Learning Goals

- Dijkstra algorithm: the problem it solves and the description of the algorithm
- Analysis: an inductive proof of correctness
- Running time of Dijkstra's algorithm
- (Optional) Implementation of Dijkstra's algorithm using priority queues

Finding minimum-cost paths in a graph

- Input: a directed graph $G = (V, E)$, with nonnegative cost $c_e \geq 0$ for each edge $e \in E$. A node $s \in V$.
- Output: for each node $v \in V$, a minimum-cost path from s to v, and its cost.

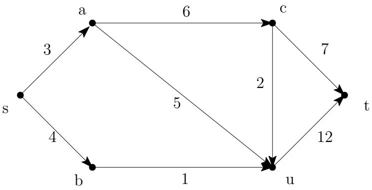
Finding minimum-cost paths in a graph

- Input: a directed graph $G = (V, E)$, with nonnegative cost $c_e \geq 0$ for each edge $e \in E$. A node $s \in V$.
- Output: for each node $v \in V$, a minimum-cost path from s to v, and its cost.
- Dijkstra's algorithm: a greedy approach

Finding minimum-cost paths in a graph

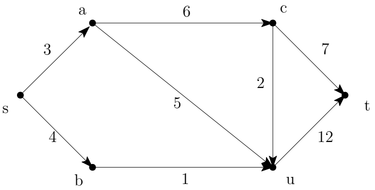
- Input: a directed graph $G = (V, E)$, with nonnegative cost $c_e \geq 0$ for each edge $e \in E$. A node $s \in V$.
- Output: for each node $v \in V$, a minimum-cost path from s to v, and its cost.
- Dijkstra's algorithm: a greedy approach
- Idea: Find a minimum-cost path to a new node in each step, and then use the cost to reach this node to update the cost to reach the other nodes one step further.

Dijkstra example

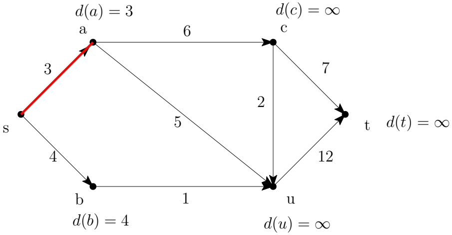


A graph with nonnegative edge costs

Dijkstra example

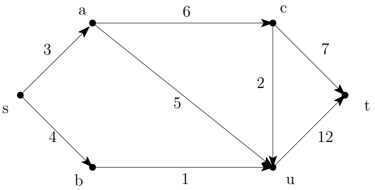


A graph with nonnegative edge costs

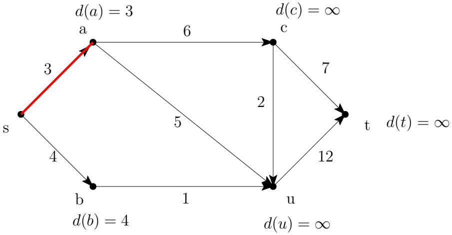


Dijkstra Step 1

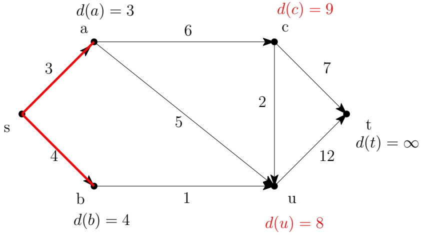
Dijkstra example



A graph with nonnegative edge costs

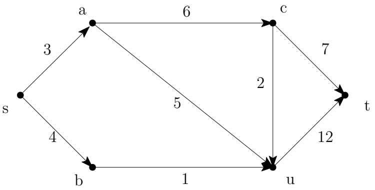


Dijkstra Step 1

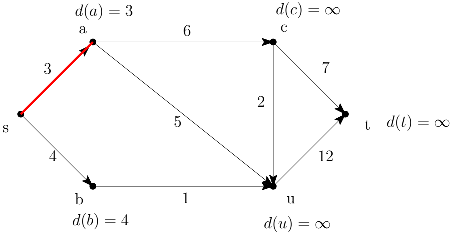


Dijkstra Step 2

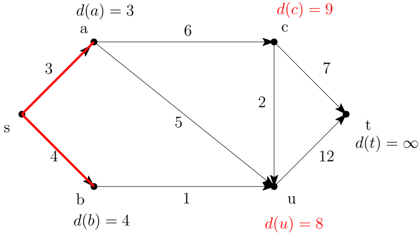
Dijkstra example



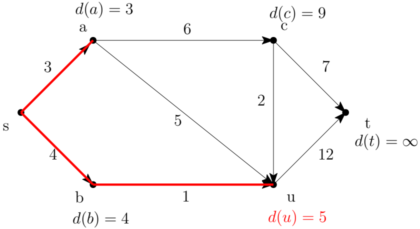
A graph with nonnegative edge costs



Dijkstra Step 1

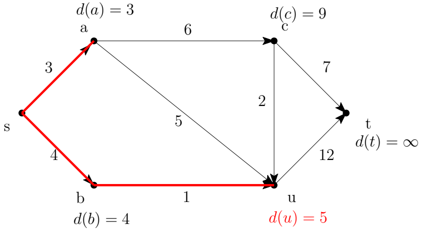


Dijkstra Step 2



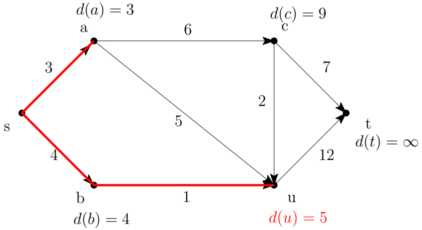
Dijkstra Step 3

Dijkstra example cont.

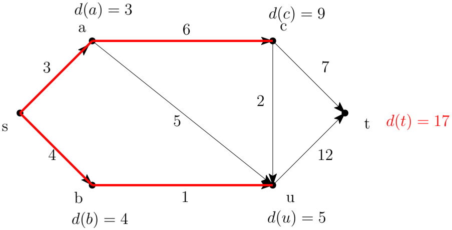


Dijkstra Step 3

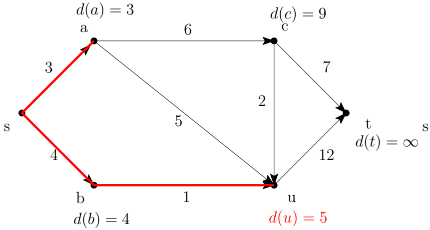
Dijkstra example cont.



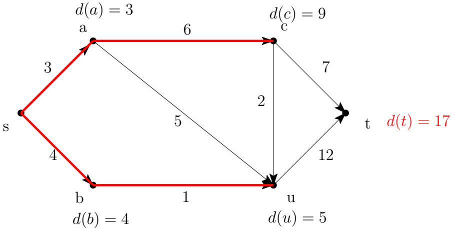
Dijkstra Step 3



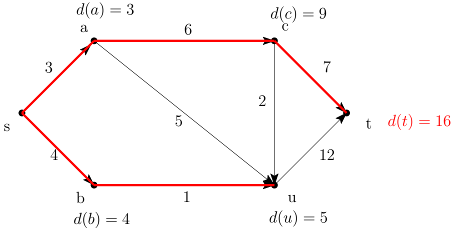
Dijkstra Step 4



Dijkstra Step 3



Dijkstra Step 4



Dijkstra Step 5

Dijkstra algorithm

- Initialize: for each $v \in V$, if $(s, v) \in E$, let $d(v) \leftarrow c_{(s,v)}$, $p(v) \leftarrow s$, otherwise $d(v) \leftarrow \infty$, $p(v) \leftarrow \perp$. Let S be $\{s\}$.
- Meaning: $d(v)$: cost of the min-cost path to v found so far; $p(v)$: the node preceding v in the minimum-cost path to v.

Dijkstra algorithm

- Initialize: for each $v \in V$, if $(s, v) \in E$, let $d(v) \leftarrow c_{(s,v)}$, $p(v) \leftarrow s$, otherwise $d(v) \leftarrow \infty$, $p(v) \leftarrow \perp$. Let S be $\{s\}$.
- Meaning: $d(v)$: cost of the min-cost path to v found so far; $p(v)$: the node preceding v in the minimum-cost path to v.
- Iterate: while $S \neq V$ and there exists $v \in V \setminus S$ such that $d(v) \neq \infty$:
  - let u be the minimizer of $d(\cdot)$ among nodes not in S;
  - add u to S
  - for each $(u, v) \in E$ with $v \notin S$, if $d(v) > d(u) + c_{(u,v)}$
    - update $d(v) \leftarrow d(u) + c_{(u,v)}$
    - $p(v) \leftarrow u$.

Dijkstra algorithm

- Initialize: for each $v \in V$, if $(s, v) \in E$, let $d(v) \leftarrow c_{(s,v)}$, $p(v) \leftarrow s$, otherwise $d(v) \leftarrow \infty$, $p(v) \leftarrow \perp$. Let S be $\{s\}$.
- Meaning: $d(v)$: cost of the min-cost path to v found so far; $p(v)$: the node preceding v in the minimum-cost path to v.
- Iterate: while $S \neq V$ and there exists $v \in V \setminus S$ such that $d(v) \neq \infty$:
  - let u be the minimizer of $d(\cdot)$ among nodes not in S;
  - add u to S
  - for each $(u, v) \in E$ with $v \notin S$, if $d(v) > d(u) + c_{(u,v)}$
    - update $d(v) \leftarrow d(u) + c_{(u,v)}$
    - $p(v) \leftarrow u$.
- Output:
  - For each $v \in S$, $d(v)$ is the cost of the min-cost path from s to v; the path is traced back to s using $p(\cdot)$.
  - For $v \notin S$, there is no path from s to v.

Proof of Correctness

- Proof by induction.

Proof of Correctness

- Proof by induction.
- Induction hypothesis: at each stage of the algorithm, for any node $u \in S$, $d(u)$ is the cost of the minimum-cost path from s to u.

Proof of Correctness

- Proof by induction.
- Induction hypothesis: at each stage of the algorithm, for any node $u \in S$, $d(u)$ is the cost of the minimum-cost path from s to u.
- Base case: $|S| = 1$: $S = \{s\}$, $d(s) = 0$, trivial.

Proof of Correctness

- Proof by induction.
- Induction hypothesis: at each stage of the algorithm, for any node $u \in S$, $d(u)$ is the cost of the minimum-cost path from s to u.
- Base case: $|S| = 1$: $S = \{s\}$, $d(s) = 0$, trivial.
- Inductive step: suppose the induction hypothesis when $|S| \leq k$. Show that, when the $(k+1)^{\text{-st}}$ node u is added to S, the hypothesis remains true.

Proof of Correctness

- Proof by induction.
- Induction hypothesis: at each stage of the algorithm, for any node $u \in S$, $d(u)$ is the cost of the minimum-cost path from s to u.
- Base case: $|S| = 1$: $S = \{s\}$, $d(s) = 0$, trivial.
- Inductive step: suppose the induction hypothesis when $|S| \leq k$. Show that, when the $(k+1)^{\text{-st}}$ node u is added to S, the hypothesis remains true.
- Denote by P_u the path output by the algorithm for node u.

Proof of Correctness

- Proof by induction.
- Induction hypothesis: at each stage of the algorithm, for any node $u \in S$, $d(u)$ is the cost of the minimum-cost path from s to u.
- Base case: $|S| = 1$: $S = \{s\}$, $d(s) = 0$, trivial.
- Inductive step: suppose the induction hypothesis when $|S| \leq k$. Show that, when the $(k+1)^{\text{-st}}$ node u is added to S, the hypothesis remains true.
- Denote by P_u the path output by the algorithm for node u.
- We should show:
    1. Among all paths within S, P_u has the minimum cost.

Proof of Correctness

- Proof by induction.
- Induction hypothesis: at each stage of the algorithm, for any node $u \in S$, $d(u)$ is the cost of the minimum-cost path from s to u.
- Base case: $|S| = 1$: $S = \{s\}$, $d(s) = 0$, trivial.
- Inductive step: suppose the induction hypothesis when $|S| \leq k$. Show that, when the $(k+1)^{\text{-st}}$ node u is added to S, the hypothesis remains true.
- Denote by P_u the path output by the algorithm for node u.
- We should show:
  1. Among all paths within S, P_u has the minimum cost.
  2. P_u has no more cost than any path from s to u that leaves S at some point.

Proof of Correctness Cont.

We should show:

1. Among all paths within S, P_u has the minimum cost.
2. P_u has no more cost than any path from s to u that leaves S at some point.

Proof of Correctness Cont.

We should show:

1. Among all paths within S, P_u has the minimum cost.
2. P_u has no more cost than any path from s to u that leaves S at some point.

Proof of each statement:

1. Let $p(u)$ be $v \in S$.
    - By algorithm, we know $d(u) = d(v) + c_{(v,u)}$ and it is no larger than $d(w) + c_{(w,u)}$ for any $w \in S$.

Proof of Correctness Cont.

We should show:

1. Among all paths within S, P_u has the minimum cost.
2. P_u has no more cost than any path from s to u that leaves S at some point.

Proof of each statement:

1. Let $p(u)$ be $v \in S$.
    - By algorithm, we know $d(u) = d(v) + c_{(v,u)}$ and it is no larger than $d(w) + c_{(w,u)}$ for any $w \in S$.
    - By induction hypothesis, $d(w)$ is the cost of the min-cost path to w, hence $d(w) + c_{(w,u)}$ is the cost of min-cost path to u with the second last node being w.

Proof of Correctness Cont.

We should show:

1. Among all paths within S, P_u has the minimum cost.
2. P_u has no more cost than any path from s to u that leaves S at some point.

Proof of each statement:

1. Let $p(u)$ be $v \in S$.
   - By algorithm, we know $d(u) = d(v) + c_{(v,u)}$ and it is no larger than $d(w) + c_{(w,u)}$ for any $w \in S$.
   - By induction hypothesis, $d(w)$ is the cost of the min-cost path to w, hence $d(w) + c_{(w,u)}$ is the cost of min-cost path to u with the second last node being w.
2. For any path P' that leaves S by an edge (w, z), $w \in S$, $z \notin S$:
   - The cost of P' is at least $d(w) + c_{(w,z)}$, because by induction hypothesis $d(w)$ is the cost of min-cost path to w, and the part of P from z to u adds nonnegative cost.

Proof of Correctness Cont.

We should show:

1. Among all paths within S, P_u has the minimum cost.
2. P_u has no more cost than any path from s to u that leaves S at some point.

Proof of each statement:

1. Let $p(u)$ be $v \in S$.
   - By algorithm, we know $d(u) = d(v) + c_{(v,u)}$ and it is no larger than $d(w) + c_{(w,u)}$ for any $w \in S$.
   - By induction hypothesis, $d(w)$ is the cost of the min-cost path to w, hence $d(w) + c_{(w,u)}$ is the cost of min-cost path to u with the second last node being w.
2. For any path P' that leaves S by an edge (w, z), $w \in S$, $z \notin S$:
   - The cost of P' is at least $d(w) + c_{(w,z)}$, because by induction hypothesis $d(w)$ is the cost of min-cost path to w, and the part of P from z to u adds nonnegative cost.
   - $d(u) \leq d(w) + c_{(w,z)}$, because when $d(u)$ is added to S, $d(z) \leq d(w) + c_{(w,z)}$, and $d(u) \leq d(z)$.

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.
- Implementation using priority queue

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.
- Implementation using priority queue
  - Store $d(\cdot)$ using a priority queue, which allows us to find and delete a minimum element in $O(\log n)$ time

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.
- Implementation using priority queue
  - Store $d(\cdot)$ using a priority queue, which allows us to find and delete a minimum element in $O(\log n)$ time
  - Updating an element now also needs $O(\log n)$ time

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.
- Implementation using priority queue
  - Store $d(\cdot)$ using a priority queue, which allows us to find and delete a minimum element in $O(\log n)$ time
  - Updating an element now also needs $O(\log n)$ time
  - We may update elements in the queue at most m times.

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.
- Implementation using priority queue
  - Store $d(\cdot)$ using a priority queue, which allows us to find and delete a minimum element in $O(\log n)$ time
  - Updating an element now also needs $O(\log n)$ time
  - We may update elements in the queue at most m times.
  - Total Running time: $O(m \log n)$.

Implementation of Dijkstra

- Implementation with an array that stores $d(\cdot)$:
  - Each time we look for a minimum $d(\cdot)$, search the array in $O(n)$ time; this is done n times, so this takes $O(n^2)$ time.
  - Each edge is traversed at most once, each involving constant time; so overall this takes $O(m)$ times.
  - Altogether $O(n^2 + m) = O(n^2)$ time.
- Implementation using priority queue
  - Store $d(\cdot)$ using a priority queue, which allows us to find and delete a minimum element in $O(\log n)$ time
  - Updating an element now also needs $O(\log n)$ time
  - We may update elements in the queue at most m times.
  - Total Running time: $O(m \log n)$.
- Choose the better one depending on how dense the graph is. Overall running time $O(\min(n^2, m \log n))$.