Pfadsuche

Mario Mohr

GWV-Tutorium 2013/14

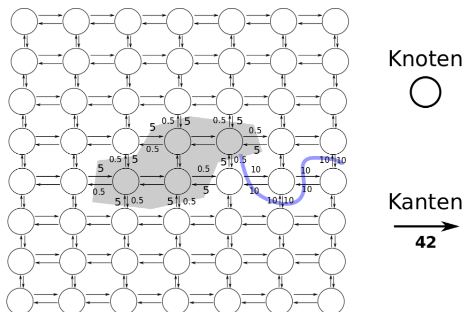
17. Januar 2014

1. Graph
2. Suchalgorithmus
3. Uninformierte Suche
   - Breadth-first
   - Loop detection & multiple-path pruning
   - Lowest-cost-first (aka Dijkstra)
   - Depth-first
   - Iterative-deepening
     - Bounded Depth-first
4. Informierte Suche
   - Heuristik
   - Best-first
   - A*

Graph

Ein **Graph** $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ besteht (für unsere Zwecke) aus

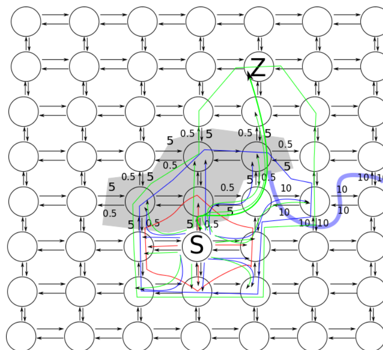
- der Knotenmenge $\boldsymbol{V}$ *("vertices")*, also der Menge von z.B. Orten oder Zuständen des Graphen und
- der Kantenmenge $\boldsymbol{E} : (W \subseteq (V \times V)) \rightarrow \mathbb{R}$, also der Menge von Verknüpfungen im Graph und deren Kosten (z.B. Bahnverbindungen und deren Fahrtdauer)



Suchalgorithmus

Ein **Suchalgorithmus**

- sucht in einem gegebenen Graphen nach einem (üblicherweise dem kürzesten) Pfad von einem Start- zu einem Endknoten,
- indem er die Frontier zum unbekannten Teil des Graphen ausdehnt, bis er einen Lösungspfad gefunden hat

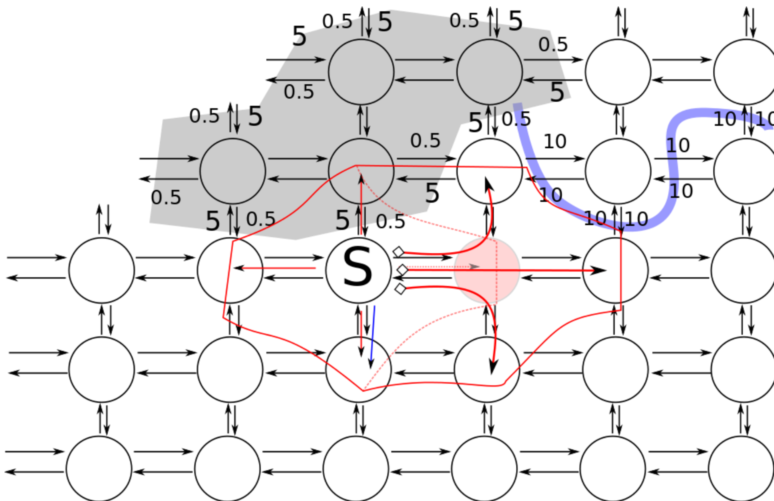


Suchalgorithmus - Pseudocode

Ein generischer Suchalgorithmus als Pseudocode:

- $_frontier = [Startknoten]$
- 1. $_aktueller_pfad = \boldsymbol{waehhle_frontier_erweiterung()}$
  2. $IF(FuehrtZuZielknoten(_aktueller_pfad)))$:
     $return\ _aktueller_pfad$
  3. $ELSE$:
     $for_each(n\ in\ _aktueller_pfad.endKnoten.alleNachbarn())$:
       $_frontier.add(_aktueller_pfad + n)$
     $_frontier.delete(_aktueller_pfad)$
     $GOTO\ 1$

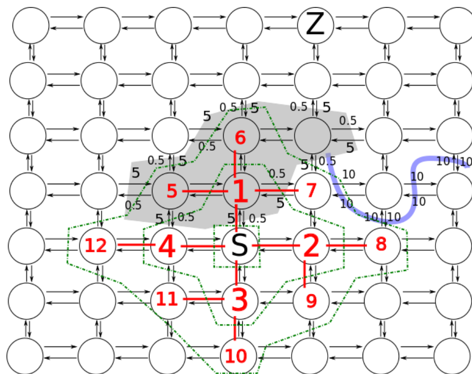
Frontier-Erweiterung



Breadth-first-search

Bei der **breadth-first-search**

- werden die Pfade der Frontier in einer Queue gespeichert:
- in jeder Iteration werden die Nachbarn des aktuellen Endknotens (der des Pfades am Queue-Kopf) am Ende der Queue angehängt



Loop detection & multiple-path pruning

Beim Einsatz von **loop detection** wird kein Pfad, der auf einem bereits von ihm besuchten Knoten endet, jemals zur Frontier hinzugefügt.

Beim Einsatz von **multiple-path pruning** werden bei mehreren Pfaden mit demselben Endknoten alle bis auf einen (üblicherweise der kürzeste) entfernt.

Lowest-cost-first-search (aka Dijkstra)

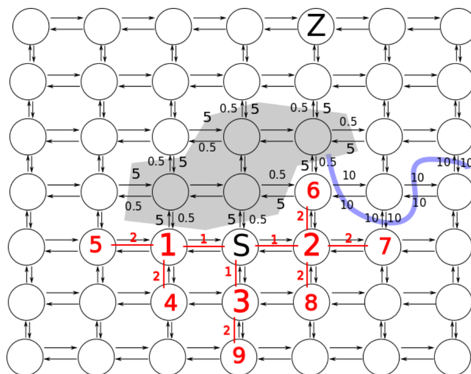
Bei der **lowest-cost-first-search**

- wird bei jeder Iteration aus der Frontier der Pfad mit den geringsten Gesamtkosten ausgewählt:
- sie verhält sich also wie eine Kantengewicht-sensitive Breitensuche

Lowest-cost-first-search (aka Dijkstra)

Bei der **lowest-cost-first-search**

- wird bei jeder Iteration aus der Frontier der Pfad mit den geringsten Gesamtkosten ausgewählt:
- sie verhält sich also wie eine Kantengewicht-sensitive Breitensuche



Depth-first-search

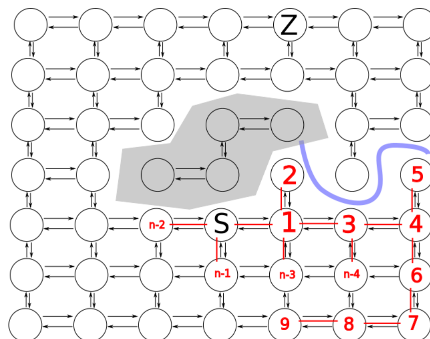
Bei der **Depth-first-search**

- werden die Pfade der Frontier in einem Stack gespeichert:
- in jeder Iteration werden die Nachbarn des aktuellen Endnkotens (der des Pfades am Stack-Kopf) am Anfang des Stacks abgelegt

Depth-first-search

Bei der **Depth-first-search**

- werden die Pfade der Frontier in einem Stack gespeichert:
- in jeder Iteration werden die Nachbarn des aktuellen Endnkotens (der des Pfades am Stack-Kopf) am Anfang des Stacks abgelegt



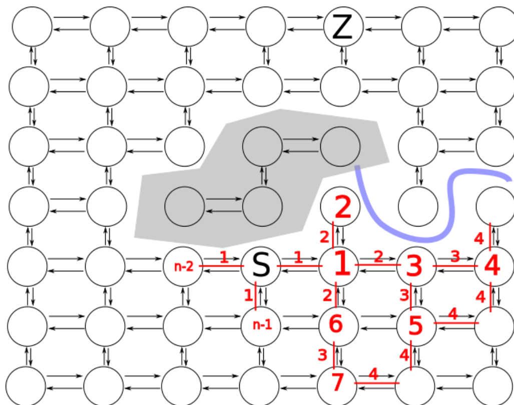
(Achtung: das hier ist depth-first-search _mit_ multiple path pruning!)

Bounded Depth-first-search

Bei der ***bounded*** **depth-first-search** werden Pfade mit einer Länge größer der festgelegten Beschränkung *(im Beispiel: 3)* ignoriert

Bounded Depth-first-search

Bei der ***bounded* depth-first-search** werden Pfade mit einer Länge größer der festgelegten Beschränkung *(im Beispiel: 3)* ignoriert



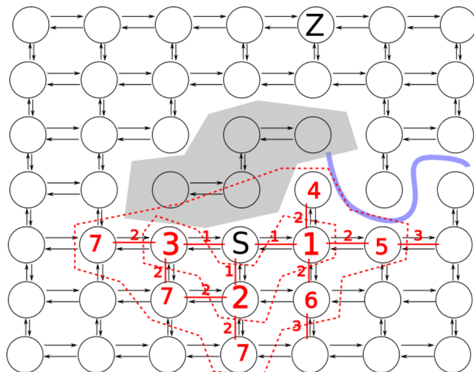
(Achtung: das hier ist bounded depth-first-search _mit_ multiple path pruning!)

Iterative-deepening-search

Bei der **Iterative-deepening-search** werden nacheinander bounded depth-first-searches mit steigender Beschränkung durchgeführt

Iterative-deepening-search

Bei der **Iterative-deepening-search** werden nacheinander bounded depth-first-searches mit steigender Beschränkung durchgeführt



(Achtung: das hier ist bounded depth-first-search _mit_ multiple path pruning!)

Heuristik

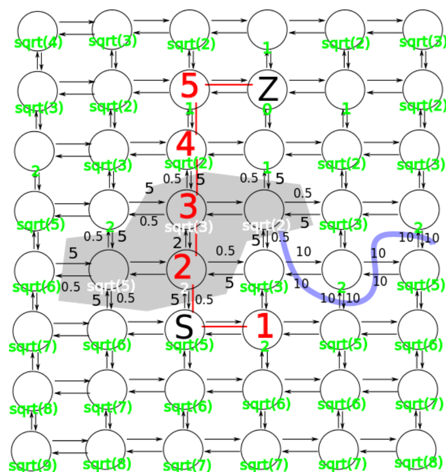
Eine **Heuristik**

- ist eine Funktion, die Teillösungen eines Problems anhand ihrer voraussichtlichen Nützlichkeit zur Gesamtlösung bewertet
- kann in Suchalgorithmen verwendet werden, um die Frontier-Ausdehnung anhand von schon vor Systemstart absehbaren Informationen zu lenken
- wird als *admissible* bezeichnet, wenn sie die Qualität einer Lösung immer realistisch oder zu positiv bewertet (also z.B. die Distanz eines Knotens zum Ziel realistisch oder zu niedrig schätzt)

Best-first search

Bei der **best-first-search**

- wird bei jeder Iteration aus der Frontier der Pfad mit dem besten heuristischen Wert (üblicherweise des Endknotens) gewählt;
- die Frontier wird also rein nach einem vordefinierten Qualitätskriterium ausgedehnt *(im Beispiel: die euklidische Distanz zum Ziel)*



Beim **A***-Algorithmus

- werden lowest-cost-first-search und best-first-search kombiniert:
- jede Iteration wird aus der Frontier der Pfad mit der geringsten Summe aus bisherigen Gesamtkosten und heuristisch geschätzten weiteren Kosten zum Ziel gewählt

Beim **A***-Algorithmus

- werden lowest-cost-first-search und best-first-search kombiniert:
- jede Iteration wird aus der Frontier der Pfad mit der geringsten Summe aus bisherigen Gesamtkosten und heuristisch geschätzten weiteren Kosten zum Ziel gewählt