Algorithms and Data Structures

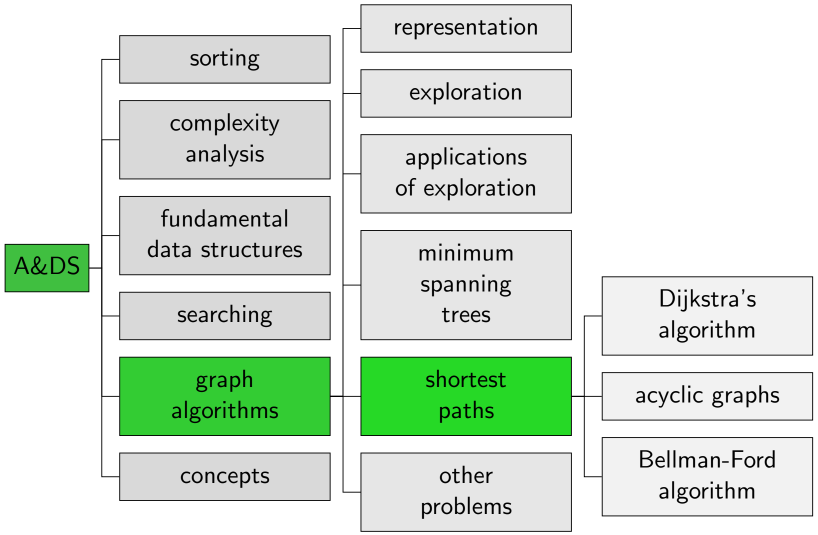
C5. Shortest Paths: Foundations

Gabriele Röger and Patrick Schnider

University of Basel

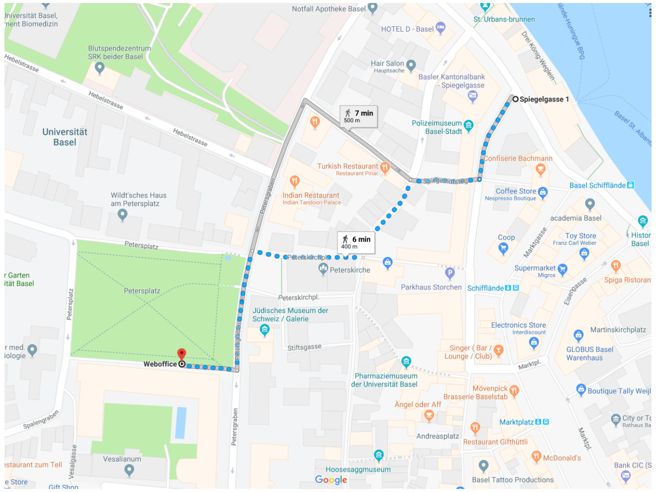
May 15, 2025

Content of the Course



C5.1 Introduction

Google Maps



Seam Carving

Applications

- Route planning
- Path planning in games
- robot navigation
- seam carving
- automated planning
- typesetting in TeX
- routing protocols in networks (OSPF, BGP, RIP)
- routing of telecommunication messages
- traffic routing

Source (partially): Network Flows: Theory, Algorithms, and Applications,
R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993

Variants

What are we interested in?

- Single source: from one vertex s to all other vertices
- Single sink: from all vertices to one vertex t
- Source-sink: from vertex s to vertex t
- All pairs: from every vertex to every vertex

Graph properties

- arbitrary / non-negative / Euclidean weights
- arbitrary / non-negative / no cycles

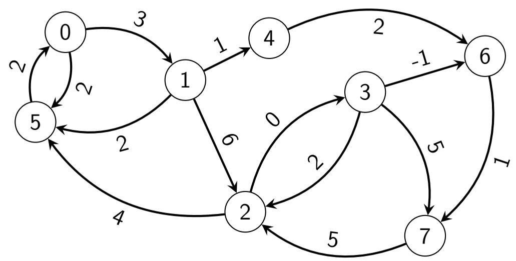
C5.2 Foundations

Weighted Directed Graphs

Same (high-level) definition of weighted graphs as before, but now we consider directed graphs.

Directed Graph

An (edge)-weighted graph associates every edge e with a weight (or cost) $\mathit{weight}(e) \in \mathbb{R}$.



Reminder: A directed graphs is also called a digraph.

API for Weighted Directed Edge

```
class DirectedEdge:
  # Edge from n1 to n2 with weight w
  def __init__(n1: int, n2: int, w: float) -> None

  # Weight of the edge
  def weight() -> float

  # Initial vertex of the edge
  def from_node() -> int

  # Terminal vertex of the edge
  def to_node() -> int
```


API for Weighted Digraphs

```
class EdgeWeightedDigraph:
  # Graph with no_nodes vertices and no edges
  def __init__(no_nodes: int) -> None

  # Add weighted edge
  def add_edge(e: DirectedEdge) -> None

  # Number of vertices
  def no_nodes() -> int

  # Number of edges
  def no_edges() -> int

  # All outgoing edges of n
  def outgoing_edges(n: int) -> Generator[DirectedEdge]

  # All edges
  def all_edges() -> Generator[DirectedEdge]
```

Shortest Path Problem

Single-source shortest path problem, SSSP

- Given: Graph and start vertex s
- Query for vertex v
  - Is there a path from s to v?
  - If yes, what is the shortest path?
- In weighted graphs:
  Shortest path is the one with lowest weight
  (= minimal sum of edge costs)

API for Shortest-path Implementation

The algorithms for shortest paths should implement the following interface:

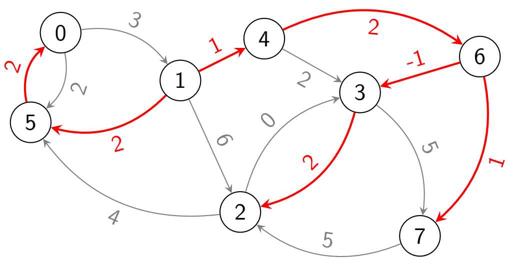
```
class ShortestPaths:
  # Initialization for start vertex s
  def __init__(graph: EdgeWeightedDigraph, s: int) -> None

  # Distance from s to v; infinity, if there is no path
  def dist_to(v: int) -> float

  # Is there a path from s to v?
  def has_path_to(v: int) -> bool

  # Path from s to v; None, if there is none
  def path_to(v: int) -> Generator[DirectedEdge]
```


Shortest-path Tree

Shortest-path Tree

For a weighted digraph G and vertex s, a shortest-path tree is a subgraph that

- forms a directed tree with root s,
- contains all vertices that are reachable from s, and
- for which every path in the tree is a shortest path in G.



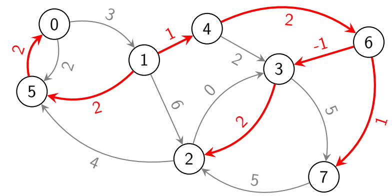
Shortest-path Tree: Representation

Representation: arrays indexed by vertex

- parent with reference to parent vertex
  None for unreachable vertices and start vertex
- distance with distance from the start vertex
  ∞ for unreachable vertices

	0	1	2	3	4	5	6	7
parent	5	None	3	6	1	1	4	6

	0	1	2	3	4	5	6	7
distance	4	0	4	2	1	2	3	4



What about parallel edges?

Extracting Shortest Paths

```
def path_to(self, node):
    if self.distance[node] == float('inf'):
        yield None
    elif self.parent[node] is None:
        yield node
    else:
        # output path from start to parent node
        self.path_to(self.parent[node])
        # finish with node
        yield node
```

This implementation generates a sequence of vertices. What do we have to change to generate a corresponding sequence of edges?

Relaxation

Relaxing edge (u, v)

- `distance[u]`: cost of the shortest known path to u
- `distance[v]`: cost of the shortest known path to v
- `parent[v]`: predecessor of v in the shortest known paths to v
- Does edge (u, v) establish a shorter path to v (through u)?
- If yes, update `distance[v]` and `parent[v]`.

Illustration: Whiteboard

Relaxation

```
def relax(self, edge):
    u = edge.from_node()
    v = edge.to_node()
    if self.distance[v] > self.distance[u] + edge.weight():
        self.parent[v] = u
        self.distance[v] = self.distance[u] + edge.weight()
```


C5.3 Optimality Criterion and Generic Algorithm

Optimality Criterion

Theorem

Let G be a weighted digraph without negative cycles. Array distance[] contains the cost of the shortest paths from s if and only if

1. distance[s] $= 0$
2. distance[w] $\leq$ distance[v] $+$ *weight*(e) *for all edges* $e = (v, w)$*, and*
3. *for all vertices* v*,* distance[v] *is the cost of some path from* s *to* v*, or* ∞ *if there is no such path.*

Optimality Criterion (Continued)

Proof

"$\Rightarrow$"

1. Since the graph has no cycles of negative cost, no path from s to s can have negative cost. Thus, the empty path is optimal and distance[s] is 0.
2. Consider an arbitrary edge e from u to v.

   The shortest path from s to u has cost distance[u]. If we extend this path by edge e, we have a path from s to v of cost distance[u] + *weight*(e). Thus, the cost of a shortest path from s to v cannot be larger and it holds that distance[v] $\leq$ distance[u] + *weight*(e).
3. Trivially true.

...

Optimality Criterion (Continued)

Proof (continued).

"$\Leftarrow$"

For unreachable vertices, the value is infinity by definition.

Consider an arbitrary vertex v and a shortest path $p = (v_0, \dots, v_n)$ from s to v, i.e. $v_0 = s$, $v_n = v$.
For $i \in \{1, \dots, n\}$, let e_i be a cheapest edge from v_{i-1} to v_i.
Since all inequalities are satisfied, we have

$$\begin{aligned} \texttt{distance}[v_n] &\leq \texttt{distance}[v_{n-1}] + \mathit{weight}(e_n) \\ &\leq \texttt{distance}[v_{n-2}] + \mathit{weight}(e_{n-1}) + \mathit{weight}(e_n) \\ &\leq \dots \leq \mathit{weight}(e_1) + \dots + \mathit{weight}(e_n) \\ &= \text{cost of an optimal path.} \end{aligned}$$

Due to 3, distance[v_n] cannot be lower than the optimal cost. □

Generic Algorithm

Generic Algorithm for Start Vertex s

- Initialize distance[s] $= 0$ and distance[v] $= \infty$ for all other vertices
- As long as the optimality criterion is not satisfied: Relax an arbitrary edge

Correct:

- Finite distance[v] always corresponds to the cost of a path from s to v.
- Every successful relaxation reduces distance[v] for some v.
- For every vertex, the distance can only be reduced finitely often.

C5.4 Summary

Summary

- Single-source shortest paths: Compute in a weighted digraph the shortest paths from a given vertex to all reachable vertices.
- Relaxation: If for edge (u,v) the best known distance to v is larger than the one to u plus the edge cost, then update the distance to v (with predecessor u).
- Generic algorithm
  - Based on relaxation and optimality criterion.
  - Every instantiation is correct for all weighted digraphs without negative-cost cycles.
  - Specific instantiations: next chapter.