Shortest Path Problems
Discrete Mathematics

Weighted Graphs

Graphs that have a number assigned to each edge are called **weighted graphs**.

A more mathematical definition: A **weighted graph** is a graph $G = (V, E, w)$, where V is a set of vertices, E is a set of edges between vertices of V, and w is a function from $V \times V$ into $\mathbb{R}$. The function w gives to each pair of vertices a weight in $\mathbb{R}$.

Note 1: Usually $w(u, v)$ are positives for all pairs of vertices.
Note 2: Usually $w(u, v) = \infty$ if the pair of vertices (u, v) is not an edge of E.

Length of a Path

Let the path p, form vertices a to z, be given by the sequence of edges $e_1, e_2, ..., e_n$ of the graph, such that $f(e_1) = (a, x_1)$, $f(e_2) = (x_1, x_2)$, ..., $f(e_n) = (x_{n-1}, z)$.

The **length $L(p)$ of a path** p, in a weighted graph, is the sum of the weights of the edges of this path, i.e.
$L(p) = w(a, x_1) + w(x_1, x_2) + \cdots + w(x_{n-1}, z)$.

A common problem is to find the path, with minimal length, between two given vertices of a weighted graph.

Edsger Wybe Dijkstra

Rotterdam, 11 May 1930 – 6 August 2002
was a Dutch computer scientist. He received the 1972 Turing Award for fundamental contributions to developing programming languages. Among his contributions to computer science is the shortest path-algorithm, also known as Dijkstra's algorithm and Reverse Polish Notation.

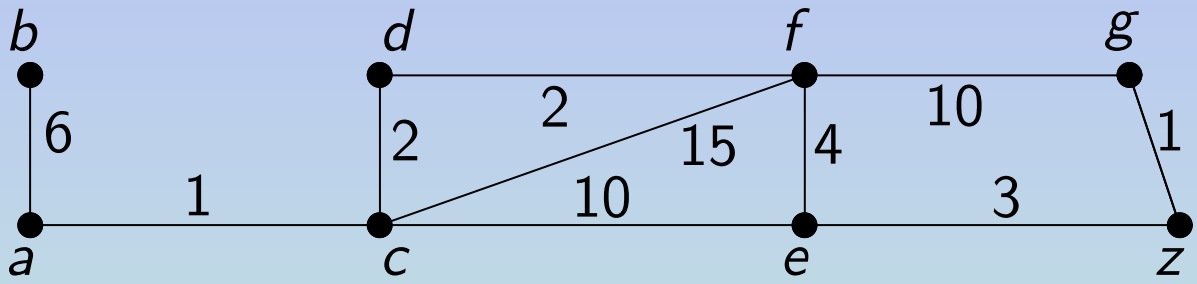
`www.cs.utexas.edu/users/EWD`

Dijkstra's Algorithm

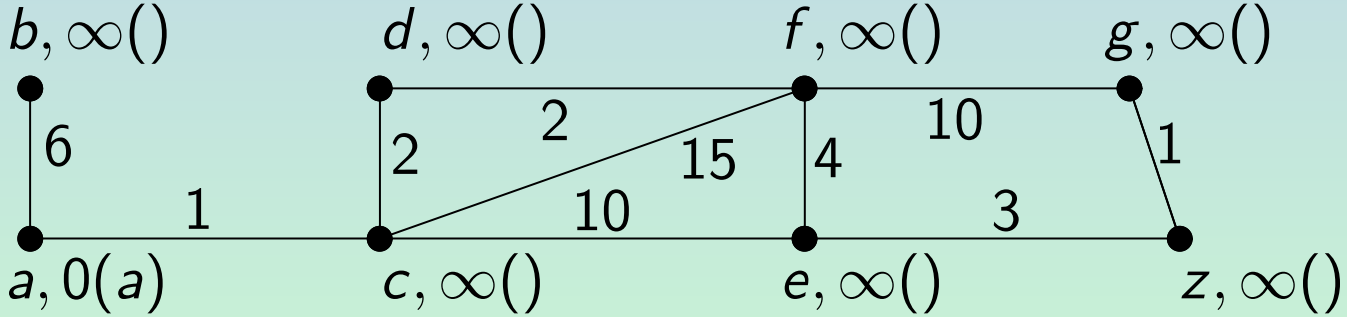
procedure *Dijkstra* (G: a simple connected weighted graph with
with positive weights.
a: initial vertex. z: final vertex)
$L(v) := \infty$ for all vertices v of G.
$L(a) := 0$
$S := \emptyset$
while $z \notin S$
begin
$u :=$ a vertex not in S with smallest $L(u)$.
$S := S \cup \{u\}$
for all vertex v not in S
if $L(u) + w(u, v) < L(v)$
then $L(v) := L(u) + w(u, v)$
end
$\{L(z) =$ length of the shortest path from a to $z\}$

Example of Dijkstra's Algorithm, Step 1 of 8

Consider the following simple connected weighted graph. What is the shortest path between vertices a and z.

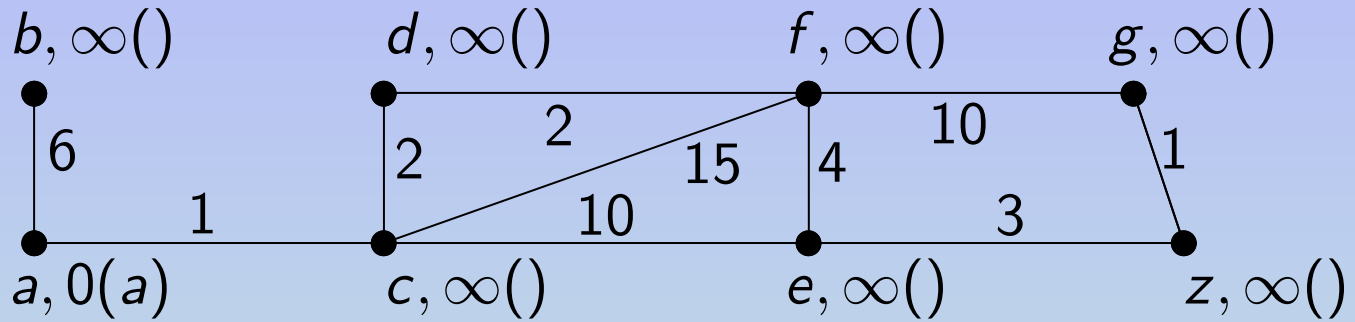


$L(v)$ is set ∞ for all vertices v of G, $L(a)$ is set to 0 and S to $\emptyset$.

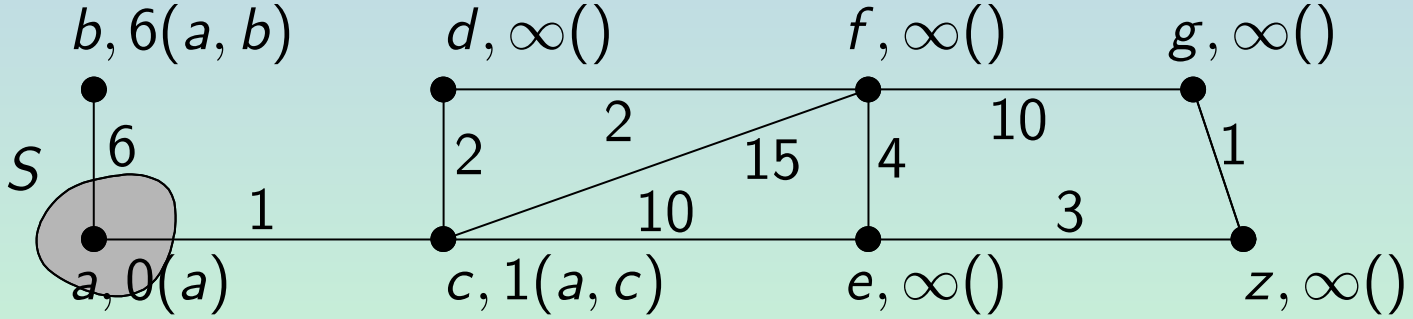


Example of Dijkstra's Algorithm, Step 2 of 8

a is the vertex u not in S such that $L(u)$ is minimal.



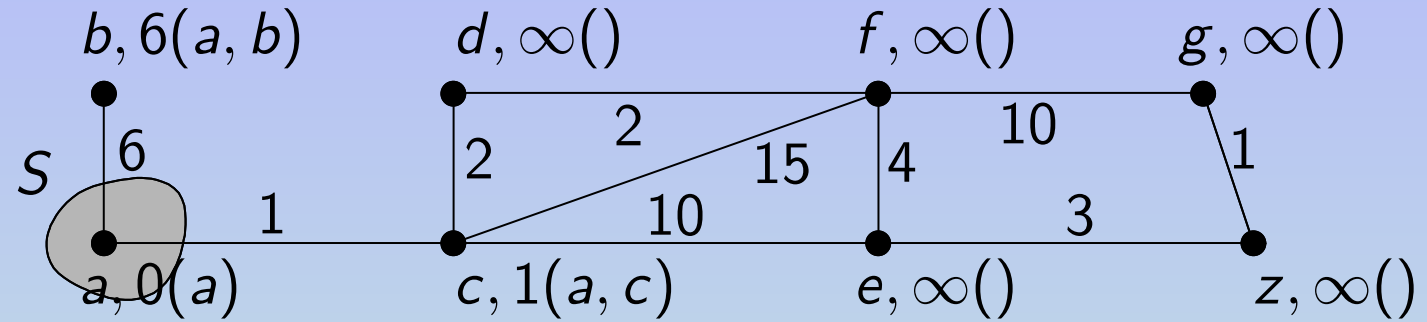
$S = S \cup \{a\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to a).



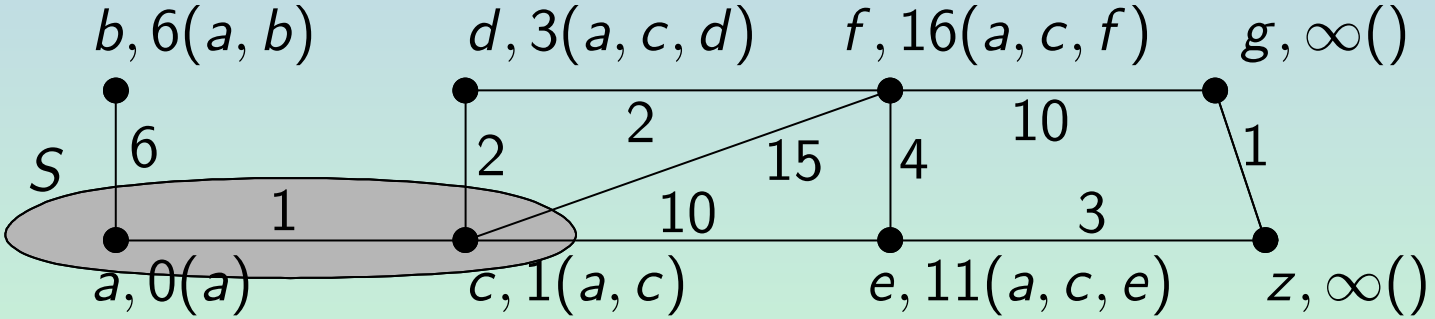
Note: To save space, $L(a, c) = 1$, the length of the path through vertices a and c, is denoted $1(a, c)$.

Example of Dijkstra's Algorithm, Step 3 of 8

c is the vertex u not in S such that $L(u)$ is minimal.



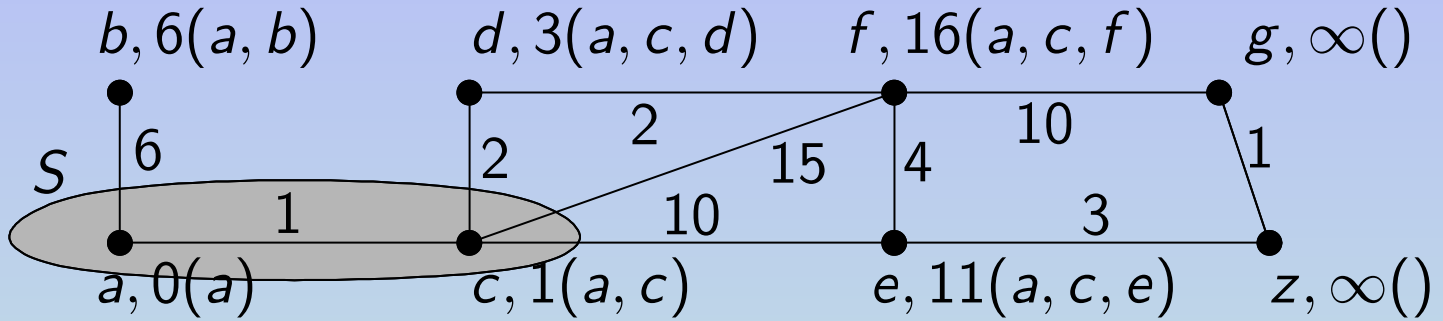
$S = S \cup \{c\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to c).



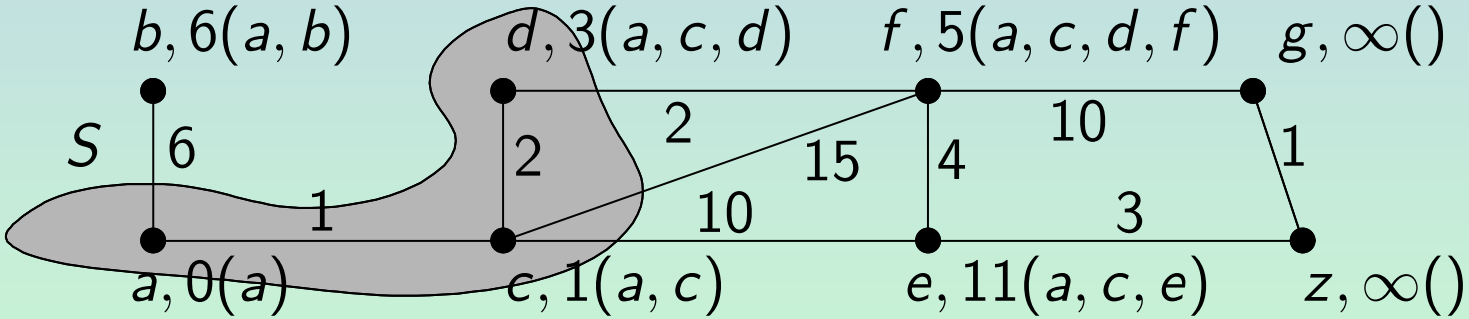
Note: To save space, $L(a, c, d) = 3$, the length of the path through vertices a, c and d, is denoted $3(a, c, d)$.

Example of Dijkstra's Algorithm, Step 4 of 8

d is the vertex u not in S such $L(u)$ is minimal.

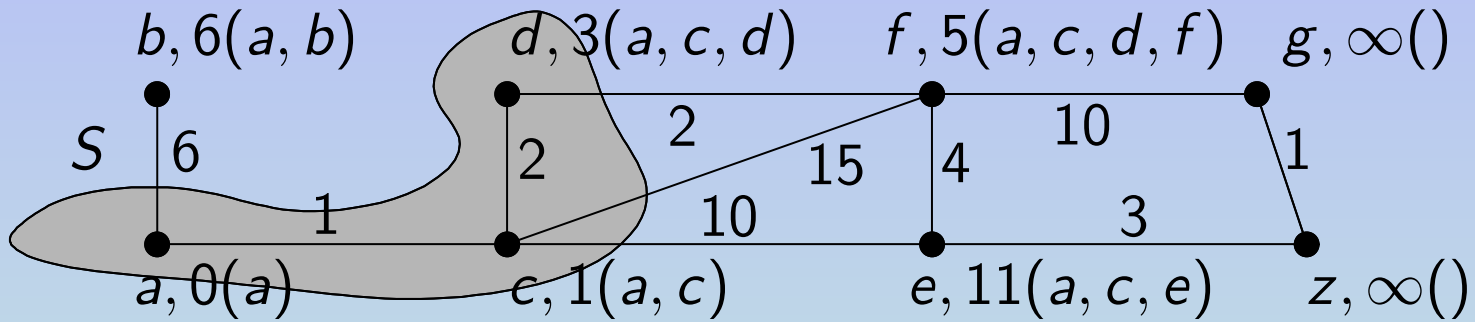


$S = S \cup \{d\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to d).

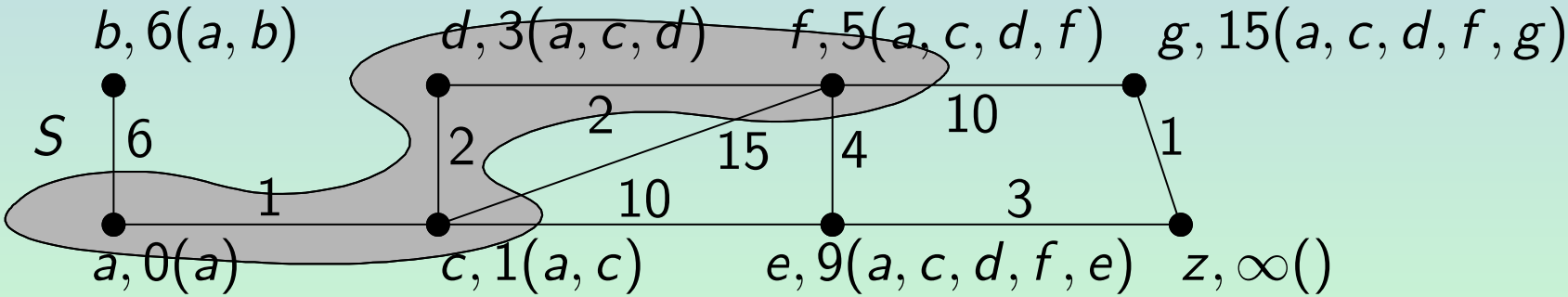


Example of Dijkstra's Algorithm, Step 5 of 8

f is the vertex u not in S such $L(u)$ is minimal.

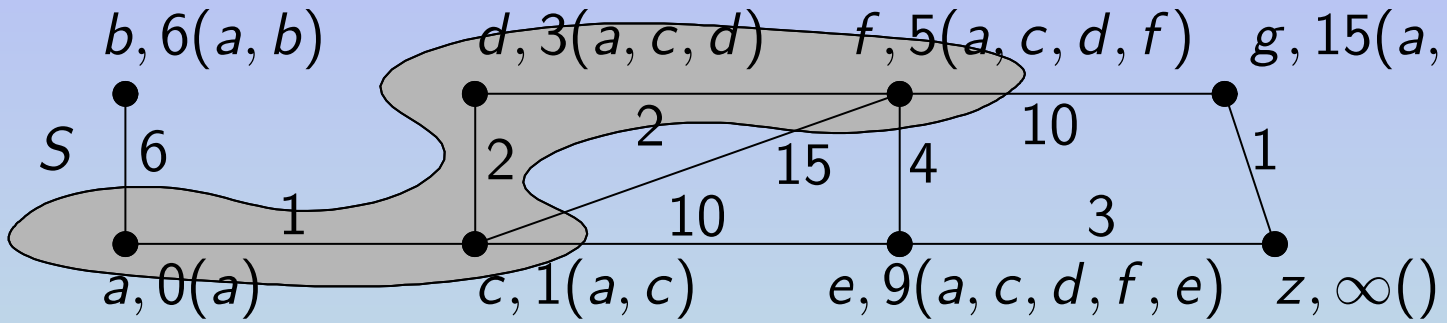


$S = S \cup \{f\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to f).

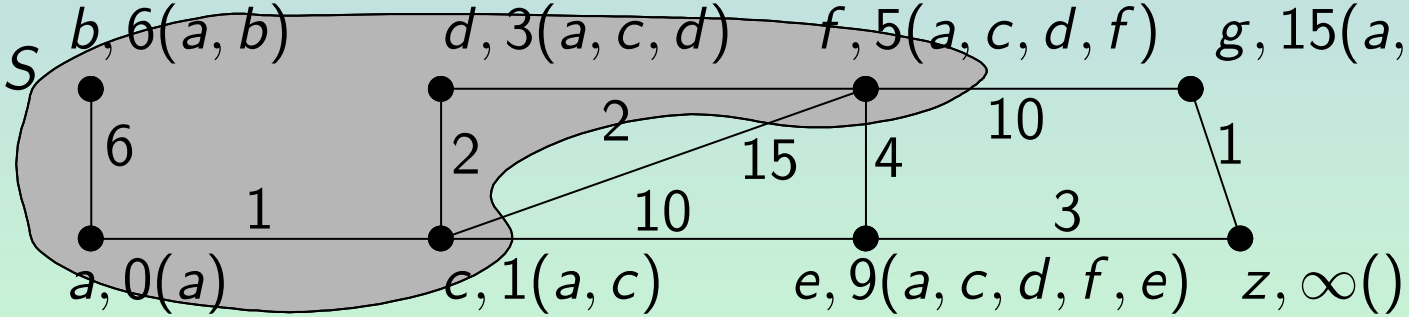


Example of Dijkstra's Algorithm, Step 6 of 8

b is the vertex u not in S such that $L(u)$ is minimal.

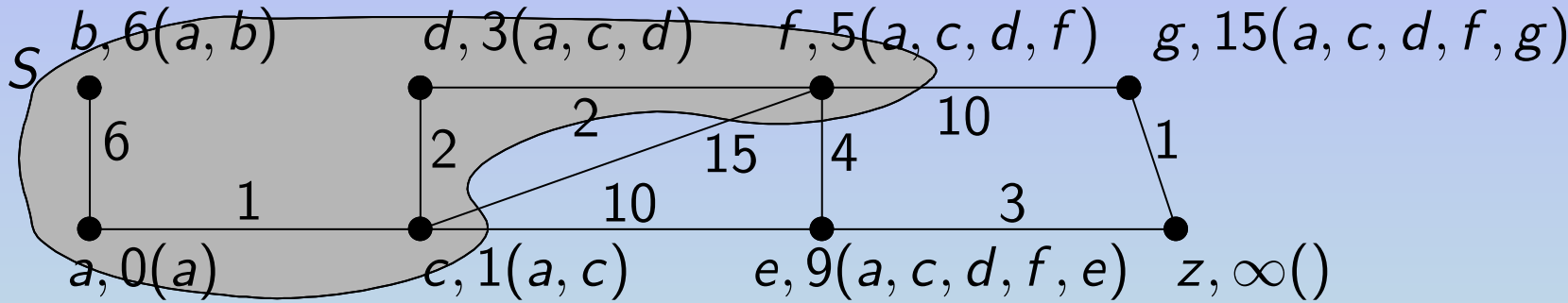


$S = S \cup \{b\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to b) (in this case, there is not vertex to update).

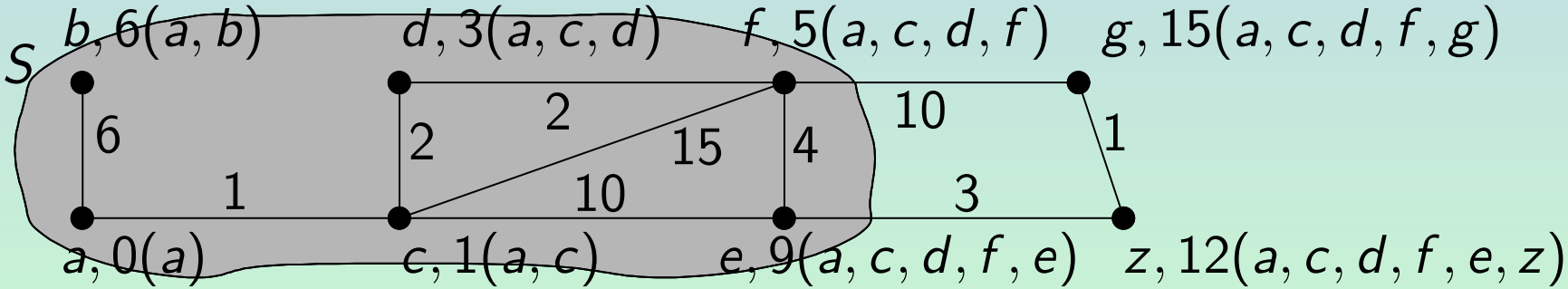


Example of Dijkstra's Algorithm, Step 7 of 8

e is the vertex u not in S such that $L(u)$ is minimal.

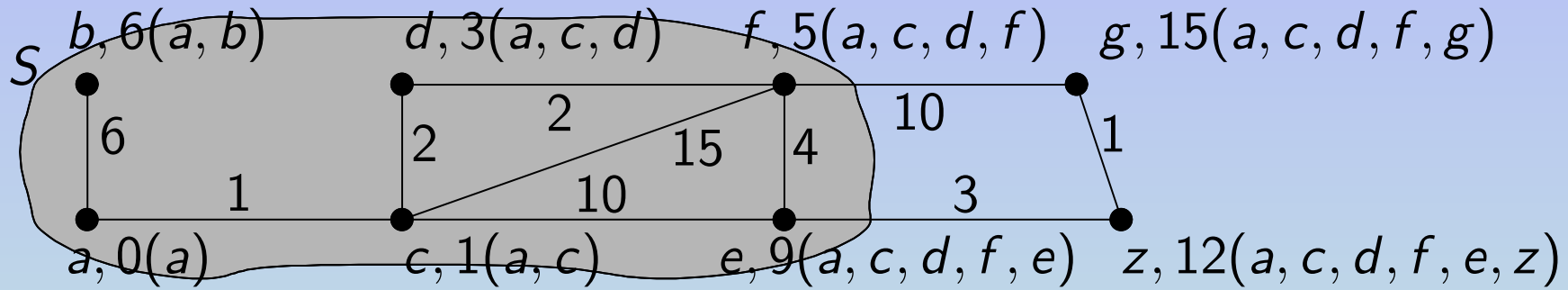


$S = S \cup \{e\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to e).

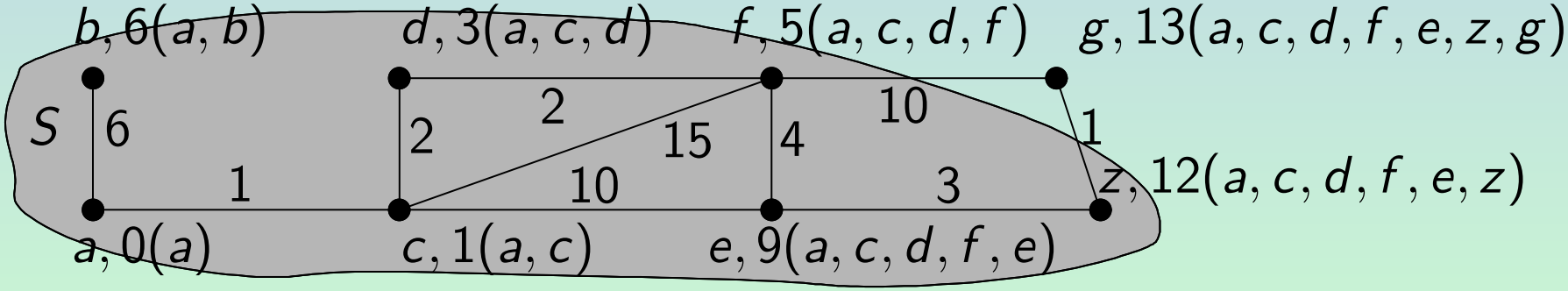


Example of Dijkstra's Algorithm, Step 8 of 8

z is the vertex u not in S such that $L(u)$ is minimal.



$S = S \cup \{z\}$ and $L(v)$ is updated for all vertex v not in S (and adjacent to z). The algorithm stops here because $z \in S$.



Properties of Dijkstra's Algorithm

Algorithm Finiteness

THEOREM: Dijkstra's algorithm always ends in n steps or less for a connected undirected weighted graph with$n$ vertices.

Algorithm Correctness

THEOREM: Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected undirected weighted graph.

Algorithm Effectiveness

THEOREM: Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected undirected weighted graph with n vertices.

Floyd's Algorithm

Floyd's algorithm can be used to find the length of the shortest path between all pairs of vertices in a weighted connected simple graph. However, this algorithm cannot be used to construct shortest paths.

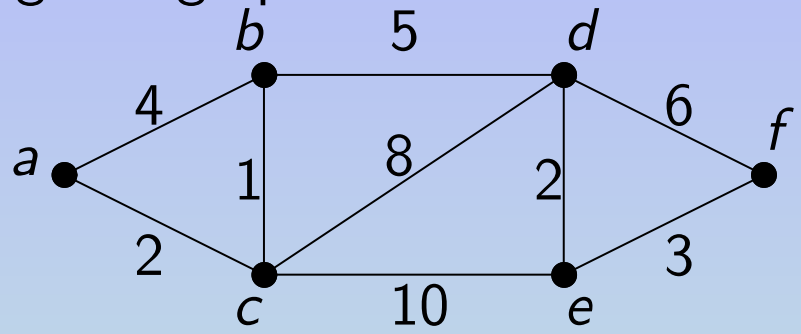
Robert W. Floyd. "*Algorithm 97: Shortest path*", Comm ACM, vol 5 (June 1962), p. 345-.

Floyd's Algorithm

procedure *Floyd* (G: weighted simple graph
with vertices $v_1, v_2, ..., v_n$
and weights $w(v_i, v_j)$.)
for $i := 1$ **to** n
  for $j := 1$ **to** n
    $d(v_i, v_j) = w(v_i, v_j)$
    $d(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge
for $i := 1$ **to** n
  for $j := 1$ **to** n
    for $k := 1$ **to** n
      if $d(v_j, v_i) + d(v_i, v_k) < d(v_j, v_k)$
      then $d(v_j, v_k) := d(v_j, v_i) + d(v_i, v_k)$
$\{d(v_i, v_j)$ is the length of the shortest path between v_i and $v_j\}$

Floyd's Algorithm

Use Floyd's algorithm to find the distance between all pairs of vertices in the weighted graph:



We can represent the distances with a 6×6 matrix, with alphabetic order. Initially it is

$$\begin{bmatrix} \infty & 4 & 2 & \infty & \infty & \infty \\ 4 & \infty & 1 & 5 & \infty & \infty \\ 2 & 1 & \infty & 8 & 10 & \infty \\ \infty & 5 & 8 & \infty & 2 & 6 \\ \infty & \infty & 10 & 2 & \infty & 3 \\ \infty & \infty & \infty & 6 & 3 & \infty \end{bmatrix}$$

Floyd's Algorithm

After completion of the main loop for $i = 1$ (vertex a), the matrix is (boxed values are those that were updated)

$$\begin{bmatrix} \infty & 4 & 2 & \infty & \infty & \infty \\ 4 & \boxed{8} & 1 & 5 & \infty & \infty \\ 2 & 1 & \boxed{4} & 8 & 10 & \infty \\ \infty & 5 & 8 & \infty & 2 & 6 \\ \infty & \infty & 10 & 2 & \infty & 3 \\ \infty & \infty & \infty & 6 & 3 & \infty \end{bmatrix}.$$

After completion of the main loop for $i = 2$ (vertex b), the matrix is

$$\begin{bmatrix} \boxed{8} & 4 & 2 & \boxed{9} & \infty & \infty \\ 4 & 8 & 1 & 5 & \infty & \infty \\ 2 & 1 & \boxed{2} & \boxed{6} & 10 & \infty \\ \boxed{9} & 5 & \boxed{6} & \boxed{10} & 2 & 6 \\ \infty & \infty & 10 & 2 & \infty & 3 \\ \infty & \infty & \infty & 6 & 3 & \infty \end{bmatrix}.$$

Floyd's Algorithm

After completion of the main loop for $i = 3$ (vertex c), the matrix is

$$\begin{bmatrix} \boxed{4} & \boxed{3} & 2 & \boxed{8} & \boxed{12} & \infty \\ \boxed{3} & \boxed{2} & 1 & 5 & \boxed{11} & \infty \\ 2 & 1 & 2 & 6 & 10 & \infty \\ \boxed{8} & 5 & 6 & 10 & 2 & 6 \\ \boxed{12} & \boxed{11} & 10 & 2 & \boxed{20} & 3 \\ \infty & \infty & \infty & 6 & 3 & \infty \end{bmatrix}.$$

After completion of the main loop for $i = 4$ (vertex d), the matrix is

$$\begin{bmatrix} 4 & 3 & 2 & 8 & \boxed{10} & \boxed{14} \\ 3 & 2 & 1 & 5 & \boxed{7} & \boxed{11} \\ 2 & 1 & 2 & 6 & \boxed{8} & \boxed{12} \\ 8 & 5 & 6 & 10 & 2 & 6 \\ \boxed{10} & \boxed{7} & \boxed{8} & 2 & \boxed{4} & 3 \\ \boxed{14} & \boxed{11} & \boxed{12} & 6 & 3 & \boxed{12} \end{bmatrix}.$$

Floyd's Algorithm

After completion of the main loop for $i = 5$ (vertex e), the matrix is (boxed values are those that were updated)

$$\begin{bmatrix} 4 & 3 & 2 & 8 & 10 & \boxed{13} \\ 3 & 2 & 1 & 5 & 7 & \boxed{10} \\ 2 & 1 & 2 & 6 & 8 & \boxed{11} \\ 8 & 5 & 6 & \boxed{4} & 2 & \boxed{5} \\ 10 & 7 & 8 & 2 & 4 & 3 \\ \boxed{13} & 10 & \boxed{11} & \boxed{5} & 3 & \boxed{6} \end{bmatrix}.$$

In this problem, there is no change after the final iteration with $i = 6$ (vertex f). Therefore this matrix represents the distances between all pairs of vertices.