Dijkstra's algorithm

October 30, 2017

Overview

1. Dijkstra's algorithm

Dijkstra's algorithm

- Let $G = (V, E)$ be a directed graph with a weight function $w : E \to \mathbb{R}_{>0}$ defined on its edges.
- For an edge $(u, v) \notin E$ we define $w(u, v) = \infty$.
- For 2 vertices in the graph G we define the weighted distance from u to v by:

$$d(u, v) = \min\left\{ \sum_{e \in P} w(e) \;\middle|\; P \text{ is a directed path from } u \text{ to } v \right\}$$

Dijkstra's algorithm

- Let $G = (V, E)$ be a directed graph with a weight function $w : E \to \mathbb{R}_{>0}$ defined on its edges.
- For an edge $(u, v) \notin E$ we define $w(u, v) = \infty$.
- For 2 vertices in the graph G we define the weighted distance from u to v by:

$$d(u,v) = \min\left\{ \sum_{e \in P} w(e) \;\middle|\; P \text{ is a directed path from } u \text{ to } v \right\}$$

Dijkstra's algorithm

- Let $G = (V, E)$ be a directed graph with a weight function $w : E \to \mathbb{R}_{>0}$ defined on its edges.
- For an edge $(u, v) \notin E$ we define $w(u, v) = \infty$.
- For 2 vertices in the graph G we define the weighted distance from u to v by:

$$d(u, v) = \min \left\{ \sum_{e \in P} w(e) \;\middle|\; P \text{ is a directed path from } u \text{ to } v \right\}$$

Dijkstra's algorithm

- Given $G = (V, E)$, $w : E \to \mathbb{R}_{>0}$ and a vertex $u \in V$ we want to compute $d(u, v)$ for all the vertices $v \in V$.
(we denote $n = |V|$)
- Dijkstra's algorithm solves this problem.
(It also finds such a path.)
- The idea is:
Define a sequence of sets $S_1 \subset S_2 \subset ... \subset S_n = V$ and a sequence of functions $d_1, d_2, ..., d_n$ such that:

$$d_i : V \to \mathbb{R}$$

and

$$d_i(u, v) = d(u, v) \text{ for any vertex } v \in S_i$$

Dijkstra's algorithm

- Given $G = (V, E)$, $w : E \to \mathbb{R}_{>0}$ and a vertex $u \in V$ we want to compute $d(u, v)$ for all the vertices $v \in V$.
  (we denote $n = |V|$)
- Dijkstra's algorithm solves this problem.
  (It also finds such a path.)
- The idea is:
  Define a sequence of sets $S_1 \subset S_2 \subset ... \subset S_n = V$ and a sequence of functions $d_1, d_2, ..., d_n$ such that:

$$d_i : V \to \mathbb{R}$$

and

$$d_i(u, v) = d(u, v) \text{ for any vertex } v \in S_i$$

Dijkstra's algorithm

- Given $G = (V, E)$, $w : E \to \mathbb{R}_{>0}$ and a vertex $u \in V$ we want to compute $d(u, v)$ for all the vertices $v \in V$.
  (we denote $n = |V|$)
- Dijkstra's algorithm solves this problem.
  (It also finds such a path.)
- The idea is:
  Define a sequence of sets $S_1 \subset S_2 \subset ... \subset S_n = V$ and a sequence of functions $d_1, d_2, ..., d_n$ such that:

  $$d_i : V \to \mathbb{R}$$

  and

  $$d_i(u, v) = d(u, v) \text{ for any vertex } v \in S_i$$

Dijkstra's algorithm

- Such approach is sometimes called relaxation.
- The function $d(v)$ is an estimate for the distance we are trying to calculate.
- The function $d(v)$ is initialized to ∞.
  (The parent function $\pi(v)$ is initialized to null.)
- If an edge (u, v) allows to get to v on a shorter path, then it is used to update the value of $d(v)$ (and $\pi(v)$).

Dijkstra's algorithm

- Such approach is sometimes called relaxation.
- The function $d(v)$ is an estimate for the distance we are trying to calculate.
- The function $d(v)$ is initialized to ∞.
  (The parent function $\pi(v)$ is initialized to null.)
- If an edge (u, v) allows to get to v on a shorter path, then it is used to update the value of $d(v)$ (and $\pi(v)$).

Dijkstra's algorithm

- Such approach is sometimes called relaxation.
- The function $d(v)$ is an estimate for the distance we are trying to calculate.
- The function $d(v)$ is initialized to ∞.
  (The parent function $\pi(v)$ is initialized to null.)
- If an edge (u, v) allows to get to v on a shorter path, then it is used to update the value of $d(v)$ (and $\pi(v)$).

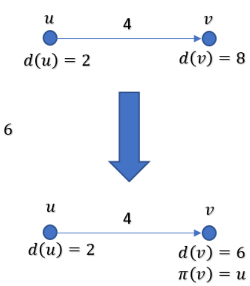
Dijkstra's algorithm

- Such approach is sometimes called relaxation.
- The function $d(v)$ is an estimate for the distance we are trying to calculate.
- The function $d(v)$ is initialized to ∞.
  (The parent function $\pi(v)$ is initialized to null.)
- If an edge (u, v) allows to get to v on a shorter path, then it is used to update the value of $d(v)$ (and $\pi(v)$).

Using the edge (u, v) we update:

$$d(u) = d(u) + w(u, v) = 2 + 4 = 6$$

$$\pi(v) = u$$



Dijkstra's algorithm

- The algorithm:
  Initialize: for $k = 1$ we set $S_1 = \{v_1\}$ where $v_1 = u$
  and:

$$d_1(u,v) = w(u,v) \text{ (Note that this is } \infty \text{ if the edge does not exist)}$$

and

$$\pi_1(v) = \begin{cases} u, & v \neq u; \\ \emptyset, & v = u. \end{cases}$$

Step: Given $S_k = \{v_1, \dots, v_k\}$, $d_k : V \to \mathbb{R}$, and $\pi_k : V \to V \cup \emptyset$,
we find $v_{k+1} \in V - S_k$ such that:

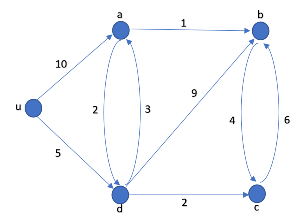
$$d(v_{k+1}) = min\{d_k(v) \mid v \in V - S_k\}$$

and we define $S_{k+1} = S_k \cup \{v_{k+1}\}$ and:

$$d_{k+1}(v) = \begin{cases} d_k(v), & v \in S_{k+1}; \\ \min\{d_k(v), d_k(v_{k+1}) + w(v_{k+1}, v)\}, & v \notin S_{k+1}. \end{cases}$$

$$\pi_{k+1}(v) = \begin{cases} \pi_k(v), & v \in S_{k+1}; \\ \pi_k(v), & v \notin S_{k+1} \text{ and } d_k(v) \leq d_k(v_{k+1}) + w(v_{k+1}, v). \\ v_{k+1}, & v \notin S_{k+1} \text{ and } d_k(v) > d_k(v_{k+1}) + w(v_{k+1}, v) . \end{cases}$$

Dijkstra's algorithm - example

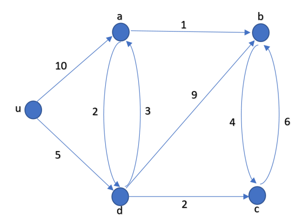


$S_1 = \{u\}$

d_1				
u	a	b	c	d
0	10	∞	∞	5

π_1				
u	a	b	c	d
$\emptyset$	u	u	u	u

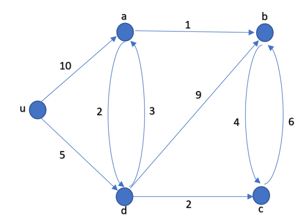
Dijkstra's algorithm - example



$$S_1 = \{u\} \quad \begin{array}{|ccccc|}\multicolumn{5}{c}{d_1}\\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d}\\ 0 & 10 & \infty & \infty & 5\end{array} \quad \begin{array}{|ccccc|}\multicolumn{5}{c}{\pi_1}\\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d}\\ \emptyset & \text{u} & \text{u} & \text{u} & \text{u}\end{array}$$

$$S_2 = \{u, d\} \quad \begin{array}{|ccccc|}\multicolumn{5}{c}{d_2}\\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d}\\ 0 & 5+3 & 5+9 & 5+2 & 5\end{array} \quad \begin{array}{|ccccc|}\multicolumn{5}{c}{\pi_2}\\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d}\\ \emptyset & \text{d} & \text{d} & \text{d} & \text{u}\end{array}$$

Dijkstra's algorithm - example



$S_2 = \{u, d\}$

d_2				
u	a	b	c	d
0	8	14	7	5

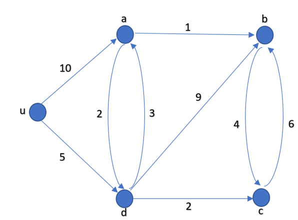
π_2				
u	a	b	c	d
$\emptyset$	d	d	d	u

$S_3 = \{u, d, c\}$

d_3				
u	a	b	c	d
0	8	7+6	7	5

π_3				
u	a	b	c	d
$\emptyset$	d	c	d	u

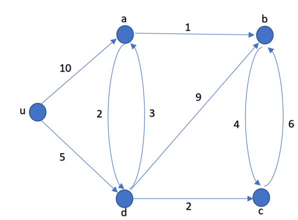
Dijkstra's algorithm - example



$$S_3 = \{u, d, c\} \quad \left|\begin{array}{ccccc} & & d_3 & & \\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d} \\ 0 & 8 & 13 & 7 & 5 \end{array}\right| \quad \left|\begin{array}{ccccc} & & \pi_3 & & \\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d} \\ \emptyset & \text{d} & \text{c} & \text{d} & \text{u} \end{array}\right|$$

$$S_4 = \{u, d, c, a\} \quad \left|\begin{array}{ccccc} & & d_4 & & \\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d} \\ 0 & 8 & 8{+}1 & 7 & 5 \end{array}\right| \quad \left|\begin{array}{ccccc} & & \pi_4 & & \\ \text{u} & \text{a} & \text{b} & \text{c} & \text{d} \\ \emptyset & \text{d} & \text{a} & \text{d} & \text{u} \end{array}\right|$$

Dijkstra's algorithm - example

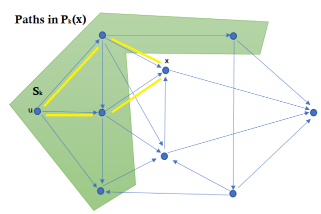


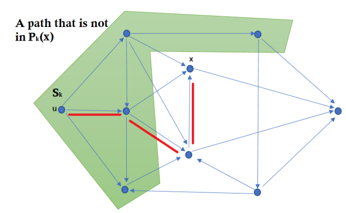
$$S_4 = \{u, d, c, a\} \quad \left|\begin{array}{ccccc} & & d_4 & & \\ u & a & b & c & d \\ 0 & 8 & 9 & 7 & 5 \end{array}\right| \quad \left|\begin{array}{ccccc} & & \pi_4 & & \\ u & a & b & c & d \\ \emptyset & d & a & d & u \end{array}\right|$$

$$S_5 = \{u, d, c, a, b\} \quad \left|\begin{array}{ccccc} & & d_5 & & \\ u & a & b & c & d \\ 0 & 8 & 9 & 7 & 5 \end{array}\right| \quad \left|\begin{array}{ccccc} & & \pi_5 & & \\ u & a & b & c & d \\ \emptyset & d & a & d & u \end{array}\right|$$

Dijkstra's algorithm - correctness

- For a vertex x let $P_k(x)$ denote the collection of all the directed paths from u to x such that all the vertices in the paths are in S_k except x.

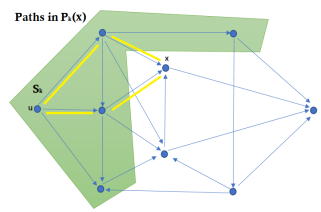


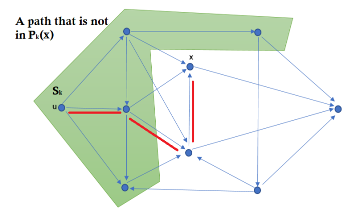


- For any path P we denote $w(P) = \sum_{e \in P} w(e)$.

Dijkstra's algorithm - correctness

- For a vertex x let $P_k(x)$ denote the collection of all the directed paths from u to x such that all the vertices in the paths are in S_k except x.





- For any path P we denote $w(P) = \sum_{e \in P} w(e)$.

Dijkstra's algorithm - correctness

- Theorem: For any $1 \leq k \leq n$ (where $n = |V|$) we have:
  1. For any $x \in S_k$: $d_k(x) = d(u, x)$
  2. For any $x \in V$: $d_k(x) = \min\{w(P) \mid P \in P_k(x)\}$
- Proof: We will prove the theorem by induction on k. The base case of $k = 1$ is clear.

Dijkstra's algorithm - correctness

- Theorem: For any $1 \leq k \leq n$ (where $n = |V|$) we have:
  1. For any $x \in S_k$: $d_k(x) = d(u, x)$
  2. For any $x \in V$: $d_k(x) = \min\{w(P) \mid P \in P_k(x)\}$
- Proof: We will prove the theorem by induction on k. The base case of $k = 1$ is clear.

Dijkstra's algorithm - correctness

- Theorem: For any $1 \leq k \leq n$ (where $n = |V|$) we have:
  1. For any $x \in S_k$: $d_k(x) = d(u, x)$
  2. For any $x \in V$: $d_k(x) = \min\{w(P) \mid P \in P_k(x)\}$
- Proof: We will prove the theorem by induction on k. The base case of $k = 1$ is clear.

Dijkstra's algorithm - correctness

- Theorem: For any $1 \leq k \leq n$ (where $n = |V|$) we have:
  1. For any $x \in S_k$: $d_k(x) = d(u, x)$
  2. For any $x \in V$: $d_k(x) = \min\{w(P) \mid P \in P_k(x)\}$
- Proof: We will prove the theorem by induction on k. The base case of $k = 1$ is clear.

Dijkstra's algorithm - correctness

- Proof: We start by proving (1). Let $x \in S_{k+1}$ be a vertex. If $x \in S_k$ then it follows from the induction hypothesis that:

$$d_{k+1} \underbrace{=}_{\text{definition of the algorithm}} d_k(x) \underbrace{=}_{\text{induction hypothesis}} d(u,x)$$

If $x = v_{k+1}$ then:

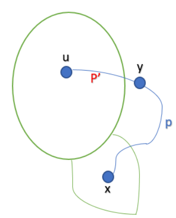
$$d_{k+1}(v_{k+1}) = d_k(v_{k+1})$$

$$\underbrace{=}_{\text{induction hypothesis}} \min\{w(P) \mid P \in P_k(x)\} \geq d(u, v_{k+1})$$

On the other hand...

Dijkstra's algorithm - correctness

- **Proof:** Let P be a path from u to v_{k+1} such that $w(P) = d(u, v_{k+1})$ and let y be the first vertex of P such that $y \notin S_k$ (y might be v_{k+1}).



Let P' be the path from u to y on P. We have:

$$d(u, v_{k+1}) = w(P) \underbrace{\geq}_{\text{positive weights}} w(P') \geq \min\{w(Q) \mid Q \in P_k(y)\} =$$

$$= d_k(y) \underbrace{\geq}_{\text{since } v_{k+1} \text{ is the vertex being added to } S_k} d_k(v_{k+1}) = d_{k+1}(v_{k+1})$$

It follows that $d(u, v_{k+1}) = d_k(v_{k+1})$.

(we got $d(u, v_{k+1}) \leq d_{k+1}(v_{k+1})$ and $d(u, v_{k+1}) \geq d_{k+1}(v_{k+1})$.)

Dijkstra's algorithm - correctness

- **Proof:** We now prove (2).
  Let $x \in V$. If $x \in S_{k+1}$ then we have:

$$d_{k+1}(x) \underbrace{=}_{\text{by (1)}} d(u,x) \leq \min\{w(P) \mid P \in P_{k+1}(x)\} \leq$$

$$\leq \min\{w(P) \mid P \in P_k(x)\} \underbrace{=}_{\text{this is the induction hypothesis}} d_k(x) \underbrace{=}_{\text{since } x \in S_{k+1}} d_{k+1}(x)$$

It follows that:

$$d_{k+1}(x) \underbrace{=} \min\{w(P) \mid P \in P_{k+1}(x)\}$$

We need to take care of $x \notin S_{k+1}$...

Dijkstra's algorithm - correctness

- **Proof:** Let $x \in V - S_{k+1}$. We have:

$$d_k(x) = \min\{w(P) \mid P \in P_k(x)\} \geq \min\{w(P) \mid P \in P_{k+1}(x)\}$$

and:

$$d_k(v_{k+1}) + w(v_{k+1}, x) = \min\{w(P) \mid P \in P_k(v_{k+1})\} + w(v_{k+1}, x) \underbrace{\geq}_{\text{The previous expression gives the weights of paths in } P_{k+1}(x)} \min\{w(P) \mid P \in P_{k+1}(x)\}$$

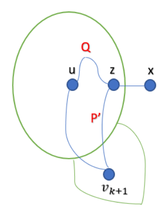
Since $d_{k+1}(x)$ is either $d_k(x)$ or $d_k(v_{k+1}) + w(v_{k+1}, x)$ we see that:

$$d_{k+1}(x) \geq \min\{w(P) \mid P \in P_{k+1}(x)\}$$

We need to show that $d_{k+1}(x) \leq \min\{w(P) \mid P \in P_{k+1}(x)\}$.

Dijkstra's algorithm - correctness

- Proof:



Let $P \in P_{k+1}(x)$ and let z be the last vertex before x in P.
If $z \in S_k$ we have:

$$d(u,z) = d_k(z) = \min\{w(Q) \mid Q \in P_k(z)\}$$

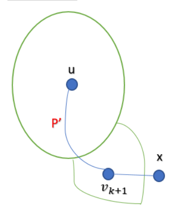
It means that there exists $Q \in P_k(z)$ such that $d(u,z) = w(Q)$. Let $P' = P - (z,x)$.
We get:

$$w(P) = w(P') + w(z,x) \geq w(Q) + w(z,x) \underbrace{\geq}_{\text{the left side is a path in } P_k(x)} d_k(x)$$

We need to consider $z \notin S_k$...

Dijkstra's algorithm - correctness

- Proof:



$z \notin S_k$ means that $z = v_{k+1}$ (it is the last vertex of $P \in P_{k+1}(x)$).
It follows that $P' = P - (v_{k+1}, x) \in P_k(v_{k+1})$. We get:

$$w(P') \geq \min\{w(Q) \mid Q \in P_k(v_{k+1})\} \underbrace{=}_{\text{induction hypothesis}} d_k(v_{k+1})$$

So:

$$w(P) = W(P') + w(v_{k+1}, x) \geq d_k(v_{k+1}) + w(v_{k+1}, x)$$

Dijkstra's algorithm - correctness

- Proof:
  For $z \in S_k$ we got:
  $$w(P) \geq d_k(x)$$
  For $z \notin S_k$ we got:
  $$w(P) \geq d_k(v_{k+1}) + w(v_{k+1}, x)$$
  It follows that in any case we have:
  $$w(P) \geq \min\{d_k(x), d_k(v_{k+1}) + w(v_{k+1}, x)\} \underbrace{=}_{\text{definition of the algorithm}} d_{k+1}(x)$$
  Since the above is true for any path $P \in P_{k+1}(x)$, we get:
  $$\min\{w(P) \mid P \in P_{k+1}(x)\} \geq d_{k+1}(x)$$
  ■

Dijkstra's algorithm

- Corollary: For any $x \in V$ we have:

$$d_n(x) = d(u, x)$$

(where $n = |V|$)
and the path:

$$\ldots \to \pi_n(\pi_n(x)) \to \pi_n(x) \to x$$

is a minimal path from u to x.

Dijkstra's algorithm - complexity

- If we implement the algorithm using an array of numbered vertices with the properties $v.d$ and $v.\pi$ then we can update the values of the function d and of π in $O(1)$ time.
- Finding the vertex with minimal value of d would require going over the array and thus would take $O(|V|)$ time.
- This leads to running time of $O(|V|^2 + |E|) = O(|V|^2)$. (We need to find the minimal element $|V|$ times. given the minimal elements the number of updates we will do depends on the degree of the vertex - this will add up to $|E|$.)

Dijkstra's algorithm - complexity

- If we implement the algorithm using an array of numbered vertices with the properties $v.d$ and $v.\pi$ then we can update the values of the function d and of π in $O(1)$ time.
- Finding the vertex with minimal value of d would require going over the array and thus would take $O(|V|)$ time.
- This leads to running time of $O(|V|^2 + |E|) = O(|V|^2)$. (We need to find the minimal element $|V|$ times. given the minimal elements the number of updates we will do depends on the degree of the vertex - this will add up to $|E|$.)

Dijkstra's algorithm - complexity

- If we implement the algorithm using an array of numbered vertices with the properties $v.d$ and $v.\pi$ then we can update the values of the function d and of π in $O(1)$ time.
- Finding the vertex with minimal value of d would require going over the array and thus would take $O(|V|)$ time.
- This leads to running time of $O(|V|^2 + |E|) = O(|V|^2)$. (We need to find the minimal element $|V|$ times. given the minimal elements the number of updates we will do depends on the degree of the vertex - this will add up to $|E|$.)

Dijkstra's algorithm - complexity

- We can use a priority queue (heap) to hold the vertices (with respect to the value $v.d$).
- If we do that we can remove the minimal element in $O(\log_2 |V|)$ time.
- Updating the value of d will now take $O(\log_2 |V|)$ time.
- This leads to running time of $O((|V| + |E|) \log_2 |V|) = O(|E| \log_2 |V|)$.
- This is better than the previous if $|E| = o\left(\frac{|V|^2}{\log_2 |V|}\right)$.

Dijkstra's algorithm - complexity

- We can use a priority queue (heap) to hold the vertices (with respect to the value $v.d$).
- If we do that we can remove the minimal element in $O(\log_2 |V|)$ time.
- Updating the value of d will now take $O(\log_2 |V|)$ time.
- This leads to running time of $O((|V| + |E|)\log_2 |V|) = O(|E| \log_2 |V|)$.
- This is better than the previous if $|E| = o\left(\frac{|V|^2}{\log_2 |V|}\right)$.

Dijkstra's algorithm - complexity

- We can use a priority queue (heap) to hold the vertices (with respect to the value $v.d$).
- If we do that we can remove the minimal element in $O(\log_2 |V|)$ time.
- Updating the value of d will now take $O(\log_2 |V|)$ time.
- This leads to running time of $O((|V| + |E|) \log_2 |V|) = O(|E| \log_2 |V|)$.
- This is better than the previous if $|E| = o\left(\frac{|V|^2}{\log_2 |V|}\right)$.

Dijkstra's algorithm - complexity

- We can use a priority queue (heap) to hold the vertices (with respect to the value $v.d$).
- If we do that we can remove the minimal element in $O(\log_2 |V|)$ time.
- Updating the value of d will now take $O(\log_2 |V|)$ time.
- This leads to running time of $O((|V| + |E|) \log_2 |V|) = O(|E| \log_2 |V|)$.
- This is better than the previous if $|E| = o\left(\frac{|V|^2}{\log_2 |V|}\right)$.

Dijkstra's algorithm - complexity

- We can use a priority queue (heap) to hold the vertices (with respect to the value $v.d$).
- If we do that we can remove the minimal element in $O(\log_2 |V|)$ time.
- Updating the value of d will now take $O(\log_2 |V|)$ time.
- This leads to running time of $O((|V| + |E|) \log_2 |V|) = O(|E| \log_2 |V|)$.
- This is better than the previous if $|E| = o\left(\frac{|V|^2}{\log_2 |V|}\right)$.