Minimum udSpændende Træer (MST)

Træer

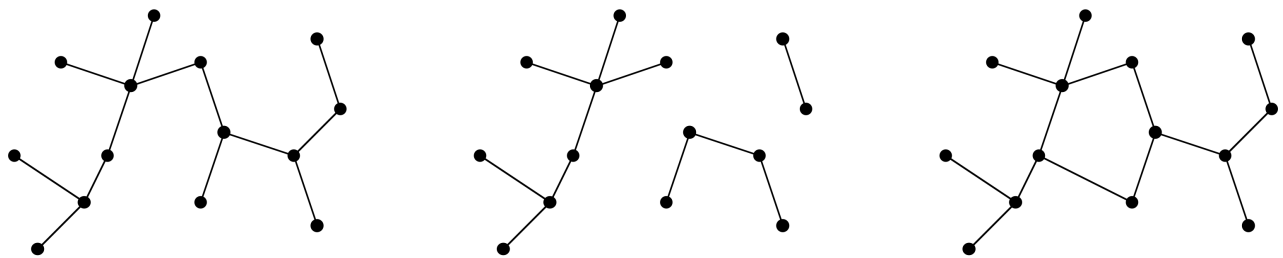
Et (frit/u-rodet) træ er en *uorienteret* graf $G = (V, E)$ som er

- Sammenhængende: der er en sti mellem alle par af knuder.
- Acyklisk: der er ingen lukket kreds af kanter (med ≥ 3 knuder, alle forskellige).

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 lukket kreds af kanter (med ≥ 3 knuder, alle forskellige).

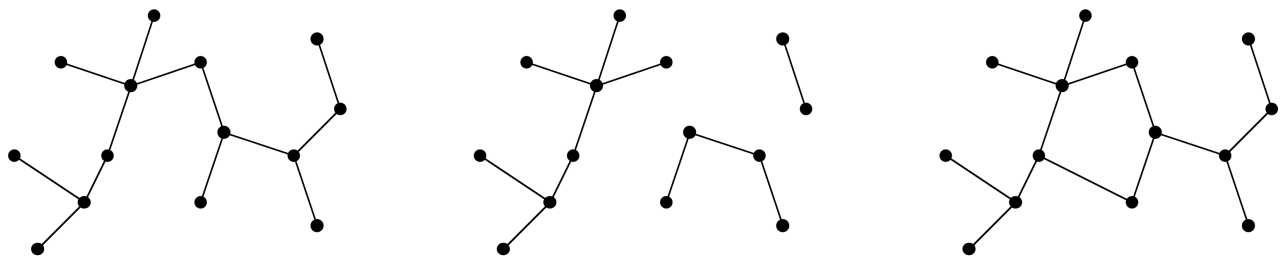


Træ Skov Graf med kreds

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 lukket kreds af kanter (med ≥ 3 knuder, alle forskellige).



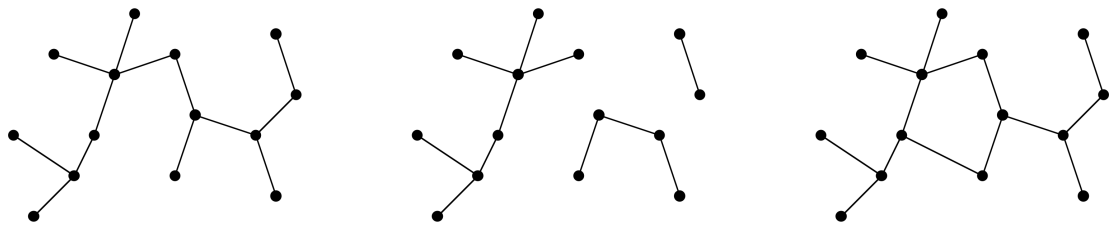
Træ Skov Graf med kreds

Uorienteret, sammenhængende graf: skov af træer.

Træer

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

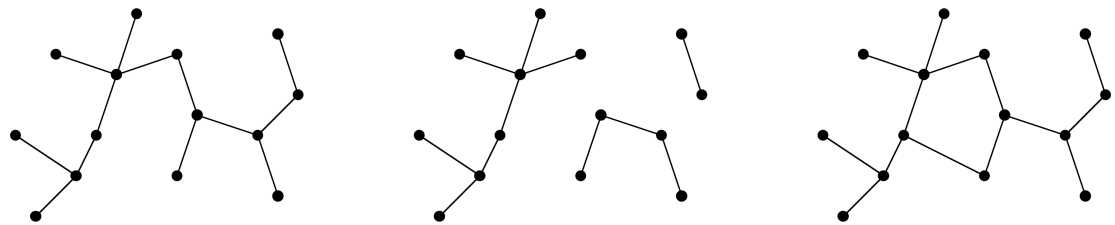
- G er et træ.
- G er sammenhængende, men er det ikke hvis en vilkårlig kant fjernes.
- G er acyklisk, men er det ikke hvis en vilkårlig kant tilføjes.
- G er sammenhængende og $m = n - 1$.
- G er acyklisk og $m = n - 1$.
- Mellem alle par af knuder er der præcis een vej.



Træer

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

- G er et træ.
- G er sammenhængende, men er det ikke hvis en vilkårlig kant fjernes.
- G er acyklisk, men er det ikke hvis en vilkårlig kant tilføjes.
- G er sammenhængende og $m = n - 1$.
- G er acyklisk og $m = n - 1$.
- Mellem alle par af knuder er der præcis een 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)

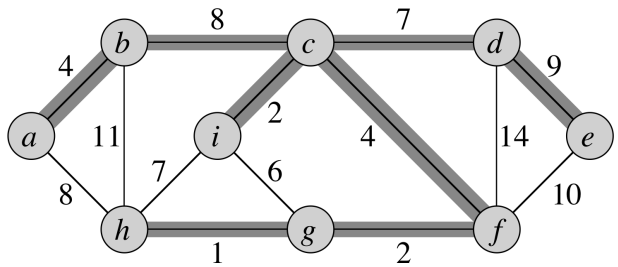
Udspændende træ for sammenhængende graf $G = (V, E)$:

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

Minimum Spanning Tree (MST)

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

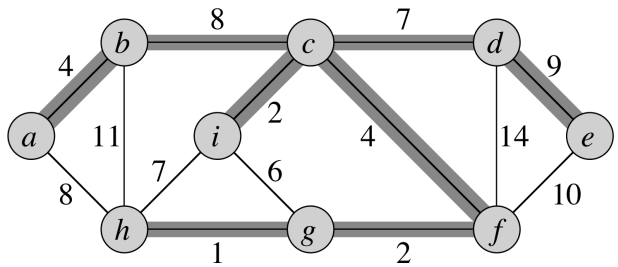
En delgraf $T = (V, E')$, $E' \subseteq E$, som er et træ.



Minimum Spanning Tree (MST)

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

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

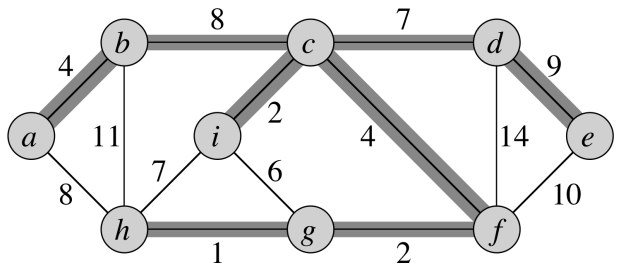


Iflg. sætning ovenfor har alle udspændende træer samme antal kanter ($m = n - 1$).

Minimum Spanning Tree (MST)

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

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



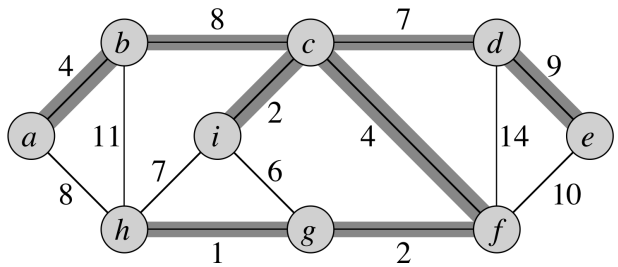
Iflg. sætning ovenfor har alle udspændende træer samme antal kanter ($m = n - 1$).

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

Minimum Spanning Tree (MST)

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

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



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 minimal sum af kantvægte.

Motivation: forbind punkter i et forsyningsnetværk (elektricitet, olie,...) billigst muligt (kant = mulig forbindelse, vægt = pris for at etablere forbindelse).

Algoritmer for MST

Grundidé: Byg MST ved at vælge kanterne een efter een. I alt $n-1$ skridt ($m = n-1$ for træer).

Algoritmer for MST

Grundidé: Byg MST ved at vælge kanterne een efter een. I alt $n-1$ skridt ($m = n-1$ for træer).

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

Invariant: Der eksisterer et MST som indeholder kanterne i A.

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

Algoritmer for MST

Grundidé: Byg MST ved at vælge kanterne een efter een. I alt $n-1$ skridt ($m = n-1$ for træer).

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

Invariant: Der eksisterer et MST som indeholder kanterne i A.

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

- Initialisering: Enhver sammenhængende graf har et ST (via sætning ovenfor, andet punkt), derfor et MST.

Algoritmer for MST

Grundidé: Byg MST ved at vælge kanterne een efter een. I alt $n-1$ skridt ($m = n-1$ for træer).

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

Invariant: Der eksisterer et MST som indeholder kanterne i A.

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

- Initialisering: Enhver sammenhængende graf har et ST (via sætning ovenfor, andet punkt), derfor et MST.
- Vedligeholdelse: OK per definition af safe.

Algoritmer for MST

Grundidé: Byg MST ved at vælge kanterne een efter een. I alt $n-1$ skridt ($m = n-1$ for træer).

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

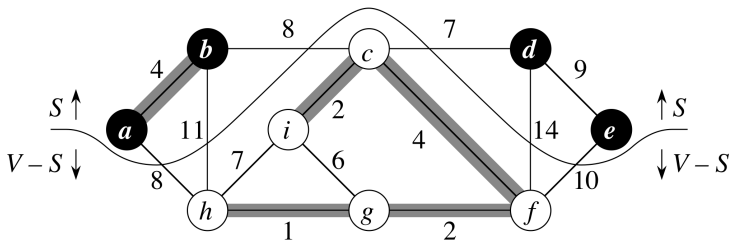
Invariant: Der eksisterer et MST som indeholder kanterne i A.

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

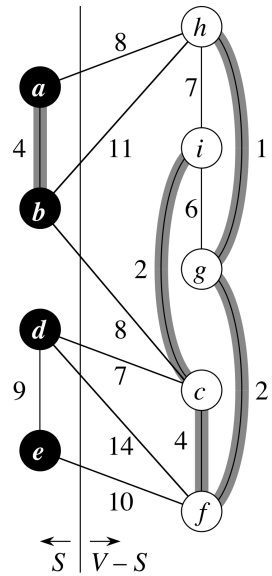
- Initialisering: Enhver sammenhængende graf har et ST (via sætning ovenfor, andet punkt), derfor et MST.
- Vedligeholdelse: OK per definition af safe.
- Terminering: ethvert (MS)T indeholder præcis $n-1$ kanter (se sætning ovenfor). Da $|A|$ vokser med een kant per iteration, giver invarianten at algoritmen terminerer, og at A er et MST.

Cut-sætning

Cut: en to-deling af knuderne i S og $V - S$.



(a)



(b)

Cut-sætning

Sætning:

Hvis

- der eksisterer et MST som indeholder A,
- S er et cut som A ikke har kanter henover (dvs. $A \cap S \times (V - S) = \emptyset$),
- $e \in S \times (V - S)$ er en letteste kant blandt kanterne $S \times (V - S)$ henover cuttet,

så

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

Cut-sætning

Bevis:

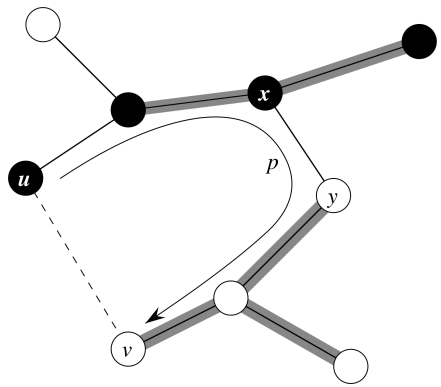
- Der findes et MST T som indeholder A.
- Vi skal lave et MST T' som indeholder $A \cup \{e\}$.

Cut-sætning

Bevis:

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

$T' = T$ med en kant udskiftet:

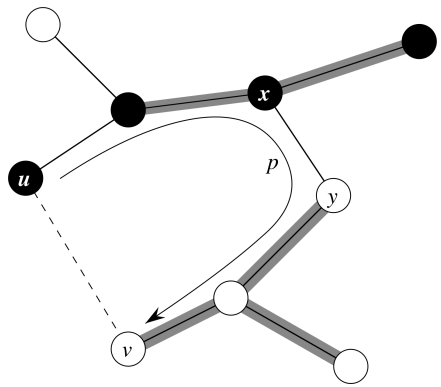


Cut-sætning

Bevis:

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

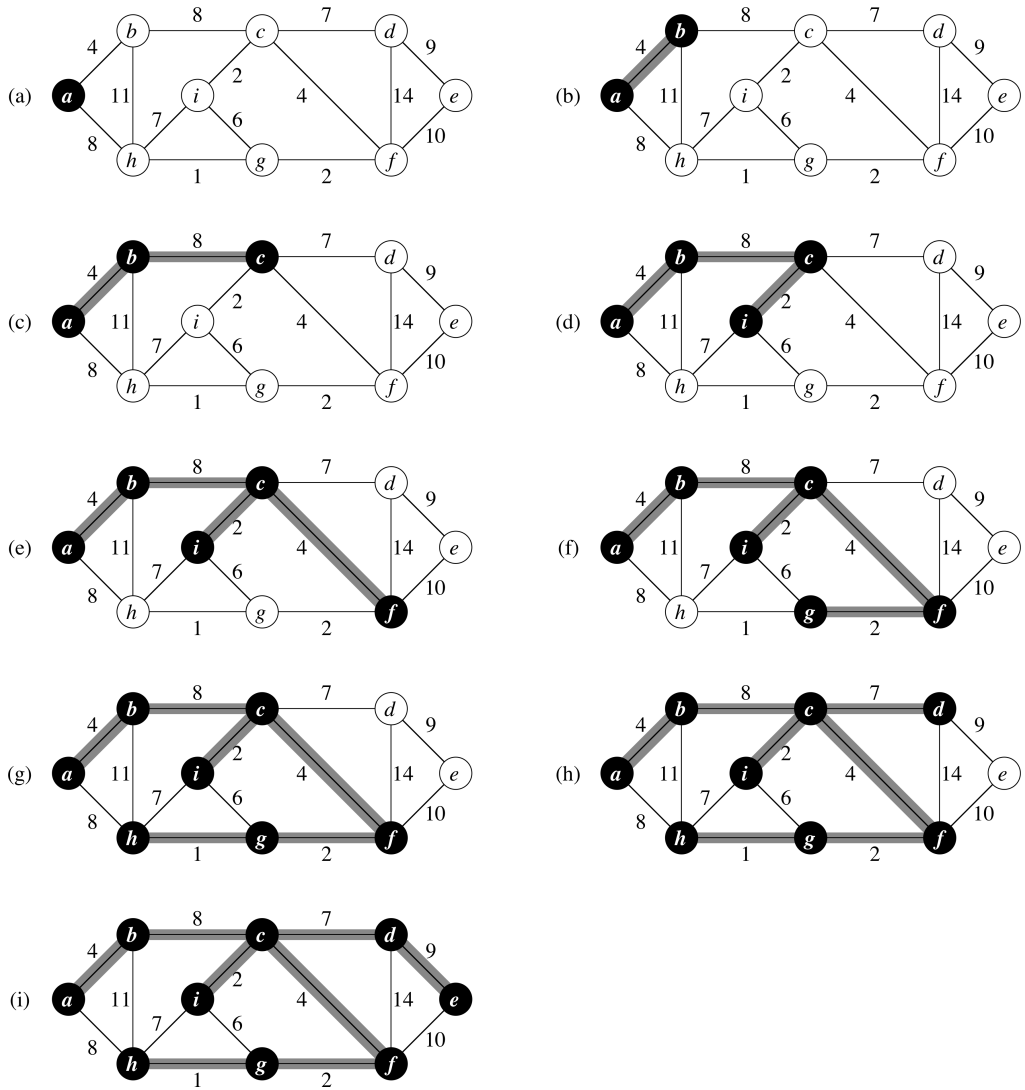
$T' = T$ med en kant udskiftet:



Som T er T' stadig sammenhængende og har T $n-1$ kanter. Det er derfor et træ (pga. sætning tidligere). Det er klart stadig udspændende. Det kan kun være lettere end T. Det indeholder $A \cup \{e\}$ (da fjernede kant ikke er i A).

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

A er eet træ. Bruger cut-sætningen med $S =$ alle knuder i A.



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

A er eet træ. Bruger cut-sætningen med $S =$ alle knuder i A.

Invariant: En knude $v \in V - S$ opbevarer information om sin korteste kant henover cut i v.key og $v.\pi$.

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

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

A er eet træ. Bruger cut-sætningen med $S =$ alle knuder i A.

Invariant: En knude $v \in V - S$ opbevarer information om sin korteste kant henover cut i $v.\text{key}$ og $v.\pi$.

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

```
PRIM(G, w, r)
  Q = ∅
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
      INSERT(Q, u)
  DECREASE-KEY(Q, r, 0)        // r.key = 0
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              DECREASE-KEY(Q, v, w(u, v))
```

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

A er eet træ. Bruger cut-sætningen med $S =$ alle knuder i A.

Invariant: En knude $v \in V - S$ opbevarer information om sin korteste kant henover cut i $v.\text{key}$ og $v.\pi$.

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

```
PRIM(G, w, r)
  Q = ∅
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
      INSERT(Q, u)
  DECREASE-KEY(Q, r, 0)        // r.key = 0
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = 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)

A er en skov. Bruger cut-sætningen med $S =$ knuderne i eet af træerne i A.

Kruskal MST-algoritmen (1956)

A er en skov. Bruger cut-sætningen med $S =$ knuderne i eet af træerne i A.

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

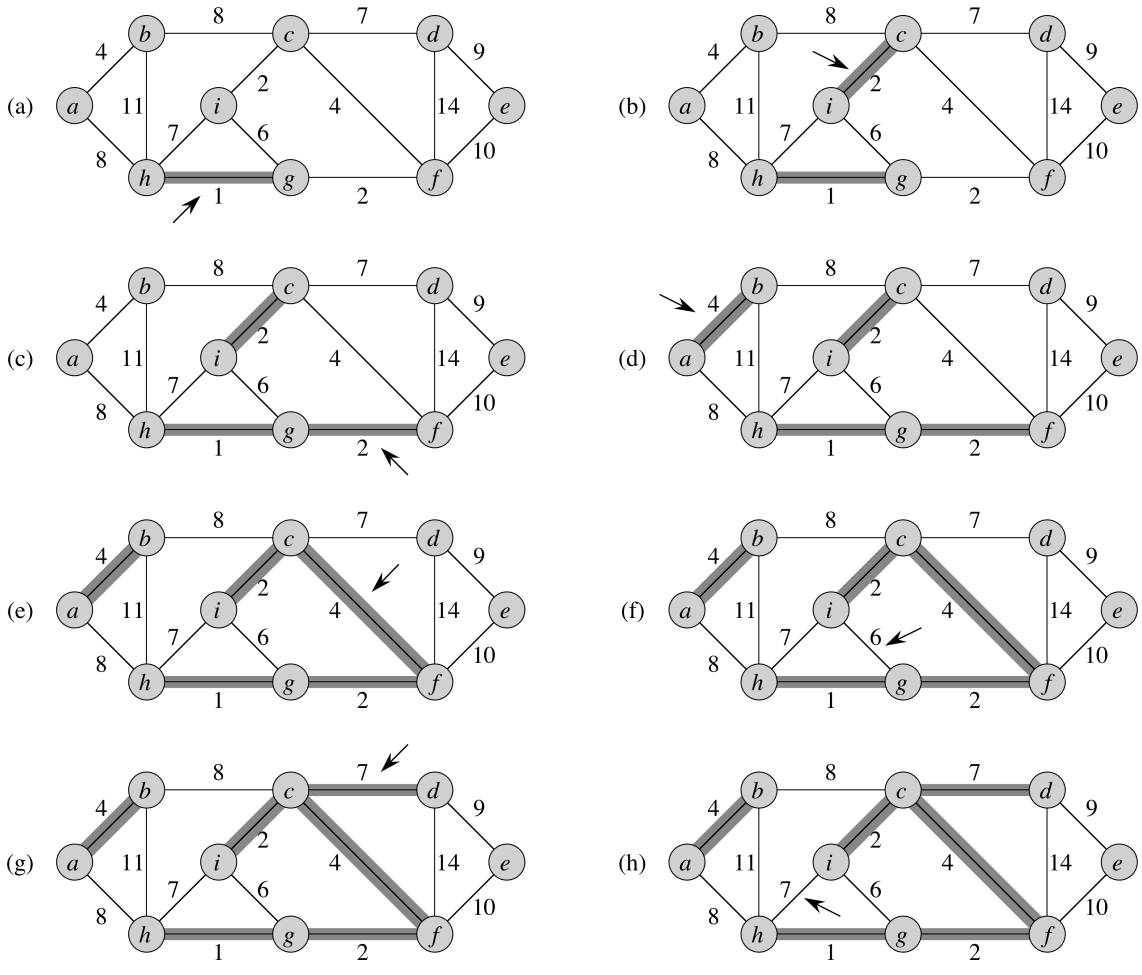
Kruskal MST-algoritmen (1956)

A er en skov. Bruger cut-sætningen med $S =$ knuderne i eet af træerne i A.

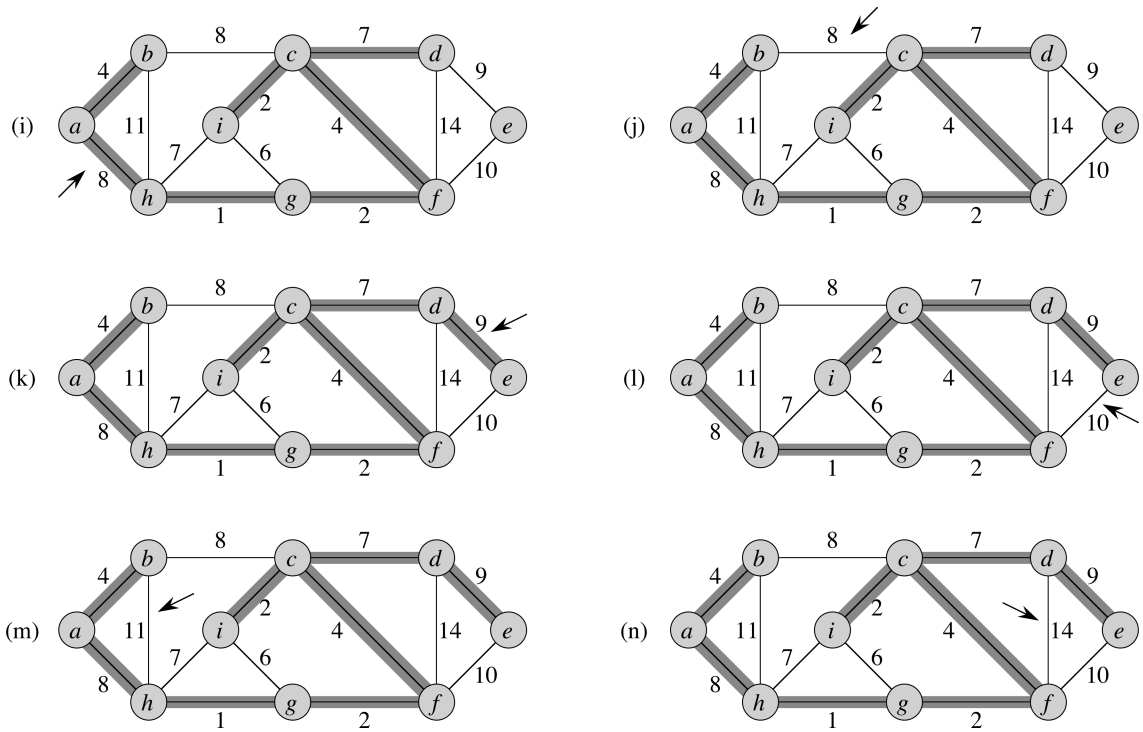
Forsøger at tilføje kanter til A i letteste-først-orden.

Tilføjer kun kant (u, v) til A hvis der ikke laves en kreds, dvs. hvis u og v ligger i forskellige træer. Hvis (u, v) tilføjes, vil disse to træer blive til eet bagefter.

Kruskal MST-algoritmen (1956)



Kruskal MST-algoritmen (1956)

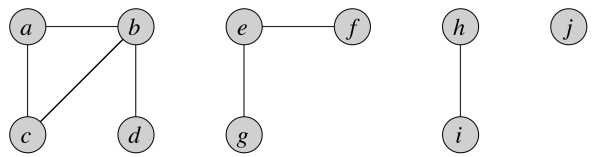


Kruskal MST-algoritmen (1956)

Tilføjer kun kant (u, v) til A hvis der ikke laves en kreds, dvs. hvis u og v ligger i forskellige træer. Hvis (u, v) tilføjes, vil disse to træer blive til eet bagefter.

Vedligeholder opdelingen i træer i A ved hjælp af en disjoint-set datastruktur på V:

$$\textsc{Make-Set}(x),\ \textsc{Union}(x, y)\ \textsc{Find-Set}(x)$$



(a)

Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

(b)

Kruskal MST-algoritmen (1956)

Lemma: 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)

Invariant: Når en kant forsøges tilføjet A, har enhver tidligere forsøgt kant begge sine endepunkter i samme træ i A.

Kruskal MST-algoritmen (1956)

Invariant: Når en kant forsøges tilføjet A, har enhver tidligere forsøgt kant begge sine endepunkter i samme træ i A.

```
KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ 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) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

Kruskal MST-algoritmen (1956)

Invariant: Når en kant forsøges tilføjet A, har enhver tidligere forsøgt kant begge sine endepunkter i samme træ i A.

```
KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ 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) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

Korrekthed: via cut-sætningen og invarianten.

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

I alt $O(m \log m)$.