Shortest Paths

Zdeněk Hanzálek
zdenek.hanzalek@cvut.cz

CTU in Prague

March 31, 2020

Table of contents

Life Guidance

A ptáček vece:
"Jednomuť tě učím: Nikdy neželej ztracené věci.
Druhé: Nehledaj dosieci toho, jehož nemuožeš dosieci.
A třetie: Nevěř tomu, co jest nepodobné k vieře."

O slavíku a střelci, Česká Exempla

Problem Statement

a) Shortest Path

- **Instance:** Digraph G, weights $c : E(G) \to \mathbb{R}$, nodes $s, t \in V(G)$.
- **Goal:** Find the shortest $s - t$ path P, i.e. one of minimum weight $c(E(P))$, or decide that t is unreachable from s.

Other problems involving the shortest path:

b) from source node s to every node (**Shortest Path Tree - SPT**)

c) from every node to sink node t

d) between all pairs of nodes (**All Pairs Shortest Path**)

Problem a) is often solved by algorithms for b), c) or d). There is no known algorithm with a better asymptotic time complexity. The algorithm can be terminated when t is reached.

Problem c) can be easily transformed to b) by reversing the edges.

Similar Problems

1) The **longest path** can be transformed to the shortest path while reversing the sign of weights. Thus, we search for the minimum instead of the maximum.

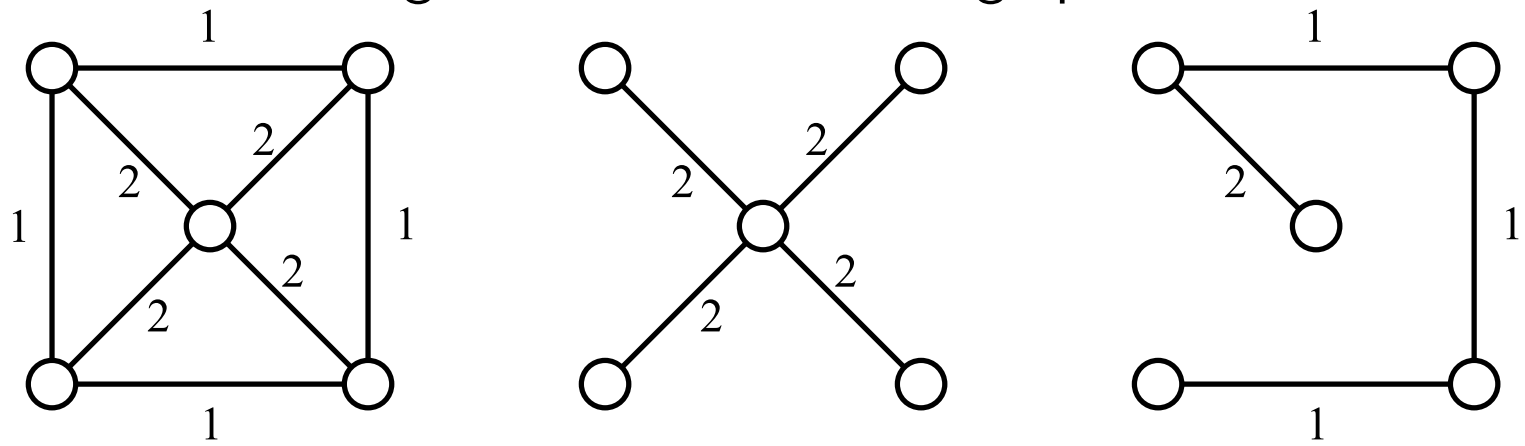
2) When the **nodes are weighted**, or both nodes and edges are weighted, the weight of the path is the sum of the edges and the nodes weights along this path. This can be transformed to the weighted edges case as follows:

- **Replace** every node v from the original graph by the **pair of nodes** v_1 and v_2 and connect them by an edge with weight equal to the weight of node v.
- the edge entering node v now enters v_1
- the edge leaving node v now leaves v_2

Different Problems

1) **Minimum Spanning Tree - MST**

In an undirected graph with the weights associated to arcs, find a spanning tree of minimum weight or decide that the graph is not connected.



The spanning tree in the middle (SPT from central node) has weight 8 and diameter 4 (the longest path between two nodes) while the spanning tree on the right side (MST) has weight 5 and diameter 5.

2) **Steiner Tree**

Given a connected undirected graph G, weights $c : E(G) \to \mathbb{R}^+$, and a set $T \subseteq V(G)$ of terminals, find a Steiner tree for T , i.e. a tree S with $T \subseteq V(S)$ and $E(S) \subseteq E(G)$, such that $c(E(S))$ is minimum.

Negative Weights YES and Negative cycles NO

We consider oriented graphs:

- negative weights are allowed
- negative cycles are not allowed, since the shortest path problem becomes **NP-hard** when the graph contains a negative cycle,

When we transform **undirected graph** to directed graph, then we consider only instances with **nonnegative weights**:

- every undirected edge connecting v and w is transformed to two edges (v,w) and (w,v)
- this transformation of the negative undirected edge results in a negative cycle

Edge Progression (sled) and Path (cesta)

Theorem - existence of the shortest path

If a path from s to t exists in the graph, then the shortest path from s to t exists too.

Note: the **theorem does not hold for edge progression** - in a graph with a negative cycle we can always find an even shorter edge progression which repeatedly goes through this negative cycle.

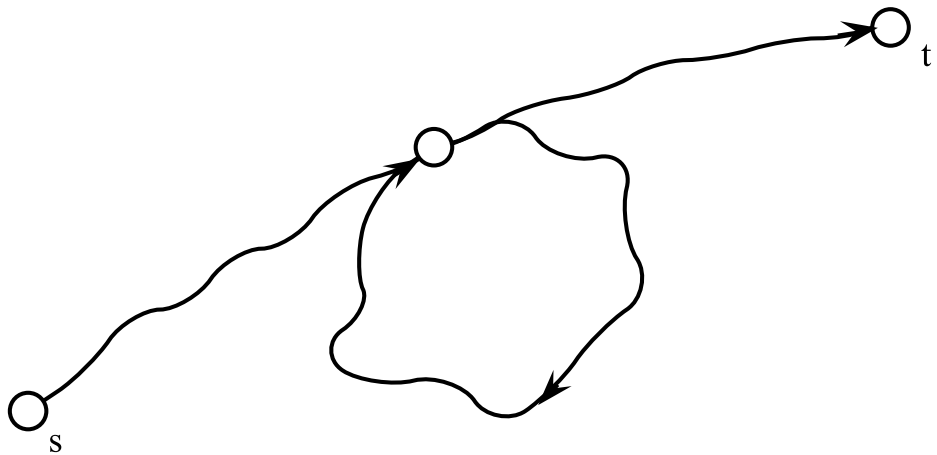
Definitions

- the **Length of the path** P is the **sum of the weights** of its edges (will be denoted simply as $c(E(P))$).
- $l(s, t)$, a distance from s to t, is defined as a length of the **shortest** path from s to t.

Edge Progression and Path

Theorem - shortest edge progression and path

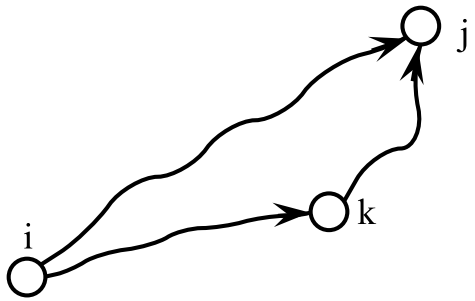
- If there is no **negative weight or zero weight cycle** in the graph, then every shortest edge progression from s to t is the shortest path from s to t.
- If there is no **negative weight cycle** in the graph, then every shortest edge progression from s to t contains the shortest path from s to t and the length of this path is the same.



Triangle Inequality of Shortest Paths

Theorem - triangle inequality

If the graph does not contain a cycle of negative weight then distances between **all triplets of nodes** i, j, k satisfy: $l(i,j) \leq l(i,k) + l(k,j)$.



Corollary: Let $c(i,j)$ **be the weight of edge** from i to j.
Then if the graph does not contain a negative cycle it holds:
$l(i,j) \leq c(i,j)$
$l(i,j) \leq l(i,k) + c(k,j)$
$l(i,j) \leq c(i,k) + l(k,j)$

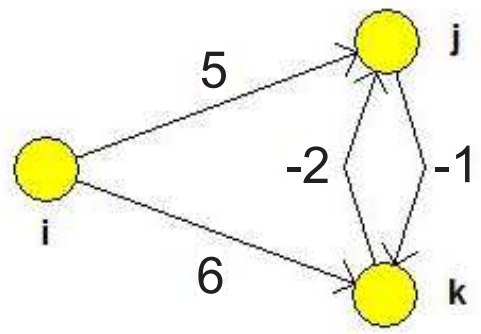
Basic Facts - Negative cycle

The algorithms listed below make use of the following

- (1) Their computational speed is based on the fact that they **do not care about the repetition of nodes along the path** (i.e. they do not distinguish the path from edge progression).
- (2) If the graph contains a negative cycle, **(1) can not be used** because the shortest edge progression does not need to exist (then it is NP-hard to find the shortest path).

Example: a graph with a negative cycle - consequently, the triangular inequality does not hold - the negative cycle of the edge progression is created while joining the two shortest paths.

$$
\begin{array}{rcrcl}
l(i,j) & \leq & l(i,k) & + & l(k,j) \\
6-2 & \leq & (5-1) & + & (-2) \\
4 & \leq & 2 & \ldots & \text{contradiction}
\end{array}
$$



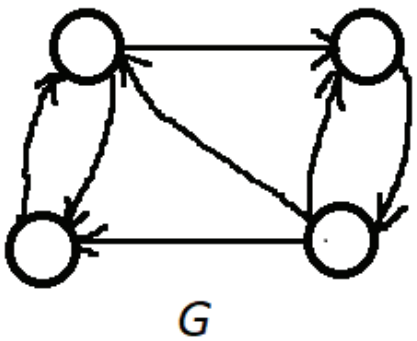
What the graph can not contain, when interested in the longest path?

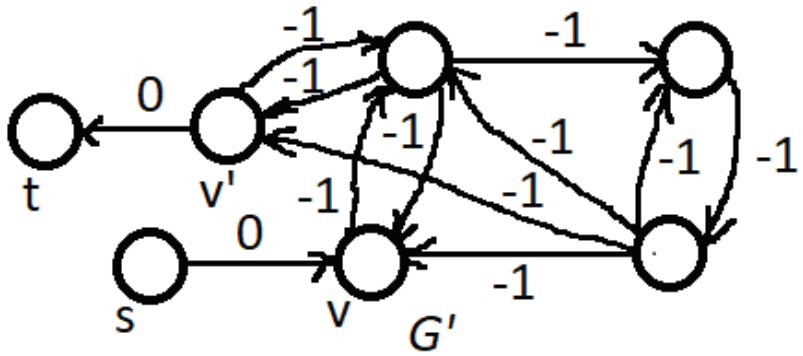
Shortest Path with Negative Cycle is NP-hard

Existence of a Hamiltonian cycle (cycle visiting every node exactly once) in directed graph G can be reduced to the Shortest path problem:

- Create G' as a weighted version of graph G with weights equal to -1.
- Create new $v' \in G'$ as a duplicate of one arbitrary vertex $v \in G'$ including its edges.
- Create source vertex s, destination vertex t and edges (s, v) and (v', t) with weights equal to 0.
- The length of the shortest path from s to t in G' is equal to $-|V(G)|$ if and only if there is a Hamiltonian cycle in graph G.

Note: If graph G has a cycle then graph G' has a negative cycle, otherwise it is trivial.





Bellman's Principle of Optimality

Theorem - Bellman's Principle of Optimality

Suppose we have digraph G, weights $c : E(G) \to \mathbb{R}$, no negative cycles. Let $k \in \mathbb{N}$, and let s and w be two vertices. Let P^k be a shortest one among all s-w-paths with at most k edges, and let $e = (v, w)$ be its final edge. Then $P^{k-1}[s, v]$ (i.e. P^k without the edge e) is a shortest one among all s-v-paths with at most $k - 1$ edges.

Proof by contradiction:

We will study two cases:

1) Suppose Q_1 is an s-v-path shorter than $P^{k-1}[s, v]$, $|E(Q_1)| \leq k - 1$ and Q_1 does not contain w, then:

$c(E(Q_1)) < c(E(P^{k-1}[s, v]))$ ... add edge e

$c(E(Q_1)) + c(e) < c(E(P^{k-1}[s, v])) + c(e) = c(E(P^k))$

This is a contradiction, since s-w-path consisting of Q_1 and e can't be shorter than P^k.

Bellman's Principle of Optimality - proof cont.

2) Suppose Q_2 is an s-v-path shorter than $P^{k-1}[s,v]$, $|E(Q_2)| \leq k-1$ and Q_2 **does** contain w, then:

$c(E(Q_2)) < c(E(P^{k-1}[s,v]))$ / split Q_2 in two parts

$c(E(Q_2[s,w])) + c(E(Q_2[w,v])) < c(E(P^{k-1}[s,v]))$ / add edge e

$$c(E(Q_2[s,w])) + \underbrace{c(E(Q_2[w,v])) + c(e)}_{\geq 0} < \underbrace{c(E(P^{k-1}[s,v])) + c(e)}_{c(E(P^k))}$$

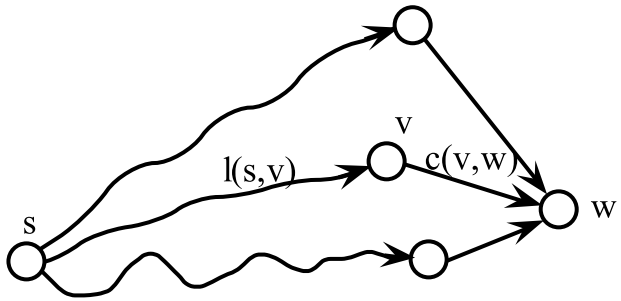
... because $Q_2[w,v]$ and e is a non-negative cycle.

$c(E(Q_2[s,w])) < c(E(P^k))$

This is a contradiction, since $Q_2[s,w]$ can't be shorter than P^k.

Bellman's equation and Generalization on Path Segments

Bellman's equation: $l(s,w) = \min_{v \neq w} \{l(s,v) + c(v,w)\}$ holds if the graph does not contain a negative cycle



A straight line indicates a single edge; a wavy line indicates a shortest path between the two vertices it connects (other nodes on these paths are not shown).

Generalization - the shortest path consists of the shortest paths

Suppose we have a graph without negative cycles. If the shortest path from s to w contains node v, the segment of the path from s to v is the shortest s-v-path and similarly the segment of the path from v to w is the shortest v-w-path and $l(s,w) = l(s,v) + l(v,w)$.

Dijkstra Algorithm [1959] - Nonnegative Weights

Input: digraph G, weights $c : E(G) \to \mathbb{R}_0^+$ and node $s \in V(G)$.

Output: Vectors l and p. For $v \in V(G)$, $l(v)$ is the length of the shortest path from s and $p(v)$ is the previous node in the path. If v is unreachable from s, $l(v) = \infty$ and $p(v)$ is undefined.

$l(s) := 0$; $l(v) := \infty$ for $v \neq s$; $R := \emptyset$;

while $R \neq V(G)$ **do**

  Find $v \in V(G) \setminus R$ such that $l(v) = \min_{i \in V(G) \setminus R} l(i)$;

  $R := R \cup \{v\}$ // in the first run we add vertex s

  // further calculate non-permanent value of $l(w)$ for every node on border of R

  for $w \in V(G) \setminus R$ such that $(v, w) \in E(G)$ **do**

    if $l(w) > l(v) + c(v, w)$ **then**

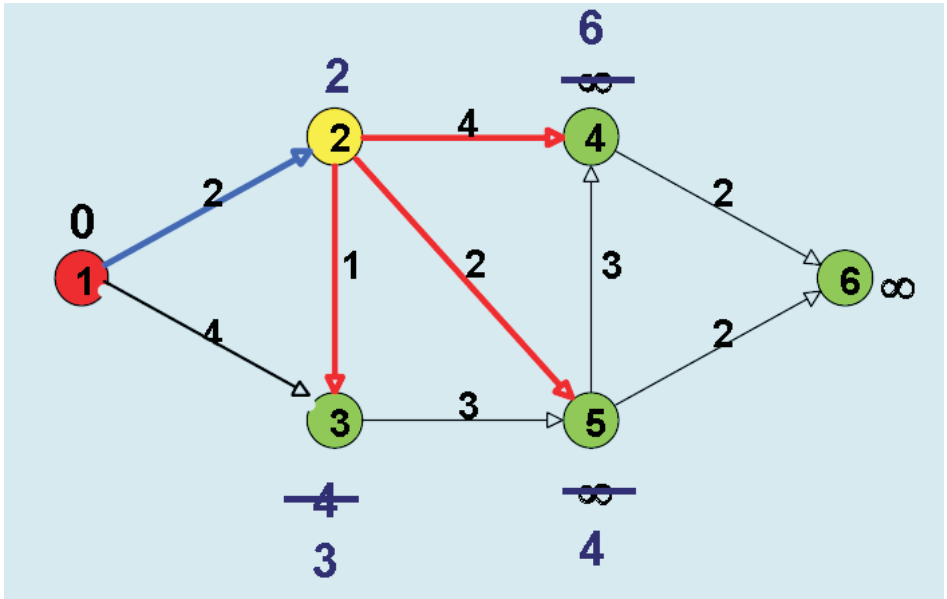
      $l(w) := l(v) + c(v, w)$; $p(w) := v$;

    end

  end

end

Iteration of Dijkstra Algorithm



Homework: What are the differences between Dijkstra algorithm for SPT and Prim algorithm for **minimum spanning tree - MST**?
Find the smallest example with a different SPT and MST.

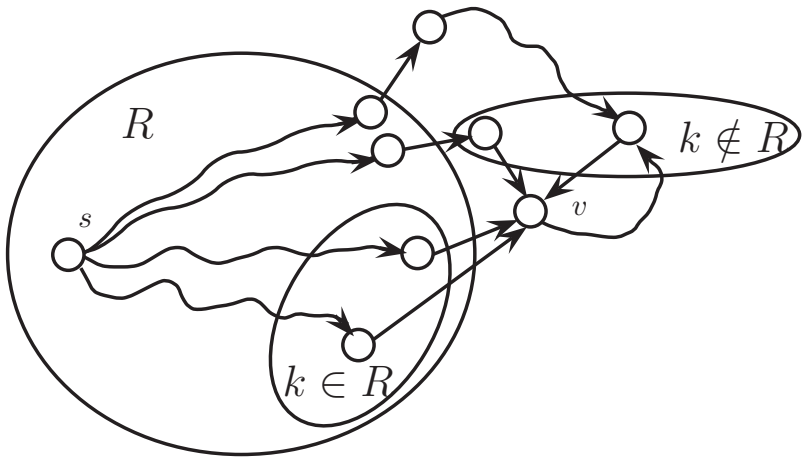
Correctness of Dijkstra Algorithm

Proof by induction:
For $|R| = 1$, the value of $l(s) = 0$ is optimal.
Inductive step: assuming that all nodes in R have optimal value of the shortest path we will prove the $l(v)$ value is **optimal/permanent** when we add the **best candidate** v to set R (the first line in the while loop).
The proof is completed on the next page.

Correctness of Dijkstra Algorithm - continuation

Proof of the inductive step: we show that Bellman's equation $l(s, v) = \min_{k \neq v} \{l(s, k) + c(k, v)\}$ holds:

- for **predecessors** $k \in R$, since $l(s, k)$ is optimal and Bellman's equation for these nodes was calculated in the internal loop (all $e \in E^+(R)$ were tested)
- for **predecessors** $k \notin R$, since the value of $l(v)$ can not be improved by the path going through $k \notin R$ because $l(v) \leq l(k \notin R)$ and the $c(k, v)$ are nonnegative.
  The value of $l(k \notin R)$ may get smaller in the next iterations (due to the path through $i \in V(G) \setminus R$), but while adding non-negative lengths it cannot get smaller than $l(v) = \min_{i \in V(G) \setminus R} l(i)$.



Properties of Dijkstra Algorithm

Dijkstra is so-called **label setting algorithm**, since label $l(v)$ becomes permanent (optimal) at each iteration.
In contrast, **label-correcting algorithms** (e.g. Bellman-Ford or Floyd algorithm) consider all labels as temporary until the final step, when they all become permanent.

Time complexity of Dijkstra is $O(n^2)$, or $O(m + nlogn)$ using priority queue

There are algorithms with linear time complexity for **specific graphs**:

- Planar Graphs - Henzinger [1997]
- Undirected graphs with integer non-negative weights - Thorup [1999]
- DAGs - later in this lecture

Exploitation of Additional Information

If we are interested in finding the shortest path from s to **just one node** t, Dijkstra algorithm can be terminated when we include t to R.

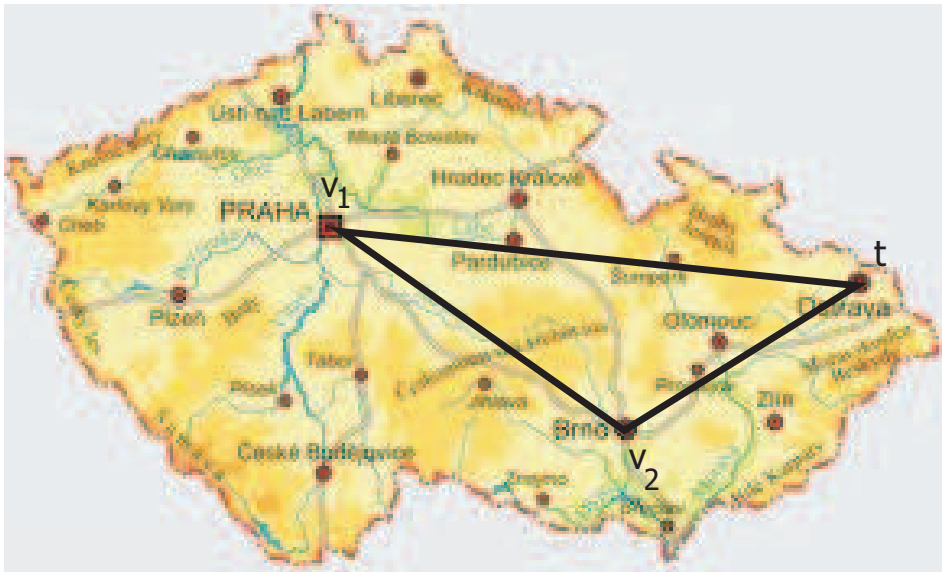
To accelerate the computation time we can add **additional information to Dijkstra algorithm** in order to perform **informed search**. This is idea of A* algorithm which is generalization of Dijkstra that cuts down on the size of the subgraph that must be explored.

- At the algorithm start, we set $h(v, t)$ for every $v \in V(G)$ such that it represents the **lower bound on distance from** v **to** t.
- For arbitrary nodes $v_1, v_2 \in V(G)$ inequality $c(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$ **must be valid**.
- we modify Dijkstra algorithm while choosing $v \in V(G) \setminus R$ such that $l(v) = \min_{i \in V(G) \setminus R} \{\boldsymbol{l(i) + h(i, t)}\}$

This algorithm is faster, since vertices in wrong direction (with a high value of h) are newer chosen. Unfortunately, in the extreme case, this algorithm has to go through all vertices as well.

Exploitation of Additional Information - Example

- Suppose the nodes are places in the two dimensional plane with coordinates $[x_v, y_v]$ and arc lengths equal to Euclidean distances between the points. The Euclidean distance from v to t is equal to $[(x_t - x_v)^2 + (y_t - y_v)^2]^{1/2}$, and it can be used as lower bound $h(v, t)$.
- Euclidean distances satisfy triangular inequality, i.e. $h(v_1, v_2) + h(v_2, t) \geq h(v_1, t)$. Therefore $c(v_1, v_2) \geq h(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$



Bidirectional Dijkstra Algorithm - Idea

This algorithm finds the shortest path from s to t.

Basic idea:

- We search "simultaneously"
  - forward from source with vector of distances l^f while using the original graph and
  - backward from target with vector of distances l^b while using the reverse graph (i.e., reversed orientation of edges)
- Stop when the search spaces "meet".
- This reduces the search space approximately by a factor of 2 only.
- On the other hand, Bidirectional Dijkstra is an **important inspiration** to many sophisticated algorithms (e.g., Contraction Hierarchies)

Bidirectional Dijkstra Algorithm - Pseudocode

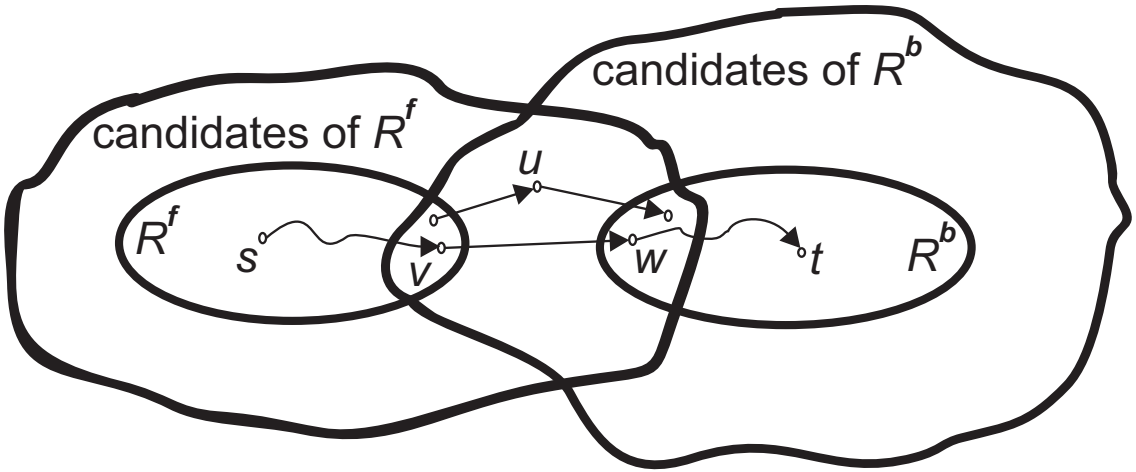
Bidirectional Dijkstra algorithm:

- initialize $l^f(s) = 0$, $l^b(t) = 0$, $D = \infty$
- maintain two sets of permanent vertices R^f and R^b
- maintain joint priority queue of candidates PQ, where each item in the PQ knows whether it belongs to forward search or backward search
- in each step, perform one iteration of Dijkstra
  - **if** top item in the PQ belongs to forward search
    then perform the forward search from s
    else perform the backward search from t
- when **v becomes permanent in forward search** (i.e., $v \in R^f$) and there is an edge (v, w) such that **w is permanent in backward search** (i.e., $w \in R^b$) then **update**
  $D = \min\left\{l^f(v) + c(v, w) + l^b(w), D\right\}$ for every such edge (v, w)
- similar update when a vertex becomes permanent in backward search
- stop when $top^f + top^b \geq D$ where top^f and top^b stand for top items in PQ belonging to forward and backward search respectively

Bidirectional Dijkstra Algorithm - Correctness

Proof by contradiction:

- Suppose there exists an $s - t$ path Q_1 with length less than D going through vertex $u \notin R^f \cup R^b$.
  - Contradiction: Q_1 cannot exist since $l^f(u) \geq top^f$ and $l^b(u) \geq top^b$ and due to the stopping criterion $l^f(u) + l^b(u) \geq top^f + top^b \geq D$
- Suppose there exists an $s - t$ path Q_2 with length less than D going through edge (v, w) such that $v \in R^f$ and $w \in R^b$
  - Contradiction: Q_2 cannot exist since the edge (v, w) was used in the update when later of vertices v and w became permanent and therefore $c(E(Q_2)) \geq D$



Example SPT.automaton.water: Measurement of Water Level [2]

You are on the bank of the lake and you have one 3-liter bottle and one 5-liter bottle. Both bottles are empty and your task is to have exactly 4 liters in the bigger bottle. You have no other equipment to measure the water level.

a) Represent the problem by the graph.

b) Set-up suitable weights and formulate the shortest path problem to find

- solution with the minimum number of manipulations,
- solution with the minimum amount of manipulated water,
- solution with the minimum amount of water poured back in the lake.

Homework c) During some manipulations you have to be very careful - for example when one bottle is filled completely but the other one does not become empty. Find the solution which minimizes the number of such manipulations.

Homework d) Is it possible to have 5 liters while using 4-liter and 6-liter bottles?

Example SPT.automaton.bridge: Bridge and Torch Problem

Homework - Formulate the shortest path problem:
Four people come to a river in the night. There is a narrow bridge, but it can only hold two people at a time. They have one torch and, because it's night, the torch has to be used when crossing the bridge. One person can cross the bridge in 1 minute, one person in 2 minutes, one person in 5 minutes, and another person in 9 minutes. When two people cross the bridge together, they must move at the slower person's pace. The question is, can they all get across the bridge in 16 minutes or less?

Example SPT.dioid.reliability: Maximum Reliability Path Problem [1]

In the communication network we associate a reliability $p(i,j)$ with every arc from i to j. We assume the failures of links to be unrelated. The reliability of a directed path Q from s to t is the product of the reliability of the arcs in the path. Find the most reliable connection from s to t.

- a) Show that we can reduce the maximum reliability path problem to a shortest path problem.
- b) Suppose you are not allowed to make reduction. Specify $O(n^2)$ algorithm for solving the maximum reliability path problem.
- c) Will your algorithms in parts a) and b) work if some of the $p(i,j)$ coefficients are strictly greater than 1?

Bellman-Ford Algorithm [1958] (Moore [1959])

Input: directed graph G without a negative cycle
weights $c : E(G) \to \mathbb{R}$ and node $s \in V(G)$.

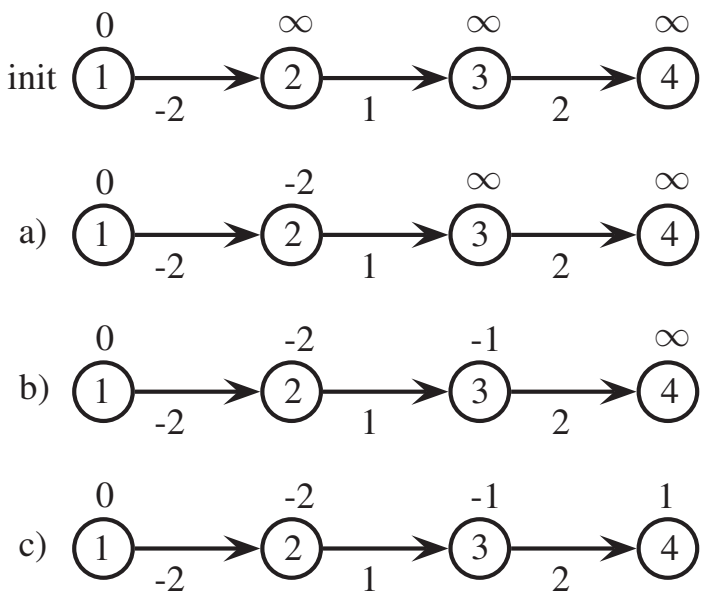
Output: vectors l and p. For all $v \in V(G)$, $l(v)$ is the length of the shortest path from s and $p(v)$ is the last but one node. If v is not reachable from s, then $l(v) = \infty$ and $p(v)$ is undefined.

$l(s) := 0$; $l(v) := \infty$ for $v \neq s$;
for $i := 1$ **to** $n - 1$ **do**
  for for every edge of graph $(v, w) \in E(G)$ **do**
    if $l(w) > l(v) + c(v, w)$ **then**
      $l(w) := l(v) + c(v, w)$; $p(w) := v$;
    end
  end
end

If an iteration of the main loop terminates without making any changes, the algorithm can be terminated, as subsequent iterations will not make changes any more. The worst case of the algorithm remains unchanged.

Bellman-Ford Algorithm - Example

This example illustrates iterations of the Bellman-Ford algorithm finding shortest paths tree from node 1.



Assuming the edges to be processed in the internal loop

- in the worst order, from right to left when only step a) is executed in the first iteration of the external loop, it requires 3 iterations of the external loop to obtain SPT
- in the best order, from left to right when steps a)b)c) are executed in the first iteration of the external loop, it requires only 1 iteration of the external loop to obtain SPT

Correctness of Bellman-Ford algorithm

We will base our reasoning on the following theorem:

Theorem: k-th iteration of Bellman-Ford algorithm

Let

- $l^k(w)$ is the $l(w)$ label after k iterations of the external loop
- P^k is the shortest s-w-path with at most k edges and let $e = (v, w)$ be the last edge of this path
- $c(E(P^k))$ is the length of path P^k

Then $l^k(w) \leq c(E(P^k))$.

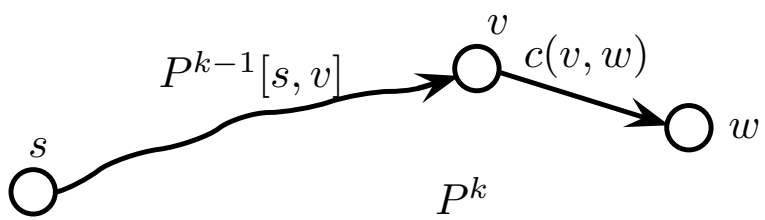
Note: the $l^k(w) \leq c(E(P^k))$ is inequality, since the $l^k(w)$ is the length of the path which might go over more that k edges (see example with four nodes in the chain), but the P^k path is defined to have at most k edges.

Theorem Proof

Prove of the theorem by induction:

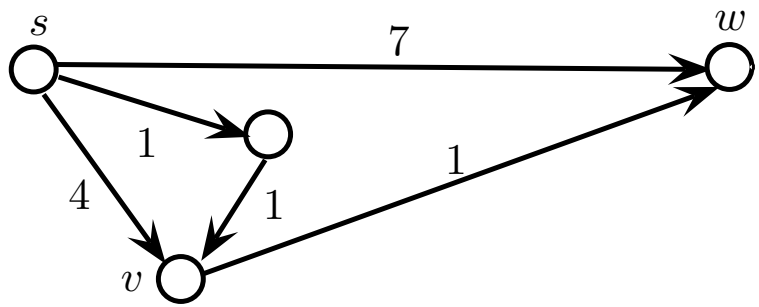
- For $k = 1$ it holds, because $l^1(w) \leq c(s, w)$
- By the **induction hypothesis** ^{IH} we have $l^{k-1}(v) \leq c(E(P^{k-1}[s, v]))$ after $k - 1$ iterations of external loop.
- In the k-th iteration of the external loop of the **algorithm** ^{ALG}, **the edge** (v, w) **is examined by the internal loop**, so the internal loop inequality holds: $l^k(w) \leq l^{k-1}(v) + c(v, w)$.
- By the **theorem assumption** (i.e., P^k is the shortest path and contains edge $e(v, w)$) and **Bellman Principle of Optimality** ^{BPO} it holds: $c(E(P^{k-1}[s, v])) + c(v, w) = c(E(P^k))$

Since in the k-th iteration edge $e(v, w)$ is also examined we conclude:
$l^k(w) \overset{ALG}{\leq} l^{k-1}(v) + c(v, w) \overset{IH}{\leq} c(E(P^{k-1}[s, v])) + c(v, w) \overset{BPO}{=} c(E(P^k))$.



Correctness of Bellman-Ford algorithm - Proof

Re-worded proof of the theorem:
While assuming ^{IH} that we had $l^k(v)$ better than or equal to the shortest s-v-path with at most $k-1$ edges in iteration $k-1$, in iteration k of the algorithm ^{ALG} we will create $l^k(w)$ better than or equal to the shortest s-w-path with at most k edges since the shortest path consists of the shortest paths ^{BPO}.



Correctness of the Bellman-Ford algorithm:
In iteration $k = n-1$ we will obtain $l^k(w) = c(E(P^k))$

- because the above theorem implies $l^k(w) \leq c(E(P^k))$
- and $l^k(w) \geq c(E(P^k))$ because no path has more than $n-1$ edges and no path is shorter than the shortest one.

Time Complexity of Bellman-Ford Algorithm and Detection of Negative cycles

Best known algorithm for **SPT** without negative cycles.

- Time complexity is $O(nm)$.
- Note that every path consists at most of $n-1$ edges and there exists an edge incident to every reachable node t.

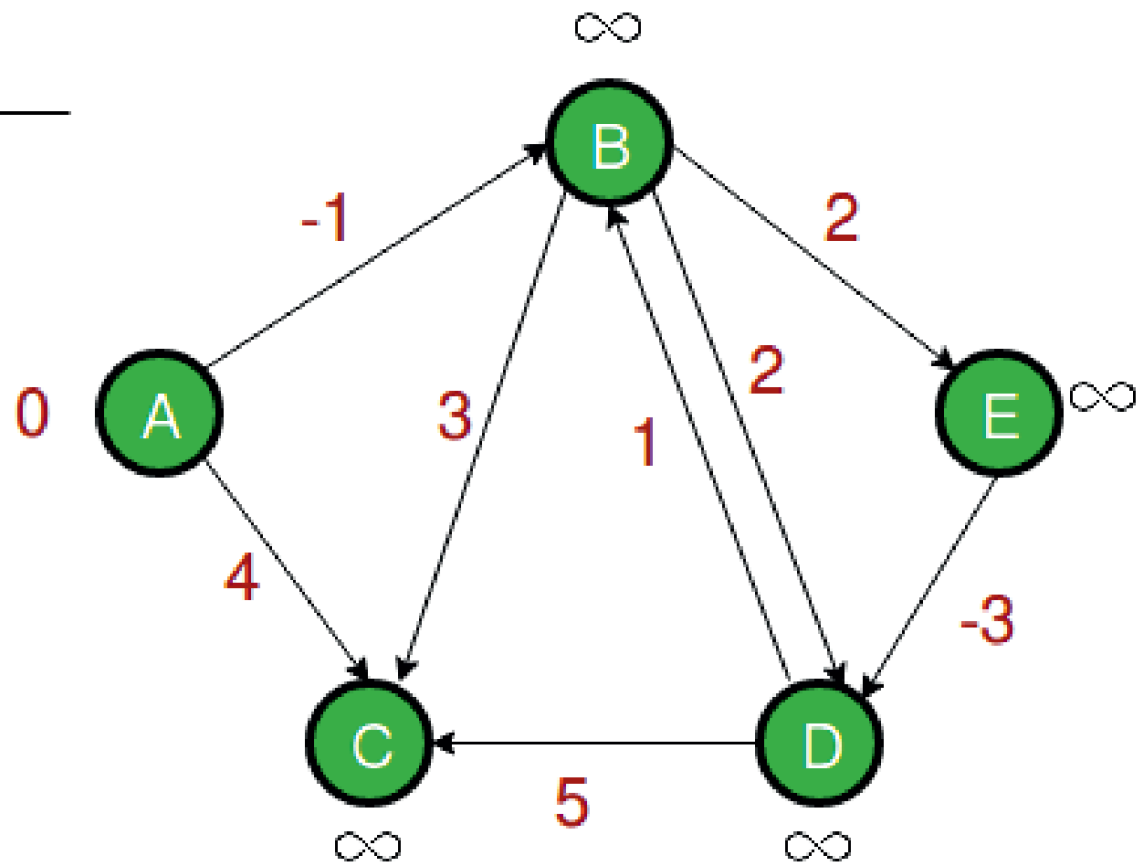
The Bellman-Ford algorithm **can detect negative cycle** that is reachable from vertex s while checking triangular inequality for resulting l. There may be negative cycles not reachable from s, in such case we add node s' and connect it to all nodes v with $c(s', v) = 0$.

for for every edge of graph $(v, t) \in E(G)$ **do**
  if $l(t) > l(v) + c(v, t)$ **then**
    error "Graph contains a negative-weight cycle"
  end
end

However there are better methods for negative cycle detection [Cherkassky&Goldberg 1999].

Example: Iterations of Bellman-Ford algorithm

A	B	C	D	E
0	∞	∞	∞	∞



(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

Dynamic Programming Perspective

Observation on SPT algorithms:

- solve a collection of subproblems, e.g. $l(w) = min_{v \neq w}\{l(v) + c(v, w)\}$
- start with simplest one, i.e. $l(s) = 0$
- proceed with larger subproblems along the topological order

This is a general technique called Dynamic Programming, which requires two key attributes: **optimal substructure** and **overlapping subproblems**.

Dynamic Programming Perspective

Optimal substructure

- The solution to a given optimization problem can be obtained by the **combination of optimal solutions to its subproblems**.
- Such optimal substructures are usually described by means of **recurrent formula** (e.g., Bellman equation).

Overlapping subproblems

- Computed solutions to **subproblems are stored** so that these don't have to recomputed.
- Dynamic Programming is **not useful when there are no overlapping subproblems** because there is no point to store the solutions if they are not needed again.
- One can observe so called **diamonds** in the in the solution space, in contrast to the solution space of Branch and Bound algorithm, which can be represented by the tree.

Example SPT.truck.negative: Truck Journey [2]

Let us assume a truck and n European cities with trailers.

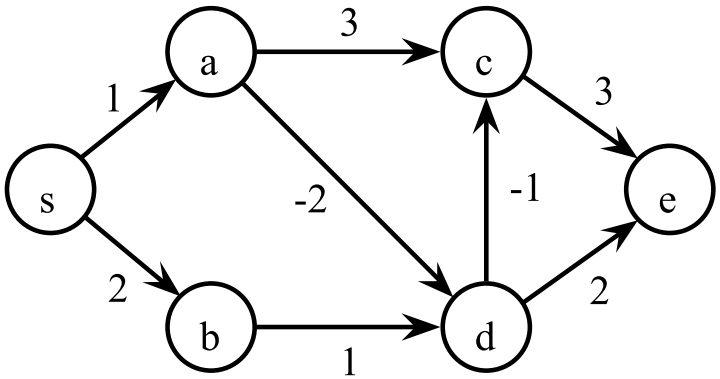
- For each couple of cities (i, j), we know $c(i, j)$, the cost of the truck transport form city i to city j.
- For some couple of cities (i, j) there are (infinitely many) trailers to be transported form city i to city j and the revenue for one trailer is $d(i, j)$.

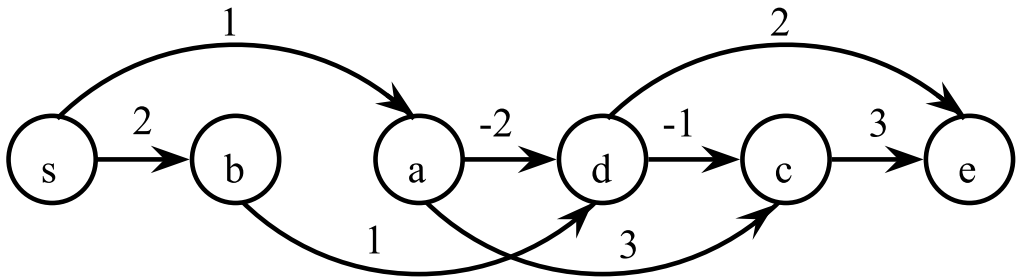
Our task is to find the track journey from city s to city t and to maximize the profit regardless of the time.

- Represent a structure of the example by a directed graph.
- Formulate an optimization problem when each city can be entered at most once. What is the complexity of the problem?
- How the problem changes when each city can be entered several times?

Shortest Paths in Directed Acyclic Graphs (DAGs)

Definition: A **topological order** of G is an order of the vertices $V(G) = v_1, \ldots, v_n$ such that for each edge $(v_i, v_j) \in E(G)$ we have $i < j$.
Proposition: A directed graph has a topological order if and only if it is acyclic (see the proof in [3]).
Consequence: DAG vertices can be arranged on a line so that all edges go from left to right.





Observation: shortest path from s to v_i cannot use any node from $v_{i+1}, \ldots, v_n$, therefore we can find the shortest paths in topological order

Algorithm for DAGs

May be seen as simplified version of Bellman-Ford algorithm.

Input: directed acyclic graph G with topologically ordered vertices $v_1, \ldots, v_n$, weights $c : E(G) \to \mathbb{R}$.

Output: vectors l and p. For all $i = 1 \ldots n$, $l(v_i)$ is the length of the shortest path from v_1 and $p(v_i)$ is the last but one node. If v_i is not reachable from v_1, then $l(v_i) = \infty$ and $p(v_i)$ is undefined.

$l(v_1) := 0$; $l(v_i) := \infty$ for $i = 2 \ldots n$;
for $i := 2$ **to** n **do**
  for for every edge of graph $(v_j, v_i) \in E(G)$ **do**
    if $l(v_i) > l(v_j) + c(v_j, w)$ **then**
      $l(v_i) := l(v_j) + c(v_j, v_i)$; $p(v_i) := v_j$;
    end
  end
end

Correctness: induction on i and observation on previous slide.

Time complexity $O(|V| + |E|)$

Example SPT.dioid.negative: Investment Opportunities [1]

Mr. Dow Jones, 50 years old, wishes to place his Individual Retirement Account funds in various investment opportunities so that at the age of 65 years, when he withdraws the funds, he has accrued maximum possible amount of money. Assume that Mr. Jones knows the investment alternatives for the next 15 years - each opportunity has starting year k, maturity period p (in years) and appreciation a it offers for the maturity period. How would you formulate this investment problem as a shortest path problem, assuming that at any point in time, Mr. Jones invests all his funds in a single investment alternative.

opportunity	a	b	c	d	e	f	g	h	i	j
starting year	0	1	3	2	5	6	7	8	11	13
maturity period	4	5	6	5	4	5	6	5	4	2
appreciation	3.9%	4.7%	6.2%	4.2%	3.8%	4.1%	5.2%	5.8%	4.1%	3.2%

Floyd Algorithm [1962] (Warshall [1962])

Input: a digraph G free of negative cycles and weights $c : E(G) \to \mathbb{R}$.
Output: matrices l and p, where l_{ij} is the length of the shortest path from i to j, p_{ij} is the last but one node on such a path (if it exists).

$l_{ij} := c((i,j))$ for all $(i,j) \in E(G)$;
$l_{ij} := \infty$ for all $(i,j) \notin E(G)$ where $i \neq j$;
$l_{ii} := 0$ for all i;
$p_{ij} := i$ for all (i,j);
for $k := 1$ **to** n **do** `// for all k check if` l_{ij} improves
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** n **do**
      if $l_{ij} > l_{ik} + l_{kj}$ **then**
        $l_{ij} := l_{ik} + l_{kj}$; $p_{ij} := p_{kj}$;
      end
    end
  end
end

Floyd Algorithm

- initialize matrix I^0 by the weights of the edges
- computes the sequence of matrices $I^0, I^1, I^2, \cdots, I^k, \cdots, I^n$ where:

  I^k_{ij} is the length of the shortest path from i to j such that with the exception of i, j it goes only through nodes from the set $\{1, 2, \cdots k\}$
- one can easily compute matrix I^k from matrix I^{k-1}:

$$I^k_{ij} = min\{I^{k-1}_{ij}, I^{k-1}_{ik} + I^{k-1}_{kj}\}$$

Floyd Algorithm - Complexity and Properties

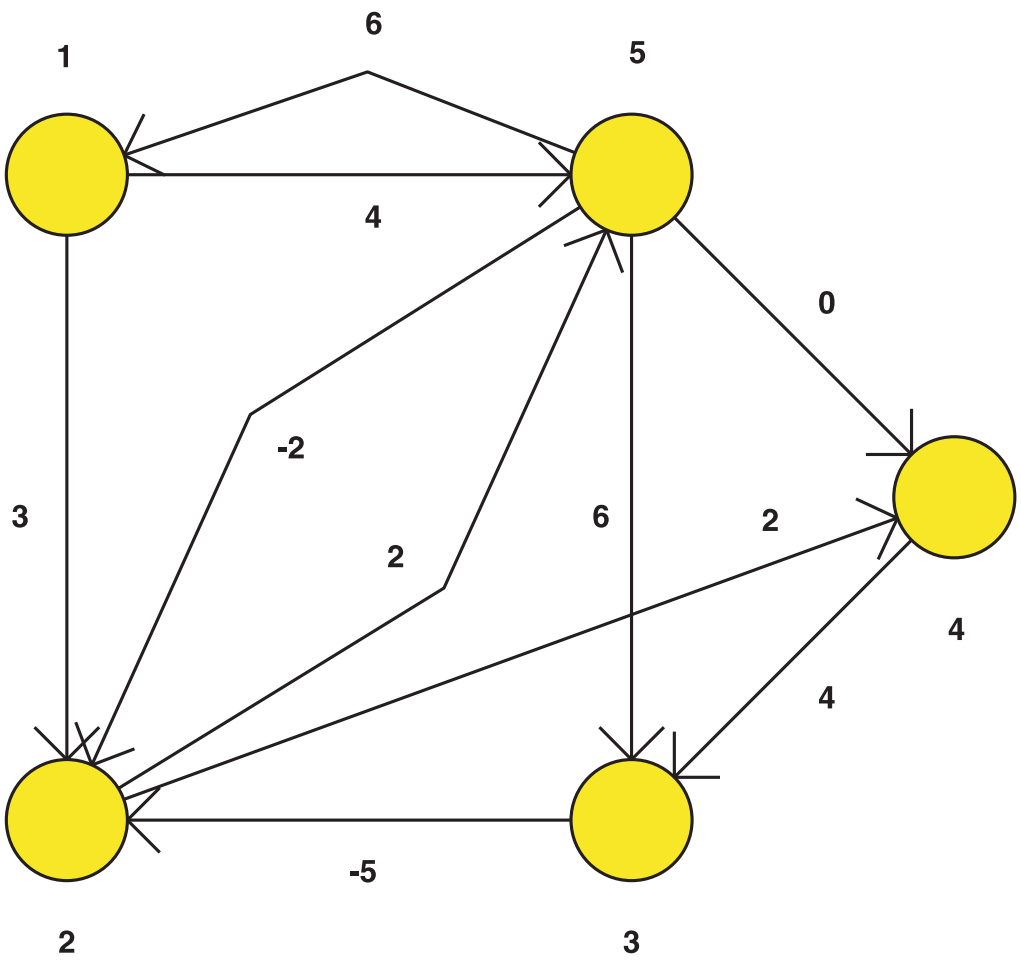
The best known algorithm for **All Pairs Shortest Path** problem without negative cycles.

- Time complexity $O(n^3)$.
- A graph contains a **negative cycle** iff (i.e. if and only if) there exists i such that $l_{ii} < 0$.
- By a small modification of the algorithm, where l_{ii}^0 is set to ∞, one can find the nonnegative **minimal weight cycle** - see the diagonal in the next example. Then the value of l_{ii}^n should not be interpreted as a distance.

Johnson's Algorithm is better suited for the sparse graphs

- uses Dijkstra and Bellman-Ford
- complexity: $O(|V| \cdot |E| \cdot log|V|)$ (in the simplest case)

Floyd Algorithm - Example



$$l^0 = \begin{pmatrix} \infty & 3 & \infty & \infty & 4 \\ \infty & \infty & \infty & 2 & 2 \\ \infty & -5 & \infty & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty \\ 6 & -2 & 6 & 0 & \infty \end{pmatrix}$$

$$l = \begin{pmatrix} 10 & 2 & 8 & 4 & 4 \\ 8 & 0 & 6 & 2 & 2 \\ 3 & -5 & 1 & -3 & -3 \\ 7 & -1 & 4 & 1 & 1 \\ 6 & -2 & 4 & 0 & 0 \end{pmatrix}$$

$$p = \begin{pmatrix} 5 & 5 & 4 & 5 & 1 \\ 5 & 5 & 4 & 2 & 2 \\ 5 & 3 & 4 & 2 & 2 \\ 5 & 3 & 4 & 2 & 2 \\ 5 & 5 & 4 & 5 & 2 \end{pmatrix}$$

Example SPT.matrix.fire: Location of Fire Station [2]

Formulate a shortest path problem:
Consider a road system in the city.
a) We are looking for the best location of the fire station while minimizing its distance form the most distant place.
Homework b) How the problem changes (gets more difficult) when we are looking for the best location of two fire stations?
Homework c) How the problem changes (gets more difficult) when the maximum allowed distance is given and we are looking for the minimum number of stations.

Example SPT.matrix.warehouse: Warehouse Location

Formulate a shortest path problem:
Consider a road system in the region. We are looking for the best location of the warehouse which supplies n customers consuming $q_1, \ldots, q_n$ units of goods per week. There is a suitable place for warehouse nearby each of the customers. Each customer is served by a separate car and the transport cost is a product (i.e. multiplication) of the distance and transported volume. Objective is to minimize the weekly transport costs.

Shortest Path - Conclusion

- An easy optimization problem with a lot of practical applications
  - OSPF (Open Shortest Path First) is a widely used protocol for Internet routing that uses Dijkstra algorithm.
  - RIP (Routing Information Protocol) is another routing protocol based on the Bellman-Ford algorithm.
  - looking for shortest/cheapest/fastest route in the map
- Basic routine for many optimization problems
  - scheduling with dependencies
  - ...

References


Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
Network Flows: Theory, Algorithms, and Applications.
Prentice Hall, 1993.


Jiří Demel.
Grafy a jejich aplikace.
Academia, 2002.


B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, fourth edition, 2008.