COMP 761: Lecture 25 – Graph Algorithms II

David Rolnick

November 2, 2020

Problem

Express finding the shortest path in a graph as a linear program.

(Please don't post your ideas in the chat just yet, we'll discuss the problem soon in class.)

Course Announcements

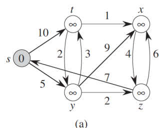
Course Announcements

- Office hours right after class!

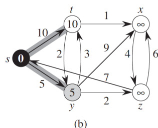
Course Announcements

- Office hours right after class!
- Problem set: can use Strong Duality without proof since we stated it in class.

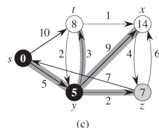
Dijkstra's algorithm



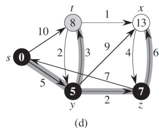
(a)



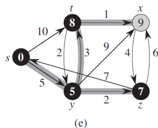
(b)



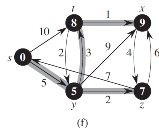
(c)



(d)

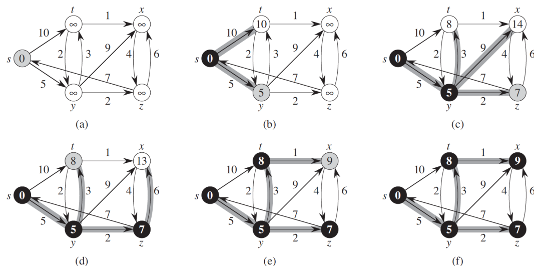


(e)



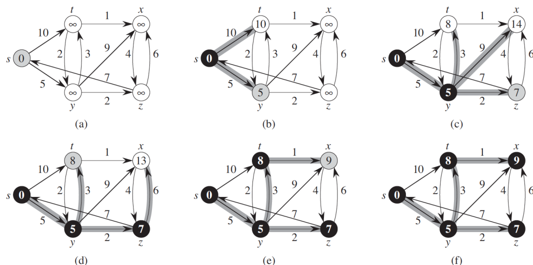
(f)

Dijkstra's algorithm



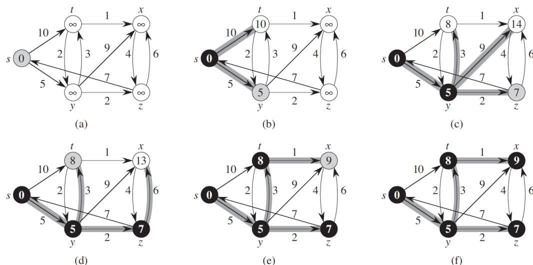
- Maintain guesses for distances of all vertices from s.

Dijkstra's algorithm



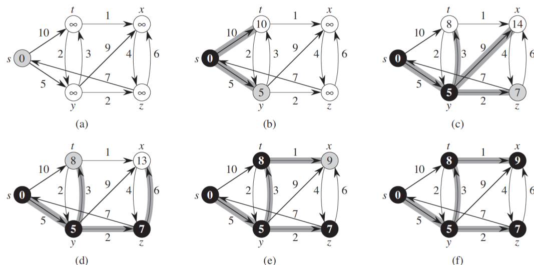
- Maintain guesses for distances of all vertices from s.
- Start out with ∞ everywhere, s at distance 0 from itself.

Dijkstra's algorithm



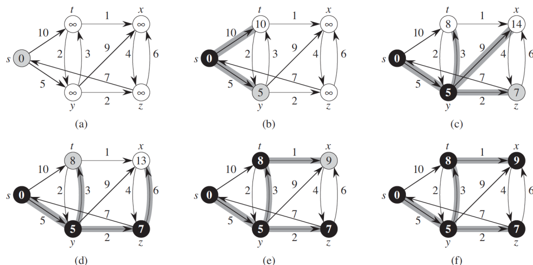
- Maintain guesses for distances of all vertices from s.
- Start out with ∞ everywhere, s at distance 0 from itself.
- At each step, pick unvisited vertex i with smallest distance estimate (starting with s) - this estimate has to be correct.

Dijkstra's algorithm



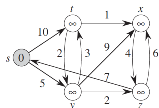
- Maintain guesses for distances of all vertices from s.
- Start out with ∞ everywhere, s at distance 0 from itself.
- At each step, pick unvisited vertex i with smallest distance estimate (starting with s) - this estimate has to be correct.
- Visit all its neighbors j, and update distance estimate $d(j)$ to $\min(d(j), d(i) + w_{ij})$.

Dijkstra's algorithm

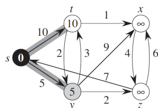


- Maintain guesses for distances of all vertices from s.
- Start out with ∞ everywhere, s at distance 0 from itself.
- At each step, pick unvisited vertex i with smallest distance estimate (starting with s) - this estimate has to be correct.
- Visit all its neighbors j, and update distance estimate $d(j)$ to $\min(d(j), d(i) + w_{ij})$.
- Repeat.

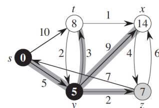
Proof of correctness (outline)



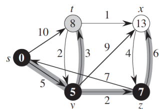
(a)



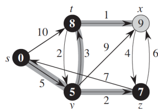
(b)



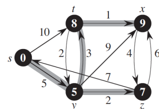
(c)



(d)

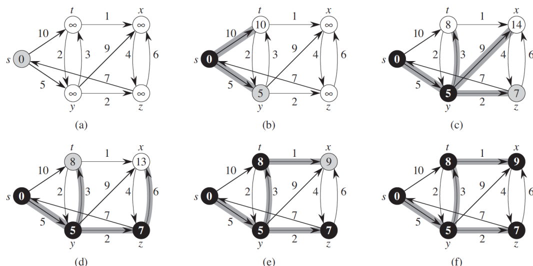


(e)



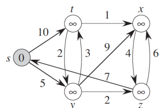
(f)

Proof of correctness (outline)

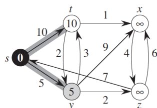


- What general technique might we use?

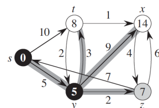
Proof of correctness (outline)



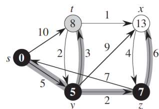
(a)



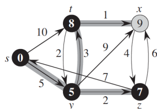
(b)



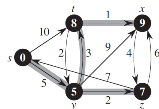
(c)



(d)



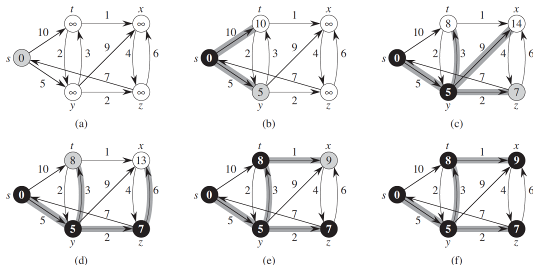
(e)



(f)

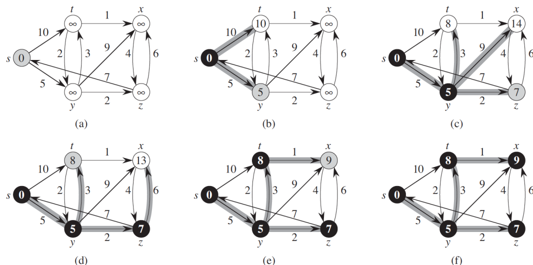
- Let’s use contradiction.

Proof of correctness (outline)



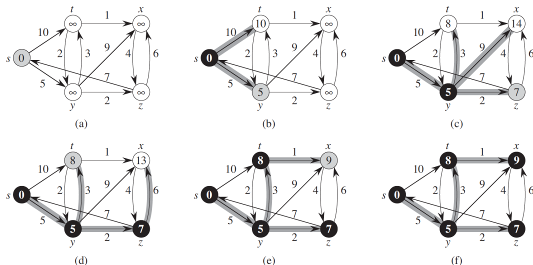
- Let's use contradiction.
- Suppose that $f(v)$ is our estimate of the distance from s to vertices v.

Proof of correctness (outline)



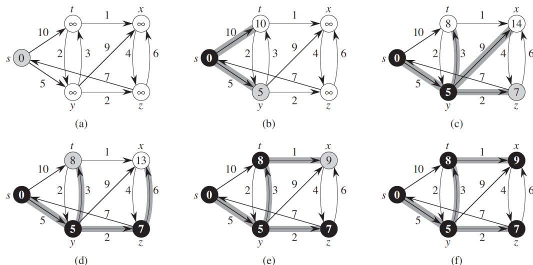
- Let's use contradiction.
- Suppose that $f(v)$ is our estimate of the distance from s to vertices v.
- Then, $f(v) \geq d(s, v)$.

Proof of correctness (outline)



- Let's use contradiction.
- Suppose that $f(v)$ is our estimate of the distance from s to vertices v.
- Then, $f(v) \geq d(s, v)$.
- Suppose when we visit some new vertex our estimated distance is actually not correct.

Proof of correctness (outline)



- Let's use contradiction.
- Suppose that $f(v)$ is our estimate of the distance from s to vertices v.
- Then, $f(v) \geq d(s, v)$.
- Suppose when we visit some new vertex our estimated distance is actually not correct.
- Let u be the first one of these bad vertices we visit, so $f(u) > d(s, u)$.

Proof of correctness (outline)

Proof of correctness (outline)

- Let S be the set of vertices we have visited before u.

Proof of correctness (outline)

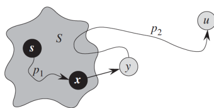
- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.

Proof of correctness (outline)

- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.

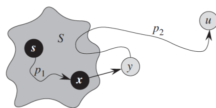
Proof of correctness (outline)

- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.
- Let x be the last vertex in the path that's in S, and y be the vertex after that.



Proof of correctness (outline)

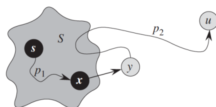
- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.
- Let x be the last vertex in the path that's in S, and y be the vertex after that.



- The path up through y must be the shortest path from s to y, or else we could make the path to u shorter.

Proof of correctness (outline)

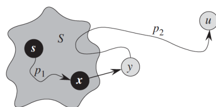
- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.
- Let x be the last vertex in the path that's in S, and y be the vertex after that.



- The path up through y must be the shortest path from s to y, or else we could make the path to u shorter.
- So $d(s, y) = d(s, x) + w_{xy} = f(y)$, where w_{xy} is the edge weight.

Proof of correctness (outline)

- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.
- Let x be the last vertex in the path that's in S, and y be the vertex after that.

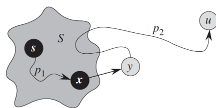


- The path up through y must be the shortest path from s to y, or else we could make the path to u shorter.
- So $d(s, y) = d(s, x) + w_{xy} = f(y)$, where w_{xy} is the edge weight.
- Therefore,

$$f(u) > d(s, u) \geq d(s, y) = f(y).$$

Proof of correctness (outline)

- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.
- Let x be the last vertex in the path that's in S, and y be the vertex after that.

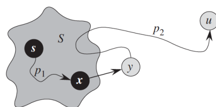


- The path up through y must be the shortest path from s to y, or else we could make the path to u shorter.
- So $d(s, y) = d(s, x) + w_{xy} = f(y)$, where w_{xy} is the edge weight.
- Therefore,

$$f(u) > d(s, u) \geq d(s, y) = f(y).$$

Proof of correctness (outline)

- Let S be the set of vertices we have visited before u.
- There is some shortest path between s and u.
- s is in S, so this path starts out in S.
- Let x be the last vertex in the path that's in S, and y be the vertex after that.



- The path up through y must be the shortest path from s to y, or else we could make the path to u shorter.
- So $d(s, y) = d(s, x) + w_{xy} = f(y)$, where w_{xy} is the edge weight.
- Therefore,

$$f(u) > d(s, u) \geq d(s, y) = f(y).$$

- So we should have picked y instead of u – contradiction.

Dijkstra's algorithm

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?
- Let's work out what the components are.

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?
- Let's work out what the components are.
- Work out the shortest distance from s to any of the unvisited vertices.

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?
- Let's work out what the components are.
- Work out the shortest distance from s to any of the unvisited vertices.
- Go through each of its neighbors and decrease some of their distances.

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?
- Let's work out what the components are.
- Work out the shortest distance from s to any of the unvisited vertices.
- Go through each of its neighbors and decrease some of their distances.
- Remove the vertex from the list of unvisited vertices.

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?
- Let's work out what the components are.
- Work out the shortest distance from s to any of the unvisited vertices.
- Go through each of its neighbors and decrease some of their distances.
- Remove the vertex from the list of unvisited vertices.
- How can we implement these operations fast? (storing some values, sometimes decreasing them, finding a minimum fast at any time, removing values one-by-one)

Dijkstra's algorithm

- How long does Dijkstra's algorithm take?
- Let's work out what the components are.
- Work out the shortest distance from s to any of the unvisited vertices.
- Go through each of its neighbors and decrease some of their distances.
- Remove the vertex from the list of unvisited vertices.
- How can we implement these operations fast? (storing some values, sometimes decreasing them, finding a minimum fast at any time, removing values one-by-one)
- We can use a min-heap!

Dijkstra’s algorithm

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).
- For each of the neighbors, run DecreaseKey ($O(\log|V|)$ time each)

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).
- For each of the neighbors, run DecreaseKey ($O(\log |V|)$ time each)
- Run ExtractMin ($O(\log |V|)$ time).

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).
- For each of the neighbors, run DecreaseKey ($O(\log|V|)$ time each)
- Run ExtractMin ($O(\log|V|)$ time).
- Have to run Minimum and ExtractMin $|V|$ times, so $O(|V|\log|V|)$.

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).
- For each of the neighbors, run DecreaseKey ($O(\log |V|)$ time each)
- Run ExtractMin ($O(\log |V|)$ time).
- Have to run Minimum and ExtractMin $|V|$ times, so $O(|V| \log |V|)$.
- Have to run DecreaseKey $|E|$ times, so $O(|E| \log |V|)$.

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).
- For each of the neighbors, run DecreaseKey ($O(\log |V|)$ time each)
- Run ExtractMin ($O(\log |V|)$ time).
- Have to run Minimum and ExtractMin $|V|$ times, so $O(|V| \log |V|)$.
- Have to run DecreaseKey $|E|$ times, so $O(|E| \log |V|)$.
- Total time:

$$O((|V| + |E|) \log |V|) = O(|E| \log |V|).$$

Dijkstra's algorithm

- Here's the algorithm again in heap terms, if there are $|V|$ vertices and $|E|$ edges.
- At each step, run Minimum ($O(|V|)$ time).
- For each of the neighbors, run DecreaseKey ($O(\log |V|)$ time each)
- Run ExtractMin ($O(\log |V|)$ time).
- Have to run Minimum and ExtractMin $|V|$ times, so $O(|V| \log |V|)$.
- Have to run DecreaseKey $|E|$ times, so $O(|E| \log |V|)$.
- Total time:

$$O((|V| + |E|) \log |V|) = O(|E| \log |V|).$$

- Note: there is a way to do in $O(|V| \log |V| + |E|)$.

Shortest paths as a linear program

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

- Intuition: want x_{ij} to be 1 if edge (i, j) is in the path, otherwise 0.

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

- Intuition: want x_{ij} to be 1 if edge (i, j) is in the path, otherwise 0.
- Objective is just weight of all edges in the path.

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

- Intuition: want x_{ij} to be 1 if edge (i, j) is in the path, otherwise 0.
- Objective is just weight of all edges in the path.
- Constraints mean that $\#$ incoming edges = $\#$ outgoing edges at every vertex except s and t.

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

- Intuition: want x_{ij} to be 1 if edge (i, j) is in the path, otherwise 0.
- Objective is just weight of all edges in the path.
- Constraints mean that $\#$ incoming edges = $\#$ outgoing edges at every vertex except s and t.
- Could be same edge more than once or fractional combo of paths.

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

- Intuition: want x_{ij} to be 1 if edge (i, j) is in the path, otherwise 0.
- Objective is just weight of all edges in the path.
- Constraints mean that $\#$ incoming edges = $\#$ outgoing edges at every vertex except s and t.
- Could be same edge more than once or fractional combo of paths.
- But can show there must exist an optimum with all x_{ij} = 0 or 1.

Shortest paths as a linear program

- Suppose we want the shortest path between s and t.
- We can express this as a linear program!

$$\min_{x \in \mathbb{R}^{|E|}} \quad \sum_{i,j} w_{ij} x_{ij}$$

$$\text{such that} \quad \sum_i x_{ij} - \sum_i x_{ji} = \begin{cases} 0 & \text{for } j \neq s, t \\ 1 & \text{for } j = t \\ -1 & \text{for } j = s \end{cases}$$

$$x_i \geq 0 \text{ for all } i.$$

- Intuition: want x_{ij} to be 1 if edge (i, j) is in the path, otherwise 0.
- Objective is just weight of all edges in the path.
- Constraints mean that $\#$ incoming edges = $\#$ outgoing edges at every vertex except s and t.
- Could be same edge more than once or fractional combo of paths.
- But can show there must exist an optimum with all x_{ij} = 0 or 1.
- Note: **not** in general possible to reduce an integer LP to an LP.

Next time!

Graph algorithms III