Karlsruhe Institute of Technology

Practical Course: Route Planning

Introduction, Winter Semester 2026/2027

Adrian Feilhauer, Patrick Steil, Michael Zündorf | October 28, 2026



Organizational Stuff

Practical Course

- First Phase: solve 1 worksheet with 4 tasks
- Second Phase: big task in groups of 3
- Advisers: Adrian Feilhauer, Patrick Steil, Michael Zündorf
- E-mail:
  {adrian.feilhauer, steil, michael.zuendorf}@kit.edu
- 6 LP/ECTS
- If you have any questions, just come by

Homepage:
`https://scale.iti.kit.edu/teaching/2026ws/routenplanung/start`

- Application for ITI account (for pool computers) Recipient: Michael:
  `https://portal.wiwi.kit.edu/forms/form/ITIregister`

Organizational Stuff

Requirements

- You are in the Master's program in Computer Science, everyone else, please raise a hand

Organizational Stuff

Requirements

- You are in the Master's program in Computer Science, everyone else, please raise a hand
- You are interested in algorithmic problems
- You like implementing algorithms

Organizational Stuff

Requirements

- You are in the Master's program in Computer Science, everyone else, please raise a hand
- You are interested in algorithmic problems
- You like implementing algorithms

Overview

- Worksheet
- Introductory Presentations
- Group Task
- Final Presentation
- Report

Organizational Stuff

Requirements

- You are in the Master's program in Computer Science, everyone else, please raise a hand
- You are interested in algorithmic problems
- You like implementing algorithms

Overview

- Worksheet
- Introductory Presentations
- Group Task
- Final Presentation
- Report

Implementation

- C++
- or Rust

Organizational Stuff

Worksheet

- Tasks give points
  - Points do not count towards the final grade
  - 2.5 million points must be reached to pass
- Groups will be formed based on points
- The group with the most points can choose the group project topic first
- After the worksheet: formal exam registration
  Meaning: from then on, nothing done → failed
- Group work is harder than the worksheet

Worksheet Points

- Each task: List of start and target node pairs
- You need to calculate the path length
- Points for task = #correctly calculated path lengths

Passing

You must reach 2.5 million points to pass!

Assistance for Worksheets

- If you have any questions or problems, feel free to reach out to us
- If you ask, we can provide feedback on your code

Assistance for Worksheets

- If you have any questions or problems, feel free to reach out to us
- If you ask, we can provide feedback on your code

- However: initiative is encouraged!
- It's your responsibility to approach one of the advisers with any issues
- Those who don't ask, don't get help

Organizational Stuff

Group Work

- Work in groups of 3
- Task: re-implement a research paper
  - Each group gets a different paper
- Visualize the results
- Reproduce some experiments from the paper
- Design and conduct some new experiments

Assignment and Topics

- Group assignment based on worksheet
- Presentation of topics during group assignment

Organizational Stuff

Introductory Presentation

- 10 minutes
- Explain the problem statement and the core approach

Report

- Describe everything you have implemented in your own words
- Document experiments and results

Final Presentation

- 20–30 minutes
- Present the content of the report

Organizational Stuff

Work

- 6 ECTS/LP
- $6 \times 30\text{h} = 180\text{h}$
- Over 20 weeks: 9h per week, which is just over 1 full day per week

Rough Distribution

- 35h Worksheet
- 95h Group task, including
  - Familiarizing with the topic
  - Implementation
- 5h Introductory presentation
- 20h Final presentation
- 20h Report
- 5h meetings/events

Schedule

When?	Where?	What?
Today 28.10. at 14:00	SR -120	introduction
17.11. 23:59	—	submission deadline
18.11.	—	results by e-mail
18.11. at 14:00	SR -120	topic and group assignment
03.12. at 14:00	SR -120	introductory presentations
18.–22.01.	—	interim meeting
03.03. 23:59	—	report deadline
24.03. at 14:00	SR -120	final presentation

Attendance is mandatory. If you cannot attend, you must give notice with a valid reason.

Problem

Goal:

- Find the best connection in a transportation network

Idea:

- Model network as a graph $G = (V, E)$
- Path through the graph corresponds to a route
- Classic problem (Dijkstra)

Problems:

- Transportation networks are large
- Dijkstra is too slow (> 1 second)



Problem

Goal:

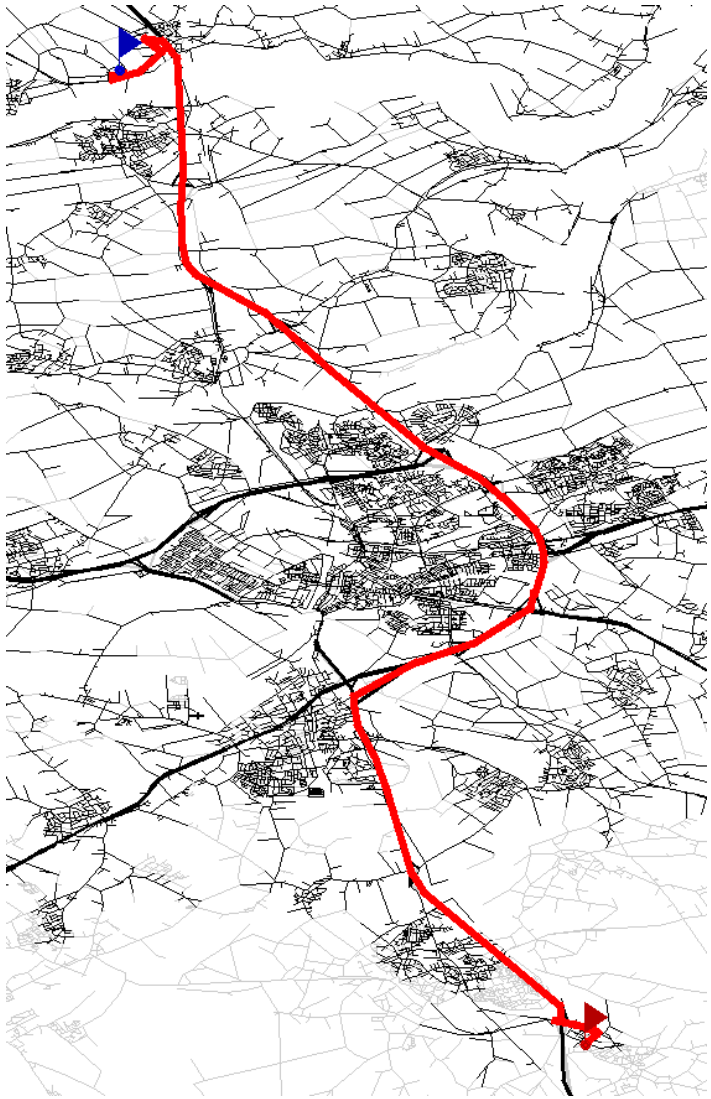
- Find the best connection in a transportation network

Idea:

- Model network as a graph $G = (V, E)$
- Path through the graph corresponds to a route
- Classic problem (Dijkstra)

Problems:

- Transportation networks are large
- Dijkstra is too slow (> 1 second)



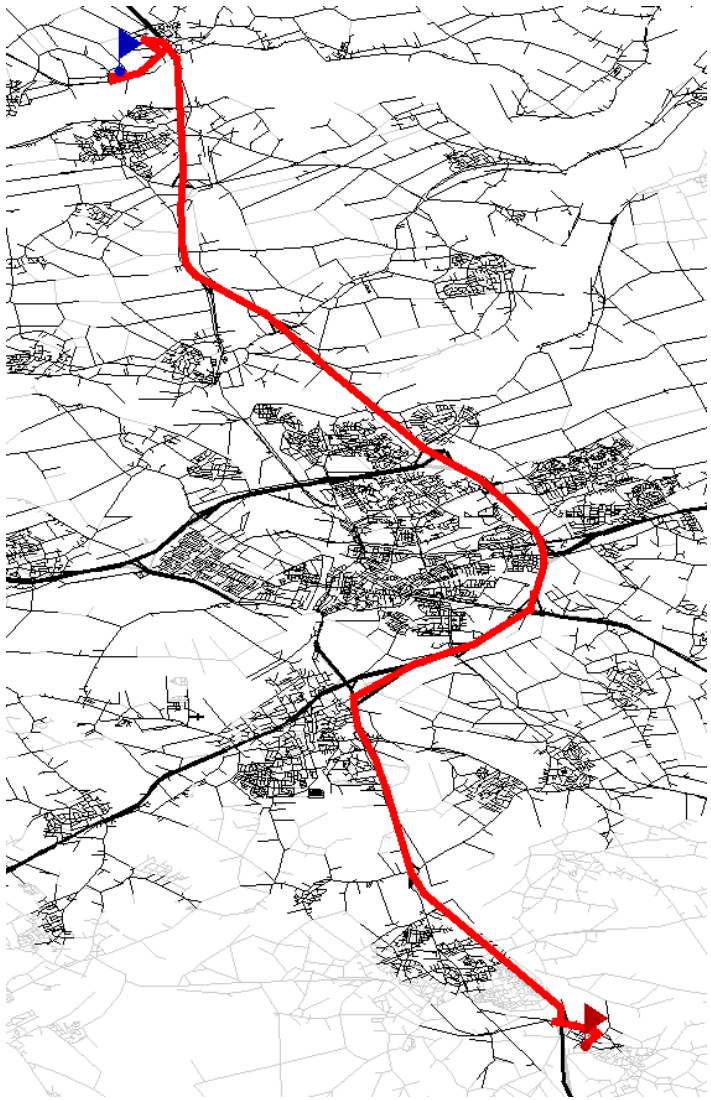
Speed-Up Techniques

Observations:

- Many queries in a (static) network
- Some computations seem unnecessary

Idea:

- Two-phase algorithm:
  - offline: compute additional information during preprocessing
  - online: accelerate computation with this additional information
- Three criteria:
  - Little additional information
  - Short preprocessing time (in the range of hours/minutes)
  - High acceleration



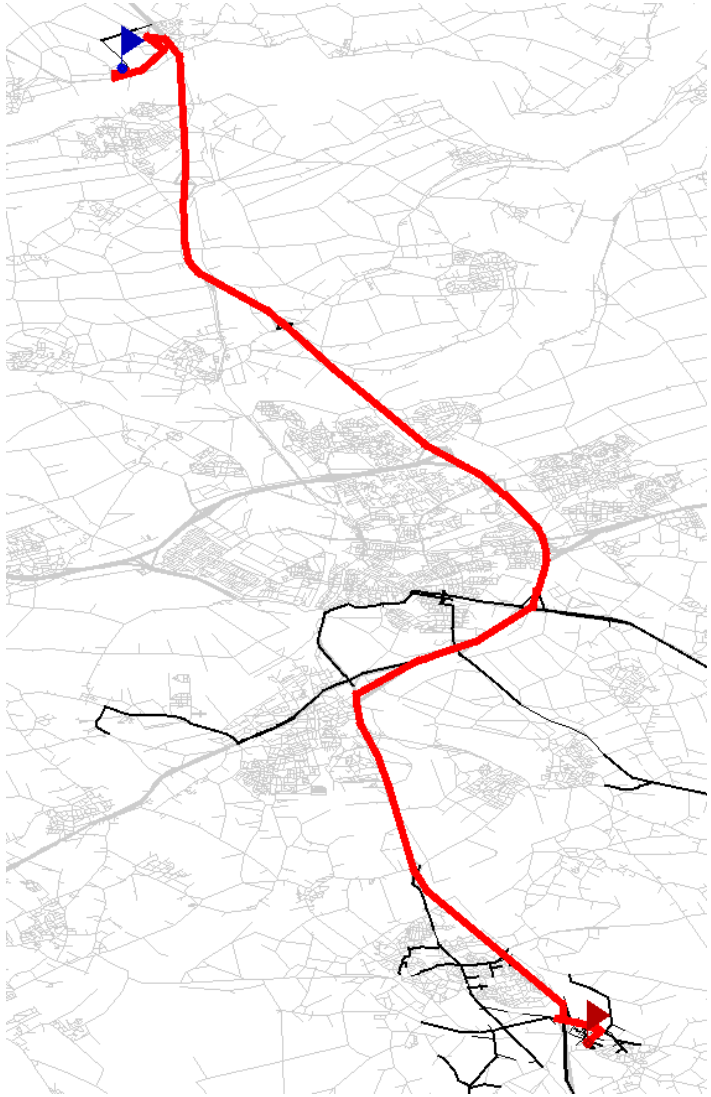
Speed-Up Techniques

Observations:

- Many queries in a (static) network
- Some computations seem unnecessary

Idea:

- Two-phase algorithm:
  - offline: compute additional information during preprocessing
  - online: accelerate computation with this additional information
- Three criteria:
  - Little additional information
  - Short preprocessing time (in the range of hours/minutes)
  - High acceleration



Modeling (Road Network)

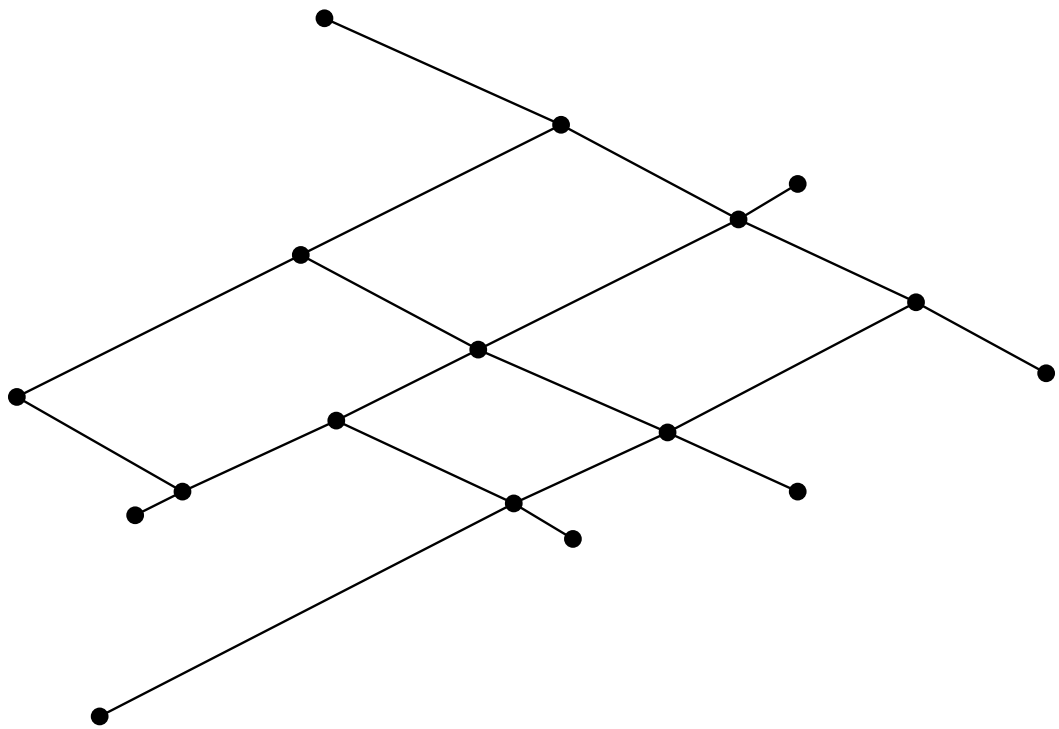


Modeling (Road Network)



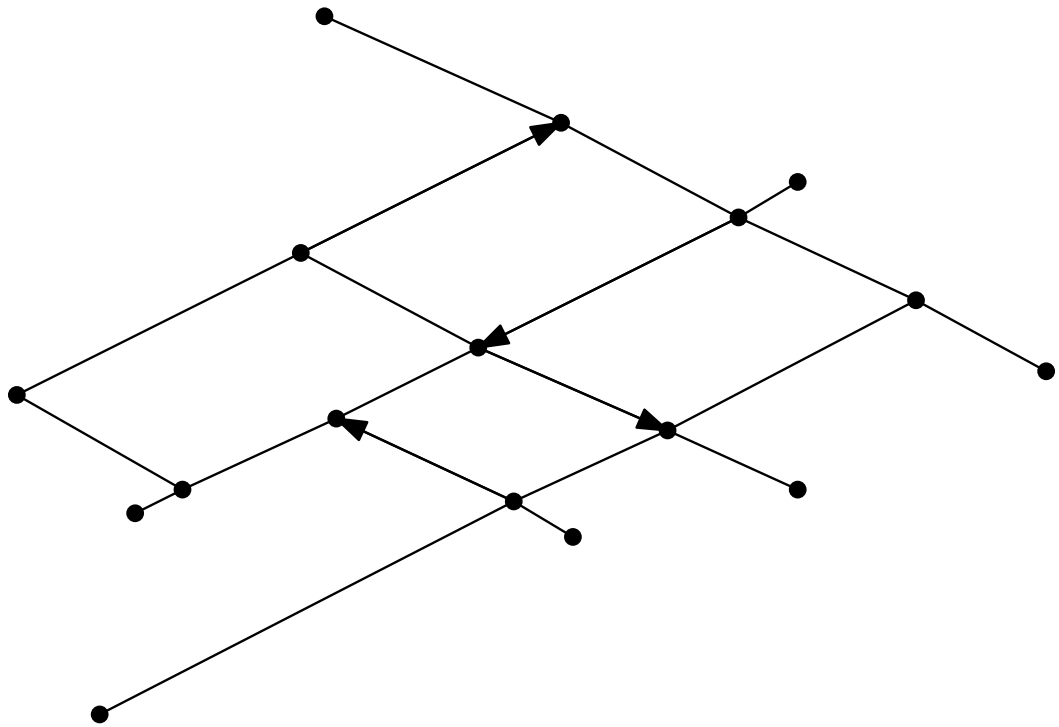
Modeling (Road Network)

- Nodes are intersections
- Edges are roads



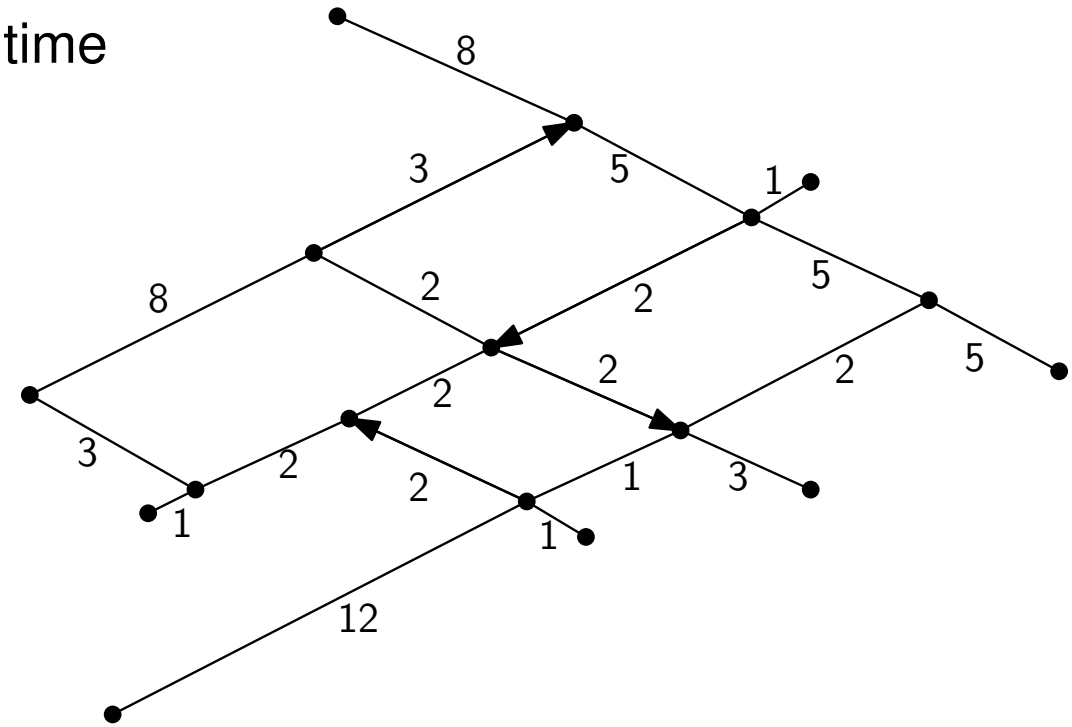
Modeling (Road Network)

- Nodes are intersections
- Edges are roads
- One-way streets

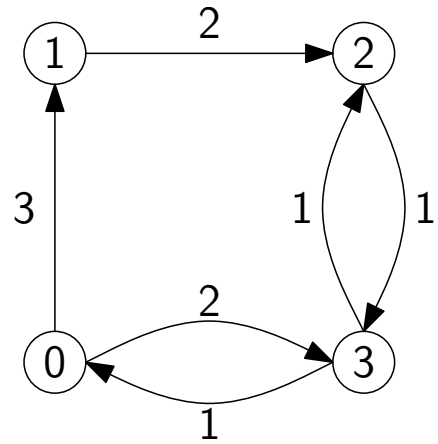


Modeling (Road Network)

- Nodes are intersections
- Edges are roads
- One-way streets
- Metric is travel time



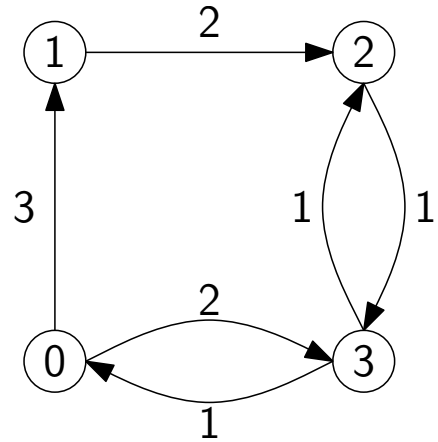
Graph Representations



Graph Representations

Three classic approaches:

- Adjacency matrix

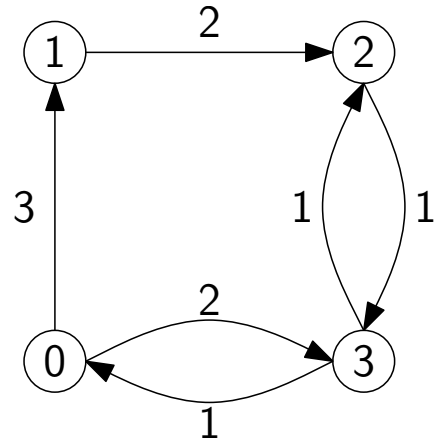


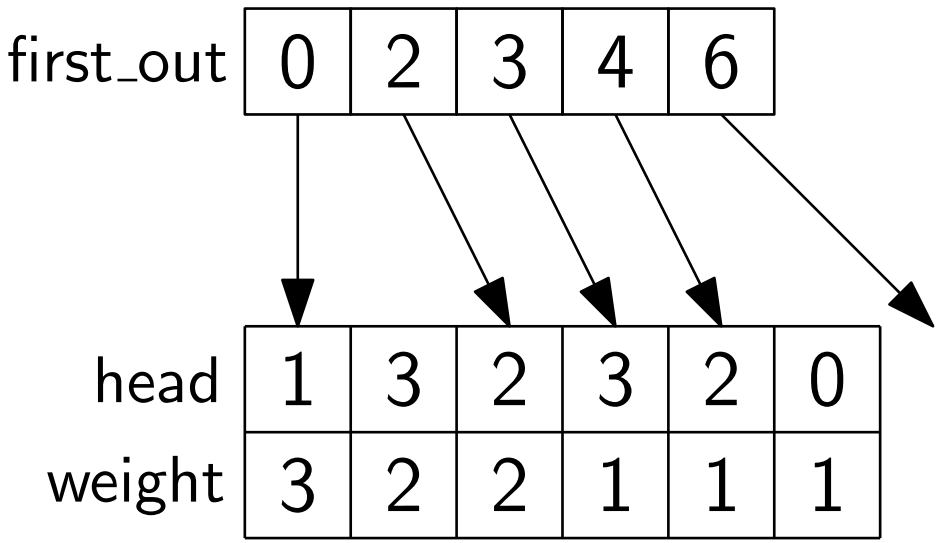
	0	1	2	3
0	–	3	–	2
1	–	–	2	–
2	–	–	–	1
3	1	–	1	–

Graph Representations

Three classic approaches:

- (static) adjacency array

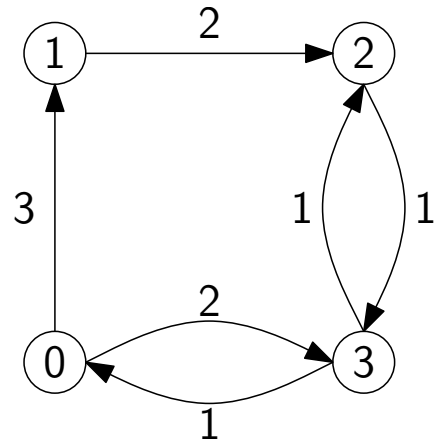




Graph Representations

Three classic approaches:

- Edge array



tail	1	0	3	0	2	3
head	2	1	2	3	3	0
weight	2	3	1	2	1	1

What do we use?

Adjacency matrix:

- Requires $O(n^2)$ memory
- $n = 18 \cdot 10^6$
- Memory $\geq 1/4$ terabyte
- Impractical

What do we use?

Adjacency matrix:

- Requires $O(n^2)$ memory
- $n = 18 \cdot 10^6$
- Memory $\geq 1/4$ terabyte
- Impractical

Edge array:

- Perfect for simple transformations (e.g., reversing the graph)
- Traversal (i.e., pathfinding) is not possible

What do we use?

Adjacency matrix:

- Requires $O(n^2)$ memory
- $n = 18 \cdot 10^6$
- Memory $\geq 1/4$ terabyte
- Impractical

Edge array:

- Perfect for simple transformations (e.g., reversing the graph)
- Traversal (i.e., pathfinding) is not possible

Adjacency array:

- Good for path searching

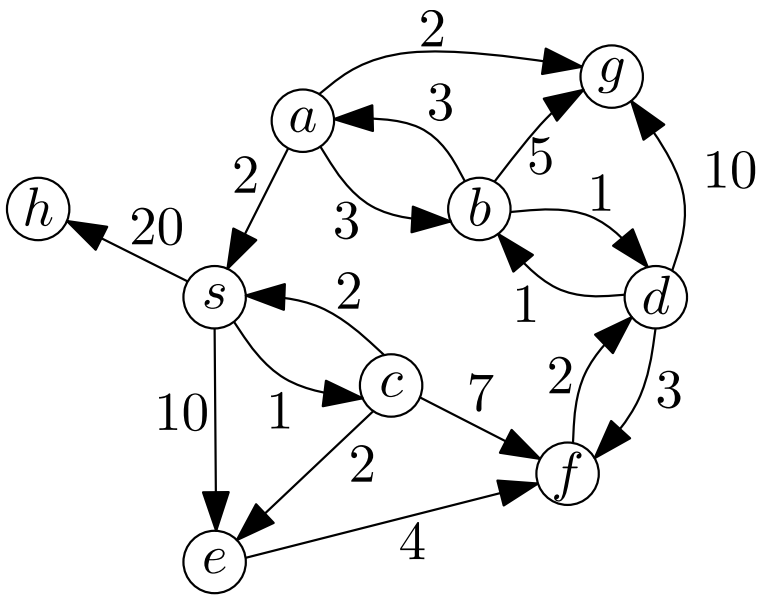
Conversion
Edge Array → Adjacency Array

- Sort by tail
- Calculate the out-degree of each node
- first_out =prefix sum over the array of out-degrees

Dijkstra's Algorithm

```
1  forall nodes v ∈ V do
2      d[v] = ∞
3  d[s] = 0
4  q.clear()
5  q.insert(s, 0)
6  while !q.empty() do
7      x ← q.pop()
8      forall edges (x, y) ∈ E do
9          if d[x] + len(x, y) < d[y] then
10             d[y] ← d[x] + len(x, y)
11             if y ∈ q then q.decreaseKey(y, d[y])
12             else q.insert(y, d[y])
```

Dijkstra's Algorithm



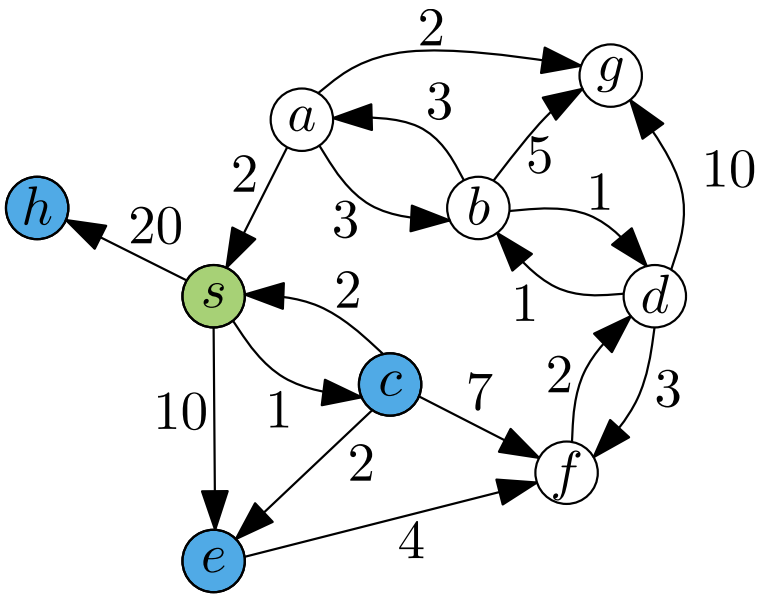
tentative distance d:

ID	Dist.
s	0
a	∞
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞

queue q:

ID	Key
s	0

Dijkstra's Algorithm



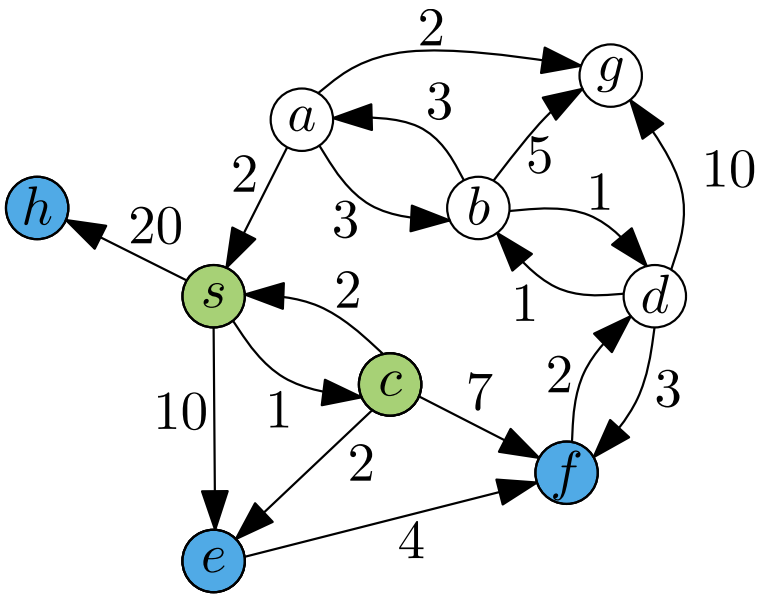
tentative distance d:

ID	Dist.
s	0
a	∞
b	∞
c	1
d	∞
e	10
f	∞
g	∞
h	20

queue q:

ID	Key
c	1
e	10
h	20

Dijkstra's Algorithm



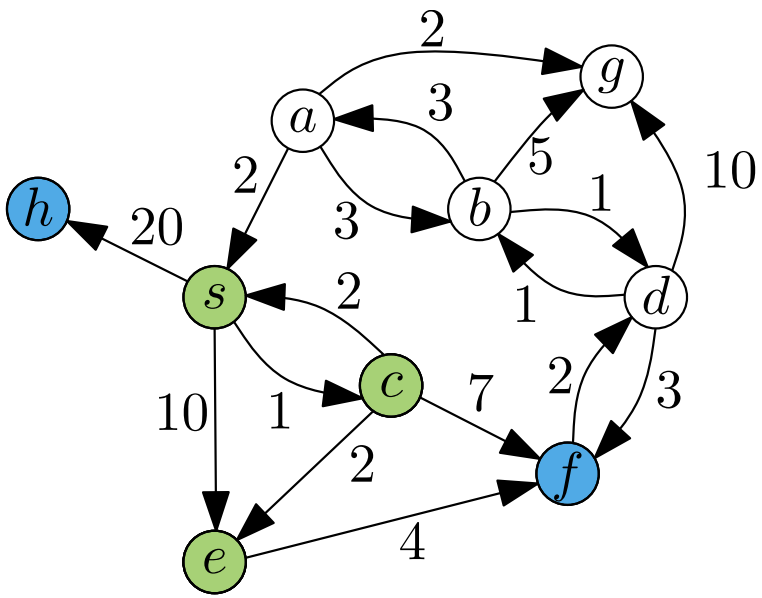
tentative distance d:

ID	Dist.
s	0
a	∞
b	∞
c	1
d	∞
e	3
f	8
g	∞
h	20

queue q:

ID	Key
e	3
f	8
h	20

Dijkstra's Algorithm



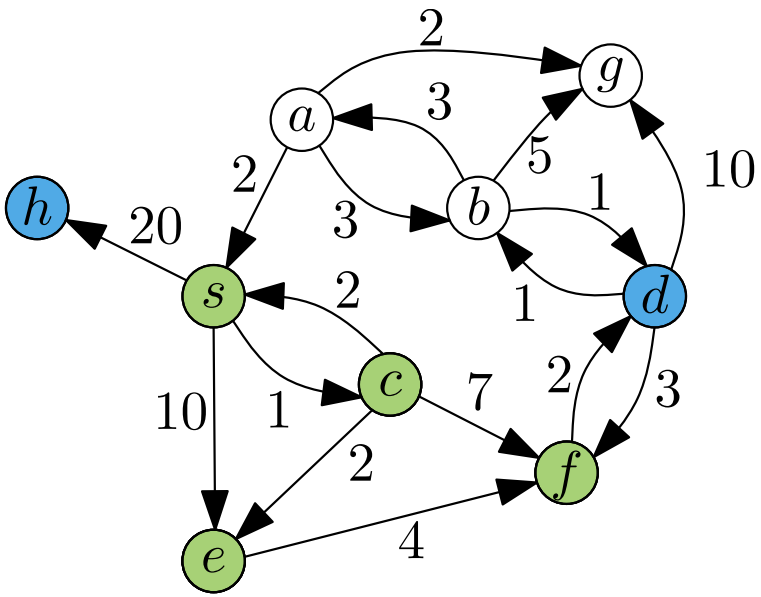
tentative distance d:

ID	Dist.
s	0
a	∞
b	∞
c	1
d	∞
e	3
f	7
g	∞
h	20

queue q:

ID	Key
f	7
h	20

Dijkstra's Algorithm



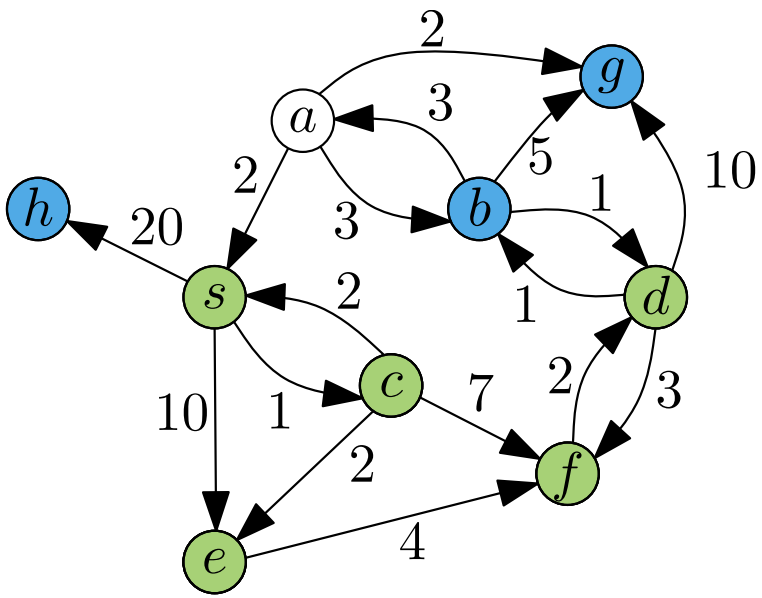
tentative distance d:

ID	Dist.
s	0
a	∞
b	∞
c	1
d	9
e	3
f	7
g	∞
h	20

queue q:

ID	Key
d	9
h	20

Dijkstra's Algorithm



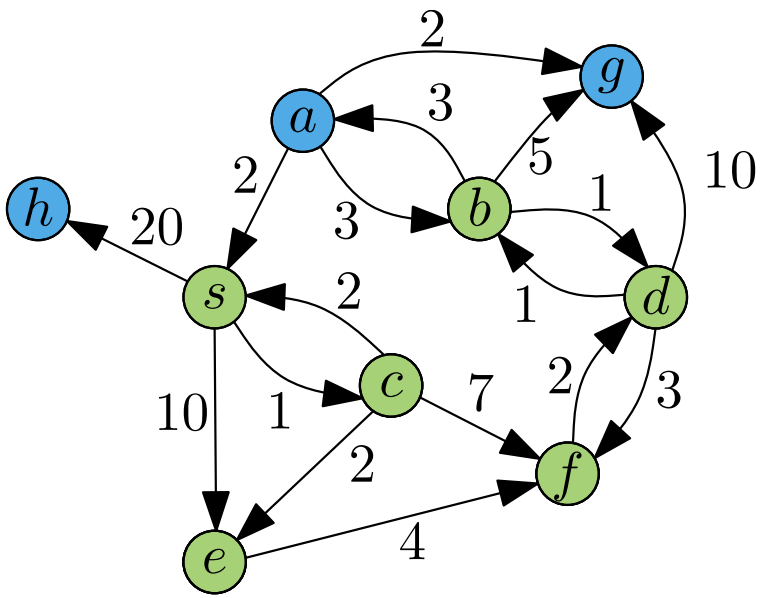
tentative distance d:

ID	Dist.
s	0
a	∞
b	10
c	1
d	9
e	3
f	7
g	19
h	20

queue q:

ID	Key
b	10
g	19
h	20

Dijkstra's Algorithm



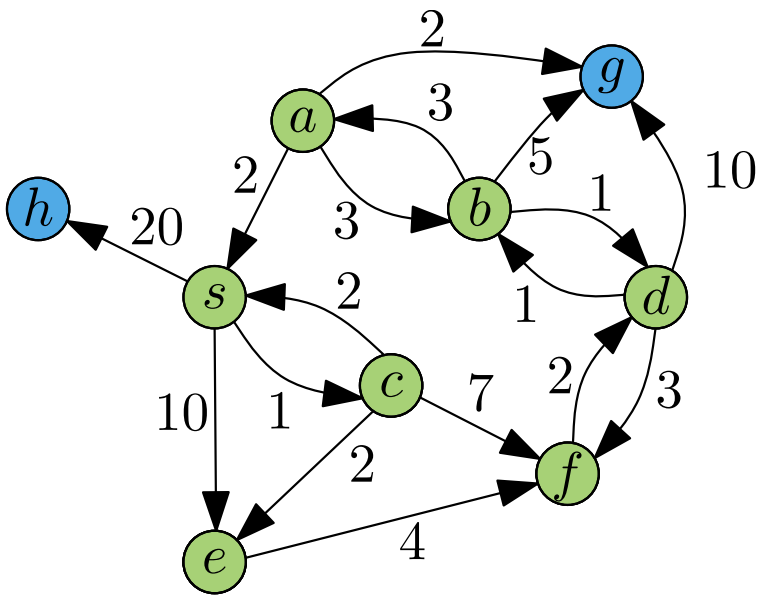
tentative distance d:

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q:

ID	Key
a	13
g	15
h	20

Dijkstra's Algorithm



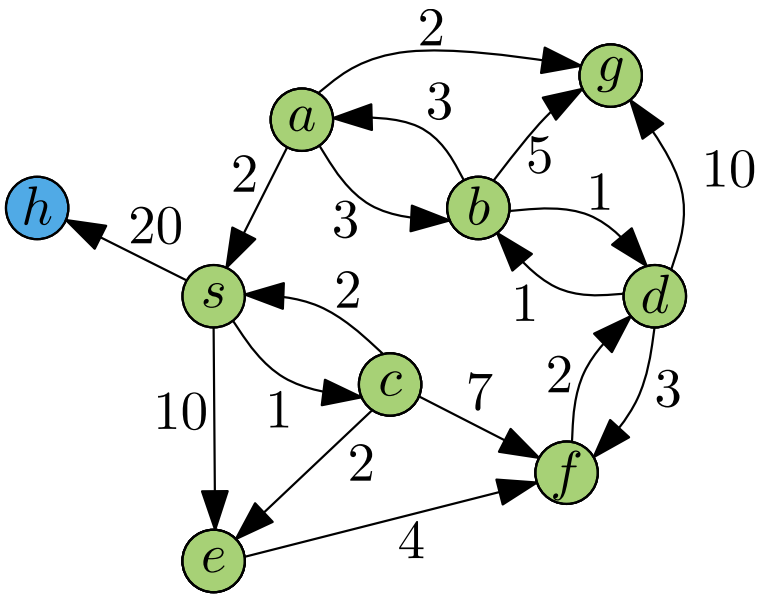
tentative distance d:

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q:

ID	Key
g	15
h	20

Dijkstra's Algorithm



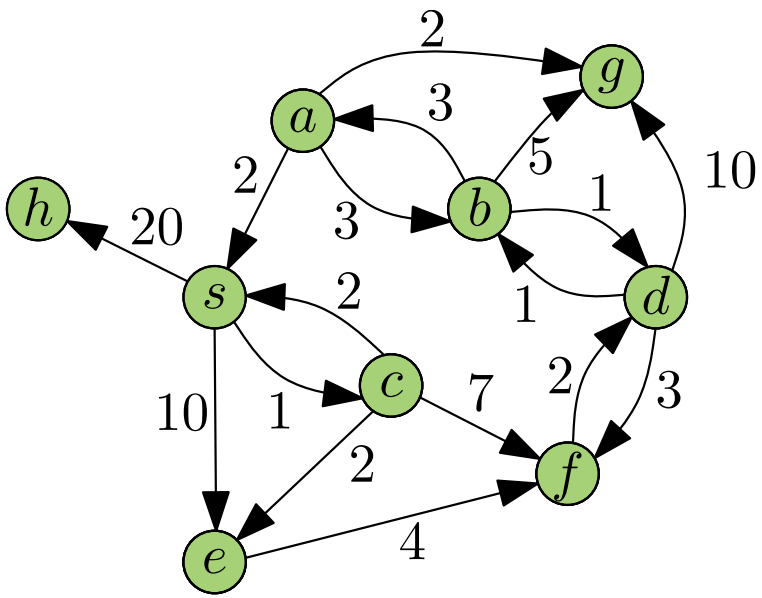
tentative distance d:

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q:

ID	Key
h	20

Dijkstra's Algorithm



tentative distance d:

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q:

ID	Key

Dijkstra's Algorithm

Karlsruhe Institute of Technology

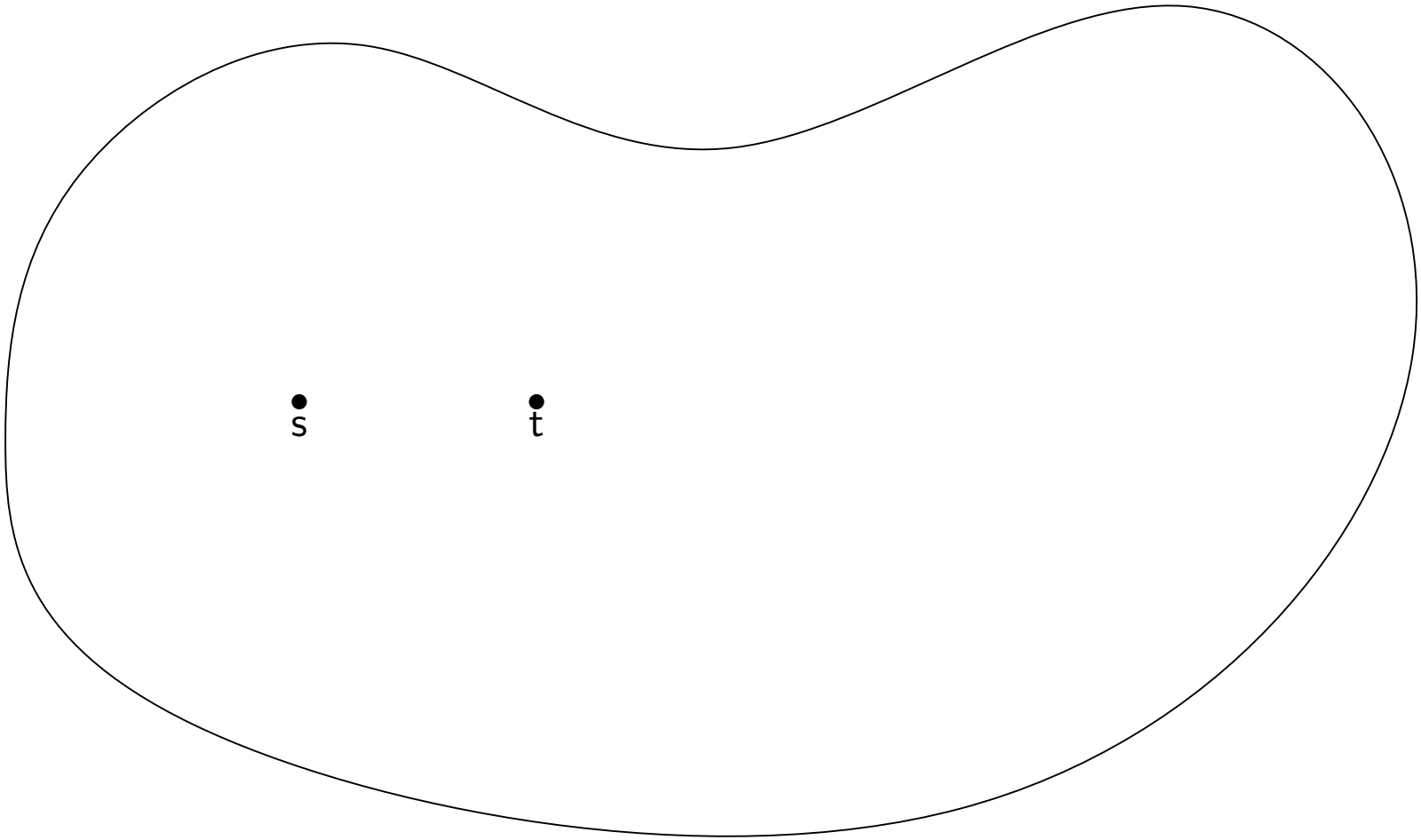
Result

- After execution: $\forall v : d[v] = \text{dist}_G(s, v)$

Stopping Criterion

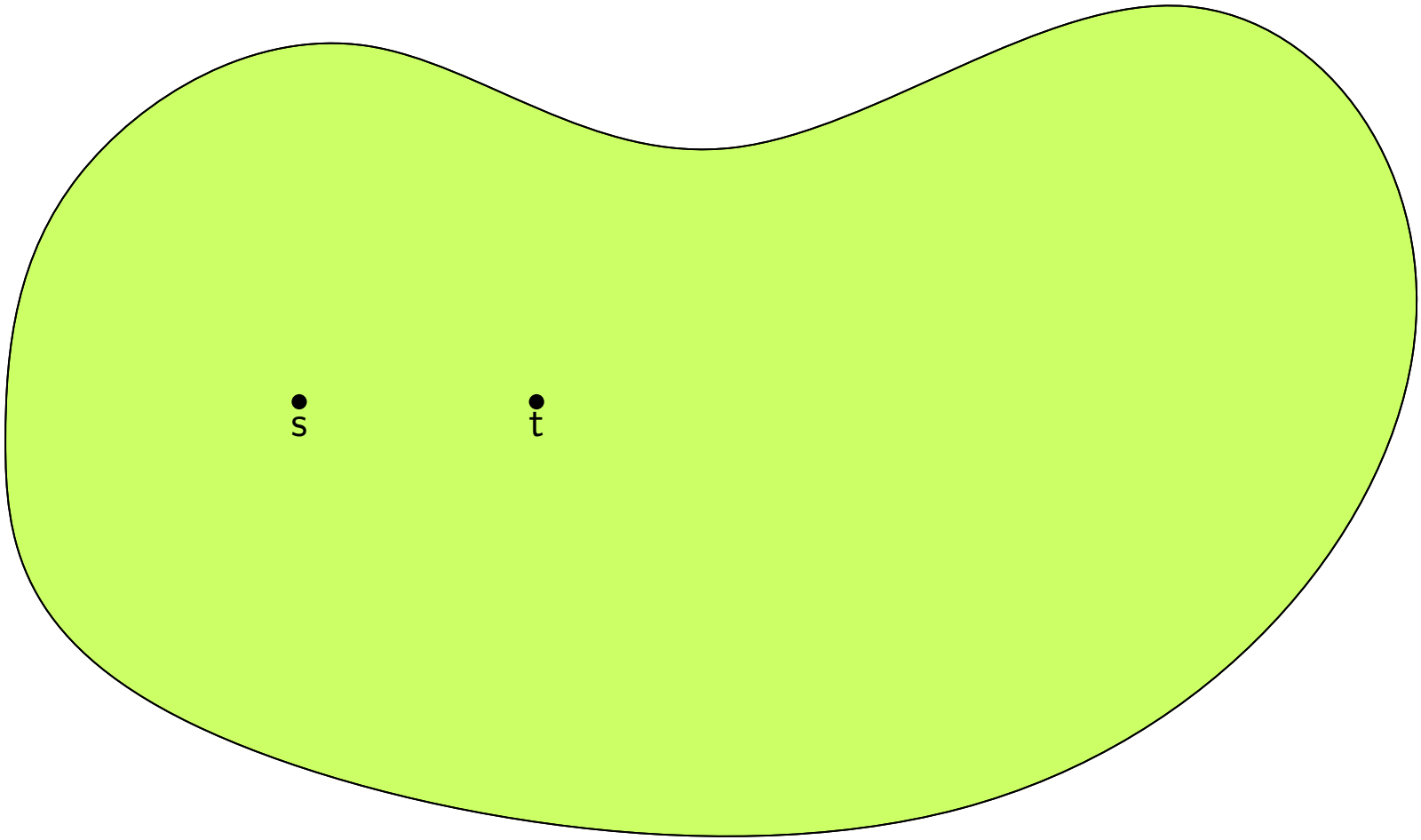
- Is it faster if we only need to determine $\text{dist}_G(s, t)$ for a single t?
- Yes: Stop the loop as soon as t is removed from the queue

Schematic Search Space



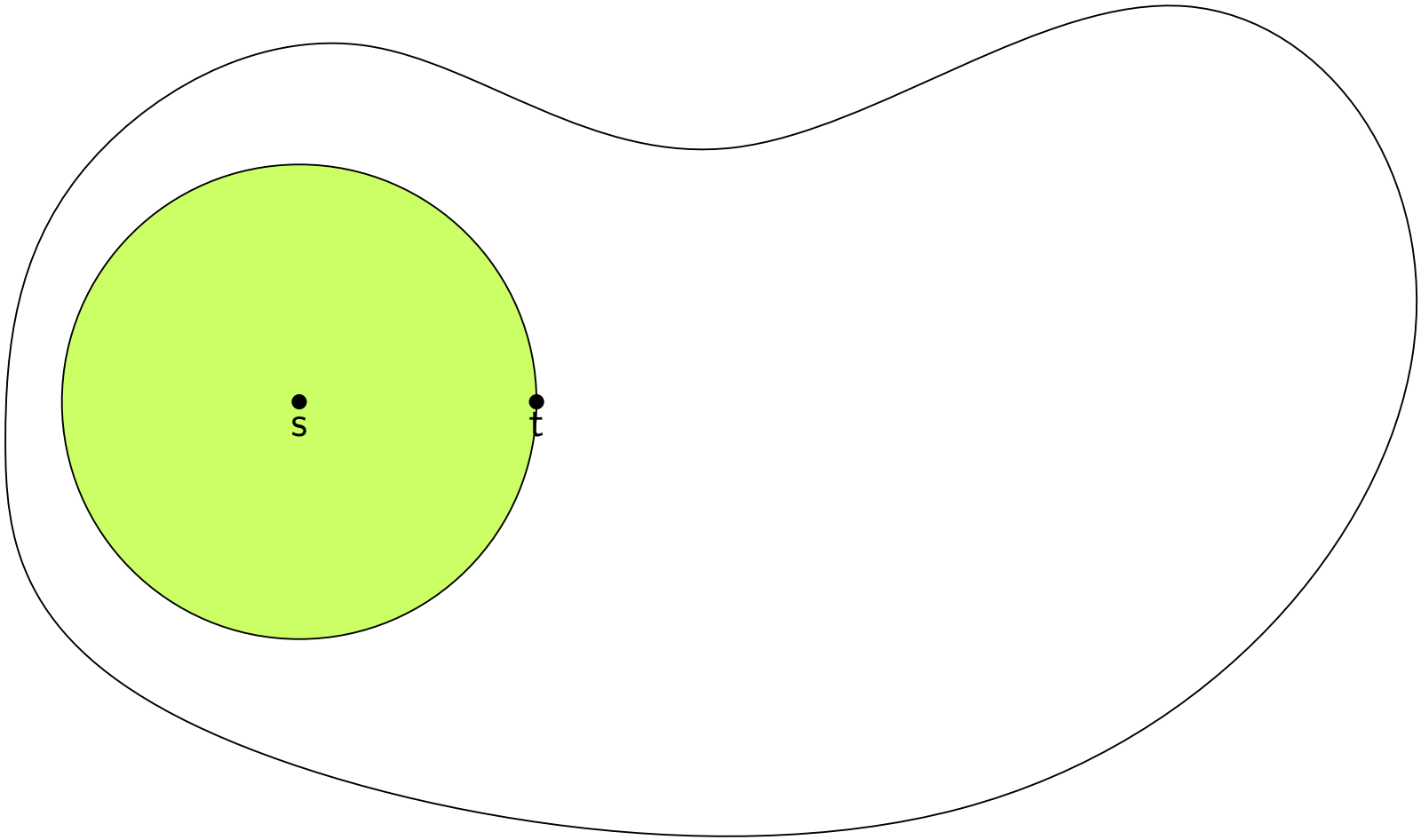
A graph

Schematic Search Space



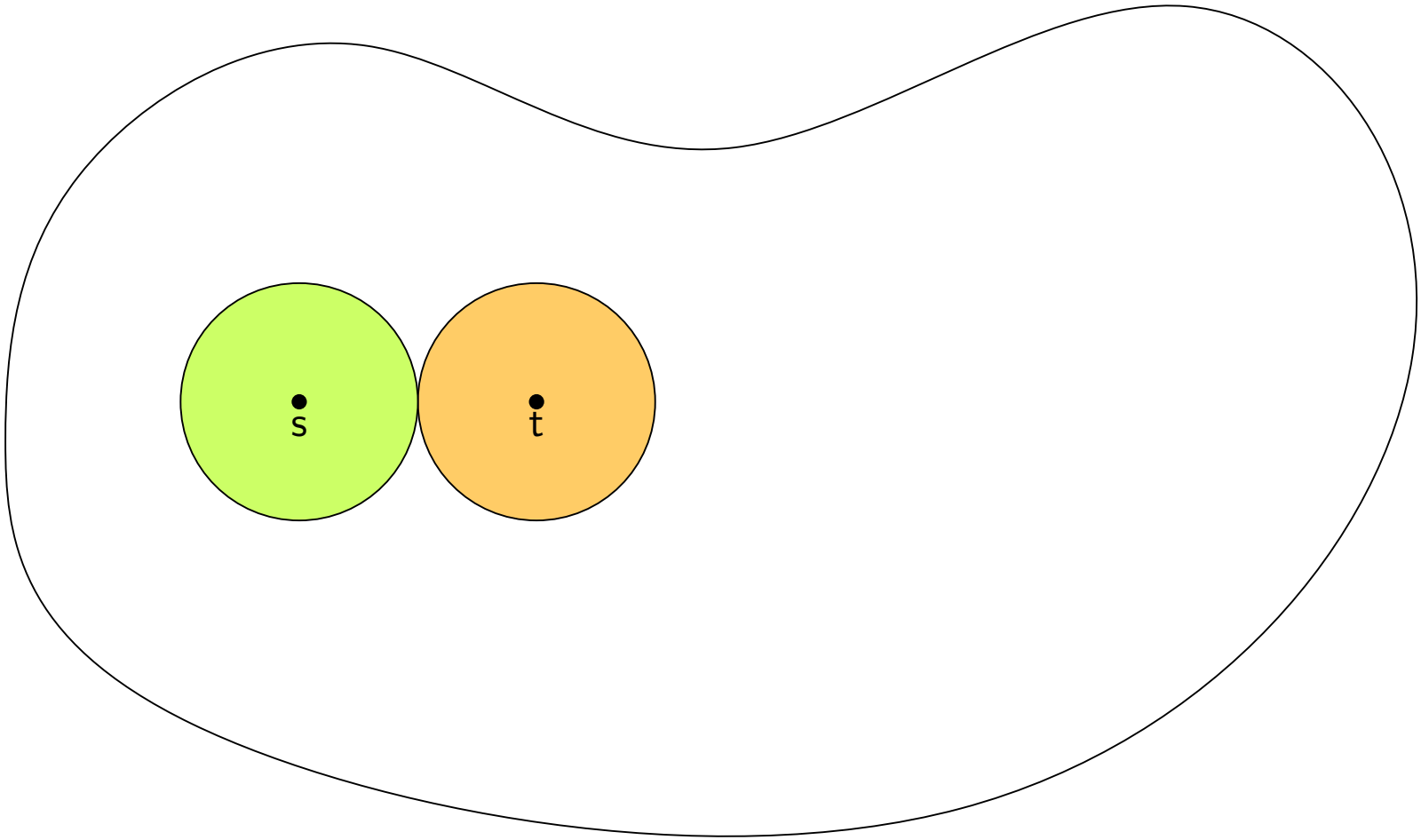
Search space without stopping criterion

Schematic Search Space



Search space with stopping criterion

Schematic Search Space

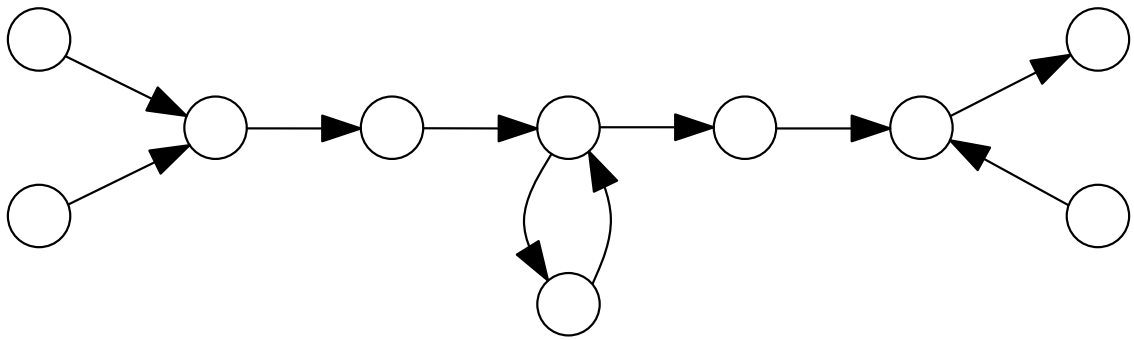


Bidirectional version of Dijkstra's algorithm

Bidirectional Dijkstra's algorithm

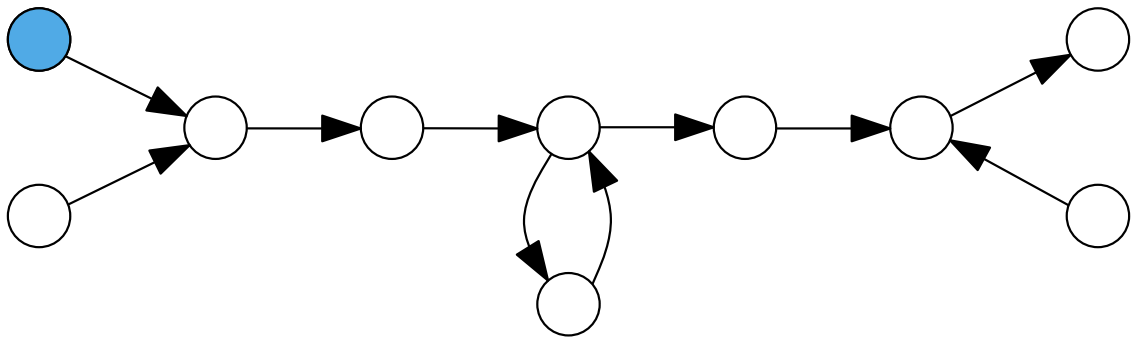
- Bidirectional version of Dijkstra's algorithm
- Two queues and two tentative distance arrays
- Process the side with fewer elements in the queue next
- Stop if the sum of the min-keys of both queues is greater than the shortest path found so far

Shortcut



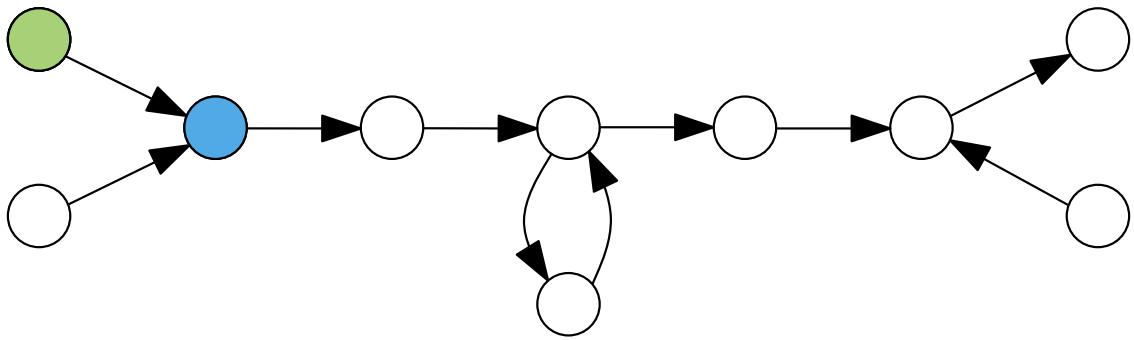
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



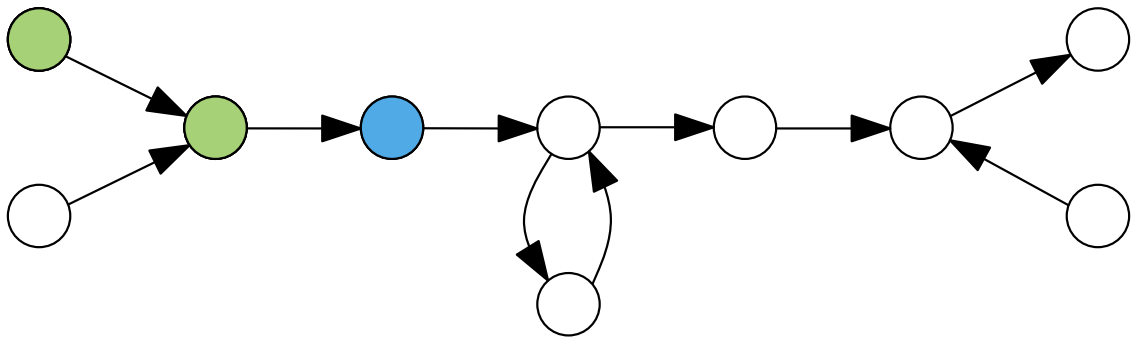
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



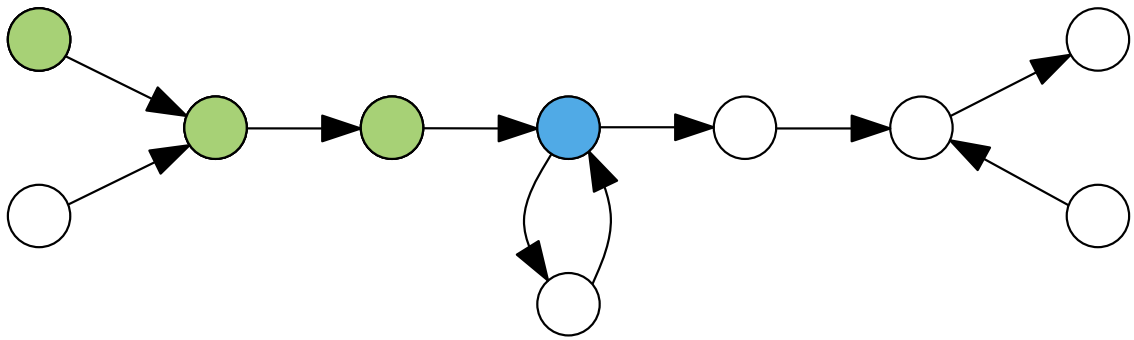
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



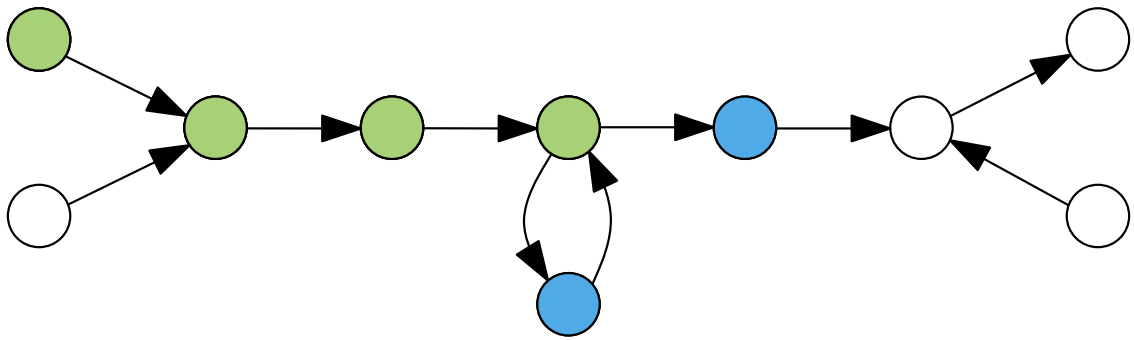
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



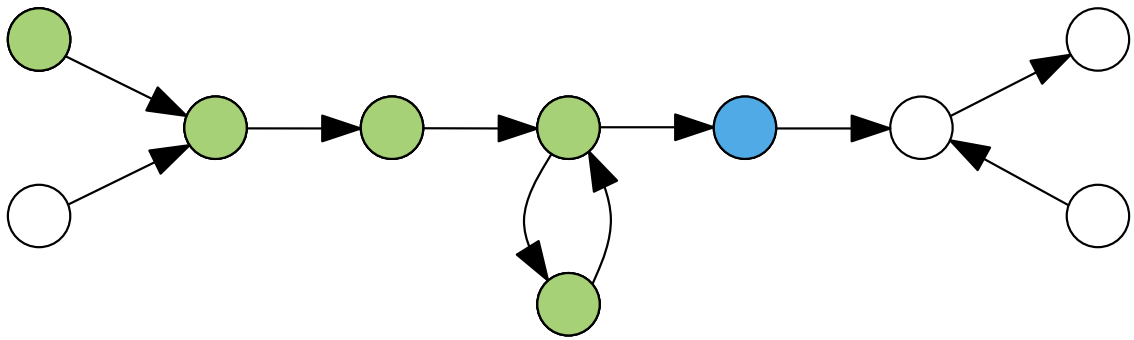
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



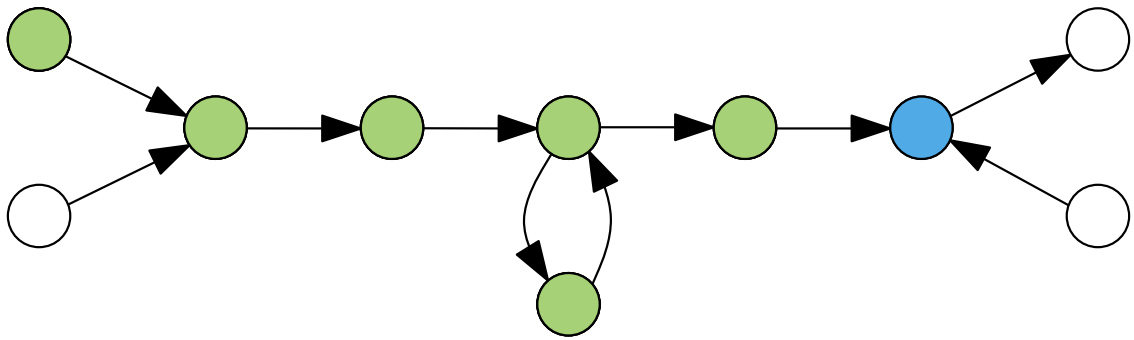
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



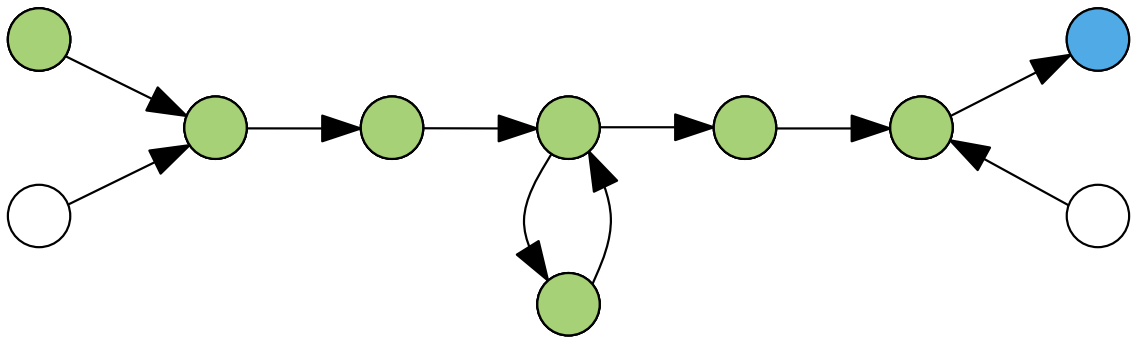
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



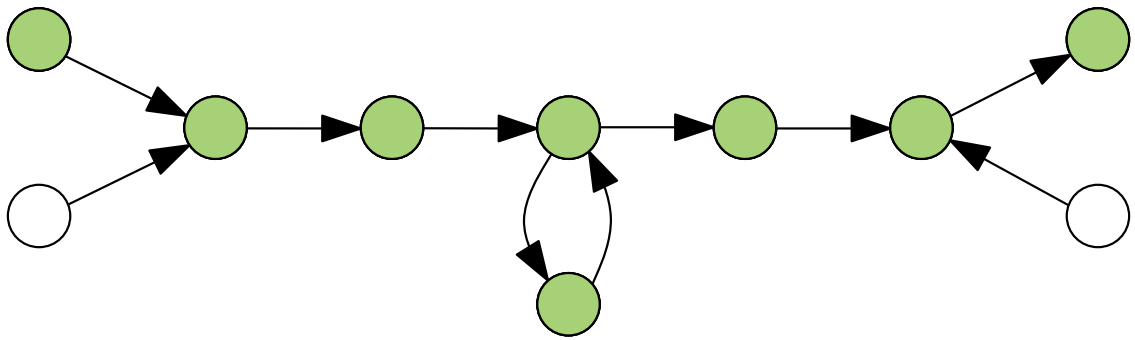
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



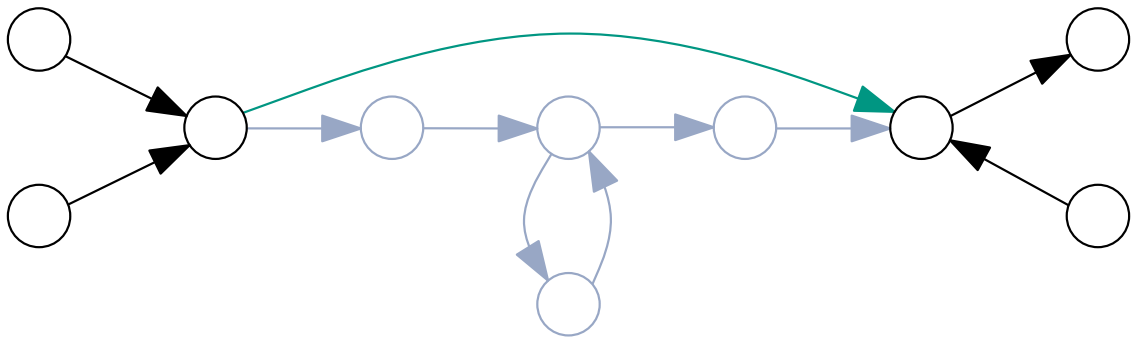
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



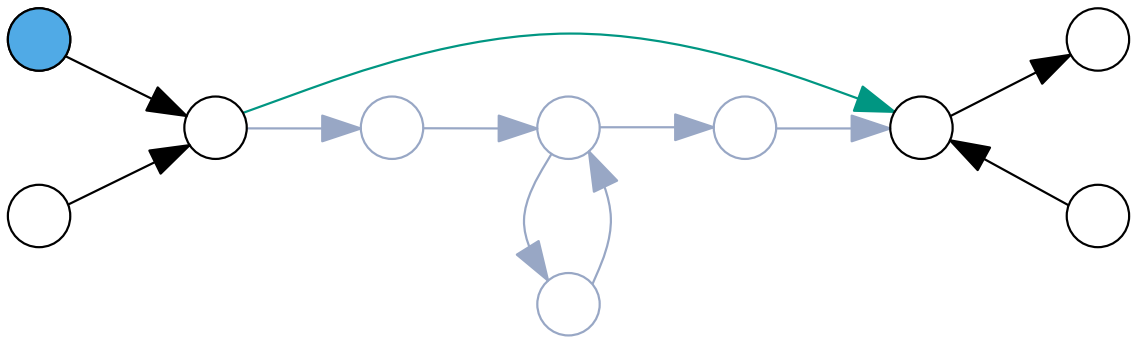
Dijkstra's algorithm visits all intermediate nodes.
This takes time.

Shortcut



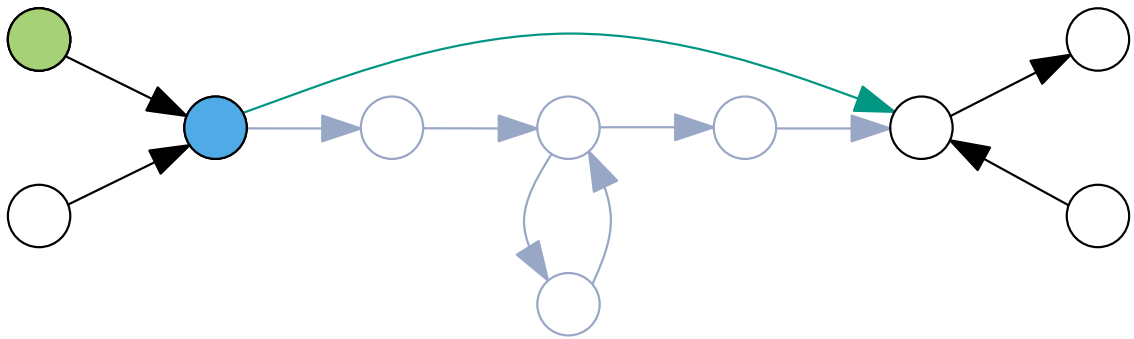
Idea: Add shortcut edges. The gray subgraph only needs to be considered if s or t are included.

Shortcut



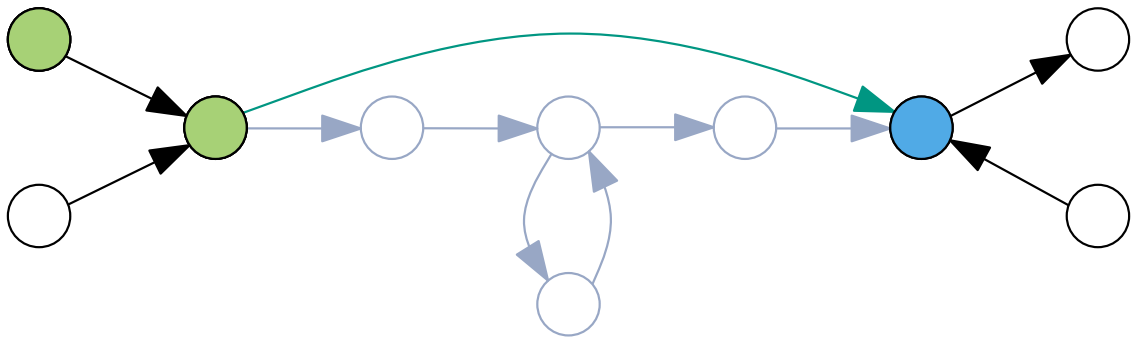
Idea: Add shortcut edges. The gray subgraph only needs to be considered if *s* or *t* are included.

Shortcut



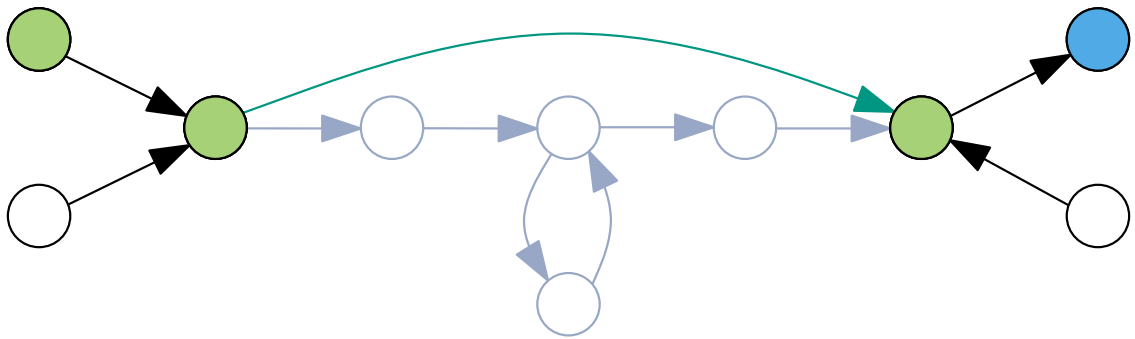
Idea: Add shortcut edges. The gray subgraph only needs to be considered if *s* or *t* are included.

Shortcut



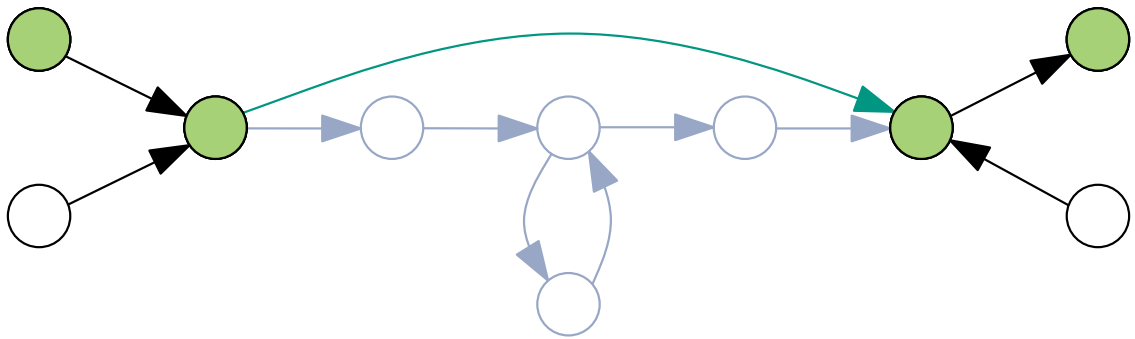
Idea: Add shortcut edges. The gray subgraph only needs to be considered if *s* or *t* are included.

Shortcut



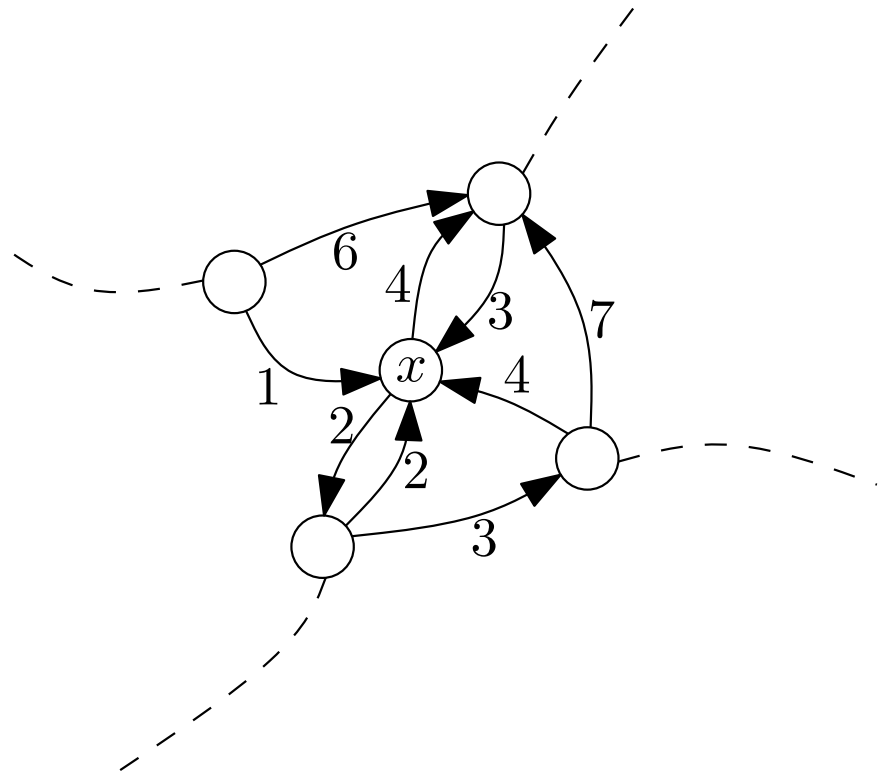
Idea: Add shortcut edges. The gray subgraph only needs to be considered if s or t are included.

Shortcut



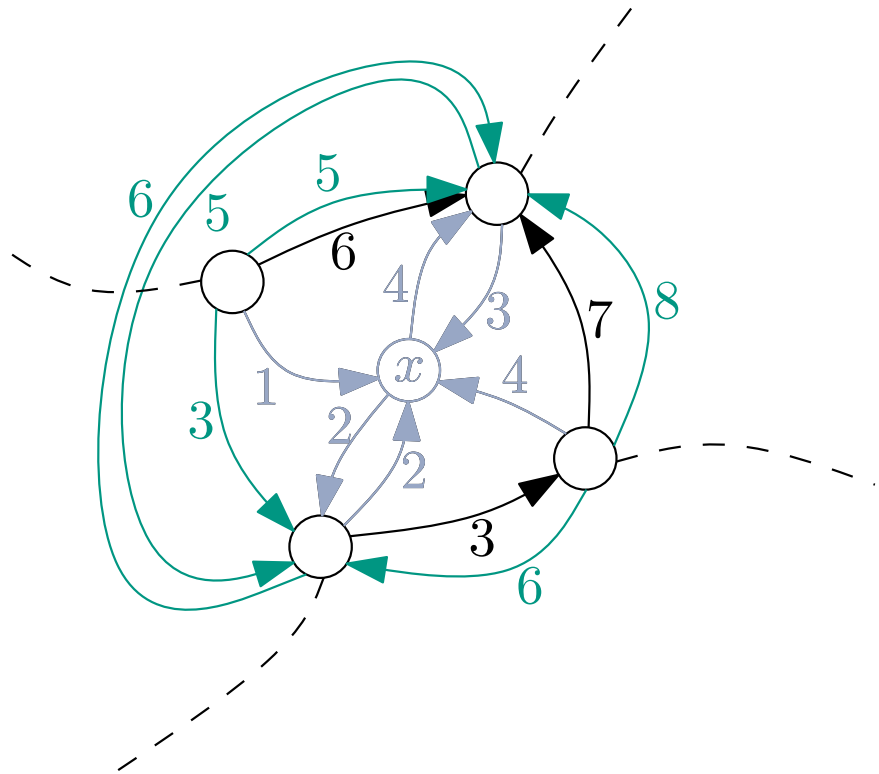
Idea: Add shortcut edges. The gray subgraph only needs to be considered if *s* or *t* are included.

Contraction of x



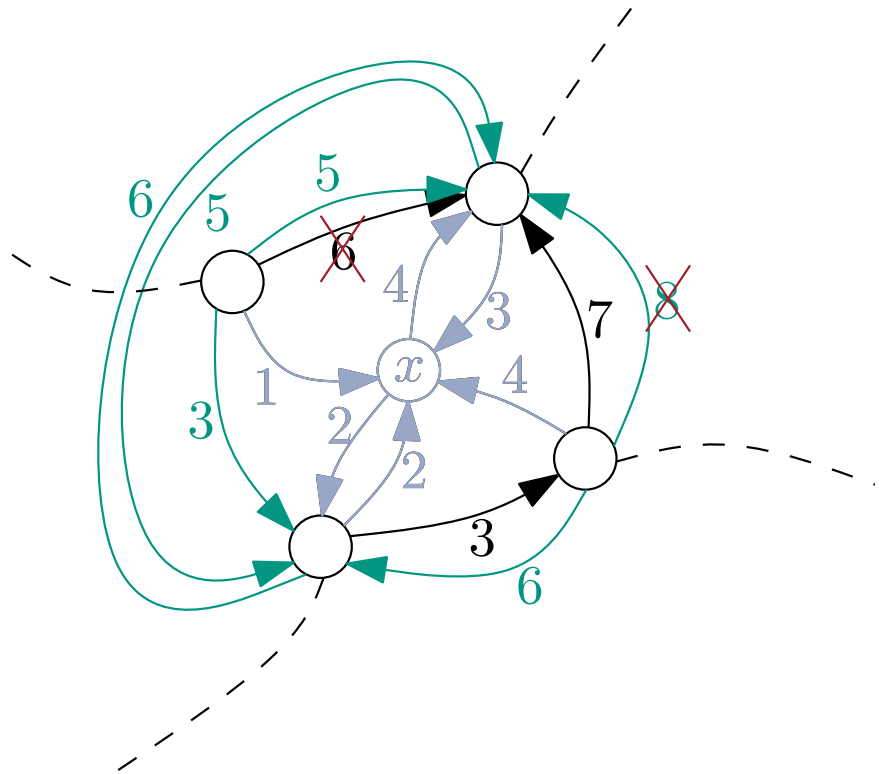
Contraction of x: Delete x and add shortcut edges between neighbors to maintain the distances between all nodes.

Contraction of x



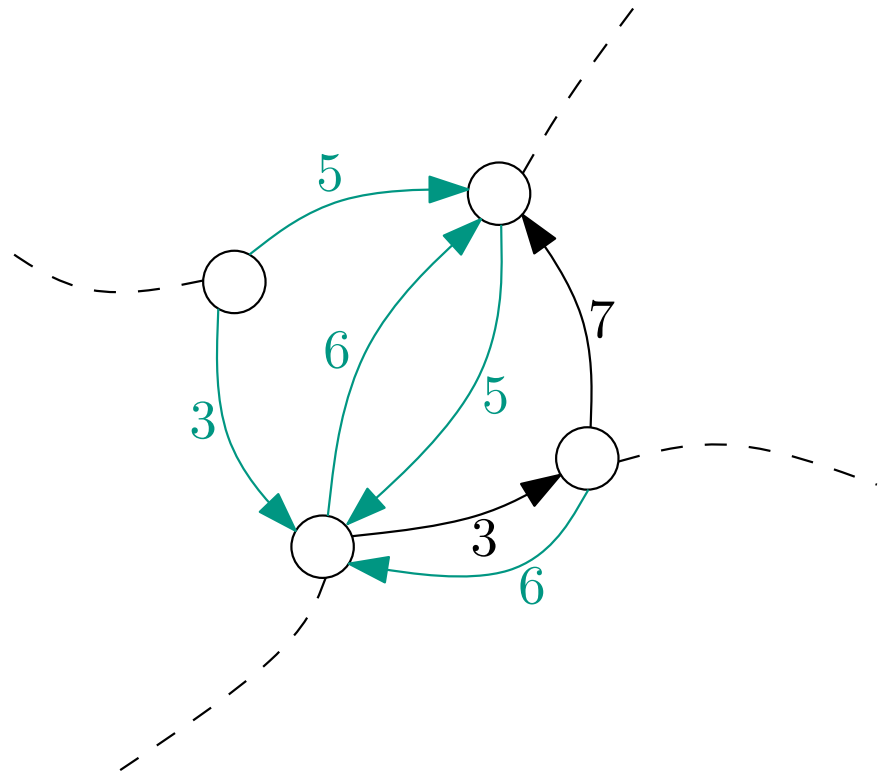
Contraction of x: Delete x and add shortcut edges between neighbors to maintain the distances between all nodes.

Contraction of x

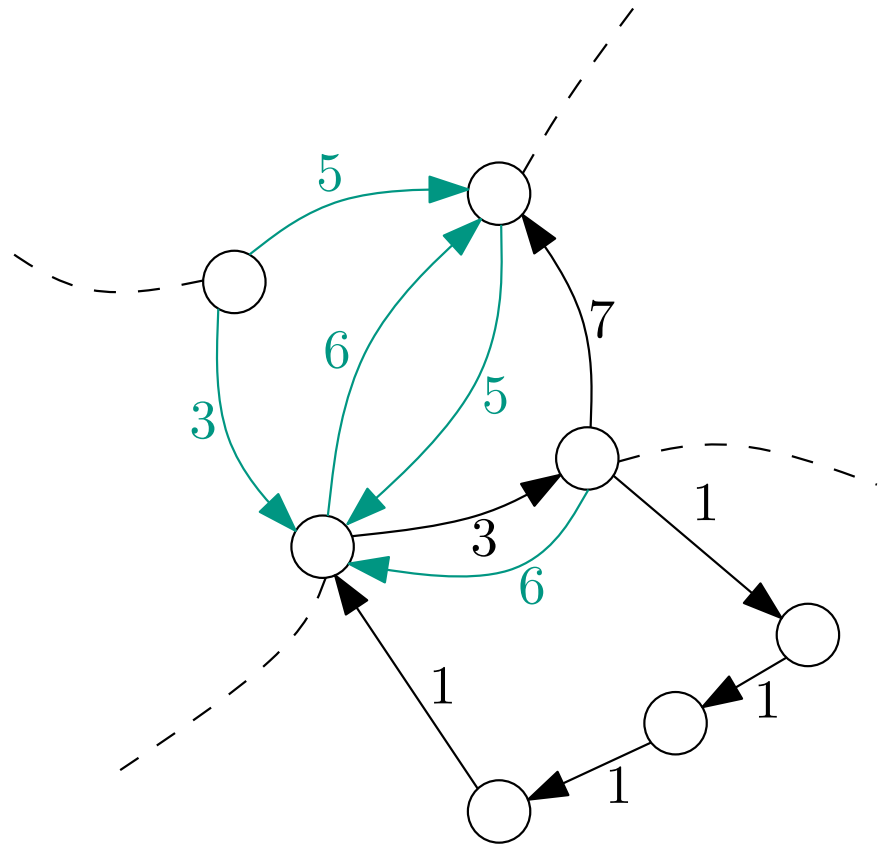


In case of multiple edges: Discard the longer edges.

Contraction of x



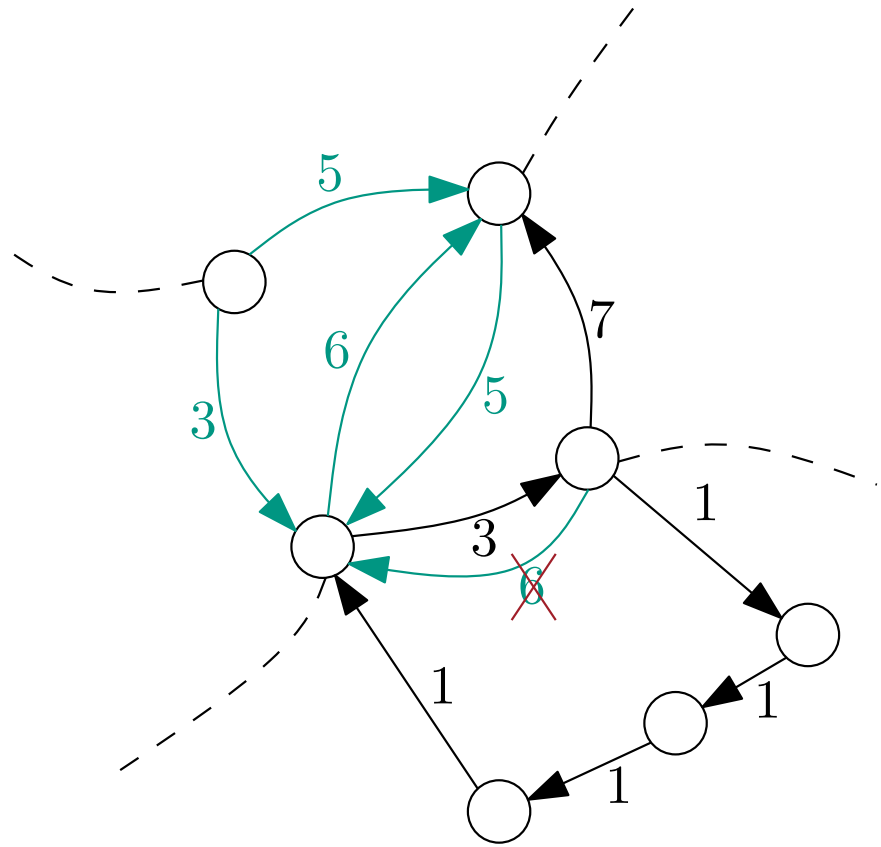
Contraction of x



If there is a shorter path through the remaining graph, a shortcut can be discarded as well.

Searching for such a path is called *Witness Search*.

Contraction of x



If there is a shorter path through the remaining graph, a shortcut can be discarded as well.

Searching for such a path is called *Witness Search*.

Witness Search

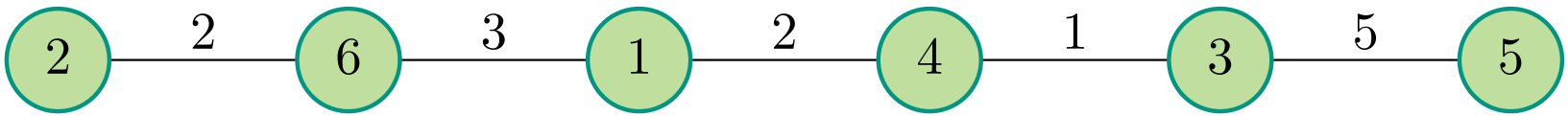
- Let y and z be two neighbors of the contracted node x
- We add a shortcut (y, z) with weight $\text{len}(y, x) + \text{len}(x, z)$ if $y \to x \to z$ is the only shortest y–z-path
- To check if there is a shorter path, start a Dijkstra search from y to z. This search can be expensive. Possible optimizations:
  - The search must not pass through node x
  - Bidirectional version of Dijkstra's algorithm
  - If the searches meet, we can stop
  - If the search front exceeds $\text{len}(y, x) + \text{len}(x, z)$, we can stop
- If this is still too slow: terminate the search after k steps. There may be a path we don't find, which leads to superfluous shortcuts, but this is not a problem in terms of correctness.

Contraction Hierarchy

Basic Idea:

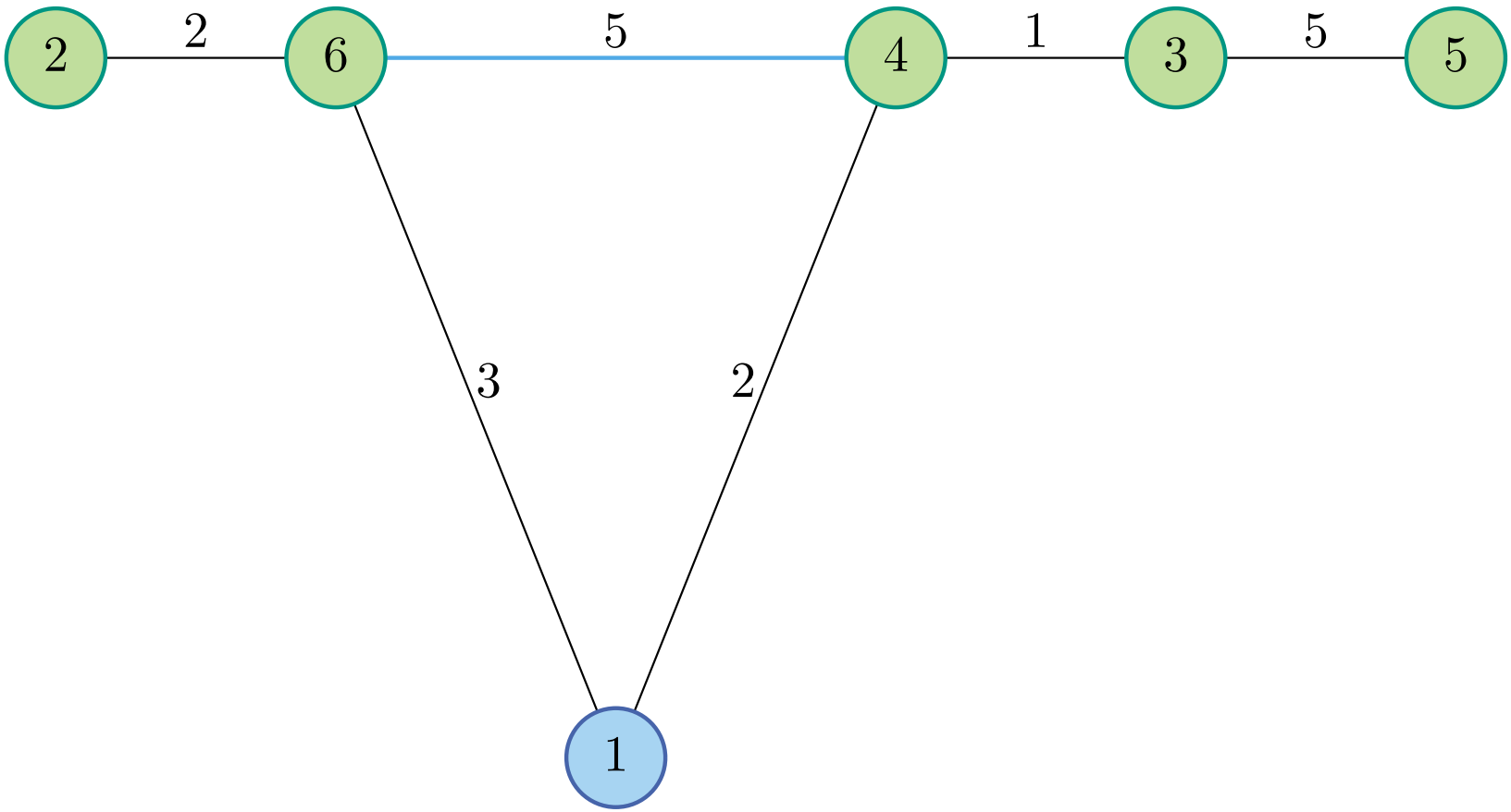
- Input graph G
- Rank the nodes of G by “importance”: $v_1 \ldots v_n$
- Iteratively contract nodes from G
  - First the “least important” node v_1
  - The “most important” node v_n last
- The graph with shortcuts is called the augmented graph

Contraction Hierarchy



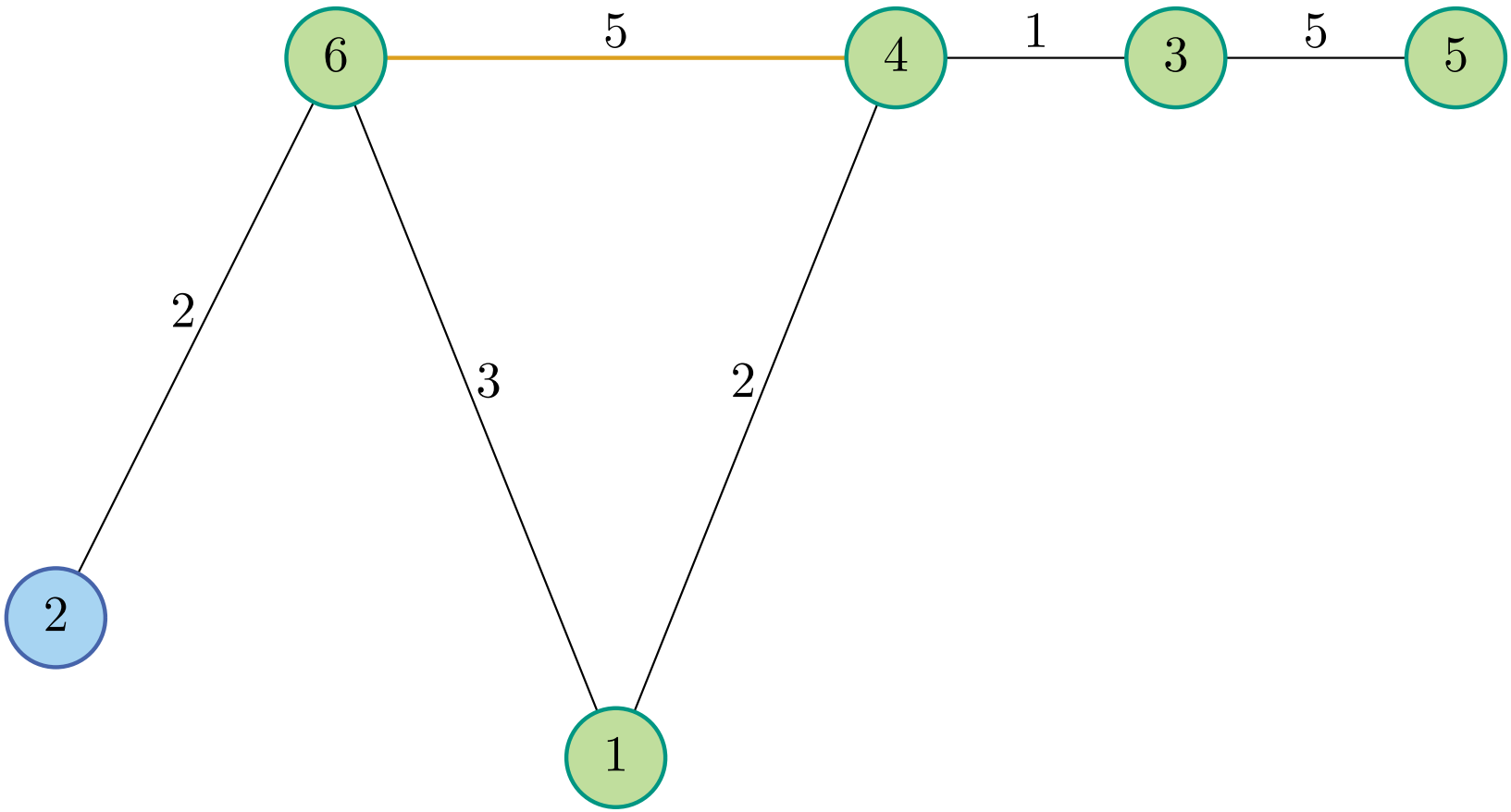
Nodes labeled by “importance”

Contraction Hierarchy



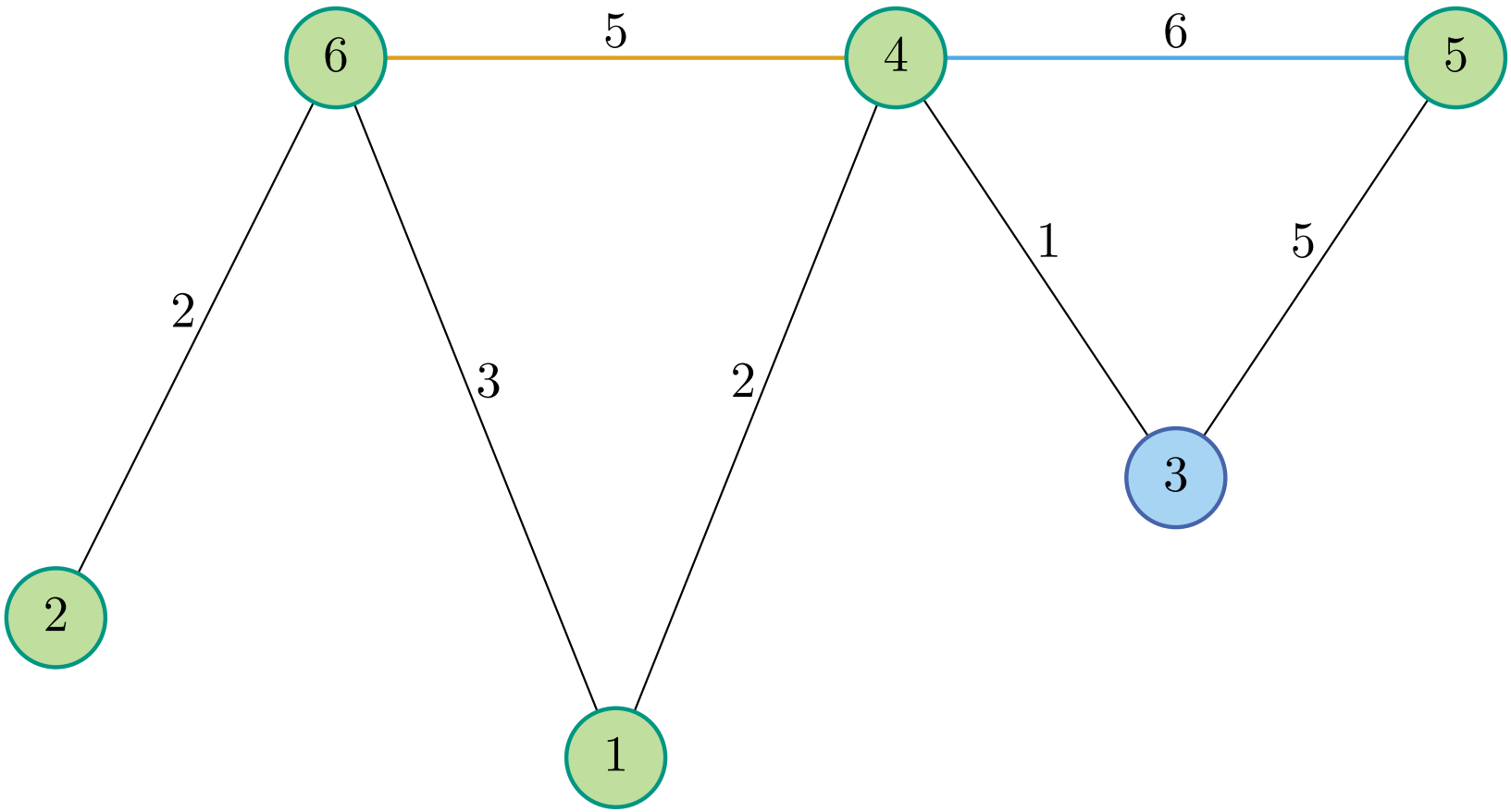
Nodes labeled by “importance”

Contraction Hierarchy



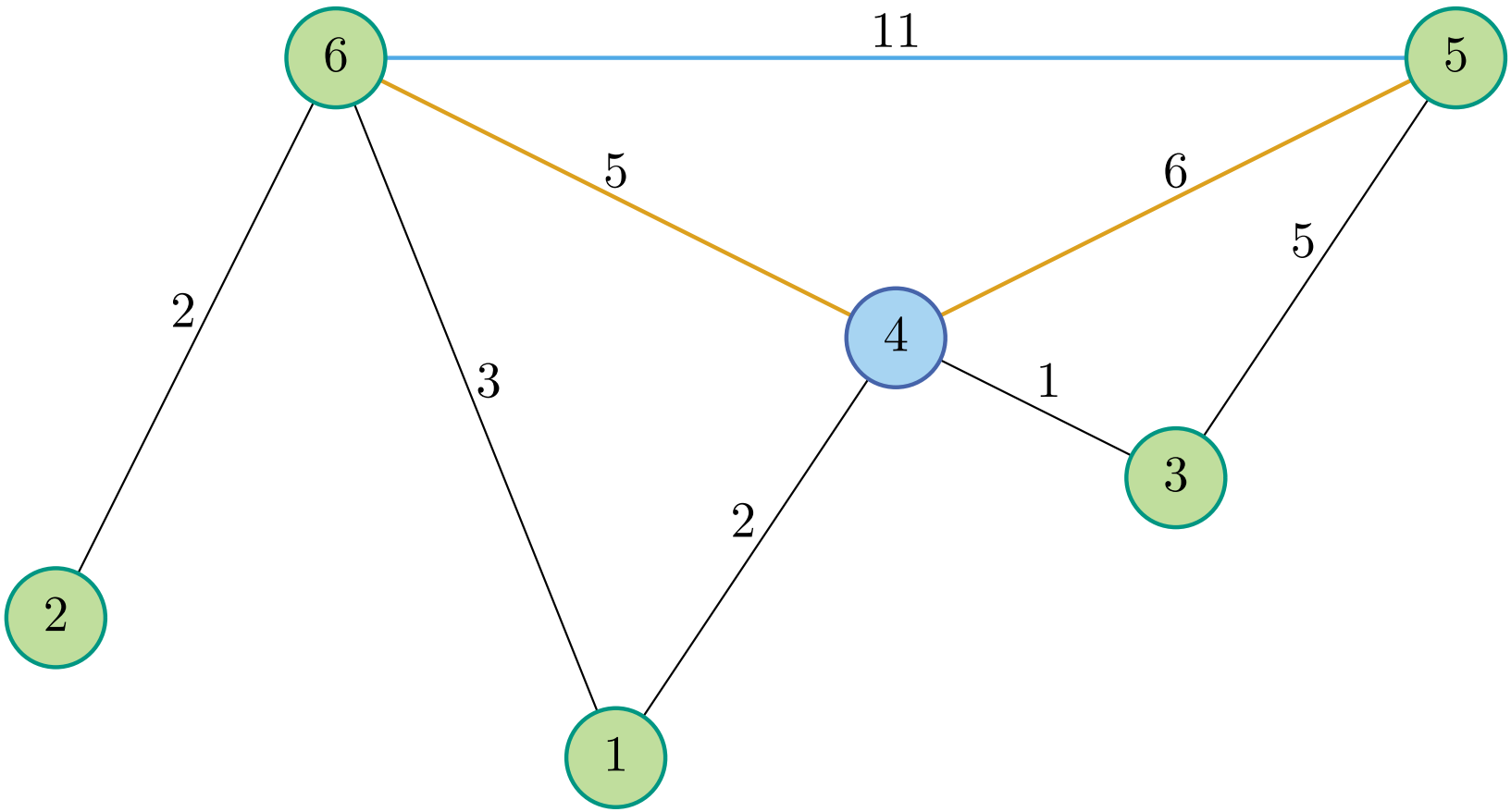
Nodes labeled by “importance”

Contraction Hierarchy



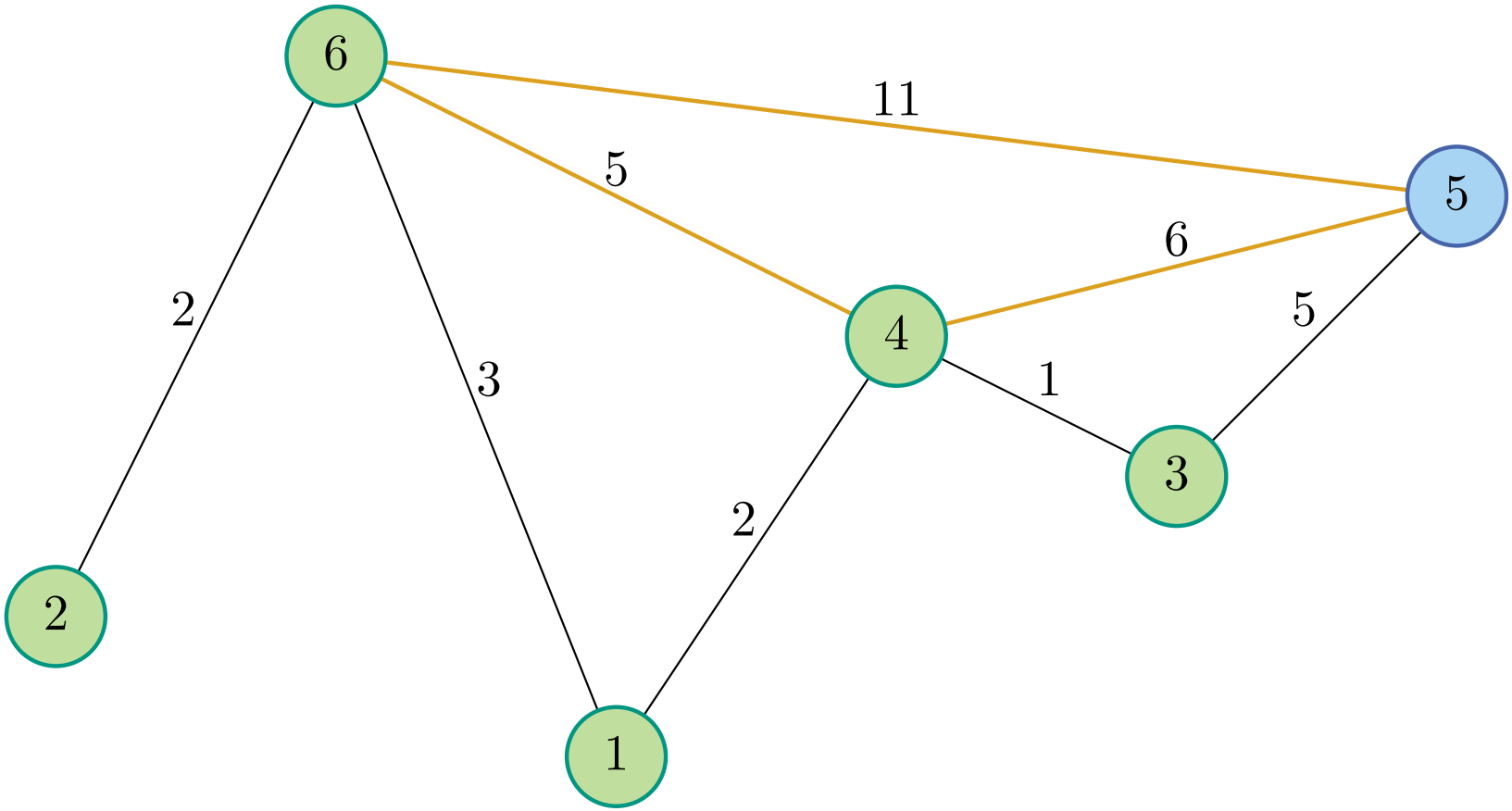
Nodes labeled by “importance”

Contraction Hierarchy



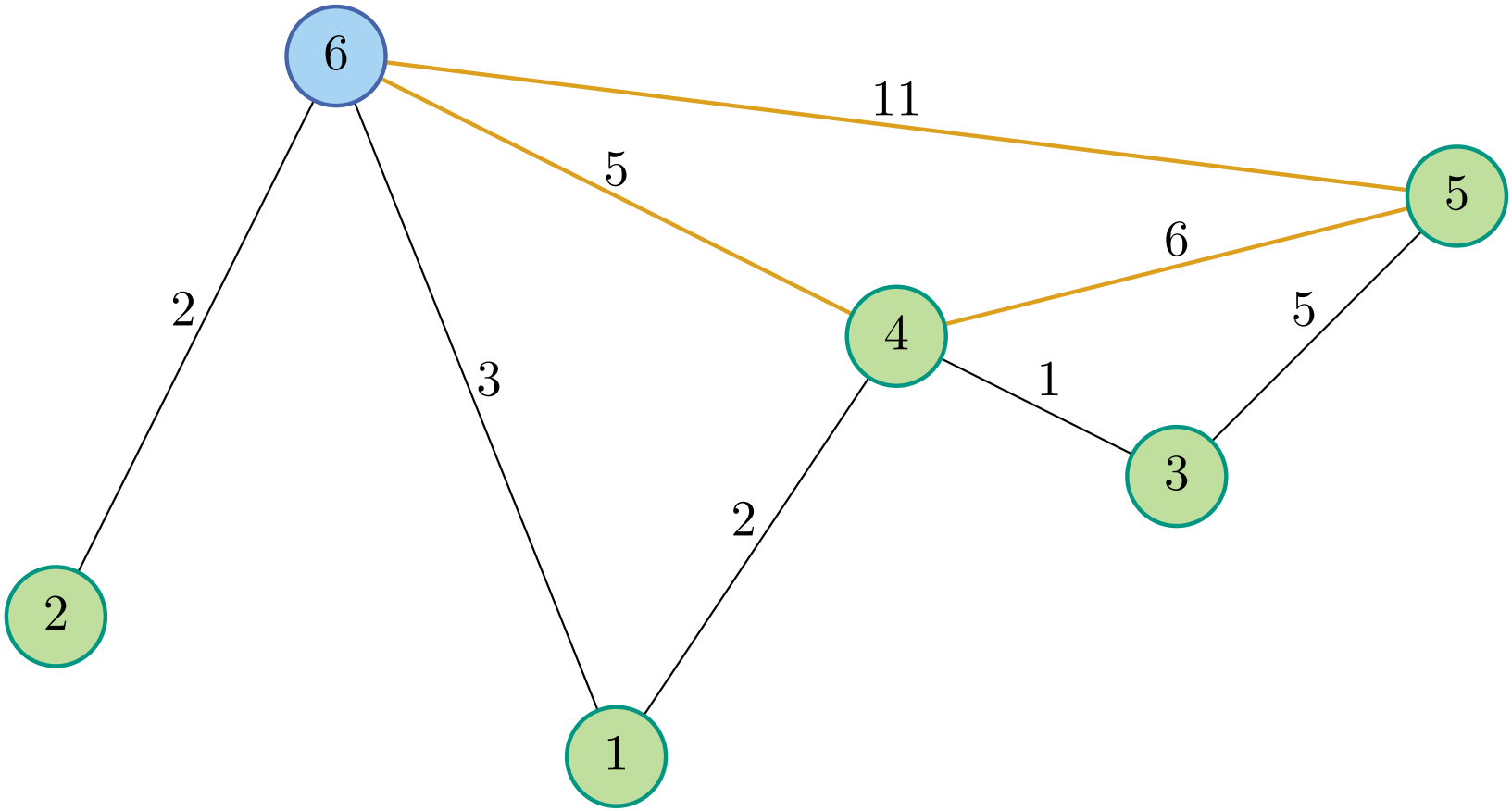
Nodes labeled by “importance”

Contraction Hierarchy



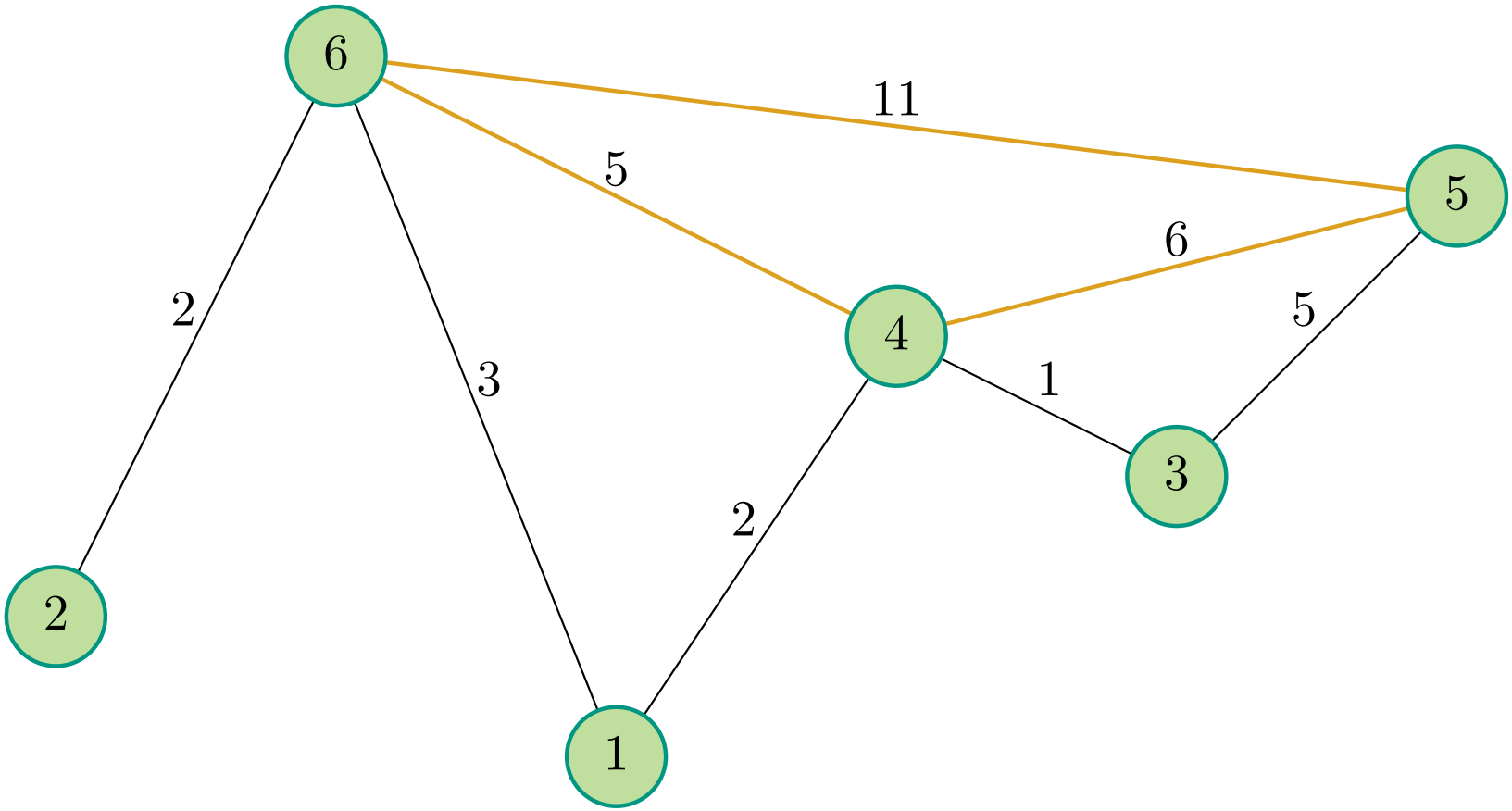
Nodes labeled by “importance”

Contraction Hierarchy



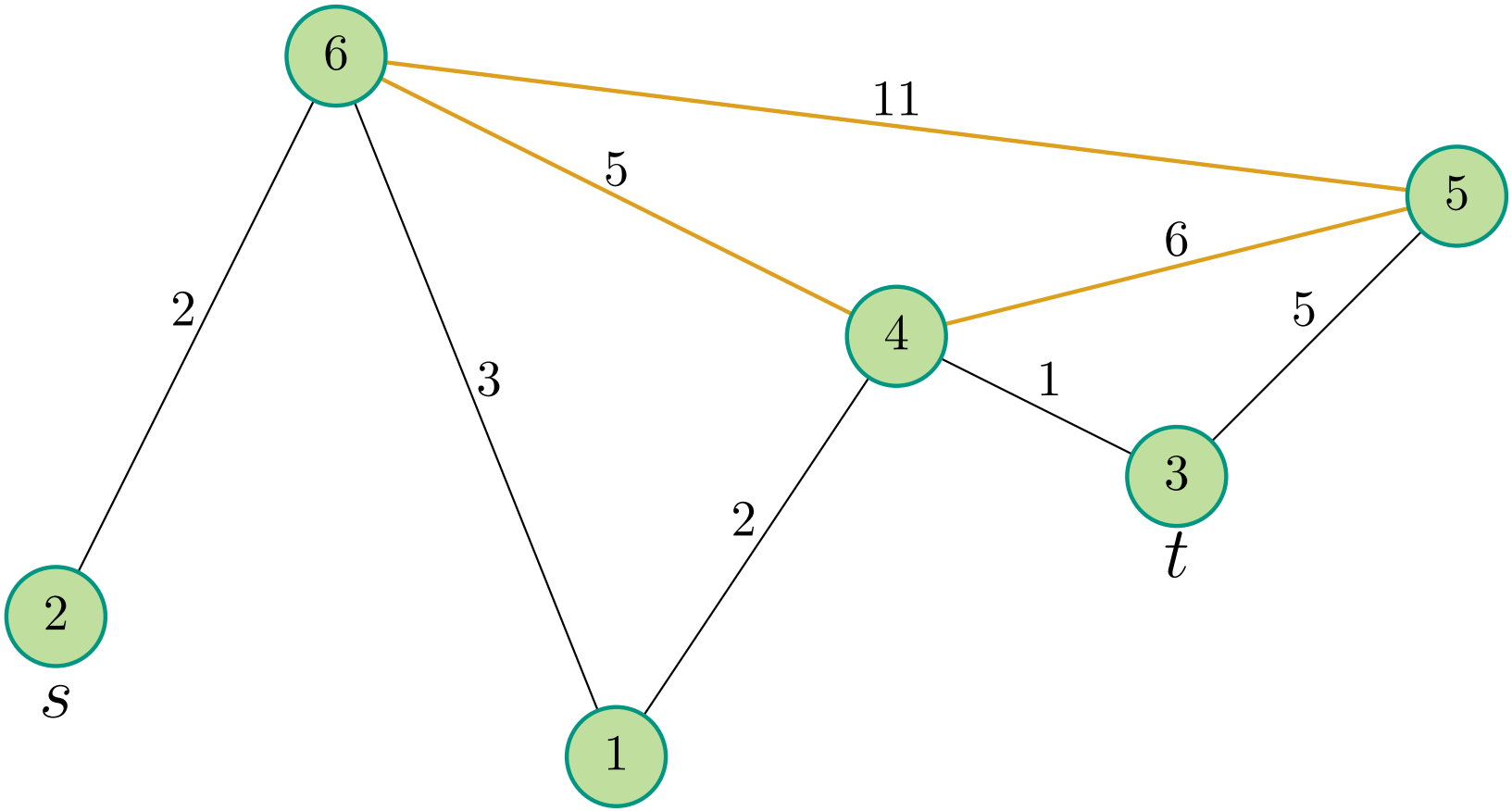
Nodes labeled by “importance”

Contraction Hierarchy



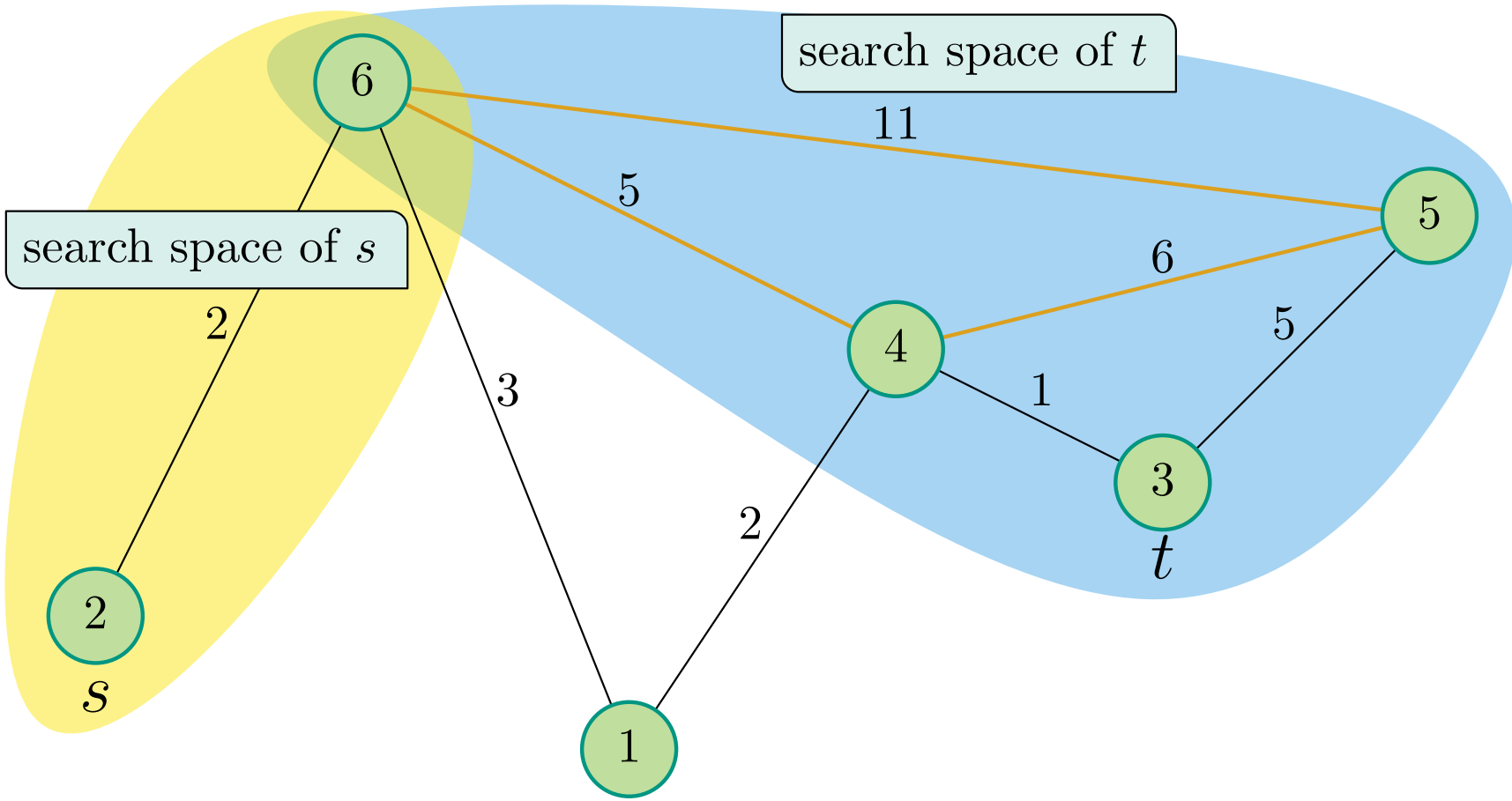
Nodes labeled by “importance”

Contraction Hierarchy



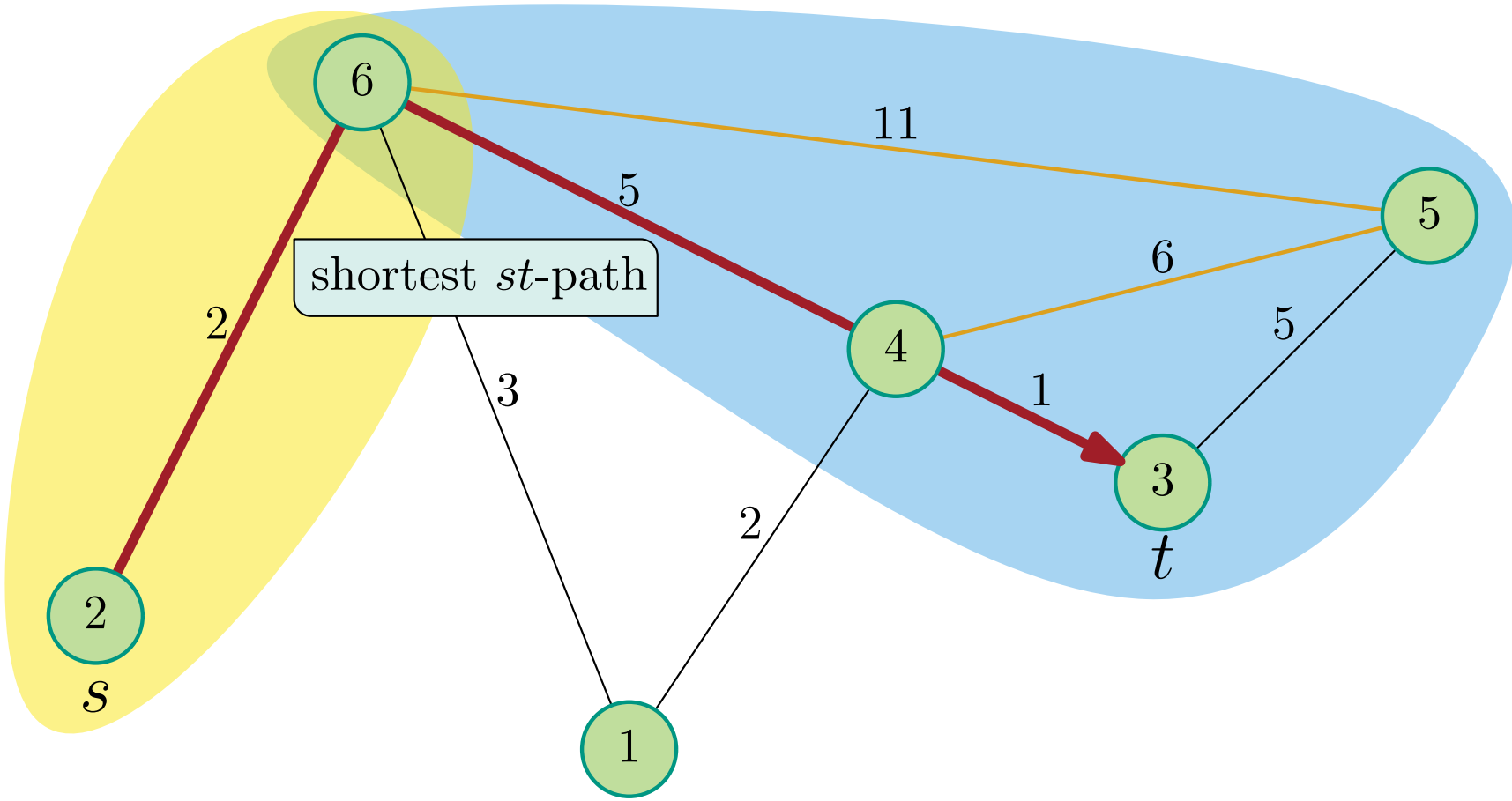
Nodes labeled by “importance”

Contraction Hierarchy



Nodes labeled by “importance”

Contraction Hierarchy



For every original shortest path, there is a up-down-path

Query

- Bidirectional version of Dijkstra's algorithm
- Only consider edges to more important nodes
- Forward search finds the "up" part of the path
- Backward search finds the "down" part of the path
- Stop when the min-key of both queues is greater than the shortest path found so far

Ordering by “Importance”

Karlsruhe Institute of Technology

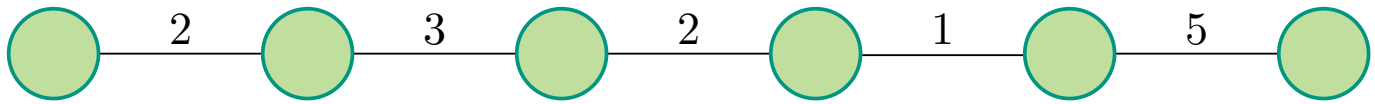
Basic Idea:

- We want few shortcuts
- A node is “unimportant” if it generates few shortcuts
- → Simulate node contraction to weight nodes

Algorithm:

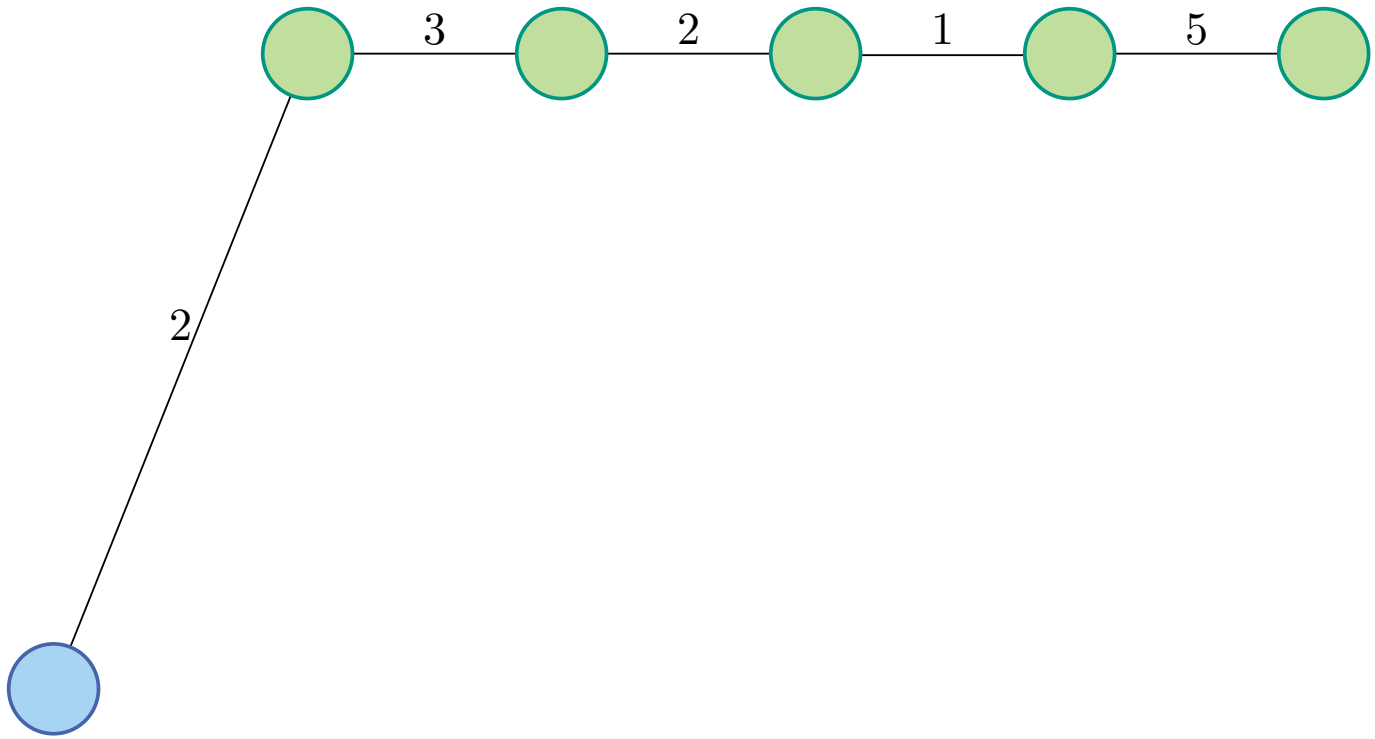
- Build a large queue with all nodes sorted by their “importance”
- Iteratively contract the least important node
- Contraction of a node can affect the “importance” of its neighbors
- → Recalculate the “importance” of the neighbors

Problem: Path



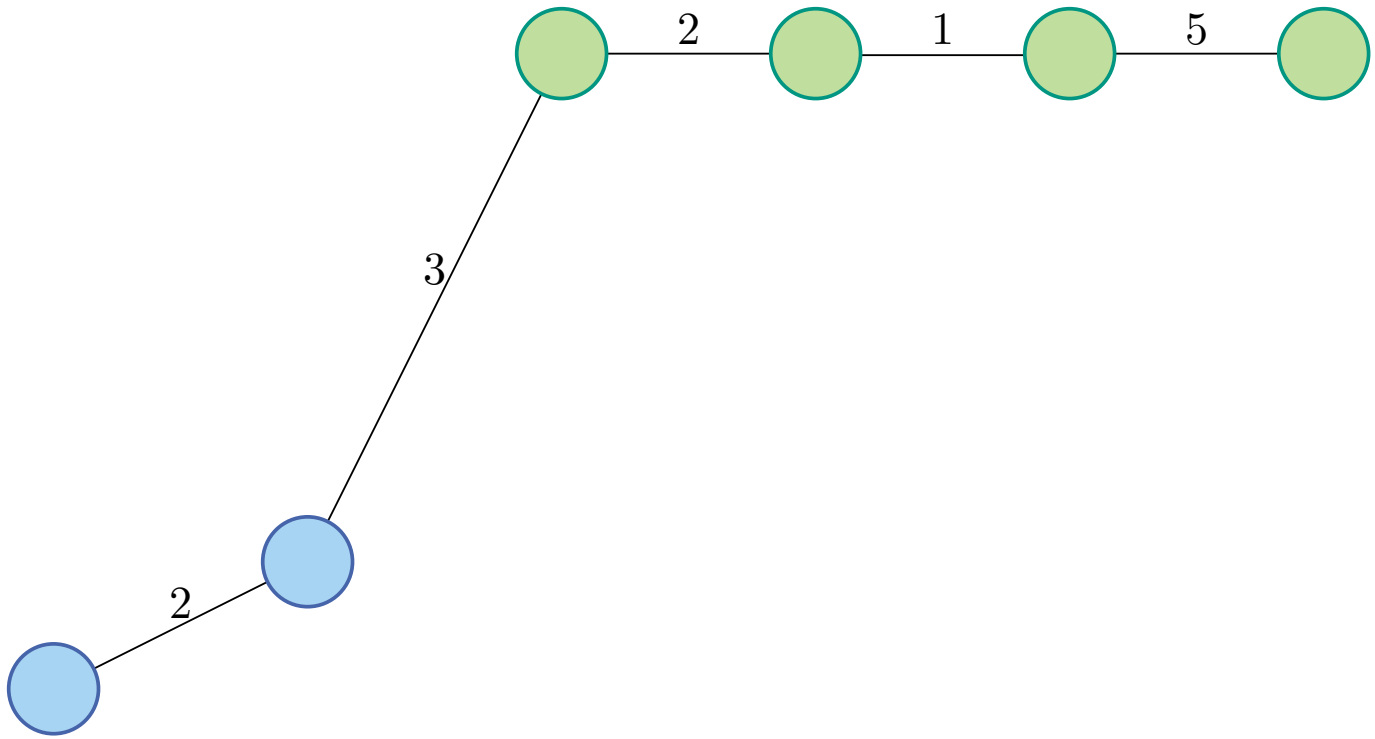
Contracting the left most node does not generate any shortcuts

Problem: Path



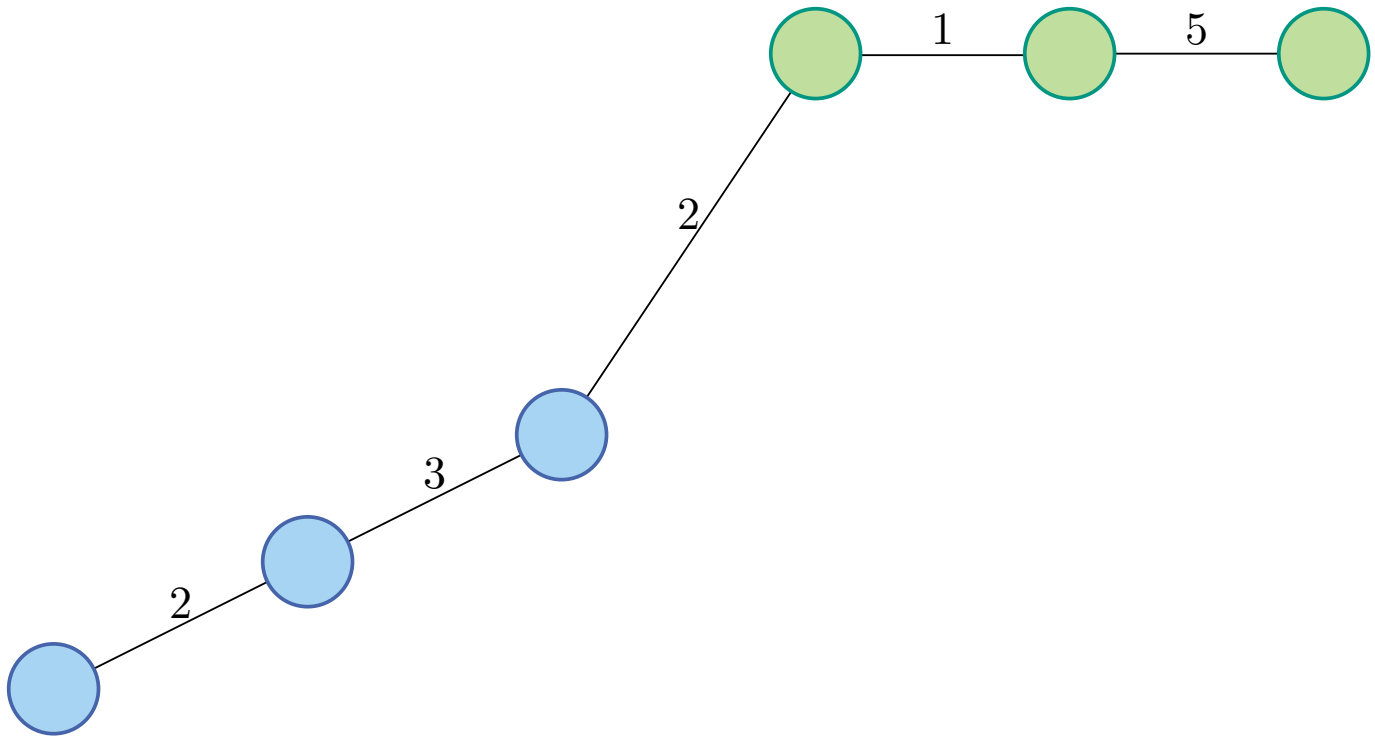
Contracting the left most node does not generate any shortcuts

Problem: Path



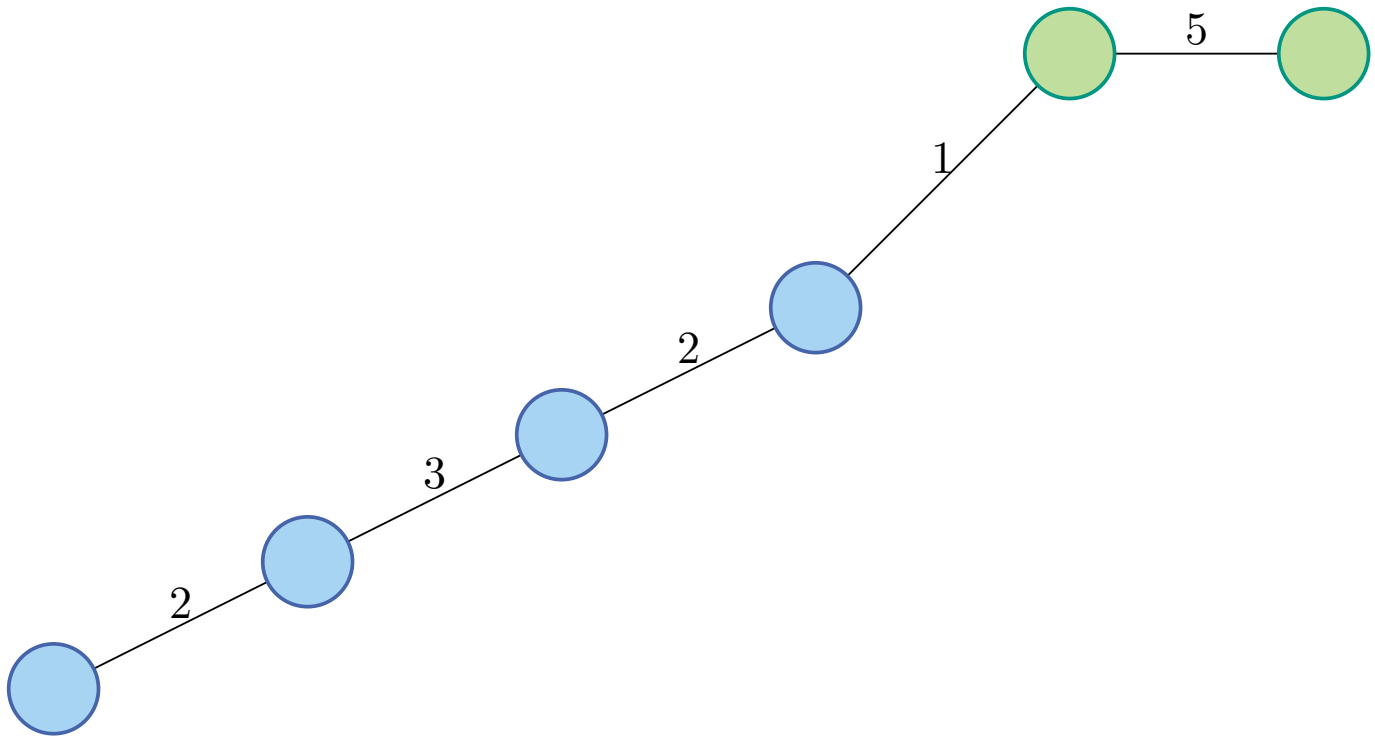
Contracting the left most node does not generate any shortcuts

Problem: Path



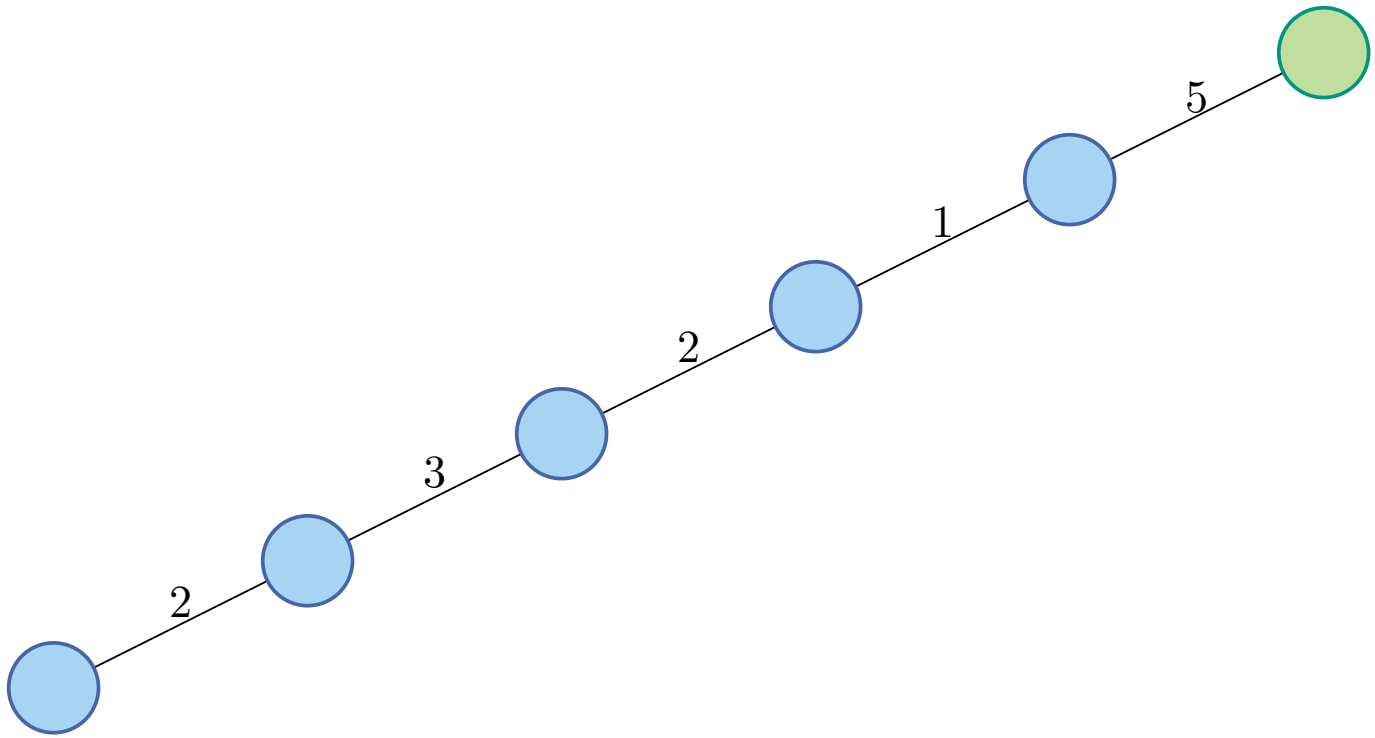
Contracting the left most node does not generate any shortcuts

Problem: Path



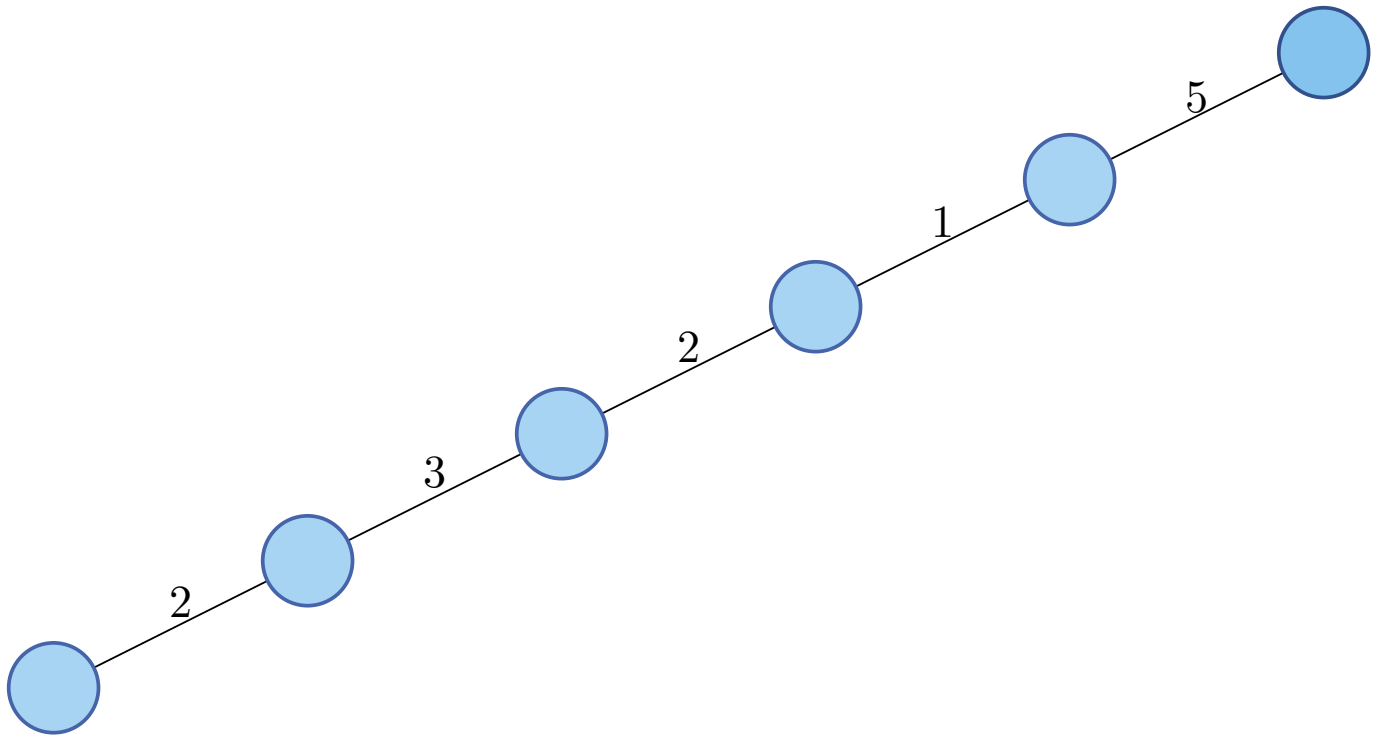
Contracting the left most node does not generate any shortcuts

Problem: Path



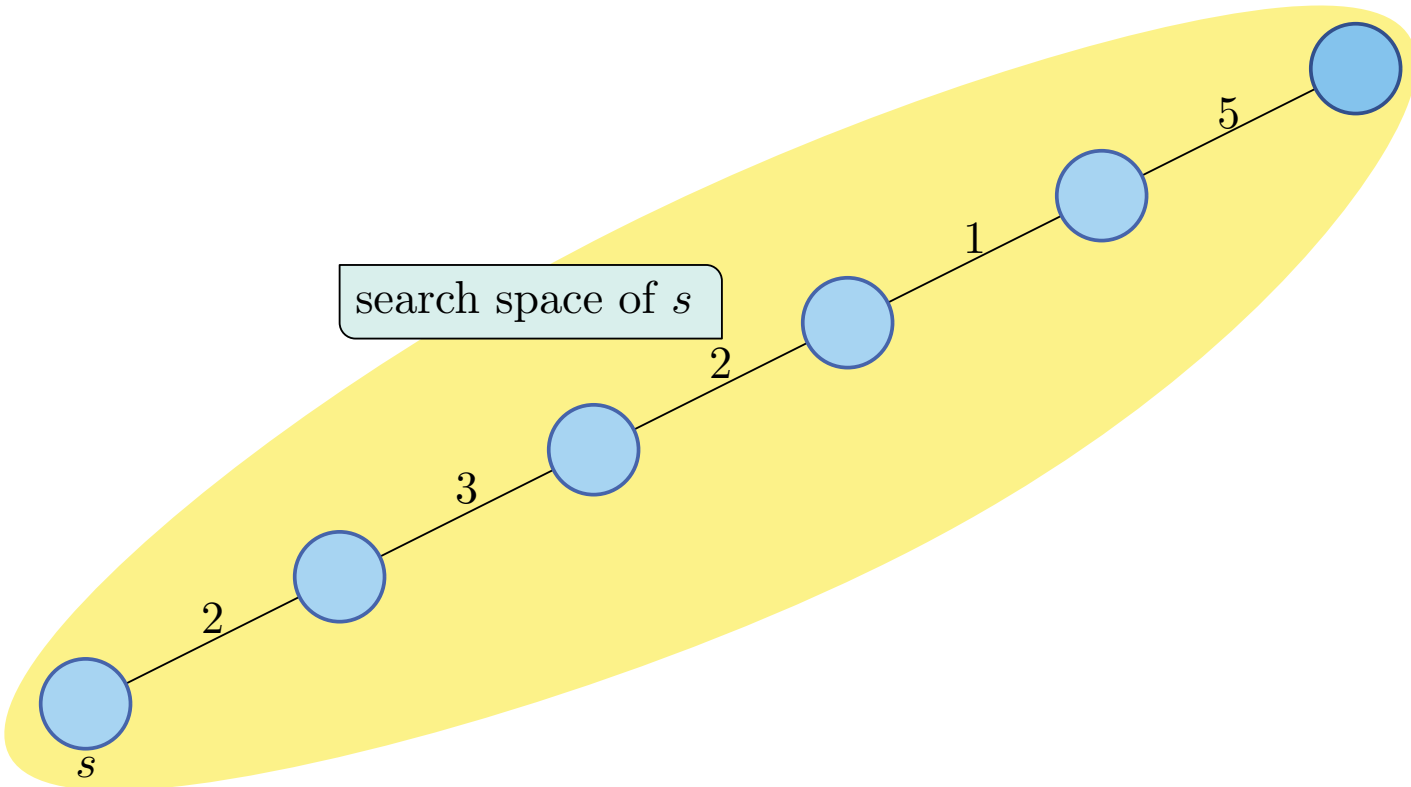
Contracting the left most node does not generate any shortcuts

Problem: Path



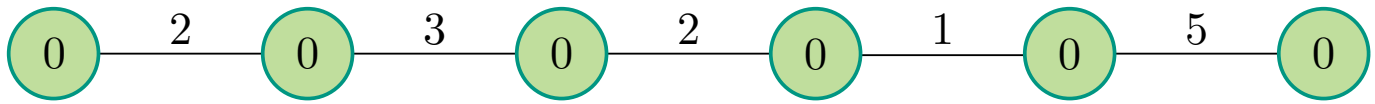
Contracting the left most node does not generate any shortcuts

Problem: Path



The search space from s is the entire graph → no acceleration

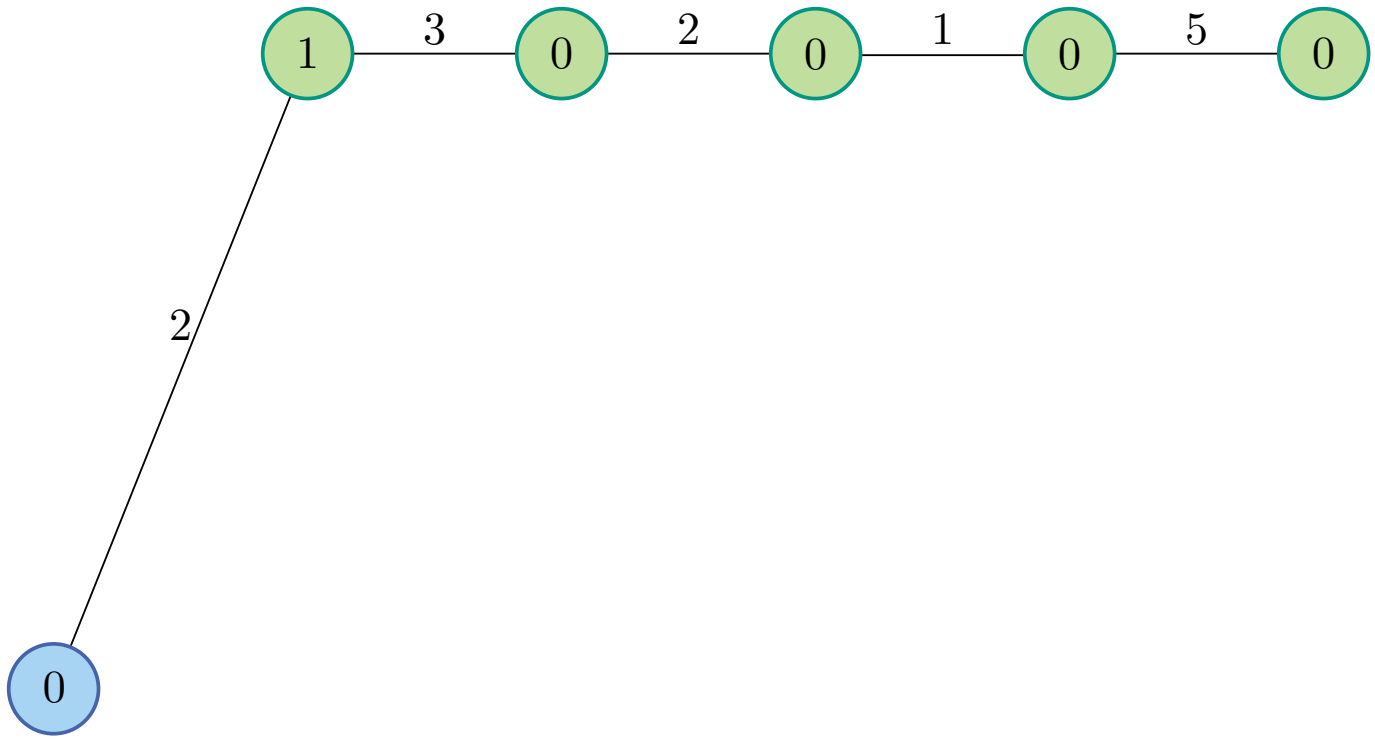
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

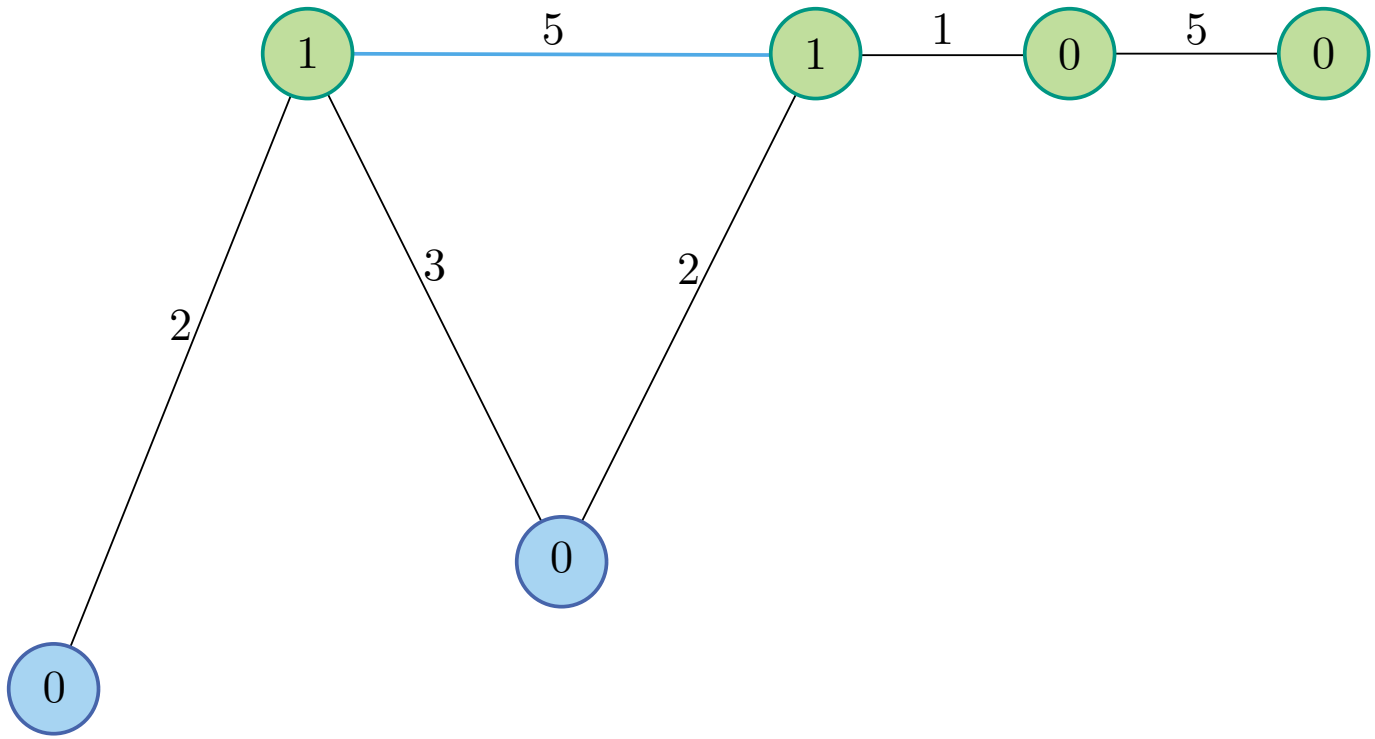
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

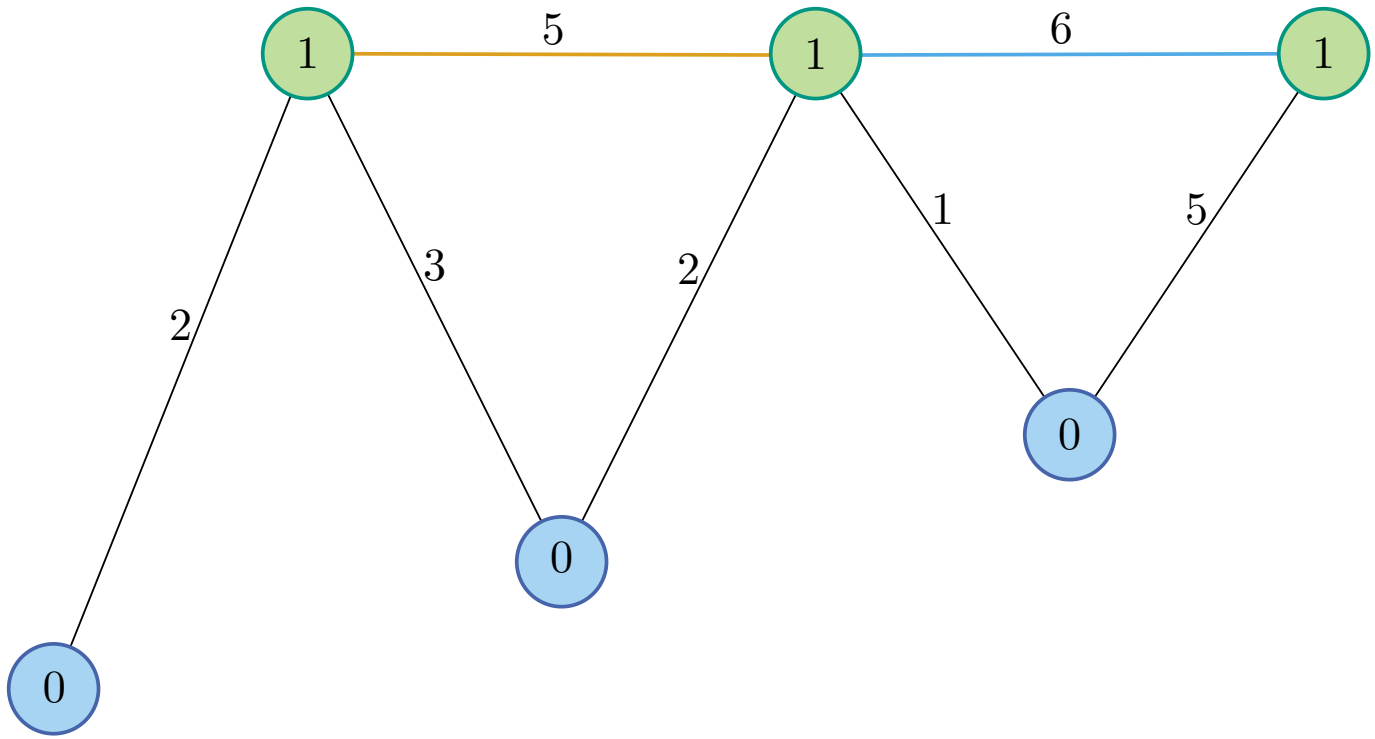
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

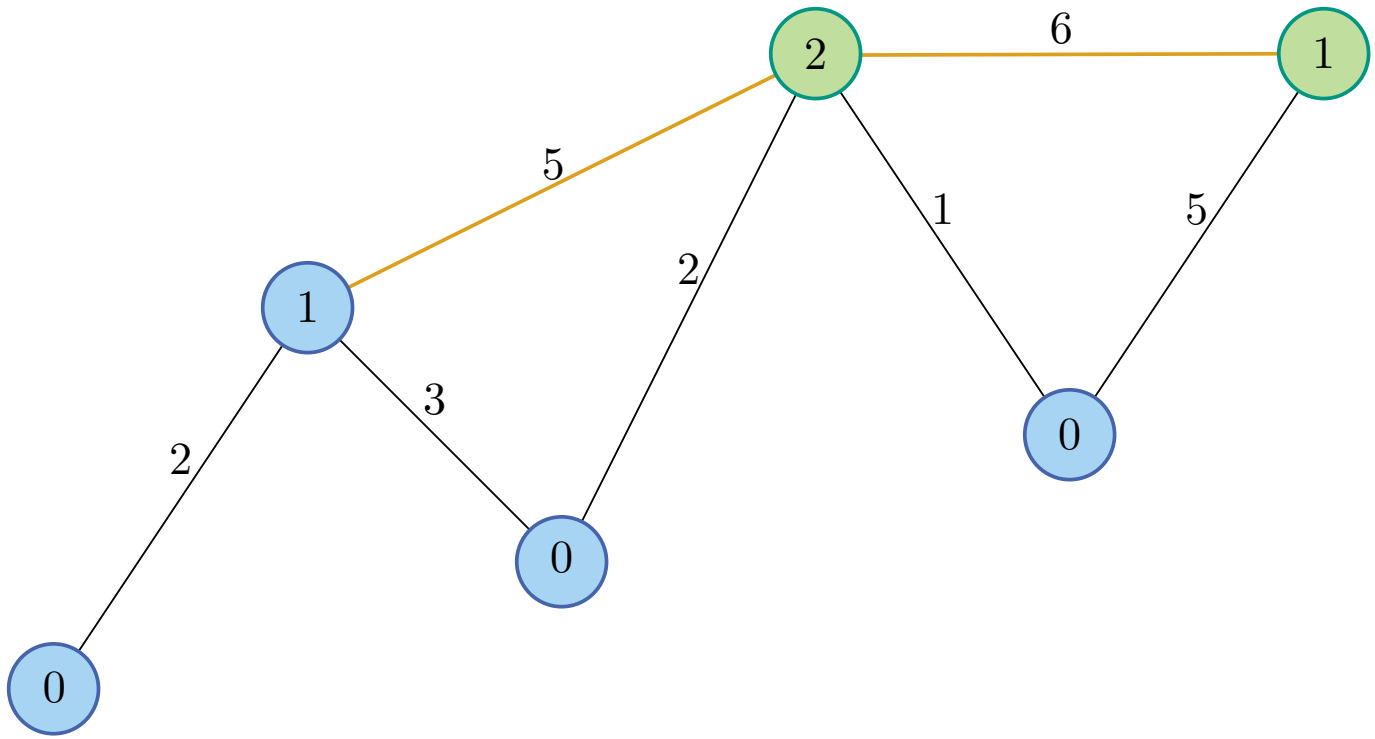
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

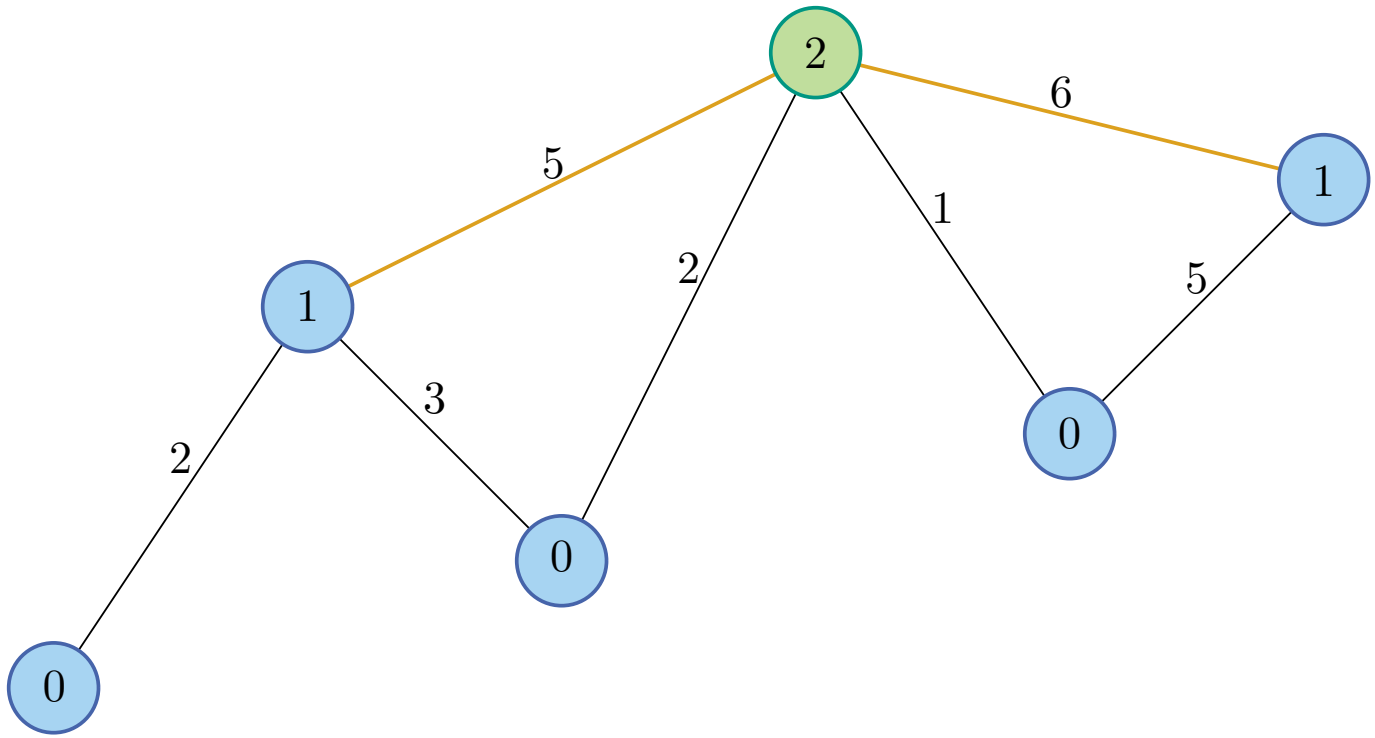
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \longrightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

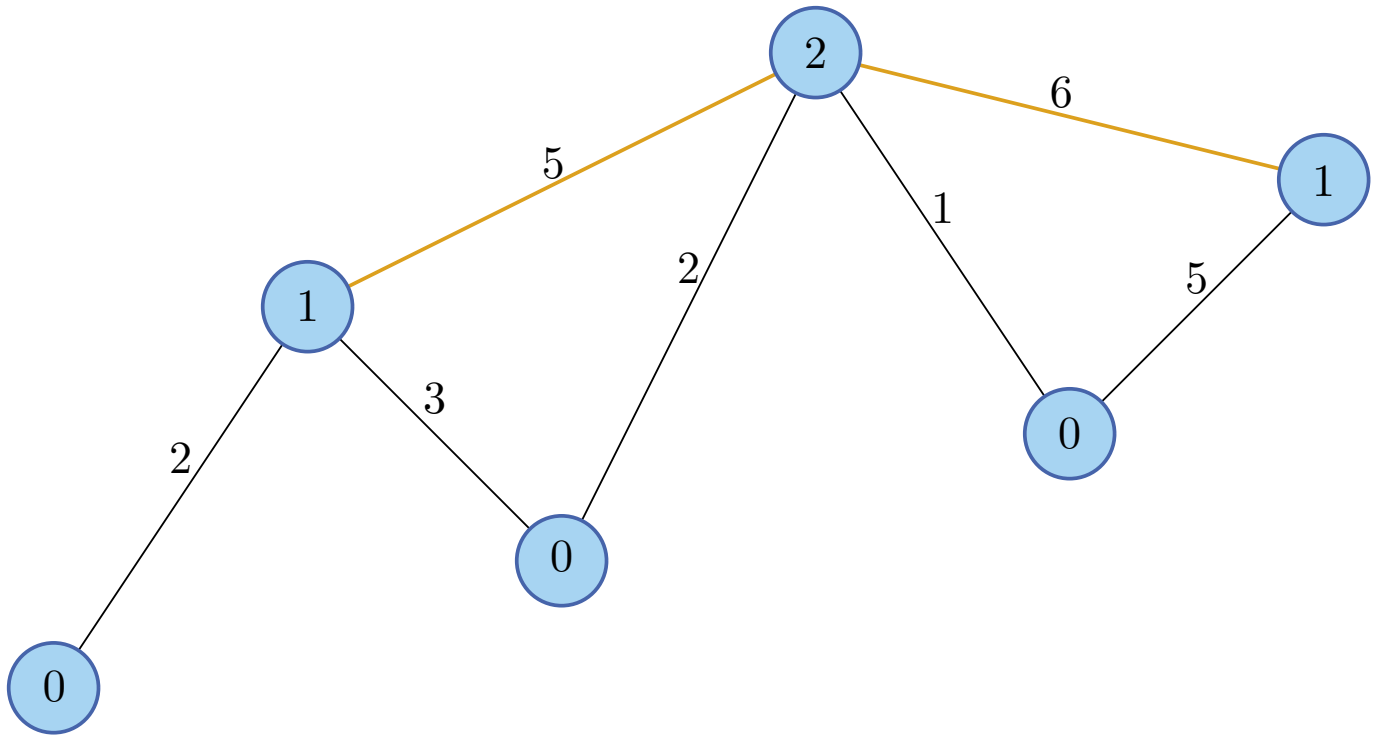
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

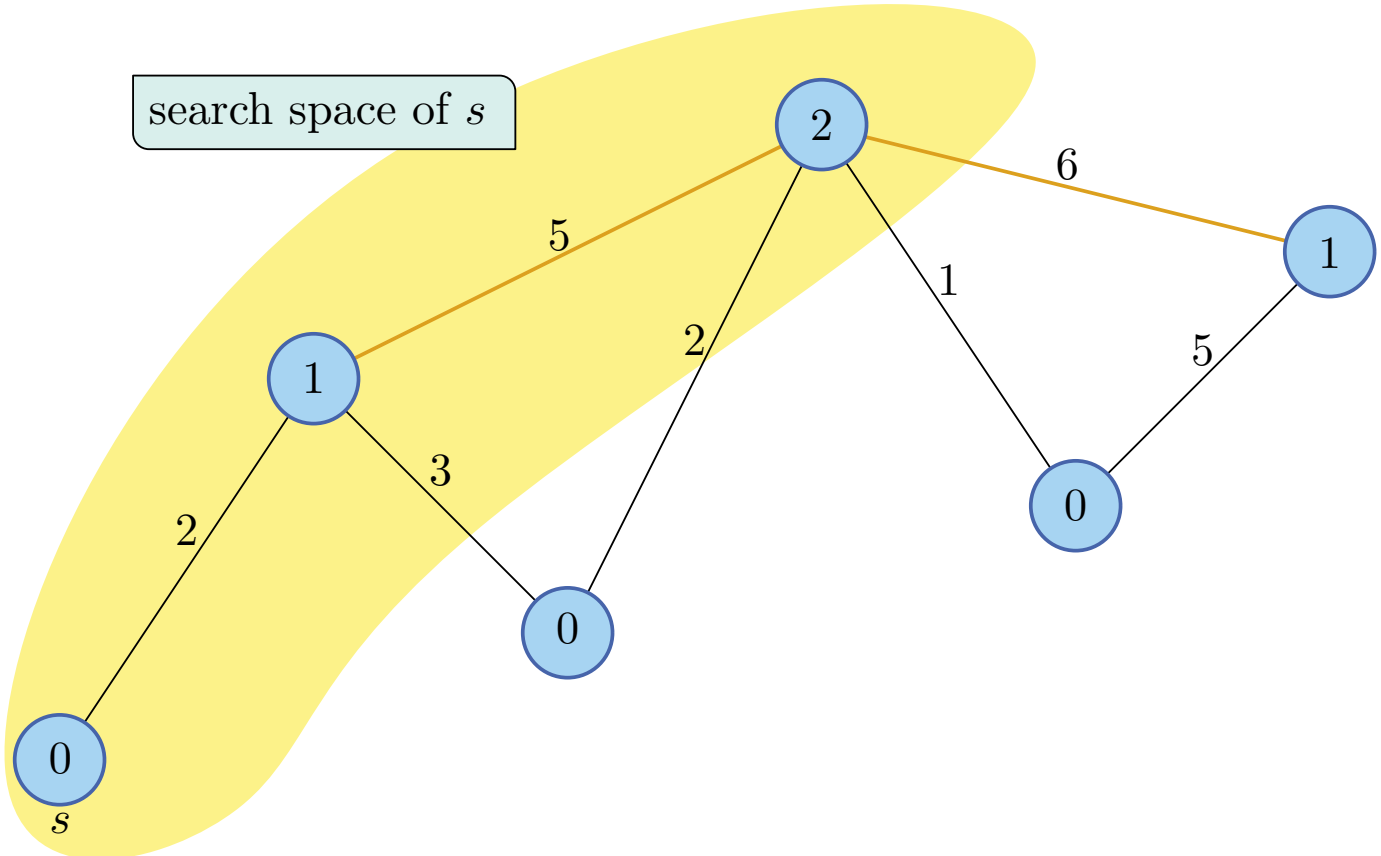
Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problem: Path



2nd criterion: The estimated level $\ell(x)$ of a node

Contract $x \rightarrow$ for all neighbors y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Combining Multiple Criteria

- For each edge e, store the number $h(e)$ of original edges it consists of
- Let $A(x)$ be the set of inserted shortcuts if x were contracted
- Similarly, let $D(x)$ be the set of deleted edges
- Let $I(x)$ be the "importance" of x

A working definition of $I(x)$ is

$$I(x) := \ell(x) + \frac{|A(x)|}{|D(x)|} + \frac{\sum_{e \in A(x)} h(e)}{\sum_{e \in D(x)} h(e)}$$

Note: There are many different definitions for I. This is just a recipe that has proven effective, and everyone may tweak it slightly differently.

Terms

- A graph with shortcuts is called an augmented graph
- An ordering π is a permutation of the nodes such that the nodes are contracted in the order $\pi(0), \pi(1) \ldots \pi(n-1)$.
- The inverse permutation π^{-1} is called the rank. The rank corresponds to the “height” of a node in the CH.