

# 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 Kantenliste als

**(define**

**\*kanten\***

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

**... )**

- Zugriff mit Funktion nachfolge-kanten,  
Hilfsfunktion ist-nachfolge-kante?

# Funktionale Analyse

## Funktionen für die Prioritätswarteschlange

- fuege-einen-ein
- fuege-alle-ein
- Hinweis:
  - Zugriffe allein funktional, keine eigene Datenstruktur

# Funktionale Analyse

## Weg expandieren

- **alle-nachfolger** liefert zu einem Weg alle fortsetzenden Wege, daher
- bildet die Hilfsfunktion **neuer-weg** zu einer fortsetzenden Kante den neuen Weg und
- filtert die Funktion **alle-nachfolger** mit Hilfe von **member** alle die Wege, die zum vorigen Knoten zurückführen.
- 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