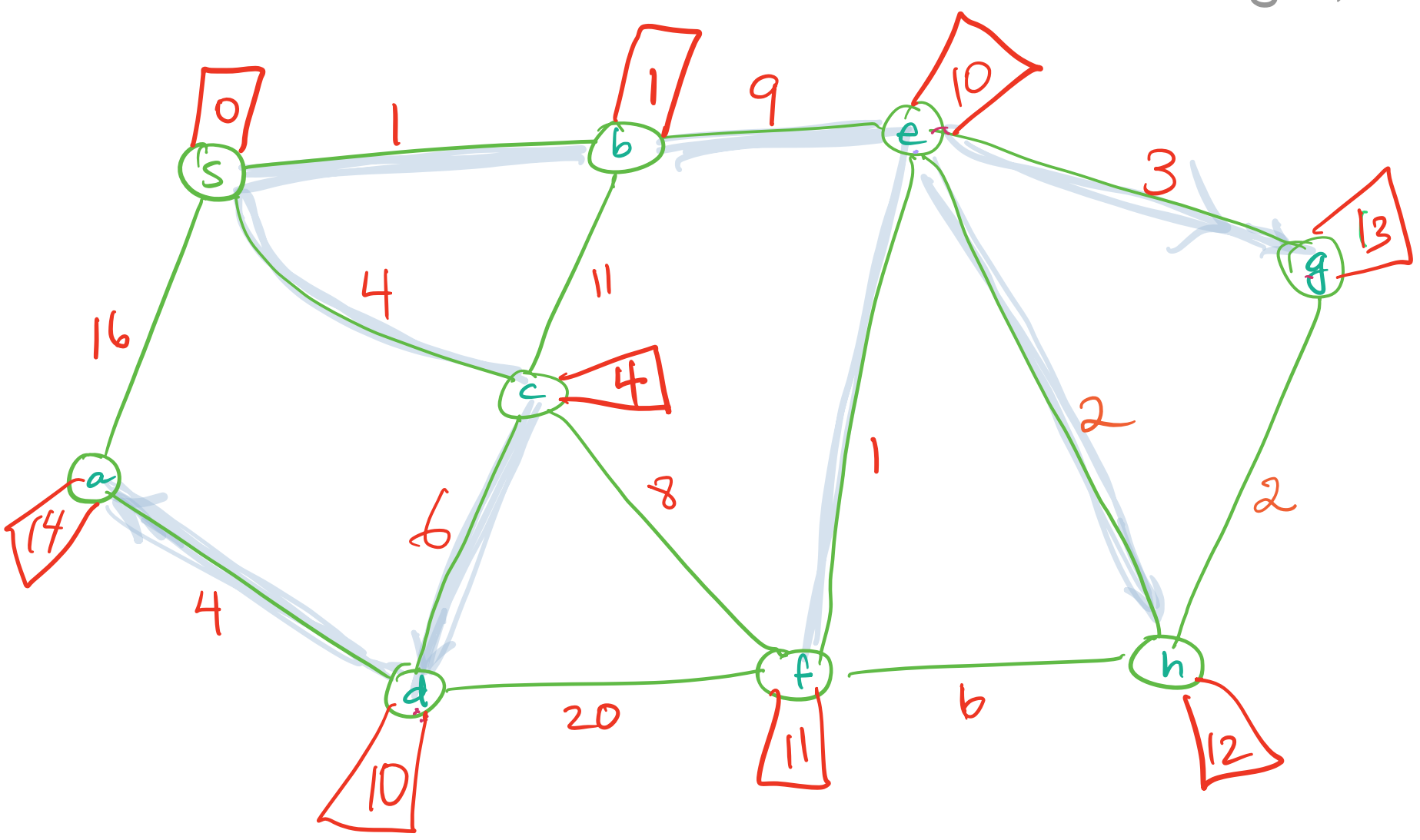
Single-Source Shortest Paths

Input: Graph $G = (V, E)$, weight function $w: E \to \mathbb{R}^+$, source s, $s \in V$

Goal: determine $\forall v \in V$ the shortest s-v path (ie path with min sum of edge weights)



relax(g, d, w)

if $d[g] > d[d] + w(d, g)$

$d[g] = d[d] + w(d, g)$

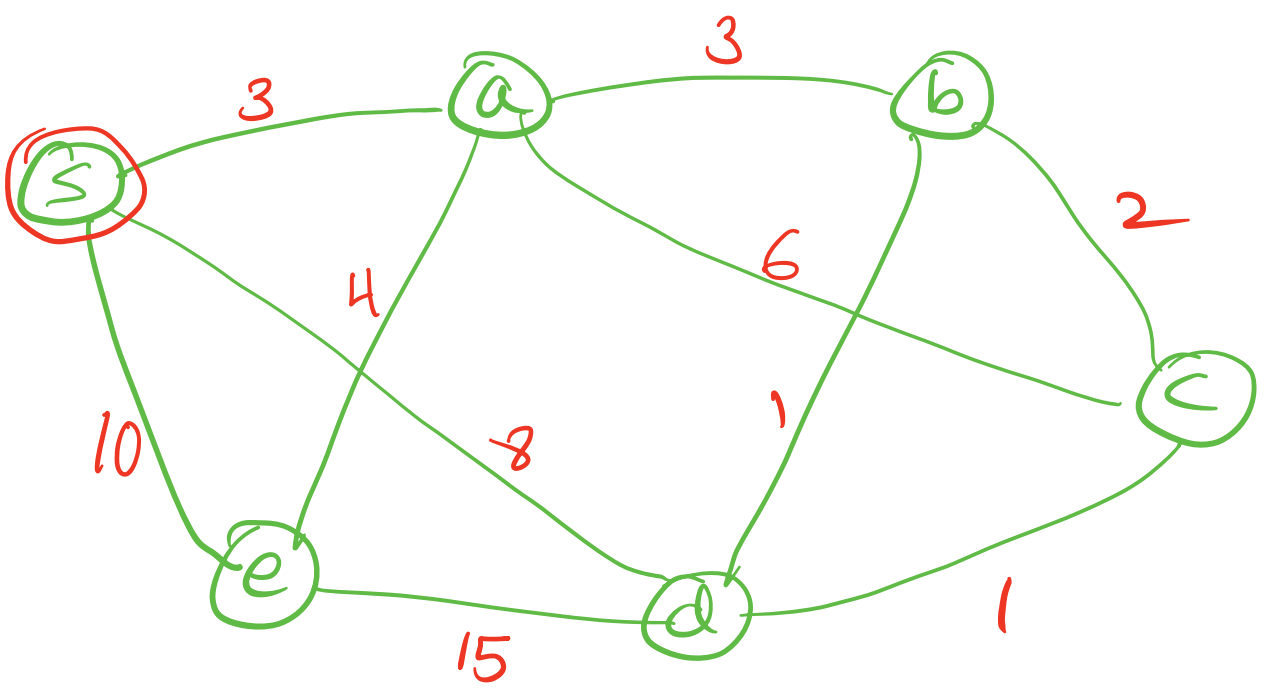
For a path $\langle v_1, v_2, \ldots, v_k \rangle$, the weight of the path is $\sum_{i=1}^{k-1} w(v_i, v_{i+1})$

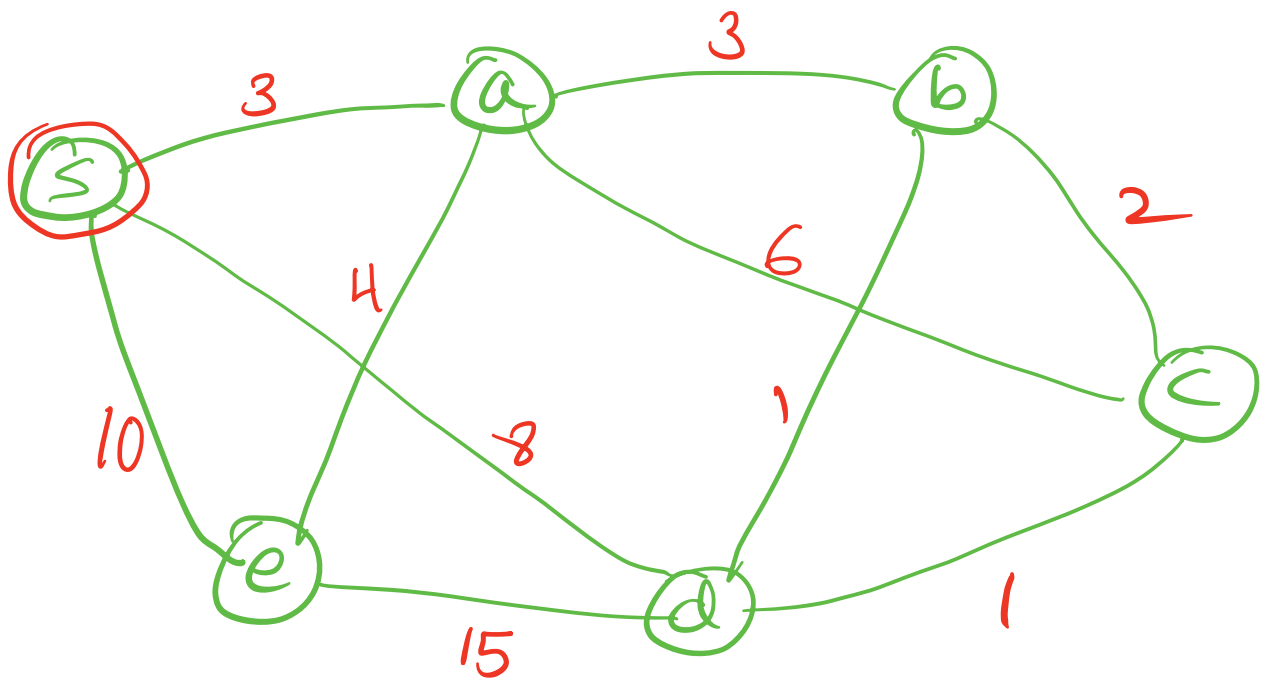
The shortest-path weight from u to v is denoted

"p is a u-v path in G"

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if } \exists \text{ a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

A shortest u-v path is a u-v path that realizes the shortest-path weight.

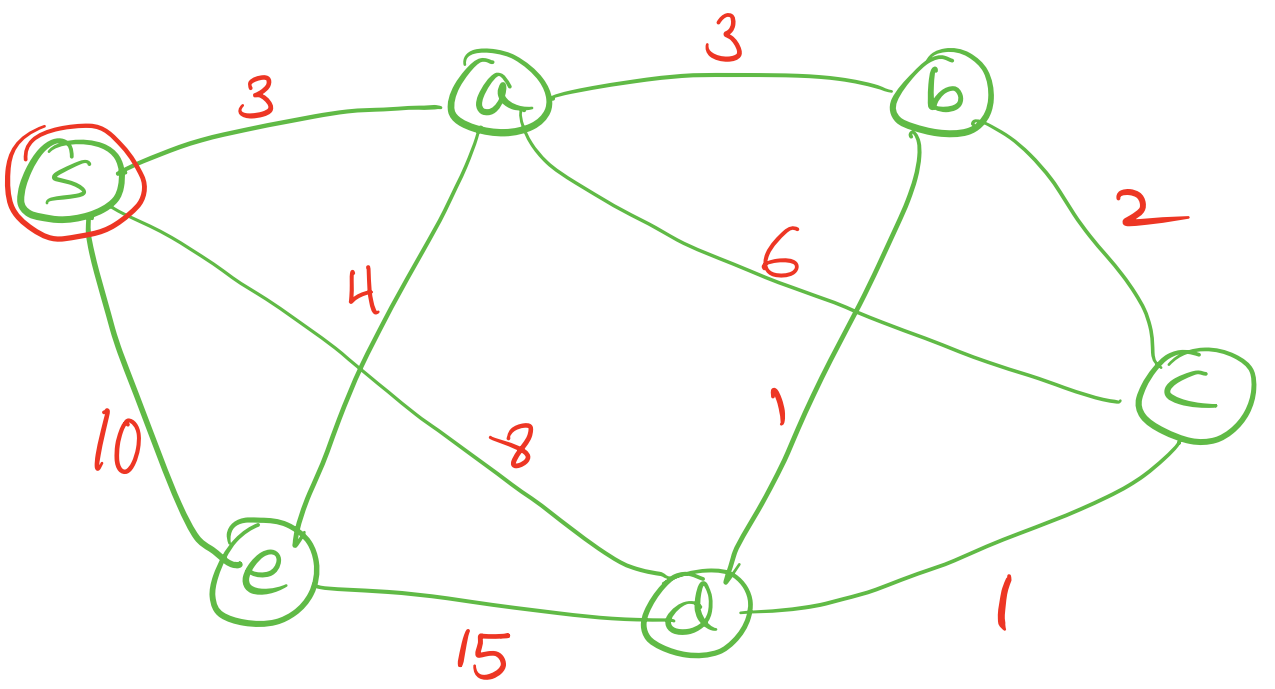


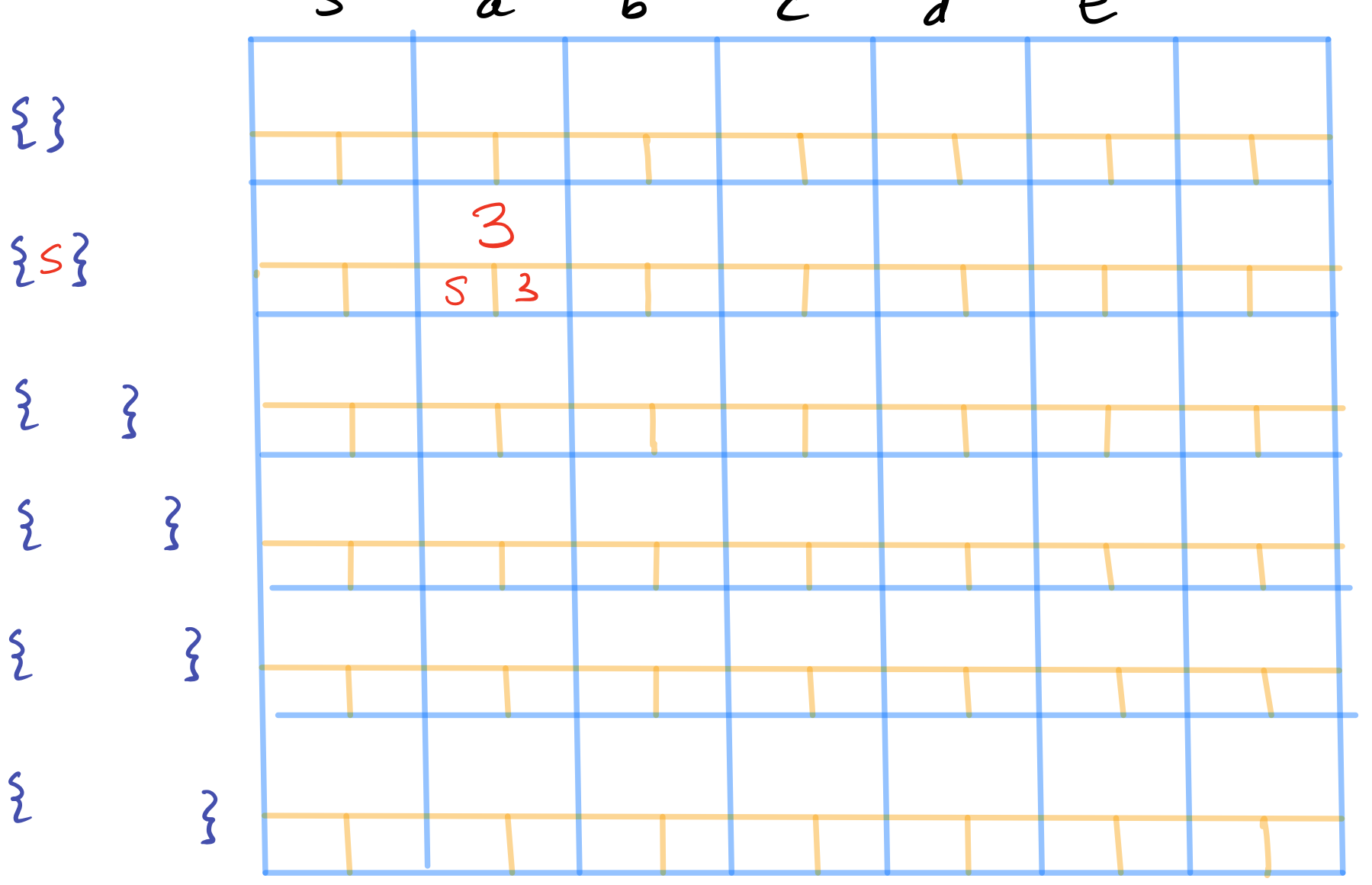


	s	a	b	c	d	e	
{}	0	∞	∞	∞	∞	∞	∞
{s}	0	3	∞	∞	8	10	∞
{s,a}	0	3	6	9	8	7	∞
{s,a,b}	0	3	6	8	7	7	∞
{s,a,b,d}	0	3	6	8	7	7	
{s,a,b,d,e}	0	3	6	8	7	7	

The set of elements that can precede v on the $s-v$ path

The min cost of any $s-v$ path that only visits those vertices on the way.


s
a
b
c
d
e
3
3
2
6
4
10
8
1
15
1


s a b c d e
{ }
{ s }
3
s 3
{ }
{ }
{ }
{ }
{ }

$d_k[s,y]$ = min weight of a s-y-path that only visits, as intermediate vertices (vertices along the way) the k closest vertices to s.

Start by calculating $d_0[s,y]$ $\forall y$.

$d_1[s,y]$ = length of shortest s-y path using $\{s\}$

$d_2[s,y]$ = shortest s-y path using $\{s, a\}$

$d_3[s,y]$ = shortest s-y path using $\{s, a, b\}$

$d_4[s,y]$ = shortest s-y path using $\{s, a, b, e\}$

etc.

Dijkstra's Algorithm

- solves single-source shortest paths on weighted, directed or undirected graphs when edge weights are non-negative.

Dijkstra (G, w, s) // w is weight function.
(G: Graph; s: source vertex)

Initialize Single Source

Set d[s] = ∅

Put each v∈V into a "special container" H.

Each v will have a key that we can set

While H is not empty

- extract vertex v from H that has minimum distance from S. (find the)
  among those verts in H i.e. vertices not yet permanently labelled
- relax all of v's neighbours that are still in H.

What kind of thing is H?

Operations:

init()

insert(v, k)

extractMin()

- removes and returns the (element, key) pair with minimum key value in H.

decreaseKey(v, k)

H is a Priority Queue ADT

- defined by the operations above.

How can H be implemented?