Lecture 17: Dijkstra and Minimum Spanning Trees

COSC 311 *Algorithms*, Fall 2022

Overview

1. Dijkstra's Algorithm Correctness
2. Implementing Dijkstra
3. Minimum Spanning Trees

Last Time: Weighted SSSP

Single Source Shortest paths

Input.

- a weighted Graph $G = (V, E)$, edge weights w
- an initial vertex $u \in V$
- each vertex $v \in V$ has associated **adjacency list**
  - list of v's neighbors
  - includes weight of edge from v to each neighbor

$w((v, x))$ } length of hop

Output.

- A **map** $d : V \to \mathbf{R}$ such that $d[v] = d_w(u, v)$ is the graph distance from u to v
  - $d[v] = \infty$ indicates no path from u to v

length of shortest path from u to v

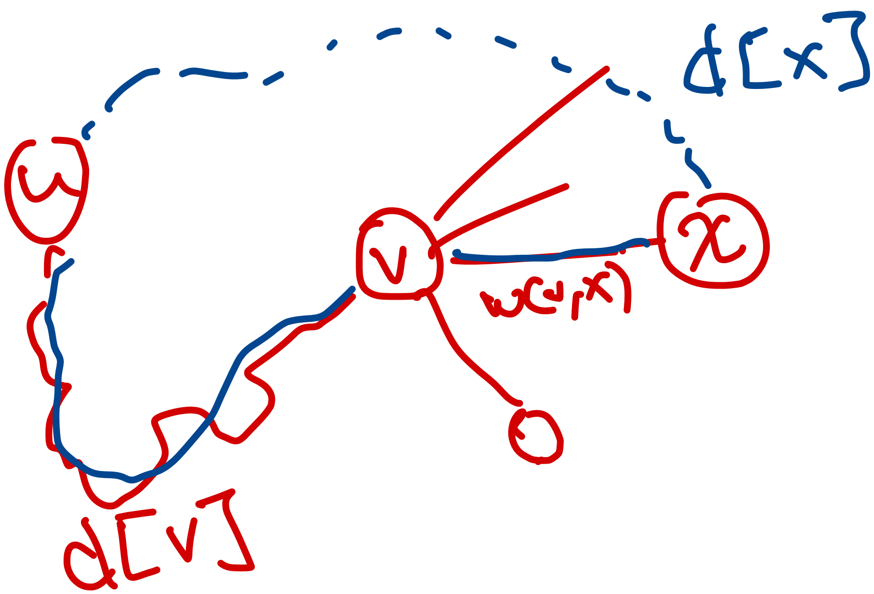
starting vtx

Dijkstra's Algorithm

1. Initialize $\boxed{d[u] = 0}$ and $d[v] = \infty$ for all $v \neq u$
2. Maintain set S of *finalized* nodes, initially empty
3. Process nodes. While $S \neq V$ do:
   - find node v in $\boxed{V - S}$ with minimal $d[v]$
   - add v to S
   - for each neighbor x of v
     - update $d[x] \leftarrow \min(d[x], \boxed{d[v] + w(v, x)})$

not all verts finalized

un-finalized



length of path from u to x that takes shortest path to v then hop (v,x)

Correctness

1. Initialize $d[u] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \neq u$
2. Maintain set S of *finalized* nodes, initially empty
3. Process nodes: while $S \neq V$
   - find node v in $V - S$ with minimal $d[v]$
   - add v to S
   - for each neighbor x of v
     - update $d[x] \leftarrow \min(d[x], d[v] + w(v, x))$

finalized

Claim. For every vertex $\boxed{v \in S}$, $d[v]$ stores the correct (weighted) distance $d_w(u, v)$.

Proof of Claim

Claim. For every vertex $v \in S$, $d[v]$ stores the correct (weighted) distance $d_w(u, v)$.

Proof. Use induction on size of S. Set k = size of S.

Base case $k = 1$. Only u is added to S. Set $d[u] \leftarrow 0$, which is correct answer.

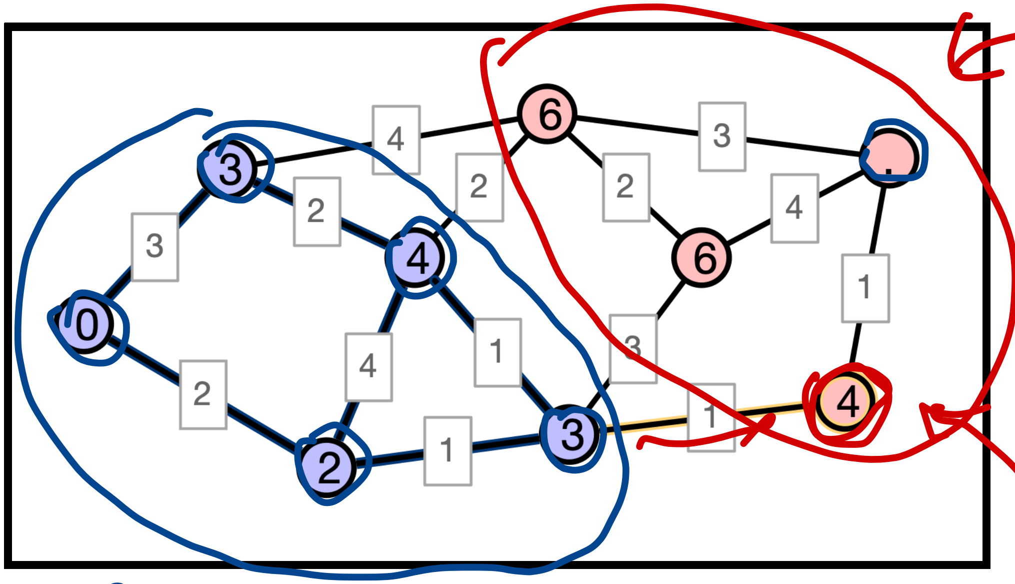
Inductive Step I

Inductive hypothesis. When S contains k elements, $d[v]$ is correct for all vertices $v \in S$.

Consider next iteration of outer loop:

- x has $d[x] = \min_{v \in S}(d[v] + w(v, x))$

not finalized



S

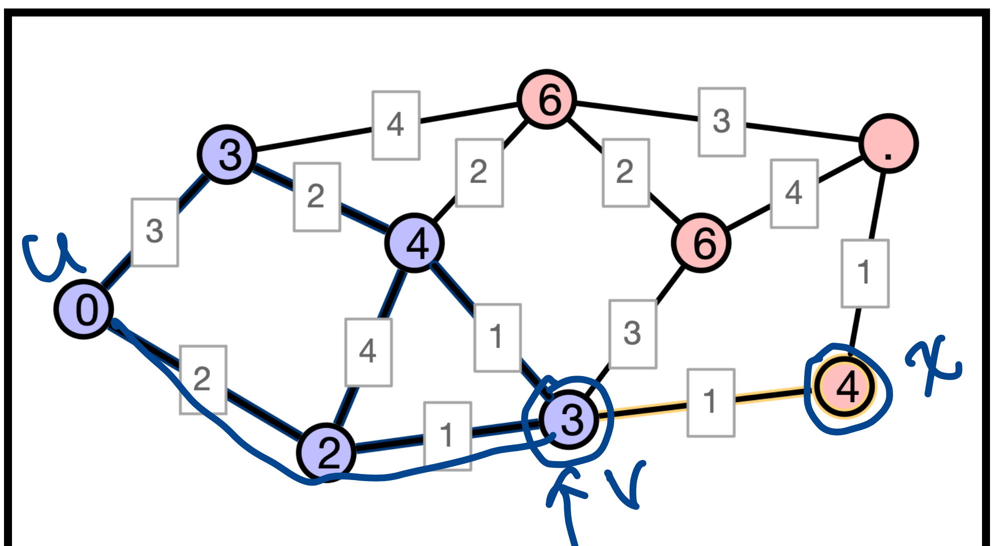
Claim: this dist is correct

Inductive Step II

Must show: $d[x] = d_w(u, x)$; argue by *contradiction*

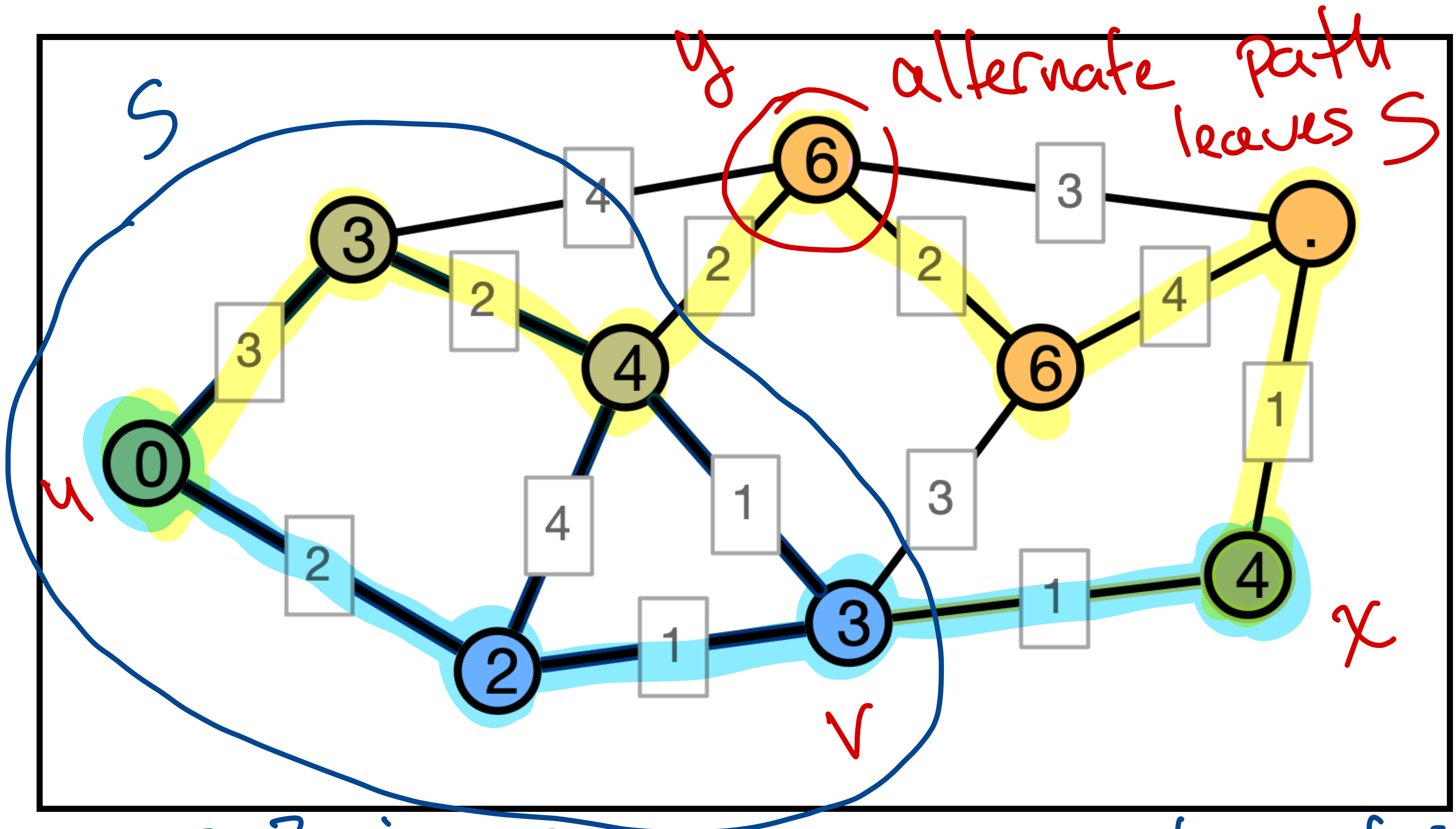
1. suppose $d[x] \neq d_w(u, x)$
2. observe: there is a path from u to x of length $d[x]$
3. $\implies$ $d_w(u, x) < d[x]$ length of shortest path from u to x
4. $\implies$ there is a path P from u to x of length $\ell < d[x]$

$d[v] + w(v,x) = d[x]$



know there is path from u to v of length $d[v]$ w/ $v \in S$
$\implies$ path from u to x of length

Shorter Path Illustration



d[x] is minimal among nodes not in S

y is not in S so... d[y] ≥ d[x]

⇒ other path has length ≥ d[x]

Inductive Step III

Must show: $d[x] = d_w(u, x)$; argue by *contradiction*

1. suppose $d[x] \neq d_w(u, x)$
2. observe: there is a path from u to x of length $d[x]$
3. $\implies d_w(u, x) < d[x]$
4. $\implies$ there is a path P from u to x of length $\boxed{\ell < d[x]}$
5. P must leave S at some point y min. $d[x]$
6. by definition of x, any path from u to y must be longer than $d[x]$
7. $\implies w(P) \geq d[x]$, which contradicts $\boxed{4}$

Conclusion. $d[x] = d_w(u, x)$, as claimed.

Dijkstra Running Time?

G has $\boxed{n}$ vertices, $\boxed{m}$ edges

Question. How many operations performed?

1. Initialize $d[u] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \neq u$
2. Maintain set S of *finalized* nodes, initially empty
3. Process nodes: while $S \neq V$
   - find node v in $V - S$ with minimal $d[v]$
   - add v to S
   - for each neighbor x of v
     - update $d[x] \leftarrow \min(\boxed{d[x]}, \boxed{d[v] + w(v, x)})$

$\boxed{n}$ times

deg(v) iterations

$\deg(v_1) + \deg(v_2) + \dots + \deg(v_n) = \boxed{2m}$

n iterations $v_1, v_2, v_3, \dots, v_n$

For Simplicity

Assume. Vertices are $1, 2, \ldots, n$

- d is an array with $d[v] =$ distance from u to v
- final is a Boolean array with $\text{final}[v] = \text{true}$ if v's distance is finalized
- keep track of number of finalized vertices
  - we're done when n vertices are finalized

n vert, m edges

Simple Implementation

For step

- find node v in $V - S$ with minimal $d[v]$

use linear search ← read all non-finalized elts, and return index of smallest

Question 1. What is running time of finding min?

$$O(n)$$

Question 2. What is overall running time of Dijkstra?

$$O(m + n^2) = O(n^2)$$

b/c $m < n^2$

all graphs on n vert have $\leq \frac{n(n-1)}{2}$ edges $< n^2$

Faster Implementation?

Since we need to access v with *minimum* $d[v]$, store non-finalized vertices in a **priority queue**

- store elements with associated *priorities*
- add element with given priority
- remove element with smallest priority

Faster Implementation?

Since we need to access v with *minimum* $d[v]$, store non-finalized vertices in a **priority queue**

- store elements with associated *priorities*
- add element with given priority
- remove element with smallest priority

Heap priority queue implementation

- supports these operations with running time $O(\log n)$

Faster Implementation?

Since we need to access v with *minimum* $d[v]$, store non-finalized vertices in a **priority queue**

- store elements with associated *priorities*
- add element with given priority
- remove element with smallest priority

Heap priority queue implementation

- supports these operations with running time $O(\log n)$

For Dijkstra:

- Store un-finalized vertices in priority queue
- priority of v is $d[v]$

One Sublety, Two Solutions

Issue. Dijkstra *decreases* priority of vertices

One Sublety, Two Solutions

Issue. Dijkstra *decreases* priority of vertices

Solution 1. Store duplicate vertices with each new distance

- will still find vertex v with smallest $d[v]$
- if finalized vertex is returned, ignore it
- requires priority queue of size m rather than n

One Sublety, Two Solutions

Issue. Dijkstra *decreases* priority of vertices

Solution 1. Store duplicate vertices with each new distance

- will still find vertex v with smallest $d[v]$
- if finalized vertex is returned, ignore it
- requires priority queue of size m rather than n

Solution 2. Use more sophisticated priority queue that supports "decrease priority" operation

- can be implemented in $O(\log n)$ time

Conclusion

Dijkstra performs

- n removals of vertices when they are finalized
- $2m$ distance updates

With efficient priority queues, these operations can each be performed in $O(\log n)$ time so...

Result. Dijkstra's algorithm can be implemented to run in time $O(\underline{m \log n})$.

better than n^2 when

$m << n^2$