

Funktionale Analyse

Funktionale Analyse
zum
Problem des kürzesten Wegs

Funktionale Analyse

Modularisierung:

- Suchfunktion
- Datenstruktur mit Zugriffsfunktionen
- Funktionen für Prioritätswarteschlange
- Funktionen zur Expansion des nächsten Schritts

Funktionale Analyse

Datenstruktur

- Kantenliste oder Knotenliste?
- Gewählt Knotenliste

```
[ ['A', [ ('B', 69), ('D', 36), ('M', 36), ('N', 22) ] ],  
  ['B', [ ('A', 69), ('D', 64), ('Z', 32), ('H', 30),  
          ('I', 34), ('X', 26) ] ],  
  ['C', [ ('I', 40), ('M', 31), ('X', 23) ] ],  
  ['D', [ ('A', 36), ('B', 64), ('L', 95), ('N', 20) ] ],  
  ... ]
```

- Zugriff über eine Funktion
`holeNachfolger`

Funktionale Analyse

Datenstruktur

- Kantenliste oder Knotenliste?
- Gewählt Knotenliste
sinnvoller: Dictionary

```
{ 'A' : [ ('B', 69), ('D', 36), ('M', 36), ('N', 22) ],  
  'B' : [ ('A', 69), ('D', 64), ('Z', 32), ('H', 30),  
          ('I', 34), ('X', 26) ],  
  'C' : [ ('I', 40), ('M', 31), ('X', 23) ],  
  'D' : [ ('A', 36), ('B', 64), ('L', 95), ('N', 20) ],  
  ... }
```

- Zugriff `d['B']`

Funktionale Analyse

Funktionen für die Prioritätswarteschlange

- fuegeEin
- fuegeAlleEin *(eventuell extern)*
- Hinweis:
 - Zugriffe allein funktional, keine eigene Datenstruktur

Funktionale Analyse

Weg expandieren

- alleNachfolger liefert zu einem Weg alle fortsetzenden Wege, daher
- bildet die Hilfsfunktion neuerWeg zu einer fortsetzenden Kante den neuen Weg und
- filtert die Hilfsfunktion entferneRueckwege ggf alle die Wege, die zum vorigen Knoten zurückführen (*geht auch über besuchtListe*).
- Schlechtere Wege brauchen wegen der PrioWS nicht gefiltert zu werden.

Funktionale Analyse

Das eigentliche Suchprogramm

- stellt wegen der verallgemeinerten Funktionen für die Prioritätswarteschlange das Prädikat vor bereit
- steuert die Rekursion und Aufrufe
- bekommt eine Aufrufhülle