MAT1030 – Diskret Matematikk

Forelesning 25: Trær

Roger Antonsen

Institutt for informatikk, Universitetet i Oslo

29. april 2009

(Sist oppdatert: 2009-04-29 00:28)

Forelesning 25

Litt repetisjon

Litt repetisjon

- Vi har snakket om grafer og trær. Av begreper vi så på var følgende.

Litt repetisjon

- Vi har snakket om grafer og trær. Av begreper vi så på var følgende.
- Eulerstier og Eulerkretser

Litt repetisjon

- Vi har snakket om grafer og trær. Av begreper vi så på var følgende.
- Eulerstier og Eulerkretser
- Hamiltonstier og Hamiltonkretser

Litt repetisjon

- Vi har snakket om grafer og trær. Av begreper vi så på var følgende.
- Eulerstier og Eulerkretser
- Hamiltonstier og Hamiltonkretser
- Vektede grafer

Litt repetisjon

- Vi har snakket om grafer og trær. Av begreper vi så på var følgende.
- Eulerstier og Eulerkretser
- Hamiltonstier og Hamiltonkretser
- Vektede grafer
- Minimale utspennende trær

Litt repetisjon

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.
- Hvis G er en sammenhengende graf, vil et utspennende tre være en delgraf T slik at

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.
- Hvis G er en sammenhengende graf, vil et utspennende tre være en delgraf T slik at
  - T har de samme nodene som G.

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.
- Hvis G er en sammenhengende graf, vil et utspennende tre være en delgraf T slik at
  - T har de samme nodene som G.
  - T er et tre, det vil si, T er sammenhengende og inneholder ingen sykler.

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.
- Hvis G er en sammenhengende graf, vil et utspennende tre være en delgraf T slik at
  - T har de samme nodene som G.
  - T er et tre, det vil si, T er sammenhengende og inneholder ingen sykler.
- Vi så forrige gang at hvis G har n noder, vil et utspennende tre ha $n - 1$ kanter.

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.
- Hvis G er en sammenhengende graf, vil et utspennende tre være en delgraf T slik at
  - T har de samme nodene som G.
  - T er et tre, det vil si, T er sammenhengende og inneholder ingen sykler.
- Vi så forrige gang at hvis G har n noder, vil et utspennende tre ha $n - 1$ kanter.
- En konsekvens er at hver gang vi velger ut $n - 1$ kanter fra G slik at vi ikke får noen sykler, så får vi et utspennende tre.

Prims algoritme

Prims algoritme

Eksempel (Fortsatt)

Prims algoritme

Eksempel (Fortsatt)

- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.

Prims algoritme

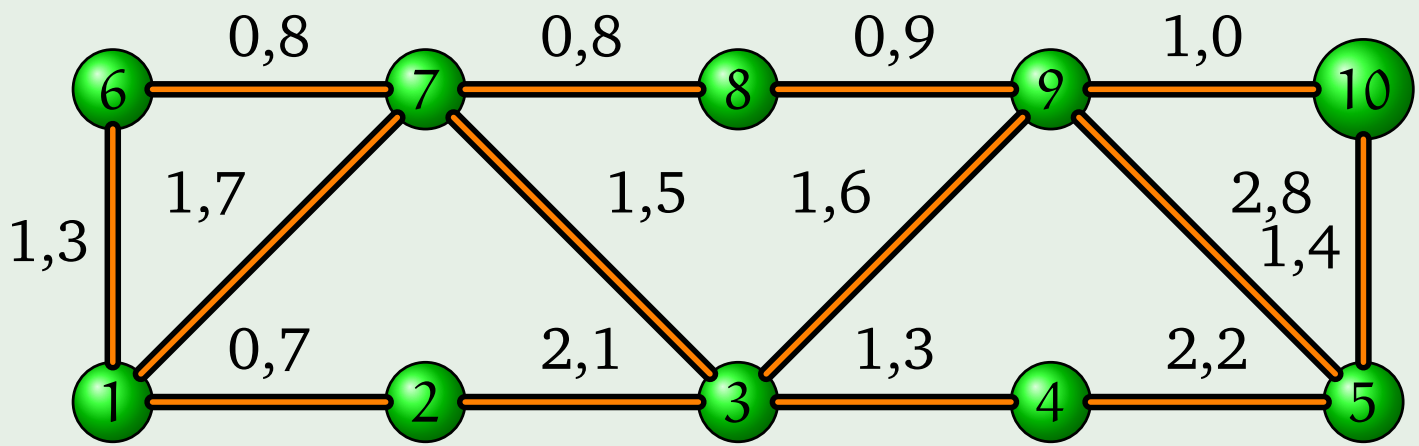
Eksempel (Fortsatt)

- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.

Prims algoritme

Eksempel (Fortsatt)

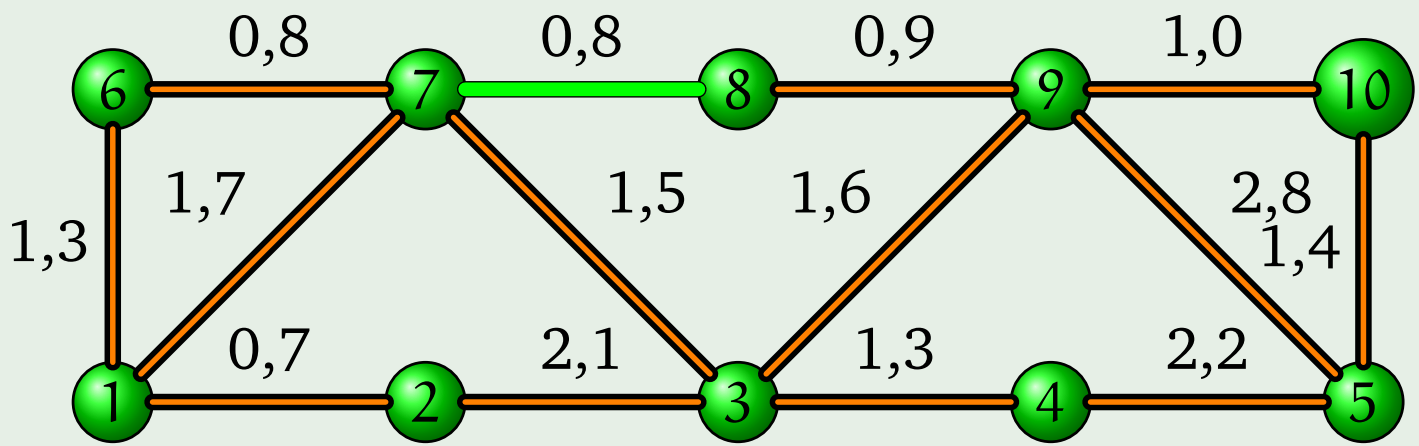
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

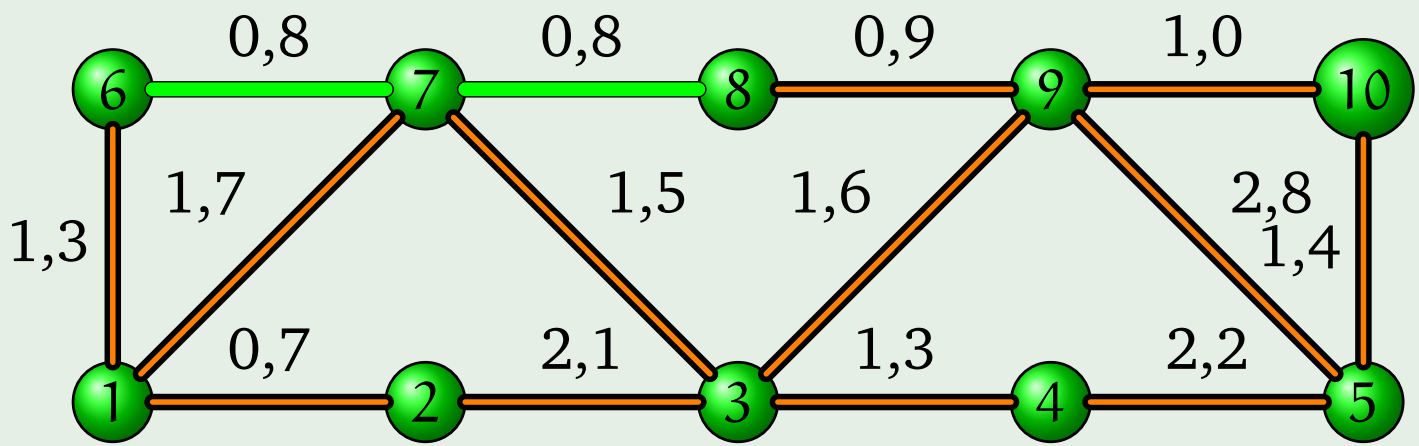
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

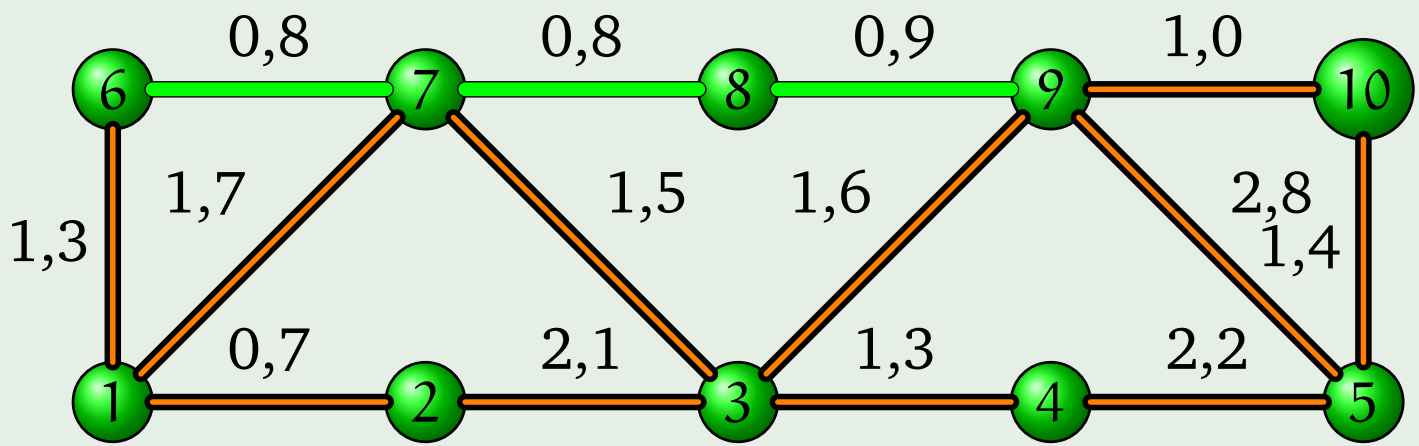
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

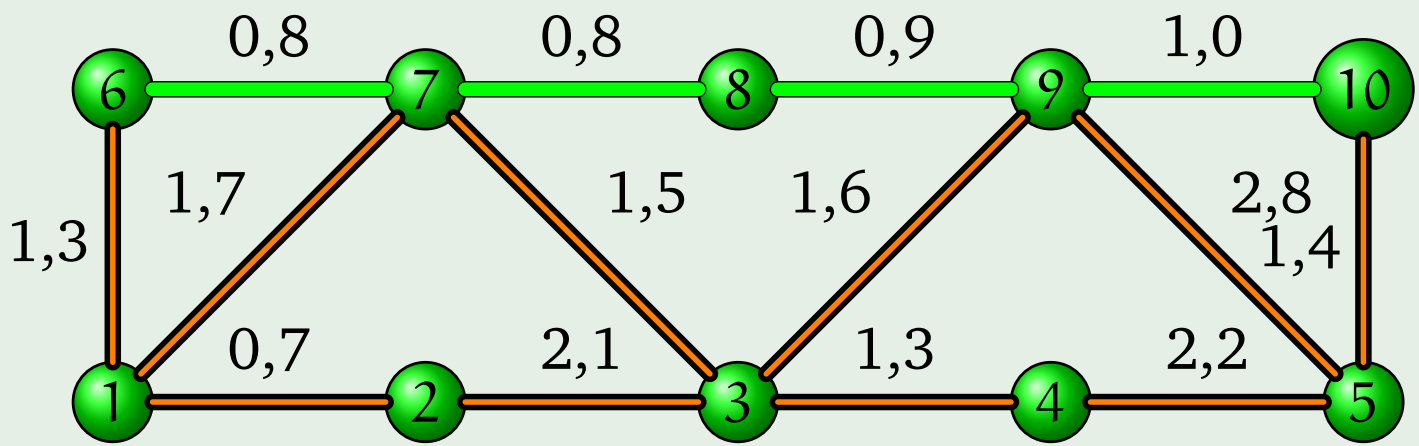
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

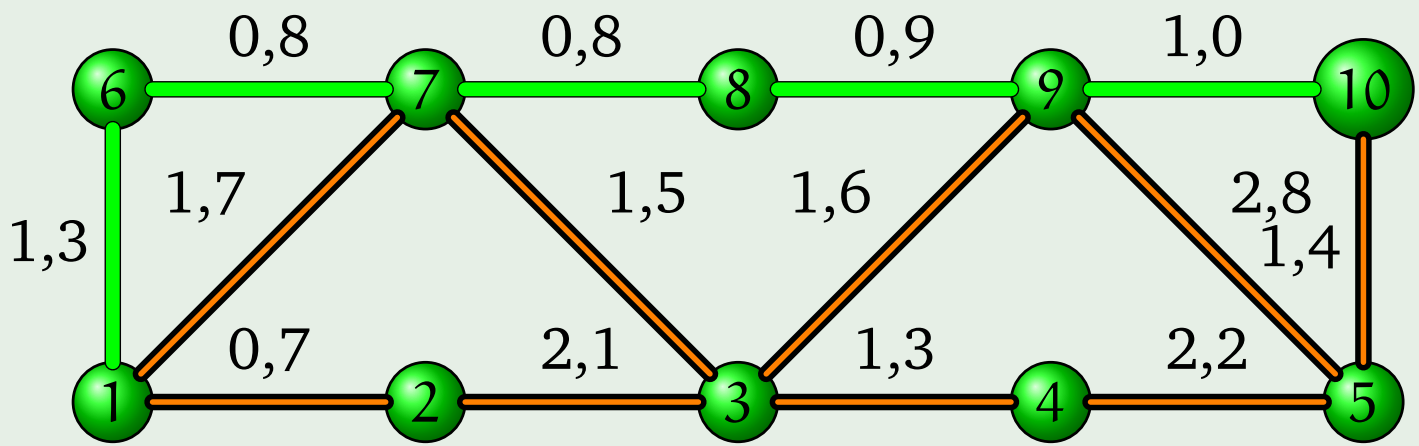
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

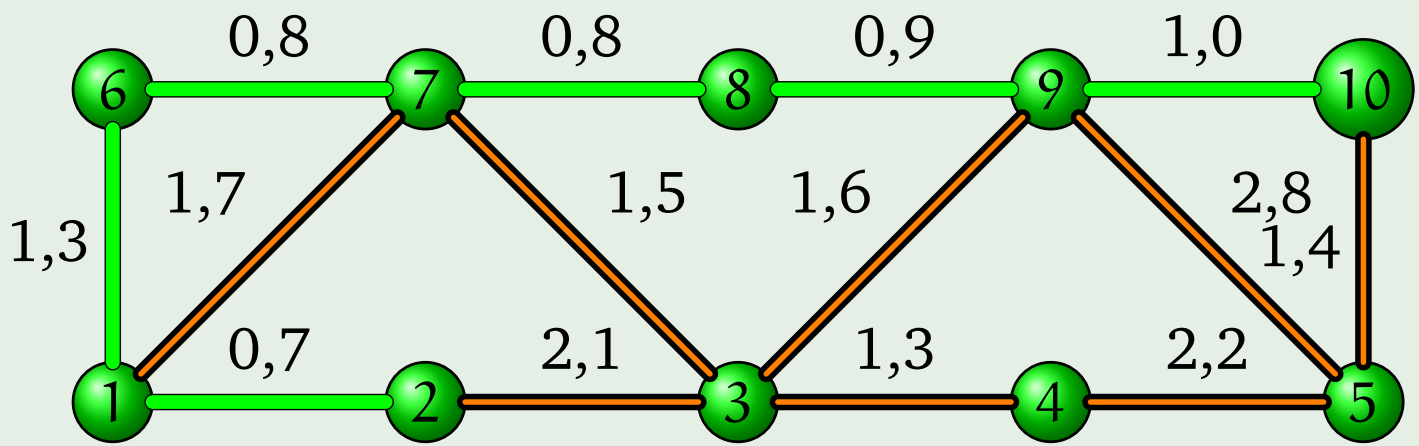
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

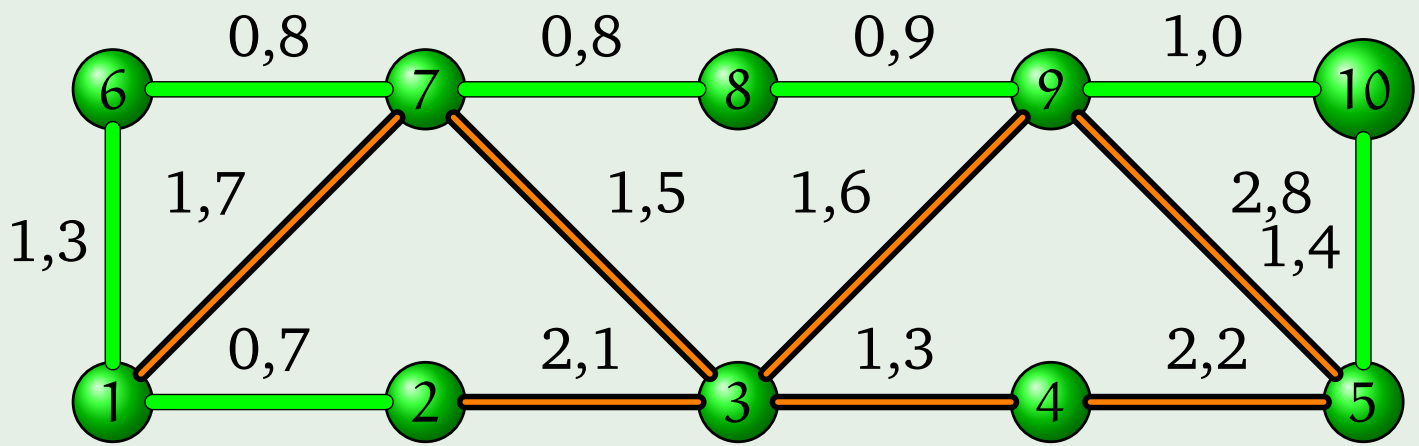
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

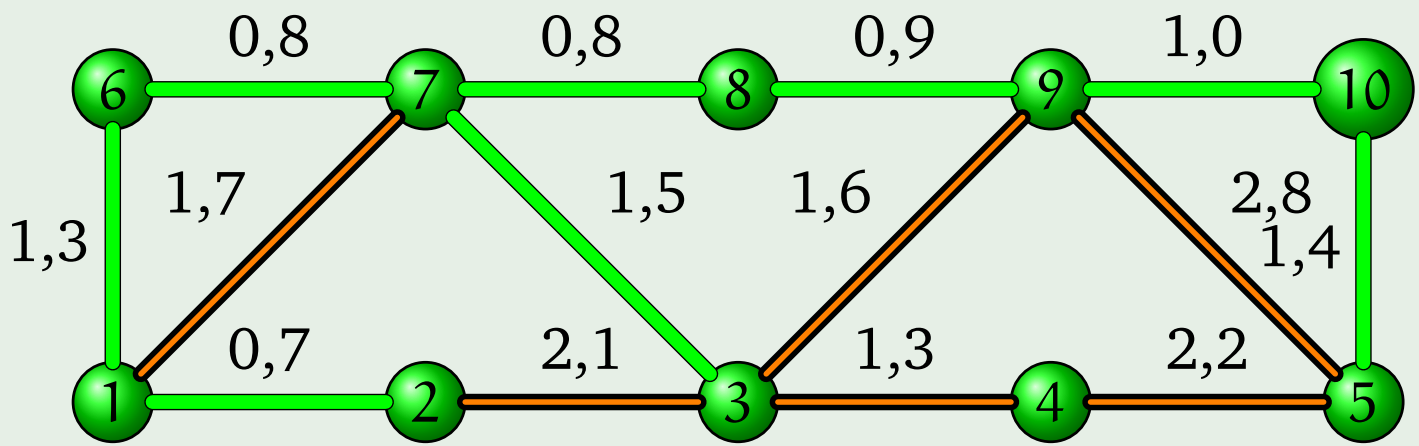
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

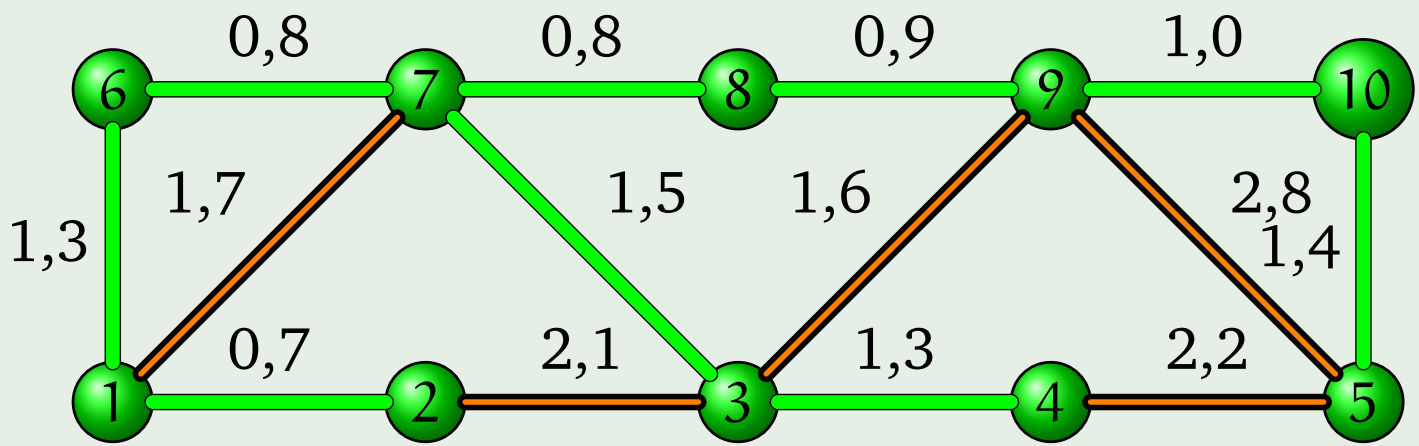
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

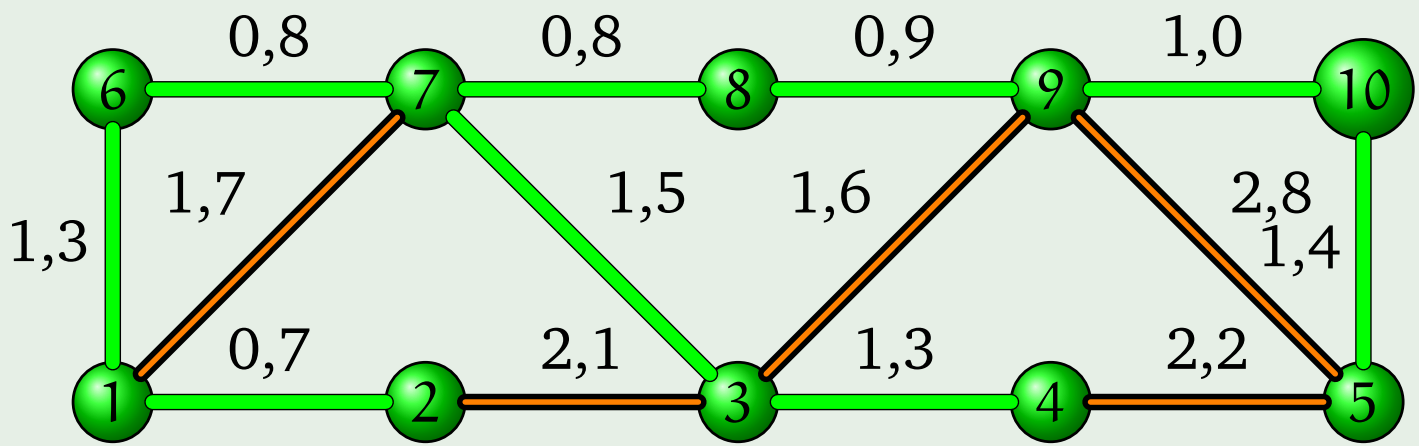
- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



Prims algoritme

Eksempel (Fortsatt)

- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.
- Hvis vi starter i Node 8 bygger vi opp treet slik.



- Det ble det samme treet til slutt.

Prims algoritme

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]
2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]
3 $T \leftarrow \{v_1\}$
4 $K \leftarrow \emptyset$

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]
2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]
3 $T \leftarrow \{v_1\}$
4 $K \leftarrow \emptyset$
5 **While** $E \neq \emptyset$ **do**

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

4 $K \leftarrow \emptyset$

5 **While** $E \neq \emptyset$ **do**

  5.1 $x \leftarrow$ *første* $e_i \in E$ *slik at* e_i *ligger inntil en node i* T *og har minimal vekt blant disse.*

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

4 $K \leftarrow \emptyset$

5 **While** $E \neq \emptyset$ **do**

  5.1 $x \leftarrow$ *første* $e_i \in E$ *slik at* e_i *ligger inntil en node i* T *og har minimal vekt blant disse.*

  5.2 $y \leftarrow$ *noden ved* x som ikke ligger i T

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

4 $K \leftarrow \emptyset$

5 **While** $E \neq \emptyset$ **do**

  5.1 $x \leftarrow$ *første* $e_i \in E$ *slik at* e_i *ligger inntil en node i* T *og har minimal vekt blant disse.*

  5.2 $y \leftarrow$ *noden ved* x som ikke ligger i T

  5.3 $K \leftarrow K \cup \{x\}$

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

4 $K \leftarrow \emptyset$

5 **While** $E \neq \emptyset$ **do**

  5.1 $x \leftarrow$ *første* $e_i \in E$ *slik at* e_i *ligger inntil en node i* T *og har minimal vekt blant disse.*

  5.2 $y \leftarrow$ *noden ved* x som ikke ligger i T

  5.3 $K \leftarrow K \cup \{x\}$

  5.4 $T \leftarrow T \cup \{y\}$

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

4 $K \leftarrow \emptyset$

5 **While** $E \neq \emptyset$ **do**

  5.1 $x \leftarrow$ *første* $e_i \in E$ *slik at* e_i *ligger inntil en node i* T *og har minimal vekt blant disse.*

  5.2 $y \leftarrow$ *noden ved* x som ikke ligger i T

  5.3 $K \leftarrow K \cup \{x\}$

  5.4 $T \leftarrow T \cup \{y\}$

  5.5 $E \leftarrow E - \{e_i \ ; \ e_i$ *ligger inntil to noder i* $T\}$.

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

3 $T \leftarrow \{v_1\}$

4 $K \leftarrow \emptyset$

5 **While** $E \neq \emptyset$ **do**

  5.1 $x \leftarrow$ *første* $e_i \in E$ *slik at* e_i *ligger inntil en node i* T *og har minimal vekt blant disse.*

  5.2 $y \leftarrow$ *noden ved* x som ikke ligger i T

  5.3 $K \leftarrow K \cup \{x\}$

  5.4 $T \leftarrow T \cup \{y\}$

  5.5 $E \leftarrow E - \{e_i \ ; \ e_i$ *ligger inntil to noder i* $T\}$.

6 *Output* (T, K).

Prims algoritme

Prims algoritme

- Det fins en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.

Prims algoritme

- Det fins en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.

Prims algoritme

- Det fins en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.
- Vi utnytter at vi får et utspennende tre bare vi tar med $n - 1$ kanter uten å lage noen sykler.

Prims algoritme

- Det fins en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.
- Vi utnytter at vi får et utspennende tre bare vi tar med $n - 1$ kanter uten å lage noen sykler.
- Hver gang legger vi til en ny kant med minimal vekt slik at vi ikke får noen sykler.

Prims algoritme

- Det fins en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.
- Vi utnytter at vi får et utspennende tre bare vi tar med $n - 1$ kanter uten å lage noen sykler.
- Hver gang legger vi til en ny kant med minimal vekt slik at vi ikke får noen sykler.
- Fins det flere aktuelle kanter med samme minimale vekt, velger vi den som står først i listen vår.

Prims algoritme

- Det fins en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.
- Vi utnytter at vi får et utspennende tre bare vi tar med $n - 1$ kanter uten å lage noen sykler.
- Hver gang legger vi til en ny kant med minimal vekt slik at vi ikke får noen sykler.
- Fins det flere aktuelle kanter med samme minimale vekt, velger vi den som står først i listen vår.
- I eksemplet vårt vil da det utspennende treet bygge seg opp slik som på neste side.

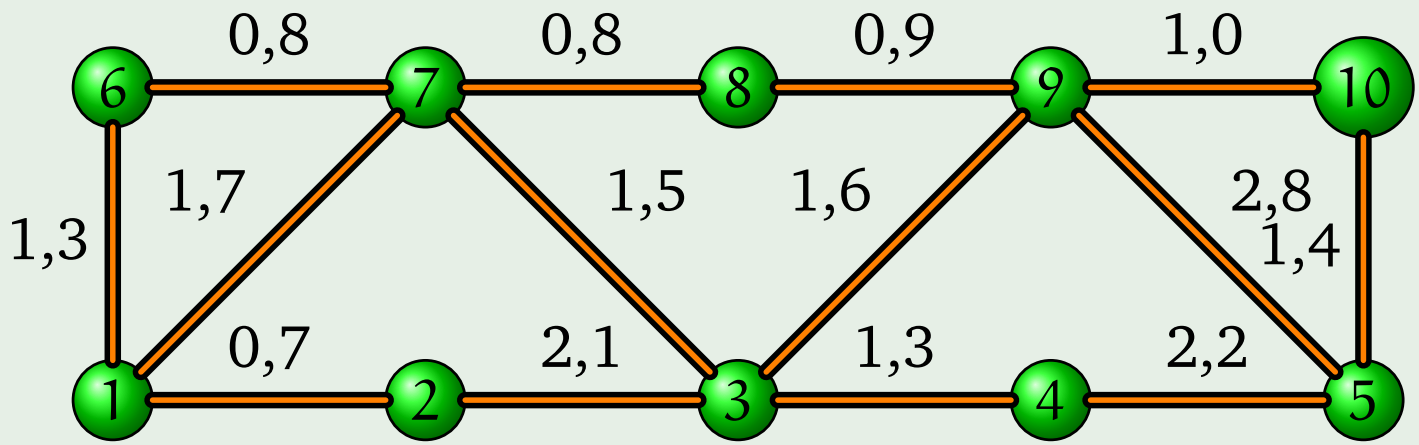
Prims algoritme

Prims algoritme

Eksempel (Fortsatt)

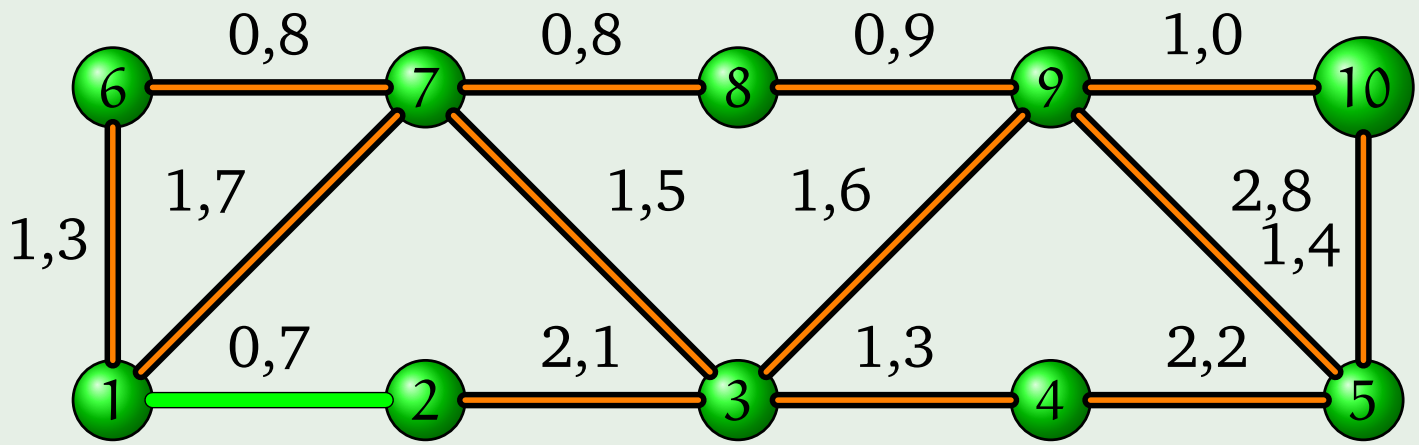
Prims algoritme

Eksempel (Fortsatt)



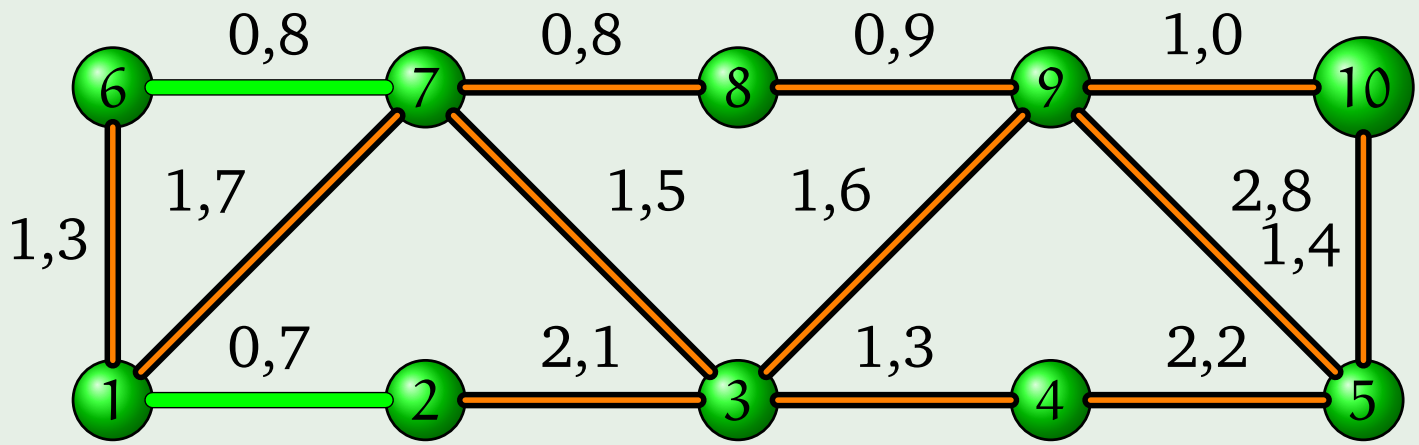
Prims algoritme

Eksempel (Fortsatt)



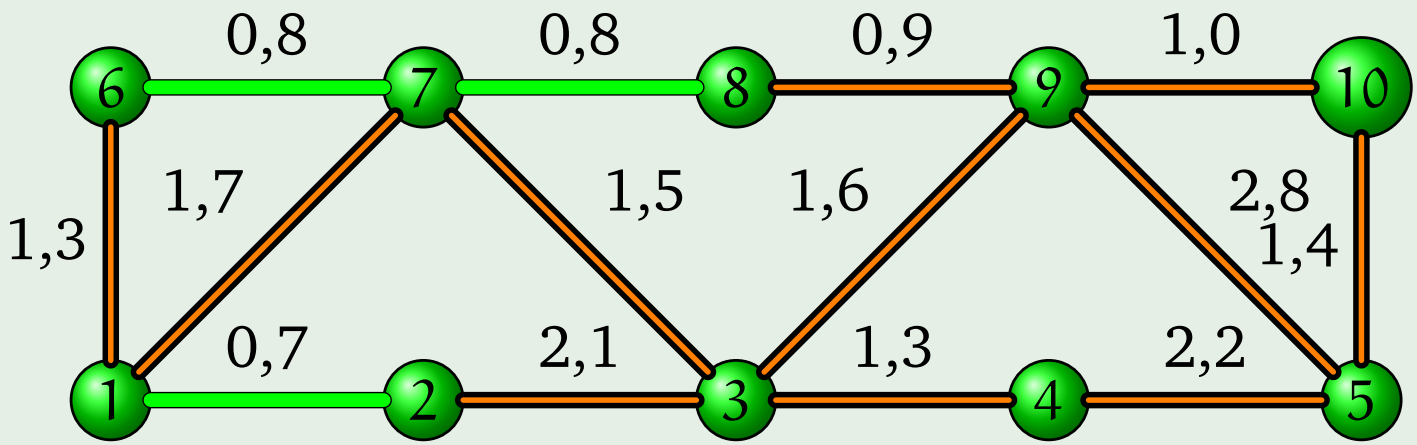
Prims algoritme

Eksempel (Fortsatt)



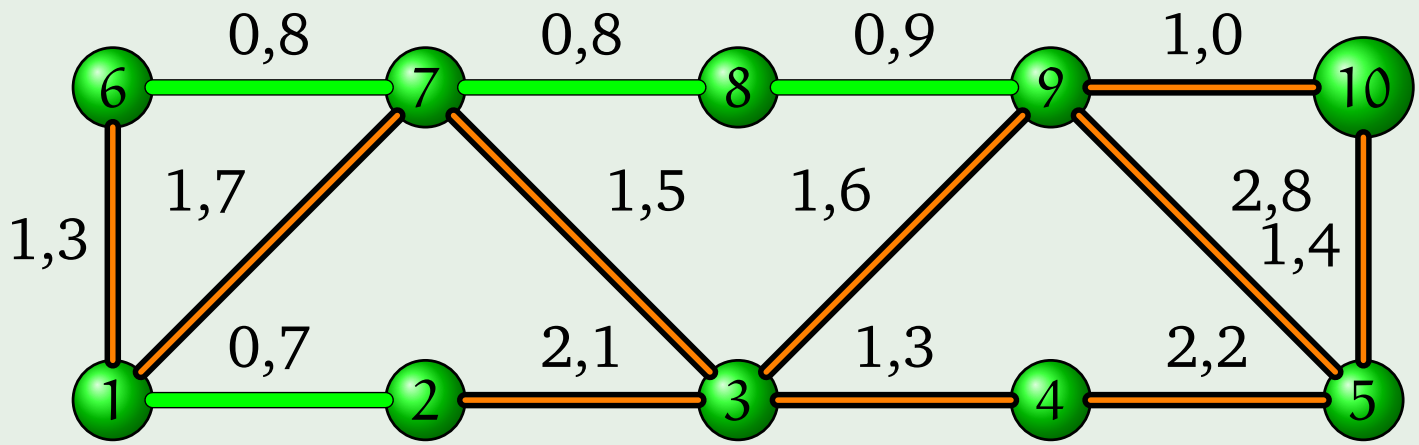
Prims algoritme

Eksempel (Fortsatt)



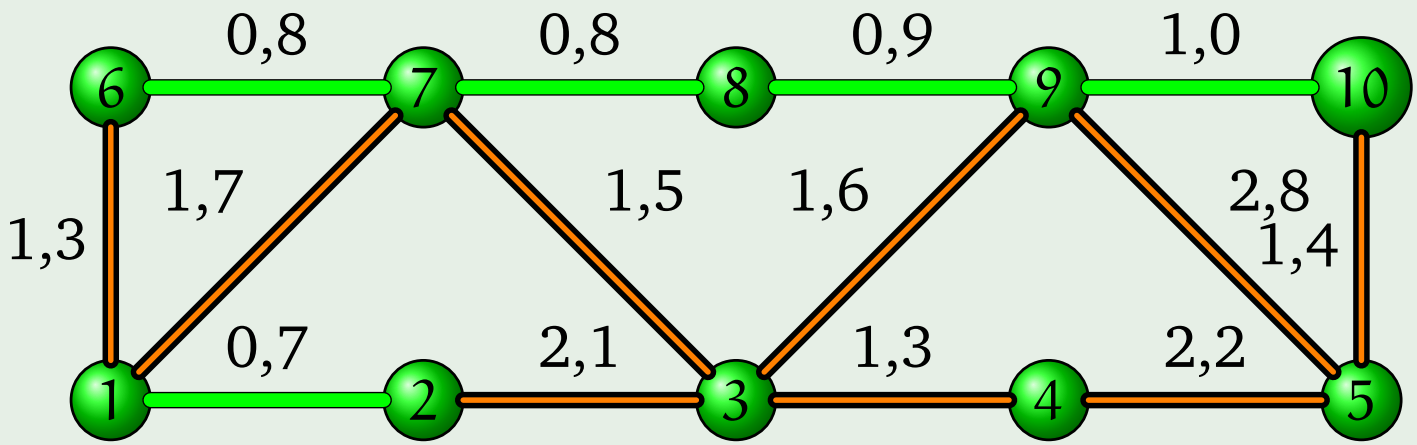
Prims algoritme

Eksempel (Fortsatt)



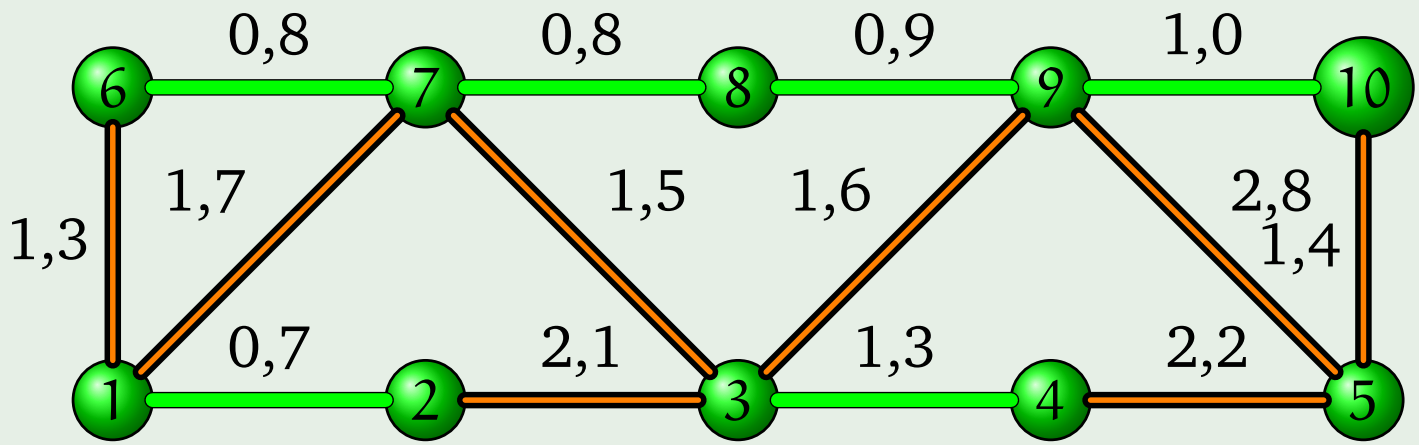
Prims algoritme

Eksempel (Fortsatt)



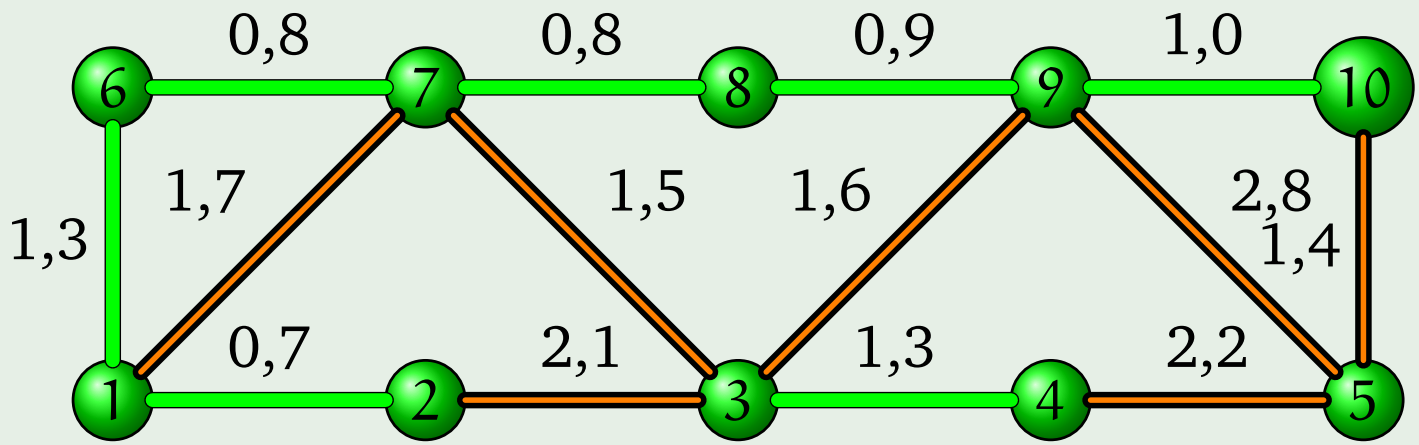
Prims algoritme

Eksempel (Fortsatt)



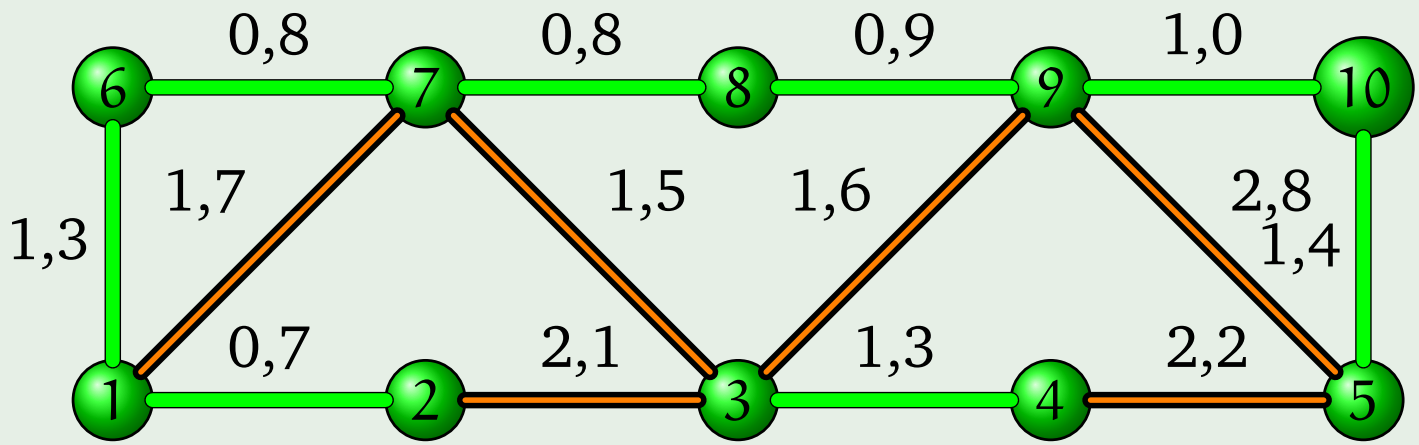
Prims algoritme

Eksempel (Fortsatt)



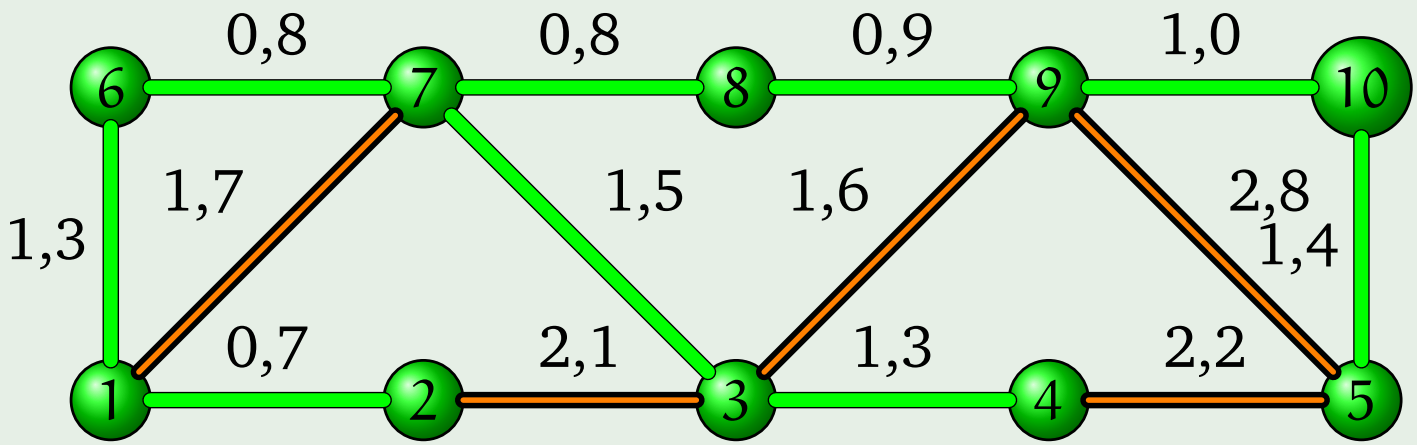
Prims algoritme

Eksempel (Fortsatt)



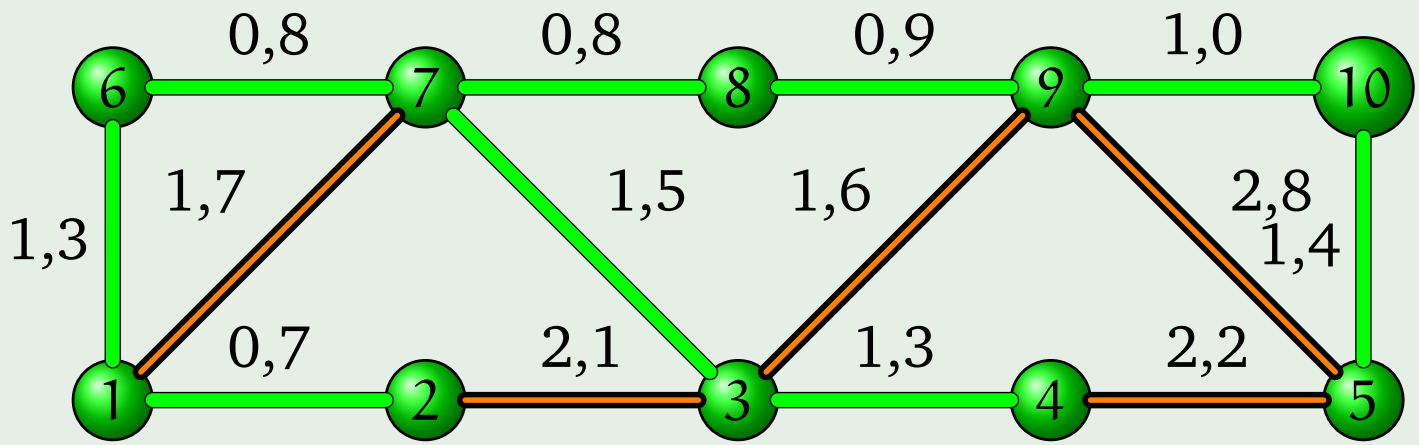
Prims algoritme

Eksempel (Fortsatt)



Prims algoritme

Eksempel (Fortsatt)



- Vi får fortsatt det samme treet

Dijkstras algoritme

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt “sentralnode”.

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt "sentralnode".
- Her tenker vi oss at vektene svarer til lengder av de enkelte kantene.

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt "sentralnode".
- Her tenker vi oss at vektene svarer til lengder av de enkelte kantene.
- Det fins effektive algoritmer for dette også.

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt "sentralnode".
- Her tenker vi oss at vektene svarer til lengder av de enkelte kantene.
- Det fins effektive algoritmer for dette også.
- Vi skal gi en uformell beskrivelse av Dijkstras algoritme og vise hvordan den virker på eksemplet vårt.

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt "sentralnode".
- Her tenker vi oss at vektene svarer til lengder av de enkelte kantene.
- Det fins effektive algoritmer for dette også.
- Vi skal gi en uformell beskrivelse av Dijkstras algoritme og vise hvordan den virker på eksemplet vårt.
- Vi skal se at vi denne gangen får et annet tre.

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt "sentralnode".
- Her tenker vi oss at vektene svarer til lengder av de enkelte kantene.
- Det fins effektive algoritmer for dette også.
- Vi skal gi en uformell beskrivelse av Dijkstras algoritme og vise hvordan den virker på eksemplet vårt.
- Vi skal se at vi denne gangen får et annet tre.
- For dette problemet er også treet vi får avhengig av hvilken node som velges som "sentrum".

Dijkstras algoritme

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.
- Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.
- Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.
- Ved rekursjon på $i \leqslant n$ konstruerer vi et tre T_i med i noder og $i-1$ kanter fra G.

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.
- Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.
- Ved rekursjon på $i \leqslant n$ konstruerer vi et tre T_i med i noder og $i-1$ kanter fra G.
- Vi konstruerer T_{i+1} fra T_i når $i < n$ ved følgende prosedyre:

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.
- Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.
- Ved rekursjon på $i \leqslant n$ konstruerer vi et tre T_i med i noder og $i-1$ kanter fra G.
- Vi konstruerer T_{i+1} fra T_i når $i < n$ ved følgende prosedyre:
  1. Finn den noden v utenom T_i og den kanten e som er slik at e forbinder v til T_i, og vi oppnår den kortest mulige veien fra v_1 til v, via T_i og e, på denne måten.

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.
- Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.
- Ved rekursjon på $i \leqslant n$ konstruerer vi et tre T_i med i noder og $i-1$ kanter fra G.
- Vi konstruerer T_{i+1} fra T_i når $i < n$ ved følgende prosedyre:
  1. Finn den noden v utenom T_i og den kanten e som er slik at e forbinder v til T_i, og vi oppnår den kortest mulige veien fra v_1 til v, via T_i og e, på denne måten.
  2. Fins det flere like gode alternativer, velg den v'en og deretter den e'en som står først i listene.

Dijkstras algoritme

- Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.
- Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.
- Ved rekursjon på $i \leqslant n$ konstruerer vi et tre T_i med i noder og $i-1$ kanter fra G.
- Vi konstruerer T_{i+1} fra T_i når $i < n$ ved følgende prosedyre:
  1. Finn den noden v utenom T_i og den kanten e som er slik at e forbinder v til T_i, og vi oppnår den kortest mulige veien fra v_1 til v, via T_i og e, på denne måten.
  2. Fins det flere like gode alternativer, velg den v'en og deretter den e'en som står først i listene.
  3. Utvid T_i til T_{i+1} ved å legge til noden v og kanten e.

Dijkstras algoritme

Dijkstras algoritme

- Dijkstras algoritme er beskrevet i form av en pseudokode i boka.

Dijkstras algoritme

- Dijkstras algoritme er beskrevet i form av en pseudokode i boka.
- Det er meningen at dere skal kunne finne det utspennende treet som gir de korteste stiene fra provinsen til sentrum når dere har fått gitt en vektet, sammenhengende graf.

Dijkstras algoritme

- Dijkstras algoritme er beskrevet i form av en pseudokode i boka.
- Det er meningen at dere skal kunne finne det utspennende treet som gir de korteste stiene fra provinsen til sentrum når dere har fått gitt en vektet, sammenhengende graf.
- La oss se på eksemplet vårt en gang til.

Dijkstras algoritme

Dijkstras algoritme

Eksempel

Dijkstras algoritme

Eksempel

- Vi starter i Node 1

Dijkstras algoritme

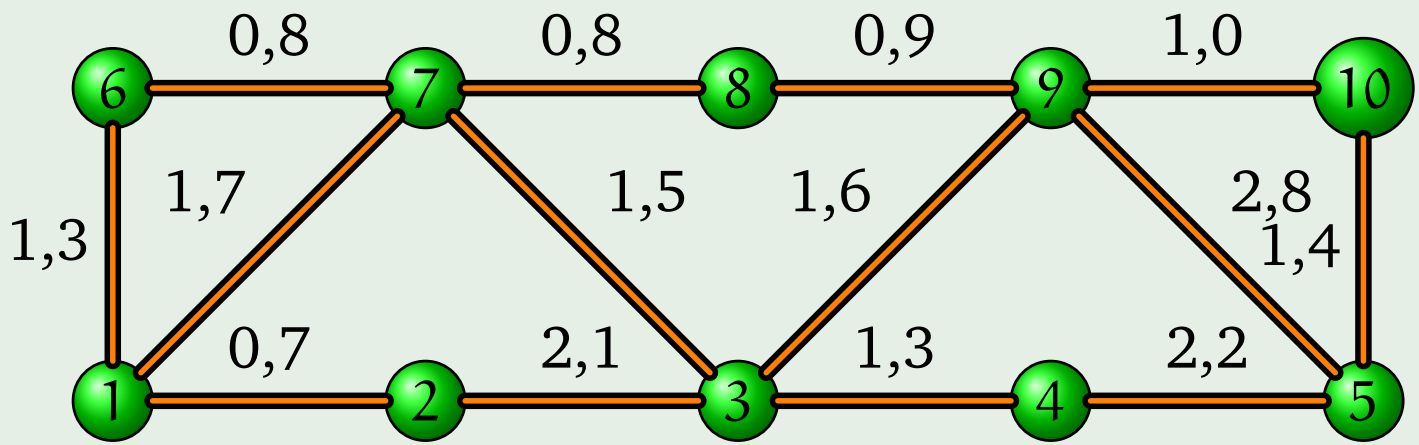
Eksempel

- Vi starter i Node 1
- Vi får et nytt tre.

Dijkstras algoritme

Eksempel

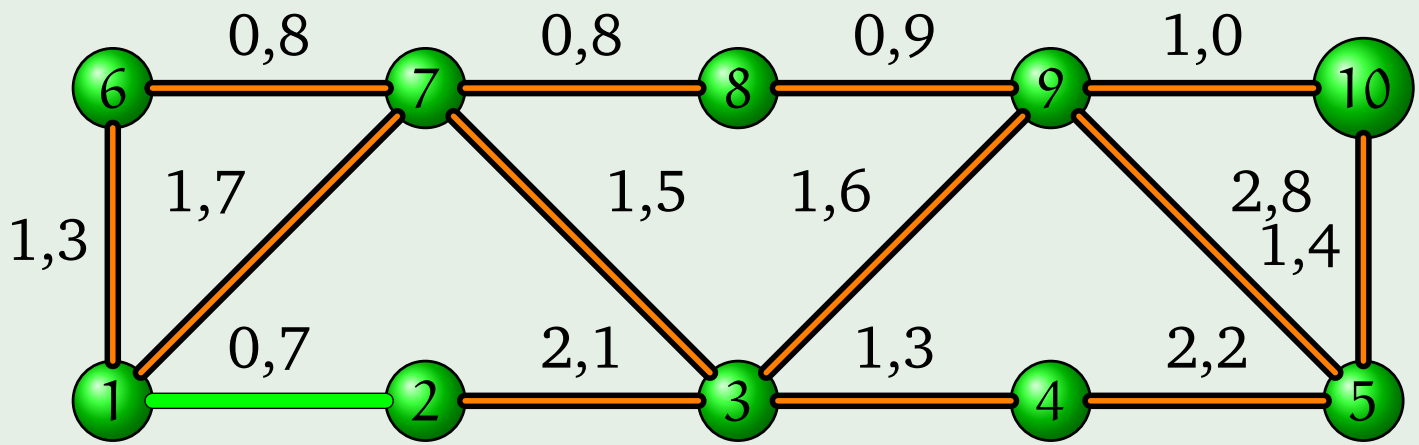
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

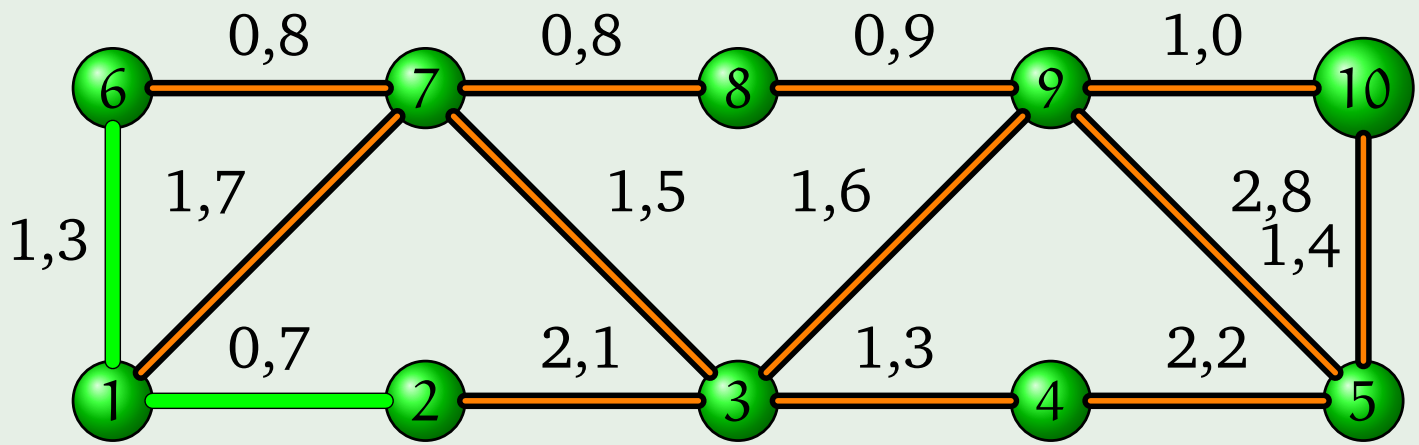
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

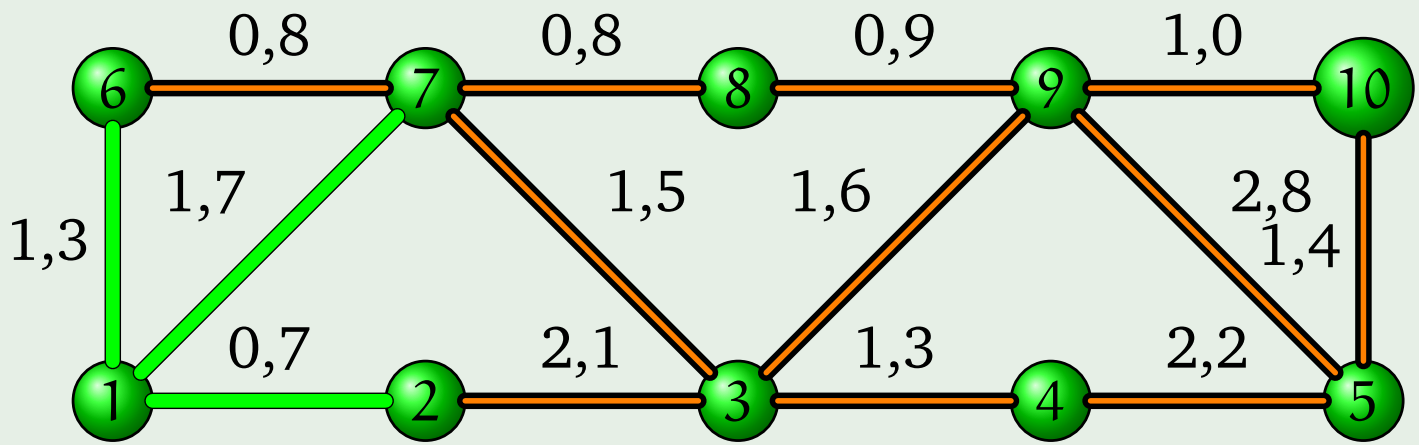
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

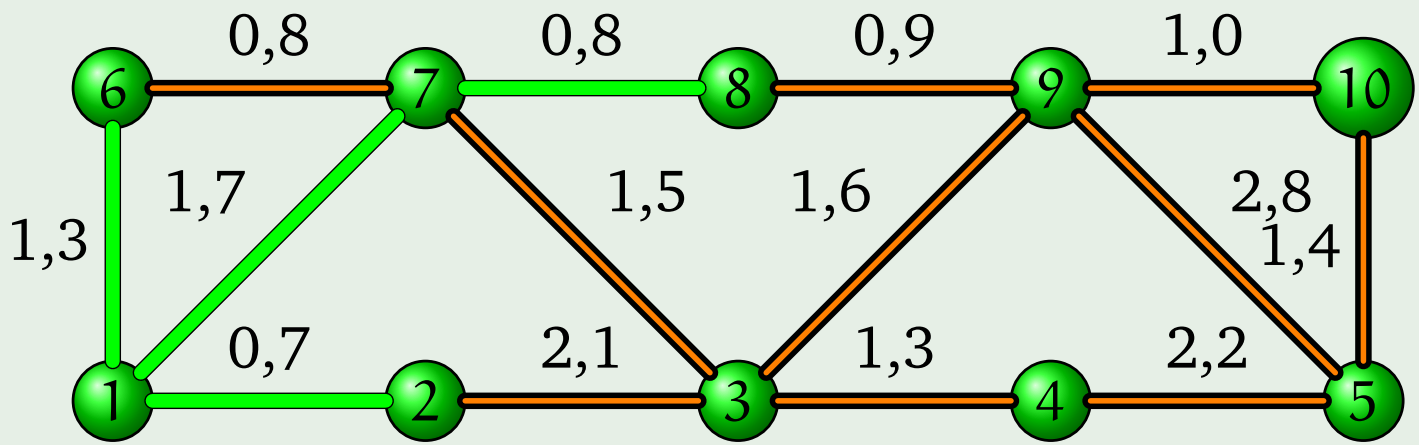
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

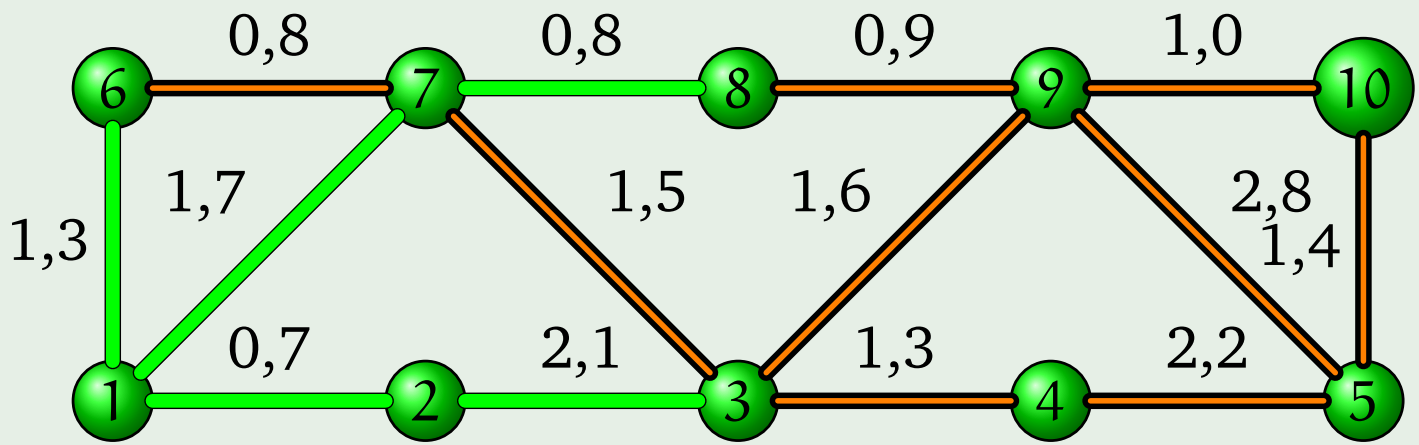
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

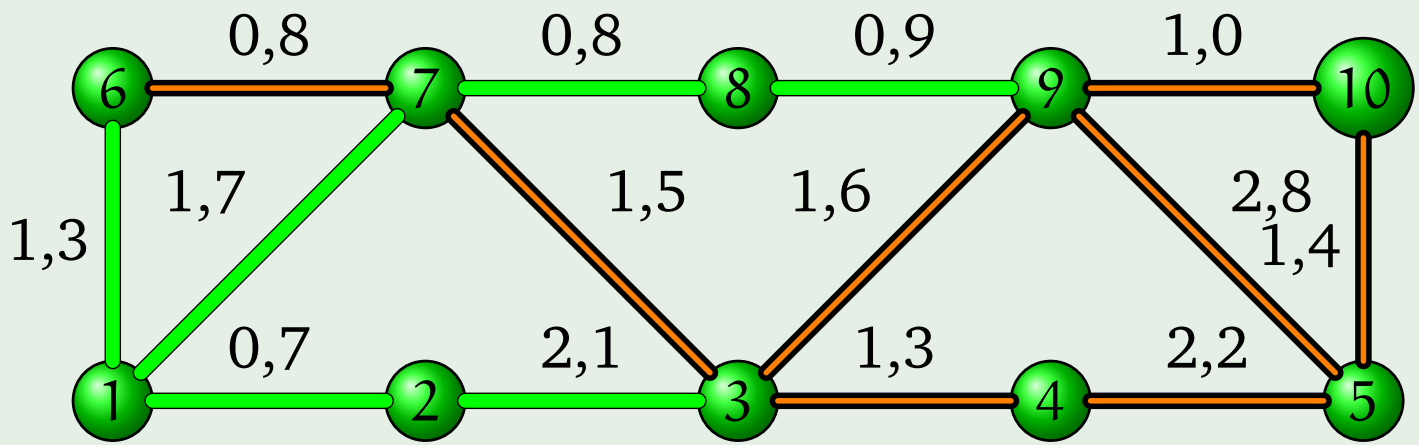
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

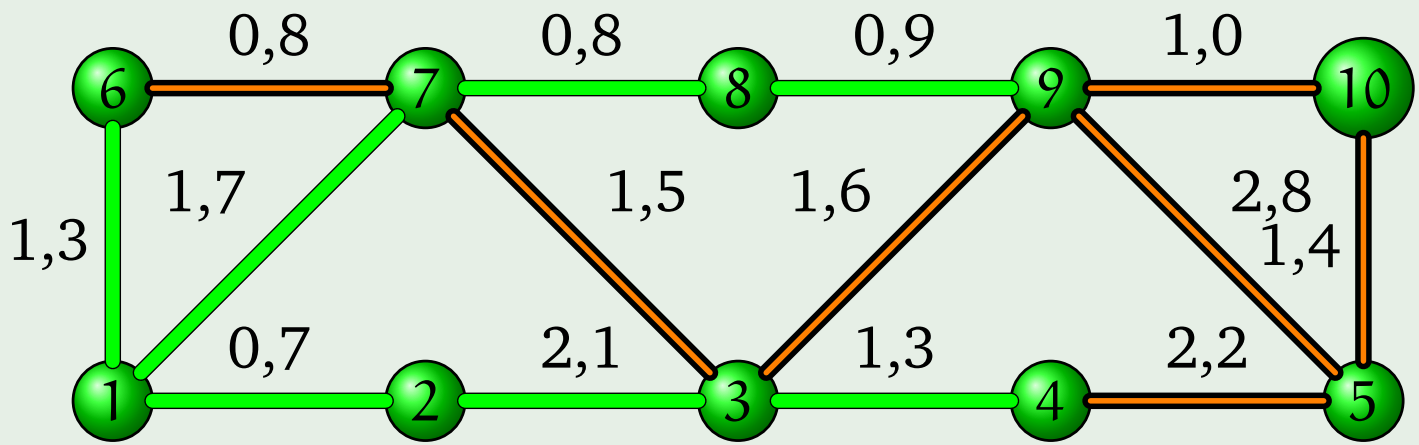
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

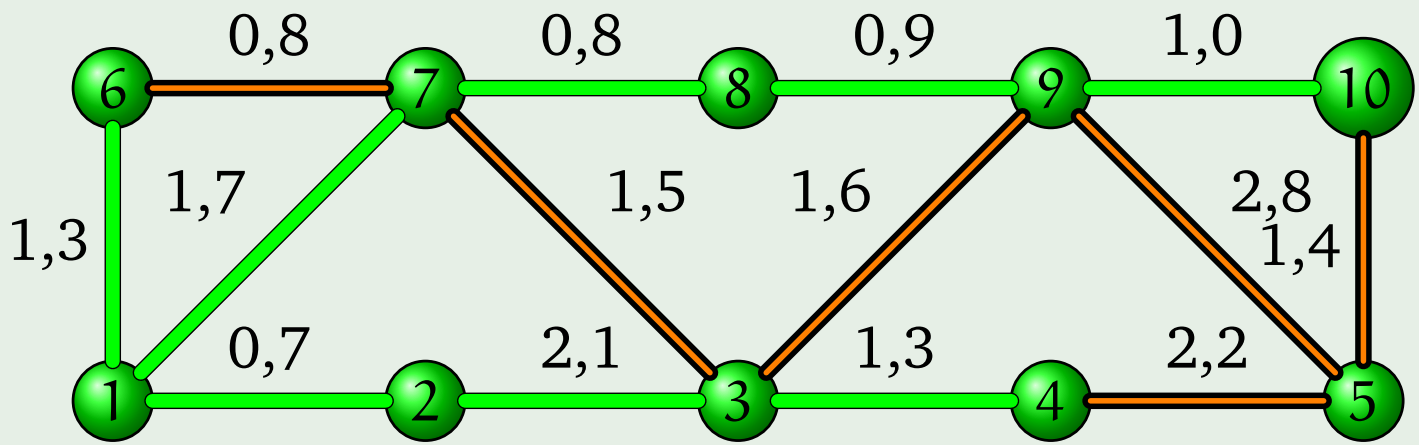
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

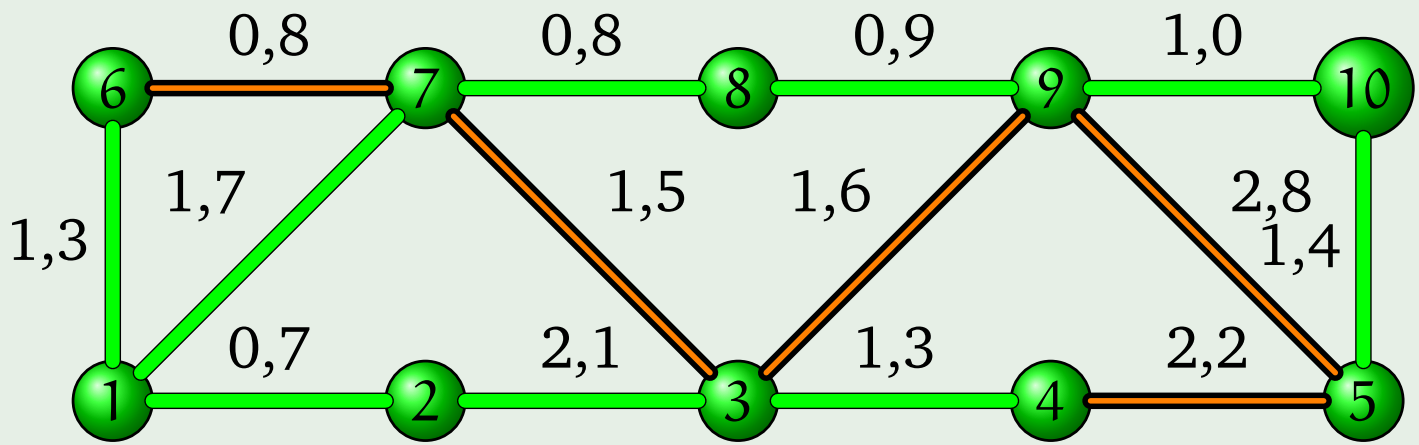
- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

Eksempel

- Vi starter i Node 1
- Vi får et nytt tre.



Representasjoner

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.
- Vi har tidligere sett hvordan grafer kan representeres som matriser.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.
- Vi har tidligere sett hvordan grafer kan representeres som matriser.
- Siden matriser kan representeres digitalt, betyr det at også grafer kan representeres digitalt.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.
- Vi har tidligere sett hvordan grafer kan representeres som matriser.
- Siden matriser kan representeres digitalt, betyr det at også grafer kan representeres digitalt.
- Vektede grafer kan også representeres som matriser.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.
- Vi har tidligere sett hvordan grafer kan representeres som matriser.
- Siden matriser kan representeres digitalt, betyr det at også grafer kan representeres digitalt.
- Vektede grafer kan også representeres som matriser.
- Vi følger boka, og lar ∞ representere at det ikke fins noen kant mellom to noder.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.
- Vi har tidligere sett hvordan grafer kan representeres som matriser.
- Siden matriser kan representeres digitalt, betyr det at også grafer kan representeres digitalt.
- Vektede grafer kan også representeres som matriser.
- Vi følger boka, og lar ∞ representere at det ikke fins noen kant mellom to noder.
- Hvis vi tolker grafen som en strømkrets hvor vektene representerer motstanden i hver enkelt ledning, vil det at vi ikke har noen direkte kobling mellom to noder svare til at vi har en ledning med uendelig motstand mellom dem.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektede grafene digitalt.
- Vi har tidligere sett hvordan grafer kan representeres som matriser.
- Siden matriser kan representeres digitalt, betyr det at også grafer kan representeres digitalt.
- Vektede grafer kan også representeres som matriser.
- Vi følger boka, og lar ∞ representere at det ikke fins noen kant mellom to noder.
- Hvis vi tolker grafen som en strømkrets hvor vektene representerer motstanden i hver enkelt ledning, vil det at vi ikke har noen direkte kobling mellom to noder svare til at vi har en ledning med uendelig motstand mellom dem.
- Vi illustrerer dette med et eksempel.

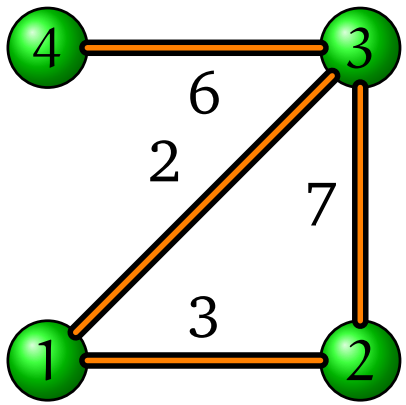
Representasjoner

Representasjoner

En vektet graf.

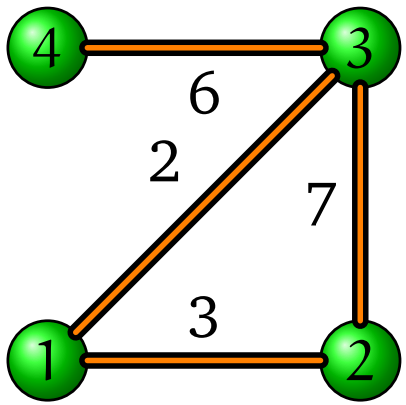
Representasjoner

En vektet graf.



Representasjoner

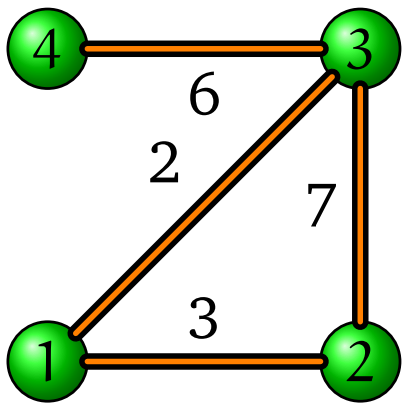
En vektet graf.



Matriserepresentasjonen.

Representasjoner

En vektet graf.



Matriserepresentasjonen.

	1	2	3	4
1	0	3	2	∞
2	3	0	7	∞
3	2	7	0	6
4	∞	∞	6	0

Trær med rot

Trær med rot

- Til nå har vi sett på trær som sammenhengende grafer uten løkker.

Trær med rot

- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.

Trær med rot

- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.
- For mange anvendelser av teorien for trær, er det nyttig å kunne betrakte den ene noden som roten til treet, den noden alt annet har vokst ut fra.

Trær med rot

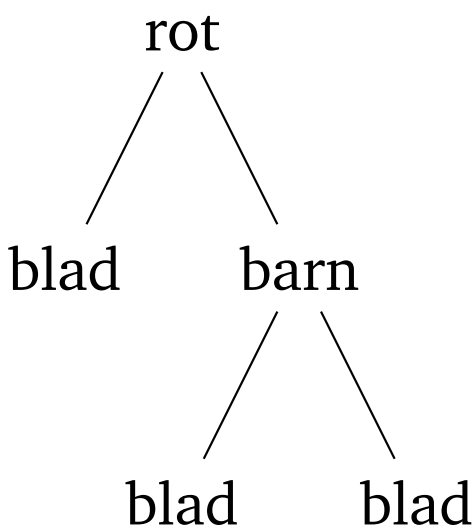
- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.
- For mange anvendelser av teorien for trær, er det nyttig å kunne betrakte den ene noden som roten til treet, den noden alt annet har vokst ut fra.
- Formelt sett er et tre med rot definert som et grafteoretisk tre hvor en av nodene er utpekt som rot.

Trær med rot

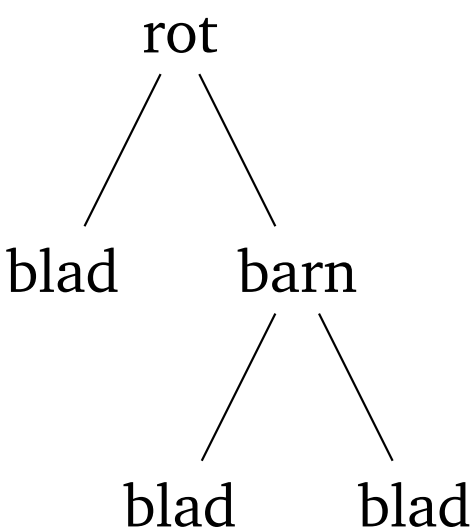
- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.
- For mange anvendelser av teorien for trær, er det nyttig å kunne betrakte den ene noden som roten til treet, den noden alt annet har vokst ut fra.
- Formelt sett er et tre med rot definert som et grafteoretisk tre hvor en av nodene er utpekt som rot.
- Det er imidlertid vanlig å tegne slike trær litt annerledes enn trær uten rot.

Trær med rot

Trær med rot

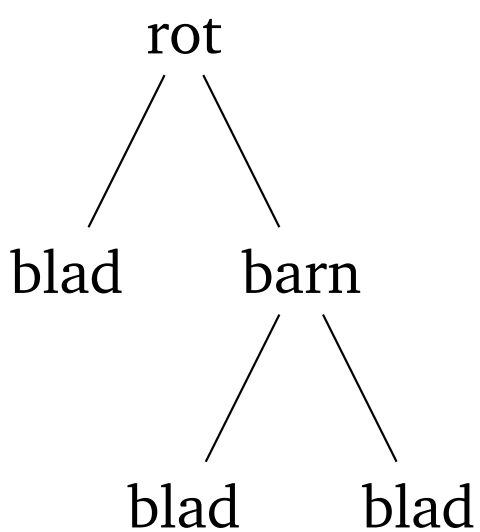


Trær med rot



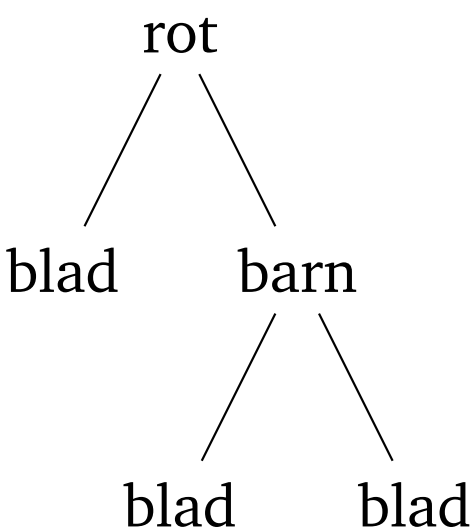
- Roten tegnes øverst, og treet "vokser" nedover.

Trær med rot



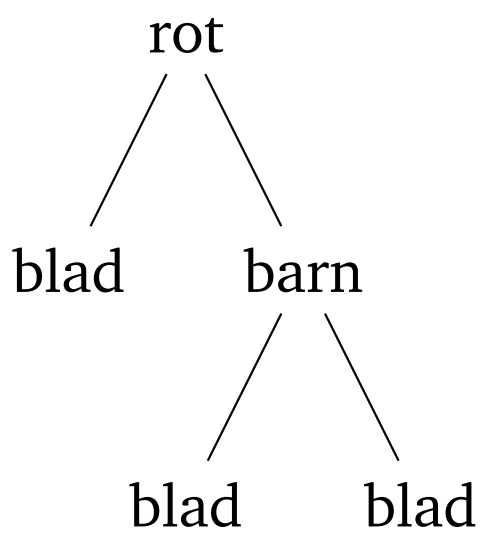
- Roten tegnes øverst, og treet "vokser" nedover.
- Nodene ligger i "lag" avhengig av avstanden til roten.

Trær med rot



- Roten tegnes øverst, og treet "vokser" nedover.
- Nodene ligger i "lag" avhengig av avstanden til roten.
- Vi kan snakke om barna til en node, og om bladene til et tre med rot.

Trær med rot



- Roten tegnes øverst, og treet "vokser" nedover.
- Nodene ligger i "lag" avhengig av avstanden til roten.
- Vi kan snakke om barna til en node, og om bladene til et tre med rot.

Vi skal se på endel eksempler før vi går nærmere inn på terminologien.

Trær med rot

Trær med rot

- Vi kan bruke trær til å gi en grafisk fremstilling av en utsagnslogisk formel.

Trær med rot

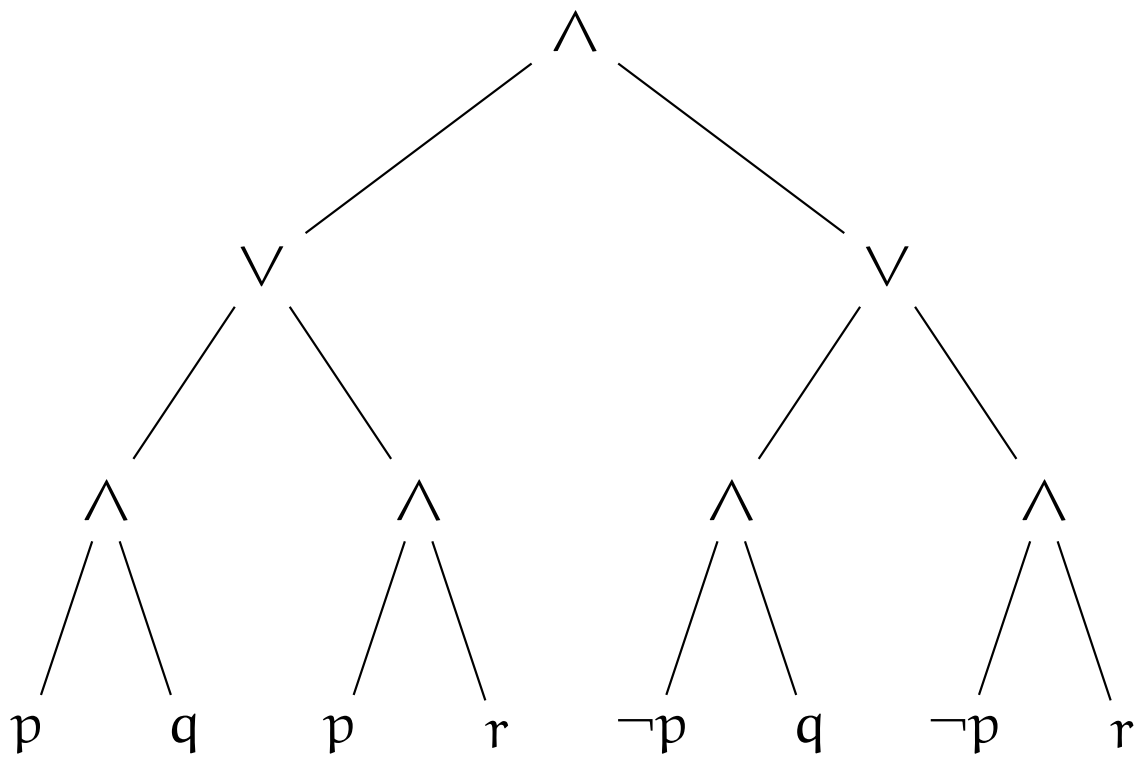
- Vi kan bruke trær til å gi en grafisk fremstilling av en utsagnslogisk formel.
- La $A = ((p \wedge q) \vee (p \wedge r)) \wedge ((\neg p \wedge q) \vee (\neg p \wedge r))$.

Trær med rot

- Vi kan bruke trær til å gi en grafisk fremstilling av en utsagnslogisk formel.
- La $A = ((p \wedge q) \vee (p \wedge r)) \wedge ((\neg p \wedge q) \vee (\neg p \wedge r))$.
- Mengden av formler er bygget opp induktivt, og oppbyggingen av hver formel kan beskrives som et tre:

Trær med rot

- Vi kan bruke trær til å gi en grafisk fremstilling av en utsagnslogisk formel.
- La $A = ((p \wedge q) \vee (p \wedge r)) \wedge ((\neg p \wedge q) \vee (\neg p \wedge r))$.
- Mengden av formler er bygget opp induktivt, og oppbyggingen av hver formel kan beskrives som et tre:



Trær med rot

Trær med rot

- Et tre som det på forrige side kaller vi ofte et syntakstre.

Trær med rot

- Et tre som det på forrige side kaller vi ofte et syntakstre.
- Et syntakstre forteller oss hvordan et ord i et litt komplisert formelt språk er bygget opp av enklere ord.

Trær med rot

- Et tre som det på forrige side kaller vi ofte et syntakstre.
- Et syntakstre forteller oss hvordan et ord i et litt komplisert formelt språk er bygget opp av enklere ord.
- Syntakstrær kan brukes i grammatikkanalyse av setninger i naturlige språk.

Trær med rot

- Et tre som det på forrige side kaller vi ofte et syntakstre.
- Et syntakstre forteller oss hvordan et ord i et litt komplisert formelt språk er bygget opp av enklere ord.
- Syntakstrær kan brukes i grammatikkanalyse av setninger i naturlige språk.
- Da setter man opp et tre som beskriver hvordan en setning f.eks. er bygget opp av subjekt og predikat, hvordan subjektet kan bestå av en artikkel og et substantiv, osv.

Trær med rot

- Et tre som det på forrige side kaller vi ofte et syntakstre.
- Et syntakstre forteller oss hvordan et ord i et litt komplisert formelt språk er bygget opp av enklere ord.
- Syntakstrær kan brukes i grammatikkanalyse av setninger i naturlige språk.
- Da setter man opp et tre som beskriver hvordan en setning f.eks. er bygget opp av subjekt og predikat, hvordan subjektet kan bestå av en artikkel og et substantiv, osv.
- Som et eksempel skal vi se et slikt tre på neste side, uten at bruken av slike trær skal være noe tema for disse forelesningene.

Trær med rot

Trær med rot

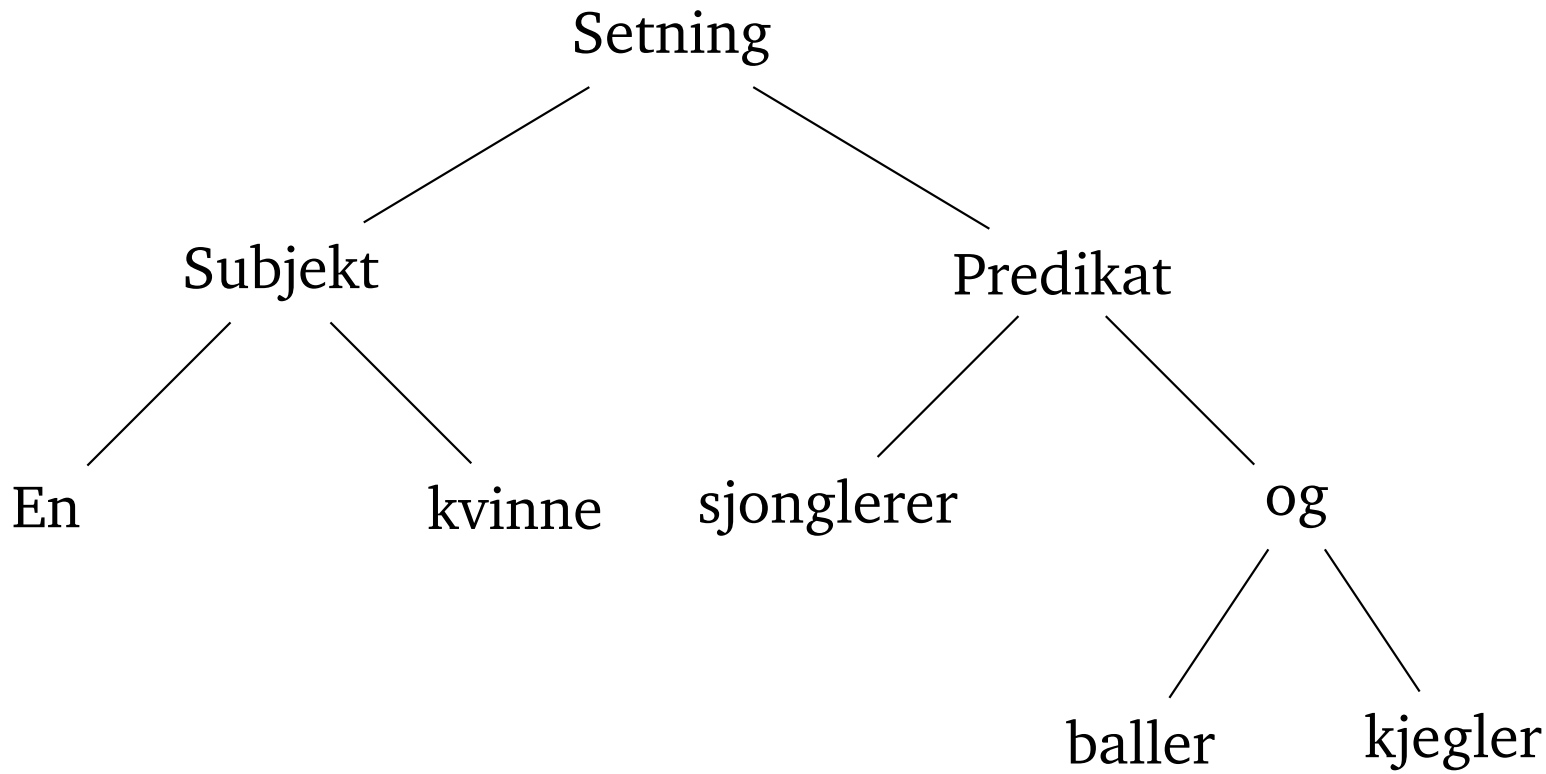
- *En kvinne sjonglerer baller og kjegler.*

Trær med rot

- *En kvinne sjonglerer baller og kjegler.*
- Denne setningen er bygget opp slik.

Trær med rot

- *En kvinne sjonglerer baller og kjegler.*
- Denne setningen er bygget opp slik.



Trær med rot

Trær med rot

- Trær kan ofte brukes til å beskrive forskjellige algoritmer hvor man stiller visse spørsmål, og prosessen videre avhenger av svarene på de enkelte spørsmålene.

Trær med rot

- Trær kan ofte brukes til å beskrive forskjellige algoritmer hvor man stiller visse spørsmål, og prosessen videre avhenger av svarene på de enkelte spørsmålene.
- I læreboka står det et eksempel på et tre av spørsmål som kan brukes til å bestemme rekkefølgen på tre forskjellige tall a, b og c.

Trær med rot

- Trær kan ofte brukes til å beskrive forskjellige algoritmer hvor man stiller visse spørsmål, og prosessen videre avhenger av svarene på de enkelte spørsmålene.
- I læreboka står det et eksempel på et tre av spørsmål som kan brukes til å bestemme rekkefølgen på tre forskjellige tall a, b og c.
- Vi skal se på en spørsmålsserie som avgjør om fire individer, a, b, c og d, tilhører samme art.

Trær med rot

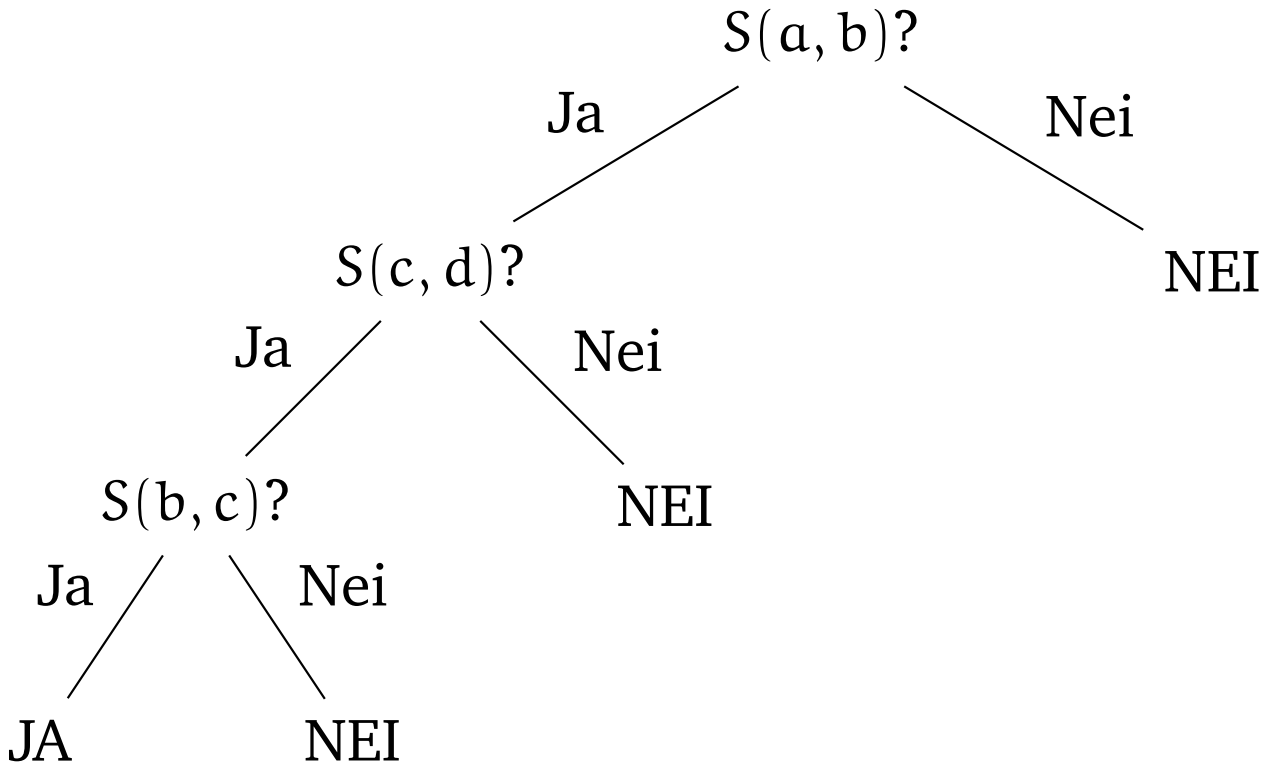
- Trær kan ofte brukes til å beskrive forskjellige algoritmer hvor man stiller visse spørsmål, og prosessen videre avhenger av svarene på de enkelte spørsmålene.
- I læreboka står det et eksempel på et tre av spørsmål som kan brukes til å bestemme rekkefølgen på tre forskjellige tall a, b og c.
- Vi skal se på en spørsmålsserie som avgjør om fire individer, a, b, c og d, tilhører samme art.
- Vi skriver $S(x, y)$ for at x og y tilhører samme art.

Trær med rot

- Trær kan ofte brukes til å beskrive forskjellige algoritmer hvor man stiller visse spørsmål, og prosessen videre avhenger av svarene på de enkelte spørsmålene.
- I læreboka står det et eksempel på et tre av spørsmål som kan brukes til å bestemme rekkefølgen på tre forskjellige tall a, b og c.
- Vi skal se på en spørsmålsserie som avgjør om fire individer, a, b, c og d, tilhører samme art.
- Vi skriver $S(x, y)$ for at x og y tilhører samme art.
- Det å tilhøre samme art er en ekvivalensrelasjon, og korrekthet av programmet bygger utelukkende på dét (på samme måte som korrekthet av eksemplet i boka bygger på at vi har en total ordning).

Trær med rot

Trær med rot



Trær med rot

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.
- Får vi positivt svar her også sjekker vi til slutt $S(b, c)$.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.
- Får vi positivt svar her også sjekker vi til slutt $S(b, c)$.
- Hvis S er transitiv vet vi at a, b, c og d ligger etter hverandre i S-relasjonen.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.
- Får vi positivt svar her også sjekker vi til slutt $S(b, c)$.
- Hvis S er transitiv vet vi at a, b, c og d ligger etter hverandre i S-relasjonen.
- En algoritme som virker for veldig mange tolkninger av input kalles gjerne en polymorf algoritme; det vil si at den virker på mange strukturer.

Trær med rot

- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.
- Får vi positivt svar her også sjekker vi til slutt $S(b, c)$.
- Hvis S er transitiv vet vi at a, b, c og d ligger etter hverandre i S-relasjonen.
- En algoritme som virker for veldig mange tolkninger av input kalles gjerne en polymorf algoritme; det vil si at den virker på mange strukturer.
- Sorteringsalgoritmer er eksempler på polymorfe algoritmer.

Trær med rot

Trær med rot

- Hvis vi tar utgangspunkt i et tre uten rot, og så tegner det som et tre med rot, vil det visuelle resultatet avhenge mye av hvor vi plasserer roten.

Trær med rot

- Hvis vi tar utgangspunkt i et tre uten rot, og så tegner det som et tre med rot, vil det visuelle resultatet avhenge mye av hvor vi plasserer roten.

Oppgave

Trær med rot

- Hvis vi tar utgangspunkt i et tre uten rot, og så tegner det som et tre med rot, vil det visuelle resultatet avhenge mye av hvor vi plasserer roten.

Oppgave

- Tegn følgende tre som et tre med rot når vi plasserer roten henholdsvis i nodene 1, 3 og 6:

Trær med rot

- Hvis vi tar utgangspunkt i et tre uten rot, og så tegner det som et tre med rot, vil det visuelle resultatet avhenge mye av hvor vi plasserer roten.

Oppgave

- Tegn følgende tre som et tre med rot når vi plasserer roten henholdsvis i nodene 1, 3 og 6: