

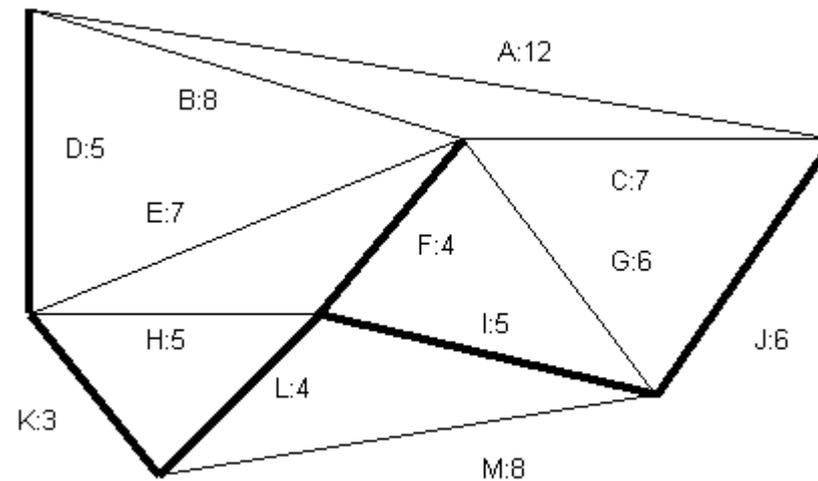


# Algorithmus von Kruskal

minimal spanning tree

# Algorithmus von Kruskal

so soll's werden !





# Algorithmus von Kruskal

- Ordne alle Kanten nach ihren Kosten.
- Beginne mit einem Baum, der allein aus einer Kante mit minimalen Kosten besteht.
- Füge dann jeweils immer die nächste Kante mit den geringsten Kosten ein, die keinen Zyklus erzeugt.
- Brich damit ab, wenn alle Ecken (Knoten) des Graphen zum Baum gehören.



# Algorithmus von Kruskal

- Beim Aufbau des Baumes können Teilgraphen entstehen, die nicht zusammenhängend sind, letztlich müssen aber alle Ecken mit dem Graphen verbunden sein.
- Anderenfalls gäbe es keine Kante von dieser Ecke zu irgendeiner Ecke des Baumes, was der Forderung nach einem zusammenhängenden Ausgangsgraphen widerspricht.

# Algorithmus von Kruskal

Wird eine Kante hinzugefügt, dann hat sie entweder ...

- keine Ecke mit dem derzeitigen Teilgraphen gemeinsam, die Kante ist dann zur Zeit noch isoliert - oder sie hat ...
- eine Ecke mit dem derzeitigen Teilgraphen gemeinsam, dann setzt sie einen Ast fort oder bildet einen neuen - oder sie ...
- verbindet zwei bis dahin noch isolierte Teile des Teilgraphen mit einander, nur dann hat sie zwei Ecken mit dem derzeitigen Teilgraphen gemeinsam.



# Algorithmus von Kruskal

## Aufgaben:

- Ordnen der Kanten nach ihren Bewertungen
- Aufnehmen der neuen Kante als neuen Teilbaum
- Verbinden zweier Teilbäume mit der neuen Kante
- Hinzufügen einer Kante zu einem Teilbaum
- Prüfen, ob alle Ecken im Baum enthalten sind
- die eigentliche Algorithmussteuerung



# Algorithmus von Kruskal

Ordnen der Kanten

# Algorithmus von Kruskal

```
(define
  (ordne-kanten kanten)
  (cond
    ((null? kanten) kanten)
    (else
     (ordne-ein
      (car kanten)
      (ordne-kanten (cdr kanten))))))
```

# Algorithmus von Kruskal

```
(define
  (ordne-ein kante kanten)
  (cond
    ((null? kanten) (cons kante kanten))
    ((kuerzer? kante (car kanten))
     (cons kante kanten))
    (else
     (cons
      (car kanten)
      (ordne-ein kante (cdr
kanten))))))
```



# Algorithmus von Kruskal

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

Die Kanten sind also gespeichert in der Form:

```
(<Ecke-1> <Ecke-2> <Bewertung>)
```



# Algorithmus von Kruskal

Bestimmung der Ecken

# Algorithmus von Kruskal

```
(define
  (alle-ecken baum)
  (cond
    ((null? baum) ())
    (else
     (let
        ((ecken (alle-ecken (cdr baum))))
        (append
         (if
          (member (caar baum) ecken)
          '() (list (caar baum)))
         (if
          (member (cadar baum) ecken)
          '() (list (cadar baum)))
         ecken))))))
```

# Algorithmus von Kruskal

```
(define
  (alle-ecken-enthalten? ecken baeume)
  (cond
    ((null? ecken) #t)
    ((ecke-enthalten?
      (car ecken) (car baeume))
     (alle-ecken-enthalten?
      (cdr ecken) baeume))
    (else #f)))
```

# Algorithmus von Kruskal

```
(define
  (ecke-enthalten? ecke kanten)
  (cond
    ((null? kanten) #f)
    ((equal? ecke (caar kanten)) #t)
    ((equal? ecke (cadar kanten)) #t)
    (else
     (ecke-enthalten?
      ecke
      (cdr kanten))))))
```



# Algorithmus von Kruskal

Bestimmung der Zahl der Teilbäume,  
in denen die Kante enthalten ist

# Algorithmus von Kruskal

```
(define
  (in-teilbaeumen kante baeume)
  (cond
    ((null? baeume) 0)
    ((and
      (ecke-enthalten?
        (car kante) (car baeume))
      (ecke-enthalten?
        (cadr kante) (car baeume)))
     -1)
```

...

# Algorithmus von Kruskal

...

```
( (ecke-enthalten?  
  (car kante) (car baeume) )  
  (add1  
    (in-teilbaeumen kante (cdr  
baeume) ) ) )  
( (ecke-enthalten?  
  (cadr kante) (car baeume) )  
  (add1  
    (in-teilbaeumen kante (cdr baeume) ) ) )  
(else  
  (in-teilbaeumen kante (cdr  
baeume) ) ) ) )
```



# Algorithmus von Kruskal

```
(define  
  (verbinde kante baeeume) ...
```

Bei dieser Funktion werden zwei Teilbäume durch die neue Kante verbunden.

Um die Bearbeitung zu erleichtern, werden die Teilbäume solange umsortiert, bis die beiden zu verbindenden Teilbäume vorn in der Liste stehen.

# Algorithmus von Kruskal

```
(define
  (verbinde kante baeume)
  (cond
    ((and
      (ecke-enthalten?
        (car kante) (car baeume))
      (ecke-enthalten?
        (cadr kante) (cadr baeume)))
     ; nun stehen sie richtig
```

...

# Algorithmus von Kruskal

...

**; und können leicht verbunden werden**

```
(cons
  (cons
    kante
    (append
      (car baeume)
      (cadr baeume)))
  (cddr baeume))
```

# Algorithmus von Kruskal

```
...      ; erste steht noch falsch:
((not (ecke-enthalten?
      (car kante) (car baeume)))
 (verbinde kante
  (reverse      ; 1. nach hinten
   (cons
    (car baeume)
    (reverse (cdr baeume))))))
```

```
...
```

# Algorithmus von Kruskal

```
...      ; zweite steht noch falsch
(else
  (verbinde kante
    (cons
      (car baeume)
      (reverse      ; 2. nach hinten
        (cons
          (cadr baeume)
          (reverse (cddr baeume)))))))
```



# Algorithmus von Kruskal

```
(define  
  (hinzufuegen kante baeume) ...
```

Bei dieser Funktion wird die neue Kante dem Teilbaum hinzugefügt.



# Algorithmus von Kruskal

```
(define
  (hinzufuegen kante baeume)
  (cond
    ((or
      (ecke-enthalten?
        (car kante) (car baeume))
      (ecke-enthalten?
        (cadr kante) (car baeume)))
      (cons (cons kante (car baeume))
        (cdr baeume)))
    (else
      (hinzufuegen kante (cdr baeume)))))
```



# Algorithmus von Kruskal

Die Funktion `kruskal` wird aufgeteilt

- in eine Aufrufhülle und
- die eigentliche Schrittfunktion

# Algorithmus von Kruskal

```
(define
  (kruskal graph)
  (let
    ((geordnete-kanten
      (ordne-kanten graph)))
    (kruskal-schritt
      (cdr geordnete-kanten)
      (alle-ecken graph)
      (list
        (list
          (car geordnete-kanten))))))
```

# Algorithmus von Kruskal

```
(define
  (kruskal-schritt kanten ecken baeume)
  (cond
    ((null? kanten) ; irregulärer Abbruch
     (error "Kantenliste leer. Keine Lösung
möglich!"))

    ((alle-ecken-enthalten? ecken baeume)
     ; regulärer Abbruch
     (car baeume))

    ...
```

# Algorithmus von Kruskal

```
; Ein Zyklus tritt auf
( (= -1
  (in-teilbaeumen (car kanten) baeume) )
  (kruskal-schritt
    (cdr kanten)
    ecken
    baeume) )
```

# Algorithmus von Kruskal

```
; die neue Kante verbindet 2 Teilbäume  
( (= 2  
  (in-teilbaeumen (car kanten) baeume))  
  (kruskal-schritt  
    (cdr kanten)  
    ecken  
    (verbinde (car kanten) baeume)))
```

# Algorithmus von Kruskal

```
; neue Kante zum Teilbaum  
( (= 1  
  (in-teilbaeumen (car kanten) baeume) )  
  (kruskal-schritt  
    (cdr kanten)  
    ecken  
    (hinzufuegen (car kanten) baeume) ) )
```

# Algorithmus von Kruskal

```
; kante bildet neuen Teilbaum  
(else  
  (kruskal-schritt  
    (cdr kanten)  
    ecken  
    (cons (list (car kanten)) baeume))))
```