

Minimum udSpændende Træer (MST)

Træer

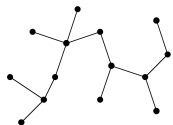
Et (frit/u-rodet) træ er en *uorienteret* graf $G = (V, E)$ som er

- ▶ 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.



(a)



(b)

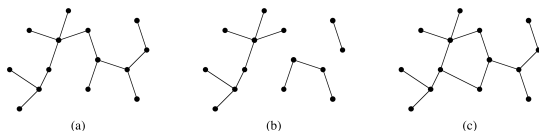


(c)

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)

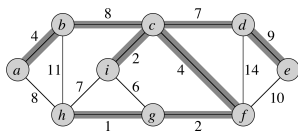
Udspændende træ for sammenhængende graf $G = (V, E)$:

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ. NB: *samme* knudemængde V .

Minimum Spanning Tree (MST)

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

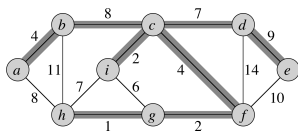
En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ. NB: *samme* knudemængde V .



Minimum Spanning Tree (MST)

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

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ. NB: *samme* knudemængde V .

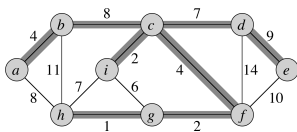


Iflg. sætning ovenfor har alle udspændende træer samme antal kanter ($m = n - 1$).

Minimum Spanning Tree (MST)

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

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ. NB: *samme* knudemængde V .



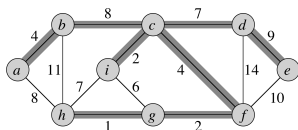
Iflg. sætning ovenfor har alle udspændende træer samme antal kanter ($m = n - 1$).

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

Minimum Spanning Tree (MST)

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

En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ. NB: *samme* knudemængde V .



Iflg. sætning ovenfor har alle udspændende træer samme antal kanter ($m = n - 1$).

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

Motivation: forbind punkter i et forsyningsnetværk (elektricitet, olie, ...) billigt 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 (grådig algoritme): Byg MST ved at vælge kanterne én efter én. Vedligehold følgende **Invariant**: Der findes et MST som indeholder de valgte kanter A .

Algoritmer for MST

Grundidé (grådig algoritme): Byg MST ved at vælge kanterne én efter én. Vedligehold 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

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

Algoritmer for MST

Grundidé (grådige algoritme): Byg MST ved at vælge kanterne én efter én. Vedligehold 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  $A$ 
```

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

- ▶ 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

Grundidé (grådige algoritme): Byg MST ved at vælge kanterne én efter én. Vedligehold 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  $A$ 
```

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

- ▶ 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 kantomængden \emptyset .
- ▶ Vedligeholdelse: Per definition af safe.

Algoritmer for MST

Grundidé (grådige algoritme): Byg MST ved at vælge kanterne én efter én. Vedligehold 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  $A$ 
```

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

- ▶ 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

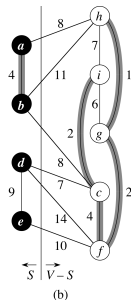
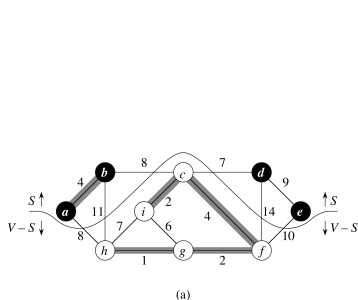
Hvordan finde en safe kant?

Cuts

Hvordan finde en safe kant?

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

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

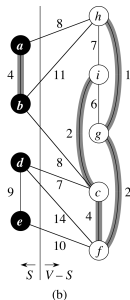
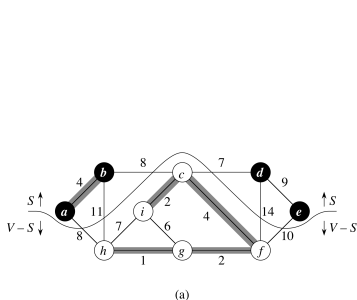


Cuts

Hvordan finde en safe kant?

Cut: En delmængde $S \subseteq$ 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 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

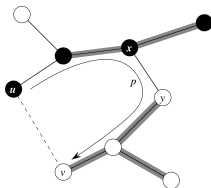
Bevis:

- ▶ 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 .

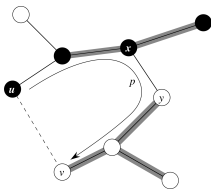
Lad T' være T med (x, y) udskiftet til $e = (u, v)$:



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

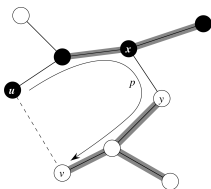
Cut-sætning

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. Det er derfor et træ (pga. sætning tidligere). Det kan kun være lettere end T . Det indeholder $A \cup \{e\}$ (da fjernede kant (x, y) ikke er i A).

Brug af cut-sætning

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 .

Grafen $G' = (V, A)$ er acyklisk pga. invarianten. Hver sammenhængskomponent er derfor et træ.

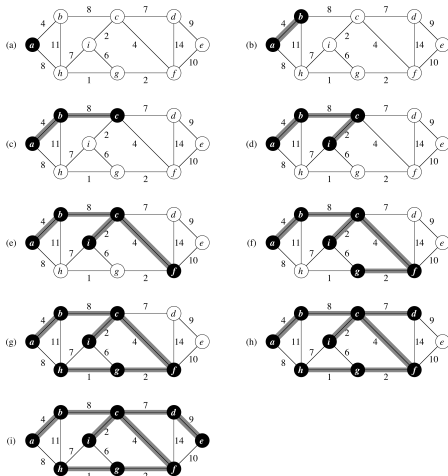
En ny kant (u, v) med begge endepunkter i samme sammenhængskomponent vil introducere en kreds og dermed ødelægge invarianten. Sådanne er derfor aldrig safe.

En ny kant (u, v) med endepunkterne i forskellige sammenhængskomponenter C_1 og C_2 er safe, hvis den er den letteste kant ud af enten C_1 eller C_2 (brug cut-sætning på cuttet C_1 eller C_2).

Man ser nemt at en sådan kant vil ændre mængden af sammenhængskomponenter 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 $v.\pi$.

$C - S$ 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 $v.\pi$.

$C - S$ 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 $v.\pi$.

$C - S$ 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 $v.\pi$.

$C - S$ 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 $v.\pi$.

$C - S$ 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 $v.\pi$.

$C - S$ 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 $v.\pi$.

$C - S$ 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)

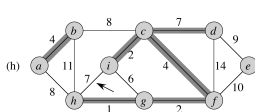
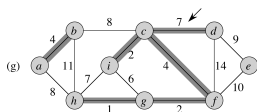
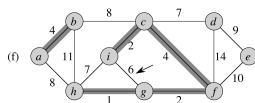
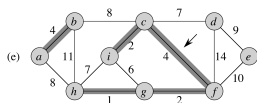
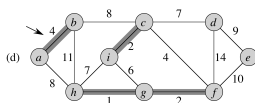
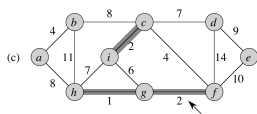
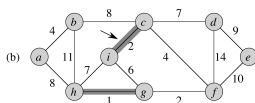
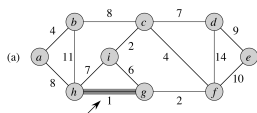
Forsøger at tilføje kanter til A i global letteste-først-orden.

Kruskal MST-algoritmen (1956)

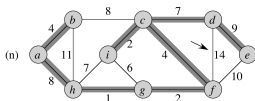
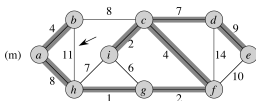
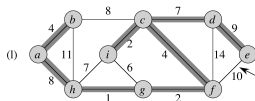
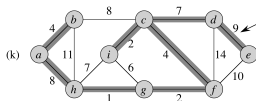
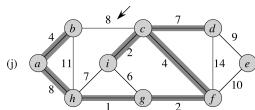
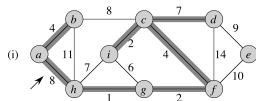
Forsøger at tilføje kanter til A i global letteste-først-orden.

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-algoritmen (1956)



Kruskal MST-algoritmen (1956)



Kruskal MST-algoritmen (1956)

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 (1956)

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  $A$ 
```

Klart ud fra tidligere diskussion om sammenhængskomponenter, at datastrukturen vedligeholder sammenhængskomponenterne i $G' = (V, A)$. Da sammenhængskomponenter kun slås sammen, har alle undersøgte kanter begge endepunkter i samme sammenhængskomponent

Kruskal MST-algoritmen (1956)

Fra tidligere kendes:

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 MST-algoritmen (1956)

Korrekthed:

Når algoritmen tilføjer en kant (u, v) til A ser vi på cuttet givet ved knudemængden $\text{FIND-SET}(u)$. A har ikke kanter hen over dette cut (da alle undersøgte kanter, herunder dem i A , har begge endepunkter i samme sammenhængskomponent), og (u, v) er en letteste kant henover dette cut (da alle lettere kanter er undersøgt, og derfor har begge endepunkter i samme sammenhængskomponent).

Kruskal MST-algoritmen (1956)

Korrekthed:

Når algoritmen tilføjer en kant (u, v) til A ser vi på cuttet givet ved knudemængden $\text{FIND-SET}(u)$. A har ikke kanter hen over dette cut (da alle undersøgte kanter, herunder dem i A , har begge endepunkter i samme sammenhængskomponent), og (u, v) er en letteste kant henover dette cut (da alle lettere kanter er undersøgt, og derfor har begge endepunkter i samme sammenhængskomponent).

Når algoritmen stopper, er alle kanter undersøgt og alle kanter har derfor begge endepunkter i samme sammenhængskomponent. Da den oprindelige graf (V, E) er sammenhængende, må der nu være én sammenhængskomponent. Derfor er der lavet præcis $n - 1$ unions, og dermed er $|A| = n - 1$. Så A er selv det MST, som indeholder A (fra invarianten).

Kruskal MST-algoritmen (1956)

Korrekthed:

Når algoritmen tilføjer en kant (u, v) til A ser vi på cuttet givet ved knudemængden $\text{FIND-SET}(u)$. A har ikke kanter hen over dette cut (da alle undersøgte kanter, herunder dem i A , har begge endepunkter i samme sammenhængskomponent), og (u, v) er en letteste kant henover dette cut (da alle lettere kanter er undersøgt, og derfor har begge endepunkter i samme sammenhængskomponent).

Når algoritmen stopper, er alle kanter undersøgt og alle kanter har derfor begge endepunkter i samme sammenhængskomponent. Da den oprindelige graf (V, E) er sammenhængende, må der nu være én sammenhængskomponent. Derfor er der lavet præcis $n - 1$ unions, og dermed er $|A| = n - 1$. Så A er selv det MST, som indeholder A (fra invarianten).

Køretid:

Kruskal MST-algoritmen (1956)

Korrekthed:

Når algoritmen tilføjer en kant (u, v) til A ser vi på cuttet givet ved knudemængden $\text{FIND-SET}(u)$. A har ikke kanter hen over dette cut (da alle undersøgte kanter, herunder dem i A , har begge endepunkter i samme sammenhængskomponent), og (u, v) er en letteste kant henover dette cut (da alle lettere kanter er undersøgt, og derfor har begge endepunkter i samme sammenhængskomponent).

Når algoritmen stopper, er alle kanter undersøgt og alle kanter har derfor begge endepunkter i samme sammenhængskomponent. Da den oprindelige graf (V, E) er sammenhængende, må der nu være én sammenhængskomponent. Derfor er der lavet præcis $n - 1$ unions, og dermed er $|A| = n - 1$. Så A er selv det MST, som indeholder A (fra invarianten).

Køretid:

Sortér m kanter, lav n MAKE-SET, $n - 1$ UNION, m FIND-SET.

Kruskal MST-algoritmen (1956)

Korrekthed:

Når algoritmen tilføjer en kant (u, v) til A ser vi på cuttet givet ved knudemængden $\text{FIND-SET}(u)$. A har ikke kanter hen over dette cut (da alle undersøgte kanter, herunder dem i A , har begge endepunkter i samme sammenhængskomponent), og (u, v) er en letteste kant henover dette cut (da alle lettere kanter er undersøgt, og derfor har begge endepunkter i samme sammenhængskomponent).

Når algoritmen stopper, er alle kanter undersøgt og alle kanter har derfor begge endepunkter i samme sammenhængskomponent. Da den oprindelige graf (V, E) er sammenhængende, må der nu være én sammenhængskomponent. Derfor er der lavet præcis $n - 1$ unions, og dermed er $|A| = n - 1$. Så A er selv det MST, som indeholder A (fra invarianten).

Køretid:

Sortér m kanter, lav n MAKE-SET, $n - 1$ UNION, m FIND-SET.

I alt $O(m \log m)$ [eftersom $m \geq n - 1$, da grafen er sammenhængende].