra Progression

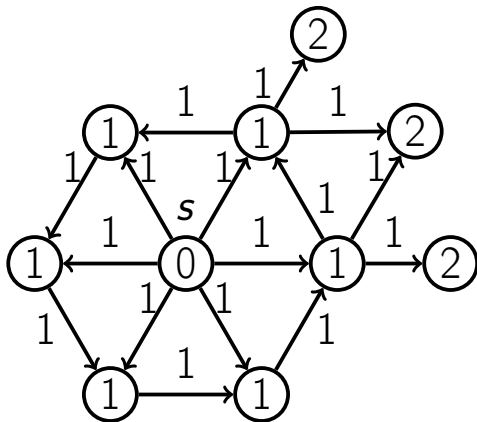


# Dijkstra Progression

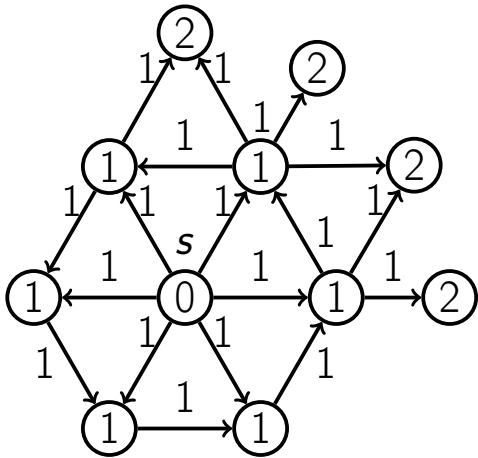




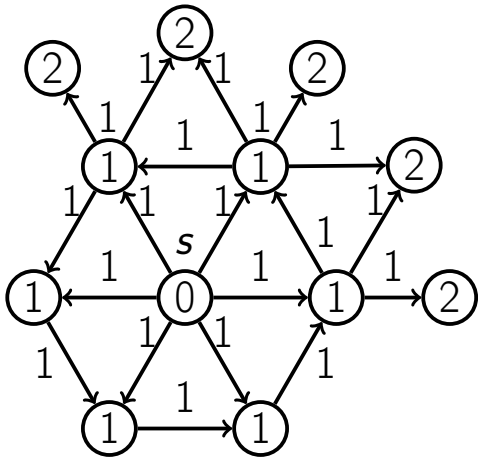
# Dijkstra Progression



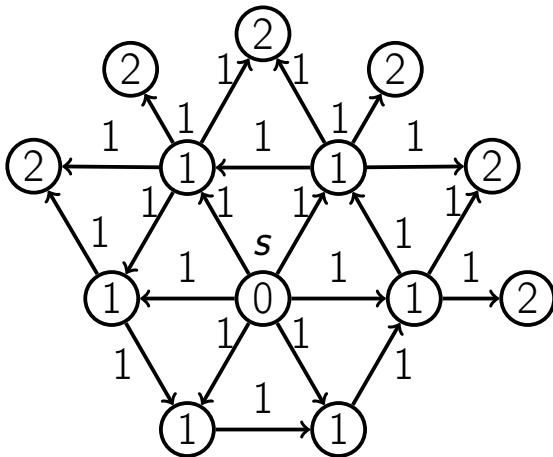
# Dijkstra Progression



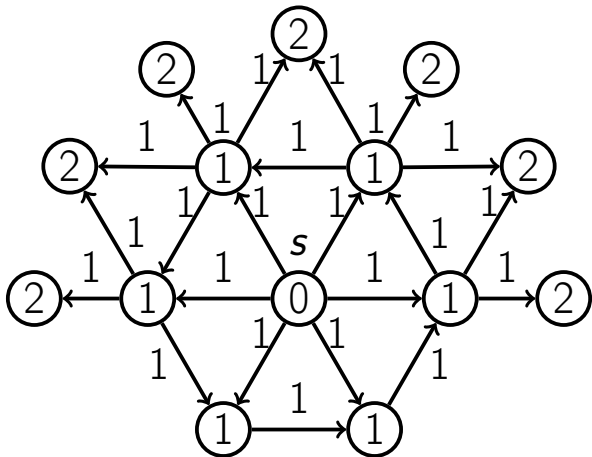
# Dijkstra Progression



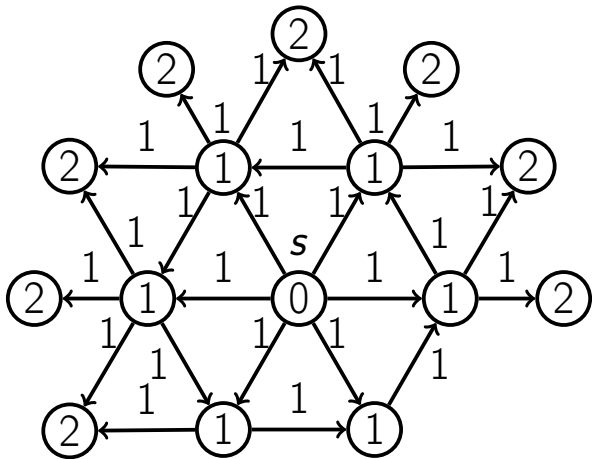
# Dijkstra Progression



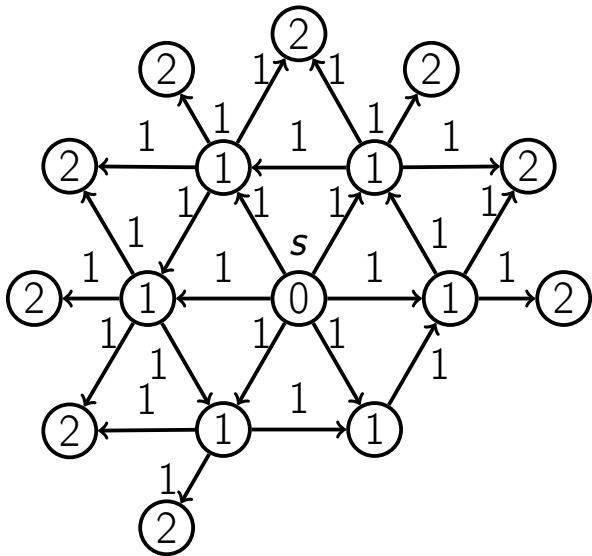
# Dijkstra Progression



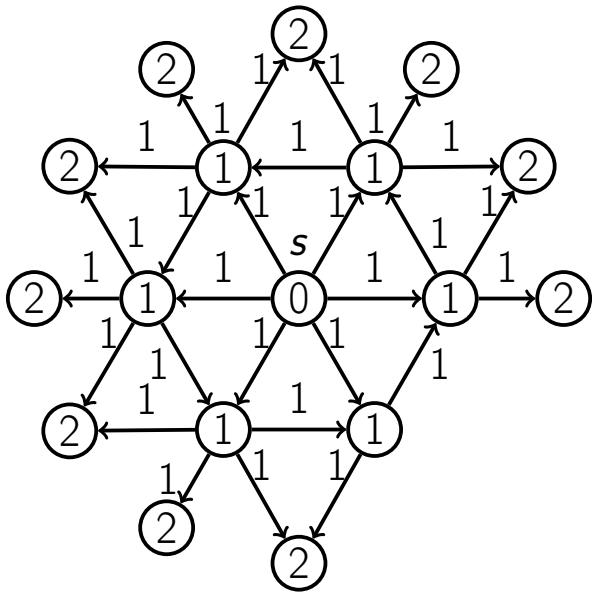
# Dijkstra Progression



# Dijkstra Progression

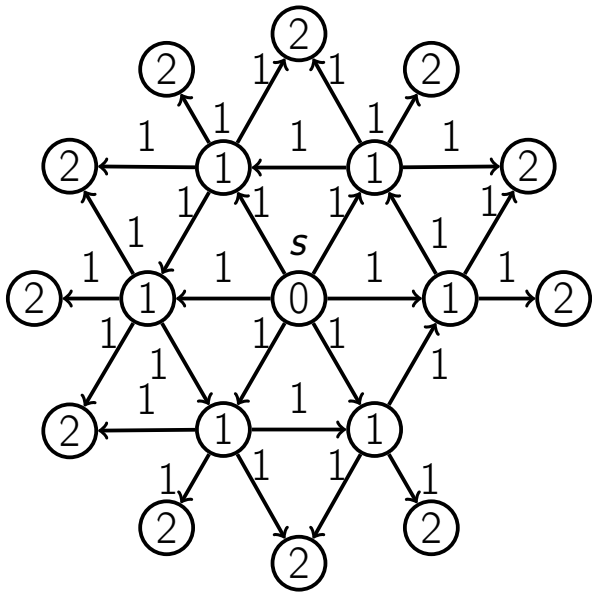


# Dijkstra Progression

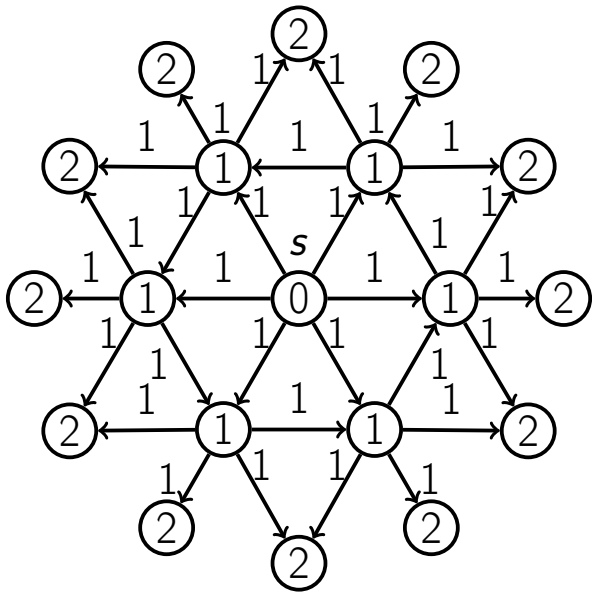




# Dijkstra Progression



# Dijkstra Progression



# Idea: Growing Circle

## Lemma

When a vertex  $u$  is selected via `ExtractMin`,  
 $\text{dist}[u] = d(s, u)$ .

- When a vertex is extracted from the priority queue for processing, all the vertices at smaller distances have already been processed

# Idea: Growing Circle

## Lemma

When a vertex  $u$  is selected via `ExtractMin`,  
 $\text{dist}[u] = d(s, u)$ .

- When a vertex is extracted from the priority queue for processing, all the vertices at smaller distances have already been processed
- A “circle” of processed vertices grows

# Idea: Growing Circle

$s$  ●

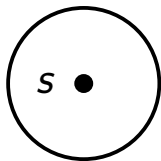
●  $t$

# Idea: Growing Circle

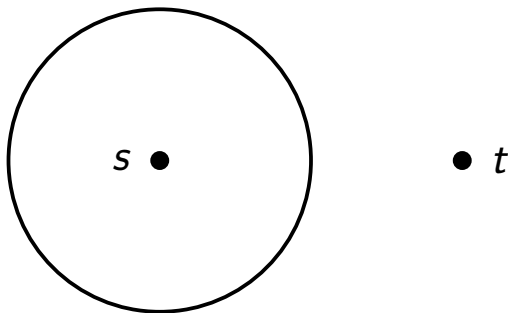
$s \odot$

$\bullet t$

# Idea: Growing Circle

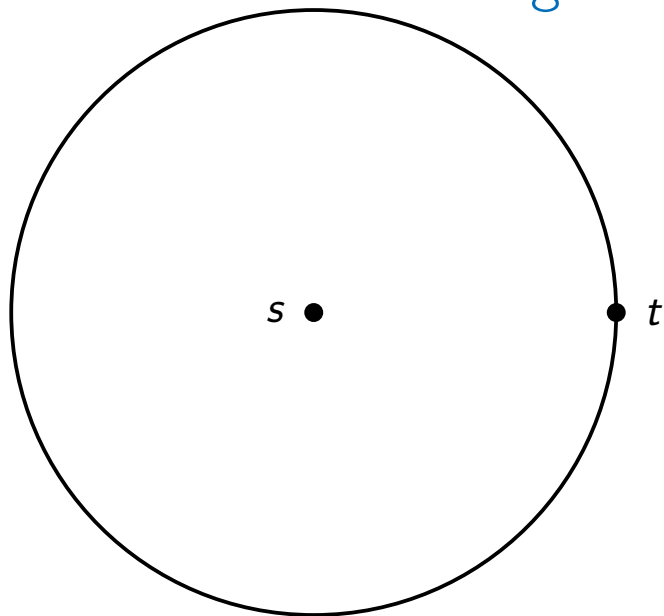


# Idea: Growing Circle





Idea: Growing Circle

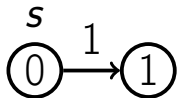


# Idea: Bidirectional Search

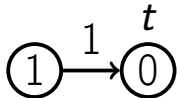
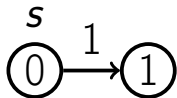
$s$   
0

$t$   
0

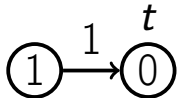
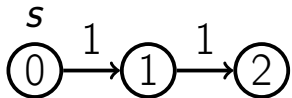
# Idea: Bidirectional Search



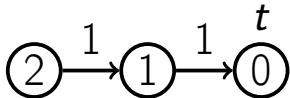
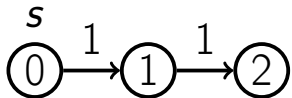
# Idea: Bidirectional Search



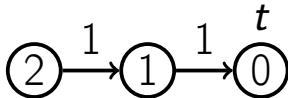
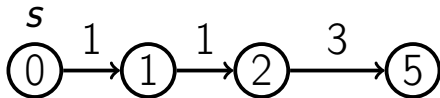
# Idea: Bidirectional Search



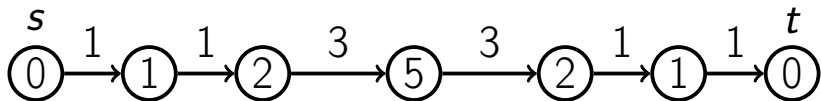
# Idea: Bidirectional Search



# Idea: Bidirectional Search

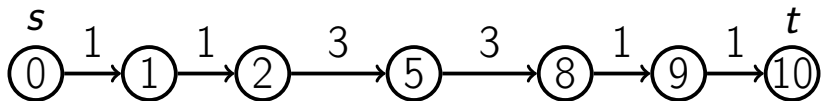


# Idea: Bidirectional Search

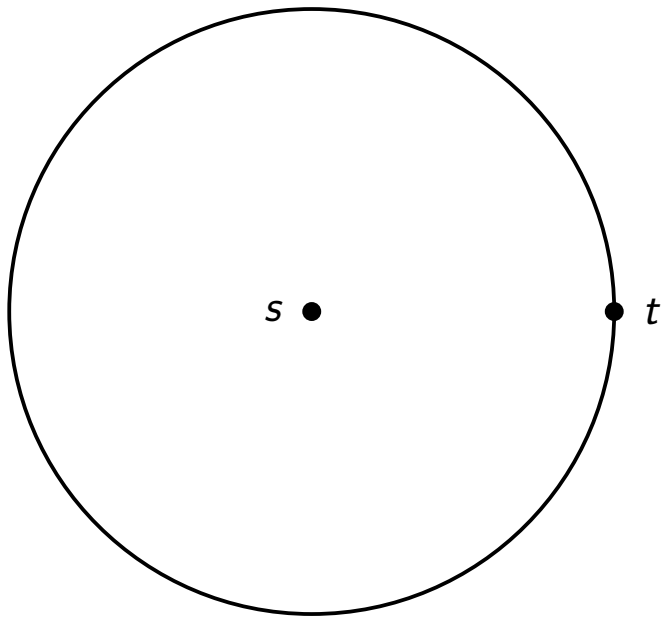




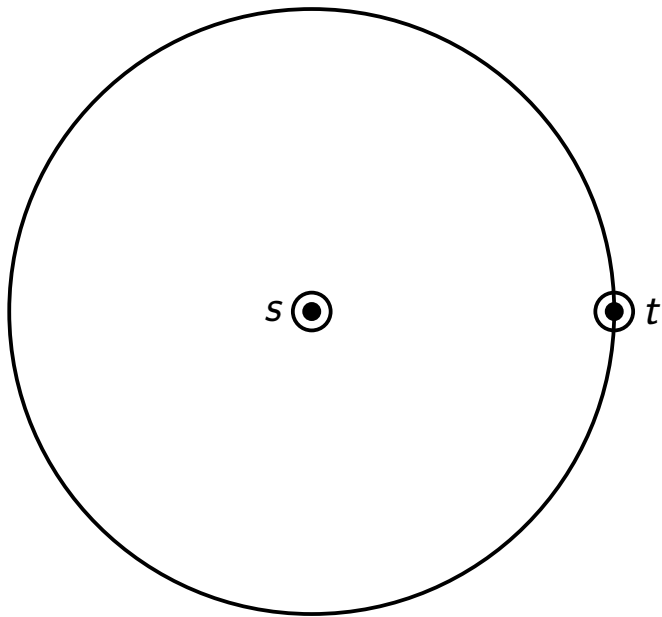
# Idea: Bidirectional Search



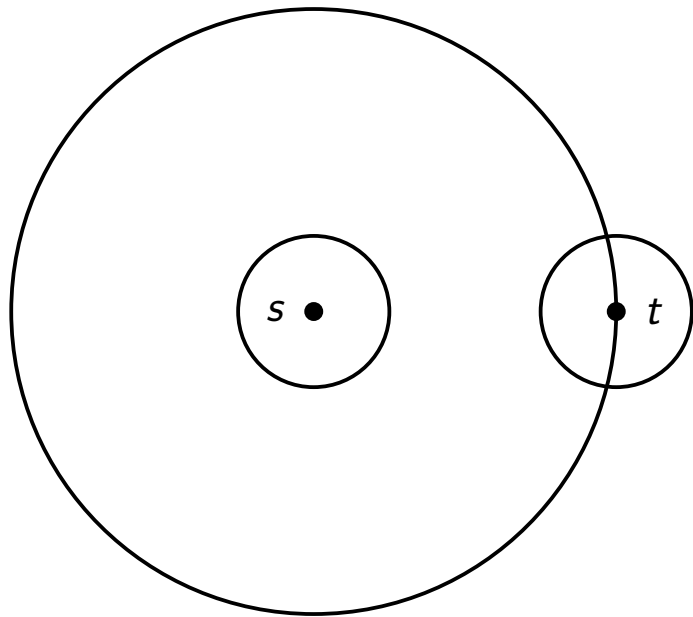
# Idea: Bidirectional Search



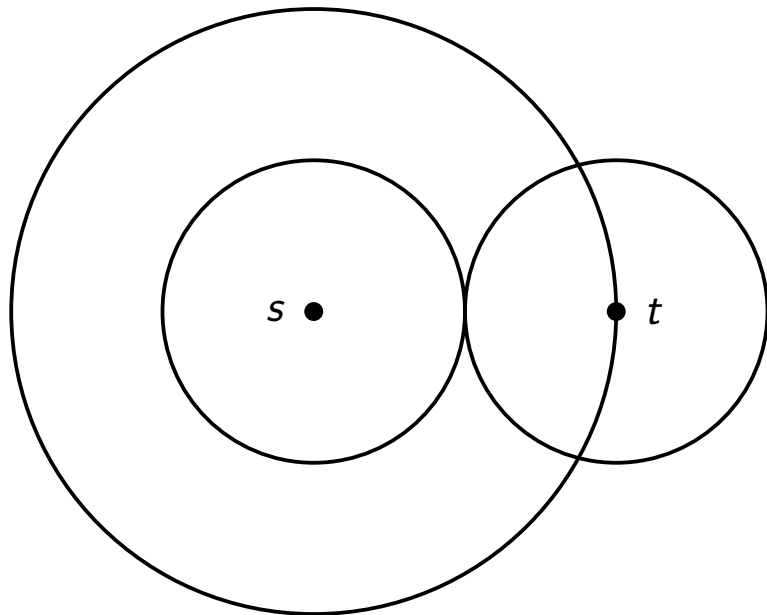
# Idea: Bidirectional Search



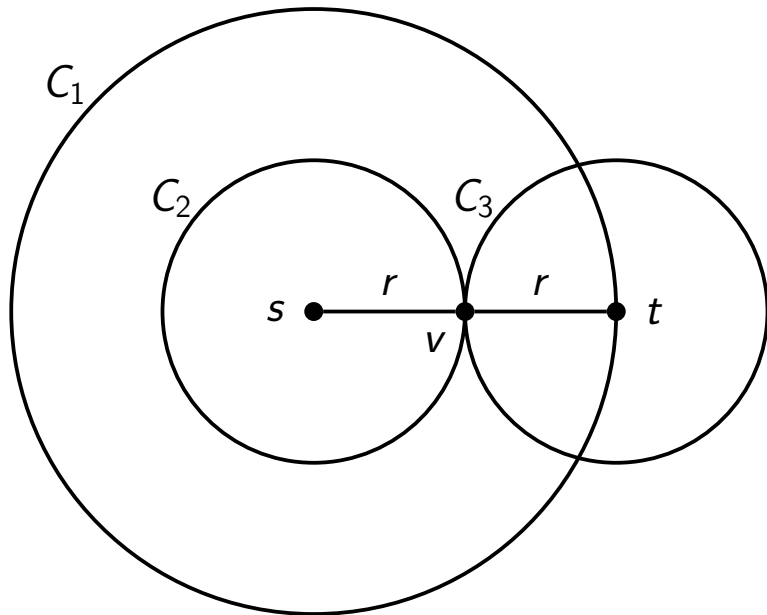
# Idea: Bidirectional Search



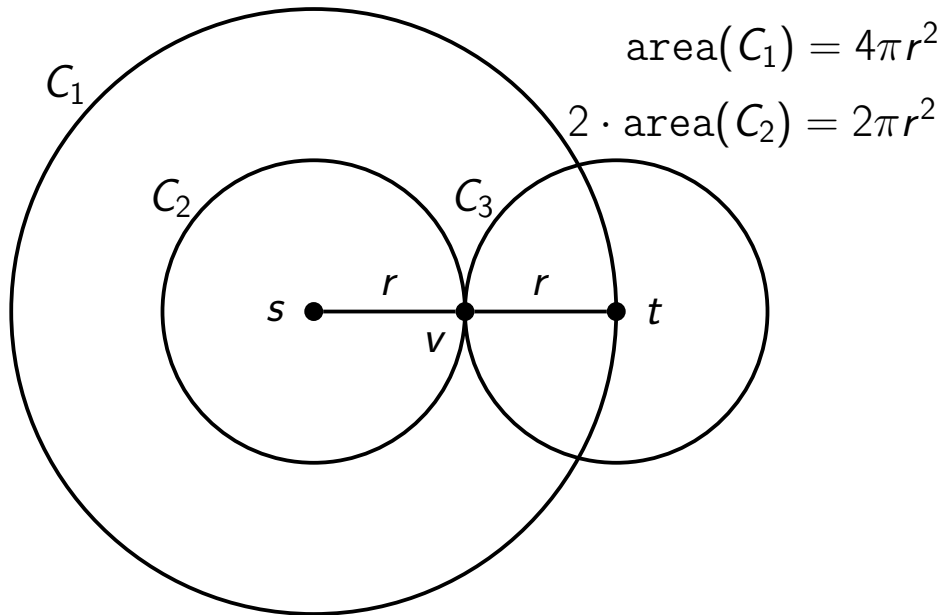
# Idea: Bidirectional Search



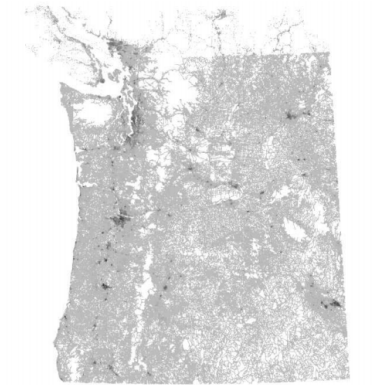
# Idea: Bidirectional Search



# Idea: Bidirectional Search



# Road networks

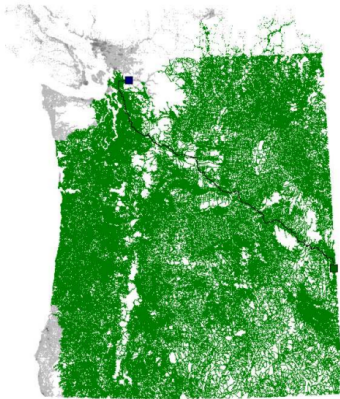


1.6M vertices, 3.8M arcs, travel time metric.

Picture by Andrew Goldberg.



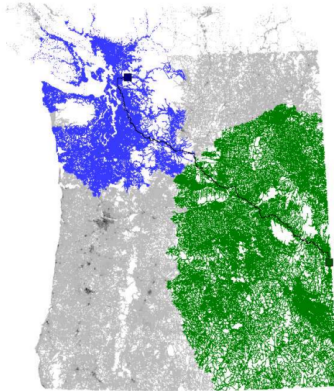
# Road networks



Searched area

Picture by Andrew Goldberg.

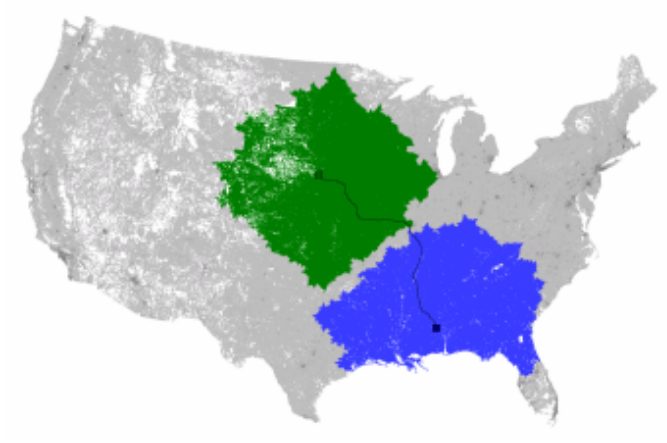
# Road networks



forward search / reverse search

Picture by Andrew Goldberg.

# Road networks



Picture by Andrew Goldberg.

- Roughly 2x speedup

- Roughly 2x speedup
- Good, but not great

- Roughly 2x speedup
- Good, but not great
- This is true for road networks

- Roughly 2x speedup
- Good, but not great
- This is true for road networks
- Let's look at social networks

# Six Handshakes

- In 1929, Hungarian mathematician Frigyes Karinty made a “Small World” conjecture



# Six Handshakes

- In 1929, Hungarian mathematician Frigyes Karinty made a “Small World” conjecture
- Can pass a message from any person to any person in at most 6 handshakes

# Six Handshakes

- In 1929, Hungarian mathematician Frigyes Karinty made a “Small World” conjecture
- Can pass a message from any person to any person in at most 6 handshakes
- This is close to truth according to experiments and is called a “six handshakes” or “six degrees of separation” idea

# Facebook

- Suppose an average person has around 100 Facebook friends

# Facebook

- Suppose an average person has around 100 Facebook friends
- Then 10000 friends of friends

# Facebook

- Suppose an average person has around 100 Facebook friends
- Then 10000 friends of friends
- 1000000 friends of friends of friends

# Facebook

- Suppose an average person has around 100 Facebook friends
- Then 10000 friends of friends
- 1000000 friends of friends of friends
- ...

# Facebook

- Suppose an average person has around 100 Facebook friends
- Then 10000 friends of friends
- 1000000 friends of friends of friends
- ...
- 1 trillion people at six handshakes

# Facebook

- Suppose an average person has around 100 Facebook friends
- Then 10000 friends of friends
- 1000000 friends of friends of friends
- ...
- 1 trillion people at six handshakes
- Not possible, as there are only about 7 billion people on earth



# Facebook

- Find the shortest path from Michael to Bob via friends connections

# Facebook

- Find the shortest path from Michael to Bob via friends connections
- For the two “farthest” people, Dijkstra has to look through 2 billion people

# Facebook

- Find the shortest path from Michael to Bob via friends connections
- For the two “farthest” people, Dijkstra has to look through 2 billion people
- If we only consider friends of friends of friends for both Michael and Bob, we will find a connection

# Facebook

- Find the shortest path from Michael to Bob via friends connections
- For the two “farthest” people, Dijkstra has to look through 2 billion people
- If we only consider friends of friends of friends for both Michael and Bob, we will find a connection
- Roughly 1M friends of friends of friends

# Facebook

- Find the shortest path from Michael to Bob via friends connections
- For the two “farthest” people, Dijkstra has to look through 2 billion people
- If we only consider friends of friends of friends for both Michael and Bob, we will find a connection
- Roughly 1M friends of friends of friends
- $1M + 1M = 2M$  people — 1000 times less

# Meet-in-the-middle

- More general idea, not just for graphs

# Meet-in-the-middle

- More general idea, not just for graphs
- Instead of searching for all possible objects, search for first halves and for second halves separately

# Meet-in-the-middle

- More general idea, not just for graphs
- Instead of searching for all possible objects, search for first halves and for second halves separately
- Then find “compatible” halves



# Meet-in-the-middle

- More general idea, not just for graphs
- Instead of searching for all possible objects, search for first halves and for second halves separately
- Then find “compatible” halves
- Typically roughly  $O(\sqrt{N})$  instead of  $O(N)$ , including the previous Facebook example

# Conclusion

- Dijkstra goes in “circles”
- Bidirectional search idea can reduce the search space
- Roughly 2x speedup for road networks
- Meet-in-the-middle —  $\sqrt{N}$  instead of  $N$
- 1000 times faster for social networks
- Next video — Bidirectional Dijkstra algorithm

# Outline

- 1 Bidirectional Search
- 2 Bidirectional Dijkstra

# Dijkstra Reminder

- To find the shortest path from  $s$  to  $t$

# Dijkstra Reminder

- To find the shortest path from  $s$  to  $t$
- Initialize  $\text{dist}[s]$  to 0, all other distances to  $\infty$

# Dijkstra Reminder

- To find the shortest path from  $s$  to  $t$
- Initialize  $\text{dist}[s]$  to 0, all other distances to  $\infty$
- ExtractMin — choose *unprocessed*  $u$  with the smallest  $\text{dist}[u]$

# Dijkstra Reminder

- To find the shortest path from  $s$  to  $t$
- Initialize  $\text{dist}[s]$  to 0, all other distances to  $\infty$
- ExtractMin — choose *unprocessed*  $u$  with the smallest  $\text{dist}[u]$
- Process  $u$  — Relax the edges outgoing from  $u$

# Dijkstra Reminder

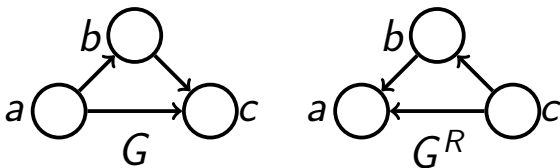
- To find the shortest path from  $s$  to  $t$
- Initialize  $\text{dist}[s]$  to 0, all other distances to  $\infty$
- ExtractMin — choose *unprocessed*  $u$  with the smallest  $\text{dist}[u]$
- Process  $u$  — Relax the edges outgoing from  $u$
- Repeat until  $t$  is processed



# Reversed Graph

## Definition

Reversed graph  $G^R$  for a graph  $G$  is the graph with the same set of vertices  $V$  and the set of reversed edges  $E^R$ , such that for any edge  $(u, v) \in E$  there is an edge  $(v, u) \in E^R$  and vice versa.



# Bidirectional Dijkstra

- Build  $G^R$

# Bidirectional Dijkstra

- Build  $G^R$
- Start Dijkstra from  $s$  in  $G$  and from  $t$  in  $G^R$

# Bidirectional Dijkstra

- Build  $G^R$
- Start Dijkstra from  $s$  in  $G$  and from  $t$  in  $G^R$
- Alternate between Dijkstra steps in  $G$  and in  $G^R$

# Bidirectional Dijkstra

- Build  $G^R$
- Start Dijkstra from  $s$  in  $G$  and from  $t$  in  $G^R$
- Alternate between Dijkstra steps in  $G$  and in  $G^R$
- Stop when some vertex  $v$  is processed both in  $G$  and in  $G^R$

# Bidirectional Dijkstra

- Build  $G^R$
- Start Dijkstra from  $s$  in  $G$  and from  $t$  in  $G^R$
- Alternate between Dijkstra steps in  $G$  and in  $G^R$
- Stop when some vertex  $v$  is processed both in  $G$  and in  $G^R$
- Compute the shortest path between  $s$  and  $t$

# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?

# Computing Distance

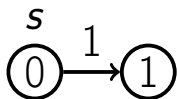
Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?





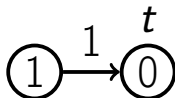
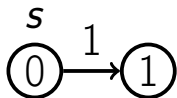
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



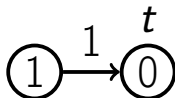
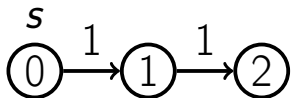
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



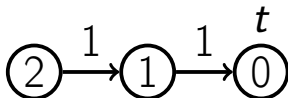
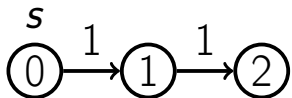
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



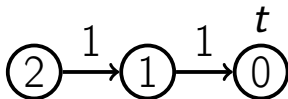
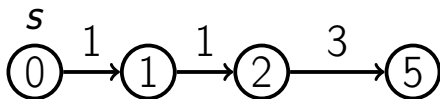
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



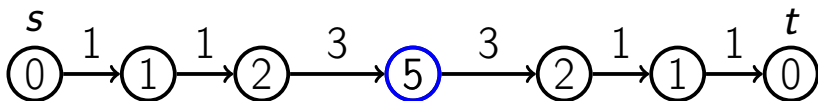
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



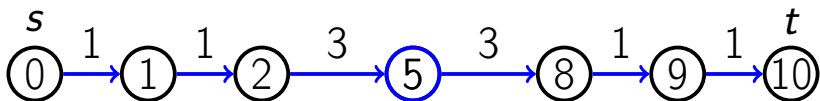
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



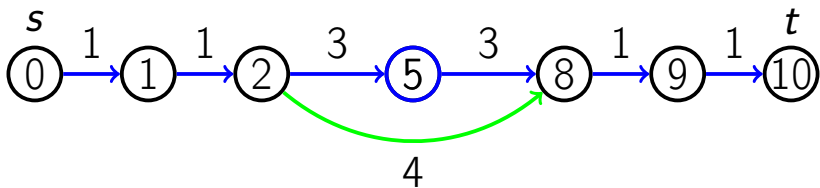
# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?



# Computing Distance

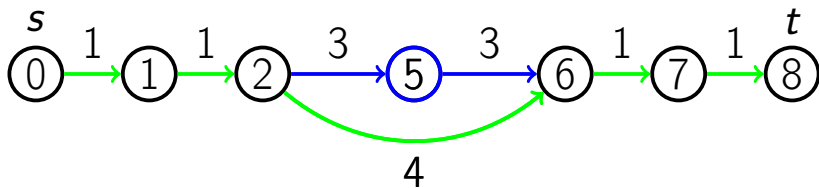
Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?





# Computing Distance

Let  $v$  be the first vertex which is processed both in  $G$  and in  $G^R$ . Does it follow that there is a shortest path from  $s$  to  $t$  going through  $v$ ?

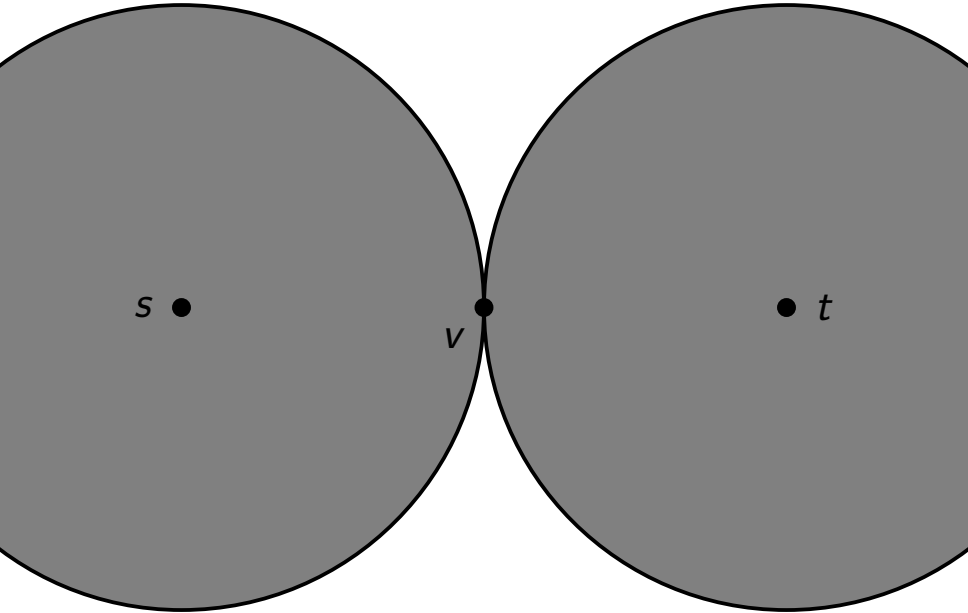


# Computing Distance

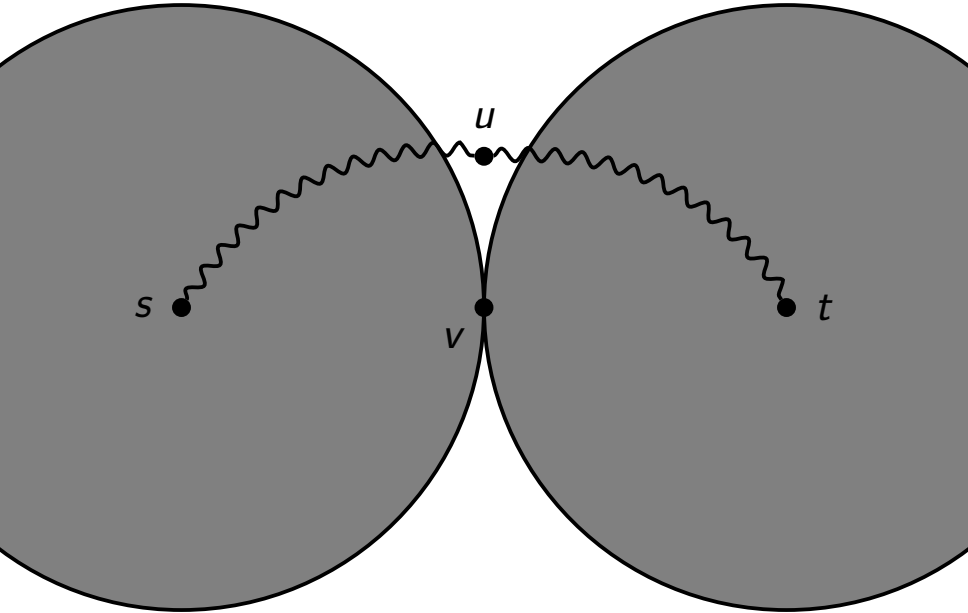
## Lemma

Let  $\text{dist}[u]$  be the distance estimate in the forward Dijkstra from  $s$  in  $G$  and  $\text{dist}^R[u]$  — the same in the backward Dijkstra from  $t$  in  $G^R$ . After some node  $v$  is processed both in  $G$  and  $G^R$ , some shortest path from  $s$  to  $t$  passes through some node  $u$  which is processed either in  $G$ , in  $G^R$ , or both, and  $d(s, t) = \text{dist}[u] + \text{dist}^R[u]$ .

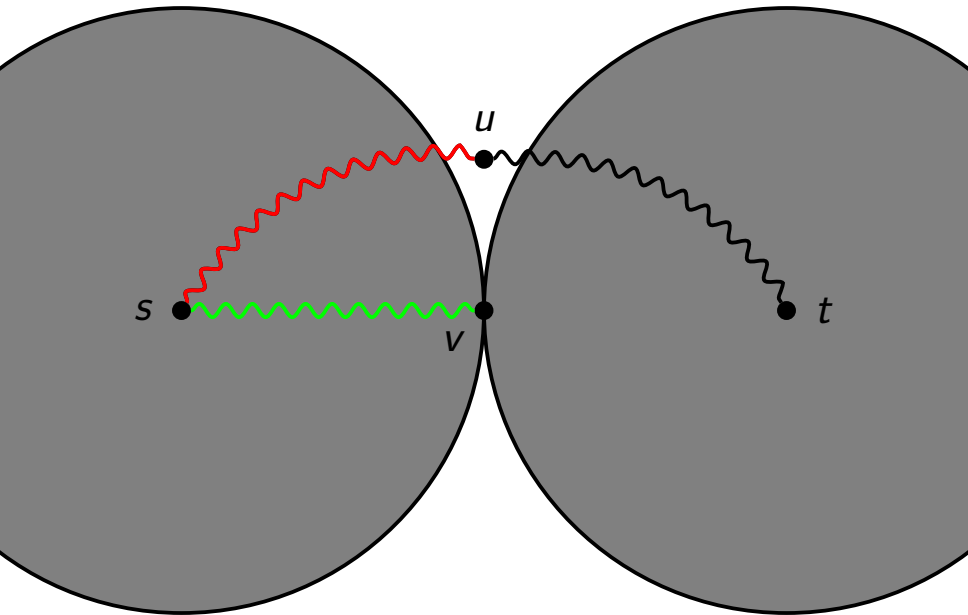
# Proof



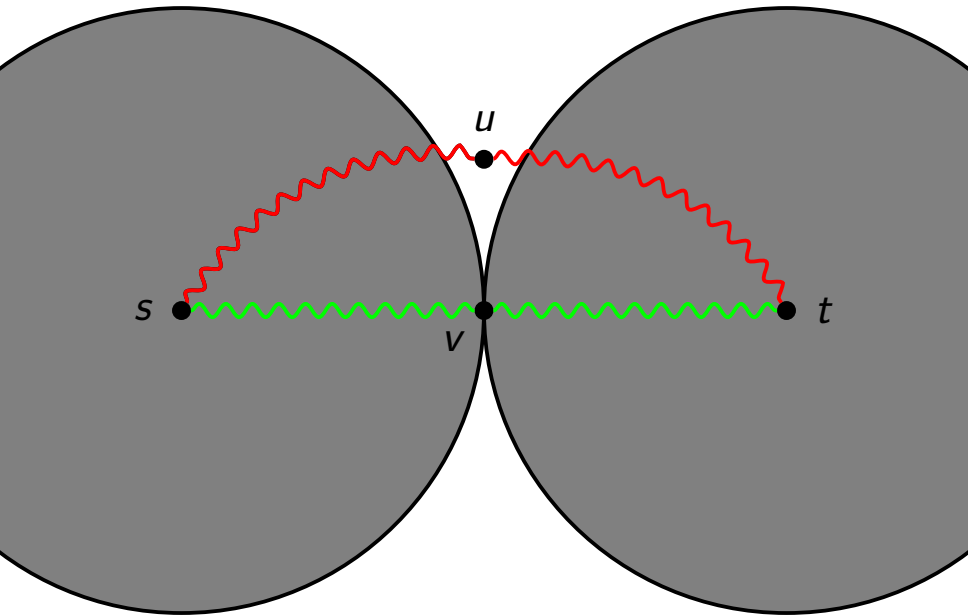
# Proof



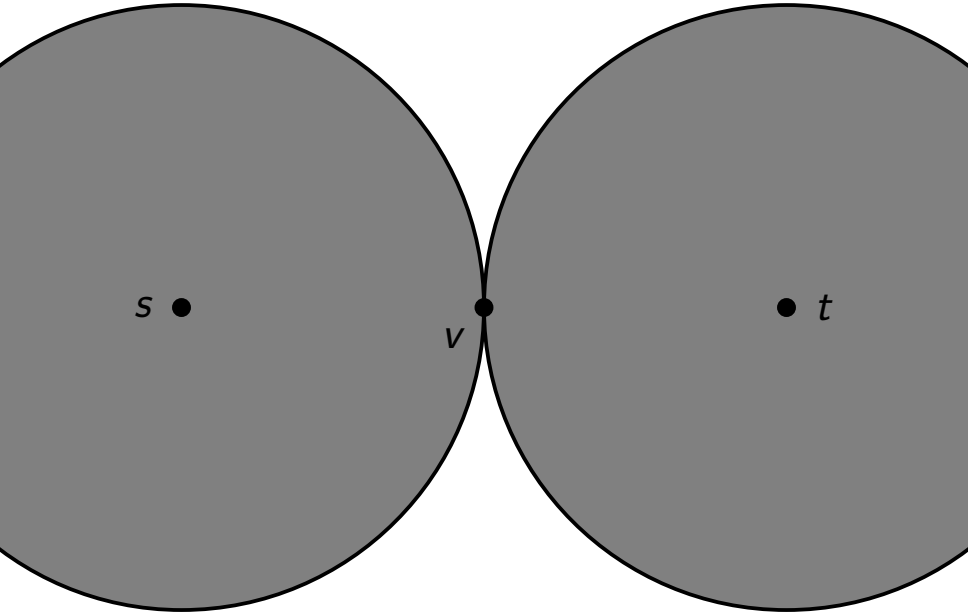
# Proof



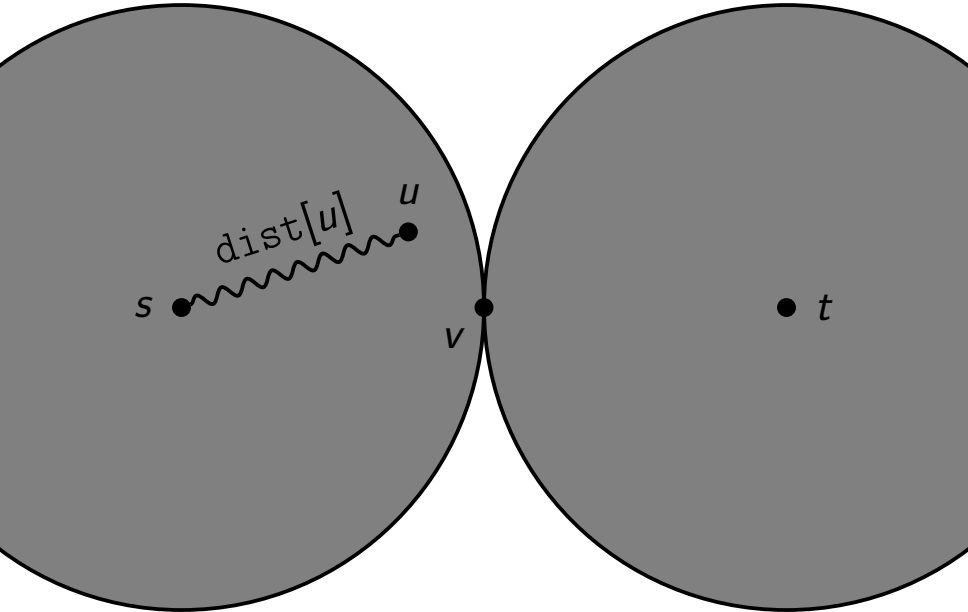
# Proof



# Proof

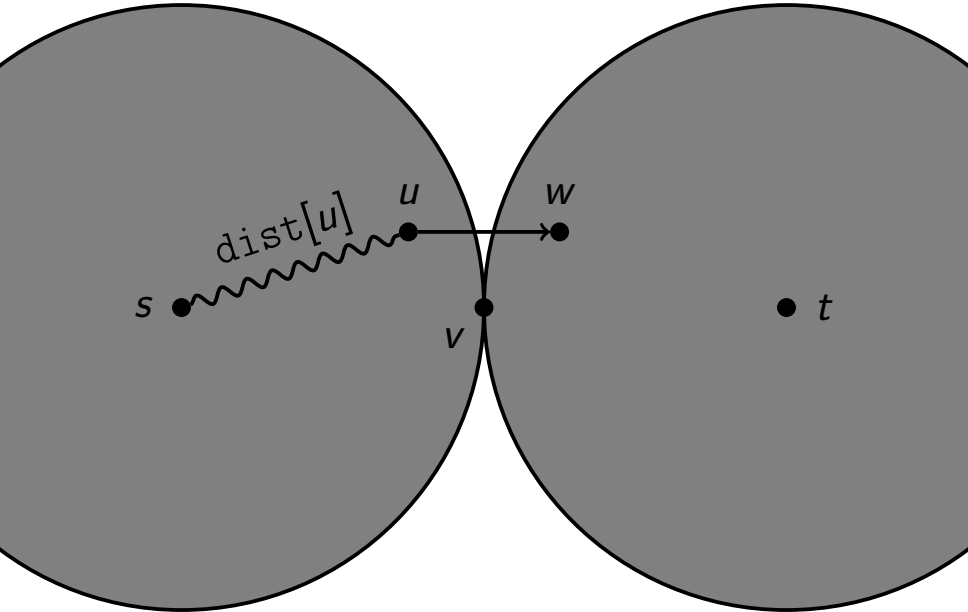


# Proof

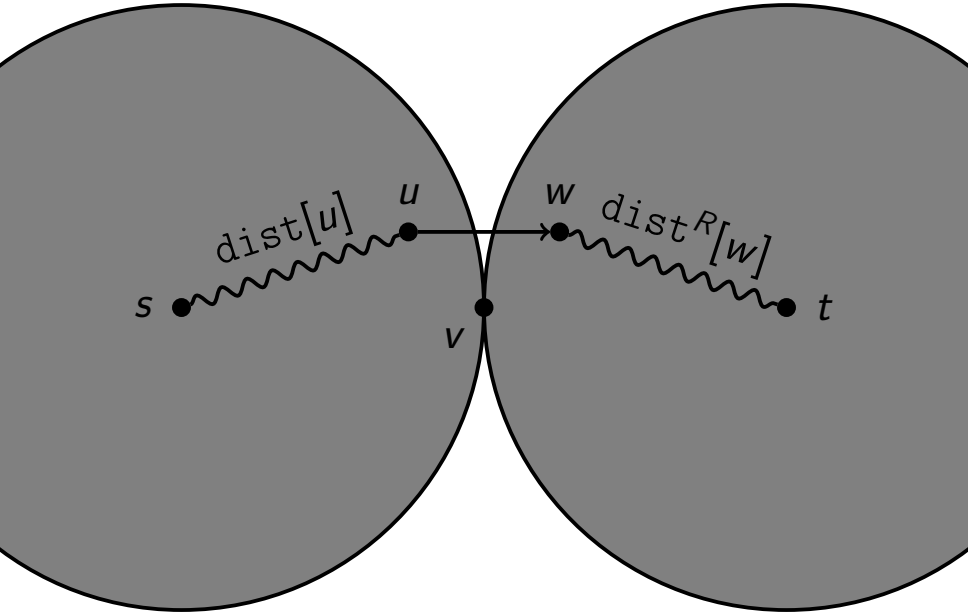




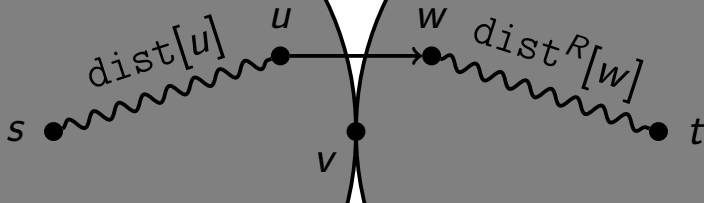
# Proof



# Proof

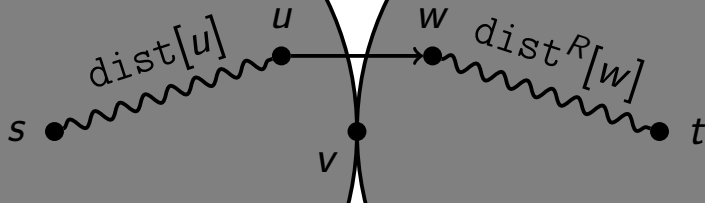


# Proof



$$d(s, t) = \text{dist}[u] + l(u, w) + \text{dist}^R[w]$$

# Proof



$$d(s, t) = \text{dist}[u] + l(u, w) + \text{dist}^R[w] =$$

$$= \text{dist}[u] + \text{dist}^R[u]$$

## BidirectionalDijkstra( $G, s, t$ )

```
 $G^R \leftarrow \text{ReverseGraph}(G)$ 
Fill  $\text{dist}, \text{dist}^R$  with  $+\infty$  for each node
 $\text{dist}[s] \leftarrow 0, \text{dist}^R[t] \leftarrow 0$ 
Fill  $\text{prev}, \text{prev}^R$  with None for each node
 $\text{proc} \leftarrow \text{empty}, \text{proc}^R \leftarrow \text{empty}$ 
do:
     $v \leftarrow \text{ExtractMin}(\text{dist})$ 
    Process( $v, G, \text{dist}, \text{prev}, \text{proc}$ )
    if  $v$  in  $\text{proc}^R$ :
        return ShortestPath( $s, \text{dist}, \text{prev}, \text{proc}, t, \dots$ )
     $v^R \leftarrow \text{ExtractMin}(\text{dist}^R)$ 
    repeat symmetrically for  $v^R$  as for  $v$ 
while True
```

Relax( $u, v, \text{dist}, \text{prev}$ )

```
if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
     $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
     $\text{prev}[v] \leftarrow u$ 
```

Process( $u, G, \text{dist}, \text{prev}, \text{proc}$ )

for  $(u, v) \in E(G)$ :

    Relax( $u, v, \text{dist}, \text{prev}$ )

proc.Append( $u$ )

ShortestPath( $s, \text{dist}, \text{prev}, \text{proc}, t, \text{dist}^R, \text{prev}^R, \text{proc}^R$ )

```
 $distance \leftarrow +\infty, u_{best} \leftarrow \text{None}$   
for  $u$  in  $\text{proc} + \text{proc}^R$ :  
    if  $\text{dist}[u] + \text{dist}^R[u] < distance$ :  
         $u_{best} \leftarrow u$   
         $distance \leftarrow \text{dist}[u] + \text{dist}^R[u]$   
 $path \leftarrow \text{empty}$   
 $last \leftarrow u_{best}$   
while  $last \neq s$ :  
     $path.\text{Append}(last)$   
     $last \leftarrow \text{prev}[last]$   
 $path \leftarrow \text{Reverse}(path)$   
 $last \leftarrow u_{best}$   
while  $last \neq t$ :  
     $last \leftarrow \text{prev}^R[last]$   
     $path.\text{Append}(last)$   
return ( $distance, path$ )
```



# Conclusion

- Worst-case running time of Bidirectional Dijkstra is the same as for Dijkstra
- Speedup in practice depends on the graph
- Memory consumption is 2x to store  $G$  and  $G^R$
- You'll see the speedup on social network graph in the Programming Assignment