Lecture 15 - Nov 5

Graphs

Visualizing Adjacency Lists Strategy
Shortest Paths in Weighted Graphs
Dijkstra's Algorithm: Intro, Example 1

Announcements/Reminders

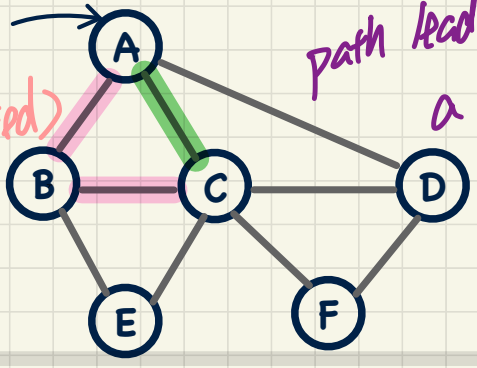
- Today's class: notes template posted
- Test 1 results released on Tuesday (Nov 4)
- Change of Dates:
  - + Assignment 2 to be released on Wed, Nov 12
  - + Assignment 2 to be due on Wed, Nov 19
  - + Test 2 to be take place on Mon, Nov 24

Back Edge (DFS) vs. Cross Edge (BFS): Cyclic?

Does a **back edge** always imply the existence of a **cycle**?

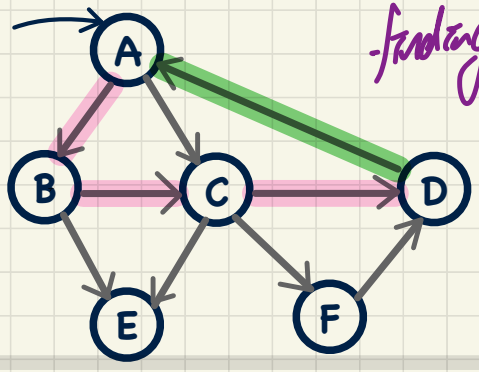
Does a **cross edge** always imply the existence of a **cycle**?

DFS



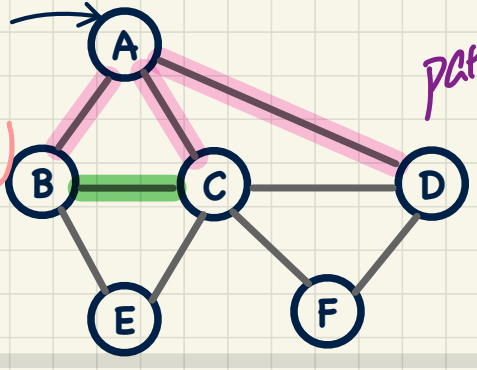
(undirected)

path leading to a back edge denotes a cycle



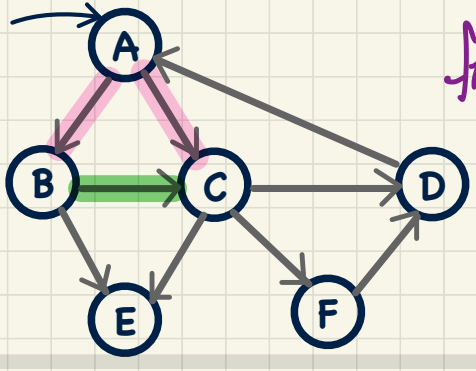
finding a back edge ⇒ ∃ a cycle.

BFS



(undirected)

path leading to a cross edge imply a cycle.



finding a cross edge ⇒ ∃ a cycle

Graphs in Java: Adjacency List Strategy (1)

```
class AdjacencyListGraph<V, E> implements Graph<V, E> {
  private DoublyLinkedList<AdjacencyListVertex<V>> vertices;
  private DoublyLinkedList<AdjacencyListEdge<E, V>> edges;
  private boolean isDirected;

  /* initialize an empty graph */
  AdjacencyListGraph(boolean isDirected) {
    vertices = new DoublyLinkedList<>();
    edges = new DoublyLinkedList<>();
    this.isDirected = isDirected;
  }
```

```
public class Vertex<V> {
  private V element;
  public Vertex(V element) { this.element = element; }
  /* setter and getter for element */
}
```

```
public class Edge<E, V> {
  private E element;
  private Vertex<V> origin;
  private Vertex<V> dest;
  public Edge(E element) { this.element = element; }
  /* setters and getters for element, origin, and destination */
}
```

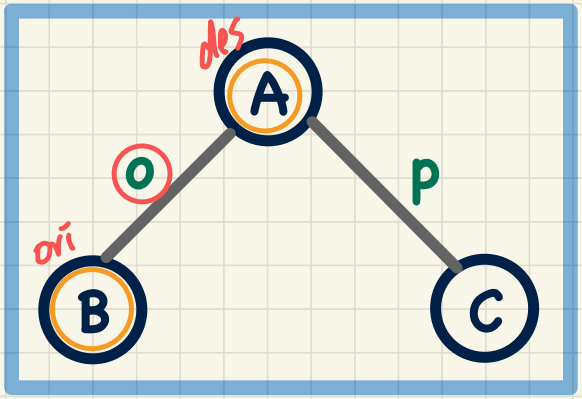
```
public class EdgeListVertex<V> extends Vertex<V> {
  public DLNode<Vertex<V>> vertextListPosition;
  /* setter and getter for vertexListPosition */
}
```

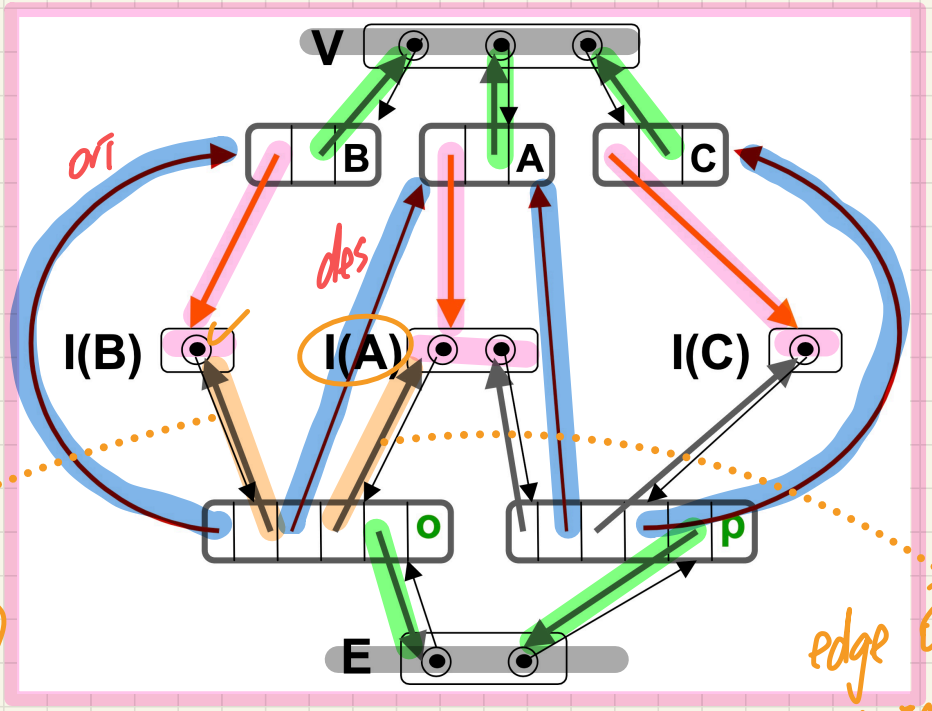
```
public class EdgeListEdge<E, V> extends Edge<E, V> {
  public DLNode<Edge<E, V>> edgeListPosition;
  /* setter and getter for edgeListPosition */
}
```

```
class AdjacencyListVertex<V> extends EdgeListVertex<V> {
  private DoublyLinkedList<AdjacencyListEdge<E, V>> incidentEdges;
  /* getter for incidentEdges */
}
```

```
class AdjacencyListEdge<V> extends EdgeListEdge<V> {
  DLNode<Edge<E, V>> originIncidentListPos;
  DLNode<Edge<E, V>> destIncidentListPos;
}
```

Graphs in Java: Adjacency List Strategy (2)

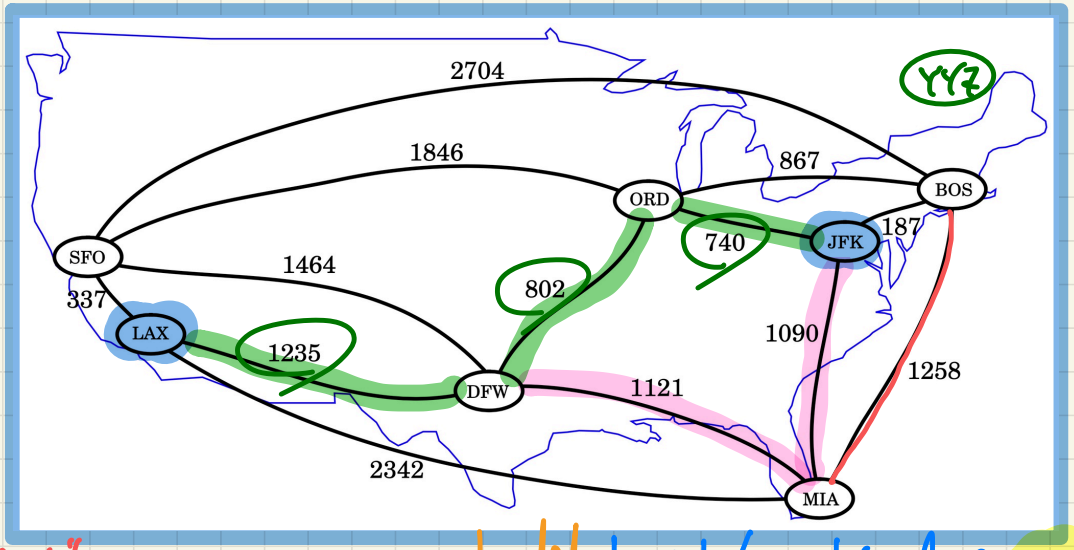




edge o's position in its origin's (B) incident edges

edge o's position in its des.'s (A) i.e. list.

Shortest Paths in Weighted Graphs



e.g. $d(YYZ, BOS) = \infty$
$d(JFK, LAX) = ??$

distance / shortest path between two vertices

out of all paths connecting u and v, $d(u, v)$ denotes the minimum length/weight among them.

$d(u, v) = \infty$ if not connected.

"cost"

weights on edges

$w(u, v)$

overloaded length/weight of a path

$w((v_0, v_1, \dots, v_k))$

weight ↳ path with k+1 vertices

$= w(v_0, v_1) + w(v_1, v_2) + \dots + w(v_{k-1}, v_k)$

↳ edge

$= \sum_{i=0}^{k-1} w(v_i, v_{i+1})$

e.g. $w(BOS, MIA) = 1258$

Assumption: $w \geq 0$ (non-negative)

Dijkstra's Shortest Path Algorithm

Starting from a **source vertex** s, perform a ***BFS***-like procedure:

1. Initially:
   - 1.1 Set $D(s) = 0$, and every other vertex $t \neq s$, $D(t) = \infty$. [**distance**]
   - 1.2 Set $a(v) = nil$ for every vertex v. [**ancestor** in **shortest path**]
   - 1.3 Insert all vertices into a **priority queue** Q [**keyed** by D]
2. While Q is not empty, repeat the following:
   - 2.1 Find vertex u in Q s.t. $D(u)$ is the **minimum**.
   - 2.2 For every vertex v **adjacent to** u **if**:
     $v \in Q \wedge D(u) + w(u,v) < D(v)$, then:
     - Set $D(v) = D(u) + w(u,v)$
     - Set $a(v) = u$
   - 2.3 Remove vertex u from Q.

Upon completion, for every vertex t ($t \neq s$):

- $D(t) = d(s,t)$ (i.e., weight of **shortest path** from s to t).
- **Reversing** t's ***ancestor path*** → **shortest path**: $\langle s, \ldots, a(t), t \rangle$

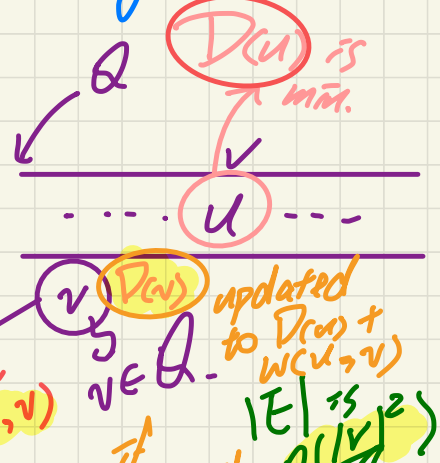
shortest path of some vertex so far (eventually)

intermediate result in loop

$D(v)$ = $d(v)$ the shortest length to get to.

heap (min-heap)

"visited"

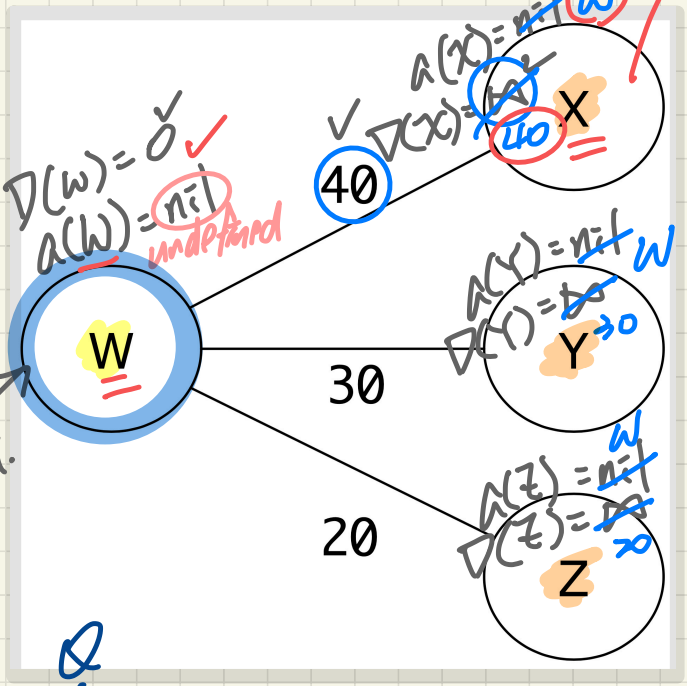


iterations ≤ degree(u)

often not necessary to go over all incident edges (time consuming if the graph is complete)

Dijkstra's Shortest Path Algorithm: Example 1

shortest path from W to X has weight: 40.



Iteration	vertex with min D	changes?
Init.		
1	W Remove: ⟨W, X⟩	D(W) + w(W,X) < D(X) 0 + 40 < ∞ D(X) = 40 a(X) = W D(Y) = 30 a(Y) = W D(Z) = 20 a(Z) = W
2	Z D(Z) = 20	N.C. ∵ W ∉ Q
3	Y D(Y) = 30	N.C. ∵ W ∉ Q
4	X D(X) = 20	N.C ∵ W ∉ Q.

Q

①	②
~~W~~	X Y ~~Z~~
①	②