Graph Algorithms

Peter Lammich

3. Februar 2020

Outline

Outline

1. Directed Graphs
2. Graph Traversal Algorithms
3. Shortest Path in Weighted Graphs
   - Single-Source Shortest Path
   - Bellman Ford Algorithm

Outline

Dijkstra's Algorithm

- *Relax node* = relax outgoing edges

procedure $\textsc{relax}(u)$
    for all v with $w(u, v) \neq \infty$ **do** $\textsc{relax}(u, v)$
procedure $\textsc{dijkstra}(s)$
    $F \leftarrow \emptyset$, $D \leftarrow \textsc{initEstimate}(\mathsf{s})$
    while $V \setminus F \neq \emptyset$ **do**
        $u \leftarrow \text{Some } u \in V \setminus F,\ D(u) \text{ minimal}$
        $P \leftarrow F \cup \{u\}$
        $\textsc{relax}(u)$
    return D

Dijkstra's Algorithm

- *Relax node* = relax outgoing edges
- Assume weights are positive: $w(u, v) > 0$

procedure RELAX(u)
  for all v with $w(u, v) \neq \infty$ **do** RELAX(u, v)

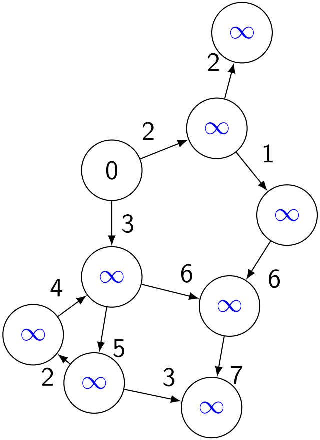
procedure DIJKSTRA(s)
  $F \leftarrow \emptyset$, $D \leftarrow$ INITESTIMATE(s)
  while $V \setminus F \neq \emptyset$ **do**
    $u \leftarrow$ Some $u \in V \setminus F$, $D(u)$ minimal
    $P \leftarrow F \cup \{u\}$
    RELAX(u)
  return D

Dijkstra's Algorithm

- *Relax node* = relax outgoing edges
- Assume weights are positive: $w(u, v) > 0$
- Relax node with minimal estimate. Iterate until all nodes relaxed.

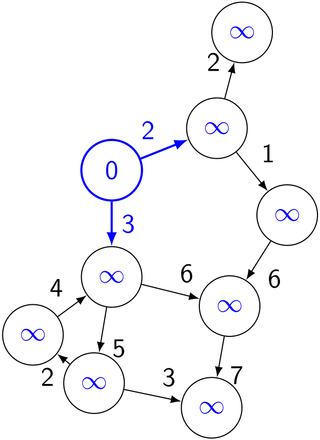
procedure $\textsc{relax}(u)$
  for all v with $w(u, v) \neq \infty$ **do** $\textsc{relax}(u, v)$
procedure $\textsc{dijkstra}(s)$
  $F \leftarrow \emptyset$, $D \leftarrow \textsc{initEstimate}(\mathsf{s})$
  while $V \setminus F \neq \emptyset$ **do**
    $u \leftarrow \text{Some } u \in V \setminus F,\ D(u) \text{ minimal}$
    $P \leftarrow F \cup \{u\}$
    $\textsc{relax}(u)$
  return D

Example



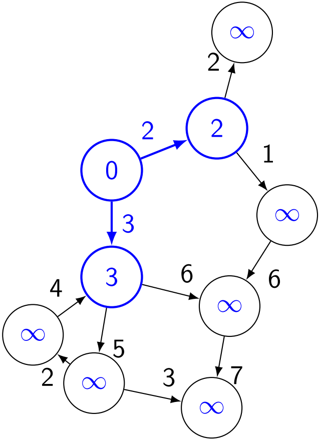
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



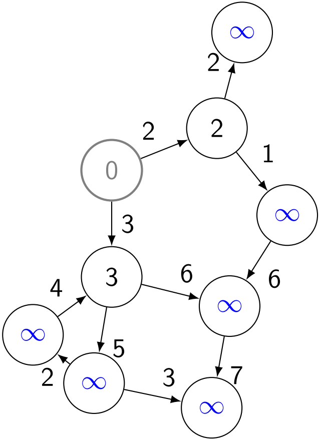
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



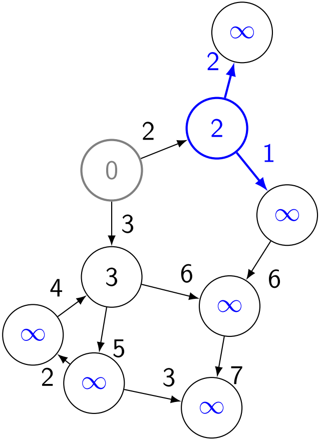
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



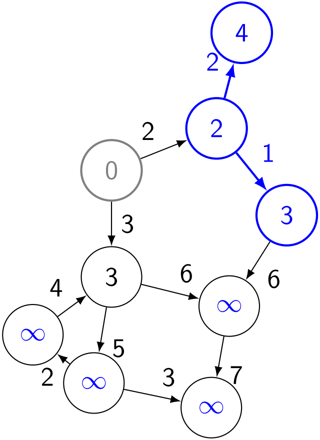
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



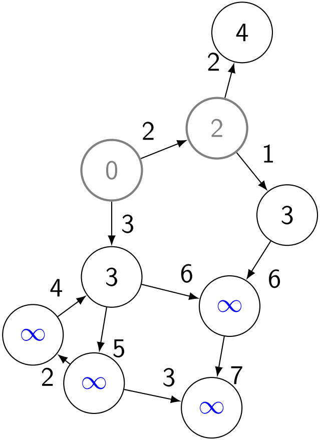
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



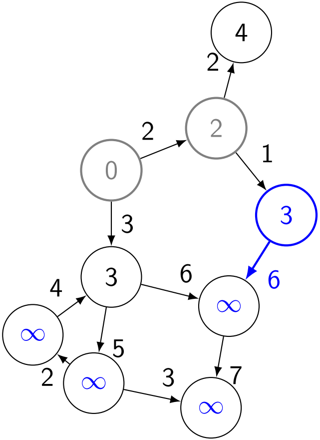
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



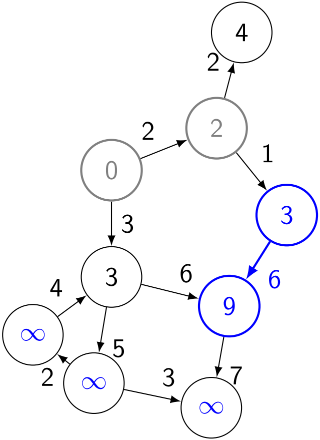
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



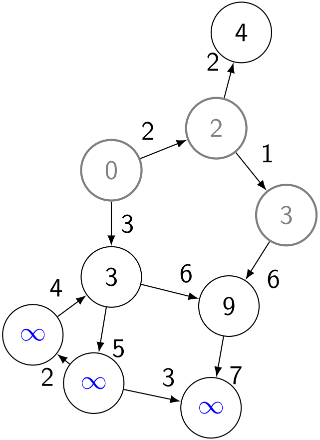
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



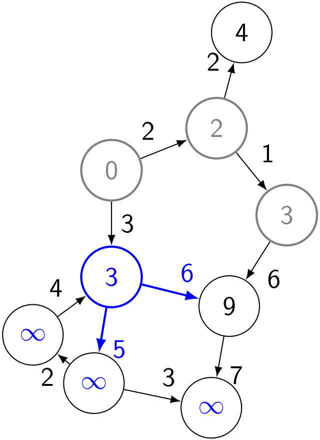
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



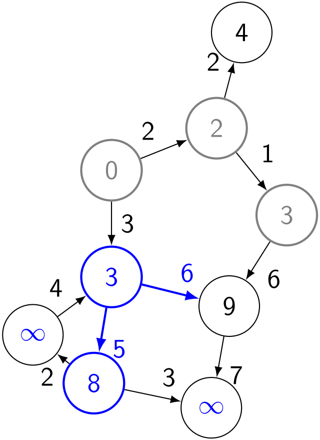
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



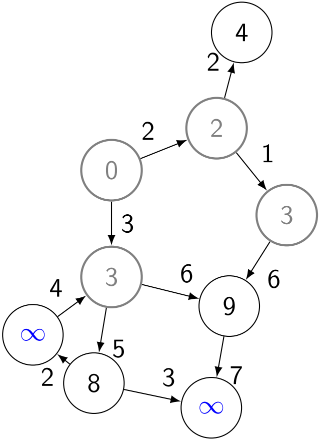
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



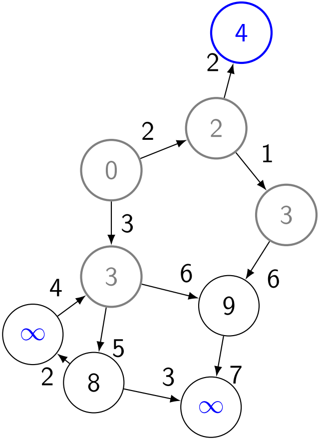
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



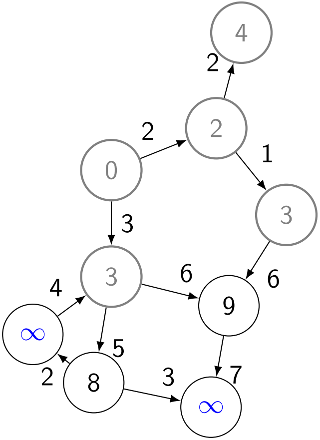
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



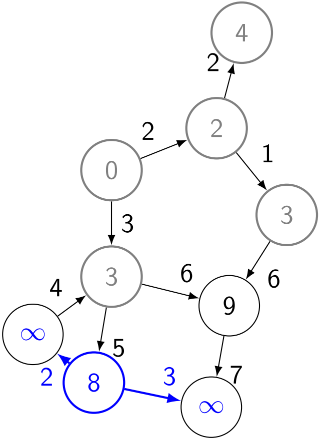
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



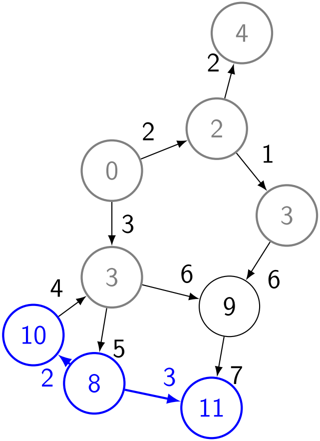
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



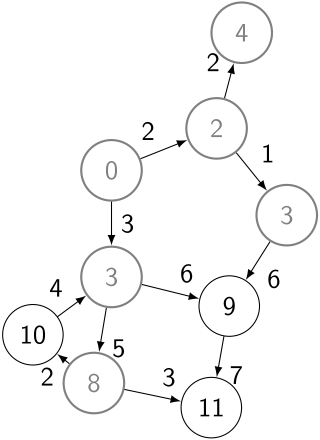
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



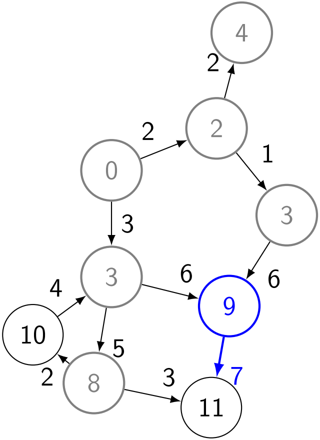
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



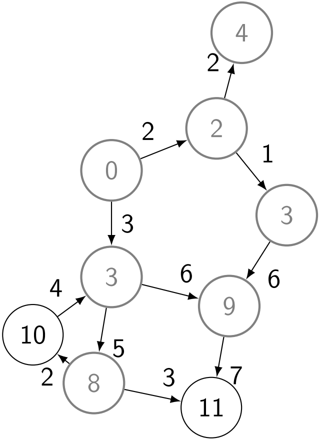
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



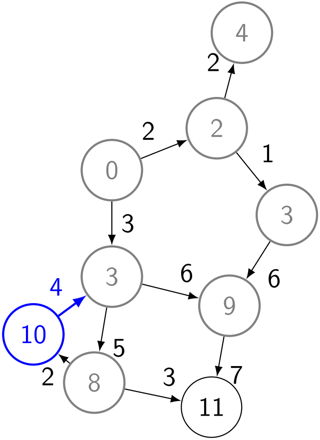
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



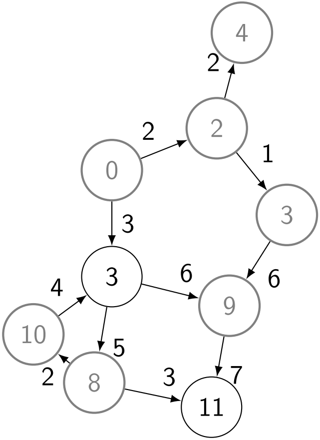
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



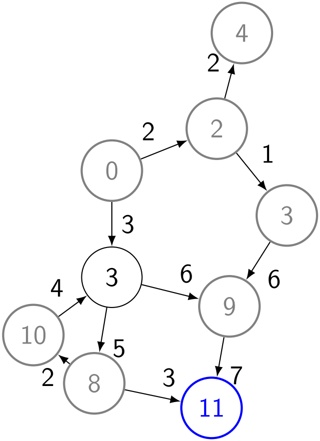
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



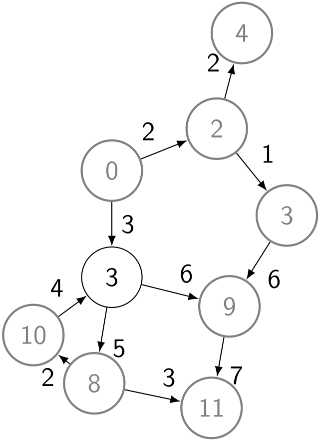
Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



Relax node with minimal estimate.
Iterate until all nodes relaxed.

Example



Relax node with minimal estimate.
Iterate until all nodes relaxed.

Dijkstra's Algorithm: Correctness

Relax node with minimal estimate. Iterate until all nodes relaxed.

Dijkstra's Algorithm: Correctness

Relax node with minimal estimate. Iterate until all nodes relaxed.

- Idea: Relaxed nodes have precise estimate
- As invariant: For all $u \in F$
  - $D(u) = \delta(u)$ (precise)
  - $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)
  - $\forall u \in V.\ D(u) \geq \delta(u)$ (over-estimate)
- Initially: $F = \emptyset$, holds trivially!
- First iteration: Relaxes s, and $D(s) = 0$ is precise
- Further iterations: See next slide!
- Finally: $F \supseteq V$, thus D precise for all nodes!

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u, v) of shortest path to v, we have $u \in F$ (*)

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u, v) of shortest path to v, we have $u \in F$ (*)
Thus, $\delta(v) = \delta(u) + w(u, v) \geq D(v)$

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u, v) of shortest path to v, we have $u \in F$ (*)
Thus, $\delta(v) = \delta(u) + w(u, v) \geq D(v)$

Show (*): Shortest path from $s \in F$ to $v \notin F$ leaves F

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u,v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u,v) of shortest path to v, we have $u \in F$ (*)
Thus, $\delta(v) = \delta(u) + w(u,v) \geq D(v)$

Show (*): Shortest path from $s \in F$ to $v \notin F$ leaves F
Let $(u', v') \in F \times (V \setminus F)$ be first edge leaving F

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u, v) of shortest path to v, we have $u \in F$ (*)
Thus, $\delta(v) = \delta(u) + w(u, v) \geq D(v)$

Show (*): Shortest path from $s \in F$ to $v \notin F$ leaves F
Let $(u', v') \in F \times (V \setminus F)$ be first edge leaving F
Assume $v' \neq v$, i.e., shortest path has form $sp_1 u' v' p_2 v$

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u, v) of shortest path to v, we have $u \in F$ (*)
Thus, $\delta(v) = \delta(u) + w(u, v) \geq D(v)$

Show (*): Shortest path from $s \in F$ to $v \notin F$ leaves F
Let $(u', v') \in F \times (V \setminus F)$ be first edge leaving F
Assume $v' \neq v$, i.e., shortest path has form $sp_1u'v'p_2v$
We have $D(v) \geq \delta(v) > \delta(v') = \delta(u') + w(u', v') \geq D(v')$

Dijkstra Algorithm: Invariant preservation

Assume $s \in F$ and for all $u \in F$

- $D(u) = \delta(u)$ (precise)
- $\forall v.\ D(v) \leq \delta(u) + w(u, v)$ (relaxed with precise $D(u)$)

We now relax $v \in V \setminus F$, with minimal $D(v)$

To show: $D(v)$ precise (i.e. $D(v) = \delta(v)$)
For last edge (u, v) of shortest path to v, we have $u \in F$ (*)
Thus, $\delta(v) = \delta(u) + w(u, v) \geq D(v)$

Show (*): Shortest path from $s \in F$ to $v \notin F$ leaves F
Let $(u', v') \in F \times (V \setminus F)$ be first edge leaving F
Assume $v' \neq v$, i.e., shortest path has form $sp_1 u' v' p_2 v$
We have $D(v) \geq \delta(v) > \delta(v') = \delta(u') + w(u', v') \geq D(v')$
Thus, we would have picked v' instead of v!

Implementing Dijkstra

- Use priority queue for nodes not yet relaxed
  - relaxation: priority of node already in queue may decrease
  - requires decrease-key operation
- Instead of adding all nodes to PQ initially, add nodes as they are discovered
  - unreachable nodes won't be explored
  - no ∞ in PQ required
- Use predecessor map to compute actual paths

Heaps with Decrease-Key

- Recall min-heaps.
- *sift-up* restores heap-property for element with too small priority
    - e.g. after we decreased its priority
    - needs index of element in heap!
- To find index of node in heap:
    - maintain map from node names to index in heap!

Complexity

- Operations:
  - every edge relaxed at most once,
  - every node added and extracted from PQ at most once
- Cost of relaxation, addition, and extraction: $O(\log |V|)$.
- Thus: $O((|E| + |V|) \log |V|)$

Outline

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map
  - for route to London, also route to Edinburgh will be computed!

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map
  - for route to London, also route to Edinburgh will be computed!
- A*-Algorithm

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map
  - for route to London, also route to Edinburgh will be computed!
- A*-Algorithm
  - heuristics $h(u)$ estimates distance to target

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map
  - for route to London, also route to Edinburgh will be computed!
- A*-Algorithm
  - heuristics $h(u)$ estimates distance to target
  - relax node with minimal $D(u) + h(u)$

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map
  - for route to London, also route to Edinburgh will be computed!
- A*-Algorithm
  - heuristics $h(u)$ estimates distance to target
  - relax node with minimal $D(u) + h(u)$
  - stop when t has been relaxed

Finding Shortest Path between two Nodes

- Dikstra's Algorithm finds shortest paths to all nodes
  - for routing, we only need paths between two nodes
  - Dijkstra computes useless information
- Stop Dijkstra when target node is relaxed
  - still computes useless information!
  - for all nodes with shorter paths than target node
  - even if they go into the opposite direction on the map
  - for route to London, also route to Edinburgh will be computed!
- A*-Algorithm
  - heuristics $h(u)$ estimates distance to target
  - relax node with minimal $D(u) + h(u)$
  - stop when t has been relaxed
  - recall: Dijkstra relaxes node with minimal $D(u)$

Pseudocode

procedure $\textsc{astar}(s, t)$
  $F \leftarrow \emptyset$, $D \leftarrow \textsc{initEstimate}(\mathsf{s})$
  while $V \setminus F \neq \emptyset$ **do**
    $u \leftarrow \text{Some } u \in V \setminus F,\ D(u) + h(u) \text{ minimal}$
    $P \leftarrow F \cup \{u\}$
    if $u = t$ **then return** $D(t)$
    $\textsc{relax}(u)$

Heuristics

- Let s be source, and t be target node

Heuristics

- Let s be source, and t be target node
- *Admissible*: distance never over-estimated

Heuristics

- Let s be source, and t be target node
- *Admissible*: distance never over-estimated
  - for all nodes u. $h(u) \leq \delta(u, t)$

Heuristics

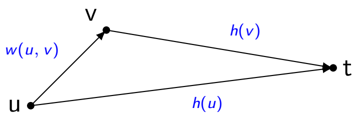
- Let s be source, and t be target node
- *Admissible*: distance never over-estimated
  - for all nodes u. $h(u) \leq \delta(u, t)$
- *Monotone*: estimate can't get better when taking edge

Heuristics

- Let s be source, and t be target node
- *Admissible*: distance never over-estimated
  - for all nodes u. $h(u) \leq \delta(u, t)$
- *Monotone*: estimate can't get better when taking edge
  - for all nodes u, v. $h(u) \leq w(u, v) + h(v)$

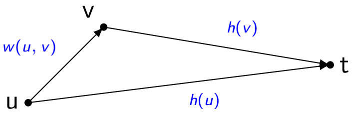
Heuristics

- Let s be source, and t be target node
- *Admissible*: distance never over-estimated
  - for all nodes u. $h(u) \leq \delta(u, t)$
- *Monotone*: estimate can't get better when taking edge
  - for all nodes u, v. $h(u) \leq w(u, v) + h(v)$
  - triangle inequation



Heuristics

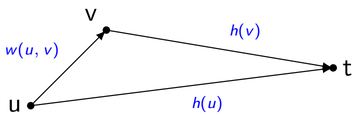
- Let s be source, and t be target node
- *Admissible*: distance never over-estimated
  - for all nodes u. $h(u) \leq \delta(u, t)$
- *Monotone*: estimate can't get better when taking edge
  - for all nodes u, v. $h(u) \leq w(u, v) + h(v)$
  - triangle inequation



  - monotonicity implies admissibility

Heuristics

- Let s be source, and t be target node
- *Admissible*: distance never over-estimated
  - for all nodes u. $h(u) \leq \delta(u, t)$
- *Monotone*: estimate can't get better when taking edge
  - for all nodes u, v. $h(u) \leq w(u, v) + h(v)$
  - triangle inequation



  - monotonicity implies admissibility
    - proof by induction over shortest path from u to t.

Correctness

- We assume monotone heuristics

Correctness

- We assume monotone heuristics
    - for non-monotone (but admissible) heuristics:

Correctness

- We assume monotone heuristics
    - for non-monotone (but admissible) heuristics:
        - relaxation may decrease estimate of finished nodes

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w
    - and we have $\forall u.\ D'(u) = D(u) + h(u)$

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w
    - and we have $\forall u.\ D'(u) = D(u) + h(u)$
    - obviously, minimal nodes coincide!

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w
    - and we have $\forall u.\ D'(u) = D(u) + h(u)$
    - obviously, minimal nodes coincide!
    - relaxation:
      Dijkstra relaxes $D'(v)$ with $D'(u) + w'(u, v)$

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w
    - and we have $\forall u.\ D'(u) = D(u) + h(u)$
    - obviously, minimal nodes coincide!
    - relaxation:
      Dijkstra relaxes $D'(v)$ with $D'(u) + w'(u, v)$
      $= D(u) + h(u) + w(u, v) + h(v) - h(u)$

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u,v) = w(u,v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w
    - and we have $\forall u.\ D'(u) = D(u) + h(u)$
    - obviously, minimal nodes coincide!
    - relaxation:
      Dijkstra relaxes $D'(v)$ with $D'(u) + w'(u,v)$
      $= D(u) + h(u) + w(u,v) + h(v) - h(u)$
      $= D(u) + w(u,v) \quad + \quad h(v)$

Correctness

- We assume monotone heuristics
  - for non-monotone (but admissible) heuristics:
    - relaxation may decrease estimate of finished nodes
    - those nodes must be *unfinished* again!
    - proof not covered in this lecture!
- Idea: Run of A* is equivalent to run of Dijkstra on *modified graph*
  - new weights: $w'(u, v) = w(u, v) + h(v) - h(u)$
    - monotonicity guarantees they are positive!
  - initialization: $D'(s) = h(s)$
  - Dijkstra on w', D' will relax the same nodes as A* on D, w
    - and we have $\forall u.\ D'(u) = D(u) + h(u)$
    - obviously, minimal nodes coincide!
    - relaxation:

      Dijkstra relaxes $D'(v)$ with $D'(u) + w'(u, v)$

      $= D(u) + h(u) + w(u, v) + h(v) - h(u)$

      $= D(u) + w(u, v) \quad + \quad h(v)$

      A* relaxes $D(v)$ with $D(u) + w(u, v)$ $\square$

Complexity

- In worst case, same as Dijkstra: $O((|V| + |E|) \log |V|)$
  - all other nodes gets relaxed before target node

Complexity

- In worst case, same as Dijkstra: $O((|V| + |E|) \log |V|)$
  - all other nodes gets relaxed before target node
- But, for practical problems, typically much better!