

Proseminar Programmverifikation

Vortrag zum Thema „minimum
spanning tree“
(Algorithmus von Kruskal)

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Inhalt

1. Grundlagen

2. Algorithmus

3. Verifikation (partielle Korrektheit)

4. Verifikation (totale Korrektheit)

5. Schlussbemerkungen

6. Quellen

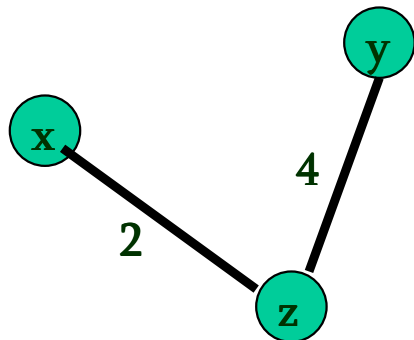
Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Grundlagen

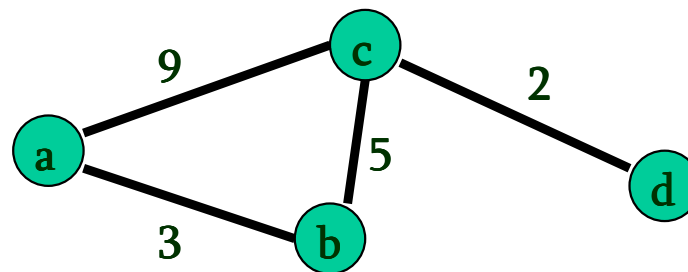
(ungerichteter) Graph: $G=(V,E)$, wobei V Knotenmenge, E Kantenmenge

(hier: gewichtete Kanten)

Beispiel(-e) für Graphen:



Kreisfreier Graph (Baum)



Graph, der einen Kreis enthält

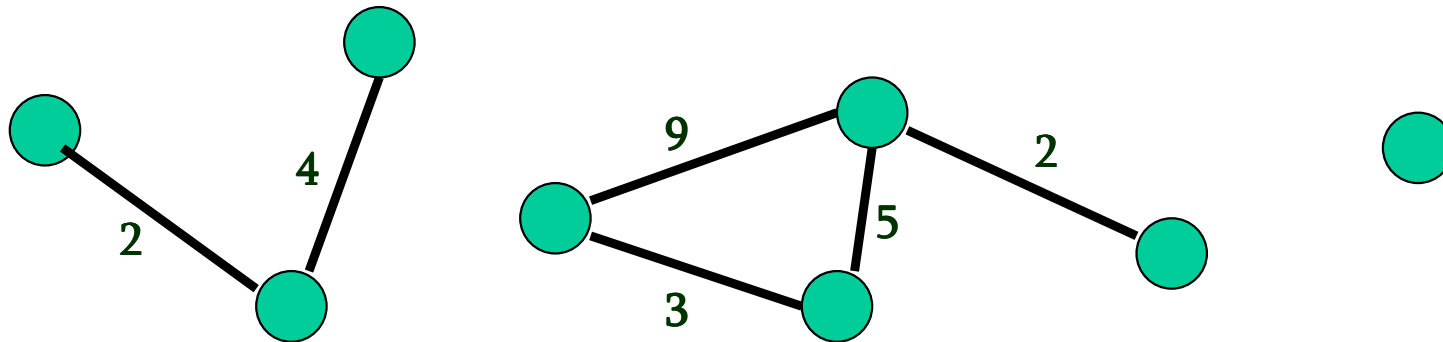
Schreibweise für Kante: $(\forall e \in E) \Rightarrow e = \{n,m\} : n,m \in V$

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Grundlagen

Zusammenhangskomponenten:
(Beispiel)

3 Zusammenhangskomponenten

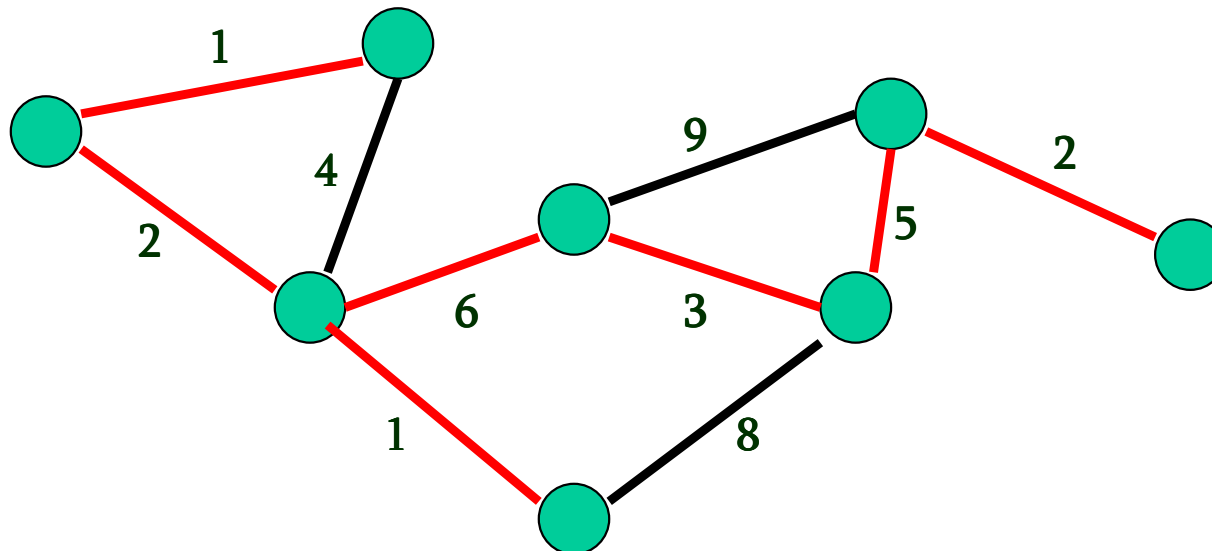


Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Grundlagen

Minimal Spannender Baum (MST):

(Baum auf einem Graph, dessen Summe der Kantengewichte minimal ist)



Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Der Algorithmus:

```
findinit; /* init the MST */
pqconstruct; /* sort given edges */
i := 0;
REPEAT
  m:=pqremove; /* dequeue an edge*/
  x:=edges[m].v1;
  y:=edges[m].v2;

  IF find(x,y,true) THEN /* no cycle in tree? */
  BEGIN
    edgefound(x,y); /* add the edge to the MST */
    i++;
  END
UNTIL pqempty OR i = V-1;
```

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Der Algorithmus:

Beschreibung des Algorithmus:

- Eingabe: Knotenmenge V , Kantenmenge E (Kanten mit Kantengewichten) auf V
- Sortiere die Kantenmenge E aufsteigend nach ihrem „Gewicht“
- wiederhole ...
 - nimm die „billigste“ Kante
 - wenn diese Kante zwei verschiedene Zusammenhangskomponenten verbindet, nimm sie in den MST auf
- > ... bis keine Kanten mehr vorhanden oder die Anzahl der Kanten im MST gleich $|V|-1$
- Ausgabe MST

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Der Algorithmus:

Laufzeit des Algorithmus:

- Eingabe: Knotenmenge V , Kantenmenge E (Kanten mit Kantengewichten) auf V
- Sortiere die Kantenmenge E aufsteigend nach ihrem „Gewicht“
- wiederhole ...
 - nimm die „billigste“ Kante
 - wenn diese Kante zwei verschiedene Zusammenhangskomponenten verbindet, nimm sie in den MST auf
- > ... bis keine Kanten mehr vorhanden oder die Anzahl der Kanten im MST gleich $|V|-1$
- Ausgabe MST

Bei Verwendung von Heapsort ist die Laufzeit beispielsweise $O(|E| \cdot \log |V|)$.

$O(1)$

$O(\log |V|)$

Laufzeit: $O(|E| \cdot \log |V|)$

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Verifikation:

PROGRAM kruskal;

CONST maxV=50; maxE=2500;

TYPE edge=record v1, v2, w : INTEGER END;

VAR i, j, m, x, y, V, E : INTEGER; edges : ARRAY[0..maxE] OF edge;

BEGIN

->Eingabe #V, #E;

FOR j := 1 TO E DO

BEGIN

->Eingabe Knoten1, Knoten2, Gewicht;

edges[j].w = Gewicht;

edges[j].v1 = Knoten1;

edges[j].v2 = Knoten2;

END;

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Verifikation (partielle Korrektheit)

```

    { V > 1 ∧ E > 0 ∧ 0 < V-1 }
findinit;          { V > 1 ∧ E > 0 ∧ 0 < V-1 ∧
                    p ≡ MST enthält V (- 0) viele Zusammenhangskomponenten }
pqconstruct;      { p ∧ Priority Queue PQ ≠ ∅ ∧ V > 1 ∧ E > 0 ∧ i < V-1 [i := 0] }
i:=0;             { p ∧ PQ ≠ ∅ ∧ V > 1 ∧ E > 0 ∧ i < V-1 }
    { inv: p ≡ MST enthält V-i viele Zusammenhangskomponenten }
REPEAT            { p ∧ PQ ≠ ∅ ∧ i < V-1 ∧ m = min{PQ} [m := pqremove] ∧
                  m = {x,y} [x := edges[m].v1; y := edges[m].v2] }
    m:=pqremove;  { p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧
                  m = {x,y} [x := edges[m].v1; y := edges[m].v2] }

    x:=edges[m].v1;
    y:=edges[m].v2;  { p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
    { B ≡ m verbindet zwei (versch.) Zusammenhangskomponenten im MST }
if(find(x,y,true)) THEN
BEGIN
    edgefound(x,y); { B ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
    i++;           { B ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i-1 < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
END               { (B ∨ ¬ B) ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i-1 < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
UNTIL pqempty OR i = V-1;  { p ∧ (PQ = ∅ ∨ i = V-1) }

```

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Verifikation (totale Korrektheit)

```

    { inv: p ≡ MST enthält V-i viele Zusammenhangskomponenten }
REPEAT
    { p ∧ |PQ| ≥ 1 }
    m:=pqremove;
    x:=edges[m].v1;
    y:=edges[m].v2;    { p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
    { B ≡ m verbindet zwei (versch.) Zusammenhangskomponenten im MST }
    if(find(x,y,true)) THEN
    BEGIN
        edgefound(x,y); { B ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
        i++;           { B ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i-1 < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
    END
    { (B ∨ ¬ B) ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i-1 < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
UNTIL pqempty OR i = V-1;    { p ∧ (PQ = ∅ ∨ i = V-1) }

```

|PQ| wird in jedem Schleifendurchlauf um 1 kleiner.

Terminierungsfunktion: $t = \min\{|PQ|, (V-i-1)\}$

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Verifikation (totale Korrektheit)

```

    { inv: p ≡ MST enthält V-i viele Zusammenhangskomponenten }
REPEAT
    { p ∧ PQ ≠ ∅ ∧ i < V-1 ∧ m = min{PQ} [m := pqremove] ∧
      m = {x,y} [x := edges[m].v1; y := edges[m].v2] }
m:=pqremove;
    { p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧
      m = {x,y} [x := edges[m].v1; y := edges[m].v2] }

x:=edges[m].v1;
y:=edges[m].v2;    { p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
    { B ≡ m verbindet zwei (versch.) Zusammenhangskomponenten im MST }
if(find(x,y,true)) THEN
BEGIN
    edgefound(x,y); { B ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
    i++;          { B ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i-1 < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
END
    { (B ∨ ¬ B) ∧ p ∧ PQ ∪ {m} ≠ ∅ ∧ i-1 < V-1 ∧ m = min{PQ ∪ {m}} ∧ m = {x,y} }
UNTIL pqempty OR i = V-1;    { p ∧ (PQ = ∅ ∨ i = V-1) }

```

Terminierungsfunktion: $t = \min\{|PQ|, (V-i-1)\}$ \Rightarrow divergiert für $E \leq 0$

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Schlussbemerkung:

- bisher unbeachtet: Eingabe des Graphen:

->Eingabe #V, #E;

```
FOR j := 1 TO E DO
```

```
BEGIN
```

```
  ->Eingabe Knoten1, Knoten2, Gewicht;
```

```
  edges[j].w = Gewicht;
```

```
  edges[j].v1 = Knoten1;
```

```
  edges[j].v2 = Knoten2;
```

```
END;
```

- auch hier gilt für die Schleife: Vorbedingung $E > 0$

- für den Algorithmus außer Betracht gelassen, da keinerlei Nachbedingungen aus Eingaben abgeleitet werden können

Proseminar Programmverifikation „minimum spanning tree“ (Kruskal-Algorithmus)

Quellen:

- K. R. Apt & E.-R. Olderog, „Programmverifikation - Sequentielle, parallele und verteilte Programme“, Springer-Verlag 1994

- Internetquellen:

„Wikipedia - die freie Enzyklopädie“

<http://de.wikipedia.org/wiki/Kruskal-Algorithmus>

Michael Gellner: „Der Umgang mit dem Hoare-Kalkül zur Programmverifikation“

<http://wwwswt.informatik.uni->

[rostock.de/deutsch/Mitarbeiter/michael/lehre/pt_2001/Hoare_akt_008.pdf](http://wwwswt.informatik.uni-rostock.de/deutsch/Mitarbeiter/michael/lehre/pt_2001/Hoare_akt_008.pdf)