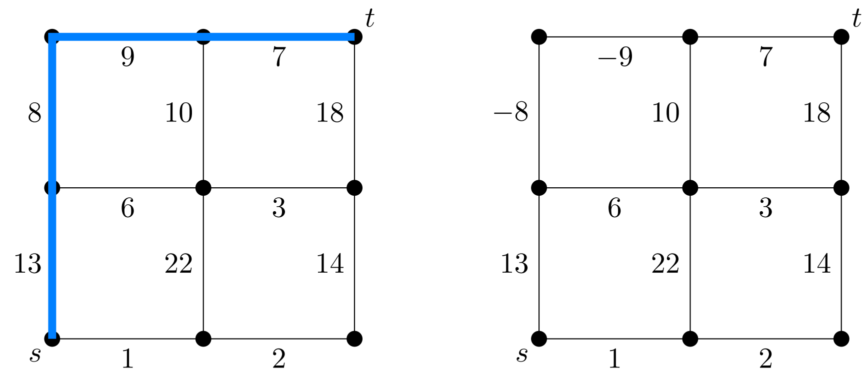
Shortest path

Source: Chapter 2.2 (Bills), Chapter 7 of Combinatorial Optimization (Korte)

Practical problem: Drive between 2 points, at every intersection you can choose, where to turn (but no U turn). What is the best way to go?

Shortest path
Input: Graph $G = (V, E)$, costs $c : E \to \mathbb{R}$, and $s, t \in V$.
Output: s-t-path P, where $\sum_{e \in P} c(e)$ is minimized.

1: Find the shortest (lowest cost) s-t-paths in the following graphs



Notice the graph on the right contains a cycle in the left upper corner with negative cost. Perhaps one would just want to keep cycling there.

The cost c is called *conservative* if there is no circuit of negative total cost.

Bellman's principle: Let $s, \ldots, v, w$ be the least cost s-w-path of length k. The $s, \ldots, v$ is the least cost s-v-path of length $k - 1$.

2: Prove Bellman's principle.

Solution: By contradiction. If there is a lower cost path to v, we could find a lower cost path to w.

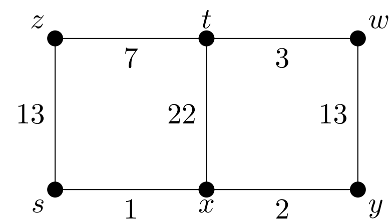
Notice: This gives a recursion for computing the shortest path.

Dijkstra's algorithm
$c : E \to \mathbb{R}^+$, computes shortest s-t-path from s to ALL other vertices $t \in V$.

1. $\ell(s) := 0$; $\forall v \neq s$ $\ell(v) = +\infty$
2. $R = \emptyset$
3. while $R \neq V$
4. find $v \in V - R$ with minimum $\ell(v)$
5. $R := R \cup \{v\}$
6. $\forall vw \in E$, $\ell(w) = \min\{\ell(w), \ell(v) + c(v, w)\}$

R … vertices with final number; ℓ … upper bound on the cost;
The running time is $O(n^2)$ easily or $O(m + n \log n)$ when implemented using Fibonacci heaps.

3: Run Dijkstra's algorithm on the following graph



R	$\ell(s)$	$\ell(x)$	$\ell(y)$	$\ell(z)$	$\ell(w)$	$\ell(t)$
$\emptyset$	0	∞	∞	∞	∞	∞
$\{s\}$	0	1	∞	13	∞	∞
$\{s,x\}$	0	1	3	13	∞	23
$\{s,x,y\}$	0	1	3	13	16	23
$\{s,x,y,z\}$	0	1	3	13	16	20
$\{s,x,y,z,w\}$	0	1	3	13	16	19
$\{s,x,y,z,w,t\}$	0	1	3	13	16	19

Notice how $\ell(t)$ is slowly decreasing and getting closer and closer to the correct answer.

4: How do we find the shortest s-v-path?

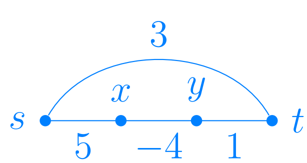
Solution: Remember previous vertex. In step 6. of the algorithm, remember why the value was changed. So called *predecessor*. Recall Bellman's principle. In step 6. of the algorithm, when we decrease $\ell(w)$, always remember which v caused it. The last such v is called *predecessor* and by using predecessors, one can reconstruct the path.

5: Why is the algorithm correct? (show that if $v \in R$, then $\ell(v) =$ cost for s-v-path.)

Solution: Suppose for contradiction the algorithm produces an incorrect result. Then some vertex w can be reached from s by a path $s, \ldots, v, w$ shorter than $\ell(w)$. It follows that the predecessor v of w in the path is also closer than $\ell(v)$. By considering the number of vertices along such a path, we arrive at a contradiction.

6: Why doesn't Dijkstra's algorithm work for negative costs?
(How about an example?)

Solution: The assumption that we can fix the cost of the lowest visited so far is not true.



R	$\ell(s)$	$\ell(x)$	$\ell(y)$	$\ell(t)$
$\emptyset$	0	∞	∞	∞
$\{s\}$	0	5	∞	3
$\{s,t\}$	0	5	4	3
$\{s,t,y\}$	0	0	4	3
$\{s,t,y,x\}$	0	0	-4	3

Costs of shortest paths: s-t: 2; s-x: 0, s-y: 1. Notice that the algorithm incorrectly assumed that shortest path to t uses edge of cost 3 and then it traversed edge xy there and back.

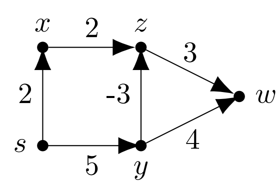
We fixing mistakes of Dijkstra's algorithms by 1) repeatedly check for possible improvements for all $\ell(v)$ where $v \in V$ and 2) working with directed graph (prevents traversing an edge with negative weight back and forth).

Moore-Bellman-Ford Algorithm
$c : E \to \mathbb{R}$, computes shortest s-t-path in a **directed** graph $G = (V, E)$ from s to ALL other vertices $t \in V$ **OR** finds a cycle of negative cost. Assume $|V| = n$.

1. $\ell(s) := 0$; $\forall v \neq s$ $\ell(v) = +\infty$
2. repeat $n-1$ times: *//computes the costs*
3. $\quad$ $\forall vw \in E$,
4. $\quad\quad$ if $\ell(w) > \ell(v) + c(v, w)$
5. $\quad\quad\quad$ $\ell(w) := \ell(v) + c(v, w)$; $p(w) = v$
6. $\forall vw \in E$, *//check for a negative cycle*
7. $\quad$ if $\ell(w) > \ell(v) + c(v, w)$ then found negative cycle

Note: ℓ gives the least cost, while p gives the **previous** vertex / **predecesor** on the shortest path from s.

7: Run the Dijkstra's algorithm and Moore-Bellman-Ford algorithm on the following graph and notice the result at w.



Solution: Dijkstra's algorithm will result in cost 7 to w using path s, x, z, w while the shortest path s, y, z, w has cost 5. We run the algorithm with edge ordering yw, zw, xz, yz, sy, sx.

R	$\ell(s)$	$\ell(x)$	$\ell(y)$	$\ell(z)$	$\ell(w)$
$\emptyset$	0	∞	∞	∞	∞
$\{s\}$	0	2	5	∞	∞
$\{s, x\}$	0	2	5	4	∞
$\{s, x, z\}$	0	2	5	4	7
$\{s, x, z, y\}$	0	2	5	2	7
$\{s, x, z, y, w\}$	0	2	5	2	7

$\ell(s), p(s)$	$\ell(x)$	$\ell(y)$	$\ell(z)$	$\ell(w)$
$0, -$	$\infty, -$	$\infty, -$	$\infty, -$	$\infty, -$
$0, -$	$2, s$	$5, s$	$\infty, -$	$\infty, -$
$0, -$	$2, s$	$5, s$	$2, y$	$9, y$
$0, -$	$2, s$	$5, s$	$2, y$	$5, z$

8: What is the time complexity of the algorithm if G has m edges and n vertices?

Solution: $O(nm)$.

9: "How does the algorithm detect a negative cycle? Why does the algorithm work?

Solution: The longest *shortest* s-t path can have up to $n-1$ edges. So the algorithm examines all of them. TODO: How about an example with a cycle and let students try it?