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.

Graph can be directional or undirectional

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

Can solve the shortest path problem using the undirectional graphs in "linear time". Such linear time Algo for directional graph is not known we can do better compared to conventional Djikstra's running time

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

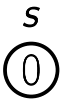
Why not just Dijkstra?

- $O((|E| + |V|) \log |V|)$ 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

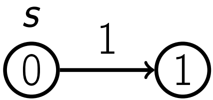
Why not just Dijkstra?

- $O((|E| + |V|) \log |V|)$ 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

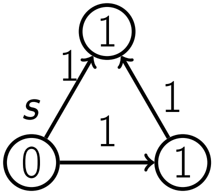
Dijkstra Progression



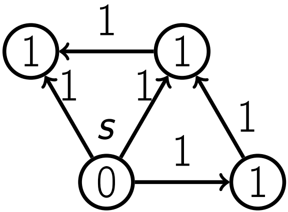
Dijkstra Progression



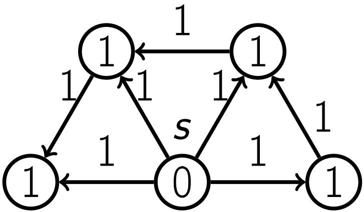
Dijkstra Progression



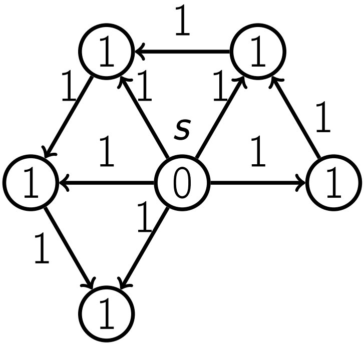
Dijkstra Progression



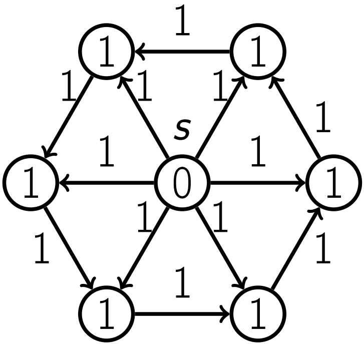
Dijkstra Progression



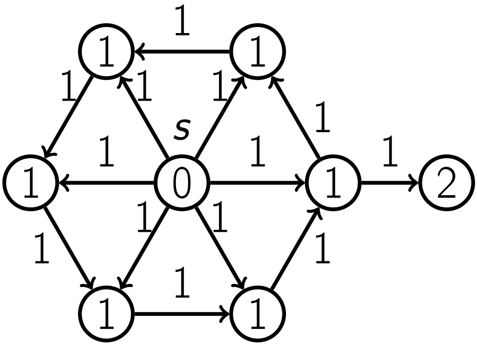
Dijkstra Progression



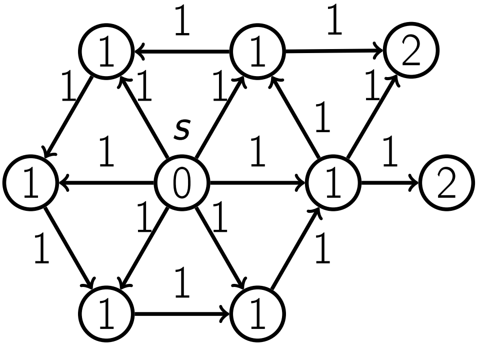
Dijkstra Progression



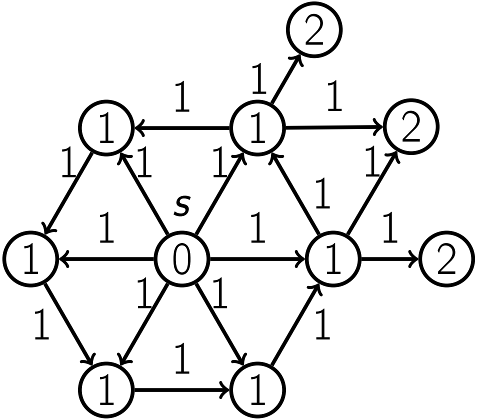
Dijkstra Progression



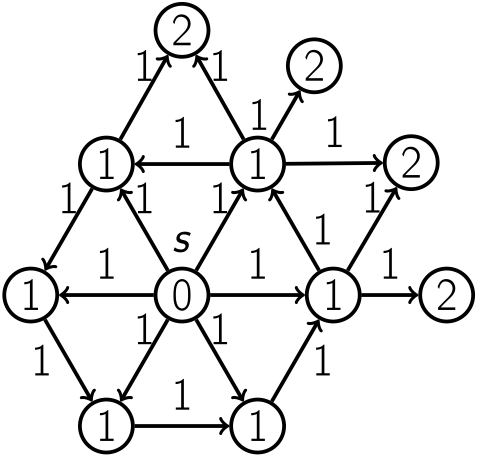
Dijkstra Progression



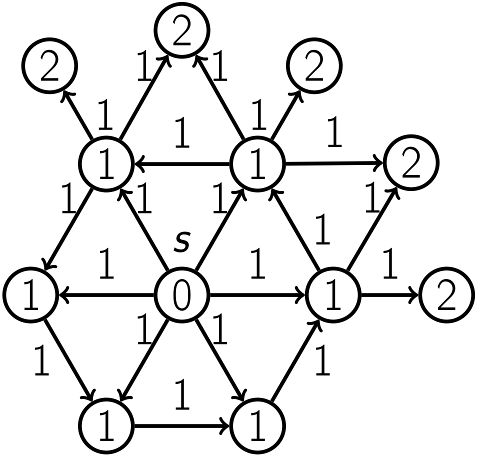
Dijkstra Progression



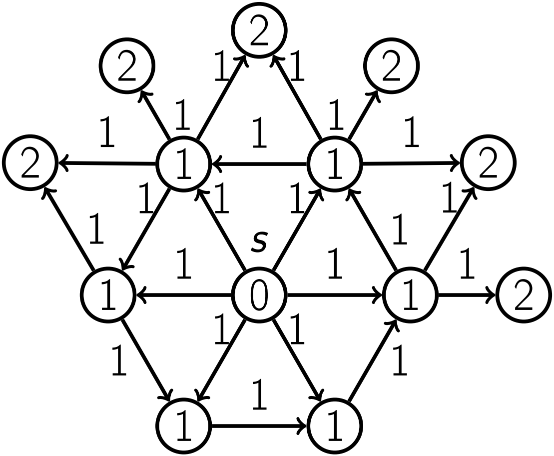
Dijkstra Progression



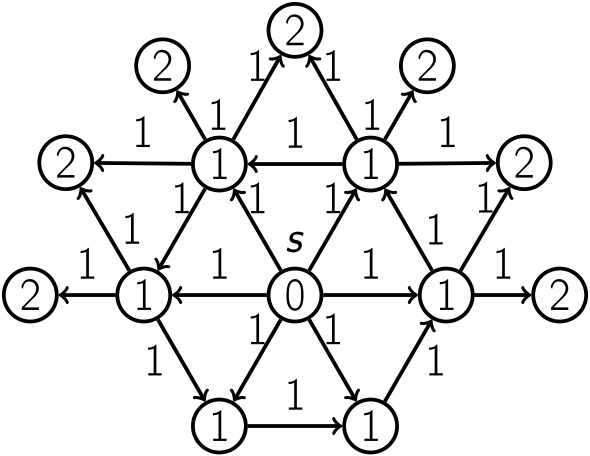
Dijkstra Progression



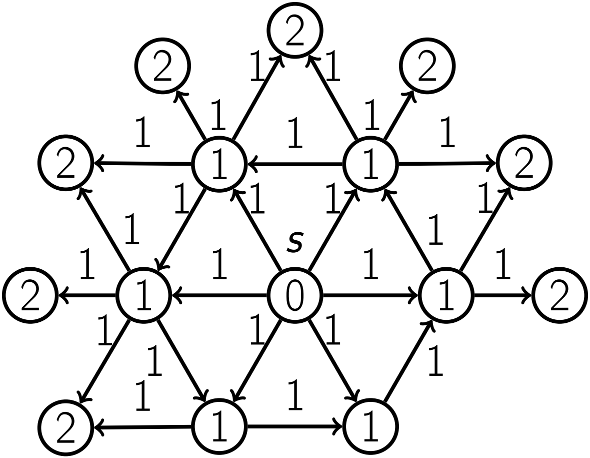
Dijkstra Progression



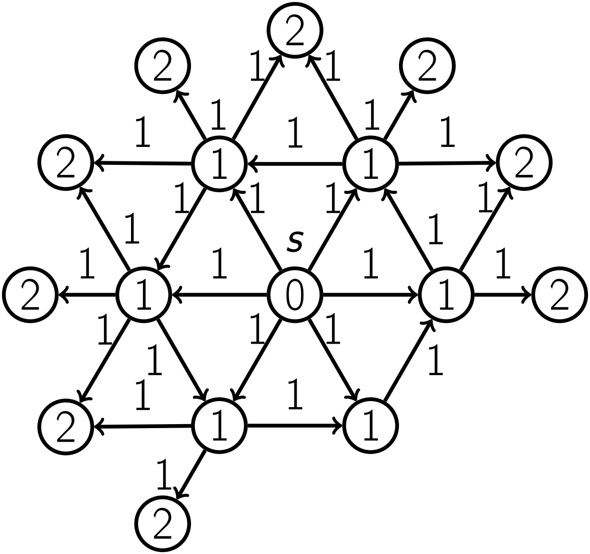
Dijkstra Progression



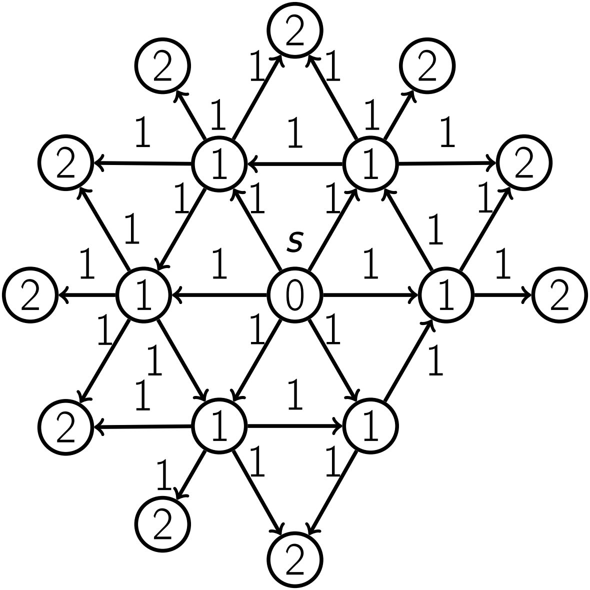
Dijkstra Progression



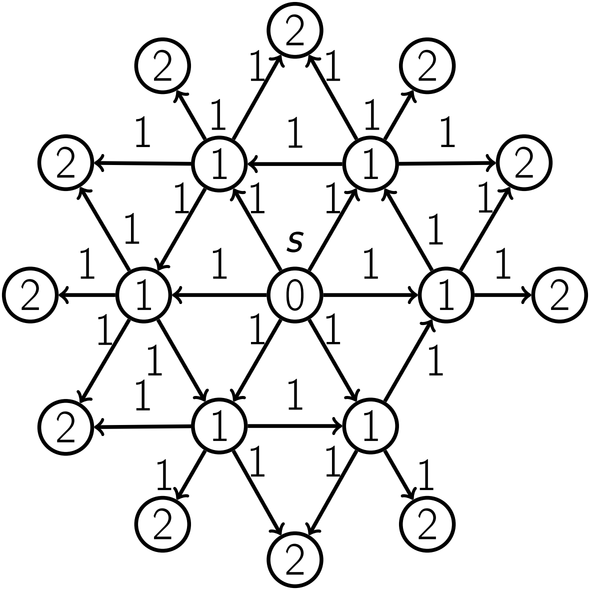
Dijkstra Progression



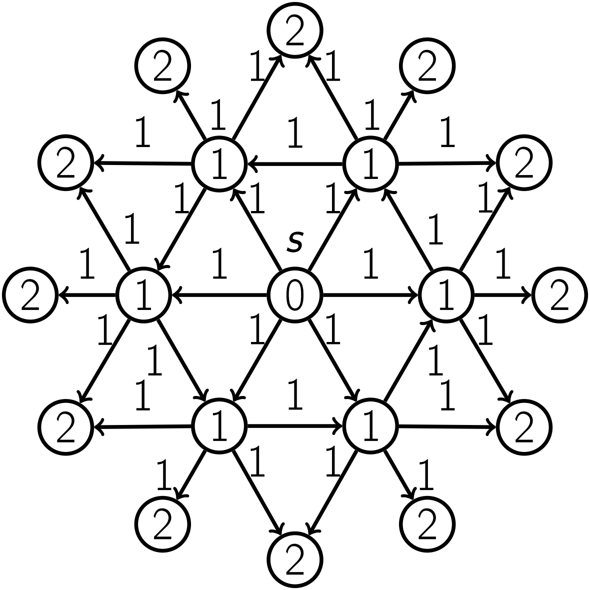
Dijkstra Progression



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

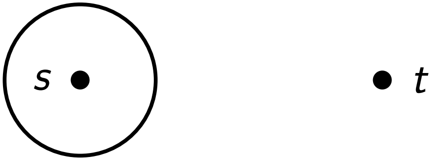
When a vertex u is selected via `ExtractMin`, $\mathrm{dist}[u] = d(s, u)$.

- 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

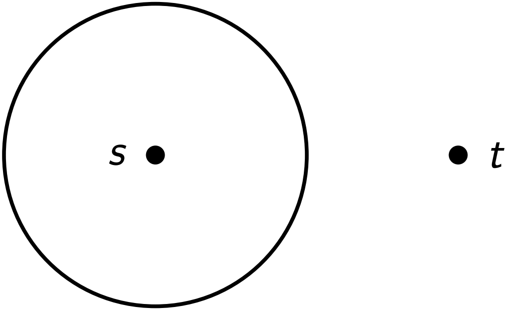
Idea: Growing Circle

Idea: Growing Circle

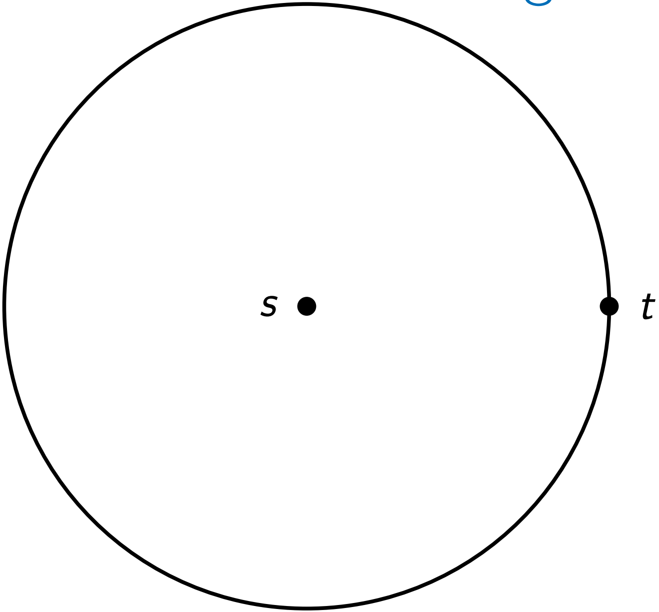
Idea: Growing Circle



Idea: Growing Circle



Idea: Growing Circle



This Indicates calculating distance only till t. Returning as soon as t is found

Idea: Bidirectional Search

Idea: Bidirectional Search

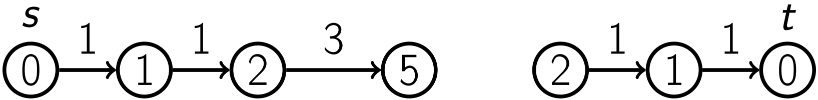
Instead of growing from s to t, Grow from both the directions. From s grow forwards and from t grow backwards. And as soon as common node is found stop the algo. In Algo we cannot grow paths from s and t simultaneously hence we will alternate growing paths from s and t

Idea: Bidirectional Search

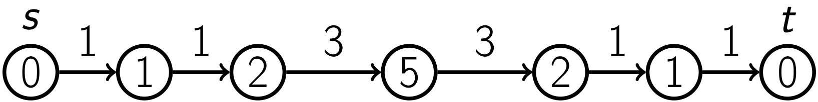
Idea: Bidirectional Search

Idea: Bidirectional Search

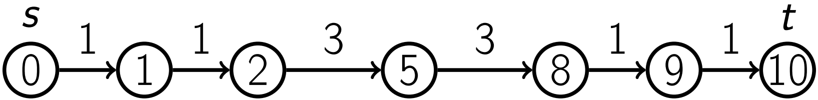
Idea: Bidirectional Search



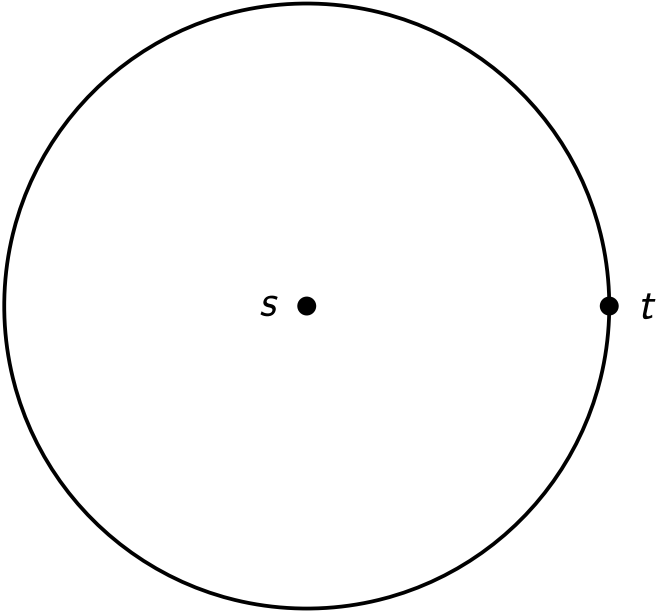
Idea: Bidirectional Search



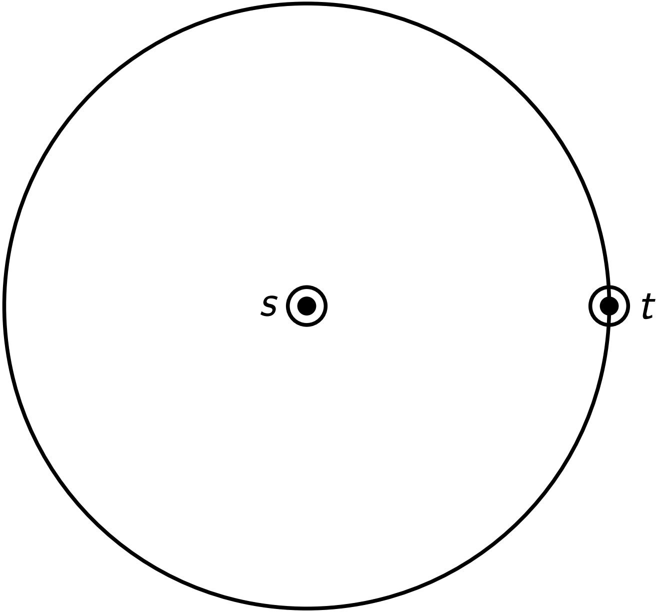
Idea: Bidirectional Search



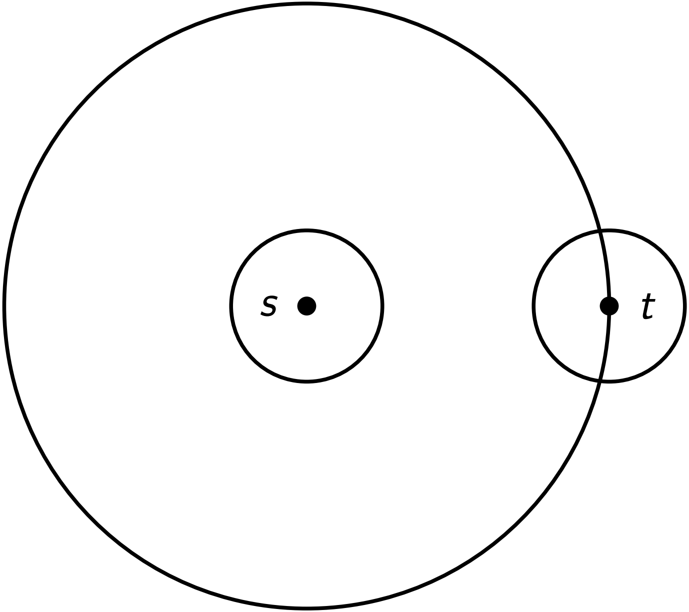
Idea: Bidirectional Search



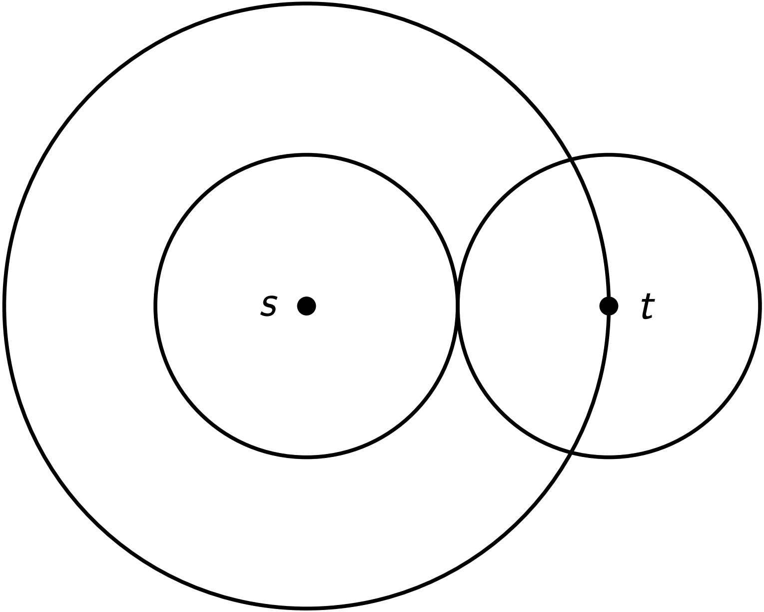
Idea: Bidirectional Search



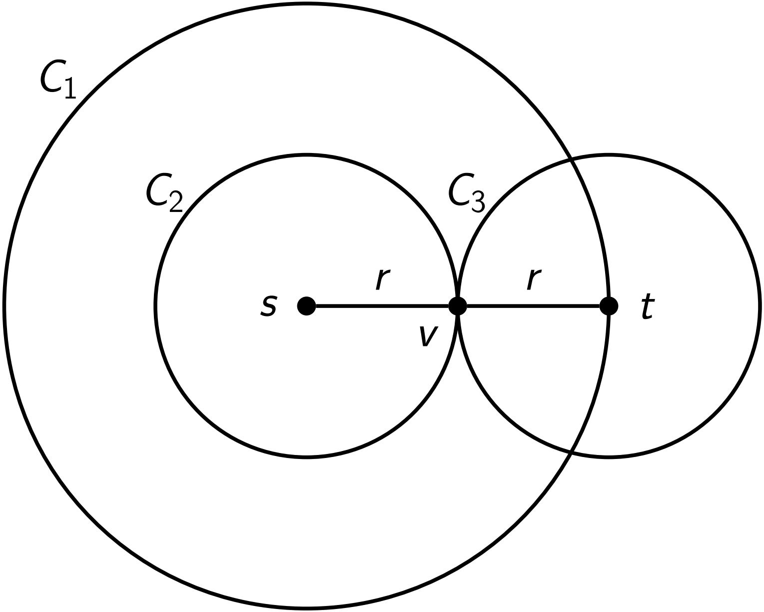
Idea: Bidirectional Search



Idea: Bidirectional Search



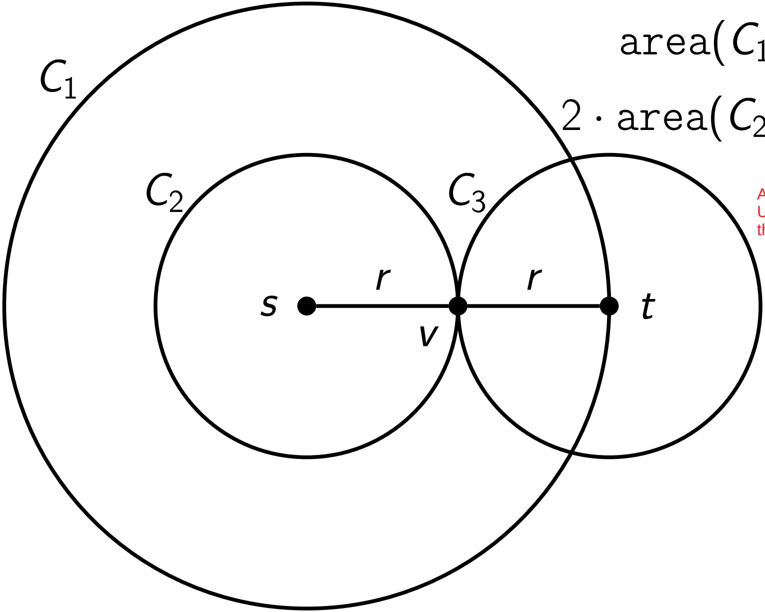
Idea: Bidirectional Search



Idea: Bidirectional Search

$$\texttt{area}(C_1) = 4\pi r^2$$

$$2 \cdot \texttt{area}(C_2) = 2\pi r^2$$



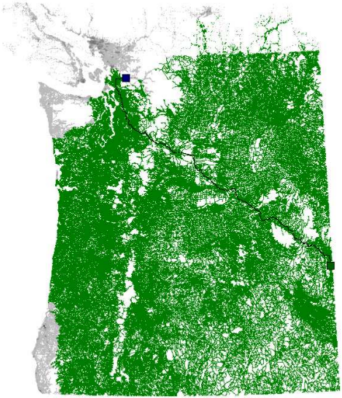
Area covered (Nodes) by Unidirectional twice to that of Bidirectional

Road networks

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

Picture by Andrew Goldberg.

Road networks

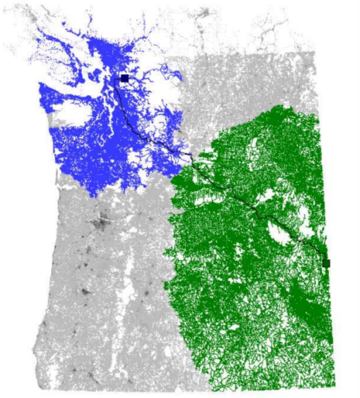


Unidirectional Djikstra's

Searched area

Picture by Andrew Goldberg.

Road networks

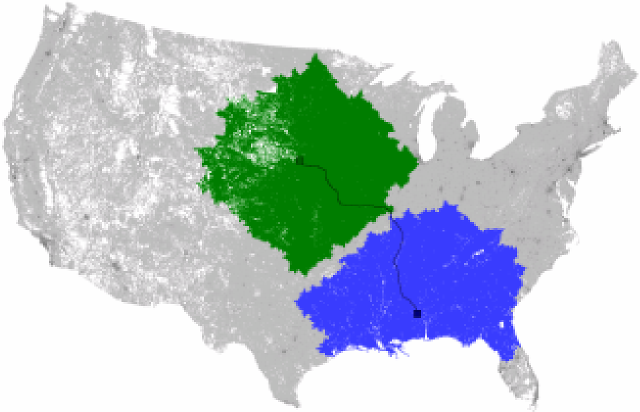


forward search/ reverse search

Bidirectional Djikstra's

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 Karinthy made a “Small World” conjecture

Six Handshakes

- In 1929, Hungarian mathematician Frigyes Karinthy 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 Karinthy 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 Middle Applicable on social network or many other problems and not roads

- 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 $\texttt{dist}[s]$ to 0, all other distances to ∞

Dijkstra Reminder

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

Dijkstra Reminder

- To find the shortest path from s to t
- Initialize `dist`[s] to 0, all other distances to ∞
- `ExtractMin` — choose *unprocessed* u with the smallest `dist`[u]
- Process u — `Relax` the edges outgoing from u

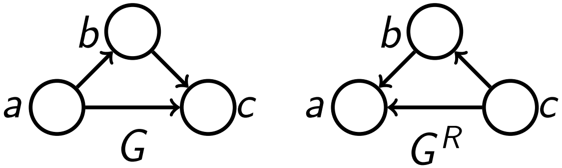
Dijkstra Reminder

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

Question

Let v be the first vertex which is processed both in the forward search from s in G and in the backward search from t in G^R in the process of Bidirectional Dijkstra algorithm launched to find the shortest path between s and t in graph G. Can we guarantee then that there exists a shortest path from s to t that goes through v?

- ○ No
- ◉ Yes

✗ **Incorrect**
See the counterexample in the lecture.

Skip Retry

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

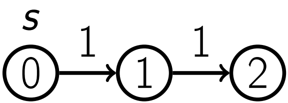
Let v be the first vertex which is processed both in G and in G^R. Does it follow that there is a shortest path from s to t going through v?

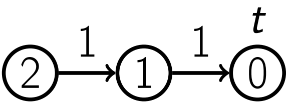
Computing Distance

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

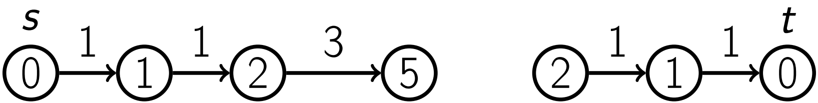
Let v be the first vertex which is processed both in G and in G^R. Does it follow that there is a shortest path from s to t going through v?





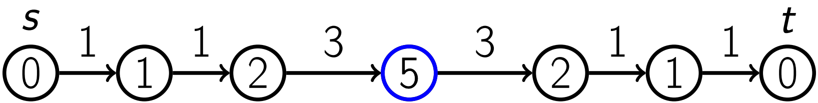
Computing Distance

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?



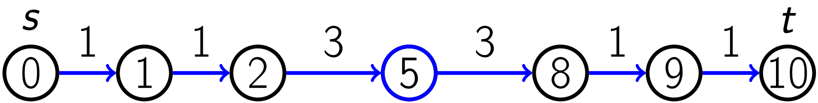
Computing Distance

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?



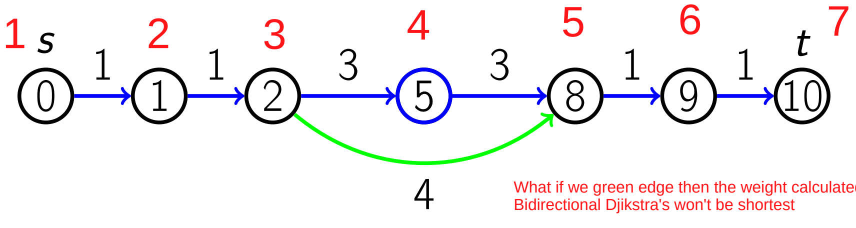
Computing Distance

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

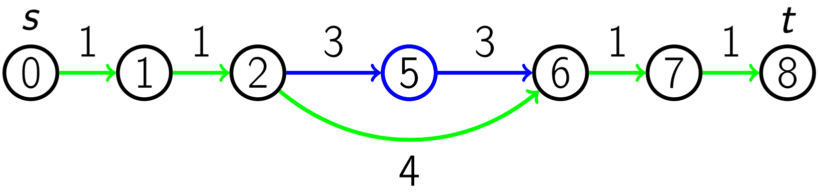
Let v be the first vertex which is processed both in G and in G^R. Does it follow that there is a shortest path from s to t going through v?



What if we green edge then the weight calculated with Bidirectional Djikstra's won't be shortest

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?

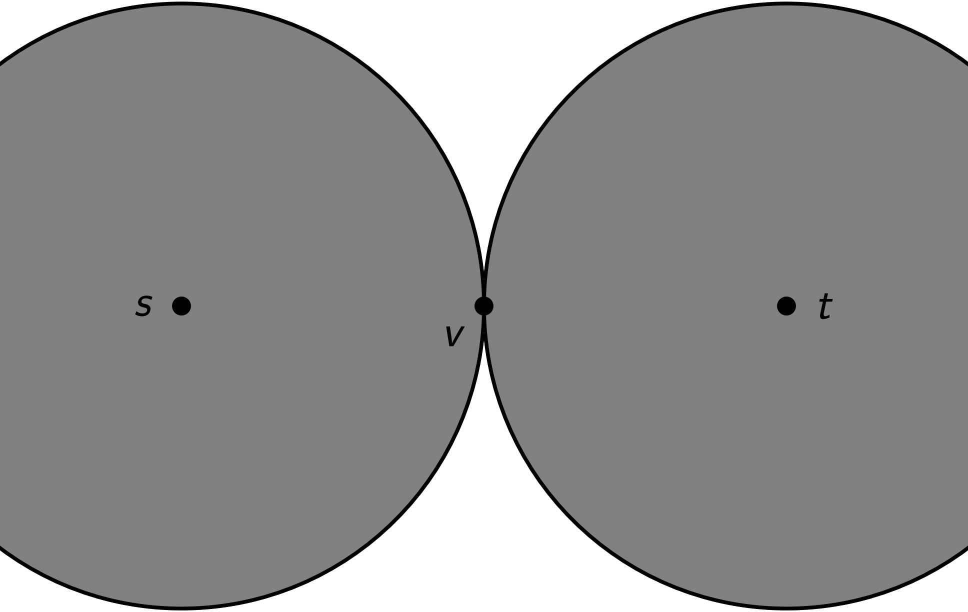


Computing Distance

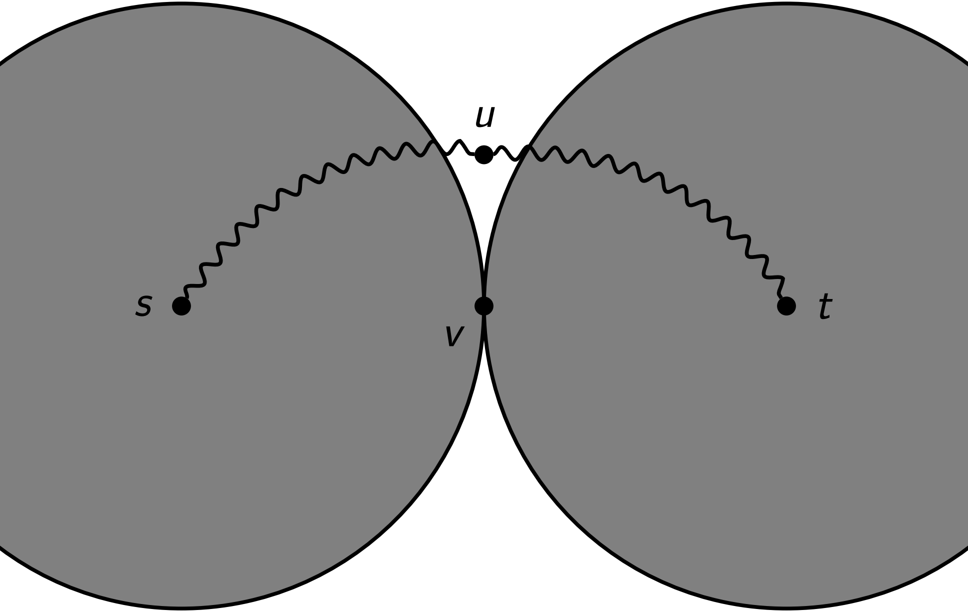
Lemma

Let $\texttt{dist}[u]$ be the distance estimate in the forward Dijkstra from s in G and $\texttt{dist}^R[u]$ — 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 $d(s,t) = \texttt{dist}[u] + \texttt{dist}^R[u]$.

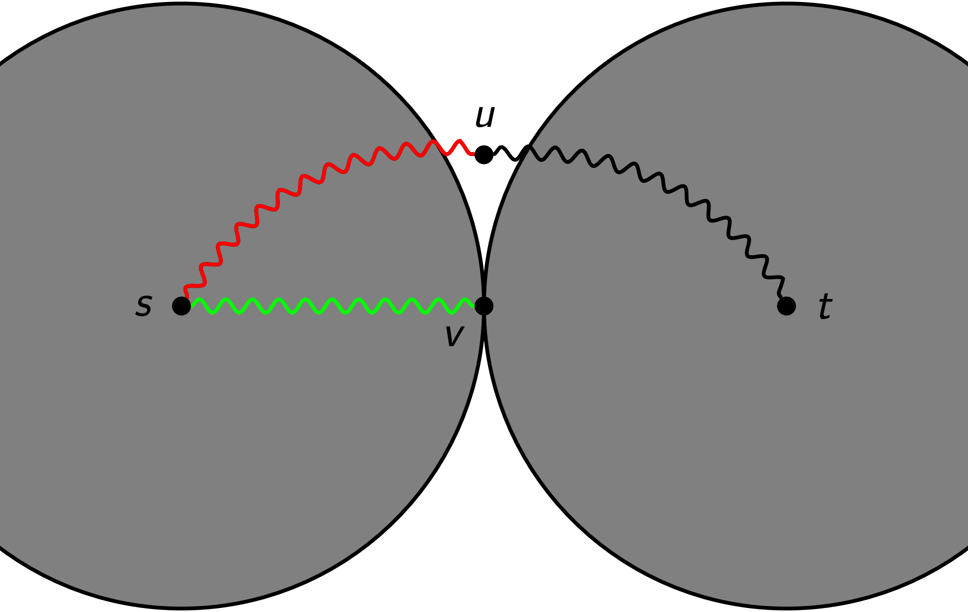
Proof



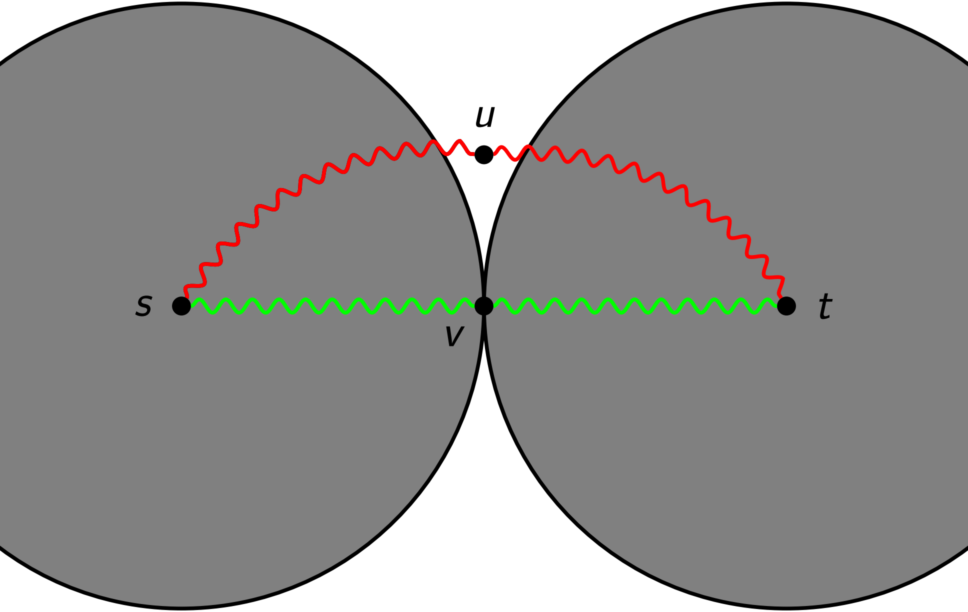
Proof



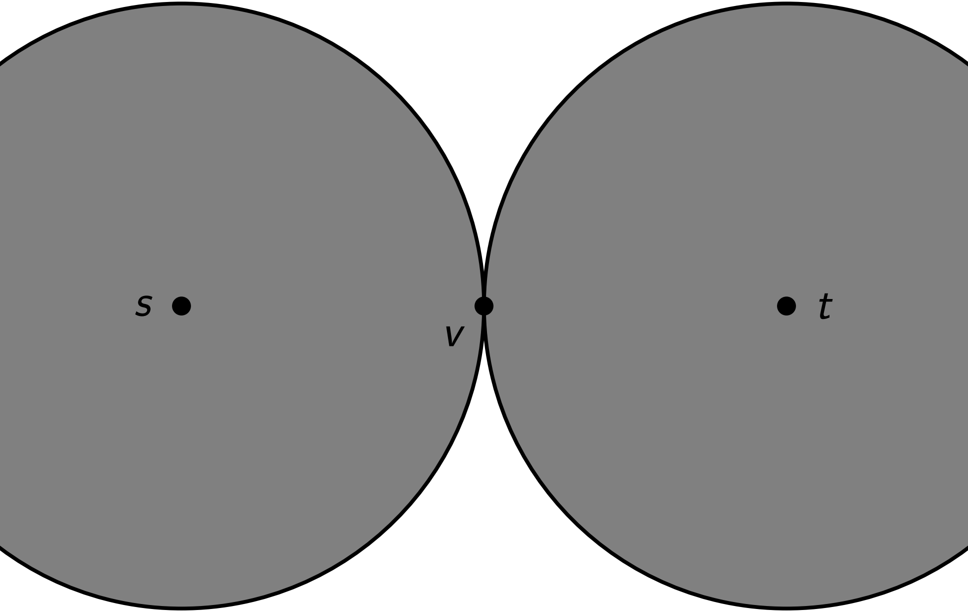
Proof



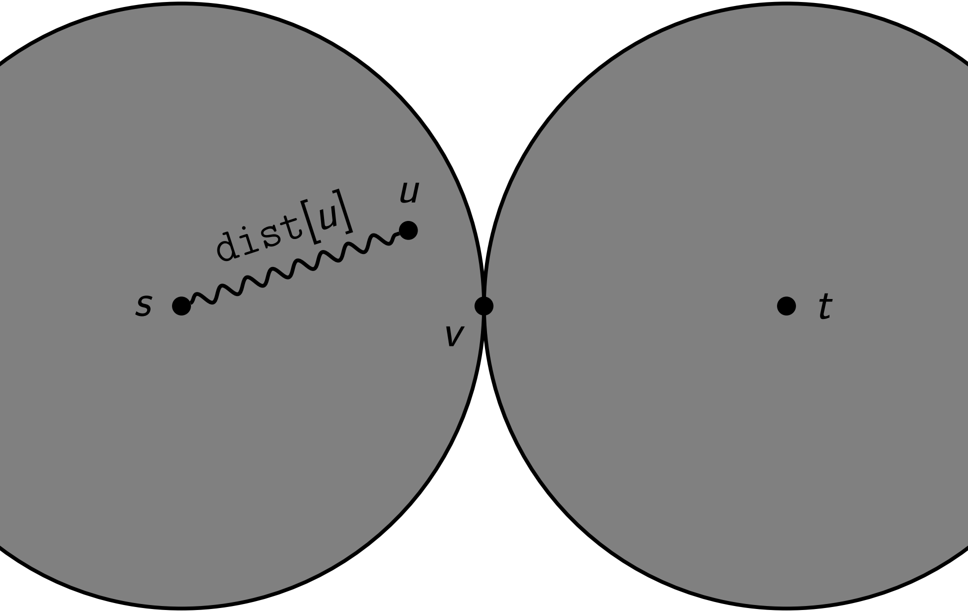
Proof



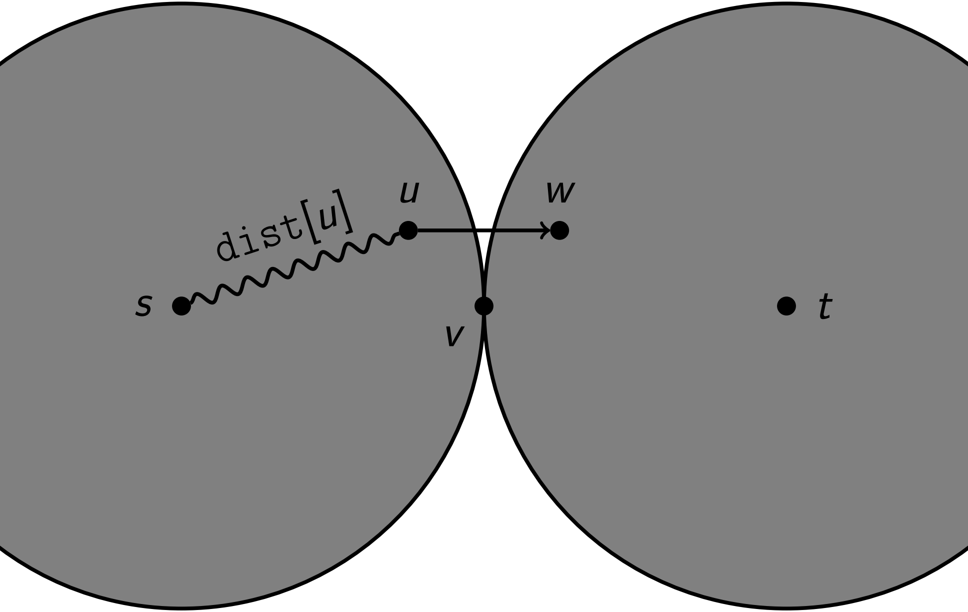
Proof



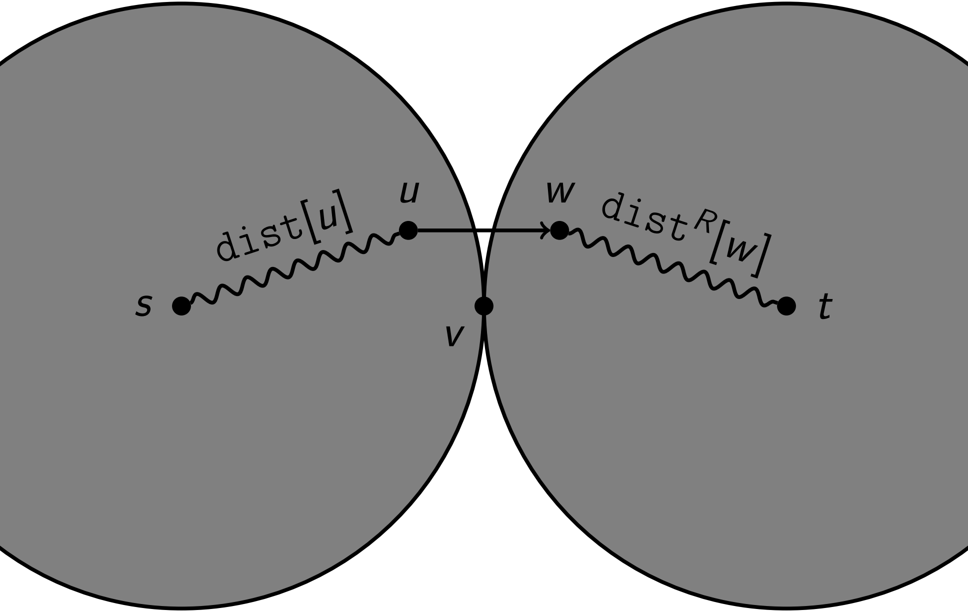
Proof



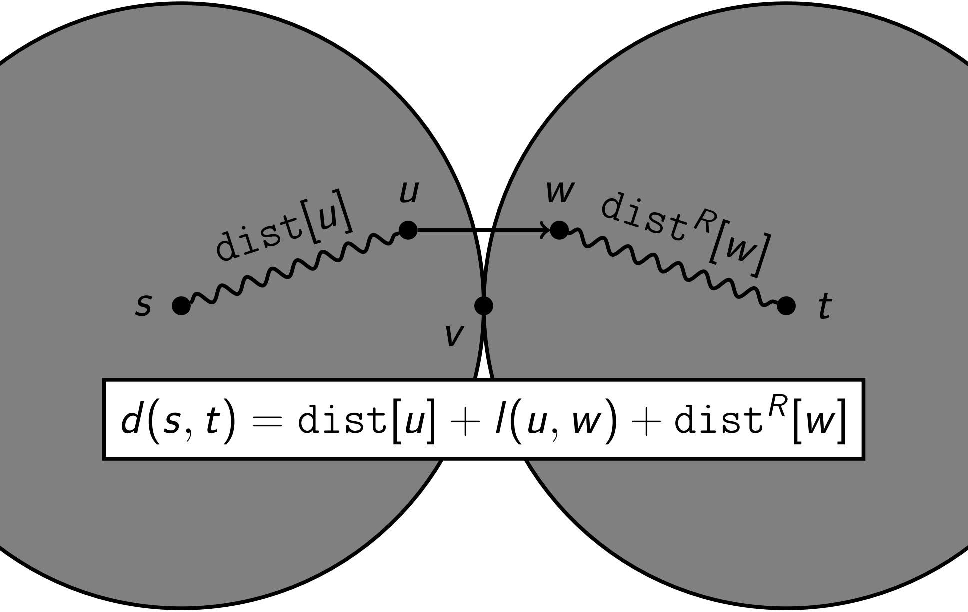
Proof



Proof

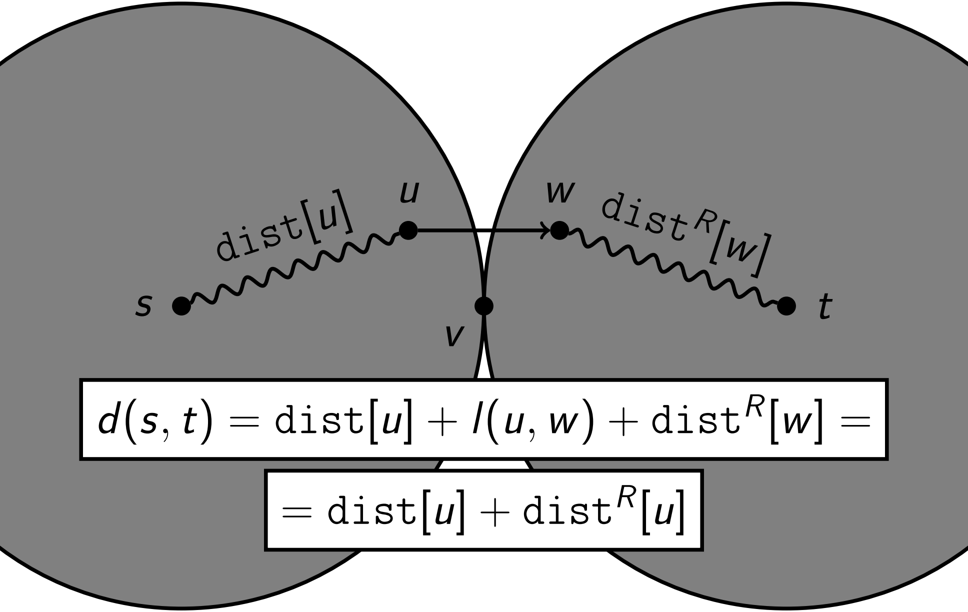


Proof



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

Proof



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

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

BidirectionalDijkstra(G, s, t)

```
G^R ← ReverseGraph(G)
Fill dist, dist^R with +∞ for each node
dist[s] ← 0, dist^R[t] ← 0
Fill prev, prev^R with None for each node
proc ← empty, proc^R ← empty
do:
  v ← ExtractMin(dist)
  Process(v, G, dist, prev, proc)
  if v in proc^R:
    return ShortestPath(s, dist, prev, proc, t, ...)
  v^R ← ExtractMin(dist^R)
  repeat symmetrically for v^R as for v
while True
```

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

$\texttt{if dist}[v] > \texttt{dist}[u] + w(u, v)$:

$\quad \texttt{dist}[v] \leftarrow \texttt{dist}[u] + w(u, v)$

$\quad \texttt{prev}[v] \leftarrow u$

```
Process(u, G, dist, prev, proc)

for (u, v) ∈ E(G):
  Relax(u, v, dist, prev)
proc.Append(u)
```

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

```
distance ← +∞,  u_best ← None
for u in proc + proc^R:
    if dist[u] + dist^R[u] < distance:
        u_best ← u
        distance ← dist[u] + dist^R[u]
path ← empty
last ← u_best
while last ≠ s:
    path.Append(last)
    last ← prev[last]
path ← Reverse(path)
last ← u_best
while last ≠ t:
    last ← prev^R[last]
    path.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 For Road Networks 2X but for social networks it can be 1000 times
- Memory consumption is 2x to store G and G^R
- You'll see the speedup on social network graph in the Programming Assignment