Algorithmus von Prim

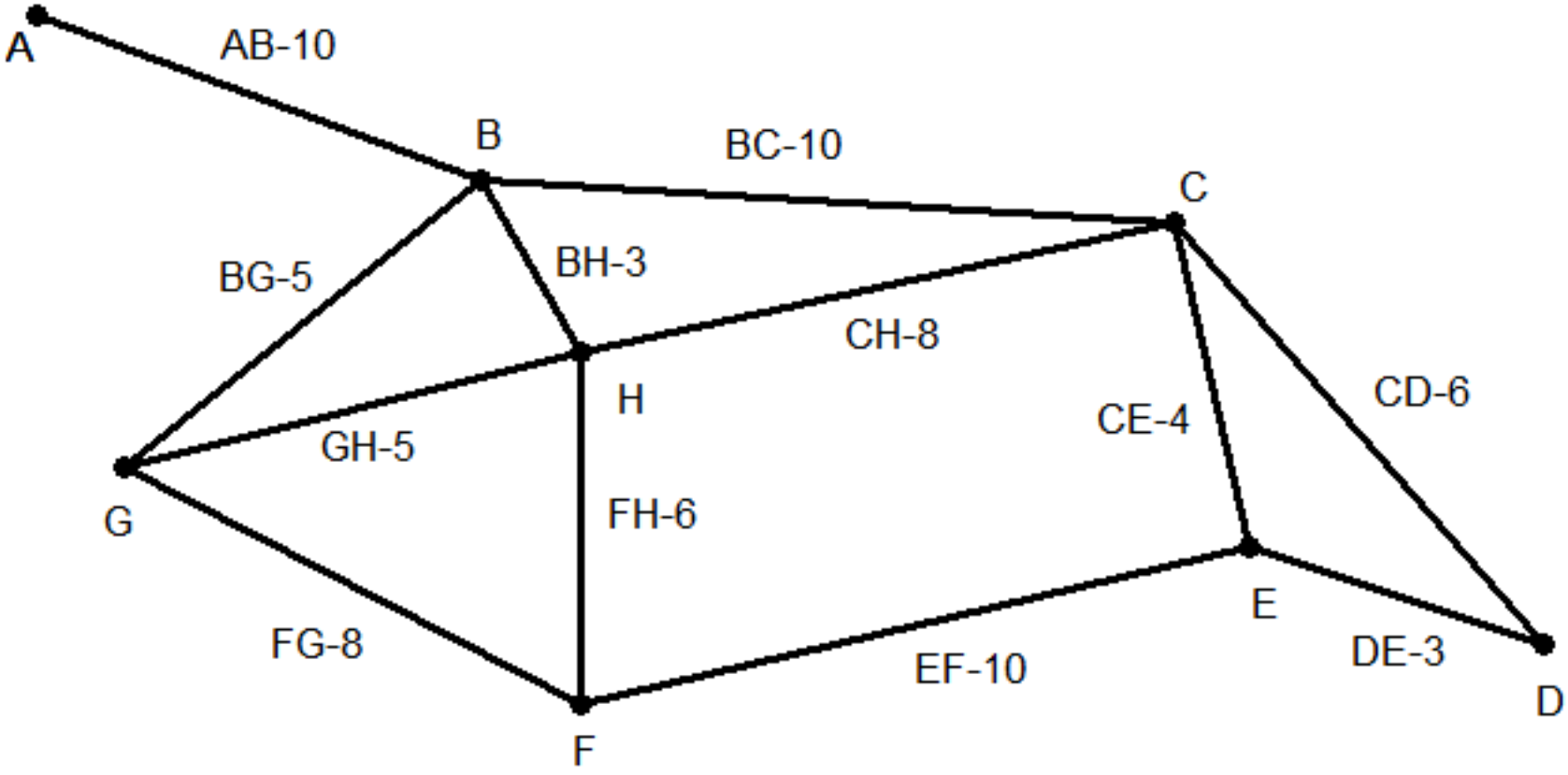
Algorithmus von Prim

ein Algorithmus zur Bestimmung eines minimalen aufspannenden Baums

(minimal spanning tree)

Algorithmus von Prim

Ein Graph



Algorithmus von Prim

- Beginne mit einem Baum, der allein aus einem Knoten (Ecke) besteht.
- Ordne alle Kanten von diesem Knoten aus nach ihren Kosten. (->Prioritätswarteschlange nach den Bewertungen)
- ...

Algorithmus von Prim

- Füge nun jeweils immer aus dieser jeweils die nächste Kante mit den geringsten Kosten ein, die keinen Zyklus erzeugt.
- Dann füge alle vom freien Knoten ausgehenden zulässigen Kanten in die Prioritätswarteschlange ein.
- Brich damit ab, wenn alle Knoten des Graphen zum Baum gehören.

Algorithmus von Prim

Aufgaben:

- Bestimmen der Kanten, die vom neuen Knoten ausgehen
- Einfügen der Kanten, die vom aktuellen Knoten ausgehen, nach ihren Bewertungen in die Prioritätswarteschlange
- Prüfen, ob alle Knoten im Baum enthalten sind
- die eigentliche Steuerung des Algorithmus

Algorithmus von Prim

Bestimmen der Kanten,
die vom neuen Knoten ausgehen

(Nachfolgekanten)

Algorithmus von Prim

```
(define
  (nachfolge-kanten aktuelle-stadt kanten)
  (define
    (intern aktuelle-stadt kanten akku)
     ...
    )
  (intern aktuelle-stadt kanten '())
  )
```

Aufrufhülle

interne Funktion

Algorithmus von Prim

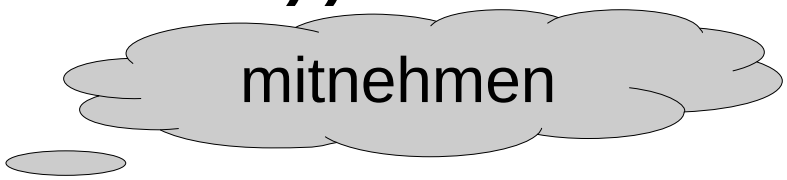
```
(define
  (nachfolge-kanten aktuelle-stadt kanten)
  (define
    (intern aktuelle-stadt kanten akku)
    (cond
      ((null? kanten) (reverse akku))
```

Verzweigung mit Elementarfall

```
    )
  (intern aktuelle-stadt kanten '())
  )
```

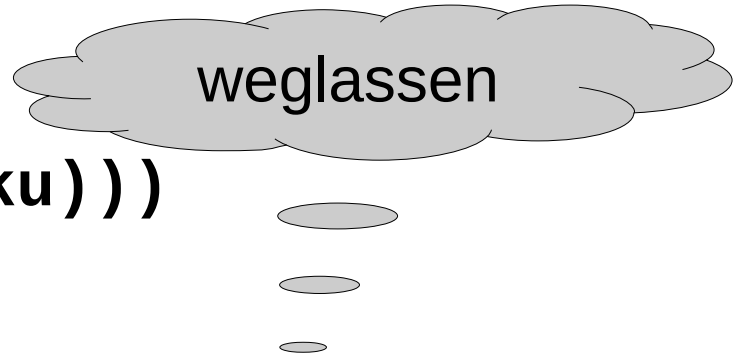
Algorithmus von Prim

```
(define
  (nachfolge-kanten aktuelle-stadt kanten)
  (define
    (intern aktuelle-stadt kanten akku)
    (cond
      ((null? kanten) (reverse akku))
      ((ist-nachfolge-kante?        ;Hilfsfunktion
         aktuelle-stadt (first kanten))
       (intern
        aktuelle-stadt
        (rest kanten)
        (cons (first kanten) akku)))


    )
  (intern aktuelle-stadt kanten '())
  )
```



Algorithmus von Prim

```
(define
  (nachfolge-kanten aktuelle-stadt kanten)
  (define
    (intern aktuelle-stadt kanten akku)
    (cond
      ((null? kanten) (reverse akku))
      ((ist-nachfolge-kante?
         aktuelle-stadt (first kanten))
       (intern
        aktuelle-stadt
        (rest kanten)
        (cons (first kanten) akku)))
      (else
        (intern
          aktuelle-stadt (rest kanten) akku))))
  (intern aktuelle-stadt kanten '())
  )
```



Algorithmus von Prim


```
(define
  (ist-nachfolge-kante? aktuelle-stadt kante)
  (equal? aktuelle-stadt (first kante)))
```

Algorithmus von Prim

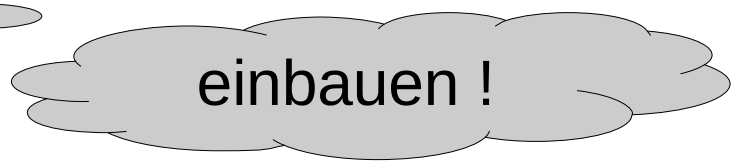
Geordnetes Einfügen
der Kanten
in eine
Prioritätswarteschlange

Algorithmus von Prim

```
(define
  (fuege-alle-ein alle vor? priows)
  (cond
    ((null? alle) priows) ; Elementarfall
    (else
     (fuege-alle-ein
      (rest alle)
      vor?
      (fuege-einen-ein
        (first alle) vor? priows)))
    )
  )
```

Algorithmus von Prim

```
(define
  (fuege-einen-ein einer vor? priows)
  (cond
    ((null? priows) (list einer))
    ((vor? einer (first priows))
     (cons einer priows))

    )
  )
```



Algorithmus von Prim

```
(define
  (fuege-einen-ein einer vor? priows)
  (cond
    ((null? priows) (list einer))

    )
  )
```

Verzweigung mit Elementarfall

Algorithmus von Prim

```
(define
  (fuege-einen-ein einer vor? priows)
  (cond
    ((null? priows) (list einer))
    ((vor? einer (first priows))
     (cons einer priows))
    (else
     (cons
      (first priows)
      (fuege-einen-ein
        einer vor? (rest priows))))
    )
  )
```

rekursiv nachklappernd

weglassen

Algorithmus von Prim


```
(define
  (vor? kante-1 kante-2)
  (< (caddr kante-1) (caddr kante-2)))
```

Die Kanten sind also gespeichert in der Form:
(<Knoten-1> <Knoten-2> <Bewertung>)

Algorithmus von Prim

Abbruchkriterium ist, dass
die Zahl der im Baum vorhandenen Knoten
gleich
der Zahl der Knoten im Graphen
ist

Algorithmus von Prim

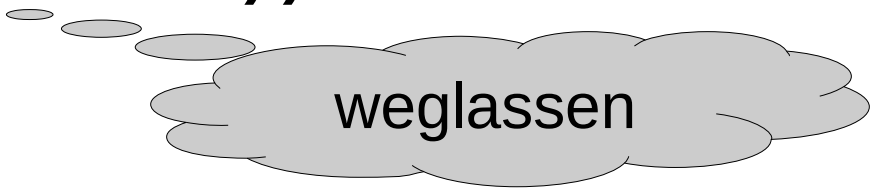
```
(define
  (anzahl-knoten kanten)
  (define
    (anzahl-intern kanten gefunden)
    (cond
      ((null? kanten) (length gefunden))

      ))
  (anzahl-intern kanten '()))
```

Aufrufhülle und interne Funktion

Elementarfall liefert hier das Ergebnis

Algorithmus von Prim

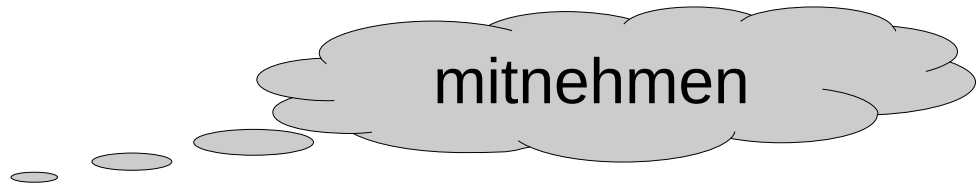
```
(define
  (anzahl-knoten kanten)
  (define
    (anzahl-intern kanten gefunden)
    (cond
      ((null? kanten) (length gefunden))
      ((member
         (first (first kanten)) gefunden)
       (anzahl-intern
         (rest kanten) gefunden))
```




```
      ))
  (anzahl-intern kanten '()))
```

Algorithmus von Prim

```
(define
  (anzahl-knoten kanten)
  (define
    (anzahl-intern kanten gefunden)
    (cond
      ((null? kanten) (length gefunden))
      ((member
         (first (first kanten)) gefunden)
       (anzahl-intern
         (rest kanten) gefunden))
      (else
       (anzahl-intern
        (rest kanten)
        (cons
          (first (first kanten)) gefunden)))
      ))
  (anzahl-intern kanten '()))
```



Algorithmus von Prim

Die Funktion prim wird aufgeteilt

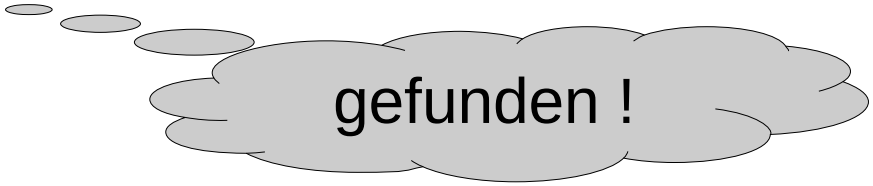
- in eine Aufrufhülle und
- die eigentliche Schrittfunktion

Algorithmus von Prim

```
(define
  (prim start-knoten kanten)
  (define
    (prim-intern besucht anzahl kuerzeste priows)

    …

      )) ; Ende interne Funktion
  ; interner Aufruf:
  (prim-intern
   (list start-knoten)
   (anzahl-knoten kanten)
   '()
   (fuege-alle-ein ; Start-PrioWS
    (nachfolge-kanten start-knoten kanten)
    vor?
    '()))
```

Algorithmus von Prim

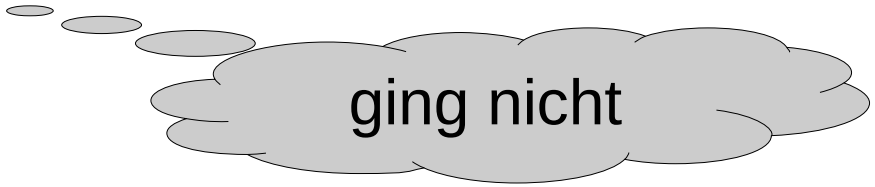
```
(define                          ; die interne Funktion
    (prim-intern besucht anzahl kuerzeste priows)
    (cond
      ((equal? (length besucht) anzahl)
       kuerzeste)
```




```
      ))
```

Algorithmus von Prim

```
(define                          ; die interne Funktion
    (prim-intern besucht anzahl kuerzeste priows)
    (cond
      ((equal? (length besucht) anzahl)
       kuerzeste)
      ((null? priows)
       #f)


      ))
```



Algorithmus von Prim

```
(define                          ; die interne Funktion
    (prim-intern besucht anzahl kuerzeste priows)
    (cond
      ((equal? (length besucht) anzahl)
       kuerzeste)
      ((null? priows)
       #f)
      ((zyklus? (first priows) besucht)
       (prim-intern
         besucht anzahl kuerzeste (rest priows)))
```


```
      ))
```

Algorithmus von Prim

```
(define                          ; die interne Funktion
    (prim-intern besucht anzahl kuerzeste priows)
    (cond
      ((equal? (length besucht) anzahl)
       kuerzeste)
      ((null? priows)
       #f)
      ((zyklus? (first priows) besucht)
       (prim-intern
         besucht anzahl kuerzeste (rest priows)))
      (else
       (prim-intern
        (aktualisiere-besuchtliste
          (second (first priows)) besucht)
        anzahl
        (cons (first priows) kuerzeste)
        (aktualisiere-priows priows)))
      ))
```

die gehört dazu

Algorithmus von Prim

```
(define                          ; die interne Funktion
    (prim-intern besucht anzahl kuerzeste priows)
    (cond
      ((equal? (length besucht) anzahl)
       kuerzeste)
      ((null? priows)
       #f)
      ((zyklus? (first priows) besucht)
       (prim-intern
         besucht anzahl kuerzeste (rest priows)))
      (else
       (prim-intern
        (aktualisiere-besuchtliste
          (second (first priows)) besucht)
        anzahl
        (cons (first priows) kuerzeste)
        (aktualisiere-priows priows)))
      ))
```

Algorithmus von Prim

die Hilfsfunktionen

Algorithmus von Prim

```
(define
  (aktualisiere-besuchtliste knoten besucht)
  (cond
   ((member knoten besucht)
    besucht)
   (else
    (cons knoten besucht))
   ))
```

Algorithmus von Prim

```
(define
  (aktualisiere-priows priows)
  (fuege-alle-ein
   (nachfolge-kanten (second (first priows)) kanten)
   vor?
   (rest priows)))
```