Algorithms and Datastructures
Shortest Path, Dijkstra Algorithm

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science

Algorithms and Datastructures, March 2018

Structure

Structure

Graphs

Paths

For a graph $G = (V, E)$:

- A path of G is a sequence of edges $u_1, u_2, \ldots, u_i \in V$ with
  - Undirected graph: $\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_{i-1}, u_i\} \in E$
  - Directed graph: $(u_1, u_2), (u_2, u_3), \ldots, (u_{i-1}, u_i) \in E$
- The length of a path is
  - Without weights: number of edges taken
  - With weights: sum of weigths of edges taken

Graphs

Paths

For a graph $G = (V, E)$:

- The shortest path between two vertices u, v is the path $P = (u, \ldots, v)$ with the shortest length $d(u, v)$ or lowest costs
- The diameter of a graph is the longest shortest path

Structure

Dijkstra Algorithm
Shortest Path without Computer

- Wanted: Shortest path from M to all other points
- Place pearls on crossings and clamp strings between them

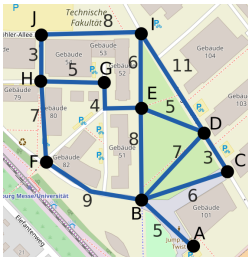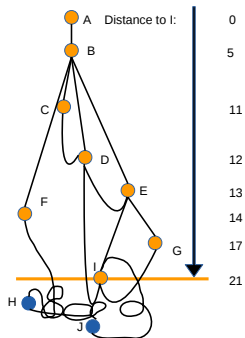# Dijkstra Algorithm
Shortest Path without Computer

Figure: Based on OpenStreetMaps; CC BY-SA 2.0

- Take the net and pull it slowly upwards until fully lifted

- Each node (pearl) now has a specific height
- The distance to M is exactly the shortest path
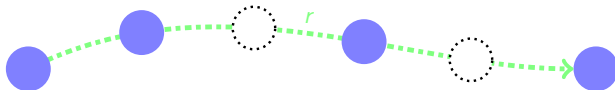
Dijkstra Algorithm

Shortest Path

Figure: Shortest path from *s* to *t*

- Let *r* be the shortest path from *s* to *t*
- For each node *u* on path *r* the path from *u* to *t* is the shortest path

Proof:

- If there was a shorter path from *s* to *u* then we could choose this path to get faster to *t*
- Then *r* would not be the shortest path
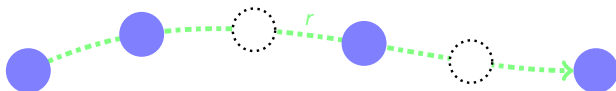
Dijkstra Algorithm

Shortest Path

Figure: Shortest path from s to t

- This is also correct for all sub paths on r
- If the shortest path from s to t passes u_1 and u_2 then the sub path (u_1, u_2) is the shortest path from u_1 to u_2
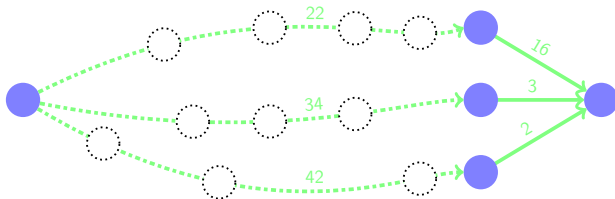
Dijkstra Algorithm

Shortest Path

Figure: Shortest paths from s to t

- If we know the shortest path form s to the preceding nodes of t (v_1, v_2, v_3) we can determine the shortest path to t

Dijkstra Algorithm
Shortest Path

Idea:

- Attach the cost of the shortest path to each node
- Let the information travel over the edges (message passing)
- In which order should we process the nodes?

Dijkstra Algorithm

Inventor:

- ▶ Edsger Dijkstra (1930 - 2002)
- ▶ Computer scientist from Netherlands
- ▶ Won Turing-Award as one of few Europeans for his studies of structured programming
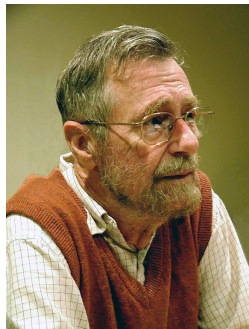- ▶ Invented the Dijkstra-Algorithm in 1959

Figure: Portrait © Hamilton Richards - manuscripts of Edsger W. Dijkstra, University Texas at Austin
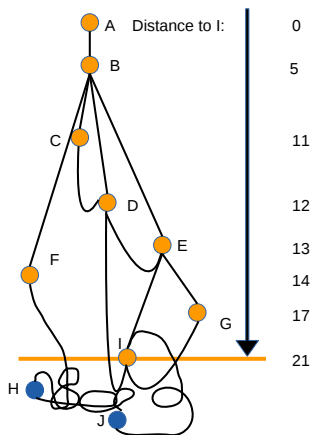
Dijkstra Algorithm

Example:

- ▶ Lift pearl A a little bit
- ▶ Connection to pearl B is hanging in the air
- ▶ Lift further until pearl B starts to lift at 5 m
- ▶ The shortest path to B is now known
- ▶ Lift further: The wires from C, D, E and F are now in the air
- ▶ One of the pearls C, D, E or F is the next one
  Which one?

A   Distance to I: 0

B   5

C   11

D   12

E   13

F   14

G   17

I   21

H

J
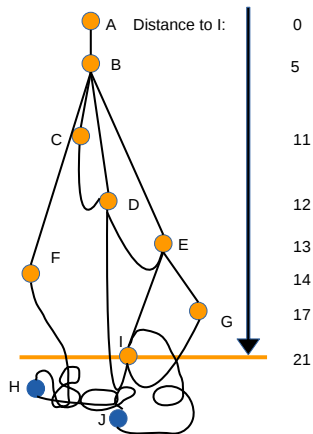
Dijkstra Algorithm

Example:

- At 11 m pearl *C* gets lifted
- The wire to *D* is now in the air
- One of the pearls *D*, *E* and *F* is the next one
  Which one?
- At 12 m pearl *D* gets lifted
  . . .
- How to translate this into an computer algorithm?

Dijkstra Algorithm

High level description: Three types of nodes

- Settled: For node u we know $\mathrm{dist}(s, u)$
  (Pearl example: This pearl is hanging in the air)

- Active: For node u we know a tentative distance
  $\mathrm{td}(u) \geq \mathrm{dist}(s, u)$ (Can be optimal but doesn't have to)
  (Pearl example: This pearl is laying on the table but
  one connected wire is already in the air)

- Unreached: We have not reached the node yet
  (Pearl example: This preal is hanging in the air)

Dijkstra Algorithm

High level description:

- Each iteration take the active node u with the smallest $\mathrm{td}(u)$
  (The pearl getting lifted next)

- We update the state of the node u to settled
  (The pearl gets lifted)

- We check for each neighbor v of node u if we can reach v
  faster than currently possible
  (Check all outgoing wires from this pearl: Activate all
  connected pearls, update tentative distance if smaller)

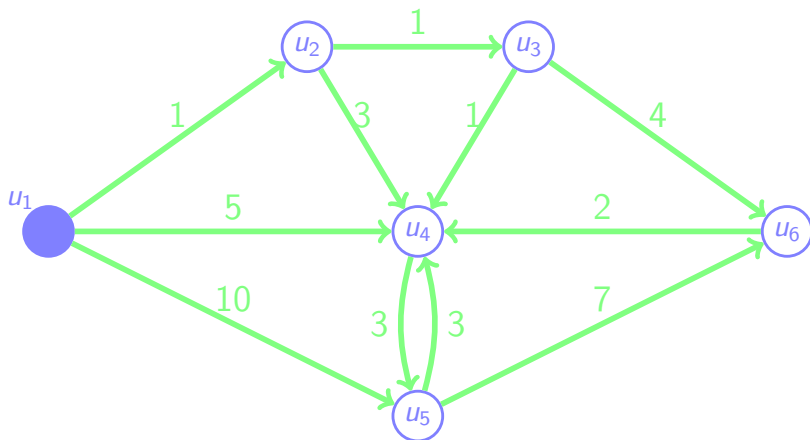- Iterate until no active nodes exist anymore

Dijkstra Algorithm

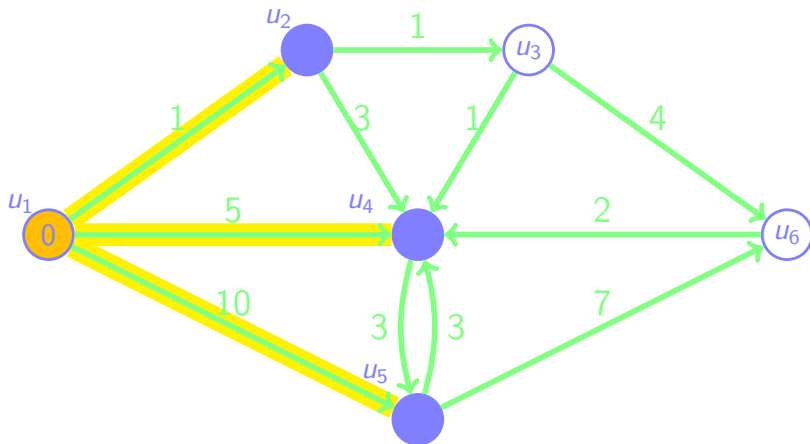Figure: Start at u_1

Dijkstra Algorithm
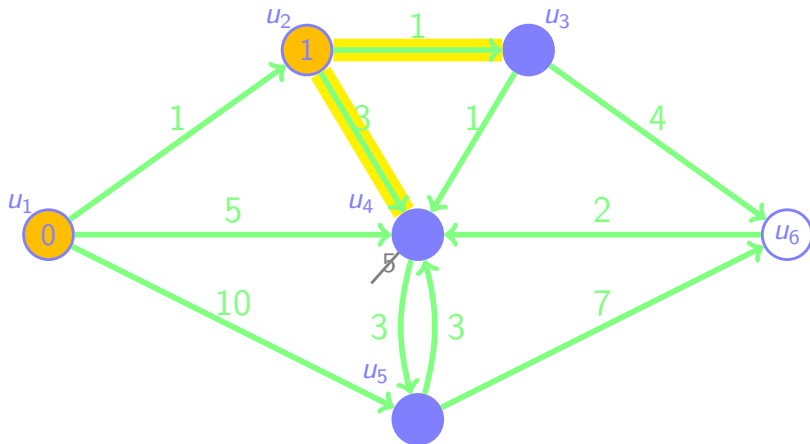

Figure: Iteration 1

Dijkstra Algorithm

Figure: Iteration 2
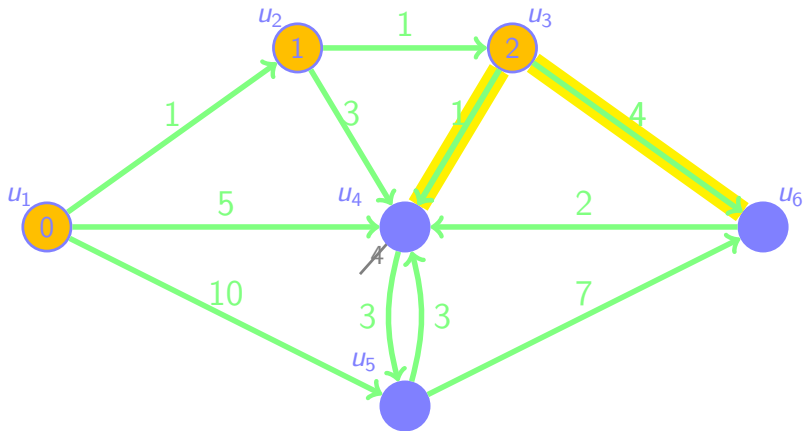
Dijkstra Algorithm
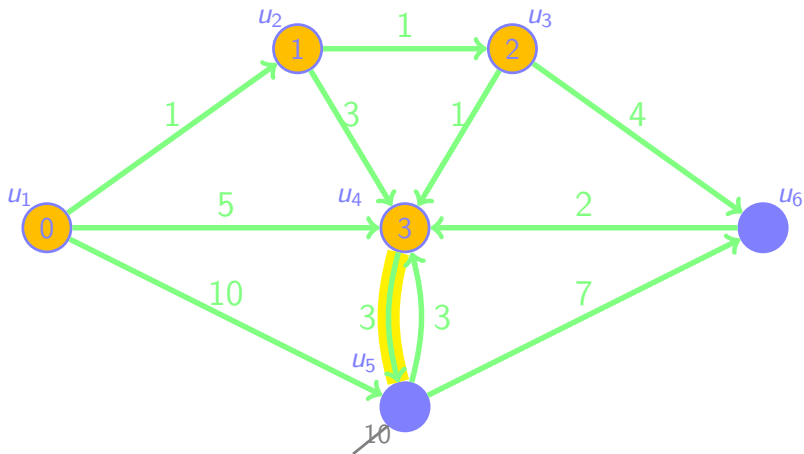

Figure: Iteration 3

Dijkstra Algorithm

Figure: Iteration 4
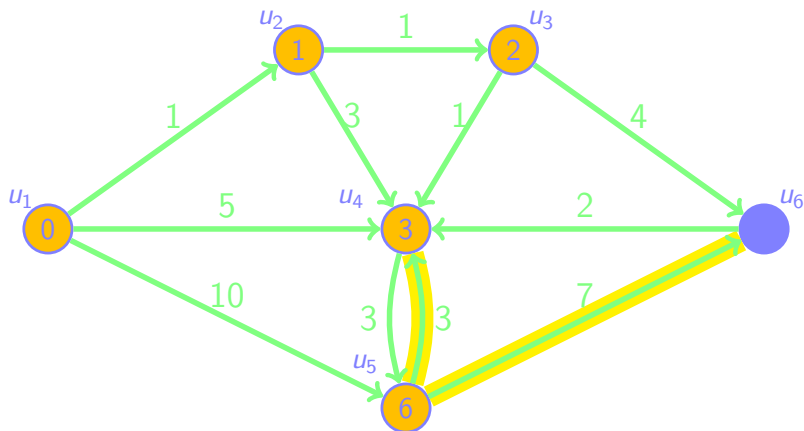
Dijkstra Algorithm
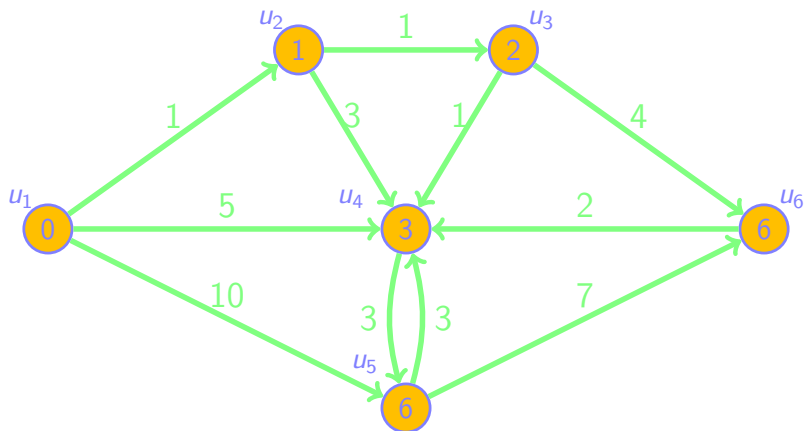

Figure: Iteration 5

Dijkstra Algorithm

Figure: Iteration 6

Dijkstra Algorithm
Proof

Proof:

- **Assumption 1:** All edges have a positive length
- **Assumption 2:** Each node has a unique distance $\mathrm{dist}(s, u)$ to the start s

  (This was not the case on the previous slides)

  This results in an easy and intuitive proof.

  It is possible to show this without assumption 2. See references if interested

- With assumption 2 there exists a sorting u_1, u_2, ... with that:

$$\mathrm{dist}(s, u_1) < \mathrm{dist}(s, u_2) < \mathrm{dist}(s, u_3) < \dots$$

Dijkstra Algorithm

Proof

Proof:

- With **assumption 2** there exists a sorting u_1, u_2, ... with that:
$$\mathrm{dist}(s, u_1) < \mathrm{dist}(s, u_2) < \mathrm{dist}(s, u_3) < \ldots$$

- We want to show that the *Dijkstra* algorithm finds the shortest path for each node u_i so that $\mathrm{td}(u_i) = \mathrm{dist}(s, u_i)$ holds

- Additionally we show that each node gets solved in order of the distance: Node u_i gets solved in iteration i

$$u_1, u_2, u_3, \ldots$$

Dijkstra Algorithm

Proof

To show: Node u_i gets solved in round i

1. Node u_i contains the correct distance ($\text{td}(u_i) = \text{dist}(s, u_i)$) and is active

2. Node u_i has the smallest value for $\text{td}(u_i)$ and gets selected by the algorithm

Induction start:

1. ▸ Only the start node $s = u_1$ is active and $\text{td}(s) = 0$
   ▸ Node u_1 gets solved and $\text{td}(u_1) = \text{dist}(s, u_1) = 0$

2. Only the start node u_1 is active
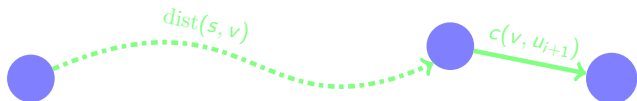
Dijkstra Algorithm
Proof

Induction step: $i = i + 1$

1. **To show:** Node u_i contains the correct distance
   $(\mathrm{td}(u_i) = \mathrm{dist}(s, u_i))$ and is active

   ▸ On the shortest path from s to u_{i+1} is a preceding node that:

   $$\mathrm{dist}(s, u_{i+1}) = \mathrm{dist}(s, v) + \mathrm{c}(v, u_{i+1})$$

   $(\mathrm{c}(v, u_{i+1})$ are the costs of the edge)

   ▸ With that results $\mathrm{dist}(s, v) < \mathrm{dist}(s, u_{i+1})$ because $c > 0$
   ▸ Because u_{i+1} is currently settled, the node v is one of the
     preceding nodes $u_1, \ldots, u_i$, hence $v = u_j$ with $0 \leq j \leq i$

Dijkstra Algorithm

Proof - Example of Iteration 6

- Preceding node of u_6 is $v = u_3$
- In round 3 $\mathrm{td}(u_6) = 2 + 4 = 6$ was already solved

Dijkstra Algorithm
Proof

1. **To show:** Node u_i contains the correct distance $\mathrm{td}(u_i) = \mathrm{dist}(s, u_i)$ and is active
   - With **induction assumption**: v already contains the correct distance which was evaluated in round j (edge from v to u_{i+1}) and is stored in $\mathrm{td}(u_{i+1})$
   - u_{i+1} is active because the preceding node was solved

Dijkstra Algorithm
Proof

2. **To show:** Node u_{i+1} has the smallest value for $\mathrm{td}(u_{i+1})$ and gets selected by the algorithm
   - All nodes with smaller dist are already solved
   - All other nodes u_k with $k > i + 1$ have a greater $\mathrm{dist}(s, u_k)$ and with that the $\mathrm{td}(u_k)$ is greater or equal
   - $\Rightarrow$ u_{i+1} is the node with the smallest td and gets selected by the algorithm

Dijkstra Algorithm

Implementation

Implementation:

- ► We have to manage a set of active nodes
- ► We start with only the start node in our set
- ► At the start of each iteration we need the node u with the smallest $\text{td}(u)$

  How to implement this?

Dijkstra Algorithm
Implementation

Implementation:

- Using a priority queue with $\mathrm{td}(u)$ as keys
- The following problem occurs:
    - The tentative distance of an active node might change multiple times before it is settled
    - We have to change the key in our priority queue without removing the entry

Limitations:

- Often only `insert`, `getMin` and `deleteMin` are implemented
- $\Rightarrow$ We only have access to the first element and not any desired one

Dijkstra Algorithm
Implementation

**Alternative:**

- If a node reoccurs with a smaller dist we insert the element one more time into the priority queue
  (We do nothing if the distance is greater or equal)

- We do not remove the old entry

- The node always gets solved with the smallest distance because of the smaller key

- If a settled node reoccurs with a higher dist we remove it and do simply nothing

Dijkstra Algorithm
Implementation - Example

Priority queue:
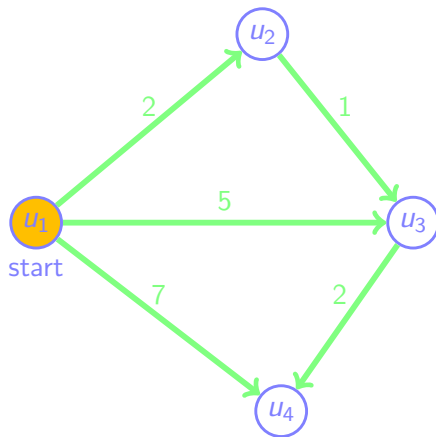
Dijkstra Algorithm

Runtime analysis

Graph with n nodes and m edges: $(m \geq n)$

- ▶ Each node gets solved exactly one time
- ▶ When solving a node it's outgoing edges are taken into account
- ▶ Each edge triggers at maximum one `insert` operation
- ▶ The number of operations on the priority queue is at maximum $O(m)$
- ▶ This results in a runtime of $O(m \cdot \log m)$
  (log m because of at max. m elements in the priority queue)

Dijkstra Algorithm
Runtime analysis

Runtime of $O(m \cdot \log m)$:

- Because of $m \leq n^2$ we have a maximum runtime of $O(m \cdot \log n)$, because $\log n^2 = 2 \log n$
- With a complex priority queue the runtime can be reduced to $O(m + n \log n)$
    - For example with a **Fibonacci heap**
    - This results in a better runtime for complex graphs $m \sim n^2$
    - Complex heaps create a management overhead
    - $\Rightarrow$ In practice $m \in O(n)$ with a **binary heap** being faster
      (See lecture 6)

Dijkstra Algorithm
Additional comments

Termination criteria:

- Terminate as soon as the target node t is settled
  . . . never before because tentative distance might change:

$$\mathrm{td}(t) \geq \mathrm{dist}(s, t)$$

- Before the node t is solved all nodes u with
  $\mathrm{dist}(s, u) \leq \mathrm{dist}(s, t)$ are settled

Dijkstra Algorithm
Additional comments

Termination criteria:

- Not only the single source **single** target shortest path problem is solved by the Dijkstra algorithm but also the single source **all** targets problem

- This sounds wasteful but there is not a (much) better method for general graphs
  Intuitive: We only know that there is no shorter path if all nodes in the distance of $\mathrm{dist}(s, t)$ are evaluated

Dijkstra Algorithm
Additional comments

Calculate the shortest path:

- With the current implementation of the Dijkstra algorithm we only get the length of the path
  How to get the path itself too?

- If we save the preceding node of the current shortest path on settling of each node we can reconstruct the path

Dijkstra Algorithm
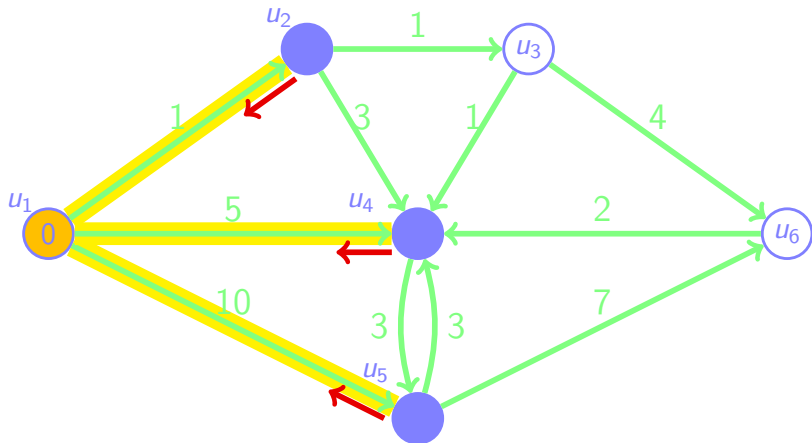
Figure: Start at u_1

Dijkstra Algorithm

Figure: Iteration 1

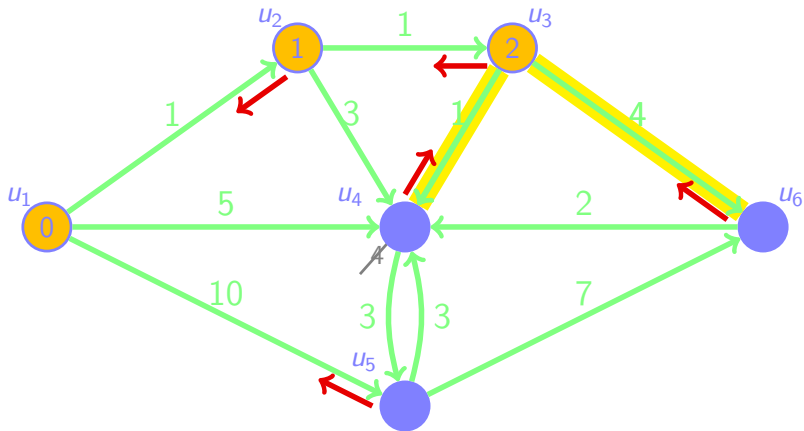# Dijkstra Algorithm

Figure: Iteration 2

Dijkstra Algorithm

Figure: Iteration 3

Dijkstra Algorithm

Figure: Iteration 4

Dijkstra Algorithm

Figure: Iteration 5
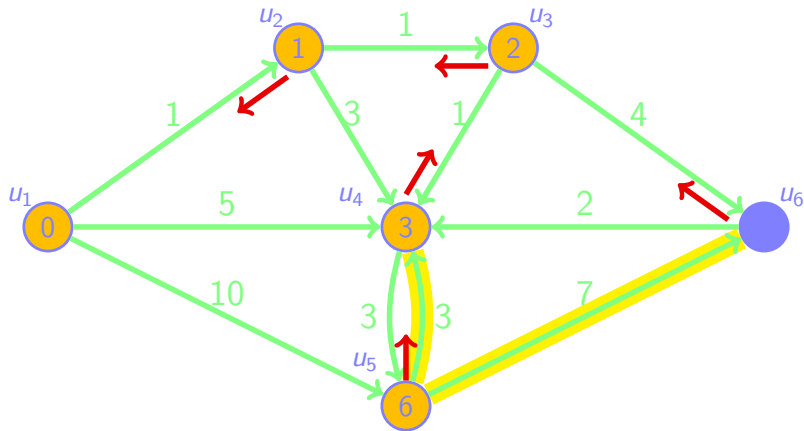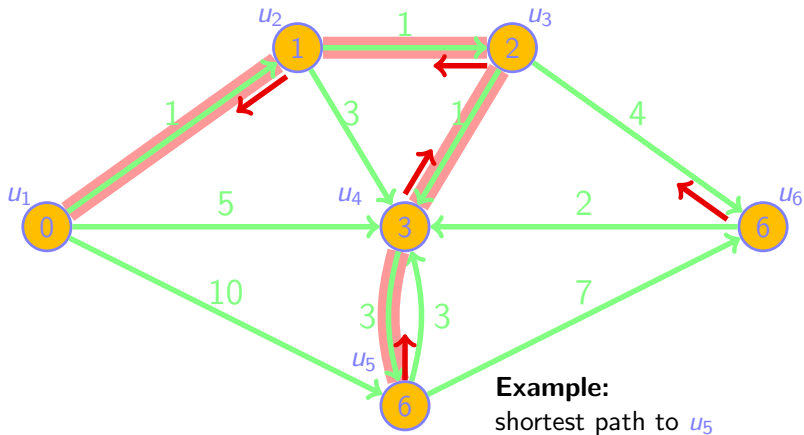
Dijkstra Algorithm

Figure: Iteration 6

Example: shortest path to u_5
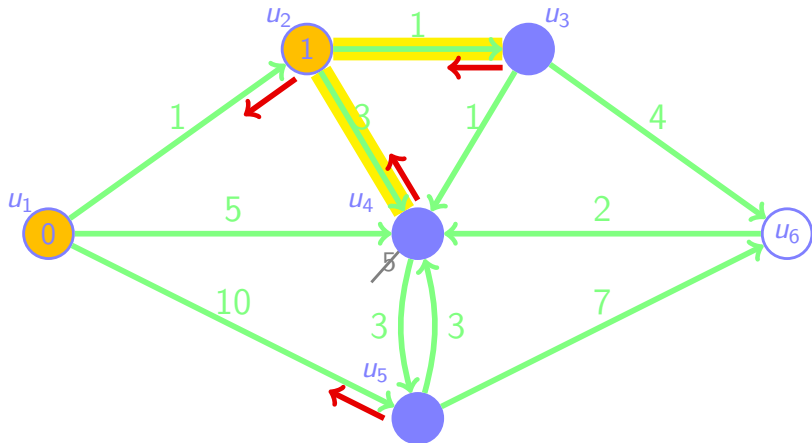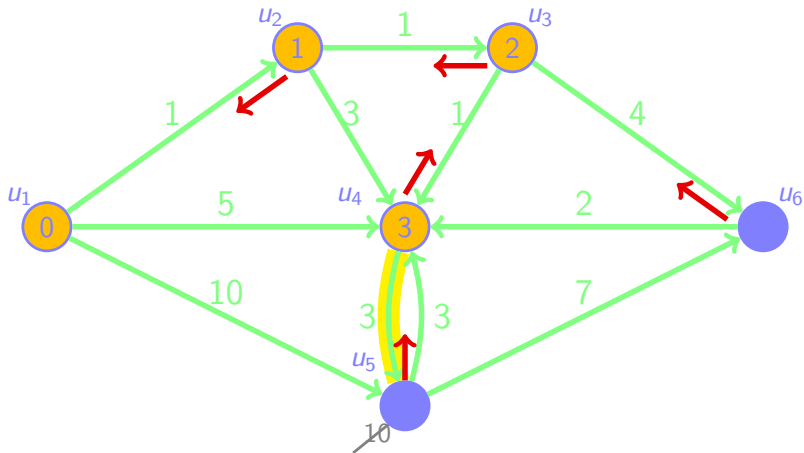
Dijkstra Algorithm

Additional comments

Enhancement:

- ▶ In our proof we used the assumption that all costs are not negative (even > 0)
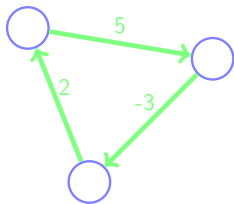- ▶ With negative costs there might be negative cycles:

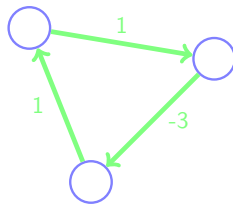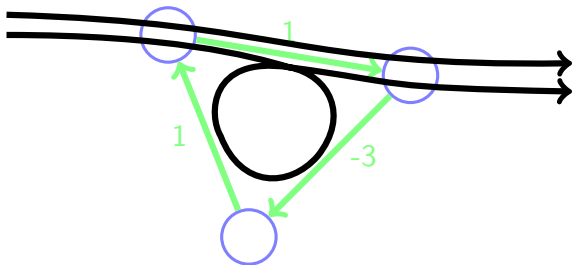Figure: Here no problem ...

Figure: ... but here

Dijkstra Algorithm

Additional comments

Negative cycles:

- No cycle:
  cost of 1
- 1 cycle:
  cost of 0
- 2 cycles:
  cost of -1
- 3 cycles:
  cost of -2
- ...

Dijkstra Algorithm
Additional comments

Enhancement:

- ▶ We need a different algorithm to deal with negative edges
  - ▶ For example the **Bellman-Ford** algorithm
  - ▶ If the graph is acyclic we can simply use a topological sorting (with DFS) and settling the nodes in order of this sorting
- ▶ Another (not only) in artificial intelligence used variant of the Dijkstra algorithm is the A* algorithm
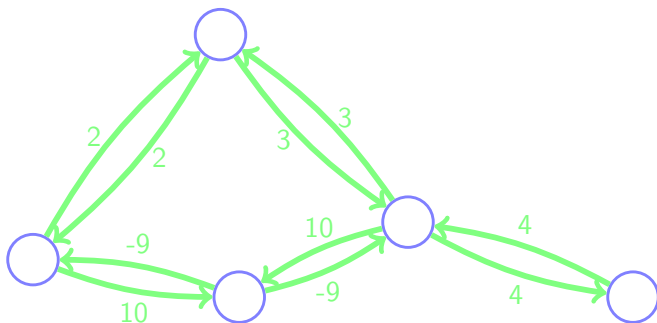 Additional information given:
  $h(u) = $ estimated value for $\mathrm{dist}(u, t)$

Dijkstra Algorithm

Example - Negative costs (e-car consumption)

Dijkstra algorithm:
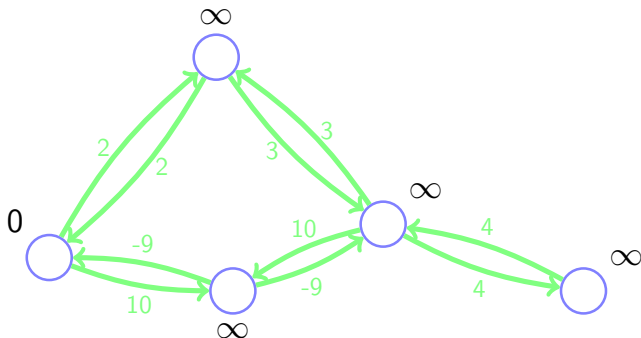Message passing only from solved nodes

Dijkstra Algorithm

Example - Negative costs (e-car consumption)

Bellman-Ford algorithm:
Message passing from all nodes until the path lengths are stable

Dijkstra Algorithm
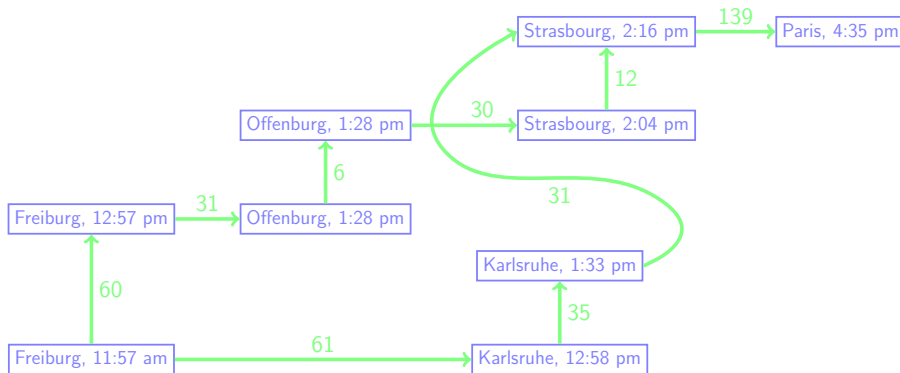
Application

Application example:

- ▶ Route planner for car trips (exercise sheet)
- ▶ Route planner for bus / train connections
  What could the graph look like?

Dijkstra Algorithm

Application

Space-time graph:

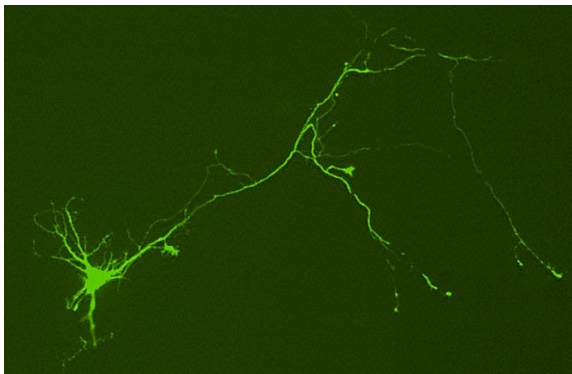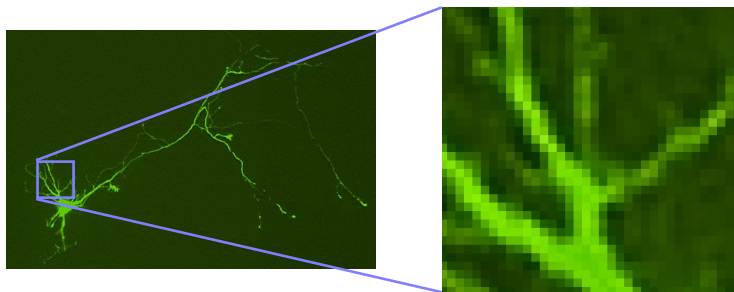# Dijkstra Algorithm

Application in image processing

-6em-6em

Figure: Neurons under fluorescence microscope

- ▶ **Task:** Measure length of axons (connections of neurons)
- ▶ Demo with ImageJ plugin NeuronJ
  http://www.imagescience.org/meijering/software/
  neuronj/

Dijkstra Algorithm

Application: Trace axons

- ▶ Image as graph: Each pixel is a node
- ▶ Implicit edges: Each pixel has an edge to it's 8 neighbours (no need to save the edges)
- ▶ Costs for nodes (not edges): bright pixels are cheap, dark pixels are costly

Further Literature

- **General**

[CRL01]   Thomas H. Cormen, Ronald L. Rivest, and
          Charles E. Leiserson.
          Introduction to Algorithms.
          MIT Press, Cambridge, Mass, 2001.

[MS08]   Kurt Mehlhorn and Peter Sanders.
          Algorithms and data structures, 2008.
          https://people.mpi-inf.mpg.de/~mehlhorn/
          ftp/Mehlhorn-Sanders-Toolbox.pdf.

Further Literature

- **Dijkstra's algorithm**

  [Wik] Dijkstra's algorithm
    https:
    //en.wikipedia.org/wiki/Dijkstra's_algorithm

- **Shortest path problem**

  [Wik] Shortest path problem
    https://en.wikipedia.org/wiki/Shortest_path_
    problem