Examples and applications on SSSP and MST

Dan (Doris) He & Junhao Gan

ITEE
University of Queensland

Dijkstra's Algorithm

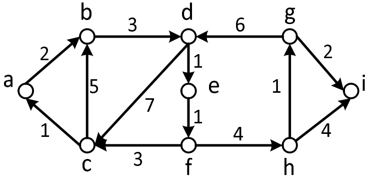
The algorithm solves the single-source shortest-paths (SSSP) problem on a directed graph $G = (V, E)$ with positive edge weights.

Let $V' \subseteq V$ be the current set of vertices whose shortest paths from the source vertex s have been found and $S = V \backslash V'$.

The crucial part of the algorithm is the edge relaxation idea. Essentially, it is to maintain, for each $v \in S$, the "current shortest" distance from s only through the vertices in V'.

Example

Suppose that the source vertex is a.

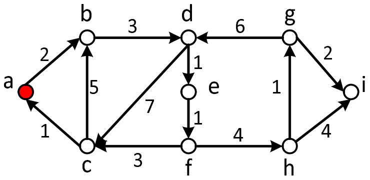


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	∞	nil
c	∞	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$V' = \emptyset$ and
$S = \{a, b, c, d, e, f, g, h, i\}$.
Since $dist(a)$ is the smallest among those of vertices in S, pick a.

Example

Relax the out-going edges of a:



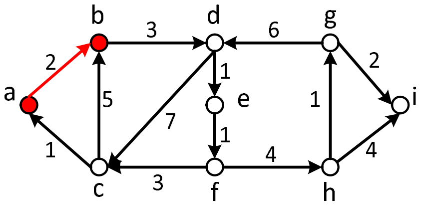
vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	$\infty \to 2$	nil $\to$ a
c	∞	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$V' = \{a\}$ and
$S = \{b, c, d, e, f, g, h, i\}$.

The "current shortest" distance of b from a only through the vertices in V' is updated. After then, $dist(b)$ is the smallest among those of vertices in S. Pick b.

Example

Relax the out-going edges of b:

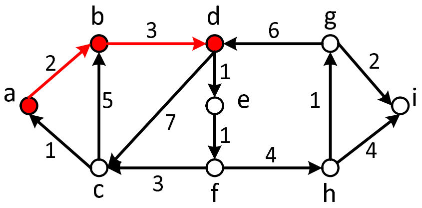


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	∞	nil
d	$\infty \to 5$	nil $\to b$
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$V' = \{a, b\}$ and
$S = \{c, d, e, f, g, h, i\}$.
Similarly, update the "current shortest" distance of d from a only through the vertices in V'. And $dist(d)$ is the smallest among those in S. Pick d.

Example

Relax the out-going edges of d:

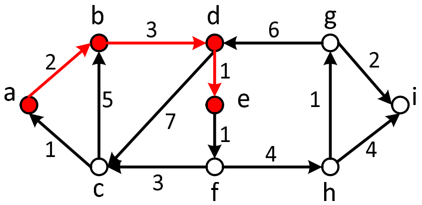


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	$\infty \to 12$	nil $\to d$
d	5	b
e	$\infty \to 6$	nil $\to d$
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$V' = \{a, b, d\}$ and
$S = \{c, e, f, g, h, i\}$.
Since after the updates, $dist(e)$ is the smallest among those in S, pick e.

Example

Relax the out-going edges of e:

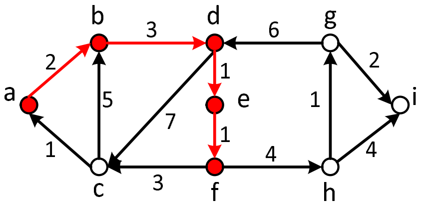


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	12	d
d	5	b
e	6	d
f	$\infty \to 7$	nil $\to e$
g	∞	nil
h	∞	nil
i	∞	nil

$V' = \{a, b, d, e\}$ and
$S = \{c, f, g, h, i\}$.

Example

Relax the out-going edges of f:

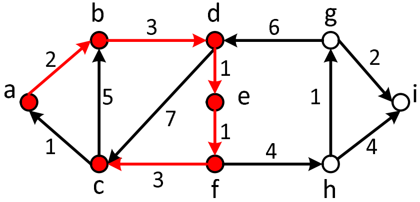


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	$12 \rightarrow 10$	$d \rightarrow f$
d	5	b
e	6	d
f	7	e
g	∞	nil
h	$\infty \rightarrow 11$	nil $\rightarrow f$
i	∞	nil

$V' = \{a, b, d, e, f\}$ and
$S = \{c, g, h, i\}$.

Example

Relax the out-going edges of c:

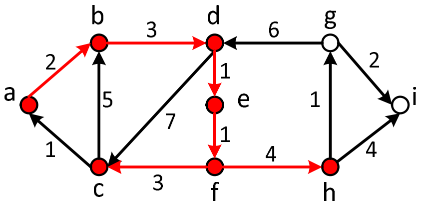


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	∞	nil
h	11	f
i	∞	nil

$V' = \{a, b, c, d, e, f\}$ and
$S = \{g, h, i\}$.

Example

Relax the out-going edges of h:

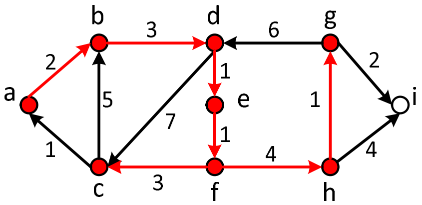


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	$\infty \to 12$	nil $\to h$
h	11	f
i	$\infty \to 15$	nil $\to h$

$V' = \{a, b, c, d, e, f, h\}$ and
$S = \{g, i\}$.

Example

Relax the out-going edges of g:

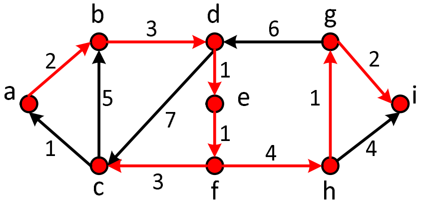


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	12	h
h	11	f
i	$15 \to 14$	$h \to g$

$V' = \{a, b, c, d, e, f, g, h\}$ and
$S = \{i\}$.

Example

Relax the out-going edges of i:



vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	12	h
h	11	f
i	14	g

$V' = \{a, b, c, d, e, f, g, h, i\}$ and $S = \{\}$.
Done.

Prim's Algorithm

The algorithm grows a tree T_{mst} by including one vertex at a time. At any moment, it divides the vertex set V into two parts:

- The set S of vertices that are already in T_{mst}.
- The set of other vertices: $V \setminus S$

At the end of the algorithm, $S = V$.

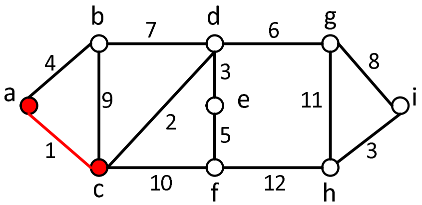
If an edge connects a vertex in V and a vertex in $V \setminus S$, we call it an extension edge.

At all times, the algorithm enforces the following lightest extension principle:

- For every vertex $v \in V \setminus S$, it remembers which extension edge of v has the smallest weight — referred to as the lightest extension edge of v, and denoted as $best\text{-}ext(v)$.

Example

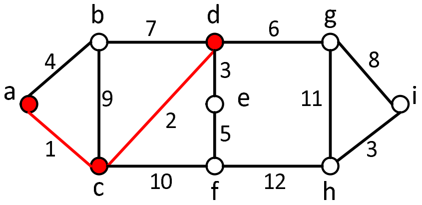
Edge $\{a, c\}$ is the lightest of all. $S = \{a, c\}$. The MST has one edge $\{a, c\}$.



vertex v	*best-ext*(v) and weight
a	n/a
b	$\{b, a\}, 4$
c	n/a
d	$\{d, c\}, 2$
e	nil, ∞
f	$\{f, c\}, 10$
g	nil, ∞
h	nil, ∞
i	nil, ∞

Example

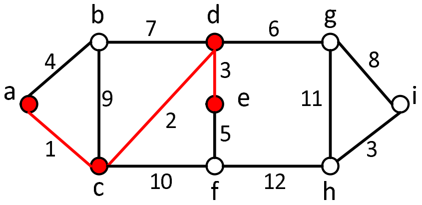
Edge $\{d, c\}$ is the lightest of all. $S = \{a, c, d\}$. Add edge $\{d, c\}$ into the MST.



vertex v	$best\text{-}ext(v)$ and weight
a	n/a
b	$\{b, a\}, 4$
c	n/a
d	n/a
e	$\{e, d\}, 3$
f	$\{f, c\}, 10$
g	$\{g, d\}, 6$
h	nil, ∞
i	nil, ∞

Example

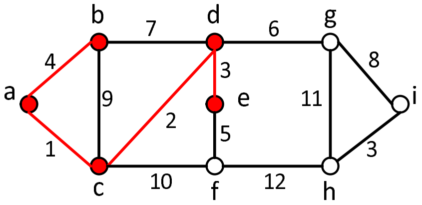
Edge $\{e, d\}$ is the lightest of all. $S = \{a, c, d, e\}$. Add edge $\{e, d\}$ into the MST.



vertex v	*best-ext*(v) and weight
a	n/a
b	$\{b, a\}, 4$
c	n/a
d	n/a
e	n/a
f	$\{f, e\}, 5$
g	$\{g, d\}, 6$
h	nil, ∞
i	nil, ∞

Example

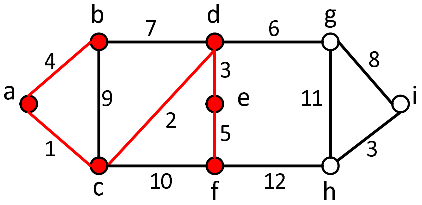
Edge $\{b, a\}$ is the lightest of all. $S = \{a, c, d, e, b\}$. Add edge $\{b, a\}$ into the MST.



vertex v	*best-ext*(v) and weight
a	n/a
b	n/a
c	n/a
d	n/a
e	n/a
f	$\{f, e\}, 5$
g	$\{g, d\}, 6$
h	nil, ∞
i	nil, ∞

Example

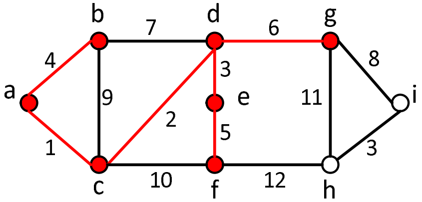
Edge $\{f, e\}$ is the lightest of all. $S = \{a, c, d, e, b, f\}$. Add edge $\{f, e\}$ into the MST.



vertex v	*best-ext*(v) and weight
a	n/a
b	n/a
c	n/a
d	n/a
e	n/a
f	n/a
g	$\{g, d\}, 6$
h	$\{h, f\}, 12$
i	nil, ∞

Example

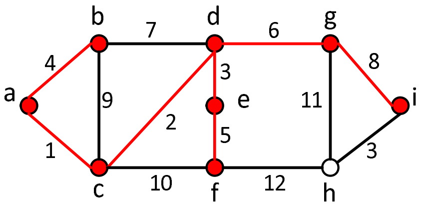
Edge $\{g, d\}$ is the lightest of all. $S = \{a, c, d, e, b, f, g\}$. Add edge $\{g, d\}$ into the MST.



vertex v	*best-ext*(v) and weight
a	n/a
b	n/a
c	n/a
d	n/a
e	n/a
f	n/a
g	n/a
h	$\{h, f\}, 12$
i	$\{i, g\}, 8$

Example

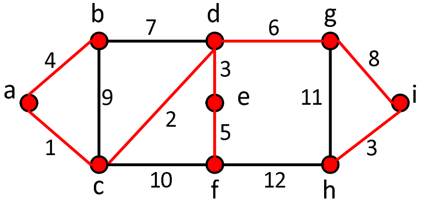
Edge $\{i, g\}$ is the lightest of all. $S = \{a, c, d, e, b, f, g, i\}$. Add edge $\{i, g\}$ into the MST.



vertex v	*best-ext*(v) and weight
a	n/a
b	n/a
c	n/a
d	n/a
e	n/a
f	n/a
g	n/a
h	$\{h, i\}, 3$
i	n/a

Example

At the end, edge $\{h, i\}$ is the lightest of all. $S = \{a, c, d, e, b, f, g, i, h\}$. Add edge $\{h, i\}$ into the MST and we get the final MST.



vertex v	$best\text{-}ext(v)$ and weight
a	n/a
b	n/a
c	n/a
d	n/a
e	n/a
f	n/a
g	n/a
h	n/a
i	n/a

Facility Allocation — Application of Dijkstra's Algorithm

Let $G = (V, E)$ be an undirected graph with positive edge weights, where each vertex $v \in V$ represents either a user or a facility.

For a user u and a facility f, the distance between u and f is the shortest distance from u to f on G denoted by $dist(u, f)$.

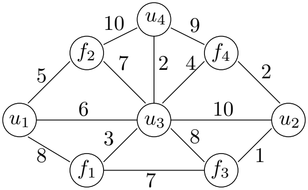
Our goal is to assign the nearest facility f to each user u such that:

- There is no facility $f' \neq f$ such that $dist(f', u) < dist(f, u)$.

Furthermore, we say $f = \text{nearest-fac}(u)$.

Example

Given a graph below, f_i (for $i \in [1, 4]$) represents a facility and u_j (for $j \in [1, 4]$) represents a user.



Then we have:

- nearest-fac$(u_1) = f_2$,
- nearest-fac$(u_2) = f_3$,
- nearest-fac$(u_3) = f_1$,
- nearest-fac$(u_4) = f_1$.

Facility Allocation — Application of Dijkstra's Algorithm

To solve the problem, the most trivial way is to run one Dijkstra's algorithm for each facility. And then, for each user u, pick the facility f with $dist(f, u)$ smallest.

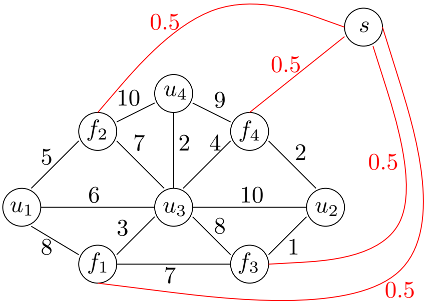
However, we can do much better. In fact, only one Dijkstra is enough!

Facility Allocation — Application of Dijkstra's Algorithm

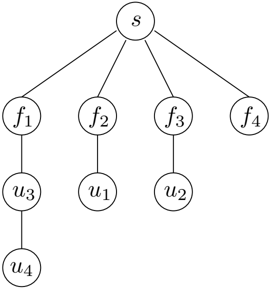
The algorithm is as follows:

- Add an auxilary vertex s to G. Create an edge $\{s, f\}$ for each facility f with edge weight α, where $\alpha > 0$ is smaller than any edge weight in G.
- Perform Dijkstra's algorithm with source vertex s on the augmented graph.
- In the resulted SSSP tree, for each facility f, assign f to each user in the subtree of f.

Example



s
0.5
0.5
0.5
0.5
u4
10
9
f2
f4
5
7
2
4
2
u1
6
u3
10
u2
3
8
8
f1
7
f3
1



s
f1
f2
f3
f4
u3
u1
u2
u4

Correctness

The correctness of the algorithm follows the facts below:

- All the nodes at level-1 (assume s is at level-0) in the SSSP tree are the facility vertices.
- All the user nodes u in the subtree of a facility f satisfy: $dist(f, u) \leq dist(f', u)$ for all facilities $f' \neq f$.

A short proof for the second bullet:

Suppose that there does exist a facility f' such that $dist(f', u) < dist(f, u)$. Then the path from s to u through f' is shorter than that through f. It contradicts with the fact that u is in the subtree of f in the SSSP tree.

Traveling Salesman — Application of MST

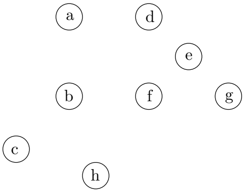
Given a weighted undirected graph $G = (V, E)$ satisfying the followings:

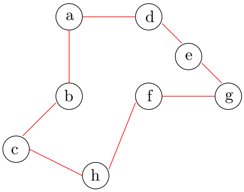
- G is complete: for $\forall u, v \in V$, $\{u, v\}$ is in E.
- The weight of each edge is positive.
- All the weights satisfy the triangle inequality: $w(u, v) \leq w(u, x) + w(x, v)$ for $\forall u, v, x \in V$, where $w(u, v)$ is the weight of edge $\{u, v\}$.

The Traveling Salesman Problem (TSP) is to find a route H^* such that:

- H^* is a cycle.
- Each vertex $v \in V$ is visited by H^* once and exactly once.
- The total weight of the edges on H^* is smallest.

Example





For simple illustration, we show the input graph on the left without drawing all the edges explicitly. Instead, we embed the graph on the plan such that the weight of each edge is the euclidean distance between its two end points.

The red cycle shown on the right is the optimal solution H^*.

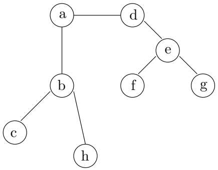
The TSP problem is NP-hard!

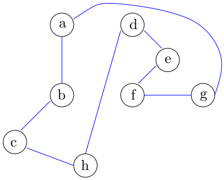
Traveling Salesman — Application of MST

For a cycle H in the input graph, denote the total weight of edges on H by $w(H)$.

In the following, we will introduce a 2-approximate algorithm for the TSP problem in the sense that it always finds a solution H with $w(H) \leq 2 \times w(H^*)$, where H^* is the optimal solution.

- Compute an MST T of G.
- Perform DFS on T from an arbitrary node and record the node visiting sequence S:
  $a, b, c, b, h, b, a, d, e, f, e, g, e, d, a.$
- Only keep the first occurrence of each vertex in the sequence S:
  $a, b, c, h, d, e, f, g.$
- Let H be the route of this first occurrence sequence. Return H as an approximate solution.





We claim that $w(H) \leq 2 \times w(H^*)$.

Proof sketch:

- Since H^* is a cycle on all the vertices, removing any edge from H^* will result to a spanning tree T'. As all the edge weights are positive and T is an MST, we have $w(T) \leq w(T') < w(H^*)$.

- Since the DFS traverses each edge of T only twice, the total weight of all the edges on the visit sequence S, denoted by $w(S)$, is $2 \times w(T)$.

- Consider any two consecutive vertices u and v on H. By the triangle inequality, we know that $w(u, v)$ is no more than the total weight of all the edges on the sub-sequence of S between their first occurrence. Therefore, $w(H) \leq w(S) = 2 \times w(T) < 2 \times w(H^*)$.