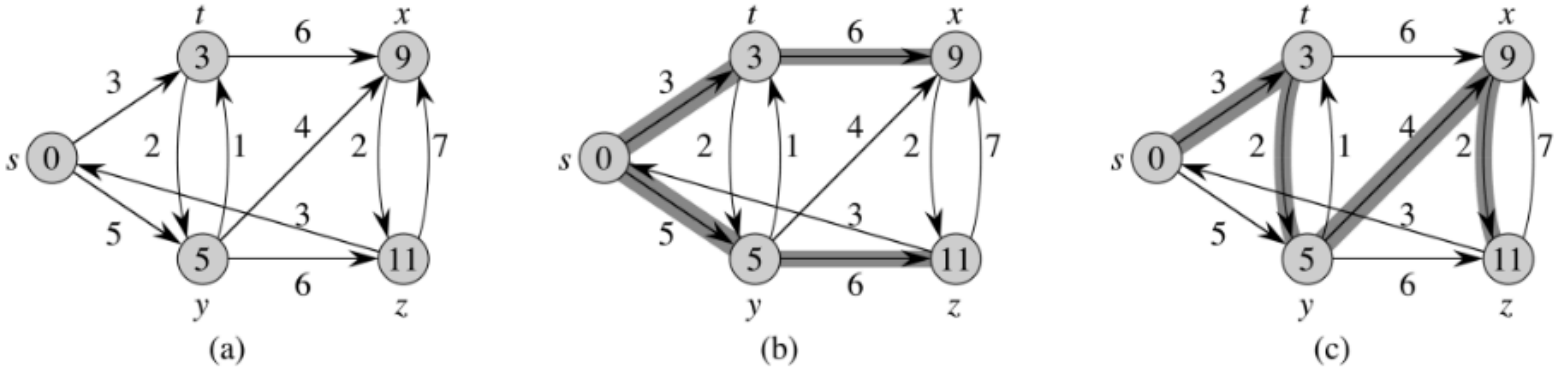
Shortest paths

Shortest paths in weighted graphs

Length of path= sum of the weights of the edges on the path.

δ(u, v)= the length of a shortest path from u to v. Set to ∞ if no path exists.

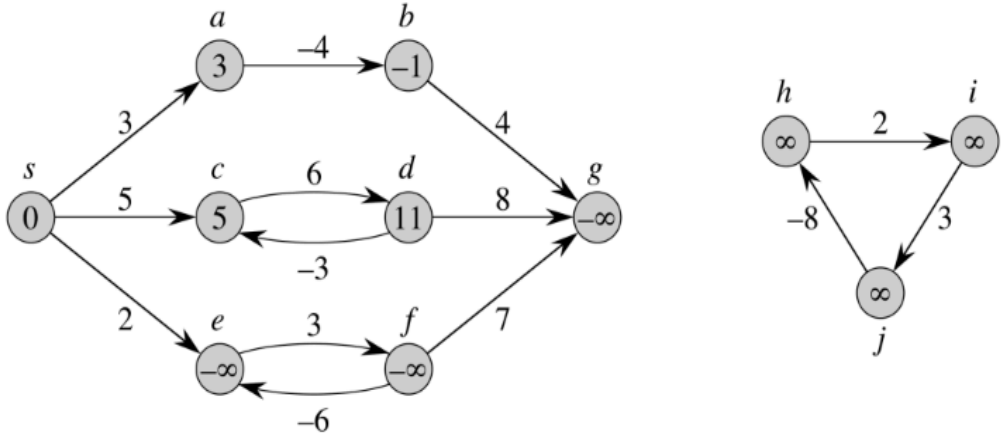
Single-source shortest-path problem: Given s ∈ V, find δ(s, v) (and a concrete path of this length) for all v ∈ V.



Note that prefixes of shortest paths must themselves be shortest paths: if v1, v2, ..., vk is a shortest path from v1 to vk, then v1, v2, ..., vi is a shortest path from v1 to vi for all i ≤ k (otherwise the path from v1 to vk could be made shorter).

Shortest paths in weighted graphs

The problem is not well-defined if there are cycles (reachable from s) with negative total weight, since then there exist paths of arbitrarily low length.



Conversely: if no such negative cycles exist, we can restrict to simple paths (no repeated nodes). There are finitely many such paths (at most n!), so the "length of a shortest path" is well-defined.

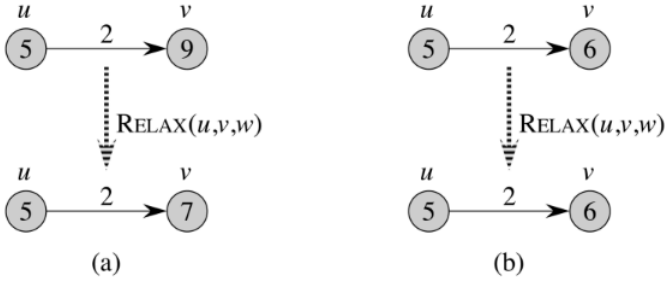
Relaxation – a general technique for finding shortest paths

Idea: Use edges to propagate information from node to node about the lengths of known paths. Called Relax an edge.

If u has information that there is a path from s to u of length u.d, and (u, v) is an edge with weight w, then there is a path of length u.d + w to v. Is this better information for v than what it currently has?

```
INIT-SINGLE-SOURCE(G, s)
  for each v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
      v.d = u.d + w(u, v)
      v.π = u
```



Detail: Implementation of “∞” must behave mathematically correctly with “>” and “+” (simply using Integer. MAX_VALUE as “∞” is not enough).

Relaxation

```
INIT-SINGLE-SOURCE(G, s)
  for each v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
      v.d = u.d + w(u, v)
      v.π = u
```

We will encounter a series of algorithms that start with Init-Single-Source and then only change v.d and v.π via Relax.

For such algorithms, the following invariant holds (easily seen by induction on the number of Relax operations):

> If v.d < ∞, there is a path from s to v of length v.d.

Relaxation

If $v.d < \infty$, there is a path from s to v of length v.d.

Therefore $\delta(s, v) \leq v.d$ always holds. Argument: if $v.d < \infty$, it follows from the above; if $v.d = \infty$, then $\delta(s, v) \leq v.d$ regardless of $\delta(s, v)$.

Since v.d can only decrease by Relax, it follows that if at some point $\delta(s, v) = v.d$, v.d cannot change later (and neither can $v.\pi$, since they change together in the code).

In particular, if $\delta(s, v) = v.d$ for all nodes, then no edge (u, v) can be relaxed (i.e., $v.d \leq u.d + w(u, v)$ holds for all edges (u, v)).

The shortest paths can be found via v.π pointers

Invariant:

> ► The set S of nodes v with $\delta(s, v) = v.d < \infty$ forms a tree with v.π as parent pointers and s as root.
>
> ► For a node v in the tree, the path to the root corresponds to a backward traversal of a path in the graph from s to v of length $\delta(s, v)$.

Proof by induction on the number of Relax.

Basis: Right after initialization, s is the only node with $v.d < \infty$. Since $s.d = 0$ and $s.\pi = \text{NIL}$, $S = \{s\}$, and the invariant holds with a single-node tree, provided $\delta(s, s) = 0$.

[Note that $\delta(s, s) \neq 0$ only if s is on a negative cycle, in which case S is empty. But then for all nodes v either $\delta(s, v) = -\infty$ (if v can be reached from s) or $\delta(s, v) = \infty$ (if v cannot be reached from s). If $v.d < \infty$, answers v.d to the length of a specific path (see earlier), and is therefore different from $-\infty$. So S is always empty and there is nothing to show.]

The shortest paths can be found via v.π pointers

Inductive step: A Relax that changes nothing is trivial. For a Relax that changes v.d (and thus v.π): v was not in S before (since $\delta(s, v) < v.d$ at that time), so existing nodes in the tree are unchanged (including their parent pointers).

If v is added to S by this Relax via edge (u, v) of weight w, then $\delta(s, v) = v.d = u.d + w$. Thus $\delta(s, u) = u.d$ (else a shorter path to v), and $u.d < \infty$ (else Relax would not occur). Hence $u \in S$, gets v as a child, and the invariant holds.

Dijkstra's algorithm [1959]

Greedy algorithm that builds the set S of nodes with correct v.d and v.π step by step. Uses a priority queue Q. Requires all edge weights ≥ 0.

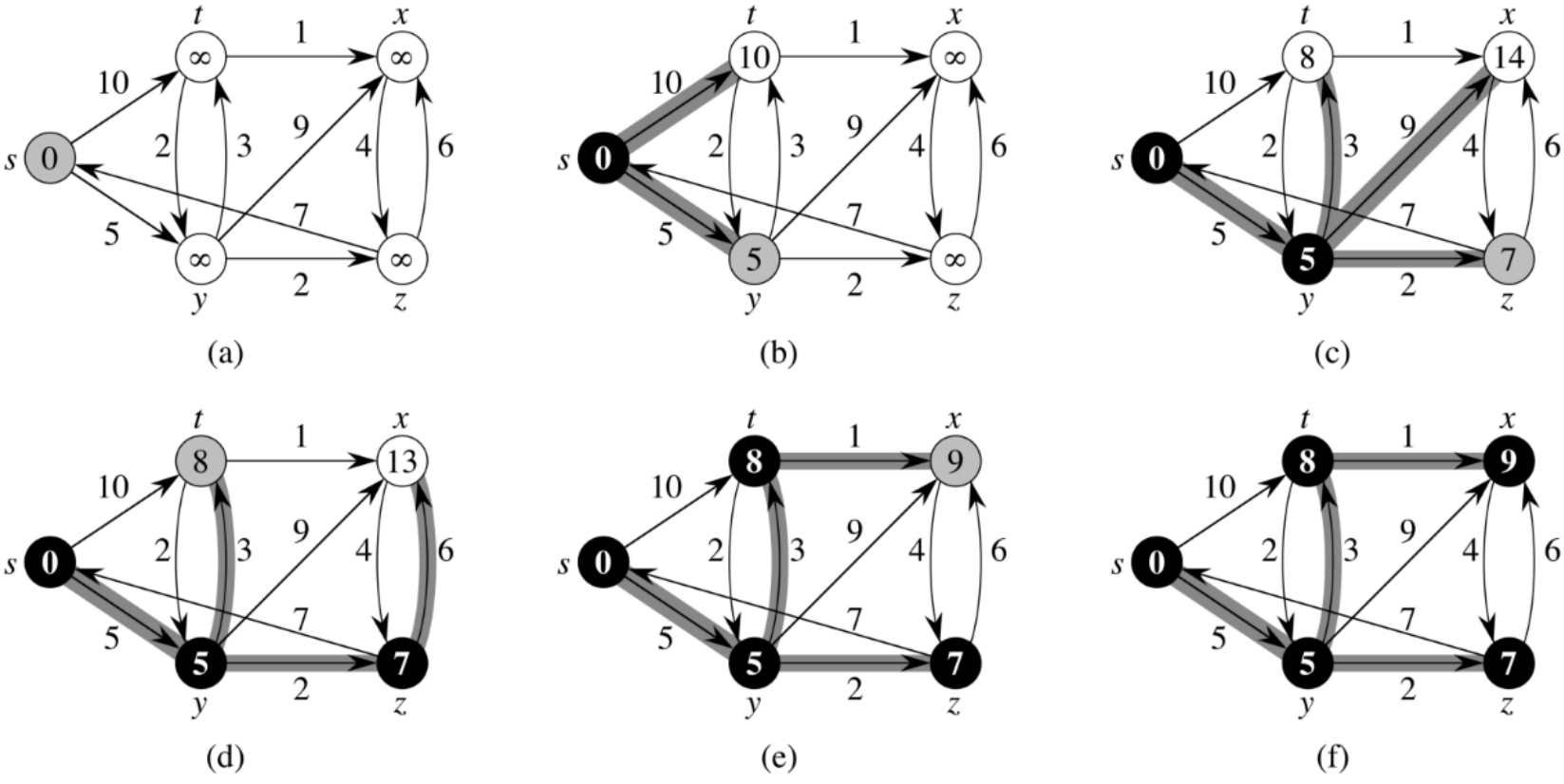
```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Running time: n Insert (or one Build-Heap), n Extract-Min, and m Decrease-Key operations in Relax. Total O(m log n) with a heap-based priority queue.

Invariant: When u is added to S (i.e., extracted by an Extract-Min), we have u.d = δ(s, u) (assuming all edge weights are ≥ 0).

Proof of the invariant: By induction (as worked through on the board). From this invariant it follows that the algorithm is correct (since eventually all vertices end up in S).

Dijkstra, example



A* [Hart, Nilsson, Raphael, 1968]

The A* algorithm can be seen as a tuning of Dijkstra's algorithm for the (often encountered) case where one is searching for a path from s to a specific target node t:

New ingredient: For every node v , attempt to make a guess h(v) of the shortest distance from v to t , i.e. an estimate of δ(v,t). One also calls h(v) a "heuristic."

Intuition: If v.d (as in Dijkstra's algorithm, when v is extracted from the priority queue) equals δ(s,v), then v.d + h(v) is a guess for δ(s, v) + δ(v,t), which is the length of the shortest path from s to t that goes through v.

Idea: Proceed exactly as in Dijkstra's algorithm (including the same updates of the v.d values), but let the key in the priority queue be v.d + h(v).

Thus, you expand your search via those nodes that are guessed to lie on the shortest path from s to t. By comparison, Dijkstra's algorithm expands in order of nodes known to be closest to s.

A* in practice

Example with a grid-based graph. Nodes are the white grid cells, with edges of length one between white neighboring cells. The heuristic $h(v)$ is the Euclidean ("as the crow flies") distance from cell v to the goal cell t.

Dijkstra (right figure): Explores uniformly in all directions.

A* with the above heuristic (left figure): Explores more toward the goal. Fewer nodes are visited, so it's faster in practice.

Correctness and worst case running time of A*?

A heuristic is called consistent if, for every node v and every edge (v,u) in v's adjacency list, the following holds:

$$h(v) \leq w(v, u) + h(u)$$

That is, the heuristic's estimate of the shortest path to t for v's neighbors is never "overly optimistic" compared to its estimate for v itself. One can show that with a consistent heuristic, A* is equivalent to running Dijkstra's algorithm on a graph with reweighted edges.

From this it follows that A* is correct (it always returns the true shortest path from s to t) and that its worst-case running time matches the worst-case running time of Dijkstra's algorithm.

Path relaxation lemma

Dijkstra (and A*) assumes non-negative weights. We now look at algorithms that can handle negative weights (but of course not negative circles, which makes the shortest path problem undefined).

We start with the following lemma:

Lemma: If $s = v_1, v_2, \ldots, v_k = v$ is a shortest path from s to v, and an algorithm makes Relax on the edges $(v_1,v_2),(v_2,v_3), \ldots,(v_{k-1},v_k)$ in that order (possibly interleaving arbitrary Relax operations on other edges), then after the last of these Relax operations we have then is $\delta(s, v) = v.d$ after the last of these Relax.

Proof: It is seen by induction on i that immediately after Relax is applied to edge (v_{i-1},v_i) in the sequence above, $v_i.d$ can be at most equal to the sum of the weights of the first $i-1$ edges in the path.

So after the last of these Relax $v.d \leq \delta(s, v)$, since the path is the shortest path to v. Then $\delta(s, v) \leq v.d$ always applies, $\delta(s, v) = v.d$.