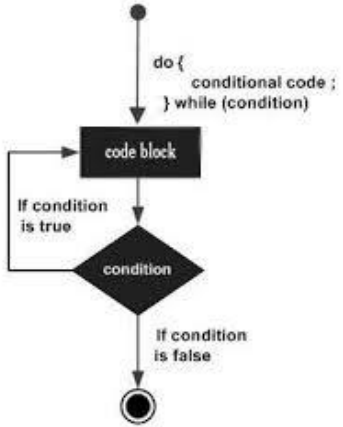
Lecture 12

Greedy Algorithms III



Announcements

- Homework 4 Reflections due Sunday night
- Group Meetings continue
- Quiz 1 due Saturday night
  - Up now
  - A few questions from each topic

Greedy Algorithms

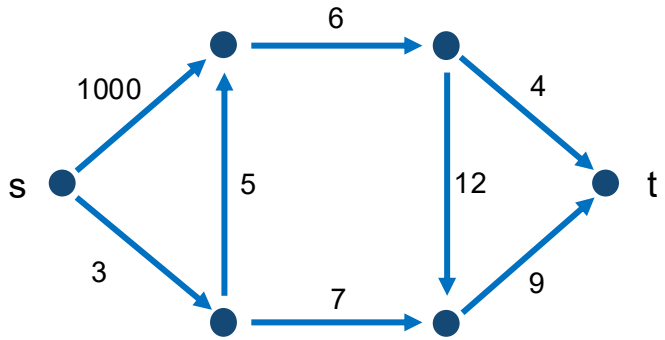
We are moving on to our study of algorithm design

- **Greedy**
- Divide-and-conquer
- Dynamic Programming
- Network Flow

Shortest path problem

New problem

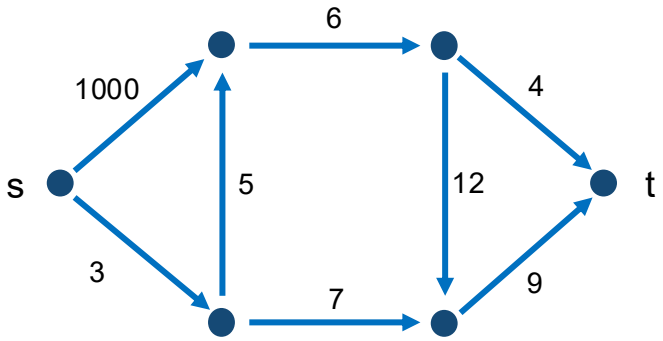
- Find shortest paths in a directed graph with edge weights (lengths)



Shortest path problem

New problem

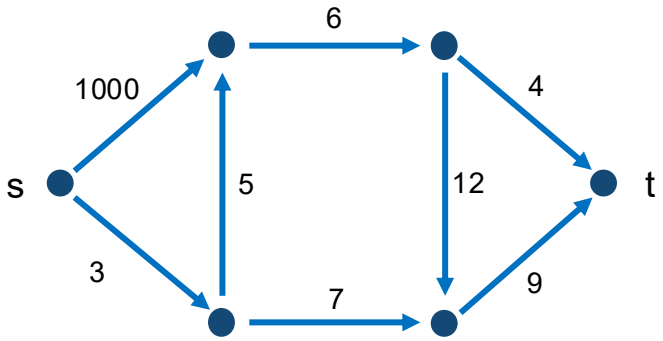
- Find shortest paths in a directed graph with edge weights (lengths)
- Directed graph $G = (V, E)$ with non-negative edge lengths $l(e) \geq 0$
- Define length of path P consisting of edges $e_1, e_2, \ldots, e_k$ as $l(P) = l(e_1) + \cdots + l(e_k)$
- Starting vertex s
- Let $d(v)$ be the length of the shortest s, v path



Shortest path problem

New problem

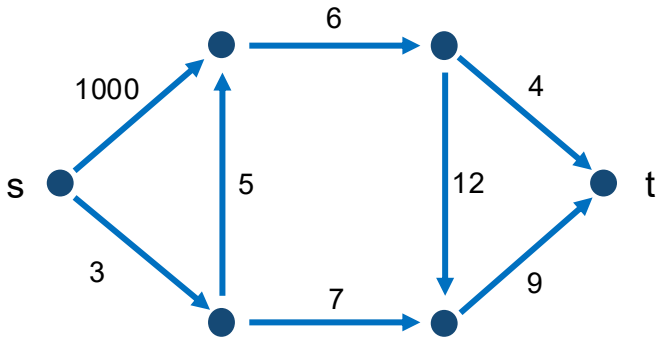
- Find shortest paths in a directed graph with edge weights (lengths)
- Directed graph $G = (V, E)$ with non-negative edge lengths $l(e) \geq 0$
- Define length of path P consisting of edges $e_1, e_2, \ldots, e_k$ as $l(P) = l(e_1) + \cdots + l(e_k)$
- Starting vertex s
- Let $d(v)$ be the length of the shortest s, v path
- **Problem**: can we efficiently find $d(v)$ for all vertices $v \in V$?



Dijkstra's Algorithm

Suppose all edges have integer lengths

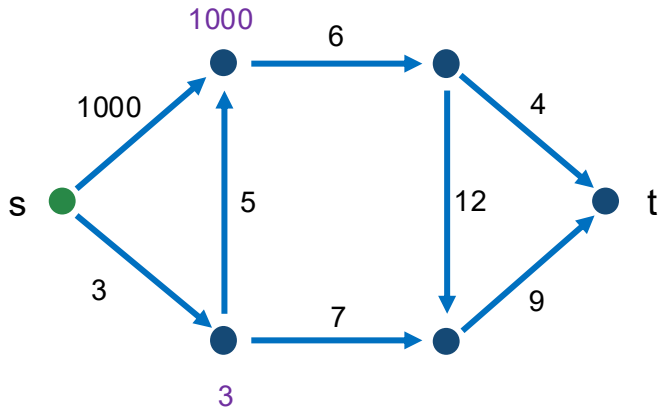
- Can we use BFS to solve this problem?
- Recall: nodes in layer L_i are at distance i from start



Dijkstra's Algorithm

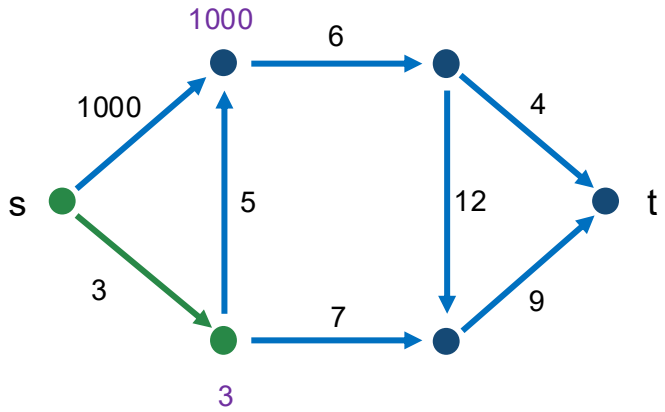
Idea: explore the "nearest" vertex from s

- Start at s and assign compute the distance to neighboring vertices



Dijkstra's Algorithm

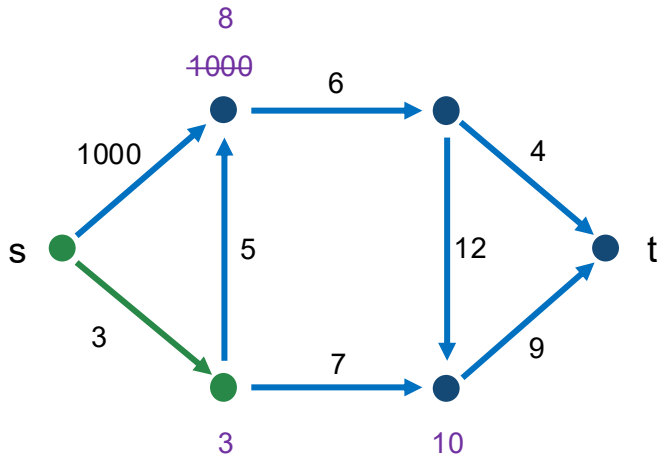
Idea: explore the "nearest" vertex from s



Dijkstra's Algorithm

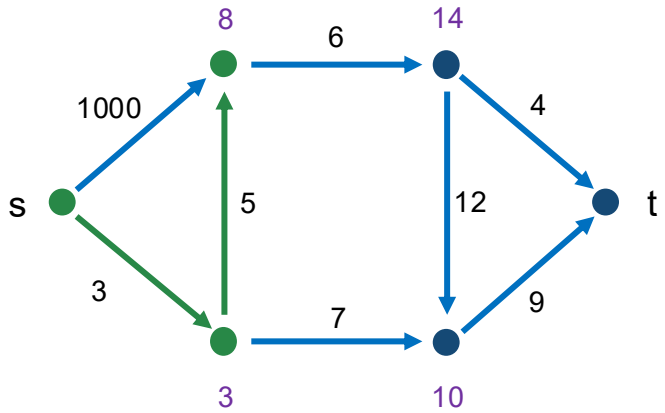
Idea: explore the "nearest" vertex from s

- Update shortest distance from s to neighbors of new vertex



Dijkstra's Algorithm

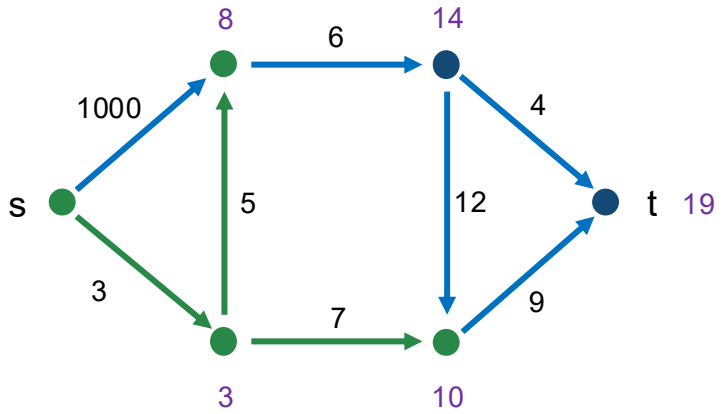
Idea: explore the "nearest" vertex from s



Dijkstra's Algorithm

Idea: explore the "nearest" vertex from s

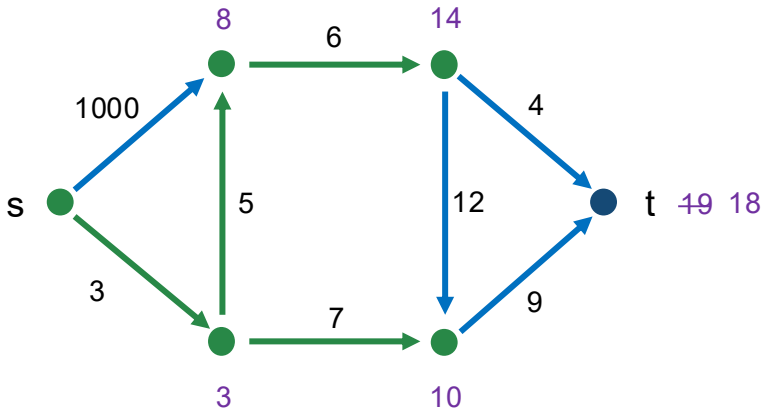
- We've found a path to t
- Are we done?



Dijkstra's Algorithm

Idea: explore the "nearest" vertex from s

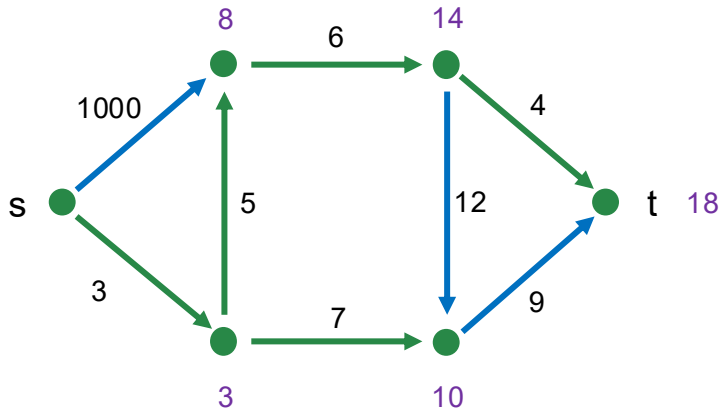
- Keep exploring!
- There may be a shorter path



Dijkstra's Algorithm

Idea: explore the "nearest" vertex from s

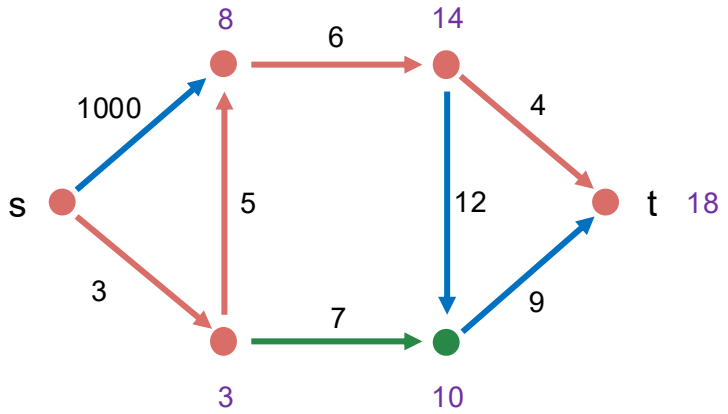
- All done!
- The shortest path from s to t has length 18



Dijkstra's Algorithm

Idea: explore the "nearest" vertex from s

- We can find the path by backtracking



Implementing Dijkstra's Algorithm

Notation

- $d'(v)$ is the earliest tentative arrival time so far for vertex v
- $d(v)$ is the shortest distance (actual arrival time)

How to keep tract of the tentative shortest distances?

- Find next arrival: node v with smallest $d'(v)$
- Set shortest distance: $d(v) = d'(v)$
- Update $d'(v)$ for neighbors of v if path through v is shorter
- Implement using a priority queue

Dijkstra's Algorithm

set $A = V$
set $d'(v) = \infty$ for all vertices
set $d'(s) = 0$
while A not empty **do**
    extract vertex $v \in A$ with smallest $d'(v)$ value
    set $d(v) = d'(v)$
    for all edges (v, w) where $w \in A$ **do**
        if $d(v) + l(v, w) < d'(w)$ **then**
            $d'(w) = d(v) + l(v, w)$

Tentative arrival time

Vertices left to explore

New shortest path

Dijkstra's Algorithm

set $A = V$
set $d'(v) = \infty$ for all vertices
set $d'(s) = 0$
while A not empty **do**
    extract vertex $v \in A$ with smallest $d'(v)$ value
    set $d(v) = d'(v)$
    for all edges (v, w) where $w \in A$ **do**
        if $d(v) + l(v, w) < d'(w)$ **then**
            $d'(w) = d(v) + l(v, w)$

Use priority queue for A

Extract min

Update key

Dijkstra's Algorithm

set $A = V$
set $d'(v) = \infty$ for all vertices
set $d'(s) = 0$
while A not empty **do**
    extract vertex $v \in A$ with smallest $d'(v)$ value
    set $d(v) = d'(v)$
    for all edges (v, w) where $w \in A$ **do**
        if $d(v) + l(v, w) < d'(w)$ **then**
            $d'(w) = d(v) + l(v, w)$

Use priority queue for A

Extract min

Update key

At most n extract min operations: $O(n \log n)$
At most m update key operations: $O(m \log n)$
Total: $O\big((m + n) \log n\big)$

Proof of Correctness

Idea: vertices are explored in increasing order of distance from s

At each step, we have:

- A is the set of vertices still to explore
- $S = V \setminus A$ are the explored vertices; $d(v)$ assigned

Proof of Correctness

Idea: vertices are explored in increasing order of distance from s

At each step, we have:

- A is the set of vertices still to explore
- $S = V \setminus A$ are the explored vertices; $d(v)$ assigned

Claim: Consider the set S at any point in the algorithm's execution. For each $v \in S$, $d(v)$ is the length of the shortest path from s to v.

Proof: By induction on $|S|$

Induction Proof

Base case

- Suppose $S = \emptyset$. Then $|S| = 1$
- Both (i) and (ii) are true trivially

Induction Proof

Base case

- Suppose $S = \emptyset$. Then $|S| = 1$
- The claim is trivially true

Inductive step

- Suppose the claim is true for $|S| = k \geq 1$

This means for every vertex u currently in S, the algorithm has successfully found a true shortest path from s to u

Induction Proof

Base case

- Suppose $S = \emptyset$. Then $|S| = 1$
- The claim is trivially true

This means for every vertex u currently in S, the algorithm has successfully found a true shortest path from s to u

Inductive step

- Suppose the claim is true for $|S| = k \geq 1$
- Now, let's add a v to S such that v has minimum $d'(v)$ for all $v \in A$
- Set $d(v) = d'(v)$

Goal: show that $d(v)$ is the minimum distance from s to v

Induction Proof

Goal: show that $d(v)$ is the minimum distance from s to v

- Suppose P is a path from s to v such that $l(P) < d(v)$
- In order to reach v, P must eventually leave S
- Let y be the first vertex on P that is not in S and let x be the vertex that comes just before y

Induction Proof

Goal: show that $d(v)$ is the minimum distance from s to v

- Suppose P is a path from s to v such that $l(P) < d(v)$
- In order to reach v, P must eventually leave S
- Let y be the first vertex on P that is not in S and let x be the vertex that comes just before y
- Let P_x be the subpath of P from s to x, and similarly for P_y
- By the inductive hypothesis, $d(x)$ is the minimum distance from s to x
- Then $d(x) \leq l(P_x)$, so $d'(y) \leq d(x) + l(x,y) \leq l(x,y) + l(P_x) = l\left(P_y\right) \leq l(P)$

Induction Proof

Goal: show that $d(v)$ is the minimum distance from s to v

- Suppose P is a path from s to v such that $l(P) < d(v)$
- In order to reach v, P must eventually leave S
- Let y be the first vertex on P that is not in S and let x be the vertex that comes just before y
- Let P_x be the subpath of P from s to x, and similarly for P_y
- By the inductive hypothesis, $d(x)$ is the minimum distance from s to x
- Then $d(x) \leq l(P_x)$, so $d'(y) \leq d(x) + l(x,y) \leq l(x,y) + l(P_x) = l\left(P_y\right) \leq l(P)$
- Because the algorithm v over y this iteration, we know that $d(v) = d'(v) \leq d'(y)$

Induction Proof

Goal: show that $d(v)$ is the minimum distance from s to v

- Suppose P is a path from s to v such that $l(P) < d(v)$
- In order to reach v, P must eventually leave S
- Let y be the first vertex on P that is not in S and let x be the vertex that comes just before y
- Let P_x be the subpath of P from s to x, and similarly for P_y
- By the inductive hypothesis, $d(x)$ is the minimum distance from s to x
- Then $d(x) \leq l(P_x)$, so $d'(y) \leq d(x) + l(x,y) \leq l(x,y) + l(P_x) = l\left(P_y\right) \leq l(P)$
- Because the algorithm v over y this iteration, we know that $d(v) = d'(v) \leq d'(y)$
- Combining these: $d(v) \leq l(P) < d(v)$ $\rightarrow\leftarrow$

Next Time

- Minimum Spanning Trees (MST)
- Start Divide and Conquer