Von Dijkstra zu Prim

minimal spanning tree

ausgehend von einem
Einstieg über
den Algorithmus von Dijkstra

Von Dijkstra zu Prim

- Voraussetzung:
  - Algorithmus von Dijkstra ist bekannt und verstanden
- Problemstellung
  - Minimales Versorgungsnetz
- Vorgehen
  - Erarbeitung und
  - Formulierung des Algorithmus von Prim

Von Dijkstra zu Prim

Einstieg:

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

Von Dijkstra zu Prim

Arbeitsschritt
(Wiederholung)

- Füge nun aus dieser Prioritätswarteschlange jeweils immer die nächste Kante mit den geringsten Kosten ein, die zu einem noch nicht erreichten Knoten führt (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.

Von Dijkstra zu Prim

Aufgaben:

- Einordnen der Kanten vom aktuellen Knoten aus in die Prioritätswarteschlange nach ihren Bewertungen
- Bestimmung der Kanten, die von dem neuen Knoten ausgehen
- Prüfen, ob der freie Knoten (Zielknoten der Kante) in der Besucht-Liste enthalten ist
- Prüfen, ob alle Knoten im Baum enthalten sind
- die eigentliche Algorithmussteuerung

Von Dijkstra zu Prim

Daten als Kantenliste in gallenbacher-graph-kanten

- Das fertige Dijkstra-Programm liefert die Datenstruktur und die Zugrifssfunktion:
  - Die Liste **`*kanten*`**
  - Auf die Funktion **`(nachfolge-kanten aktuelle-stadt kanten)`** wird von außen zugegriffen

Von Dijkstra zu Prim

Prioritätswarteschlange

- Das fertige Dijkstra-Programm liefert die Funktionen zur Verwaltung der Prioritätswarteschlange:
  - Auf **`(fuege-alle-ein alle vor? priows)`** wird von außen zugegriffen
  - Intern wird **`(fuege-einen-ein einer vor? priows)`** verwendet

Von Dijkstra zu Prim

- Die Funktion **`vor?`** (das Prädikat)

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

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

Von Dijkstra zu Prim

Erarbeitung der Funktion **`prim`**

- Kopf
  - besucht Liste der besuchten Knoten
  - anzahl aller Knoten im Graphen
  - kuerzeste bereits gefundene Kanten aus dem Baum

```
(define
  (prim besucht anzahl kuerzeste kanten priows)
  'fehlt)
```

Von Dijkstra zu Prim

Erarbeitung der Funktion **`prim`**

- Der elementare Abbruchfall

```
(define
  (prim besucht anzahl kuerzeste kanten priows)
  (cond
    ((null? priows)
     #f)
    (else
     'fehlt)
    ))
```

Von Dijkstra zu Prim

Erarbeitung der Funktion **`prim`**

- Das Ziel ist erreicht: Alle Knoten wurden erreicht

```
(define
  (prim besucht anzahl kuerzeste kanten priows)
  (cond
    ((equal? (length besucht) anzahl)
     kuerzeste)
    ((null? priows)
     #f)
    (else
     'fehlt)
    ))
```

Von Dijkstra zu Prim

Erarbeitung der Funktion **prim**

- Hier war man schon, weiter ohne

```
(define
  (prim besucht anzahl kuerzeste kanten priows)
  (cond
    ((equal? (length besucht) anzahl)
     kuerzeste)
    ((null? priows)
     #f)
    ((member (second (first priows)) besucht)
     (prim besucht anzahl kuerzeste kanten
                                  (rest priows)))
    (else
     'fehlt)
    ))
```

Von Dijkstra zu Prim

Erarbeitung der Funktion **`prim`**

- Der Normalfall

```
(define
  (prim besucht anzahl kuerzeste kanten priows)
  ...
    (else
     (prim
      (aktualisiere-besuchtliste
        (second (first priows)) besucht)
      anzahl
      (cons (first priows) kuerzeste)
      kanten
      (aktualisiere-priows priows)))
    ))
```

Von Dijkstra zu Prim

Hilfsfunktion der Funktion **`prim`**

- Knoten der Liste besucht hinzu fügen

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

Von Dijkstra zu Prim

Hilfsfunktion der Funktion **`prim`**

- Die aktuelle Kante steht vorn in der priows und zu deren Zielknoten werden die Nachfolgekanten in die priows eingefügt

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

Von Dijkstra zu Prim

Hilfsfunktion der Funktion **`prim`**

- Die Anzahl der Knoten (mit Aufrufhülle)

```
(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 '()))
```

Von Dijkstra zu Prim

Die Funktion **`prim`** wird aufgeteilt

- in eine Aufrufhülle und
- die eigentliche Schrittfunktion **`prim-intern`**

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

Von Dijkstra zu Prim

In **`prim`** der Aufruf der internen Funktion:

```
      …

  (prim-intern
   (list start-knoten)          ; Start besucht
   (anzahl-knoten kanten)       ; anzahl
   '()                          ; kuerzeste
   kanten                       ; Gesamtgraph
   (fuege-alle-ein              ; Start-PrioWS
    (nachfolge-kanten start-knoten kanten)
    vor?
    '())
   ))
```