Graph Theory and Its Applications

Dr. G.H.J. Lanel

Lecture 9

Dr. G.H.J. Lanel (USJP)

Graph Theory and Its Applications

Lecture 9 1 / 22

Outline

Outline

イロト イロト イヨト イヨト

Outline

1 The Shortest Path Problem

2 Shortest Path Algorithms

Shortest Path Problems

- Single-source shortest path problem
- Point to point shortest path problem
- All pairs shortest path problem
- Negative weights shortest path problem

Unweighted Shortest Paths

Weighted Shortest PathsDijkstra's algorithm

- A 🖻 🕨

• Find the shortest path from point *A* to point *B*.

- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation,
 - Elight itineraries
 - Circuit wiring,
 - Network routing.

• Find the shortest path from point *A* to point *B*.

- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation.
 - Elight itineraries
 - Circuit wiring,
 - Network routing.

イロト イポト イヨト イヨト

- Find the shortest path from point *A* to point *B*.
- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation,
 - Flight itineraries,
 - Circuit wiring,
 - Network routing.

- Find the shortest path from point *A* to point *B*.
- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation,
 - Flight itineraries,
 - Circuit wiring,
 - Network routing.

- Find the shortest path from point *A* to point *B*.
- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation,
 - Flight itineraries,
 - Circuit wiring,
 - Network routing.

- Find the shortest path from point *A* to point *B*.
- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation,
 - Flight itineraries,
 - Circuit wiring,
 - Network routing.

(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

- Find the shortest path from point *A* to point *B*.
- Shortest in time, distance, cost, etc.
- Numerous applications:
 - Map navigation,
 - Flight itineraries,
 - Circuit wiring,
 - Network routing.

(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

Outline

The Shortest Path Problem

2 Shortest Path Algorithms

Shortest Path Problems

- Single-source shortest path problem
- Point to point shortest path problem
- All pairs shortest path problem
- Negative weights shortest path problem

Unweighted Shortest Paths

Weighted Shortest PathsDijkstra's algorithm

- A 🖻 🕨

• Weighted graphs:

- Input is a weighted graph where each edge (v_i, v_j) has cost c_{i,j} to traverse the edge.
- Cost of a path $v_1 v_2 \dots v_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$.
- Goal: to find a smallest cost path.
- Unweighted graphs:
 - Input is an unweighted graph...
 - Goal: to find a path with shortest length.

Weighted graphs:

• Input is a weighted graph where each edge (*v_i*, *v_j*) has cost *c_{i,j}* to traverse the edge.

• Cost of a path $v_1 v_2 \dots v_N$ is $\sum c_{i,i+1}$.

• Goal: to find a smallest cost path.

Unweighted graphs:

- Input is an unweighted graph...
- Goal: to find a path with shortest length.

Weighted graphs:

- Input is a weighted graph where each edge (v_i, v_j) has cost c_{i,j} to traverse the edge.
- Cost of a path $v_1 v_2 \dots v_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$.

Goal: to find a smallest cost path.

• Unweighted graphs:

- Input is an unweighted graph...
- Goal: to find a path with shortest length.

• Weighted graphs:

• Input is a weighted graph where each edge (*v_i*, *v_j*) has cost *c_{i,j}* to traverse the edge.

• Cost of a path
$$v_1 v_2 \dots v_N$$
 is $\sum_{i=1}^{N-1} c_{i,i+1}$.

- Goal: to find a smallest cost path.
- Unweighted graphs:
 - Input is an unweighted graph.
 - Goal: to find a path with shortest length.

- Weighted graphs:
 - Input is a weighted graph where each edge (v_i, v_j) has cost c_{i,j} to traverse the edge.
 - Cost of a path $v_1 v_2 \dots v_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$.
 - Goal: to find a smallest cost path.
- Unweighted graphs:
 - Input is an unweighted graph.
 - Goal: to find a path with shortest length.

- Weighted graphs:
 - Input is a weighted graph where each edge (v_i, v_j) has cost c_{i,j} to traverse the edge.
 - Cost of a path $v_1 v_2 \dots v_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$.
 - Goal: to find a smallest cost path.
- Unweighted graphs:
 - Input is an unweighted graph.
 - Goal: to find a path with shortest length.

- Weighted graphs:
 - Input is a weighted graph where each edge (v_i, v_j) has cost c_{i,j} to traverse the edge.
 - Cost of a path $v_1 v_2 ... v_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$.
 - Goal: to find a smallest cost path.
- Unweighted graphs:
 - Input is an unweighted graph.
 - Goal: to find a path with shortest length.

Outline



2 Shortest Path Algorithms

Shortest Path Problems

- Single-source shortest path problem
- Point to point shortest path problem
- All pairs shortest path problem
- Negative weights shortest path problem

Unweighted Shortest Paths

Weighted Shortest Paths
Dijkstra's algorithm

Single-source shortest path problem

Given a weighted graph G = (V, E), and a source vertex *s*, find the minimum weighted path from *s* to every other vertex in *G*.

Single-source shortest path problem

Given a weighted graph G = (V, E), and a source vertex *s*, find the minimum weighted path from *s* to every other vertex in *G*.



(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

• Given G = (V, E) and two vertices A and B, find a shortest path from A (source) to B (destination).

• Solution:

Run the code for Single Source Shortest Path using source as A.

Stop algorithm when B is reached...

(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

- Given G = (V, E) and two vertices A and B, find a shortest path from A (source) to B (destination).
- Solution:
 - Run the code for Single Source Shortest Path using source as A.
 - Stop algorithm when B is reached.

- Given G = (V, E) and two vertices A and B, find a shortest path from A (source) to B (destination).
- Solution:
 - Run the code for Single Source Shortest Path using source as A.
 - Stop algorithm when B is reached.

- Given G = (V, E) and two vertices A and B, find a shortest path from A (source) to B (destination).
- Solution:
 - Run the code for Single Source Shortest Path using source as A.
 - Stop algorithm when B is reached.

All pairs shortest path problem

• Given *G* = (*V*, *E*), find a shortest path between all pairs of vertices.

Solutions: Solve Single Source Shortest Path for each vertex as source

All pairs shortest path problem

- Given *G* = (*V*, *E*), find a shortest path between all pairs of vertices.
- Solutions: Solve Single Source Shortest Path for each vertex as source

• Graphs can have negative weights.



● E.g.,

- Shortest positive-weight path is a net gain.
- Path may include individual losses.
- Problem: Negative weight cycles

Allow arbitrarily-low path costs

Solution: Detect presence of negative-weight cycles

• Graphs can have negative weights.



• E.g.,

- Shortest positive-weight path is a net gain.
- Path may include individual losses.
- Problem: Negative weight cycles

Allow arbitrarily-low path costs

Solution: Detect presence of negative-weight cycles

• Graphs can have negative weights.



- E.g.,
 - Shortest positive-weight path is a net gain.
 - Path may include individual losses.
- Problem: Negative weight cycles

Allow arbitrarily-low path costs

Solution: Detect presence of negative-weight cycles

• Graphs can have negative weights.



- E.g.,
 - Shortest positive-weight path is a net gain.
 - Path may include individual losses.
- Problem: Negative weight cycles

Allow arbitrarily-low path costs

Solution: Detect presence of negative-weight cycles

• Graphs can have negative weights.



- E.g.,
 - Shortest positive-weight path is a net gain.
 - Path may include individual losses.
- Problem: Negative weight cycles

Allow arbitrarily-low path costs

Solution: Detect presence of negative-weight cycles

4 Th

< (□) < 三 > (□)

• Graphs can have negative weights.



- E.g.,
 - Shortest positive-weight path is a net gain.
 - Path may include individual losses.
- Problem: Negative weight cycles

Allow arbitrarily-low path costs

• Solution: Detect presence of negative-weight cycles

< ∃ ►

-

Outline

- The Shortest Path Problem
- 2 Shortest Path Algorithms
 - Shortest Path Problems
 - Single-source shortest path problem
 - Point to point shortest path problem
 - All pairs shortest path problem
 - Negative weights shortest path problem

Unweighted Shortest Paths

Weighted Shortest Paths
 Diikstra's algorithm

- ∢ ∃ ▶

• No weights on edges.

- Find shortest length paths.
- Same as weighted shortest path with all weights equal.
- Use BFS algorithm.

- No weights on edges.
- Find shortest length paths.
- Same as weighted shortest path with all weights equal.
- Use BFS algorithm.

- No weights on edges.
- Find shortest length paths.
- Same as weighted shortest path with all weights equal.
- Use BFS algorithm.

- No weights on edges.
- Find shortest length paths.
- Same as weighted shortest path with all weights equal.
- Use BFS algorithm.

- For each vertex, keep track of
 - Whether we have visited it (known).
 - Its distance from the start vertex (d_v) .
 - Its predecessor vertex along the shortest path from the start vertex (p_v).

For each vertex, keep track of

- Whether we have visited it (known).
- Its distance from the start vertex (d_v) .
- Its predecessor vertex along the shortest path from the start vertex (p_v).

For each vertex, keep track of

- Whether we have visited it (known).
- Its distance from the start vertex (d_v) .
- Its predecessor vertex along the shortest path from the start vertex (p_v).



ν	known	d_v	p_{ν}
v ₁	F	∞	0
v ₂	F	∞	0
v ₃	F	0	0
v_4	F	∞	0
v_5	F	∞	0
v ₆	F	∞	0
v_7	F	∞	0

(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

	Initial State			v3 Dequeued		v_1 Dequeued			v ₆ Dequeued			
ν	known	d_v	p_{ν}	known	d_v	p_{ν}	known	d_v	p_{ν}	known	d_v	p_{ν}
v ₁	F	∞	0	F	1	v ₃	Т	1	v ₃	Т	1	v ₃
v ₂	F	∞	0	F	∞	0	F	2	v_1	F	2	v_1
v ₃	F	0	0	Т	0	0	Т	0	0	Т	0	0
ν ₄	F	∞	0	F	∞	0	F	2	v_1	F	2	v_1
v ₅	F	∞	0	F	∞	0	F	∞	0	F	∞	0
v ₆	F	∞	0	F	1	v ₃	F	1	v ₃	Т	1	v ₃
v7	F	∞	0	F	∞	0	F	∞	0	F	∞	0
Q:	v ₃		v_1, v_6		v_6, v_2, v_4			v_2, v_4				
	v ₂ Dequeued			v ₄ Dequeued		v5 Dequeued			v7 Dequeued			
ν	known	d	n	1	1					-		
V1		αų	Pν	rnown	a_v	Рv	known	d_v	pν	known	d_v	p_{v}
· 1	Т	1	Pv V3	T	<i>a</i> _v 1	<i>p</i> _ν ν ₃	known T	<i>d</i> _v 1	Pv v3	known T	$\frac{d_v}{1}$	$\frac{p_v}{v_3}$
v ₂	T T	1 2	$\frac{v_3}{v_1}$	T T	1 2	$\frac{p_v}{v_3}$	known T T	<i>d_v</i> 1 2	p_{v} v_{3} v_{1}	known T T	<i>d_v</i> 1 2	$\frac{p_v}{v_3}$ v_1
ν ₂ ν ₃	T T T	1 2 0	v_3 v_1 0	T T T T	1 2 0	p_{v} v_{3} v_{1} 0	known T T T	<i>d_v</i> 1 2 0	p_v v_3 v_1 0	known T T T	d _v 1 2 0	
ν ₂ ν ₃ ν ₄	T T T F	1 2 0 2	$ \begin{array}{c} $	T T T T T	1 2 0 2	$ \begin{array}{c} p_{v} \\ v_{3} \\ v_{1} \\ 0 \\ v_{1} \end{array} $	known T T T T T	<i>d</i> _v 1 2 0 2	$ \begin{array}{c} p_{\nu} \\ \nu_{3} \\ \nu_{1} \\ 0 \\ \nu_{1} \end{array} $	known T T T T T	d _v 1 2 0 2	
v ₂ v ₃ v ₄ v ₅	T T F F	1 2 0 2 3		Rnown T T T T F	1 2 0 2 3	$ \begin{array}{c} p_{v} \\ \nu_{3} \\ \nu_{1} \\ 0 \\ \nu_{1} \\ \nu_{2} \end{array} $	known T T T T T T	d _v 1 2 0 2 3	$ \begin{array}{c} p_{\nu} \\ \nu_{3} \\ \nu_{1} \\ 0 \\ \nu_{1} \\ \nu_{2} \end{array} $	known T T T T T T	d _v 1 2 0 2 3	$ \begin{array}{c} p_{\nu} \\ \nu_{3} \\ \nu_{1} \\ 0 \\ \nu_{1} \\ \nu_{2} \end{array} $
$v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6$	T T F F T	1 2 0 2 3 1	p_{v} v_{3} v_{1} 0 v_{1} v_{2} v_{3}	Rnown T T T T F T	$\begin{array}{c} a_{v} \\ 1 \\ 2 \\ 0 \\ 2 \\ 3 \\ 1 \end{array}$	$ \begin{array}{c} p_{v} \\ \nu_{3} \\ \nu_{1} \\ 0 \\ \nu_{1} \\ \nu_{2} \\ \nu_{3} \end{array} $	known T T T T T T T	d _v 1 2 0 2 3 1	$ \begin{array}{c} p_{\nu} \\ v_{3} \\ v_{1} \\ 0 \\ v_{1} \\ v_{2} \\ v_{3} \end{array} $	known T T T T T T T	d _v 1 2 0 2 3 1	p_{v} v_{3} v_{1} 0 v_{1} v_{2} v_{3}
v ₂ v ₃ v ₄ v ₅ v ₆ v ₇	T T F F T F	$ \begin{array}{c} 1\\ 2\\ 0\\ 2\\ 3\\ 1\\ \infty \end{array} $		Rnown T T T T F T F	a_v 1 2 0 2 3 1 3	$ \begin{array}{r} P_{\nu} \\ v_{3} \\ v_{1} \\ 0 \\ v_{1} \\ v_{2} \\ v_{3} \\ v_{4} \end{array} $	known T T T T T T F	d _v 1 2 0 2 3 1 3	p_{v} v_{3} v_{1} 0 v_{1} v_{2} v_{3} v_{4}	known T T T T T T T T	d_{v} 1 2 0 2 3 1 3	$ \begin{array}{r} p_{v} \\ v_{3} \\ v_{1} \\ 0 \\ v_{1} \\ v_{2} \\ v_{3} \\ v_{4} \end{array} $

(日)



▲□ > ▲圖 > ▲ 国 > ▲ 国 >

Outline

- The Shortest Path Problem
- 2 Shortest Path Algorithms
 - Shortest Path Problems
 - Single-source shortest path problem
 - Point to point shortest path problem
 - All pairs shortest path problem
 - Negative weights shortest path problem
 - Unweighted Shortest Paths
 - Weighted Shortest PathsDijkstra's algorithm

- ∢ ∃ ▶

Use Dijkstras algorithm:

• GREEDY strategy: Always pick the next closest vertex to the source.

• Use priority queue to store unvisited vertices by distance from *s*.

• Does not work with negative weights.

(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

Use Dijkstras algorithm:

- GREEDY strategy: Always pick the next closest vertex to the source.
- Use priority queue to store unvisited vertices by distance from *s*.

• Does not work with negative weights.

(I) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1)) < ((1))

Use Dijkstras algorithm:

- GREEDY strategy: Always pick the next closest vertex to the source.
- Use priority queue to store unvisited vertices by distance from *s*.
- Does not work with negative weights.

• Input: Vertices with cost.

- **Output**: The shortest path with the cost between *s* and *v*.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s, \lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

- Input: Vertices with cost.
- **Output**: The shortest path with the cost between *s* and *v*.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s, \lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

A D > A B > A B > A B >

- Input: Vertices with cost.
- Output: The shortest path with the cost between s and v.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s, \lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

イロト 不得 トイヨト イヨト 二日

- Input: Vertices with cost.
- **Output**: The shortest path with the cost between *s* and *v*.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s$, $\lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

- Input: Vertices with cost.
- **Output**: The shortest path with the cost between *s* and *v*.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s, \lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

- Input: Vertices with cost.
- **Output**: The shortest path with the cost between *s* and *v*.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s, \lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

- Input: Vertices with cost.
- **Output**: The shortest path with the cost between *s* and *v*.
- Step 1. Set $\lambda(s) = 0$ and for all vertices $v \neq s, \lambda(v) = \infty$. Set T = V, the vertex set of *G*.
- Step 2. Let *u* be a vertex in *T* for which $\lambda(u)$ is minimum.
- **Step 3**. If *u* = *t*, then stop.
- Step 4. For every edge e = uv incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + \omega(e)$ change the value of $\lambda(v)$ to $\lambda(u) + \omega(e)$.
- Step 5. Change T to $T \{u\}$ and go to step 2.

An Example













イロト イヨト イヨト イヨト

Dijkstra's algorithm

Why Dijkstra's Algorithm Works

- A least-cost path from vertex X to vertex Y contains least-cost paths from X to every vertex on the path to Y.

- A least-cost path from vertex X to vertex Y contains least-cost paths from X to every vertex on the path to Y.
- E.g., if $X \to C_1 \to C_2 \to C_3 \to Y$ is the least-cost path from X to Y, then
 - $X \to C_1 \to C_2 \to C_3$ is the least-cost path from X to C_3 .
 - $X \to C_1 \to C_2$ is the least-cost path from X to C_2 .
 - $X \to C_1$ is the least-cost path from X to C_1 .

- A least-cost path from vertex X to vertex Y contains least-cost paths from X to every vertex on the path to Y.
- E.g., if $X \to C_1 \to C_2 \to C_3 \to Y$ is the least-cost path from X to Y, then
 - $X \rightarrow C_1 \rightarrow C_2 \rightarrow C_3$ is the least-cost path from X to C_3 .
 - $X \rightarrow C_1 \rightarrow C_2$ is the least-cost path from X to C_2 .
 - $X \to C_1$ is the least-cost path from X to C_1 .

- A least-cost path from vertex X to vertex Y contains least-cost paths from X to every vertex on the path to Y.
- E.g., if $X \to C_1 \to C_2 \to C_3 \to Y$ is the least-cost path from X to Y, then
 - $X \rightarrow C_1 \rightarrow C_2 \rightarrow C_3$ is the least-cost path from X to C_3 .
 - $X \rightarrow C_1 \rightarrow C_2$ is the least-cost path from X to C_2 .
 - $X \rightarrow C_1$ is the least-cost path from X to C_1 .

- A least-cost path from vertex X to vertex Y contains least-cost paths from X to every vertex on the path to Y.
- E.g., if $X \to C_1 \to C_2 \to C_3 \to Y$ is the least-cost path from X to Y, then
 - $X \rightarrow C_1 \rightarrow C_2 \rightarrow C_3$ is the least-cost path from X to C_3 .
 - $X \rightarrow C_1 \rightarrow C_2$ is the least-cost path from X to C_2 .
 - $X \rightarrow C_1$ is the least-cost path from X to C_1 .

What about graphs with negative edges?

• Problem: Will the Dijkstras algorithm work as it is?

• Solution: Do not mark any vertex as known. Instead allow multiple updates.

What about graphs with negative edges?

- Problem: Will the Dijkstras algorithm work as it is?
- Solution: Do not mark any vertex as known. Instead allow multiple updates.