

Minimum udSpændende Træer (MST)

Træer

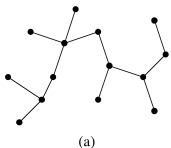
Et (frit/u-rodet) træ er en **uorienteret** graf $G = (V, E)$ som er

- ▶ **Sammenhængende**: der er en sti mellem alle par af knuder.
- ▶ **Acyklisk**: der er ingen kreds af kanter.

Træer

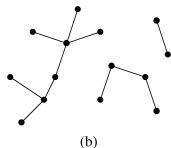
Et (frit/u-rodet) træ er en **uorienteret** graf $G = (V, E)$ som er

- ▶ **Sammenhængende**: der er en sti mellem alle par af knuder.
- ▶ **Acyklisk**: der er ingen kreds af kanter.



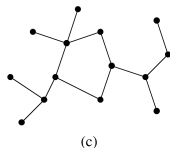
(a)

Træ



(b)

Skov



(c)

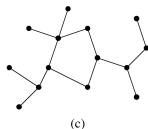
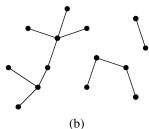
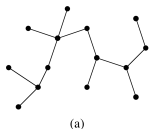
Graf med kreds (ikke træ)

(Uorienteret, acyklisk graf = skov af træer.).

Træer

Sætning (B.2): For *uorienteret* graf $G = (V, E)$ er flg. ækvivalent (gælder det ene, gælder det andet):

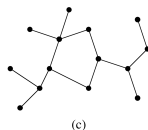
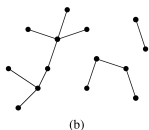
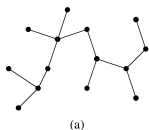
- ▶ G er et træ (dvs. sammenhængende og acyklisk).
- ▶ G er sammenhængende, men er det ikke hvis nogen kant fjernes.
- ▶ G er sammenhængende og $m = n - 1$.
- ▶ G er acyklisk, men er det ikke hvis nogen kant tilføjes.
- ▶ G er acyklisk og $m = n - 1$.
- ▶ Mellem alle par af knuder er der præcis én vej.



Træer

Sætning (B.2): For *uorienteret* graf $G = (V, E)$ er flg. ækvivalent (gælder det ene, gælder det andet):

- ▶ G er et træ (dvs. sammenhængende og acyklisk).
- ▶ G er sammenhængende, men er det ikke hvis nogen kant fjernes.
- ▶ G er sammenhængende og $m = n - 1$.
- ▶ G er acyklisk, men er det ikke hvis nogen kant tilføjes.
- ▶ G er acyklisk og $m = n - 1$.
- ▶ Mellem alle par af knuder er der præcis én vej.



Bevis (ikke pensum): se appendix B.5.

Læs (pensum) appendix B.4 og B.5 for basale definitioner for grafer.

Minimum Spanning Tree (MST)

Udspændende træ for uorienteret, sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.

Minimum Spanning Tree (MST)

Udspændende træ for uorienteret, sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.

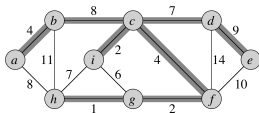
NB: *samme* knudemængde V . Vi tænker fra nu af på T blot som E' .

Minimum Spanning Tree (MST)

Udspændende træ for uorienteret, sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.

NB: *samme* knudemængde V . Vi tænker fra nu af på T blot som E' .

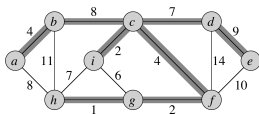


Minimum Spanning Tree (MST)

Udspændende træ for uorienteret, sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.

NB: *samme* knudemængde V . Vi tænker fra nu af på T blot som E' .



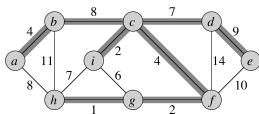
Iflg. sætning B.2 har alle udspændende træer $n - 1$ kanter.

Minimum Spanning Tree (MST)

Udspændende træ for uorienteret, sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.

NB: *samme* knudemængde V . Vi tænker fra nu af på T blot som E' .



Iflg. sætning B.2 har alle udspændende træer $n - 1$ kanter.

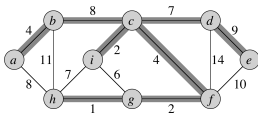
Minimum udspændende Træ (MST) for en *vægtet* uorienteret sammenhængende graf G : et udspændende træ for G som har mindst mulig sum af kantvægte (dvs. intet udspændende træ har mindre sum).

Minimum Spanning Tree (MST)

Udspændende træ for uorienteret, sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.

NB: *samme* knudemængde V . Vi tænker fra nu af på T blot som E' .



Iflg. sætning B.2 har alle udspændende træer $n - 1$ kanter.

Minimum udspændende Træ (MST) for en *vægtet* uorienteret sammenhængende graf G : et udspændende træ for G som har mindst mulig sum af kantvægte (dvs. intet udspændende træ har mindre sum).

Motivation: forbind punkter i et forsyningsnetværk (elektricitet, olie, ...) billigst muligt. Kant i G : mulig forbindelse, vægt: pris for at etablere forbindelse. Dette var motivationen for den første algoritme for problemet (Borůvka, 1926, Østrig-Ungarn, nu Tjekkiet).

Algoritmer for MST

Grundidé er grådig algoritme: byg MST ved at vælge kanterne én efter én ved hjælp af passende lokalt (dvs. lokalt i tid) godt valg.

Algoritmer for MST

Grundidé er grådig algoritme: byg MST ved at vælge kanterne én efter én ved hjælp af passende lokalt (dvs. lokalt i tid) godt valg.

Korrekthed: via den sædvanlige invariant for korrekthed af grådige algoritmer: Hvad vi har bygget indtil nu er en del af en optimal løsning.

Dvs. følgende **Invariant**:

Der findes et MST som indeholder de valgte kanter A .

Algoritmer for MST

Grundidé er grådige algoritme: byg MST ved at vælge kanterne én efter én ved hjælp af passende lokalt (dvs. lokalt i tid) godt valg.

Korrekthed: via den sædvanlige invariant for korrekthed af grådige algoritmer: Hvad vi har bygget indtil nu er en del af en optimal løsning.

Dvs. følgende **Invariant**:

Der findes et MST som indeholder de valgte kanter A .

GENERIC-MST(G, w)

$A = \emptyset$

while A is not a spanning tree

 find an edge (u, v) that is safe for A

$A = A \cup \{(u, v)\}$

return A

Terminologi: **safe** kant for A er en kant som kan tilføjes uden at ødelægge invarianten (mindst én må findes når invarianten gælder og $|A| < n - 1$).

Algoritmer for MST

Invariant: Der findes et MST som indeholder de valgte kanter A .

Algoritmer for MST

Invariant: Der findes et MST som indeholder de valgte kanter A .

GENERIC-MST(G, w)

$A = \emptyset$

while A is not a spanning tree

 find an edge (u, v) that is safe for A

$A = A \cup \{(u, v)\}$

return A

Algoritmer for MST

Invariant: Der findes et MST som indeholder de valgte kanter A .

GENERIC-MST(G, w)

$A = \emptyset$

while A is not a spanning tree

 find an edge (u, v) that is safe for A

$A = A \cup \{(u, v)\}$

return A

- ▶ Initialisering: Enhver sammenhængende graf har et mindst ét ST (via sætningen fra B.5, punkt 2 - fjern kanter til betingelsen nås), og har derfor et MST. Dette indeholder kantmængden \emptyset .

Algoritmer for MST

Invariant: Der findes et MST som indeholder de valgte kanter A .

GENERIC-MST(G, w)

$A = \emptyset$

while A is not a spanning tree

 find an edge (u, v) that is safe for A

$A = A \cup \{(u, v)\}$

return A

- ▶ Initialisering: Enhver sammenhængende graf har et mindst ét ST (via sætningen fra B.5, punkt 2 - fjern kanter til betingelsen nås), og har derfor et MST. Dette indeholder kantmængden \emptyset .
- ▶ Vedligeholdelse: Per definition af safe.

Algoritmer for MST

Invariant: Der findes et MST som indeholder de valgte kanter A .

GENERIC-MST(G, w)

$A = \emptyset$

while A is not a spanning tree

 find an edge (u, v) that is safe for A

$A = A \cup \{(u, v)\}$

return A

- ▶ Initialisering: Enhver sammenhængende graf har et mindst ét ST (via sætningen fra B.5, punkt 2 - fjern kanter til betingelsen nås), og har derfor et MST. Dette indeholder kantmængden \emptyset .
- ▶ Vedligeholdelse: Per definition af safe.
- ▶ Terminering: ethvert (M)ST indeholder præcis $n - 1$ kanter. Da A vokser med én kant per iteration, giver invarianten at algoritmen terminerer, og at A da er et MST (A er indeholdt i et MST, og har samme antal kanter som dette, er derfor lig dette).

Cuts

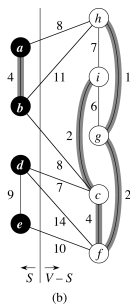
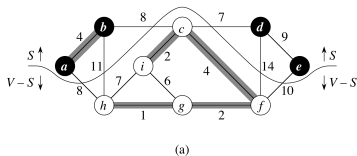
Hvordan finde en safe kant?

Cuts

Hvordan finde en safe kant?

Cut: En delmængde $S \subseteq V$ af knuderne.

Kan ses som en to-delning af knuderne i to mængder S og $V - S$.

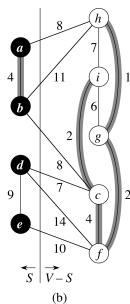
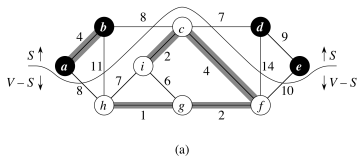


Cuts

Hvordan finde en safe kant?

Cut: En delmængde $S \subseteq V$ af knuderne.

Kan ses som en to-delning af knuderne i to mængder S og $V - S$.



Kant henover cut: en kant i $S \times (V - S)$.

Cut-sætning

Sætning:

Hvis

- ▶ der eksisterer et MST som indeholder A ,
- ▶ S er et cut som A ikke har kanter henover,
- ▶ e er en letteste kant blandt kanterne henover cuttet,

så

- ▶ er e safe for A (dvs. der eksisterer et MST som indeholder $A \cup \{e\}$).

Cut-sætning

Bevis:

- ▶ Der findes et MST T som indeholder A .
- ▶ Vi skal lave et MST T' som indeholder $A \cup \{e\}$.

Cut-sætning

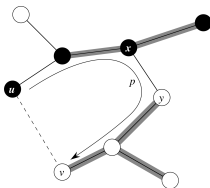
Bevis:

- ▶ Der findes et MST T som indeholder A .
- ▶ Vi skal lave et MST T' som indeholder $A \cup \{e\}$.

Lad $e = (u, v)$ være en letteste kant henover cuttet S .

Da T er sammenhængende, må der være en sti i T mellem u og v , hvorpå der er mindst én kant (x, y) henover cuttet S .

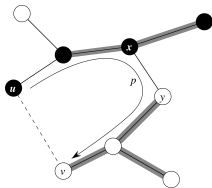
Lad T' være T med (x, y) udskiftet til $e = (u, v)$:



(Viste kanter = T , fede kanter = A , cut er angivet med knudefarver.)

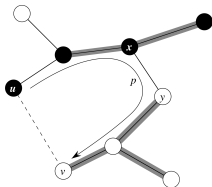
Cut-sætning

Lad T' være T med (x, y) udskiftet til $e = (u, v)$:



Cut-sætning

Lad T' være T med (x, y) udskiftet til $e = (u, v)$:



Som T er T' stadig sammenhængende (i alle stier kan (x, y) erstattes af resten af stien fra u til v , samt kanten (u, v)), og har n knuder og $n - 1$ kanter. T' er derfor et træ (pga. sætning tidligere). Det kan kun være lettere end T . Derfor er T' også et MST.

T' indeholder $A \cup \{e\}$, da den fjernede kant (x, y) ikke er i A , eftersom A ingen kanter har henover cuttet. \square

Brug af cut-sætning i MST-algoritmer

```
GENERIC-MST( $G, w$ )  
   $A = \emptyset$   
  while  $A$  is not a spanning tree  
    find an edge  $(u, v)$  that is safe for  $A$   
     $A = A \cup \{(u, v)\}$   
  return  $A$ 
```

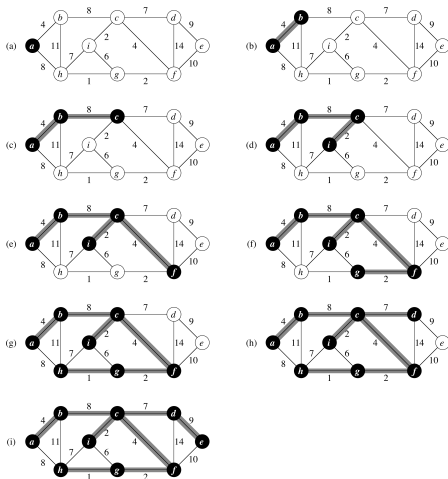
Invariant: Der findes et MST som indeholder de valgte kanter A .

- ▶ En ny kant (u, v) med begge endepunkter i *samme* sammenhængskomponent i $G' = (V, A)$ vil introducere en kreds og dermed ødelægge invarianten. Sådanne er derfor aldrig safe.
- ▶ En ny kant (u, v) med endepunkterne i *forskellige* sammenhængskomponenter C_1 og C_2 i G' er safe, hvis den er en letteste kant ud af C_1 : brug cut-sætning på cuttet C_1 .

Man ser nemt, at hvis A udvides med en kant med endepunkterne i forskellige sammenhængskomponenter C_1 og C_2 i G' , vil det ændre sammenhængskomponenterne i G' ved at C_1 og C_2 slås sammen til én sammenhængskomponent.

Prim-Jarnik MST-algoritmen (Prim 1957, Jarnik 1930)

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent.



Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

```
PRIM( $G, w, r$ )
   $Q = \emptyset$ 
  for each  $u \in G.V$ 
     $u.key = \infty$ 
     $u.\pi = \text{NIL}$ 
  INSERT( $Q, u$ )
  DECREASE-KEY( $Q, r, 0$ )    //  $r.key = 0$ 
  while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in G.Adj[u]$ 
      if  $v \in Q$  and  $w(u, v) < v.key$ 
         $v.\pi = u$ 
        DECREASE-KEY( $Q, v, w(u, v)$ )
```

Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

```
PRIM( $G, w, r$ )
   $Q = \emptyset$ 
  for each  $u \in G.V$ 
     $u.key = \infty$ 
     $u.\pi = \text{NIL}$ 
  INSERT( $Q, u$ )
  DECREASE-KEY( $Q, r, 0$ )    //  $r.key = 0$ 
  while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in G.Adj[u]$ 
      if  $v \in Q$  and  $w(u, v) < v.key$ 
         $v.\pi = u$ 
        DECREASE-KEY( $Q, v, w(u, v)$ )
```

Korrekthed: via cut-sætningen og invarianten.

Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

```
PRIM( $G, w, r$ )
   $Q = \emptyset$ 
  for each  $u \in G.V$ 
     $u.key = \infty$ 
     $u.\pi = \text{NIL}$ 
  INSERT( $Q, u$ )
  DECREASE-KEY( $Q, r, 0$ )    //  $r.key = 0$ 
  while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in G.Adj[u]$ 
      if  $v \in Q$  and  $w(u, v) < v.key$ 
         $v.\pi = u$ 
        DECREASE-KEY( $Q, v, w(u, v)$ )
```

Korrekthed: via cut-sætningen og invarianten.

Køretid:

Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

```
PRIM( $G, w, r$ )
   $Q = \emptyset$ 
  for each  $u \in G.V$ 
     $u.key = \infty$ 
     $u.\pi = \text{NIL}$ 
  INSERT( $Q, u$ )
  DECREASE-KEY( $Q, r, 0$ )    //  $r.key = 0$ 
  while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in G.Adj[u]$ 
      if  $v \in Q$  and  $w(u, v) < v.key$ 
         $v.\pi = u$ 
        DECREASE-KEY( $Q, v, w(u, v)$ )
```

Korrekthed: via cut-sætningen og invarianten.

Køretid: n INSERT, n EXTRACTMIN, m DECREASEKEY

Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

```
PRIM( $G, w, r$ )
   $Q = \emptyset$ 
  for each  $u \in G.V$ 
     $u.key = \infty$ 
     $u.\pi = \text{NIL}$ 
  INSERT( $Q, u$ )
  DECREASE-KEY( $Q, r, 0$ )    //  $r.key = 0$ 
  while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in G.Adj[u]$ 
      if  $v \in Q$  and  $w(u, v) < v.key$ 
         $v.\pi = u$ 
        DECREASE-KEY( $Q, v, w(u, v)$ )
```

Korrekthed: via cut-sætningen og invarianten.

Køretid: n INSERT, n EXTRACTMIN, m DECREASEKEY på prioritetskø af størrelse $O(n)$,

Prim-Jarnik MST-algoritmen

Tager udgangspunkt i en (tilfældig) startknode r . Udvider hele tiden r 's sammenhængskomponent C .

En knude $v \in C - S$ opbevarer information om sin korteste kant henover cut i $v.key$ og $v.\pi$. Mængden A er $\{(v, v.\pi) \mid v \in C - \{s\}\}$.

$C - S$ opbevares i en (min-)prioritetskø.

```
PRIM( $G, w, r$ )
   $Q = \emptyset$ 
  for each  $u \in G.V$ 
     $u.key = \infty$ 
     $u.\pi = \text{NIL}$ 
  INSERT( $Q, u$ )
  DECREASE-KEY( $Q, r, 0$ )    //  $r.key = 0$ 
  while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in G.Adj[u]$ 
      if  $v \in Q$  and  $w(u, v) < v.key$ 
         $v.\pi = u$ 
        DECREASE-KEY( $Q, v, w(u, v)$ )
```

Korrekthed: via cut-sætningen og invarianten.

Køretid: n INSERT, n EXTRACTMIN, m DECREASEKEY på prioritetskø af størrelse $O(n)$, i alt $O(m \log n)$.

Kruskal MST-algoritmen (1956)

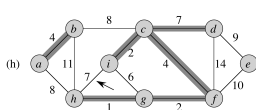
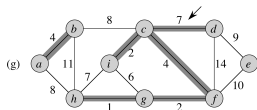
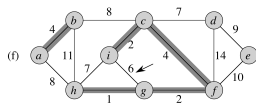
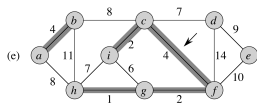
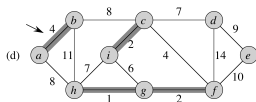
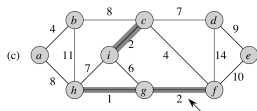
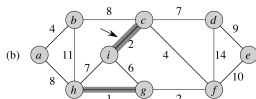
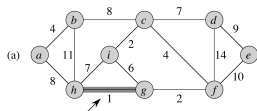
Forsøger at tilføje kanter til A i global letteste-først-orden.

Kruskal MST-algoritmen (1956)

Forsøger at tilføje kanter til A i global letteste-først-orden.

Recall: Vi kan kun tilføje kant (u, v) til A hvis der ikke laves en kreds, dvs. hvis u og v ligger i forskellige sammenhængskomponenter. Hvis (u, v) tilføjes, vil disse to sammenhængskomponenter blive til én bagefter.

Kruskal MST-algorithmen



Kruskal MST-algoritmen

Vedligeholder sammenhængskomponenterne i $G' = (V, A)$ ved hjælp af en *disjoint-set* datastruktur på V :

MAKE-SET(x), UNION(x, y) FIND-SET(x)

Kruskal MST-algoritmen

Vedligeholder sammenhængskomponenterne i $G' = (V, A)$ ved hjælp af en *disjoint-set* datastruktur på V :

MAKE-SET(x), UNION(x, y) FIND-SET(x)

Mere præcist:

KRUSKAL(G, w)

$A = \emptyset$

for each vertex $v \in G.V$

 MAKE-SET(v)

sort the edges of $G.E$ into nondecreasing order by weight w

for each (u, v) taken from the sorted list

if FIND-SET(u) \neq FIND-SET(v)

$A = A \cup \{(u, v)\}$

 UNION(u, v)

return A

Kruskal MST-algoritmen

KRUSKAL(G, w)

$A = \emptyset$

for each vertex $v \in G.V$

 MAKE-SET(v)

sort the edges of $G.E$ into nondecreasing order by weight w

for each (u, v) taken from the sorted list

if FIND-SET(u) \neq FIND-SET(v)

$A = A \cup \{(u, v)\}$

 UNION(u, v)

return A

Klart ud fra diskussion side 11 om sammenhængskomponenter at:

- ▶ Datastrukturen vedligeholder sammenhængskomponenterne i $G' = (V, A)$.
- ▶ En kant undersøgt (IF-sætningen) har begge endepunkter i samme sammenhængskomponent efter undersøgelsen, uanset udfaldet af testen i IF-sætningen. Da sammenhængskomponenter i G' kun slås sammen undervejs, gælder dette også for kanten i resten af algoritmen.

Kruskal, korrekthed

På det tidspunkt hvor algoritmen tilføjer en kant (u, v) til A , ligger u og v i forskellige sammenhængskomponenter C_1 og C_2 i $G' = (V, A)$. Vi ser vi på cuttet givet ved u 's sammenhængskomponent C_1 .

Alle lettere kanter er allerede undersøgt, og har derfor begge endepunkter i samme sammenhængskomponent i G' . Derfor er (u, v) en letteste kant henover dette cut, og vi kan derfor bruge cut-sætningen.

Kruskal, korrekthed

På det tidspunkt hvor algoritmen tilføjer en kant (u, v) til A , ligger u og v i forskellige sammenhængskomponenter C_1 og C_2 i $G' = (V, A)$. Vi ser vi på cuttet givet ved u 's sammenhængskomponent C_1 .

Alle lettere kanter er allerede undersøgt, og har derfor begge endepunkter i samme sammenhængskomponent i G' . Derfor er (u, v) en letteste kant henover dette cut, og vi kan derfor bruge cut-sætningen.

Når algoritmen stopper, er alle kanter undersøgt. Enhver kant i inputgrafene $G = (V, E)$ har derfor begge endepunkter i samme sammenhængskomponent i $G' = (V, A)$. Sådanne kanter kan ikke tilføjes A uden at introducere en kreds.

Så A er selv det MST (fra invarianten), som indeholder A , da ingen kanter kan tilføjes A . Så algoritmen er korrekt.

Kruskal, korrekthed

På det tidspunkt hvor algoritmen tilføjer en kant (u, v) til A , ligger u og v i forskellige sammenhængskomponenter C_1 og C_2 i $G' = (V, A)$. Vi ser vi på cuttet givet ved u 's sammenhængskomponent C_1 .

Alle lettere kanter er allerede undersøgt, og har derfor begge endepunkter i samme sammenhængskomponent i G' . Derfor er (u, v) en letteste kant henover dette cut, og vi kan derfor bruge cut-sætningen.

Når algoritmen stopper, er alle kanter undersøgt. Enhver kant i inputgrafene $G = (V, E)$ har derfor begge endepunkter i samme sammenhængskomponent i $G' = (V, A)$. Sådanne kanter kan ikke tilføjes A uden at introducere en kreds.

Så A er selv det MST (fra invarianten), som indeholder A , da ingen kanter kan tilføjes A . Så algoritmen er korrekt.

Bemærk at der laves præcis $n - 1$ UNION operationer undervejs, da hver tilføjer én kant til A , og da et MST har $n - 1$ kanter.

Kruskal, køretid

Arbejde:

Sortér m kanter

Lav n MAKE-SET, $n - 1$ UNION, m FIND-SET.

Kruskal, køretid

Arbejde:

Sortér m kanter

Lav n MAKE-SET, $n - 1$ UNION, m FIND-SET.

Fra tidligere: der findes en datastruktur for disjoint-sets hvor

- ▶ n MAKE-SET(x)
- ▶ $n - 1$ UNION(x, y)
- ▶ m FIND-SET(x)

tager i alt $O(m + n \log n)$ tid.

Kruskal, køretid

Arbejde:

Sortér m kanter

Lav n MAKE-SET, $n - 1$ UNION, m FIND-SET.

Fra tidligere: der findes en datastruktur for disjoint-sets hvor

- ▶ n MAKE-SET(x)
- ▶ $n - 1$ UNION(x, y)
- ▶ m FIND-SET(x)

tager i alt $O(m + n \log n)$ tid.

Samlet køretid for Kruskal er

$$O(m \log m)$$

eftersom $m \geq n - 1$, da inputgrafene er sammenhængende.