

Algorithms and Data Structures

C6. Shortest Paths: Algorithms

Gabriele Röger and Patrick Schneider

University of Basel

May 20, 2026

1 / 31

Algorithms and Data Structures

May 20, 2026 — C6. Shortest Paths: Algorithms

C6.1 Dijkstra's Algorithm

C6.2 Acyclic Graphs

C6.3 Bellman-Ford Algorithm

C6.4 Summary

2 / 31

Edsger Dijkstra



Edsger Dijkstra

- ▶ Dutch mathematician, 1930–2002
- ▶ Advocate and co-developer of **structured programming**
 - ▶ Contributed to the development of programming language Algol 60
 - ▶ 1968: Essay "**Go To Statement Considered Harmful**"
- ▶ 1959: **Shortest-path** algorithm
- ▶ Winner of **Turing Award** (1972)

"Do only what only you can do."

3 / 31

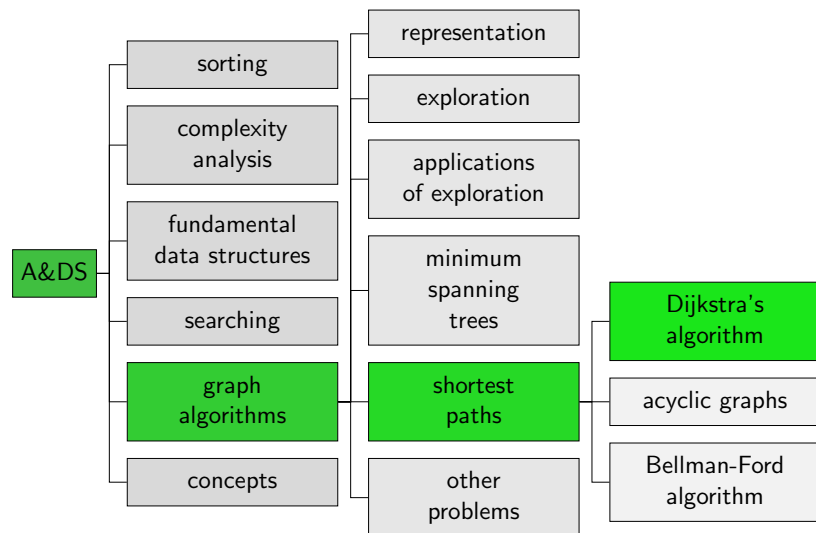
C6. Shortest Paths: Algorithms

Dijkstra's Algorithm

C6.1 Dijkstra's Algorithm

4 / 31

Content of the Course



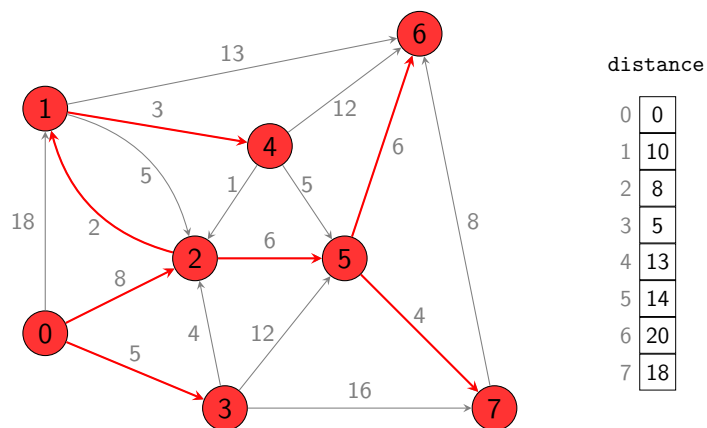
Dijkstra's Algorithm: High-Level Perspective

Dijkstra's algorithm (for non-negative edge weights)

Grow shortest-paths tree starting from vertex s :

- ▶ Consider vertices (that are not yet in the tree) in increasing order of their distance from s .
- ▶ Add the next vertex to the tree and relax its outgoing edges.

Dijkstra's Algorithm: Illustration



Data Structures

- ▶ **edge_to**: vertex-indexed array, containing at position v the last edge of a shortest known path.
- ▶ **distance**: vertex-indexed array, containing at position v the cost of the shortest known paths from the start vertex to v .
- ▶ **pq**: indexed priority queue of vertices
 - ▶ vertex not yet in the tree
 - ▶ some path to the vertex is known
 - ▶ sorted by the cost of the shortest known path to the vertex.

Dijkstra's Algorithm

```

1 class DijkstraSSSP:
2     def __init__(self, graph, start_node):
3         self.edge_to = [None] * graph.no_nodes()
4         self.distance = [float('inf')] * graph.no_nodes()
5         pq = IndexMinPQ()
6         self.distance[start_node] = 0
7         pq.insert(start_node, 0)
8         while not pq.empty():
9             self.relax(graph, pq.del_min(), pq)
10
11     def relax(self, graph, v, pq):
12         for edge in graph.outgoing_edges(v):
13             w = edge.to_node()
14             if self.distance[v] + edge.weight() < self.distance[w]:
15                 self.edge_to[w] = edge
16                 self.distance[w] = self.distance[v] + edge.weight()
17                 if pq.contains(w):
18                     pq.change(w, self.distance[w])
19                 else:
20                     pq.insert(w, self.distance[w])

```

Correctness

Theorem

*Dijkstra's algorithm solves the **single-source shortest path** problem in digraphs with **non-negative edge weights**.*

Proof.

- ▶ If v is reachable from the start vertex, every outgoing edge $e = (v, w)$ will be relaxed exactly once (when v is relaxed).
- ▶ It then holds that $distance[w] \leq distance[v] + weight(e)$.
- ▶ Inequality stays satisfied:
 - ▶ $distance[v]$ won't be changed because the value was minimal and there are no negative edge weights.
 - ▶ $distance[w]$ can only become smaller.
- ▶ If all reachable edges have been relaxed, the optimality criterion is satisfied. □

Comparison to Prim's Algorithm

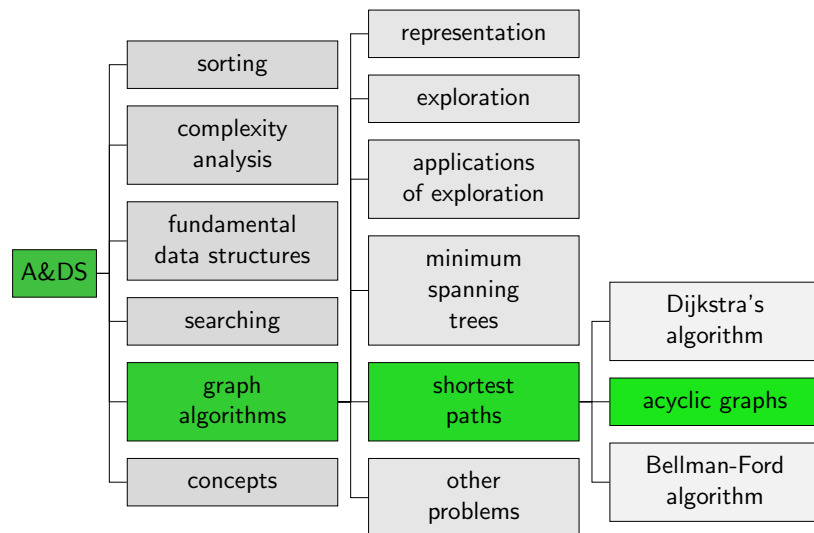
Dijkstra's algorithm is very similar to the eager variant of Prim's algorithm for minimum spanning trees.

- ▶ Both successively grow a tree.
- ▶ Prim's next vertex: minimal distance from the **grown tree**.
- ▶ Dijkstra's next vertex: minimal distance from the **start vertex**.

Running time $O(|E| \log |V|)$ and memory $O(|V|)$ directly transfer.

C6.2 Acyclic Graphs

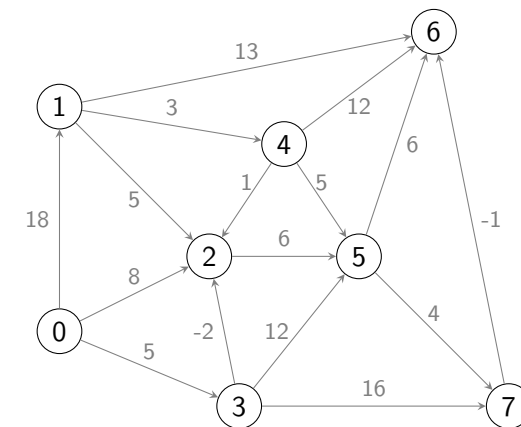
Content of the Course



13 / 31

Exploiting Acyclicity

Given: acyclic weighted digraph

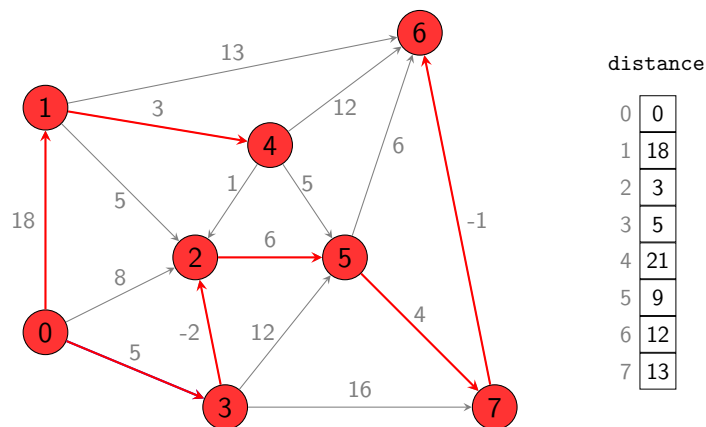


Can we exploit acyclicity during the computation of shortest paths?

14 / 31

Example

Idea: Relax vertices in **topological order**
e.g. 0, 1, 3, 4, 2, 5, 7, 6



15 / 31

Theorem

Theorem

Relaxing the vertices in **topological order**, we can solve the **single-source shortest path problem** for weighted **acyclic digraphs** in time $O(|E| + |V|)$.

Proof.

- ▶ Every edge $e = (v, w)$ gets relaxed exactly once. Directly afterwards it holds that $\text{distance}[w] \leq \text{distance}[v] + \text{weight}(e)$.
- ▶ Inequality satisfied until termination
 - ▶ $\text{distance}[w]$ never becomes larger.
 - ▶ $\text{distance}[v]$ does not get changed anymore because all incoming edges have already been relaxed.

→ Optimality criterion is satisfied at termination. □

16 / 31

Related Problems: Longest Path

Definition (Longest paths in acyclic graphs)

Given: weighted acyclic digraph, start vertex s

Question: Is there a path from s to vertex v ?

If yes, return such a path with maximum weight.

Multiply all weights with -1 and use shortest-path algorithm.

Related Problems: Critical Path

Given:

- ▶ Set of jobs a , each requires time t_a
- ▶ Constraints $a \rightarrow a'$, requiring that a must have been finished before a' can be started (in solvable problems acyclic).

Question:

- ▶ **Assumption:** We can do arbitrarily many jobs in parallel.
- ▶ How long do we need for getting all jobs done?

Related Problems: Critical Path

Create a weighted digraph:

- ▶ Vertices s, e + for every job a two vertices a_s and a_e
- ▶ for all a :
 - ▶ edge (s, a_s) with weight 0
 - ▶ edge (a_e, e) with weight 0
 - ▶ edge (a_s, a_e) with weight t_a
- ▶ for every constraint $a \rightarrow a'$ edge (a_e, a'_s) with weight 0

Critical path for job a is longest path from s to a_s .

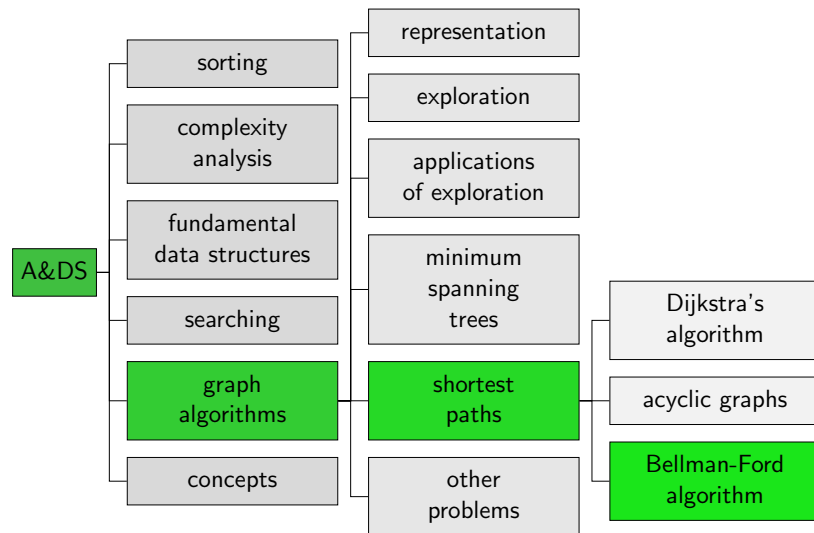
Define start time for a as weight of a critical path.

→ Results in optimal total execution time

(= weight of longest path from s to e)

C6.3 Bellman-Ford Algorithm

Content of the Course



21 / 31

Problem

- ▶ With negative edge weights there can be **negative cycles**, i.e. cycles, where the sum of edge weights is negative.
- ▶ If a vertex of such a cycle is on a path from s to v , we can find paths whose weight is lower than any given value.
→ not a well-defined problem
- ▶ Alternative question: Find a shortest **simple path**?
→ NP-hard (= very hard) problem

22 / 31

Question

In many practical applications, negative cycles indicate a modeling error.

New Questions

Given: Weighted digraph, start vertex s

Question: Is there a negative cycle that is reachable from s ?
If not, compute the shortest-path tree to all reachable vertices.

23 / 31

Bellman-Ford Algorithm: High-Level Perspective

In graphs **without negative cycles** (but with negative weights);

Bellman-Ford Algorithm

- ▶ Initialize $distance[s] = 0$ for start vertex s ,
 $distance[v] = \infty$ for all other vertices.
- ▶ Afterwards $|V|$ iterations, each relaxing all edges.

Proposition

The approach solves the single-source shortest path problem for graphs without negative cycles in time $O(|E||V|)$ and with additional memory $O(|V|)$.

Proof idea: After i iterations, every found path to v has at most the weight as any path to v with at most i edges.

24 / 31

More Efficient Variant

- ▶ If $distance[v]$ did not change in iteration i , relaxing an outgoing edge of v in iteration $i + 1$ has no effect.
- ▶ **Idea:** Remember the vertices with a changed $distance$ in a **queue**.
- ▶ Does not improve the worst-case behavior but in practice much faster.

25 / 31

What about Negative Cycles?

- ▶ If **no** negative cycles is reachable from s , then in the $|V|$ -th iteration no vertex distance will get updated anymore.
- ▶ If there is a reachable negative cycle, this will lead to a cycle in the edges stored in `edge_to`.
- ▶ In practice, we test this after relaxing the outgoing edges of certain number of vertices (e.g. $|V|$ many).

26 / 31

Bellman-Ford Algorithm

```

1 class BellmanFordSSSP:
2     def __init__(self, graph, start_node):
3         self.edge_to = [None] * graph.no_nodes()
4         self.distance = [float('inf')] * graph.no_nodes()
5         self.in_queue = [False] * graph.no_nodes()
6         self.queue = deque()
7         self.calls_to_relax = 0
8         self.cycle = None
9
10        self.distance[start_node] = 0
11        self.queue.append(start_node)
12        self.in_queue[start_node] = True
13        while (not self.has_negative_cycle() and
14              self.queue): # queue not empty
15            node = self.queue.popleft()
16            self.in_queue[node] = False
17            self.relax(graph, node)
18

```

27 / 31

Bellman-Ford Algorithm (Continued)

```

19     def relax(self, graph, v):
20         for edge in graph.outgoing_edges(v):
21             w = edge.to_node()
22             if self.distance[v] + edge.weight() < self.distance[w]:
23                 self.edge_to[w] = edge
24                 self.distance[w] = self.distance[v] + edge.weight()
25                 if not self.in_queue[w]:
26                     self.queue.append(w)
27                     self.in_queue[w] = True
28         self.calls_to_relax += 1
29         if self.calls_to_relax % graph.no_nodes() == 0:
30             self.find_negative_cycle()
31

```

28 / 31

Bellman-Ford Algorithm (Continued)

```
32     def has_negative_cycle(self):
33         return self.cycle is not None
34
35     def find_negative_cycle(self):
36         no_nodes = len(self.distance)
37         graph = EdgeWeightedDigraph(no_nodes)
38         for edge in self.edge_to:
39             if edge is not None:
40                 graph.add_edge(edge)
41
42         cycle_finder = WeightedDirectedCycle(graph)
43         self.cycle = cycle_finder.get_cycle()
```

WeightedDirectedCycle detects directed cycles in weighted graphs.

→ Sequence of depth-first searches as in DirectedCycle (C2)

C6.4 Summary

Summary

- ▶ **Non-negative weights**
 - ▶ Very common problem.
 - ▶ **Dijkstra's Algorithm** with running time $O(|E| \log |V|)$
- ▶ **Acyclic Graphs**
 - ▶ Should be exploited if it occurs in an application.
 - ▶ With **topological order** in linear time $O(|E| + |V|)$
- ▶ **Negative weights or negative cycles**
 - ▶ If there is no negative cycle, the **Bellman-Ford algorithm** finds **shortest paths**.
 - ▶ Otherwise it identifies a **negative cycle**.