Graph Theory

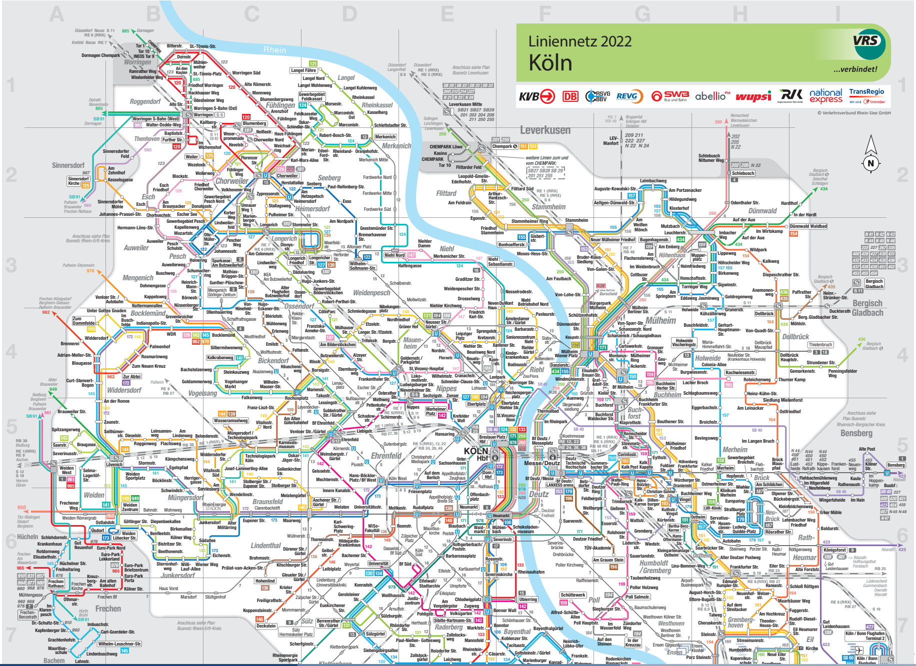
Dr. Jens Dörpinghaus

doerpinghaus@uni-koblenz.de

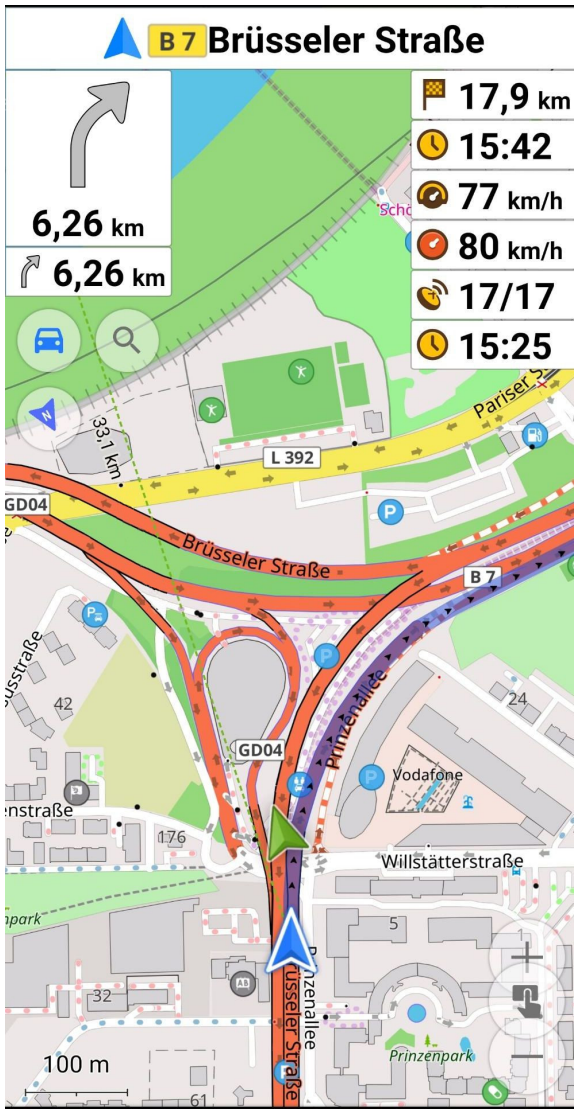
Mathematisches Institut
Universität Koblenz-Landau
Campus Koblenz

Federal Institute for Vocational Education and Training (BIBB)
Robert-Schuman-Platz 3
53175 Bonn

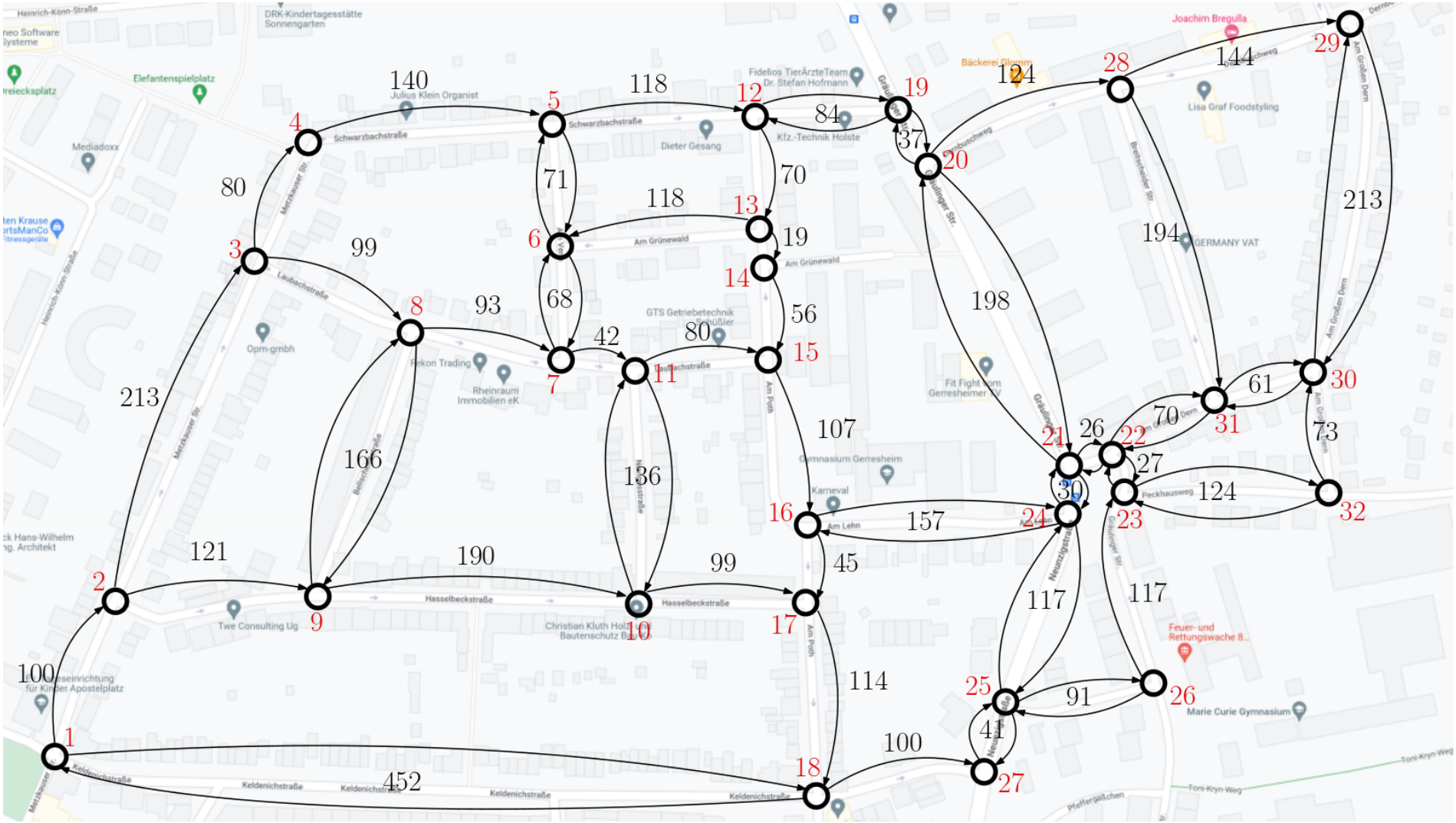
Summer 2022


Liniennetz 2022
Köln
VRS
...verbindet!
© Verkehrsverbund Rhein-Sieg GmbH

This problem is probably one of the easiest and an everyday problem:
Finding the *shortest path* in a network.
It is widely used, not only for navigation.



Düsseldorf, Germany:



Definition 3.1

Let $D = (V, A)$ be a directed graph and $s, t \in V$ two vertices.
For a path $P = [v_0, v_1, \ldots, v_m]$, we set $s = v_0$ the starting node and $t = v_m$ the end node and call P an *s-t path*.

We need lengths for the edges to determine the length of a path P.

Therefore we define the weight function on edges to set the length

$$\ell : E \to \mathbb{R}.$$

Often, especially when searching for a shortest path, we will talk about the length or distance function instead of weight and restrict it to nonnegative or integer lengths.

For a path $P = (v_0, ..., v_m)$,

$$\ell(P) := \sum_{i=1}^{m} \ell(\{v_{i-1}, v_i\})$$

is the length of P and the *distance* $dist(v, w)$ of two nodes $v, w \in V$ in a weighted graph is defined as the minimum of the lengths of all paths from v to w.

If no such path exists, we set $dist(v, w) = \infty$.

There are several reasons why we consider directed graphs:

- A route between two nodes might not be accessible for both directions (e.g. one-way streets).
- Different directions may get different weights (measuring duration instead of length).
- However, undirected weighted graphs can be seen as directed graphs with edges of the same weight in both directions.

We can divide into four basic problem classes:

1. s – t path or *single-pair shortest path* (SPSP): Finding a shortest path from s to t.
2. *Single-source shortest path* (SSSP): Finding a shortest path from s to all nodes $v \in V$.
3. *Single-destination shortest path* (SDSP): Finding a shortest path from all nodes $v \in V$ to a end node t.
4. *All-pairs shortest path* (APSP): Finding the distance (a shortest path) between every pair of vertices $u, v \in V$.

In addition we may consider the following extra restrictions:

- non-negative weights,
- without negative circuits,
- arbitrary weights.

- An algorithms for s – t path problem may also compute the shortest path to all nodes at least partially.
- We can convert SDSP to SSSP by inverting the edges and using t as the starting node.
- For APSP, either the shortest path can be computed for each node using an algorithms s – t path, or there are special algorithms (e.g., the Floyd-Warshall algorithm or the min-plus-matrix multiplication algorithm), which we will not consider in the lecture.
- Negative weights in a graph make the problem more complex and we will consider this special case separately.
- Negative circles make the problem $\mathcal{NP}$-complete.

There is a long list of algorithms for shortest paths. These are some of the most popular:

- Dijkstra's algorithm (SSSP with nonnegative weights).
- Bellman–Ford algorithm (SSSP - edge weights may be negative).
- A* search algorithm (SPSP – can be seen as a extension of Dijkstra by using heuristics to try to speed up the search).
- Floyd–Warshall algorithm (APSP).
- Johnson's algorithm (APSP – possibly faster than Floyd–Warshall on special graphs)

In addition there is a much longer list of different modifications and implementations of the mentioned algorithms.

First, we take a closer look at the algorithm developed by Dijkstra. After that, we will discuss Bellmann and Ford's algorithm.

Edsger Wybe Dijkstra (1930–2002) was a Dutch computer scientist who is considered the first Dutch programmer and was a pioneer in computing science. He developed this algorithm in 1956 and published it in 1959.

Algorithm 3 Dijkstra's algorithm

Input Directed Graph $D = (V, A)$, nonnegative length function ℓ on A,
Input starting node $s \in V$

1: $dist(s) := 0$,
2: $pred(v) = undef$, $dist(v) := \infty \; \forall v \in V \setminus \{s\}$
3: $U := V$
4: **if** $U = \emptyset$ **then**
5: STOP
6: Find a $u \in U$, for which $dist(u)$ is minimal.
7: **for** $v \in U$ with $(u, v) \in A$ **do**
8: set $dist(v) := min\{dist(v), dist(u) + \ell(u, v)\}$
9: set $pred(v) = u$
10: Set $U := U \setminus \{u\}$. Goto 4.

Theorem 3.2

Let $D = (V, A)$ be a connected digraph, $s \in V$ and $\ell : A \to \mathbb{Q}_+$ a non-negative weight function on the edges. The Dijkstra's algorithm computes the distances (the length of a shortest path s-v-path) from a starting node s to all $v \in V$.

We can find the shortest paths by backtracking.

Proof.

First, we note that each node from V in U is selected exactly once and one node is removed from U in each iteration of the for loop.
Thus, the algorithm is finite.

- With $\widetilde{U}$ we denote the ordered set of the respective active nodes, i.e. the nodes which are removed from U in step 10.
- We show by induction over $|\widetilde{U}|$ that at any point in the algorithm, for all nodes $v \in \widetilde{U}$, the distance $dist(v)$ is the shortest s-v-path that passes only over visited nodes (i.e., only with nodes from $\widetilde{U}$).
- (By backtracking over the $prev(v)$, the shortest path can be determined.)

First, $\widetilde{U} = s$, i.e. $|\widetilde{U}| = 1$.

- Let $|\widetilde{U}| > 1$, $v = v_k$ be the last node visited and
- with $s = v_0$ let $P = [v_0, ..., v_k]$ we denote the v_0-v_k path given by the $pred(v_i)$, where $v_{i-1} = pred(v_i)$.
- If in the fourth step $dist(v_k)$ is applied, we inductively conclude that the length $\ell(P) := \sum_{i=1}^{k} \ell(\{v_{i-1}, v_i\}) = dist(v_k)$.

Proof.

- Suppose $\hat{P} = [v_0, \hat{v}_1, ..., \hat{v}_{\hat{k}}]$, with $s = \hat{v}_0$ is a second path with $\ell(\hat{P}) < dist(v_k)$.
- Let i be the largest index for which $\hat{v}_i \in S \setminus \{v\}$ and $(\hat{v}_i, \hat{v}_{i+1}) \in A$.
- If $\hat{v}_i$ was selected in the third step, the fourth step is to check if

$$dist(\hat{v}_i) + \ell((\hat{v}_i, \hat{v}_{i+1})) < dist(\hat{v}_{i+1}).$$

- Then

$$dist(\hat{v}_{i+1}) \leq dist(\hat{v}_i) + \ell((\hat{v}_i, \hat{v}_{i+1})$$

must hold.

- By induction assumption $dist(\hat{v}_i)$ is the length of a shortest s-$\hat{v}_i$-path and
- thus follows

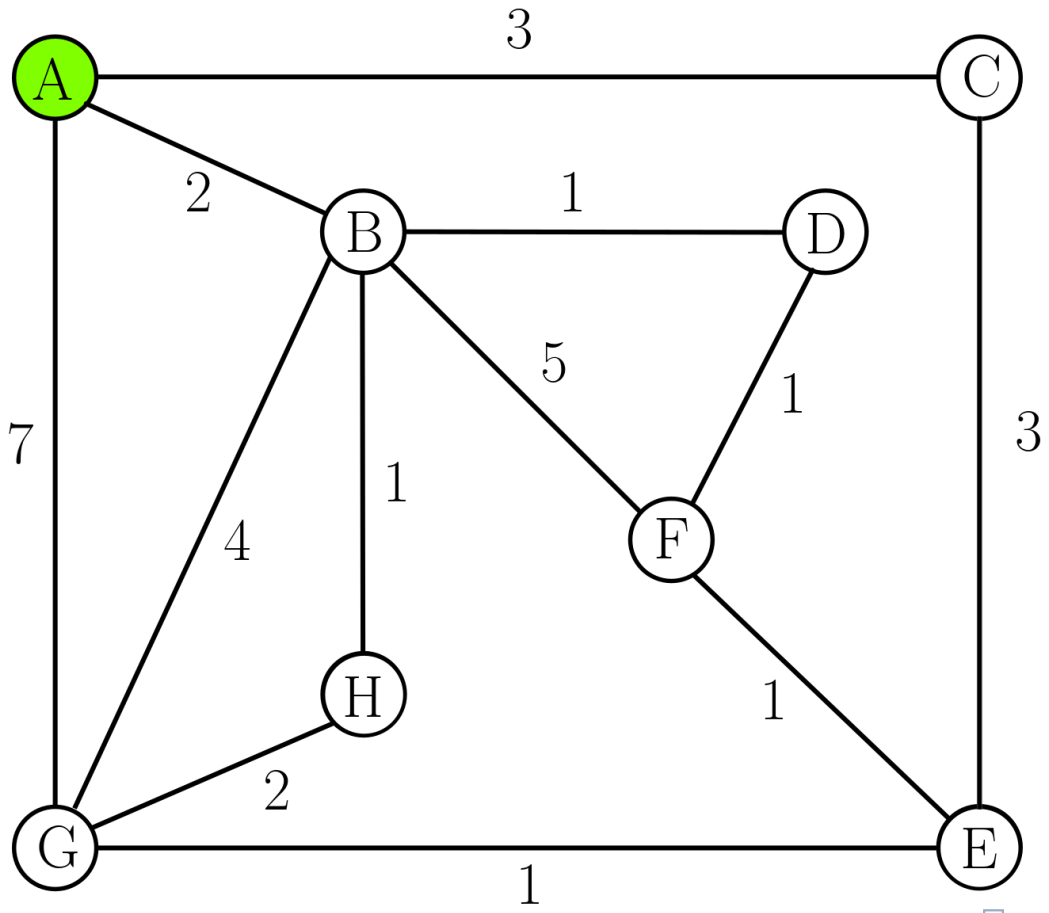
$$dist(\hat{v}_i) + \ell((\hat{v}_i, \hat{v}_{i+1})) \leq \ell(\hat{P}) < \ell(P) = dist(v).$$

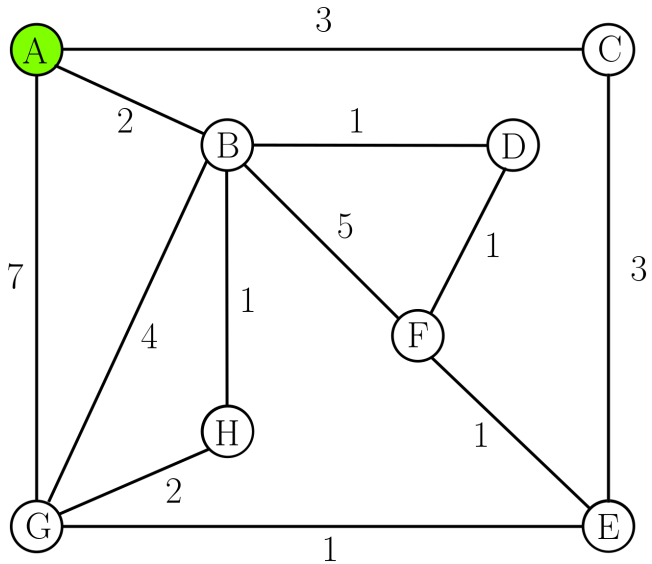
- Thus we should have chosen $\hat{v}_{i+1} \notin \widetilde{U}$ in the third step before v from U, which results in a contradiction.

□

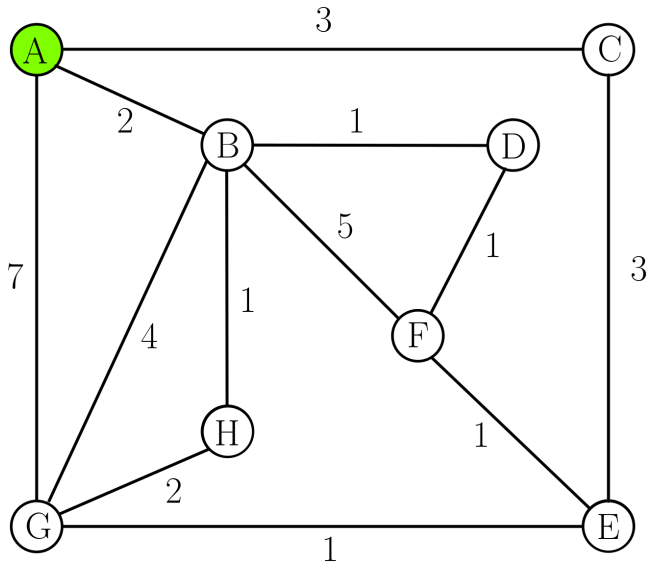
Example 3.3

We apply Dijkstra's algorithm to the following example of an undirected graph and calculate the shortest paths from node A to all other nodes. In the set $\widetilde{U}$ we collect the respective active nodes in order. In the table we update the current distances and predecessors for every step.

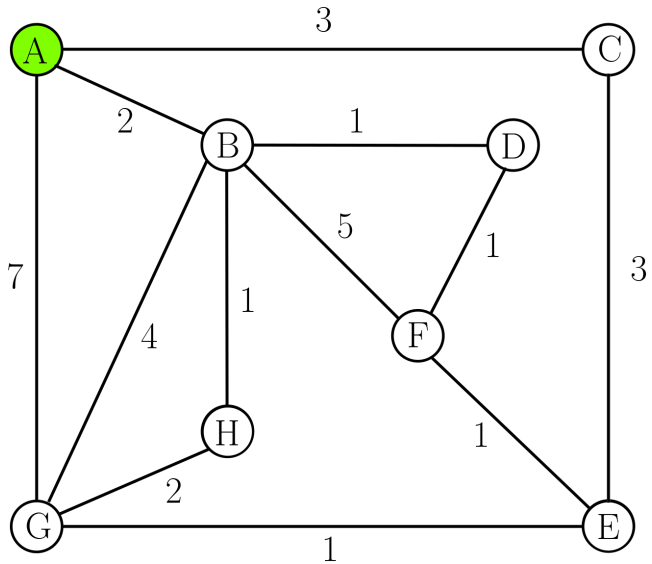




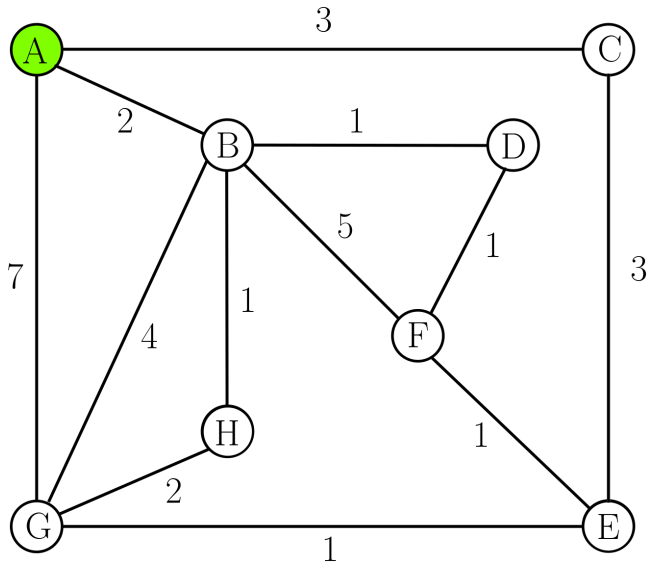
$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-



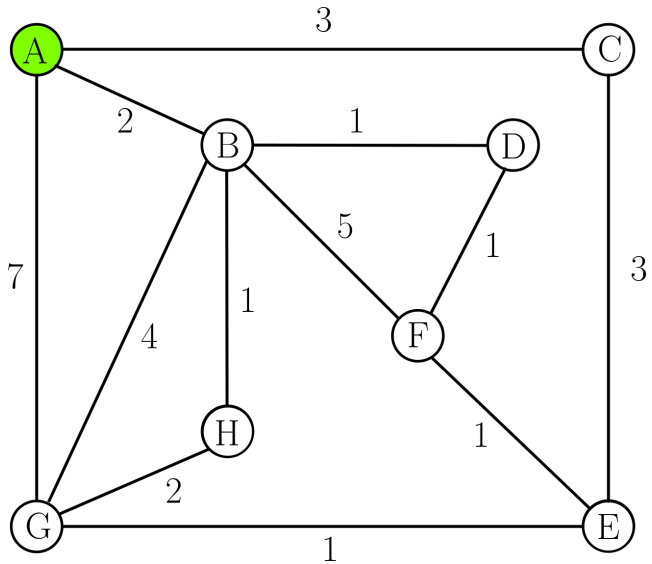
$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-



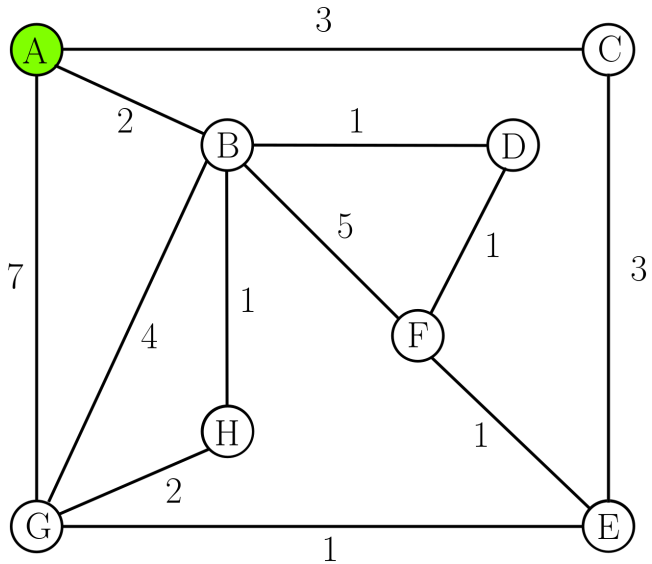
$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B



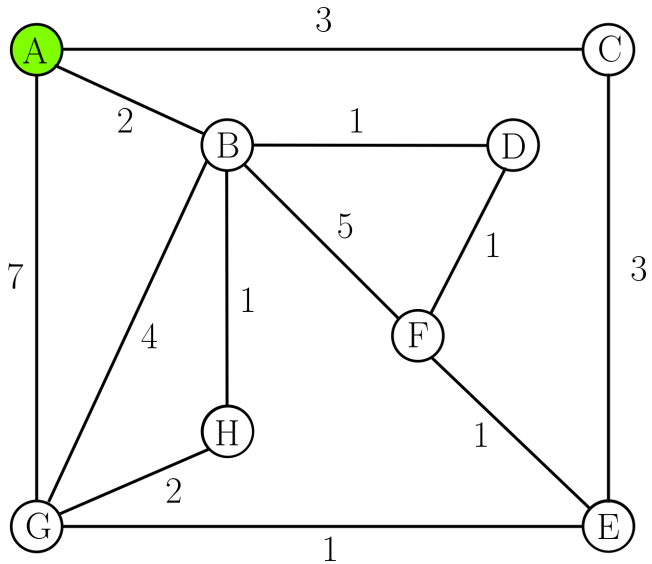
$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B



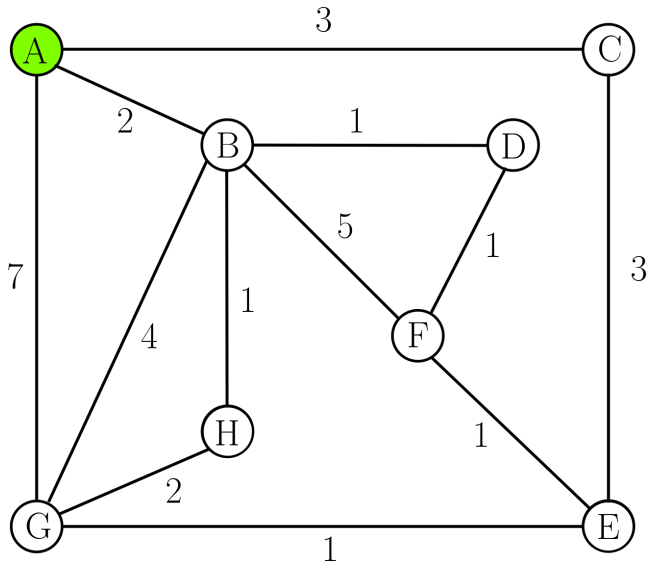
$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B
$\widetilde{U} = \{A, B, C, D\}$	dist					6	4	6	3
	pred					C	D	B	B



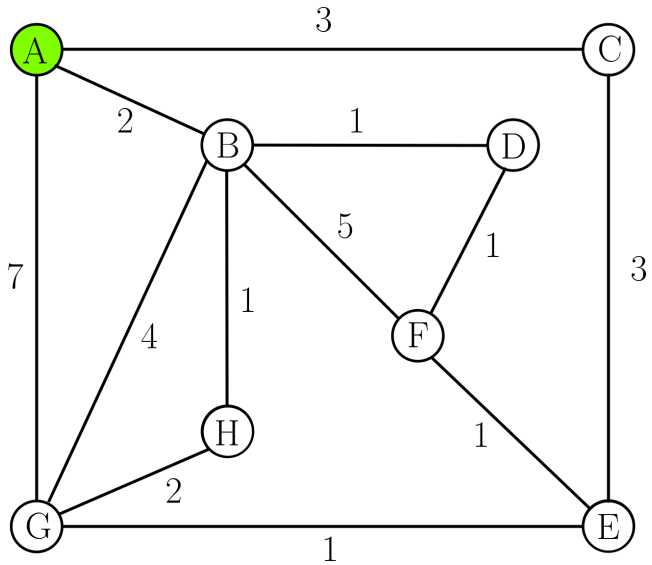
$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B
$\widetilde{U} = \{A, B, C, D\}$	dist					6	4	6	3
	pred					C	D	B	B
$\widetilde{U} = \{A, B, C, D, H\}$	dist					6	4	5	
	pred					C	D	H	



$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B
$\widetilde{U} = \{A, B, C, D\}$	dist					6	4	6	3
	pred					C	D	B	B
$\widetilde{U} = \{A, B, C, D, H\}$	dist					6	4	5	
	pred					C	D	H	
$\widetilde{U} = \{A, B, C, D, H, F\}$	dist					5		5	
	pred					F		H	



$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B
$\widetilde{U} = \{A, B, C, D\}$	dist					6	4	6	3
	pred					C	D	B	B
$\widetilde{U} = \{A, B, C, D, H\}$	dist					6	4	5	
	pred					C	D	H	
$\widetilde{U} = \{A, B, C, D, H, F\}$	dist					5		5	
	pred					F		H	
$\widetilde{U} = \{A, B, C, D, H, F, E\}$	dist							5	
	pred							H	



$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B
$\widetilde{U} = \{A, B, C, D\}$	dist					6	4	6	3
	pred					C	D	B	B
$\widetilde{U} = \{A, B, C, D, H\}$	dist					6	4	5	
	pred					C	D	H	
$\widetilde{U} = \{A, B, C, D, H, F\}$	dist					5		5	
	pred					F		H	
$\widetilde{U} = \{A, B, C, D, H, F, E\}$	dist							5	
	pred							H	
$\widetilde{U} = \{A, B, C, D, H, F, E, G\}$									

$\widetilde{U}$		A	B	C	D	E	F	G	H
$\widetilde{U} = \{\}$	dist	0	∞	∞	∞	∞	∞	∞	∞
	pred	-	-	-	-	-	-	-	-
$\widetilde{U} = \{A\}$	dist		2	3	∞	∞	∞	7	∞
	pred		A	A	-	-	-	A	-
$\widetilde{U} = \{A, B\}$	dist			3	3	∞	7	6	3
	pred			A	B	-	B	B	B
$\widetilde{U} = \{A, B, C\}$	dist				3	6	7	6	3
	pred				B	C	B	B	B
$\widetilde{U} = \{A, B, C, D\}$	dist					6	4	6	3
	pred					C	D	B	B
$\widetilde{U} = \{A, B, C, D, H\}$	dist					6	4	5	
	pred					C	D	H	
$\widetilde{U} = \{A, B, C, D, H, F\}$	dist					5		5	
	pred					F		H	
$\widetilde{U} = \{A, B, C, D, H, F, E\}$	dist							5	
	pred							H	
$\widetilde{U} = \{A, B, C, D, H, F, E, G\}$									

Runtime and implementations

- The runtime depends strongly on the implementation and in particular on the selected data structure for the node set, since this influences the determination of the node with the minimum distance.
- Since each edge in the for loop is called at least once, there are few possibilities for optimization here. The simplest implementation as an array or list for the node set results in a runtime of

$$O(|A| + |V^2|) = O(V^2).$$

- If the length is bounded by a constant c and all edge weights are integers, it is possible to sort via *bucket sort* in linear time.
- Extending this idea to the general situation by logarithmic buckets, we obtain a running time of

$$O(|V|log(|B|)),$$

  where B is the maximum edge length.
- We can improve the runtime if we store the nodes in a sorted list with a suitable data structure (e.g., a heap) and obtain a runtime of $O(|V|log(|V|))$ for the initial construction and updates. In general, this runtime is optimal with a *Fibonacci heap*.

In the next example, we will see that sometimes negative edge lengths can be useful, if you want to find a longest s-t path (i.e. a shortest negative path).

Example 3.4 (Knapsack problem)

We have a backpack with a volume of 10l and 7 useful items.

item i	volume a_i	benefit b_i
1: sleeping bag	5 l	5
2: thermos flask	1 l	2
3: tent	4 l	4
4: camping stove	1 l	3
5: camping mat	3 l	3
6: food	2 l	5
7: light	1 l	3

Task: Find a selection of 1,...,7 so that the items fit into the backpack and their summed benefit is maximal.

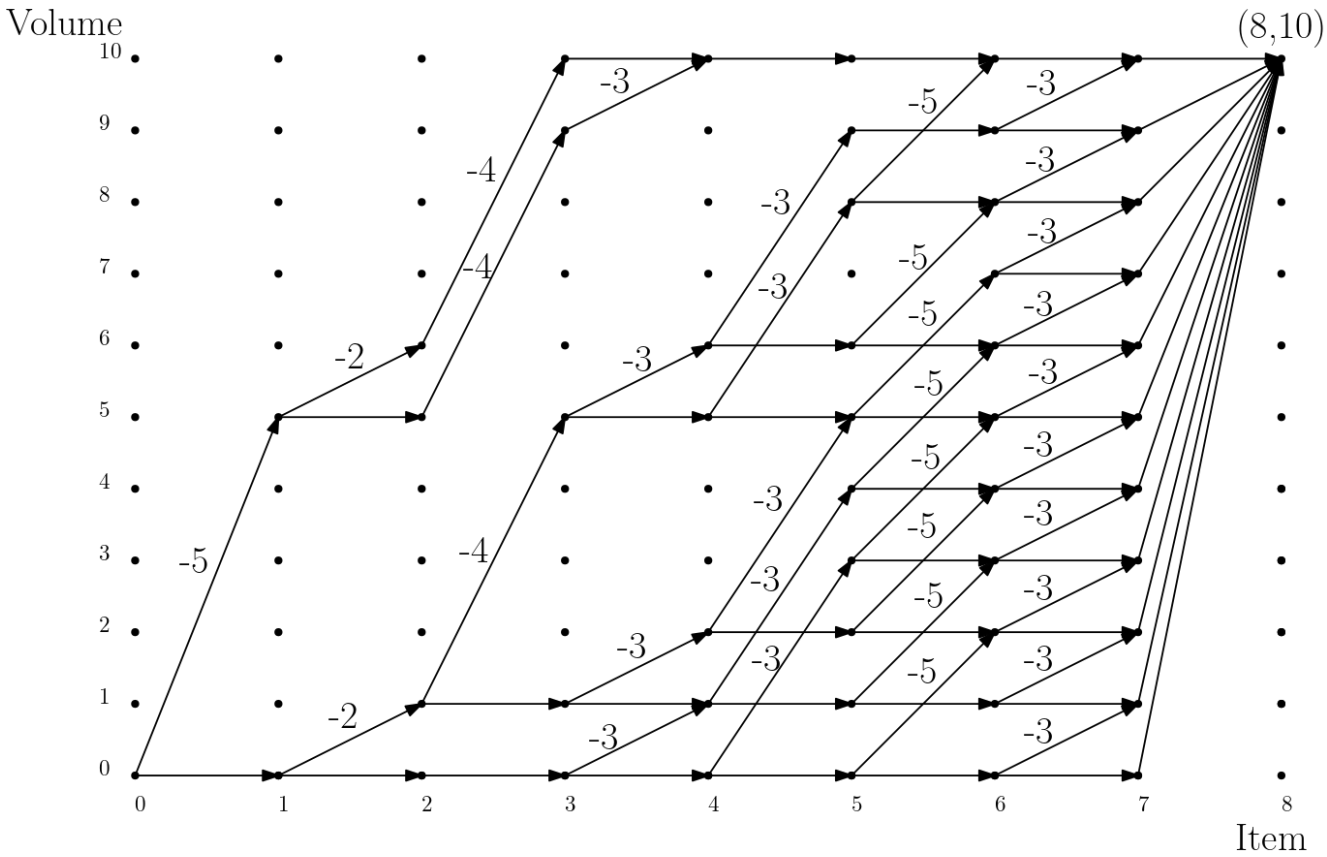
To solve this, we formulate it as a shortest path problem:

We construct a weighted directed graph $D = (V, A)$ as follows:

$$
\begin{array}{rl}
\text{Vertices:} & (i,x) \in V \qquad i = 0,1,...,8 \qquad x = 0,1,...,10 \\
\text{Edges:} & ((i-1,x),(i,x)) \text{ with length } 0 \qquad i = 1,...,7 \\
& ((i-1,x),(i,x+a_i)) \text{ with length } -b_i \qquad i = 1,...,7 \\
& ((7,x),(8,10)) \text{ with length } 0
\end{array}
$$

I.e. the horizontal edges correspond to 'not carrying an object' and therefore get the weight 0.

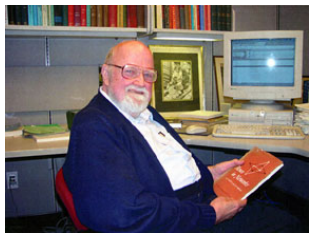
After deleting all irrelevant edges, the following graph is obtained. Shortest $(0, 0)$ – $(8, 10)$ paths in this graph correspond to optimal knapsack packs.



To handle the problem of negative edge weights, we will now look at the next well-known Bellman-Ford algorithm. This method only requires that there is no directed circuit of negative length.

Richard Ernest Bellman (1920–1984) was an American applied mathematician, who introduced dynamic programming in 1953.

Lester Randolph Ford Jr. (1927–2017) was an American mathematician specializing in network flow problems



The method was first described by Shimbel (1955), an later by Bellman (1958), and Moore (1959).
For more details ”On the history of combinatorial optimization” see:
https://homepages.cwi.nl/ lex/files/histco.pdf

Label Setting vs. Label Correcting

- A fundamental difference between the two algorithms of Dijkstra and Bellman-Ford is that in Dijkstra's algorithm the labels of a processed node are not changed a second time.
- As long as there are no negative edge weights, no more optimization can result from the extension.
- But even if there are negative edges (or as in the example all edges are negative), one can calculate a shortest path:

  As long as there is no negative circle we can use a label correcting method, like the algorithm of Bellman-Ford.

Algorithm 4 Bellman-Ford's algorithm

Input $D = (V, A)$, $|V| = n$ and a length function $\ell : A \to \mathbb{Z}$
Input such that there is no directed negative circuit,
Input and let $s \in V$ be a starting node.

```
1:  dist(v) := ∞, pred(v) := undef, ∀v ∈ V
2:  dist(s) := 0
3:  for i ∈ {1, ..., n − 1} do
4:      for (u, v) ∈ A do
5:          if dist(v) > dist(u) + ℓ(u, v) then
6:              dist(v) = dist(u) + ℓ(u, v)
7:              pred(v) := u
8:              for (u, v) ∈ A (check for negative-weight cycles) do
9:                  if dist(v) > dist(u) + ℓ(u, v) then
10:                     Negative Circle Detected
```

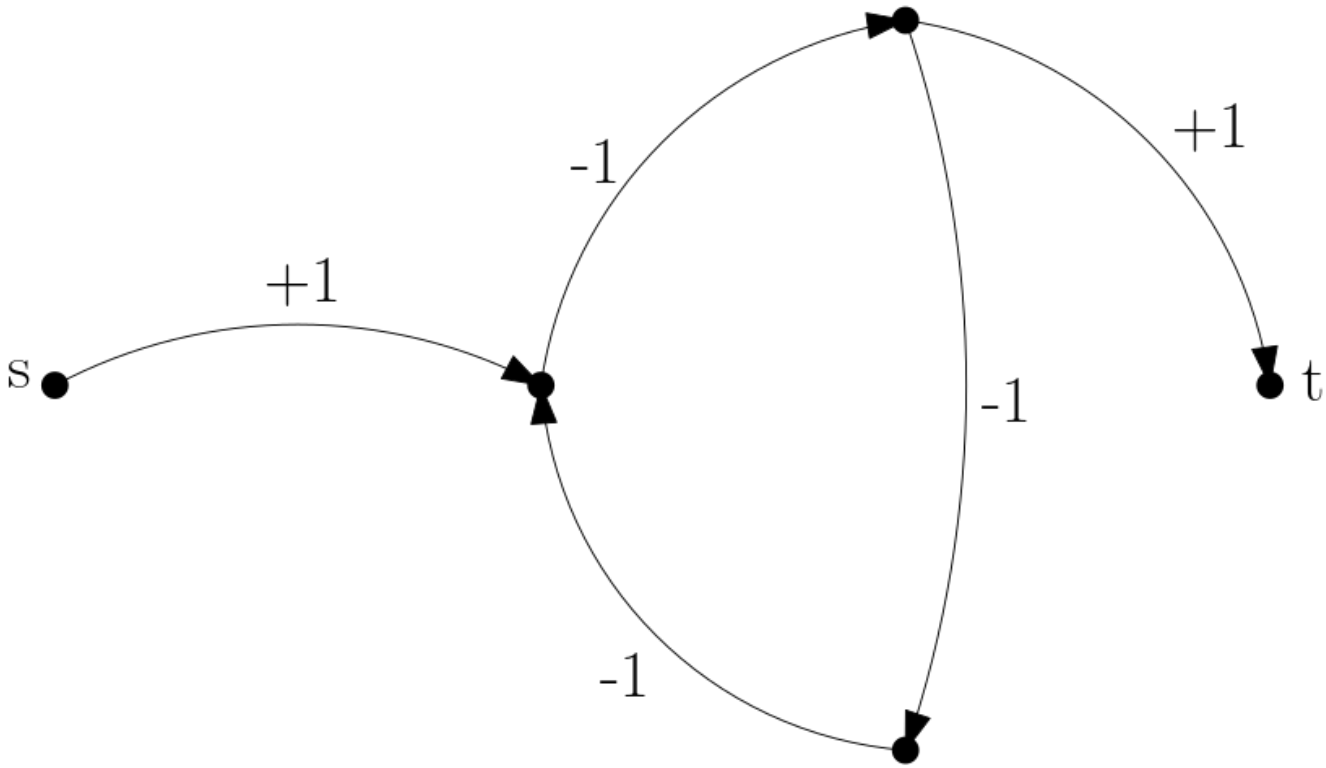
Theorem 3.5

Let $D = (V, A)$ be a connected digraph, $s \in V$ and $\ell : A \to \mathbb{Z}$ a weight function on the edges, such that there is no directed circuit of negative length in D. The Bellman-Ford Algorithm computes the shortest path from s to all other vertices.

Proof.

Exercise. □

In the graph



there is no shortest s-t-path, because the negative circle can be passed infinitely often.

Observations:

- Bellman-Ford detects negative cycles. This means if there is a negative cycle reachable from the source s, then for some edge (u, v), $dist_{n-1}(v) > dist_{n-1}(u) + \ell(u, v)$ (steps 6 and 7).
- If the graph has no negative cycles, then the distance estimates on the last iteration are equal to the true shortest distances. That is, $d_{n-1}(v) = \delta(s, v)$ for all vertices v.
- Backtracking the pred-pointers yields a shortest s-v-path of length $dist(v)$ for all $v \in V$, since there are no negative circuits.

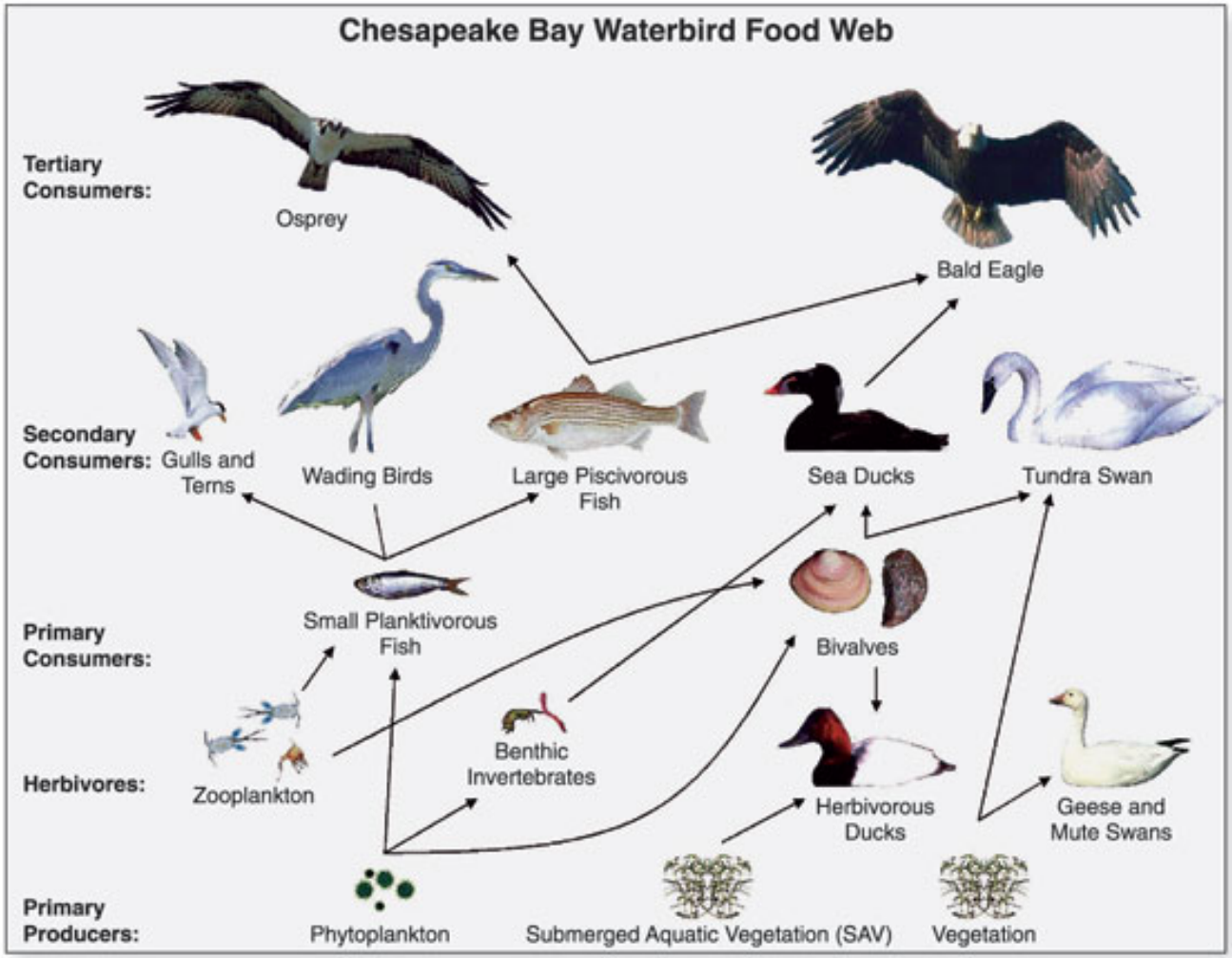


Figure 2: The food chain of waterbirds from Chesapeake Bay. Figure cited from: Matthew C. Perry - US Geological Survey. ”Chapter 14: Changes in Food and Habitats of Waterbirds.” Figure 14.1. Synthesis of U.S. Geological Survey Science for the Chesapeake Bay Ecosystem and Implications for Environmental Management. USGS Circular 1316. (Public Domain)

- A food chain C is an network $C = (V, E)$ of organisms and species containing information about the “who-eats-whom”.
- Thus we have a directed graph C with nodes containing information about the species and edges that give information about their role in the food chain.
- We can identify the producer organisms (they utilize solar or chemical energy to synthesize starch) and the apex predator at the top of the food chain.
- A food chain can be expanded to a *food web* which contains more detail information and is not so much simplified.

- Whenever an edge points from a to b, the species a is eaten by b.
- Hence, for every node v the indegree $deg^-(v)$ is the number of species eaten by v and $deg^+(v)$ is the number of species that eat v.
- We will calculate the indegree and outdegree of every node.

  If $deg^-(v) = 0$ we have a node that is a producer organism.

  If $deg^+(v) = 0$ we have an apex predator.
- It follows that all nodes $v \in B \subset C$ with $deg^-(v) = 0$ build the base of the food chain.
- Moveover, for every node $v \in V$ we can calculate the shortest path $p = [v, ..., n]$ for $n \in B$ which determines the shortest number of links between a trophic consumer and the base.

Some concluding remarks on the history finding shortest paths:

> *It is difficult to trace back the history of the shortest path problem. One can imagine that even in very primitive (even animal) societies, finding short paths (for instance, to food) is essential. Compared with other combinatorial optimization problems, like shortest spanning tree, assignment and transportation, the mathematical research in the shortest path problem started relatively late. (Schrijver, 2012: 155)*