39: Shortest Paths
Lecture 39: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra’s Algorithm

Some slides adopted from Algorithms 4th Edition or COS226
Edge-weighted digraph

- **Edge-weighted digraph**: a digraph where we associate weights or costs with each edge.
Shortest Paths

- **Shortest path from vertex s to vertex t**: a directed path from s to t with the property that no other such path has a lower weight (total weight sum of edges it consists of).
Shortest Path variants

- **Single source**: from one vertex $s$ to every other vertex.
- **Single sink**: from every vertex to one vertex $t$.
- **Source-sink**: from one vertex $s$ to another vertex $t$.
- **All pairs**: from every vertex to every other vertex.

- **What version is there in your navigation app?**
Shortest Paths Assumptions

- Not all vertices need to be reachable.
  - We will assume so in this lecture.
- Weights are non-negative.
  - There are algorithms that can handle negative weights.
- Shortest paths are not necessarily unique but they are simple.
Lecture 39: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra’s Algorithm
Weighted directed edge API

- **public class** DirectedEdge
  - DirectedEdge(int v, int w, double weight)
    - Constructs a weighted edge from v to w (v->w) with the provided weight.
  - int from()
    - Returns vertex source of this edge.
  - int to()
    - Returns vertex destination of this edge.
  - double weight()
    - Returns weight of this edge.
  - String toString()
    - Returns the string representation of this edge.
Weighted directed edge in Java

```java
public class DirectedEdge {
    private final int v;
    private final int w;
    private final double weight;

    public DirectedEdge(int v, int w, double weight) {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int from() {
        return v;
    }

    public int to() {
        return w;
    }

    public double weight() {
        return weight;
    }
}
```
Edge-weighted digraph API

- **public class** `EdgeWeightedDigraph`
  - `EdgeWeightedDigraph(int v)`
    - Constructs an edge-weighted digraph with `V` vertices.
  - `void addEdge(DirectedEdge e)`
    - Add weighted directed edge `e`.
  - `Iterable<DirectedEdge> adj(int v)`
    - Returns edges adjacent from `v`.
  - `int V()`
    - Returns number of vertices.
  - `int E()`
    - Returns number of edges.
  - `Iterable<DirectedEdge> edges()`
    - Returns all edges.
Edge-weighted digraph adjacency list representation
Edge-weighted digraph in Java

```java
public class EdgeWeightedDigraph {
    private final int V;  // number of vertices in this digraph
    private int E;       // number of edges in this digraph
    private Bag<DirectedEdge>[] adj; // adj[v] = adjacency list for vertex v

    public EdgeWeightedDigraph(int V) {
        this.V = V;
        this.E = 0;
        adj = (Bag<DirectedEdge>[][]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<DirectedEdge>();
    }

    public void addEdge(DirectedEdge e) {
        int v = e.from();
        int w = e.to();
        adj[v].add(e);
        E++;
    }

    public Iterable<DirectedEdge> adj(int v) {
        return adj[v];
    }
}
```
**Single-source shortest path API**

- **Goal**: find shortest path from $s$ to every other vertex in the digraph.

- **public class SP**
  - SP(EdgeWeightedDigraph G, int s)
    - Shortest paths from $s$ in digraph $G$.
  - double distTo(int v)
    - Length of shortest path from $s$ to $v$.
  - Iterable<DirectedEdge> pathTo(int v)
    - Returns edges along the shortest path from $s$ to $v$.
  - boolean hasPathTo(int v)
    - Returns whether there is a path from $s$ to $v$. 
Lecture 39: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra’s Algorithm
Data structures for single-source shortest paths

- **Goal**: find shortest path from s to every other vertex in the digraph.

- **Shortest-paths tree (SPT)**: a subgraph containing s and all the vertices reachable from s that forms a directed tree rooted at s such that every tree path in the SPT is a shortest path in the digraph.

- Representation of shortest paths with two vertex-indexed arrays.
  - **Edges on the shortest-paths tree**: edgeTo[v] is the last edge on a shortest path from s to v.
  - **Distance to the source**: distTo[v] is the length of the shortest path from s to v.
public Iterable<DirectedEdge> pathTo(int v) {
    Stack<DirectedEdge> path = new Stack<DirectedEdge>();
    for (DirectedEdge e = edgeTo[v]; e != null; e = edgeTo[e.from()]) {
        path.push(e);
    }
    return path;
}
Edge relaxation

- Relax edge $e = v \rightarrow w$
  - $\text{distTo}[v]$ is the length of the shortest known path from $S$ to $v$.
  - $\text{distTo}[w]$ is the length of the shortest known path from $S$ to $w$.
  - $\text{edgeTo}[w]$ is the last edge on shortest known path from $S$ to $w$.
  - If $e = v \rightarrow w$ yields shorter path to $w$, update $\text{distTo}[w]$ and $\text{edgeTo}[w]$. 
Edge relaxation
```java
private void relax(DirectedEdge e) {
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight()) {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
    }
}
```
Framework for shortest-paths algorithm

- Generic algorithm to compute a SPT from $s$
  - $distTo[v]=\infty$ for each vertex $v$.
  - $edgeTo[v]=null$ for each vertex $v$.
  - $distTo[s]=0$.
- Repeat until done:
  - Relax any edge.
- $distTo[v]$ is the length of a simple path from $s$ to $v$.
- $distTo[v]$ does not increase.
Framework for shortest-paths algorithm

- **Generic algorithm to compute a SPT from s**
  - distTo[v] = \infty for each vertex v.
  - edgeTo[v] = null for each vertex v.
  - distTo[s] = 0.
  - Repeat until done:
    - Relax any edge.
  - distTo[v] is the length of a simple path from s to v.
  - distTo[v] does not increase.
Lecture 39: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra’s Algorithm
DIJKSTRA'S ALGORITHM DEMO
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

![Dijkstra's algorithm demo graph]

**an edge-weighted digraph**
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from \( s \) (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges adjacent from that vertex.

relax all edges adjacent from 0
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[\cdot]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>8.0</td>
<td>0→7</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

relax all edges adjacent from 0
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$
  (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

choose vertex 1

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from \( s \) (non-tree vertex with the lowest \( \text{distTo[]} \) value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8.0</td>
<td>0→7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

relax all edges adjacent from 1
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo[]}$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>17.0</td>
<td>1→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```

relax all edges adjacent from 1
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest `$\text{distTo}[]$` value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

\[
\begin{array}{c|c|c}
\text{v} & \text{distTo[]} & \text{edgeTo[]} \\
0 & 0.0 & - \\
1 & 5.0 & 0→1 \\
2 & 17.0 & 1→2 \\
3 & 20.0 & 1→3 \\
4 & 9.0 & 0→4 \\
5 & & \\
6 & & \\
7 & 8.0 & 0→7 \\
\end{array}
\]

choose vertex 7
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

relax all edges adjacent from 7
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[\cdot]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

relax all edges adjacent from 7
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[\cdot]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
select vertex 4

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>14.0</td>
<td>7→5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>14.0</td>
<td>7→5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges adjacent from 4
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>29.0</td>
<td>4→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges adjacent from 4
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>29.0</td>
<td>4→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```

select vertex 5
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest \( \text{distTo}[] \) value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
<table>
<thead>
<tr>
<th></th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>26.0</td>
<td>5→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```

relax all edges adjacent from 5
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>26.0</td>
<td>5→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

select vertex 2
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from \( s \) (non-tree vertex with the lowest \( \text{distTo}[] \) value).
- Add vertex to tree and relax all edges adjacent from that vertex.

relax all edges adjacent from 2
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges adjacent from that vertex.

relax all edges adjacent from 2

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from \( s \) (non-tree vertex with the lowest \( \text{distTo[]} \) value).
- Add vertex to tree and relax all edges adjacent from that vertex.

\[
\begin{array}{c|c|c}
\text{v} & \text{distTo[]} & \text{edgeTo[]} \\
0 & 0.0 & - \\
1 & 5.0 & 0\rightarrow1 \\
2 & 14.0 & 5\rightarrow2 \\
3 & 17.0 & 2\rightarrow3 \\
4 & 9.0 & 0\rightarrow4 \\
5 & 13.0 & 4\rightarrow5 \\
6 & 25.0 & 2\rightarrow6 \\
7 & 8.0 & 0\rightarrow7 \\
\end{array}
\]
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

!![Graph representation of Dijkstra's algorithm]

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

select vertex 3
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

```
\begin{array}{ccc}
\text{v} & \text{distTo[]} & \text{edgeTo[]} \\
0 & 0.0 & - \\
1 & 5.0 & 0\rightarrow1 \\
2 & 14.0 & 5\rightarrow2 \\
3 & 17.0 & 2\rightarrow3 \\
4 & 9.0 & 0\rightarrow4 \\
5 & 13.0 & 4\rightarrow5 \\
6 & 25.0 & 2\rightarrow6 \\
7 & 8.0 & 0\rightarrow7 \\
\end{array}
```

relax all edges adjacent from 3
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from \( s \) (non-tree vertex with the lowest \( \text{distTo[]} \) value).
- Add vertex to tree and relax all edges adjacent from that vertex.

![Graph diagram]

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges adjacent from 3
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[\cdot]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

**select vertex 6**
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[\cdot]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[\cdot]$</th>
<th>$\text{edgeTo}[\cdot]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges adjacent from 6
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges adjacent from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges adjacent from that vertex.

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Indexed min-priority queue (Section 2.4 in textbook)

- Associate an index between 0 and n-1 with each key in a priority queue.
  - Insert a key associated with a given index.
  - Delete a minimum key and return associated index.
  - Decrease the key associated with a given index.

```java
public class IndexMinPQ<Key extends Comparable<Key>>

IndexMinPQ(int n)
  - Create indexed PQ with indices 0,1,...n-1

void insert(int i, Key key)
  - Associate key with index i.

int delMin()
  - Remove a minimal key and return its associated index.

void decreaseKey(int i, Key key)
  - Decrease the key with index i to the specified value.
```
public class DijkstraSP {
    private double[] distTo;  // distTo[v] = distance of shortest s->v path
    private DirectedEdge[] edgeTo;  // edgeTo[v] = last edge on shortest s->v path
    private IndexMinPQ<Double> pq;  // priority queue of vertices

    public DijkstraSP(EdgeWeightedDigraph G, int s) {
        distTo = new double[G.V()];
        edgeTo = new DirectedEdge[G.V()];

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        // relax vertices in order of distance from s
        pq = new IndexMinPQ<Double>(G.V());
        pq.insert(s, distTo[s]);
        while (!pq.isEmpty()) {
            int v = pq.delMin();
            for (DirectedEdge e : G.adj(v))
                relax(e);
        }
    }

    // relax edge e and update pq if changed
    private void relax(DirectedEdge e) {
        int v = e.from(), w = e.to();
        if (distTo[w] > distTo[v] + e.weight()) {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
            else pq.insert(w, distTo[w]);
        }
    }
}
Running time depends on PQ implementation

- Many variations. Assuming binary heap, running time is proportional to $|E| \log |V|$ and $|V|$ extra space.
  - Cost of insert, delete-min, decrease-key are all $\log V$.
- More complicated version with a Fibonacci heap (CS140...) takes $O(E + V \log V)$ time but in practice it’s not worth implementing.
Practice Time

- Run Dijkstra’s algorithm on the following graph with 0 being the starting vertex.
**Answer**

![Graph with edge weights and table]

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>3-&gt;1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0-&gt;2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2-&gt;3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3-&gt;4</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>6-&gt;5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>4-&gt;6</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>5-&gt;7</td>
</tr>
</tbody>
</table>
Lecture 39: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra’s Algorithm
Readings:

- Textbook: Chapter 4.4 (Pages 638-657)
- Website: https://algs4.cs.princeton.edu/44sp/

Practice Problems: