MAT1030 – Diskret matematikk

Forelesning 24: Grafer og trær

Dag Normann

Matematisk Institutt, Universitetet i Oslo

21. april 2008

Vektede grafer

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

Vektede grafer

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

Vektede grafer

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

Vektede grafer

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

Vektede grafer

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

Vektede grafer

- Vi har snakket om grafer og trær. Av begreper vi så på var
  - Eulerkretser og Eulerstier
  - Hamiltonkretser
  - Vektede grafer
  - Minimale utspennende trær.
- Vi skal nå se på et realistisk eksempel på en situasjon som langt på vei kan modelleres som en vektet graf, og hvor det vil være relevant å finne en Eulerkrets eller sti, en Hamiltonkrets og et minimalt utspennende tre for å løse visse samfunnsoppgaver.

Vektede grafer

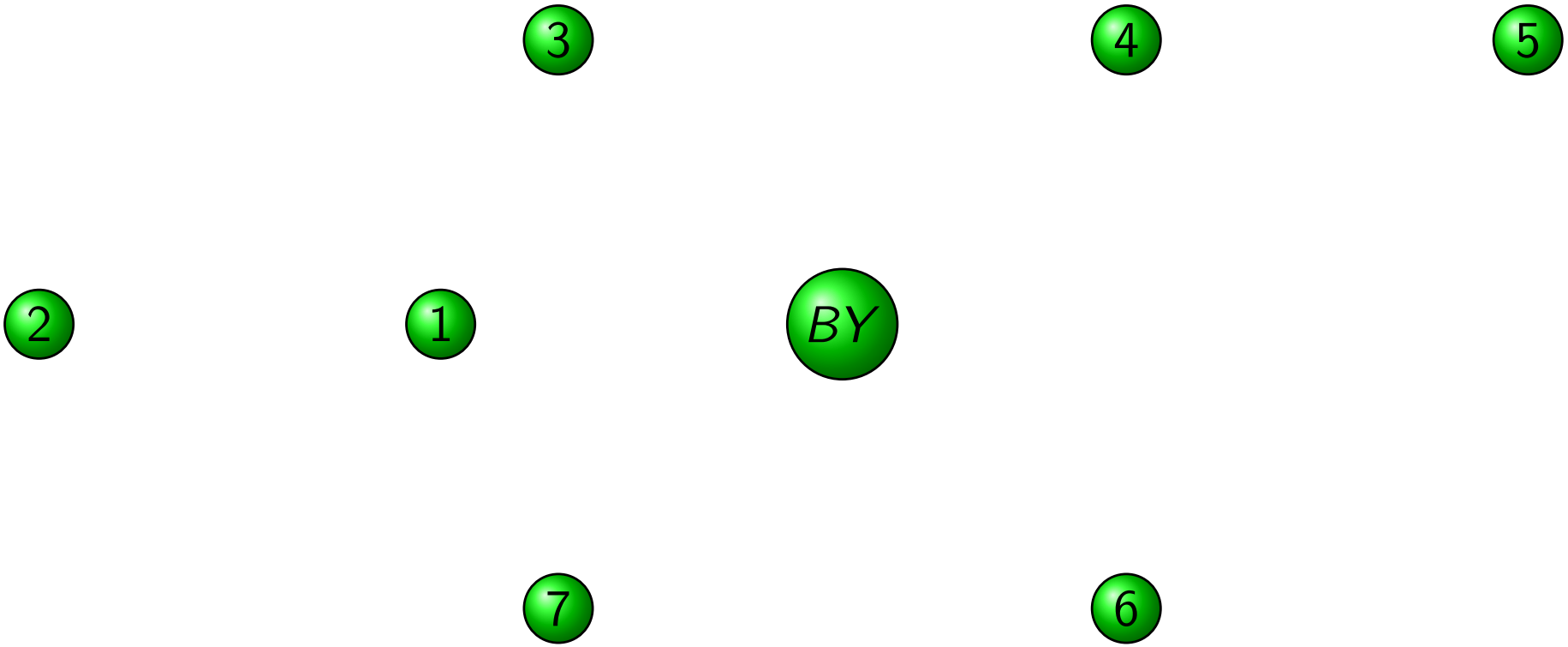
- Vi har snakket om grafer og trær. Av begreper vi så på var
  - Eulerkretser og Eulerstier
  - Hamiltonkretser
  - Vektede grafer
  - Minimale utspennende trær.
- Vi skal nå se på et realistisk eksempel på en situasjon som langt på vei kan modelleres som en vektet graf, og hvor det vil være relevant å finne en Eulerkrets eller sti, en Hamiltonkrets og et minimalt utspennende tre for å løse visse samfunnsoppgaver.
- I virkelighetens verden finner man ofte ikke en Eulersti når man trenger en eller en Hamiltonkrets når man trenger en, men som vi skal se, kan man alltid finne minimale utspennende trær.

Vektede grafer

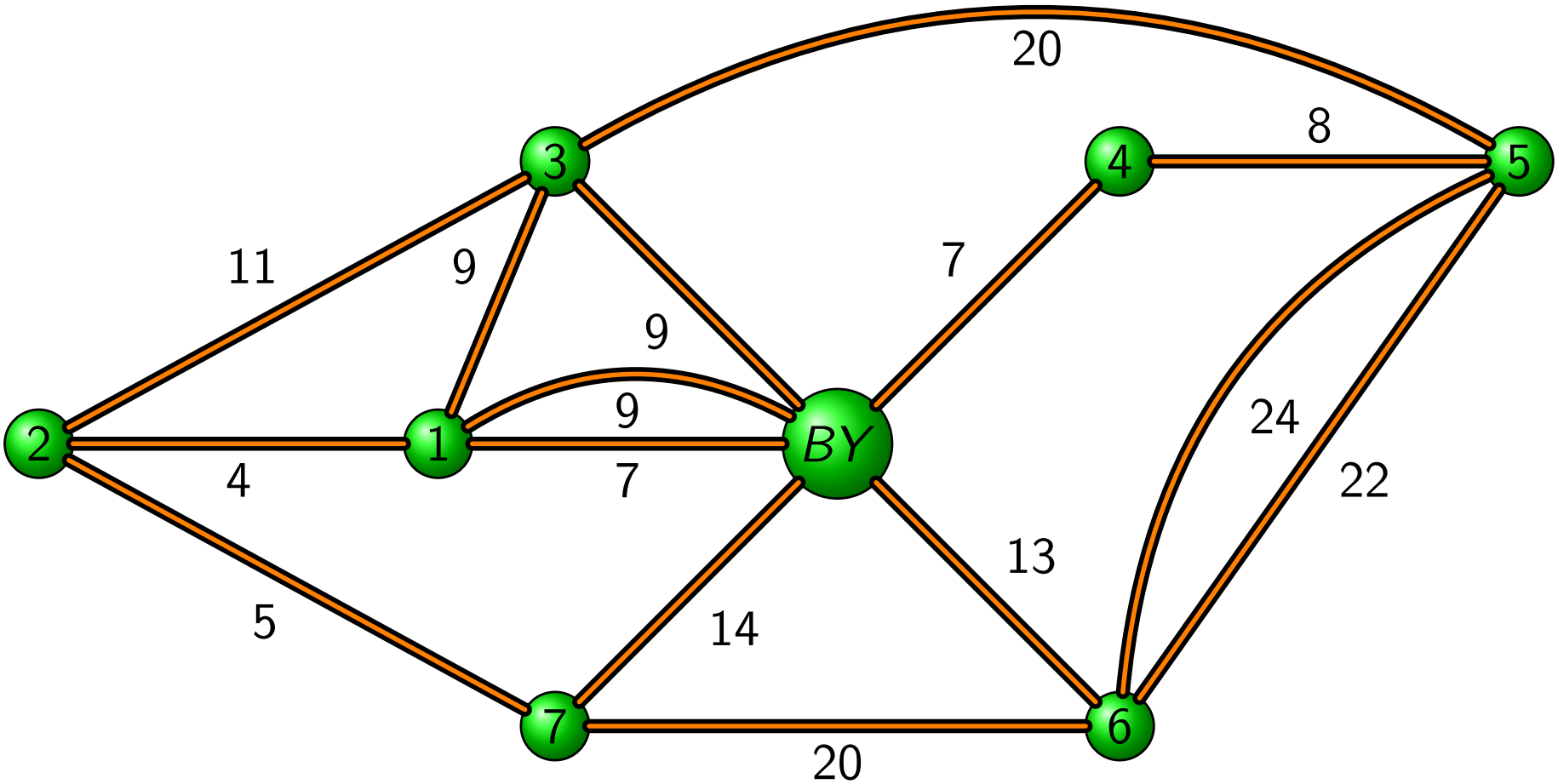
- Vi har snakket om grafer og trær. Av begreper vi så på var
  - Eulerkretser og Eulerstier
  - Hamiltonkretser
  - Vektede grafer
  - Minimale utspennende trær.
- Vi skal nå se på et realistisk eksempel på en situasjon som langt på vei kan modelleres som en vektet graf, og hvor det vil være relevant å finne en Eulerkrets eller sti, en Hamiltonkrets og et minimalt utspennende tre for å løse visse samfunnsoppgaver.
- I virkelighetens verden finner man ofte ikke en Eulersti når man trenger en eller en Hamiltonkrets når man trenger en, men som vi skal se, kan man alltid finne minimale utspennende trær.
- Vårt eksempel er en graf som modellerer veinettet mellom de lokale tettstedene i en kommune, og vektingen av kantene er antall kilometer hver enkelt veistrekning er på. Grafen er ikke *enkel*, men bortsett fra det er den som en vektet graf.

En kommunegraf

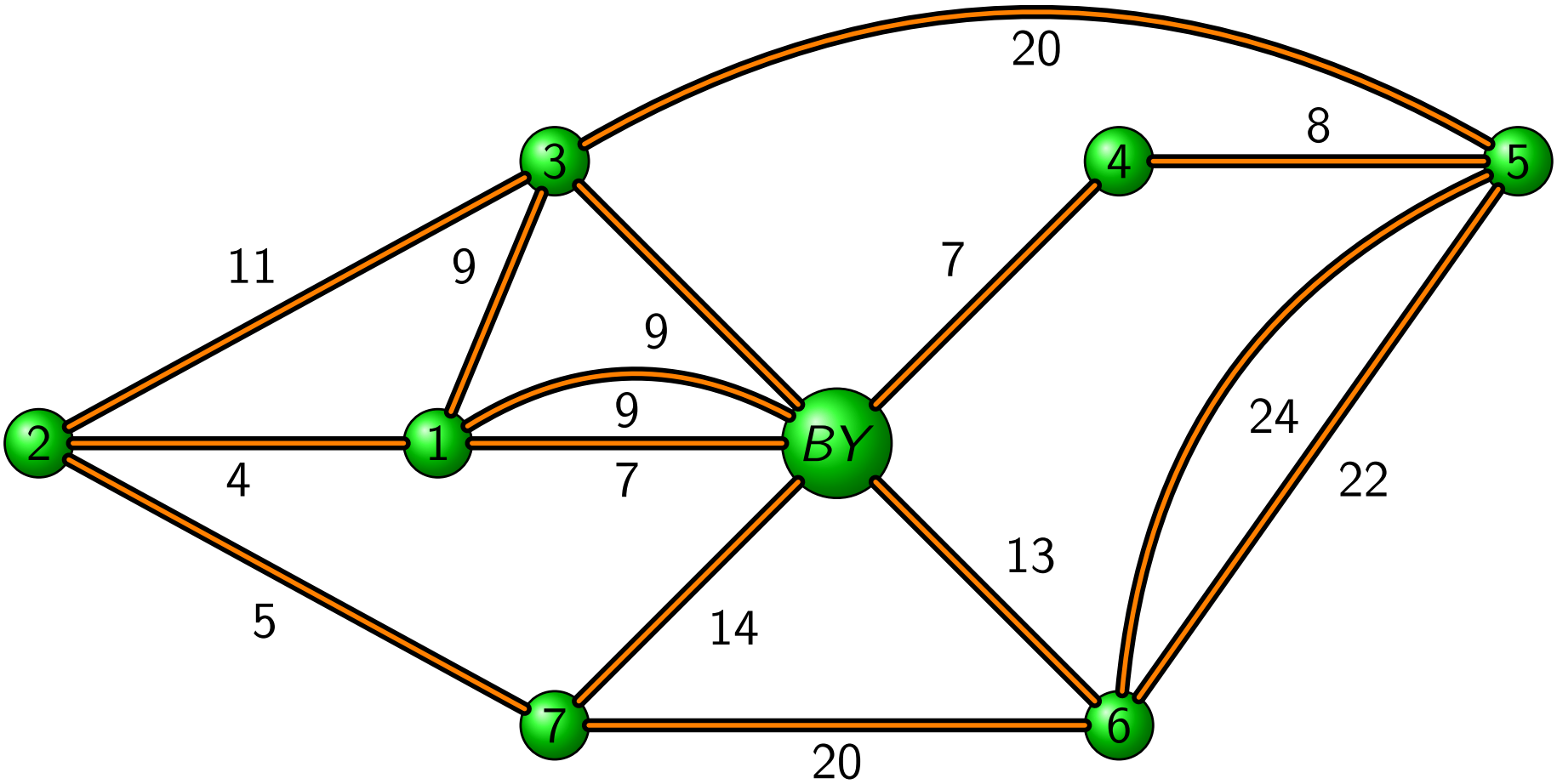
En kommunegraf



En kommunegraf

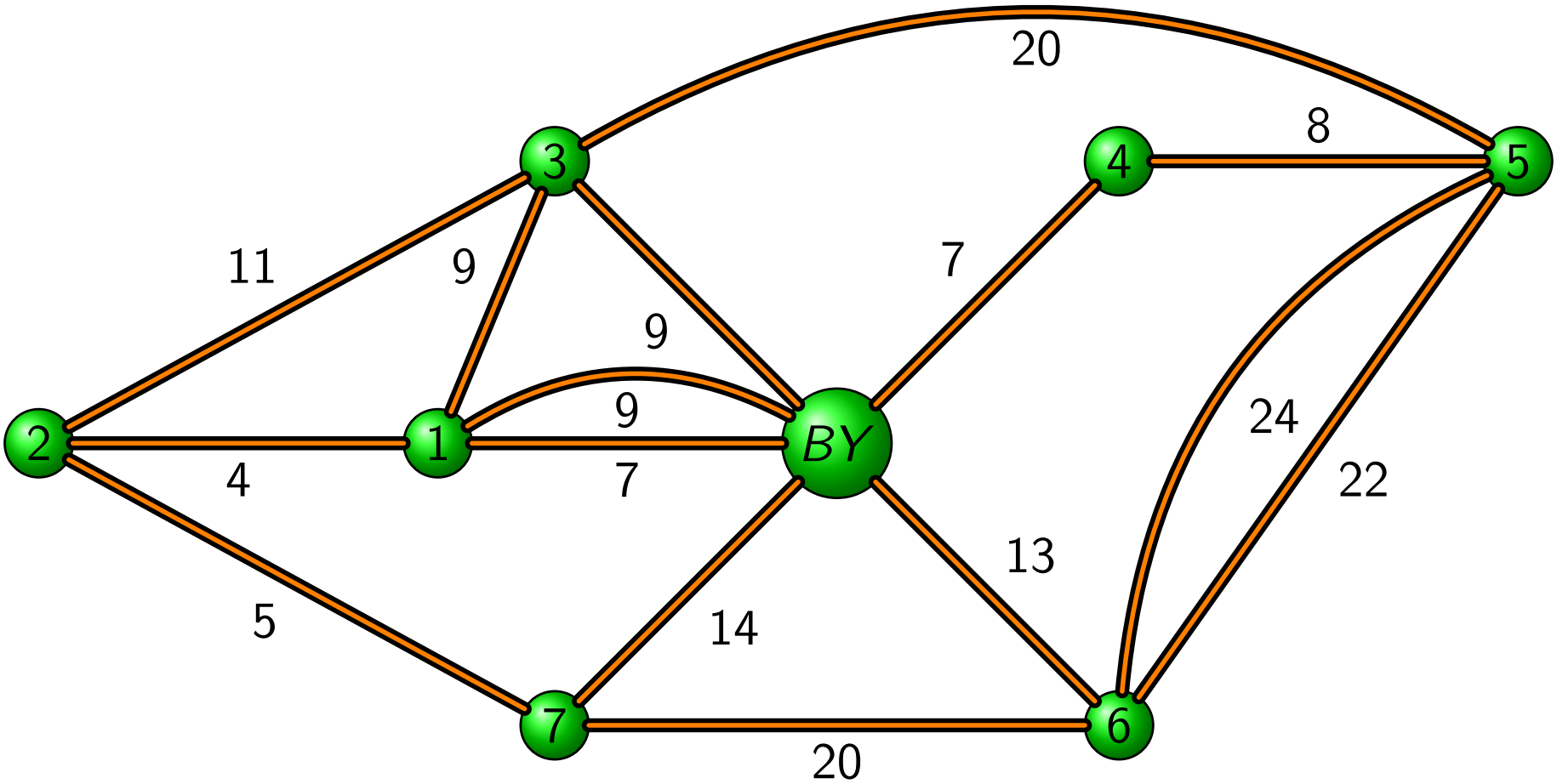


En kommunegraf



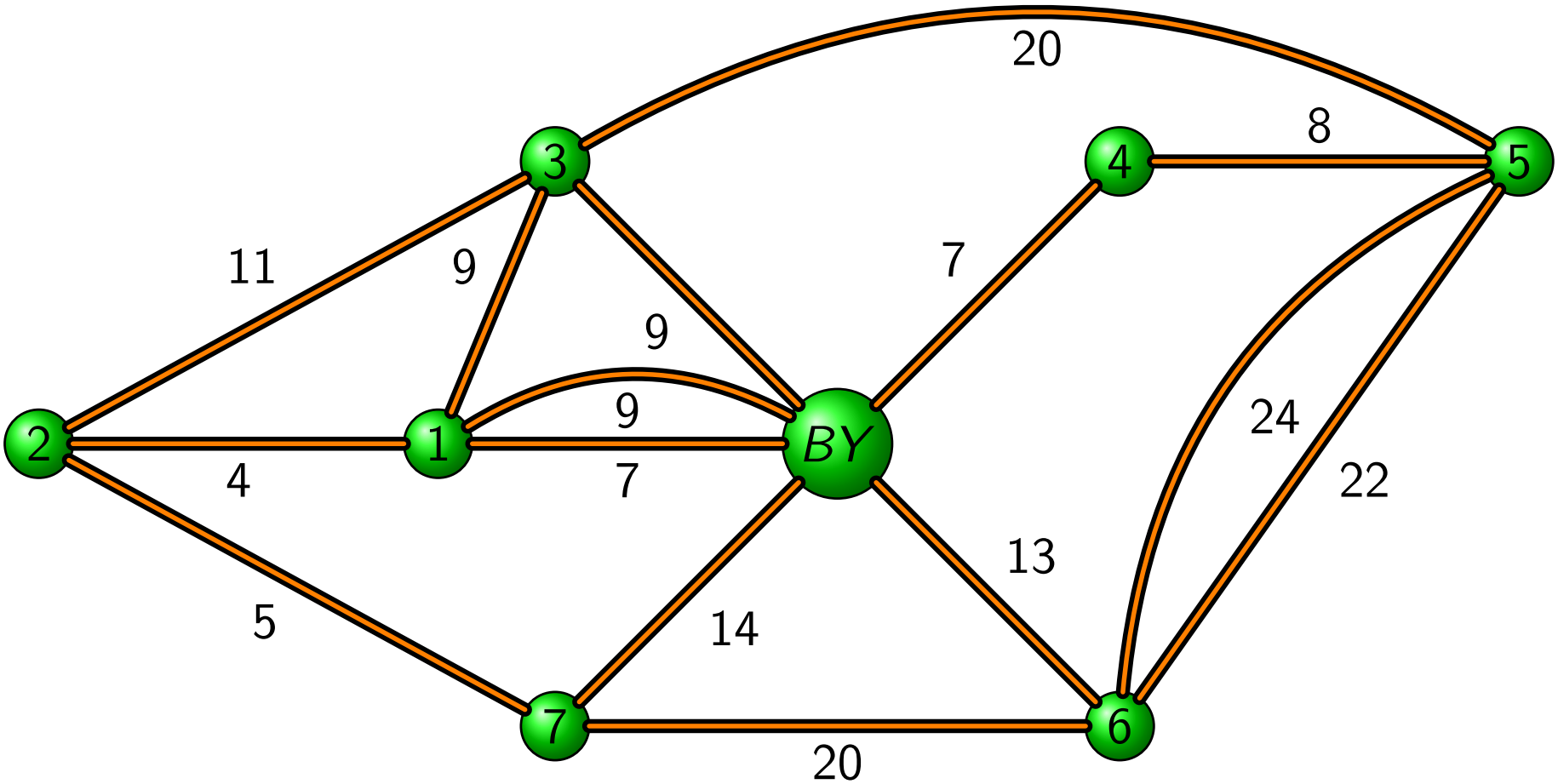
- Snøbrøyterne: *Finnes det en Eulersti?*

En kommunegraf



- Snøbrøyterne: *Finnes det en Eulersti?*
- Postutkjørerne:*Finnes det en Hamiltonkrets?*

En kommunegraf



- Snøbrøyterne: *Finnes det en Eulersti?*
- Postutkjørerne:*Finnes det en Hamiltonkrets?*
- Bredbåndutbyggerne:*Finnes det et minimalt utspennende tre?*

En kommunegraf

Oppgave

En kommunegraf

Oppgave

a) Avgjør om det finnes en Eulerkrets eller en Eulersti i kommunegrafen, og finn i så fall denne.

En kommunegraf

Oppgave

a) Avgjør om det finnes en Eulerkrets eller en Eulersti i kommunegrafen, og finn i så fall denne.

b) Er spørsmålet om det finnes en Hamiltonkrets det rette spørsmålet?

En kommunegraf

Oppgave

a) Avgjør om det finnes en Eulerkrets eller en Eulersti i kommunegrafen, og finn i så fall denne.

b) Er spørsmålet om det finnes en Hamiltonkrets det rette spørsmålet?

Kunne postutkjørerne stilt et mer fornuftig grafteoretisk spørsmål?

En kommunegraf

Oppgave

a) Avgjør om det finnes en Eulerkrets eller en Eulersti i kommunegrafen, og finn i så fall denne.

b) Er spørsmålet om det finnes en Hamiltonkrets det rette spørsmålet?

Kunne postutkjørerne stilt et mer fornuftig grafteoretisk spørsmål?

c) Finn et minimalt utspennende tre (bruker stoff fra resten av forelesningen).

En kommunegraf

Oppgave

a) Avgjør om det finnes en Eulerkrets eller en Eulersti i kommunegrafen, og finn i så fall denne.

b) Er spørsmålet om det finnes en Hamiltonkrets det rette spørsmålet?

Kunne postutkjørerne stilt et mer fornuftig grafteoretisk spørsmål?

c) Finn et minimalt utspennende tre (bruker stoff fra resten av forelesningen).

For å få en vektet graf i tråd med definisjonen, kan du ta bort unødige kanter med mye vekt der det finnes parallelle kanter.

Vektede grafer

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

Vektede grafer

- 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

Vektede grafer

- 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.

Vektede grafer

- 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.

Vektede grafer

- 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 på onsdag at hvis G har n noder, vil et utspennende tre ha $n - 1$ kanter.

Vektede grafer

- 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 på onsdag 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.

Vektede grafer

- Det kan finnes mange forskjellige utspennende trær i en graf.

Vektede grafer

- Det kan finnes mange forskjellige utspennende trær i en graf.
- Hvert slikt tre vil ha en samlet vekt, ved at vi legger sammen vektene på kantene.

Vektede grafer

- Det kan finnes mange forskjellige utspennende trær i en graf.
- Hvert slikt tre vil ha en samlet vekt, ved at vi legger sammen vektene på kantene.
- I en situasjon hvor vektene representerer kostnader, og hvor det er teknologisk nødvendig eller tilstrekkelig å erstatte grafen med et utspennende tre, er det av interesse å kunne finne et utspennende tre med minst mulig vekt.

Vektede grafer

- Det kan finnes mange forskjellige utspennende trær i en graf.
- Hvert slikt tre vil ha en samlet vekt, ved at vi legger sammen vektene på kantene.
- I en situasjon hvor vektene representerer kostnader, og hvor det er teknologisk nødvendig eller tilstrekkelig å erstatte grafen med et utspennende tre, er det av interesse å kunne finne et utspennende tre med minst mulig vekt.
- Det finnes effektive algoritmer for å kunne gjøre dette.

Vektede grafer

- Det kan finnes mange forskjellige utspennende trær i en graf.
- Hvert slikt tre vil ha en samlet vekt, ved at vi legger sammen vektene på kantene.
- I en situasjon hvor vektene representerer kostnader, og hvor det er teknologisk nødvendig eller tilstrekkelig å erstatte grafen med et utspennende tre, er det av interesse å kunne finne et utspennende tre med minst mulig vekt.
- Det finnes effektive algoritmer for å kunne gjøre dette.
- Vi skal se på en slik algoritme.

Vektede grafer

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.

Vektede grafer

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.

Vektede grafer

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.
- Her vil vi beskrive algoritmen litt mer uformelt.

Vektede grafer

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.
- Her vil vi beskrive algoritmen litt mer uformelt.
- Det viser seg at hvis man bygger opp et tre ved i hvert skritt å gjøre det som i øyeblikket virker mest fornuftig, så kommer man frem.

Vektede grafer

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.
- Her vil vi beskrive algoritmen litt mer uformelt.
- Det viser seg at hvis man bygger opp et tre ved i hvert skritt å gjøre det som i øyeblikket virker mest fornuftig, så kommer man frem.
- Vi skal ikke gi et korrekthetsbevis for Prims algoritme, men det forventes at man kan praktisere den på eksempler.

Vektede grafer

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.
- Her vil vi beskrive algoritmen litt mer uformelt.
- Det viser seg at hvis man bygger opp et tre ved i hvert skritt å gjøre det som i øyeblikket virker mest fornuftig, så kommer man frem.
- Vi skal ikke gi et korrekthetsbevis for Prims algoritme, men det forventes at man kan praktisere den på eksempler.
- Vi beskriver Prims algoritme litt anderledes enn den er formulert i læreboka, men effekten er den samme, vi får det samme treet bygget opp i den samme rekkefølgen.

Vektede grafer

- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \ldots, v_n\}$.

Vektede grafer

- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \ldots, v_n\}$.
- La T_1 være treet som består av v_1 og ingen kanter.

Vektede grafer

- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \ldots, v_n\}$.
- La T_1 være treet som består av v_1 og ingen kanter.
- Start med node v_1 og la $V_1 = \{v_2, \ldots, v_n\}$, altså resten av nodene.

Vektede grafer

- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \ldots, v_n\}$.
- La T_1 være treet som består av v_1 og ingen kanter.
- Start med node v_1 og la $V_1 = \{v_2, \ldots, v_n\}$, altså resten av nodene.
- Finn $v_{i_1} \in V_1$ med minimal avstand til v_1 via kant e_1.

Vektede grafer

- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \ldots, v_n\}$.
- La T_1 være treet som består av v_1 og ingen kanter.
- Start med node v_1 og la $V_1 = \{v_2, \ldots, v_n\}$, altså resten av nodene.
- Finn $v_{i_1} \in V_1$ med minimal avstand til v_1 via kant e_1.
- Vi får V_2 ved å fjerne v_{i_1} fra V_1 og vi får T_2 ved å legge til v_{i_1} og e_1 til T_1.

Vektede grafer

Fortsettelse av algoritmen

Vektede grafer

Fortsettelse av algoritmen

- Deretter fortsetter vi med alltid å velge den ubrukte noden som ligger nærmest, via en kant, til treet bygget opp så langt, og vi bygger ut treet med denne noden og den tilsvarende kanten.

Vektede grafer

Fortsettelse av algoritmen

- Deretter fortsetter vi med alltid å velge den ubrukte noden som ligger nærmest, via en kant, til treet bygget opp så langt, og vi bygger ut treet med denne noden og den tilsvarende kanten.
- Siden grafen er sammenhengende, vil vi alltid finne en ny node som er "nabo" til treet bygget opp på et gitt tidspunkt, og da finner vi alltid en ny node som ligger nærmest.

Vektede grafer

Fortsettelse av algoritmen

- Deretter fortsetter vi med alltid å velge den ubrukte noden som ligger nærmest, via en kant, til treet bygget opp så langt, og vi bygger ut treet med denne noden og den tilsvarende kanten.
- Siden grafen er sammenhengende, vil vi alltid finne en ny node som er "nabo" til treet bygget opp på et gitt tidspunkt, og da finner vi alltid en ny node som ligger nærmest.
- Vi skal illustrere hvordan denne algoritmen virker på eksemplet vi har gitt på en vektet graf.

Vektede grafer

Fortsettelse av algoritmen

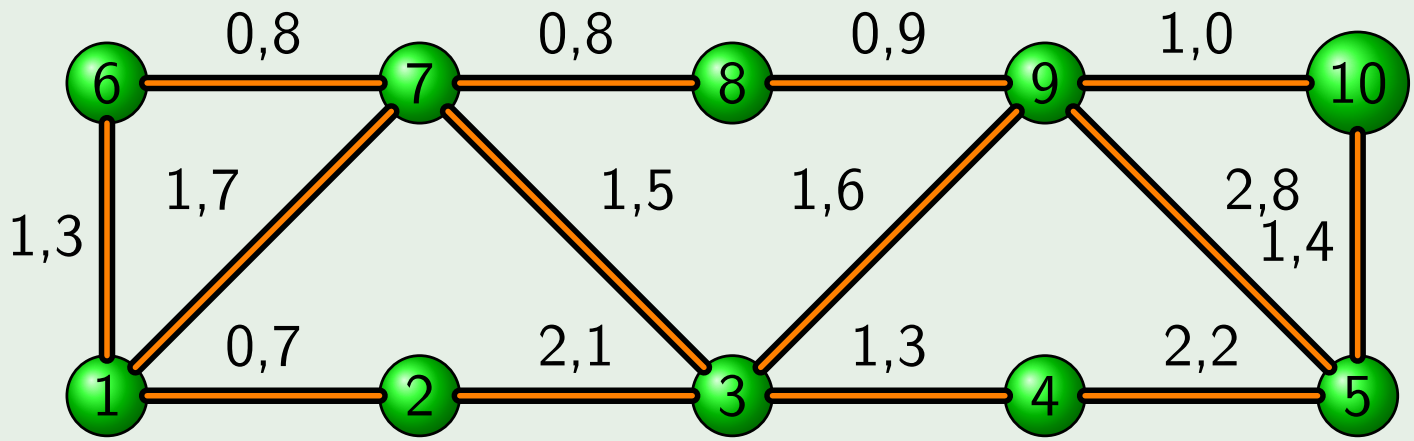
- Deretter fortsetter vi med alltid å velge den ubrukte noden som ligger nærmest, via en kant, til treet bygget opp så langt, og vi bygger ut treet med denne noden og den tilsvarende kanten.
- Siden grafen er sammenhengende, vil vi alltid finne en ny node som er "nabo" til treet bygget opp på et gitt tidspunkt, og da finner vi alltid en ny node som ligger nærmest.
- Vi skal illustrere hvordan denne algoritmen virker på eksemplet vi har gitt på en vektet graf.
- Forklaringen på hvorfor hvert enkelt skritt blir som det blir, gis muntlig på forelesningen.

Vektede grafer

Eksempel (Fortsatt)

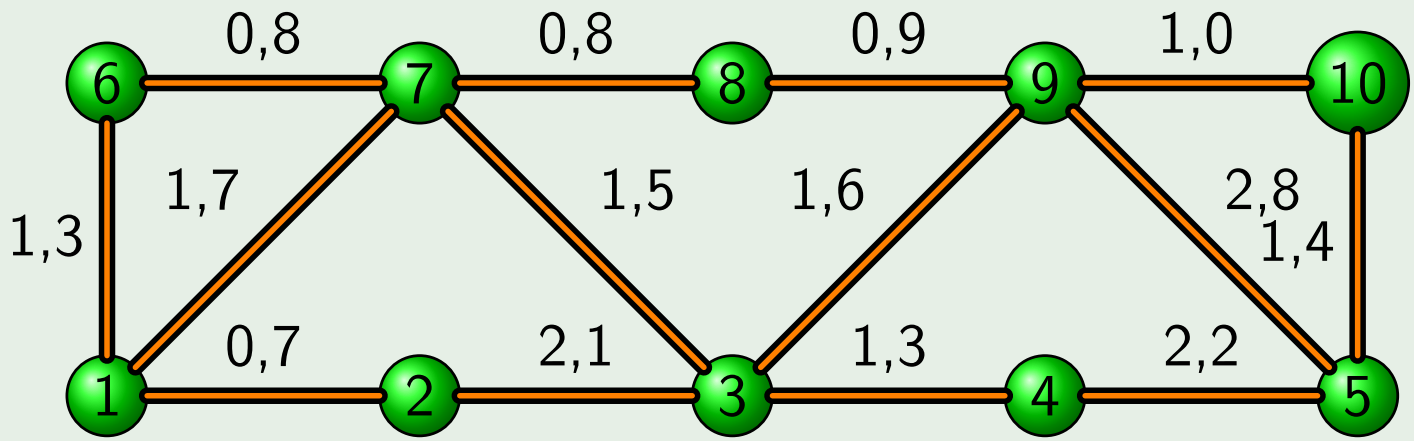
Vektede grafer

Eksempel (Fortsatt)



Vektede grafer

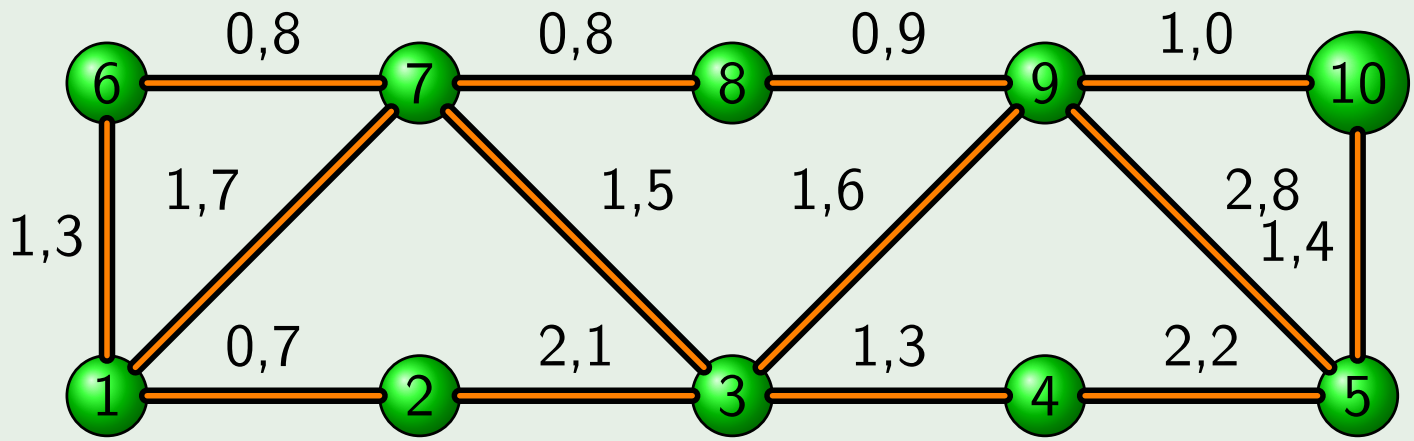
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.

Vektede grafer

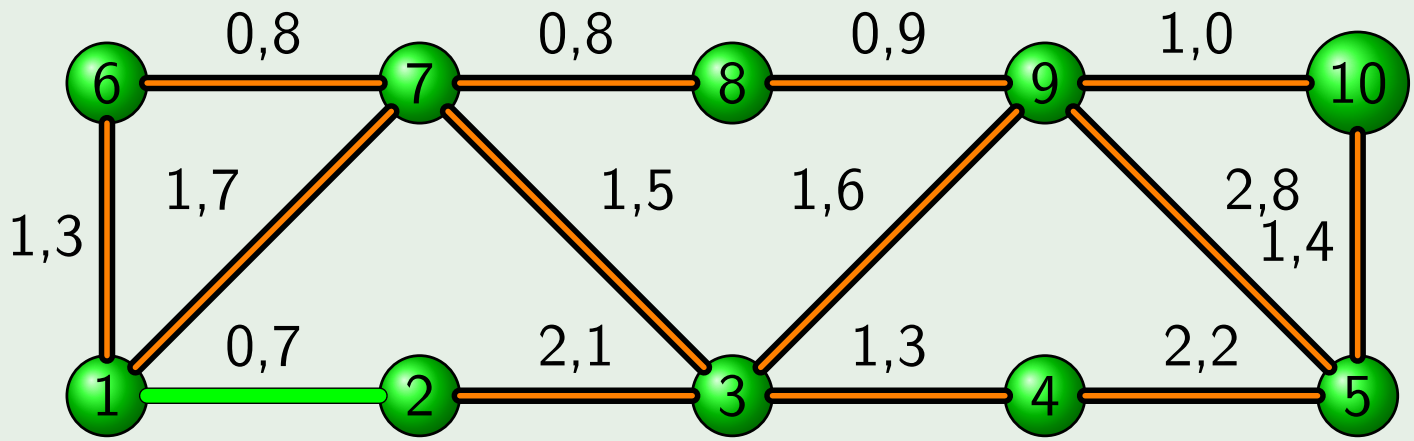
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

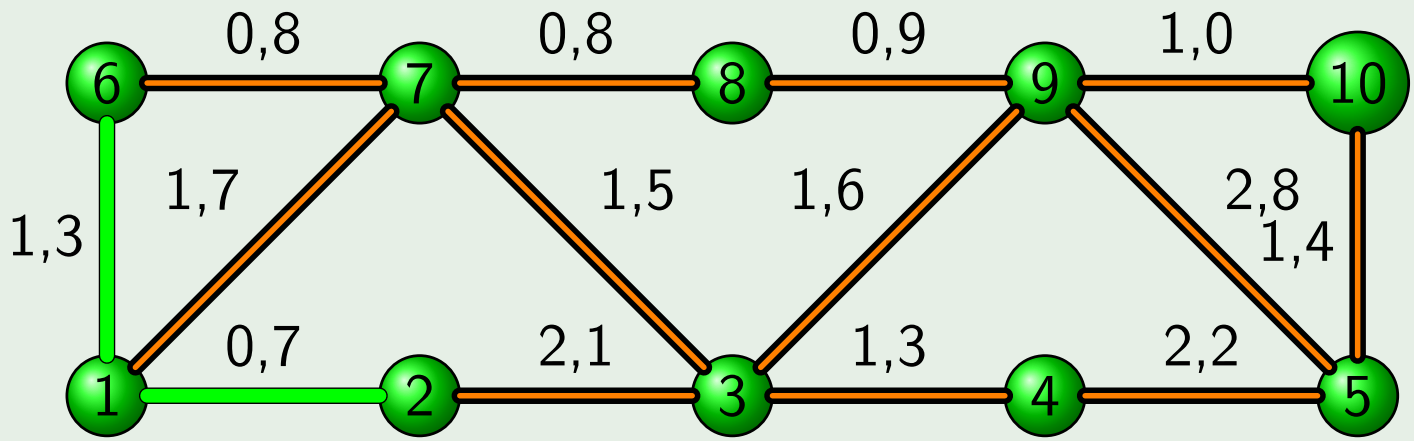
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

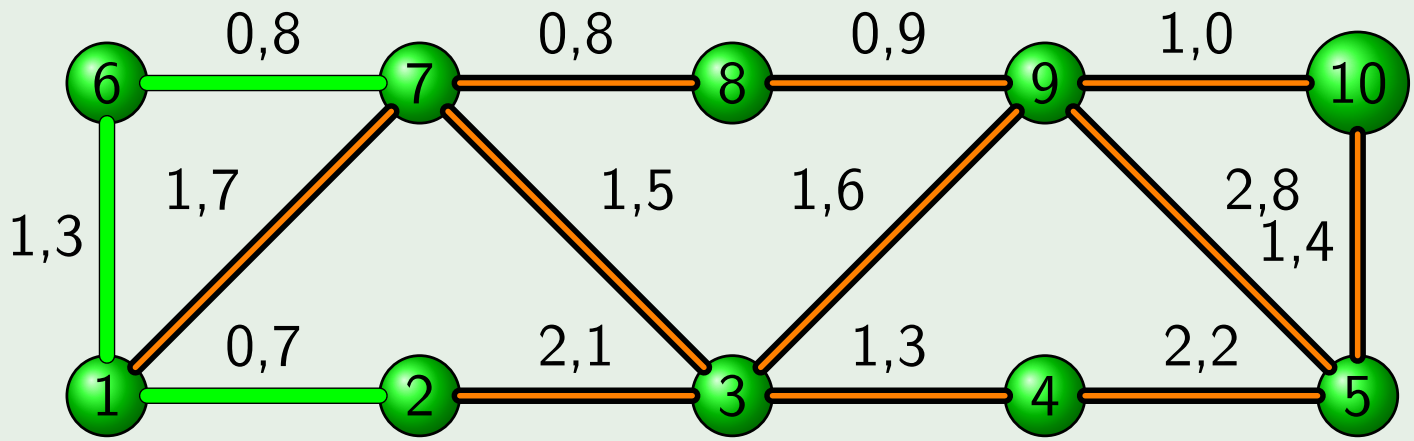
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

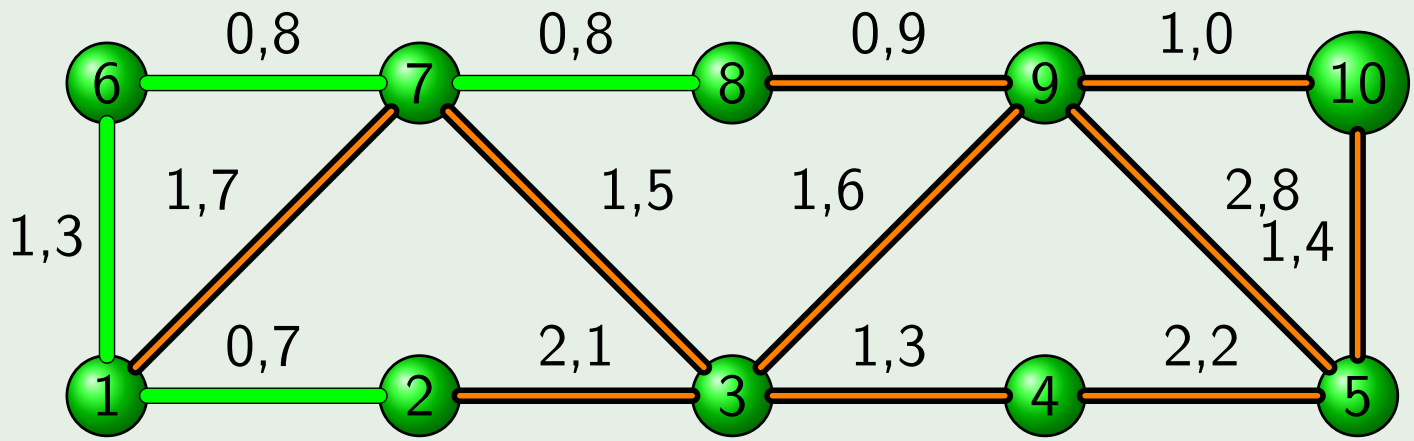
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

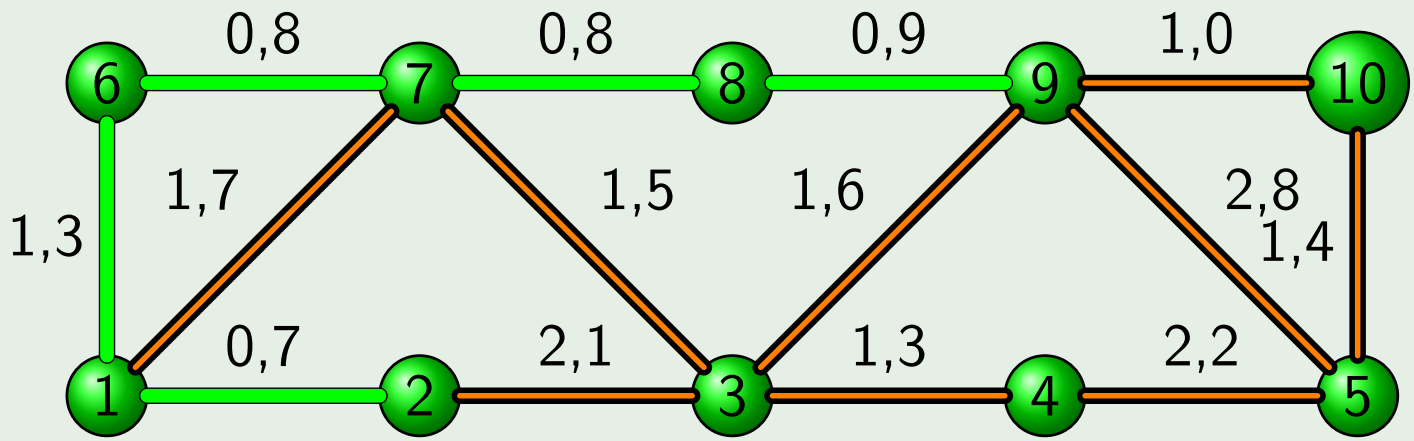
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

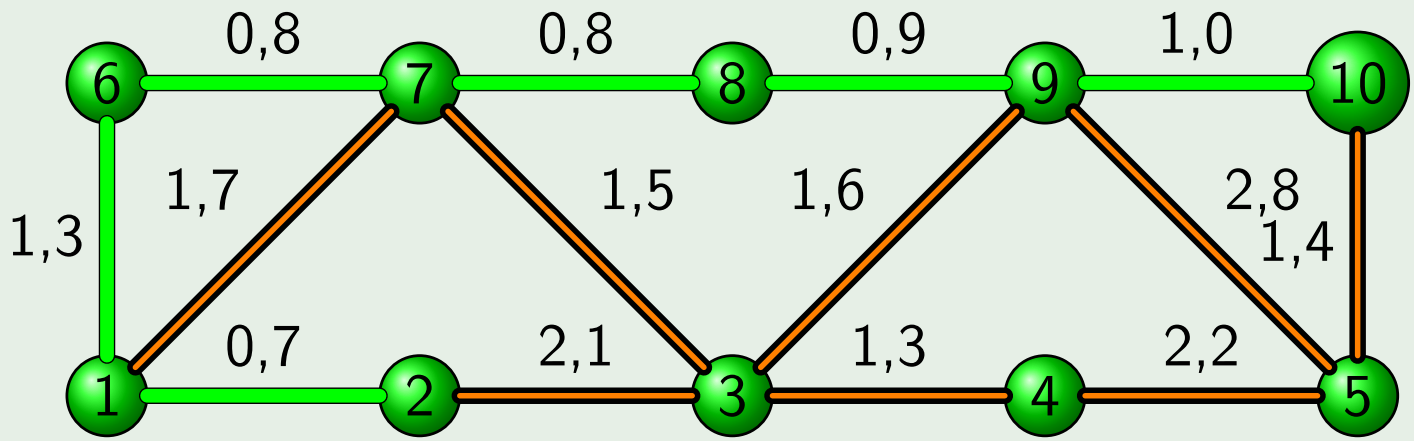
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

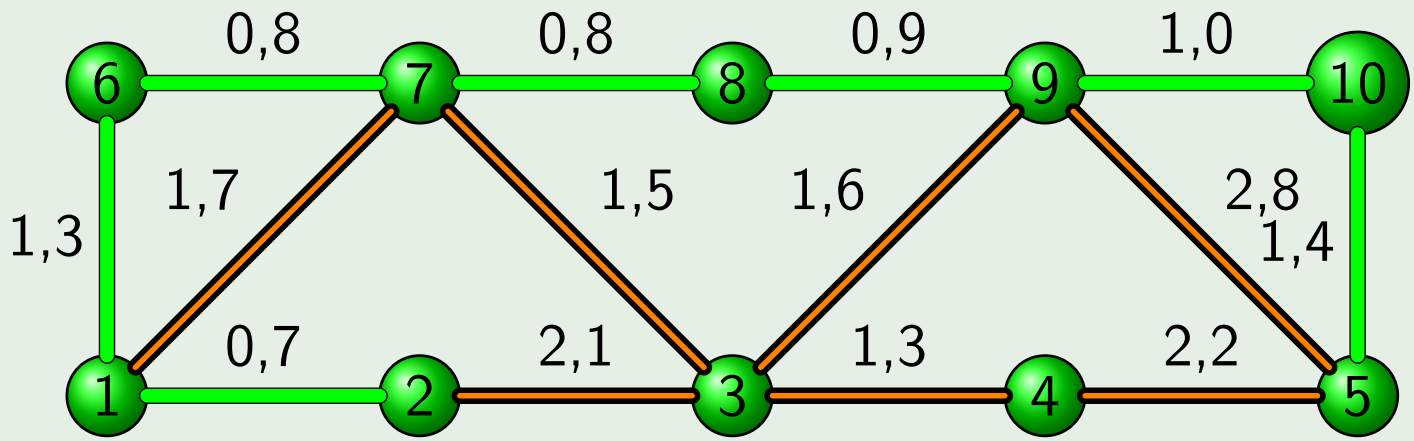
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

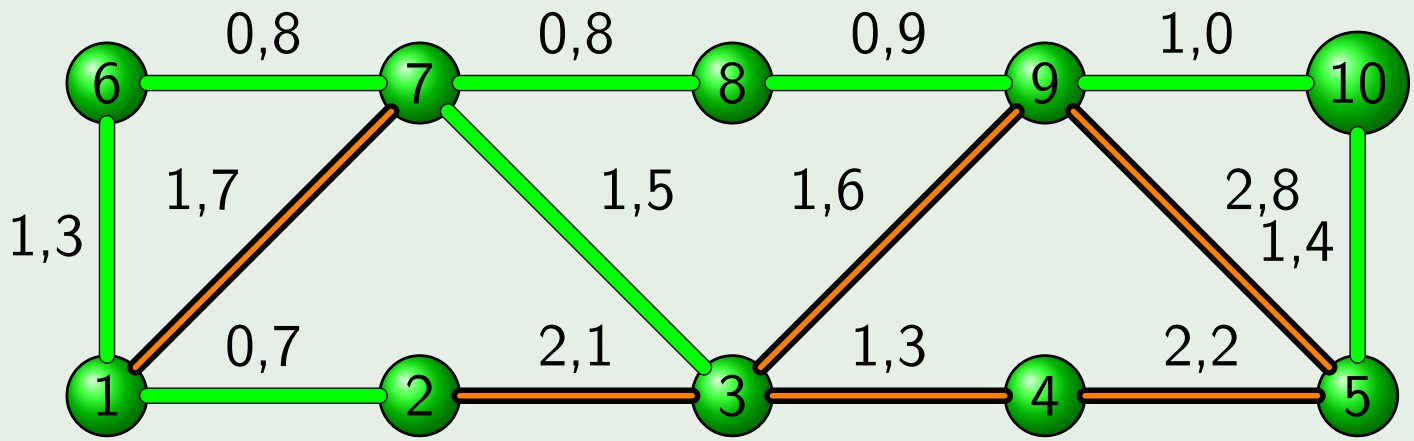
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

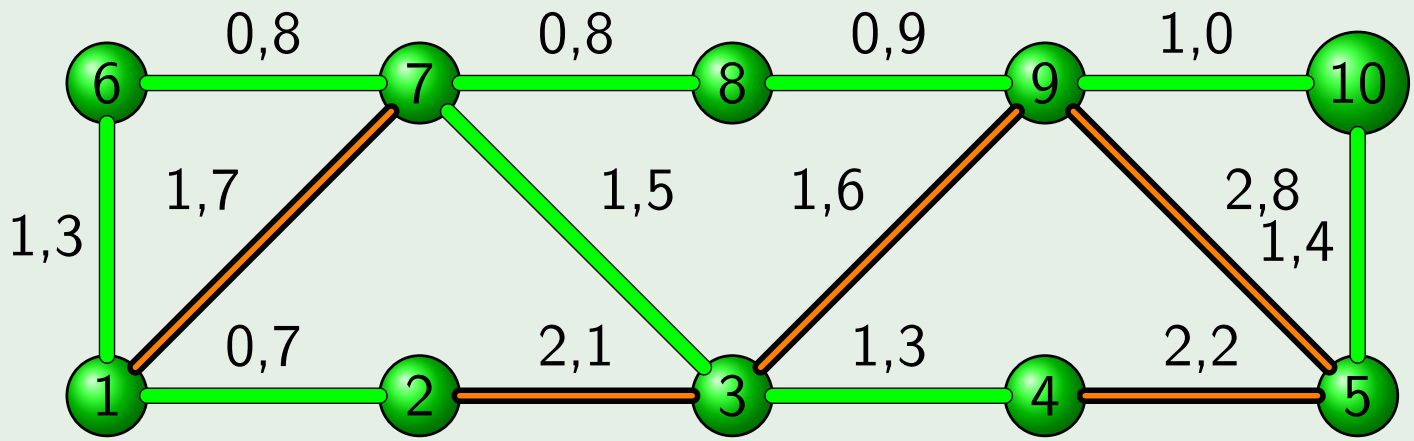
Eksempel (Fortsatt)



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

Eksempel (Fortsatt)



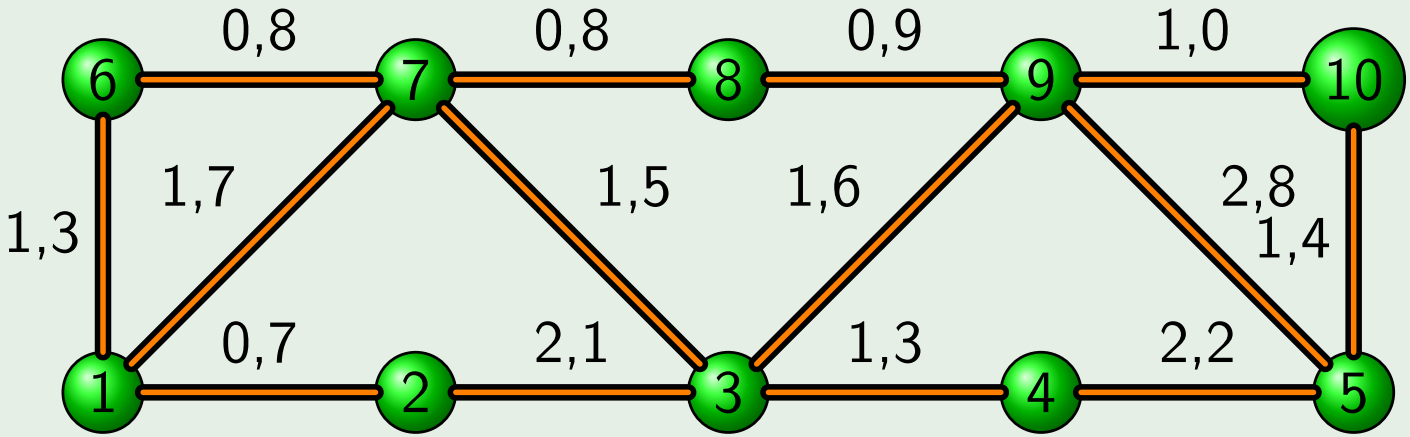
- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vekting skritt for skritt.
- Vi starter i Node 1:

Vektede grafer

Eksempel (Fortsatt)

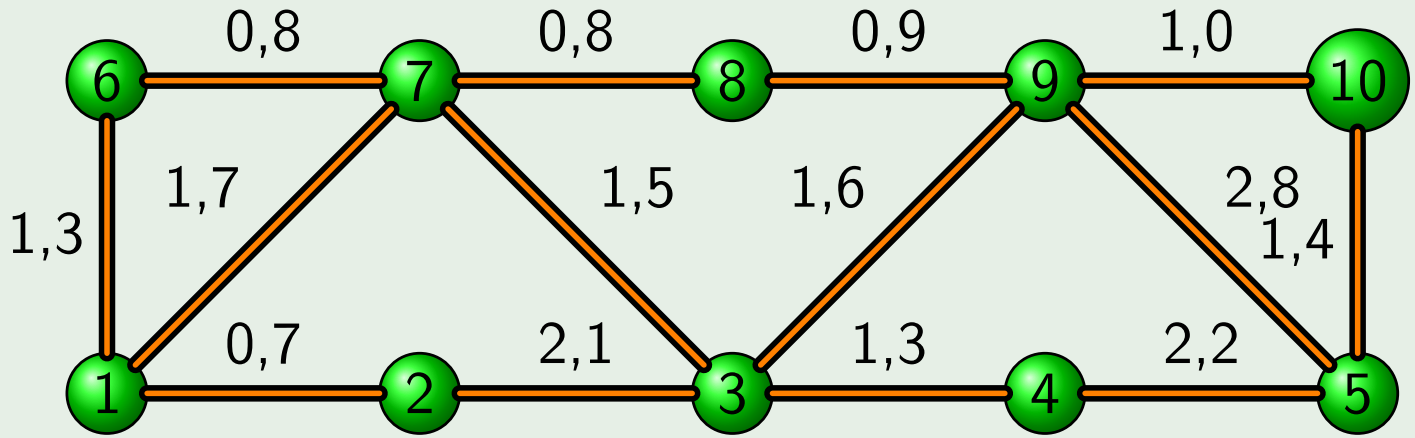
Vektede grafer

Eksempel (Fortsatt)



Vektede grafer

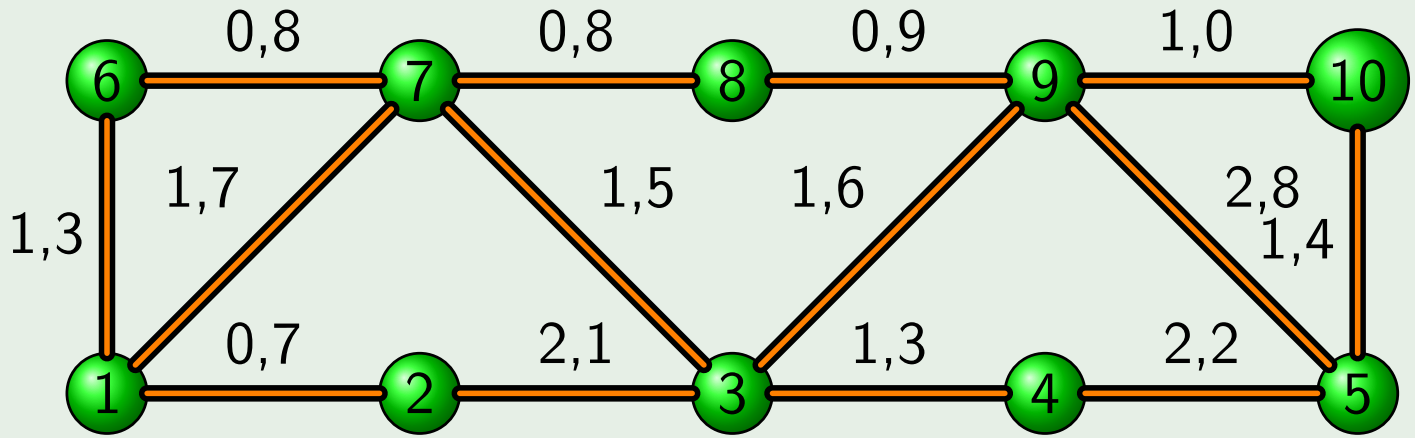
Eksempel (Fortsatt)



- 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.

Vektede grafer

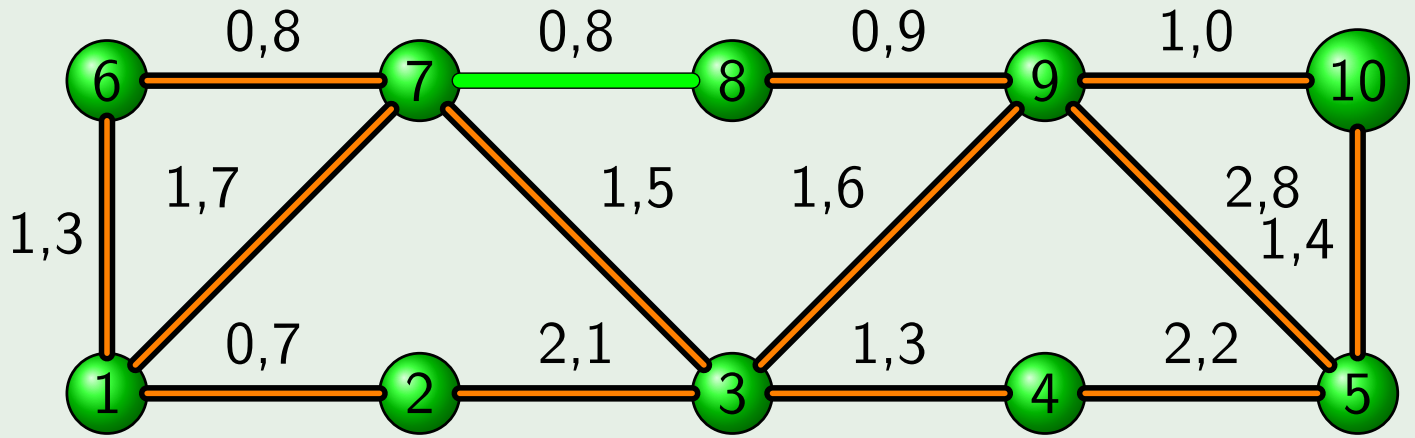
Eksempel (Fortsatt)



- 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:

Vektede grafer

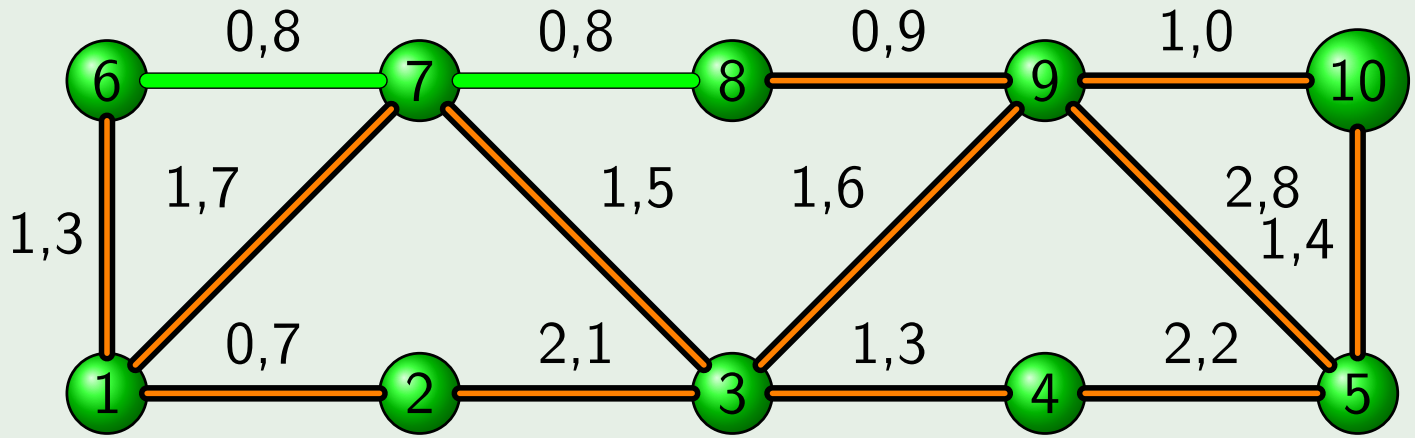
Eksempel (Fortsatt)



- 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:

Vektede grafer

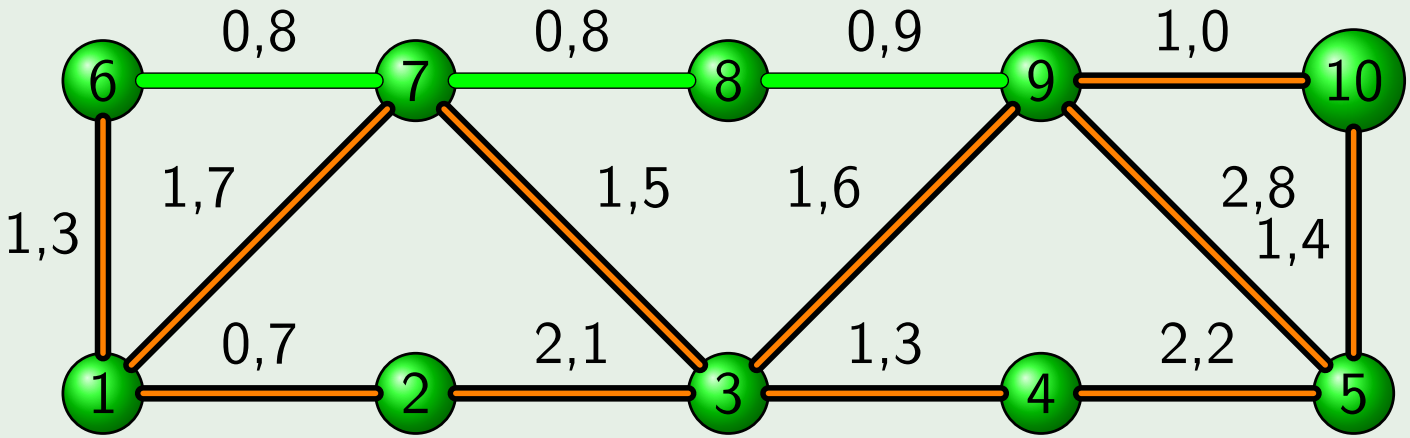
Eksempel (Fortsatt)



- 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:

Vektede grafer

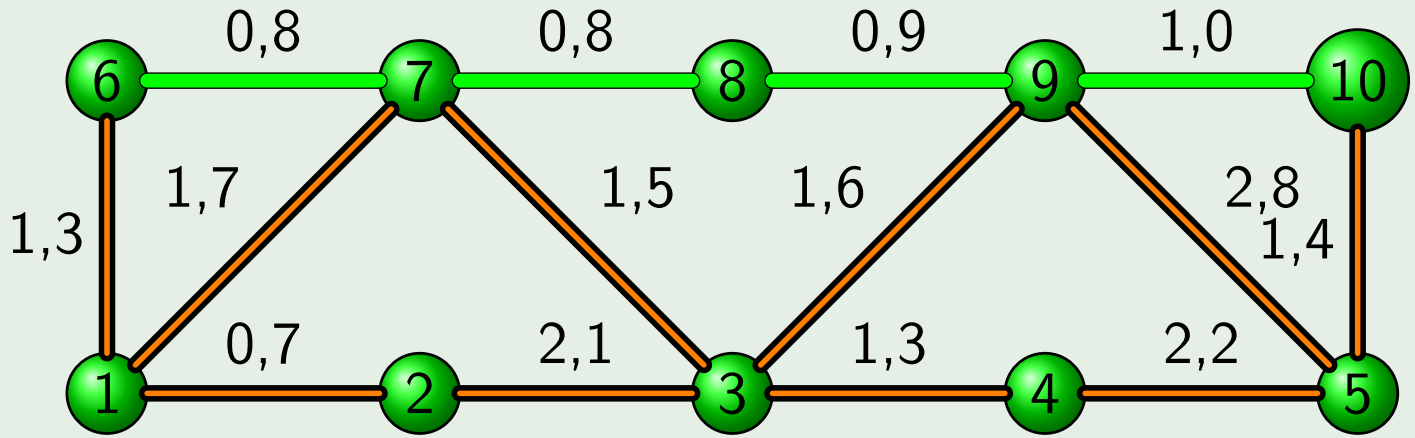
Eksempel (Fortsatt)



- 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:

Vektede grafer

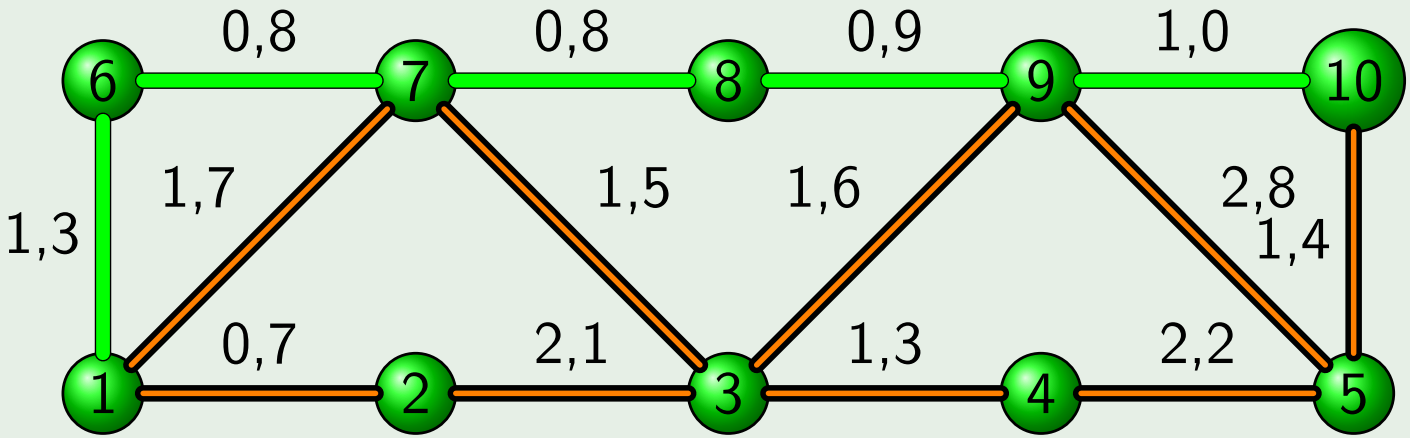
Eksempel (Fortsatt)



- 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:

Vektede grafer

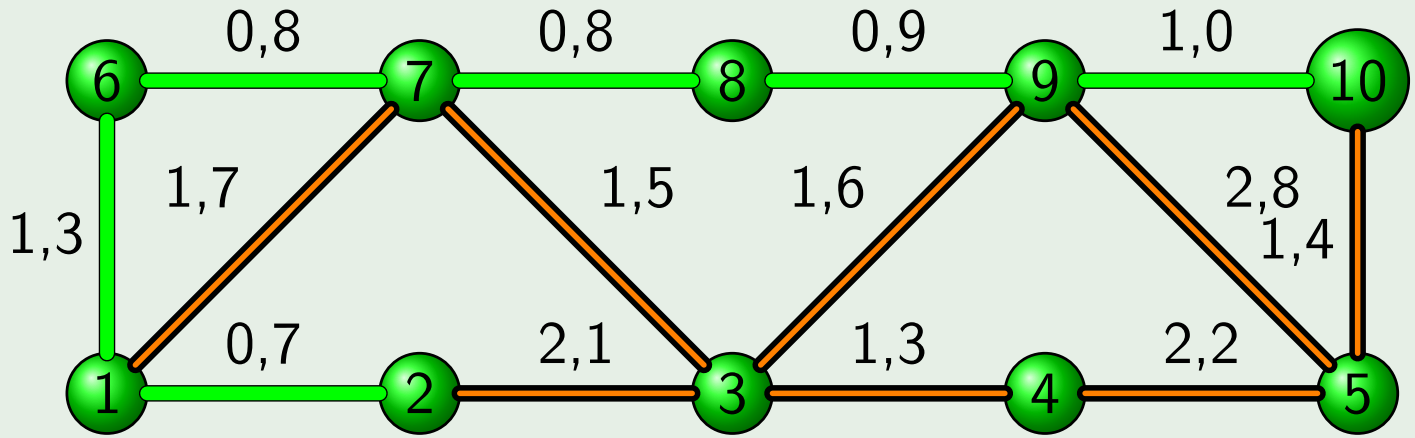
Eksempel (Fortsatt)



- 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:

Vektede grafer

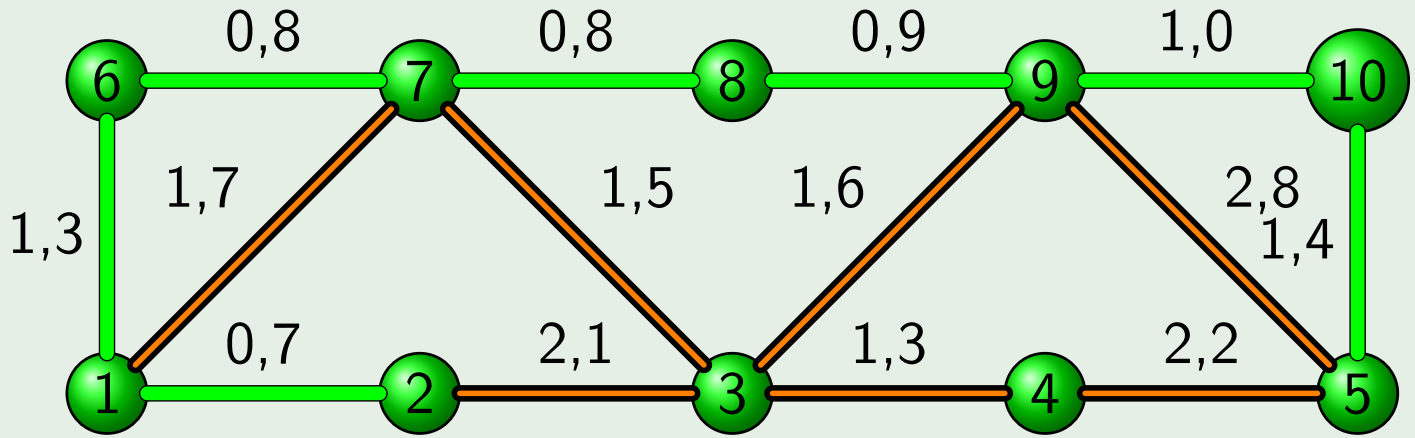
Eksempel (Fortsatt)



- 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:

Vektede grafer

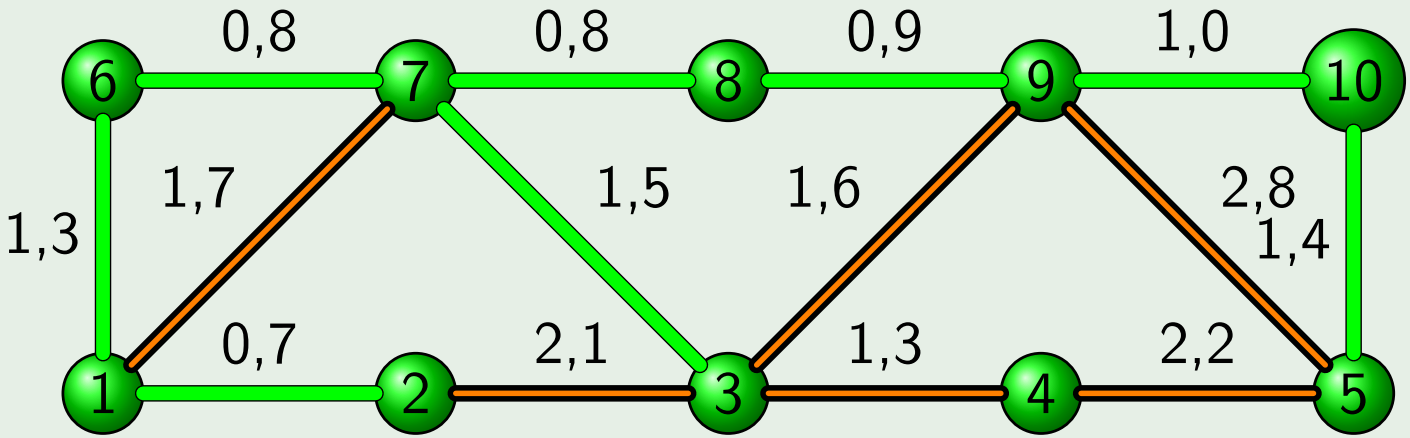
Eksempel (Fortsatt)



- 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:

Vektede grafer

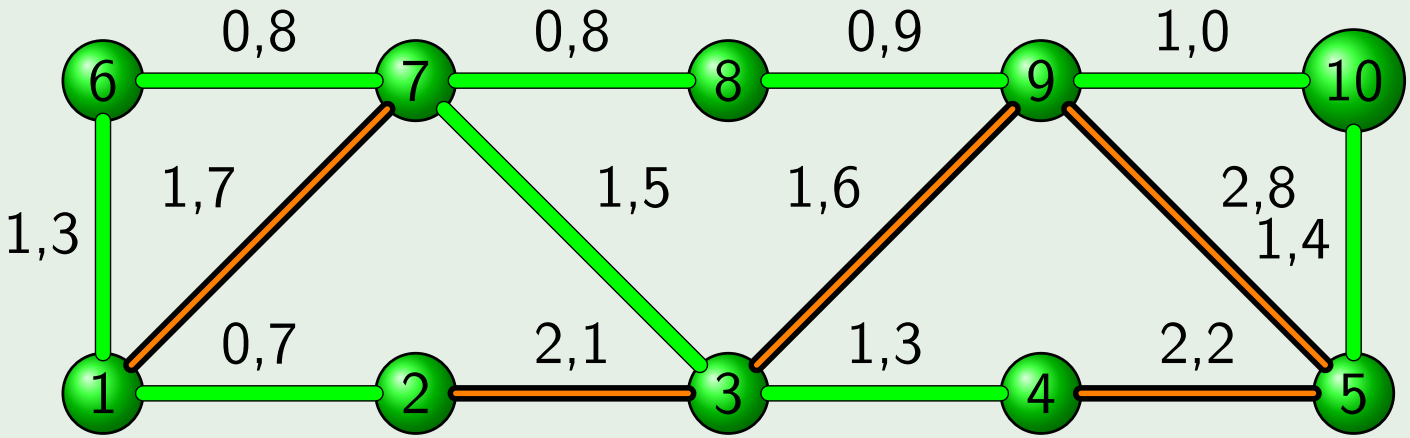
Eksempel (Fortsatt)



- 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:

Vektede grafer

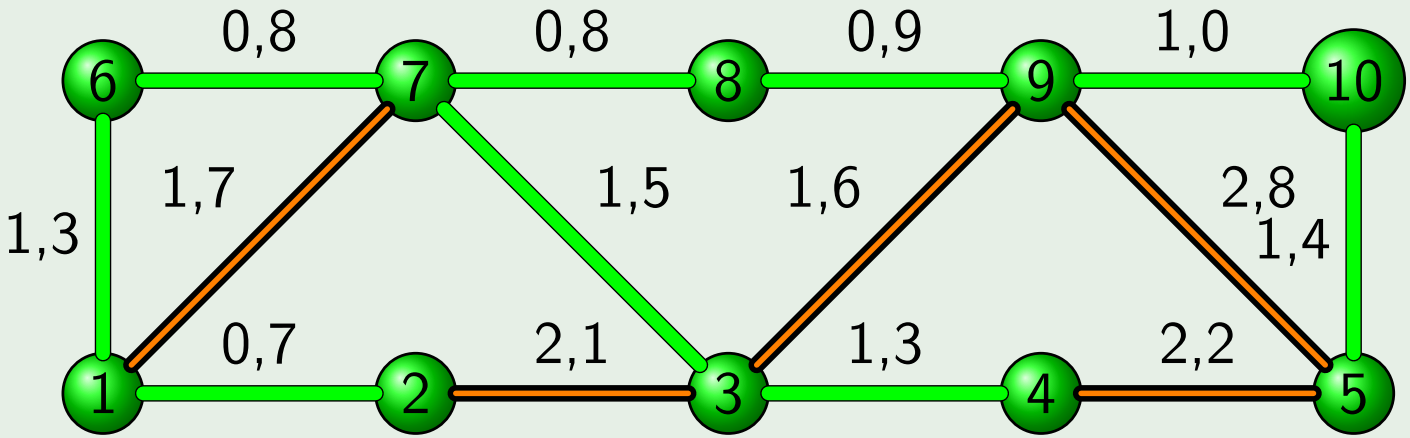
Eksempel (Fortsatt)



- 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:

Vektede grafer

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.

Vektede grafer

Vi gir en pseudokode for Prims algoritme.

Vektede grafer

Vi gir en pseudokode for Prims algoritme.

Den ser litt anderledes ut enn den som står i boka, men effekten skritt for skritt er den samme.

Vektede grafer

Vi gir en pseudokode for Prims algoritme.

Den ser litt anderledes ut enn den som står i boka, men effekten skritt for skritt er den samme.

Detaljene står på neste side:

Vektede grafer

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

Vektede grafer

1. *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]
2. *Input* $E = \{e_1, \ldots, e_k\}$ [Kantene i G]

Vektede grafer

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

Vektede grafer

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. $K \leftarrow \emptyset$

Vektede grafer

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**

Vektede grafer

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.*

Vektede grafer

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

Vektede grafer

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

Vektede grafer

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

Vektede grafer

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* $T\}$.

Vektede grafer

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).

Vektede grafer

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

Vektede grafer

- Det finnes 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.

Vektede grafer

- Det finnes 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.

Vektede grafer

- Det finnes 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.

Vektede grafer

- Det finnes 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.
- Finnes det flere aktuelle kanter med samme minimale vekt, velger vi den som står først i listen vår.

Vektede grafer

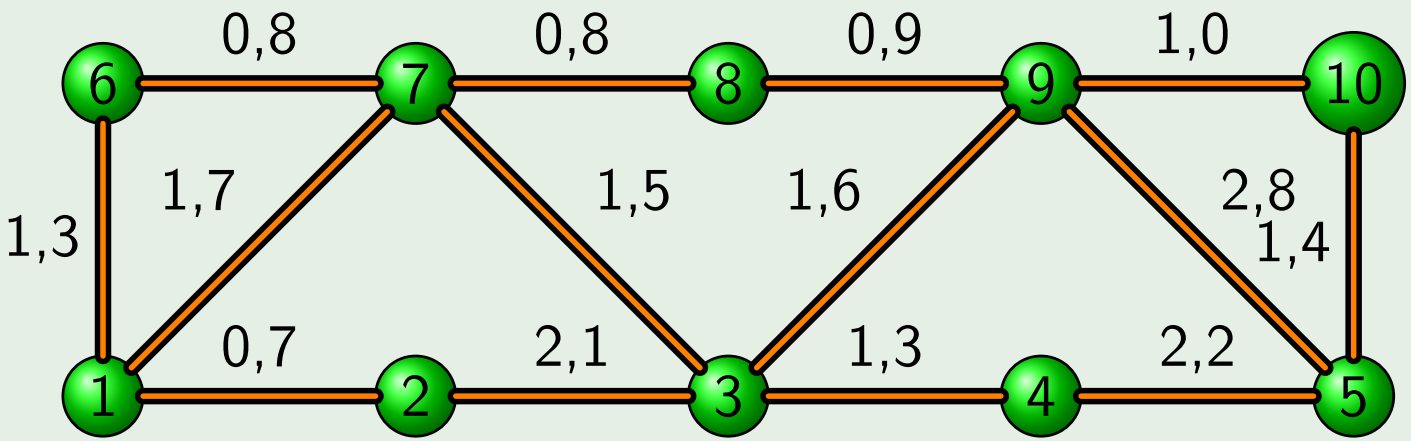
- Det finnes en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.
- Vi utnytter at vi får et utspennende tre bare vi tar med $n - 1$ kanter uten å lage noen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

Vektede grafer

Eksempel (Fortsatt)

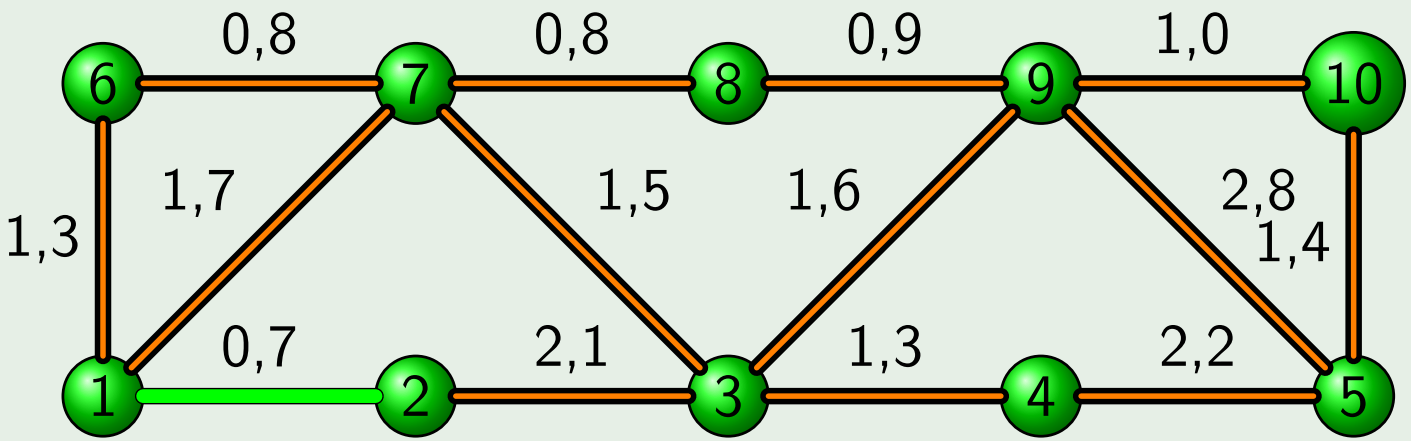
Vektede grafer

Eksempel (Fortsatt)



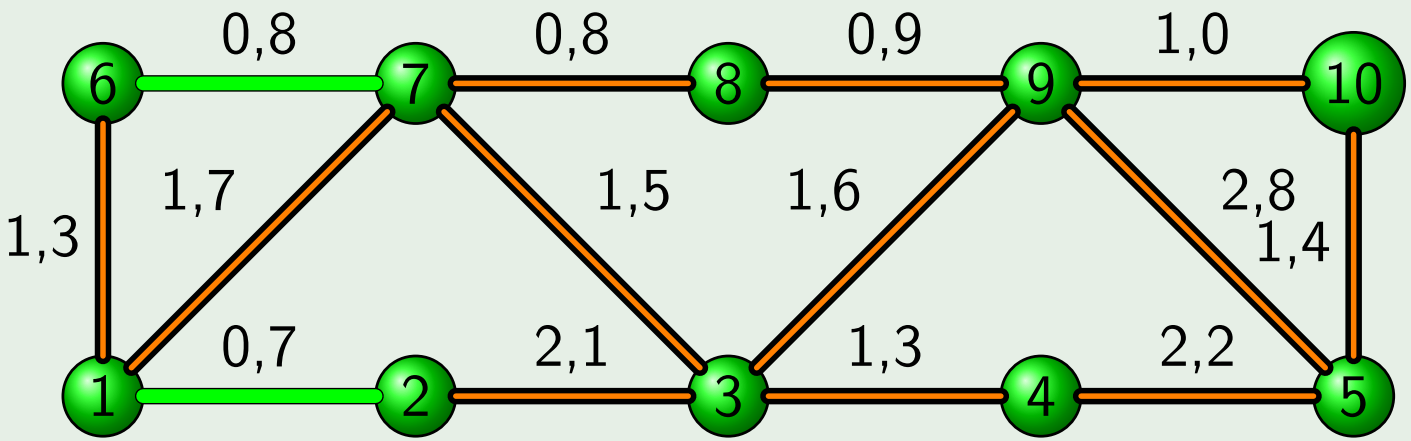
Vektede grafer

Eksempel (Fortsatt)



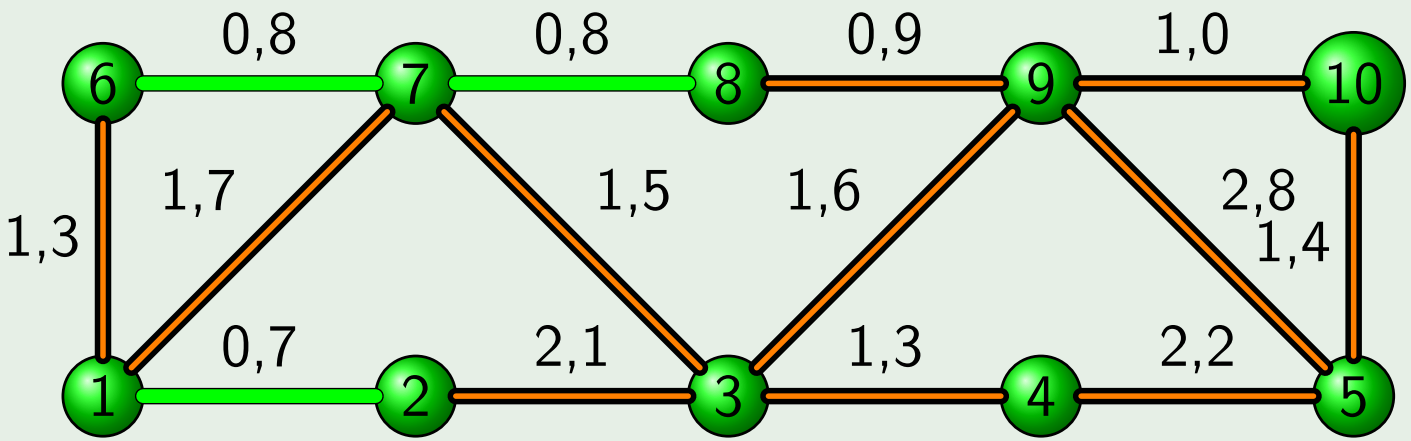
Vektede grafer

Eksempel (Fortsatt)



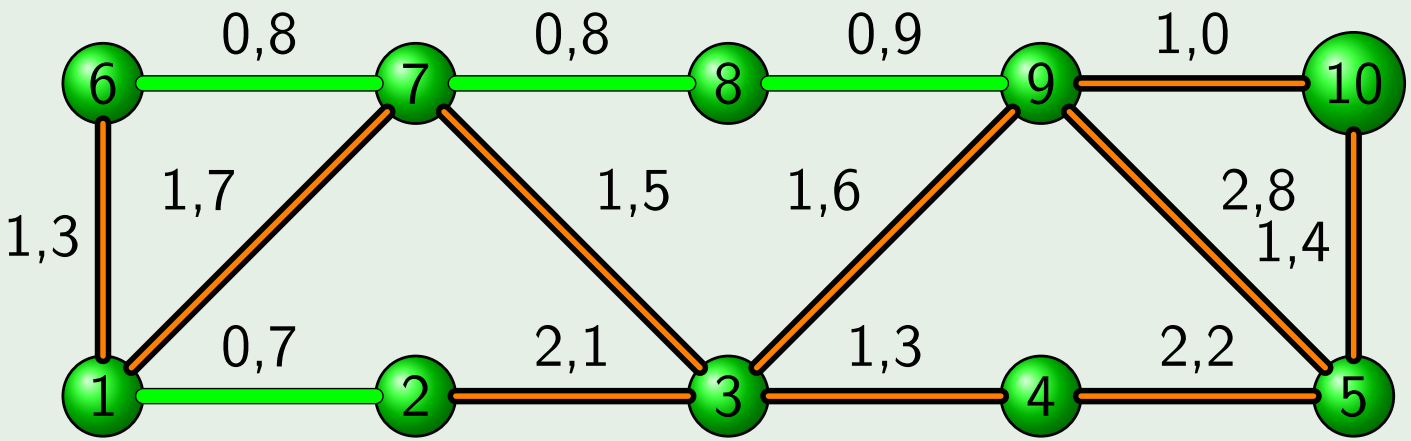
Vektede grafer

Eksempel (Fortsatt)



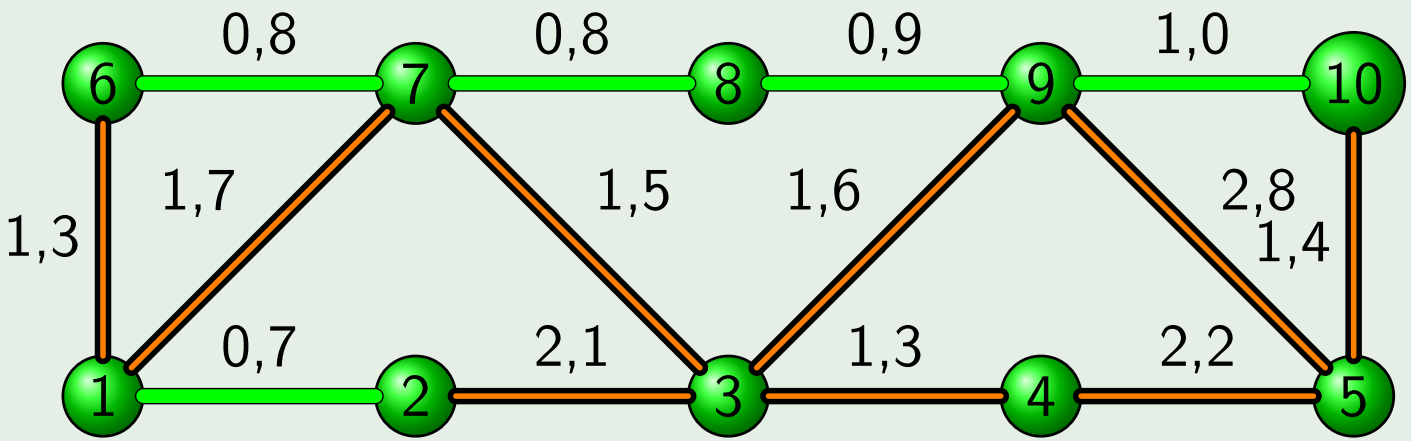
Vektede grafer

Eksempel (Fortsatt)



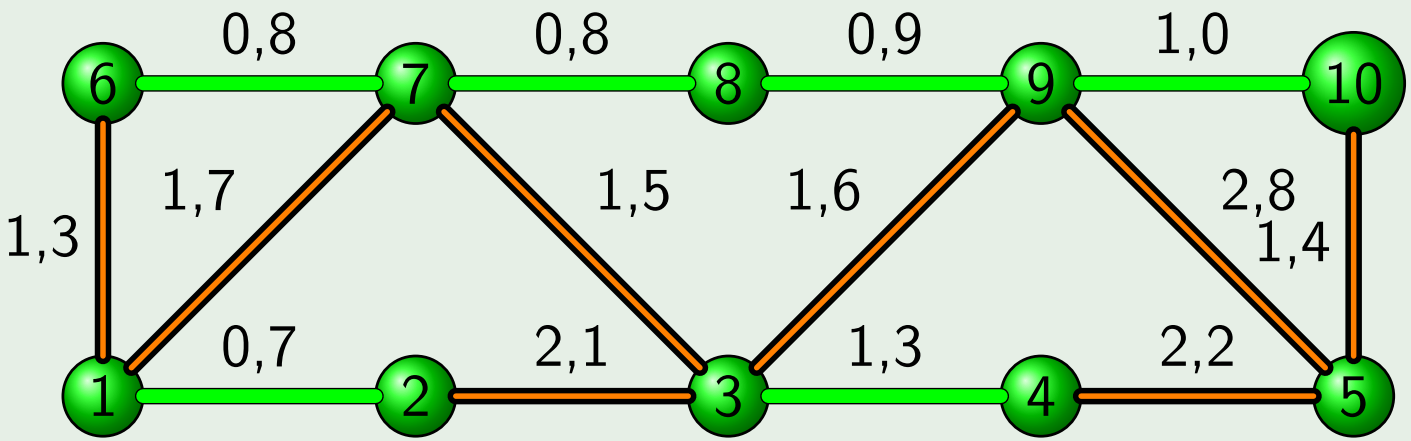
Vektede grafer

Eksempel (Fortsatt)



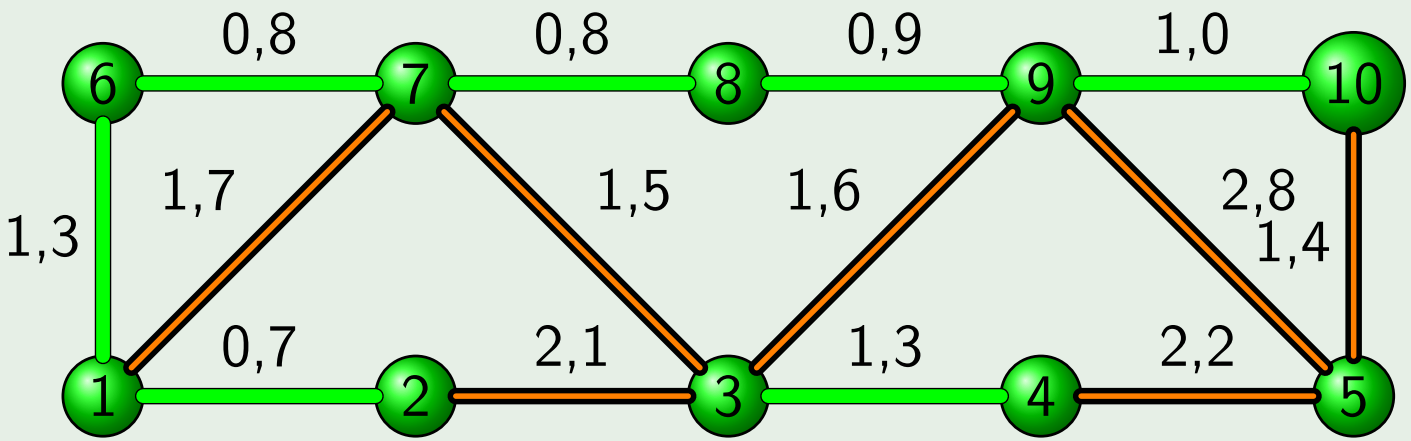
Vektede grafer

Eksempel (Fortsatt)



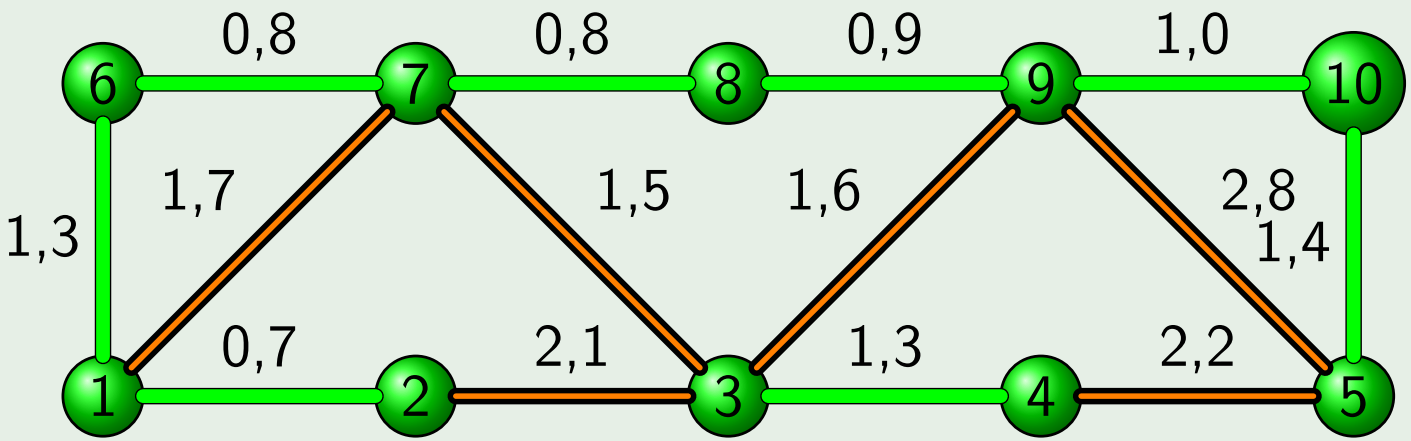
Vektede grafer

Eksempel (Fortsatt)



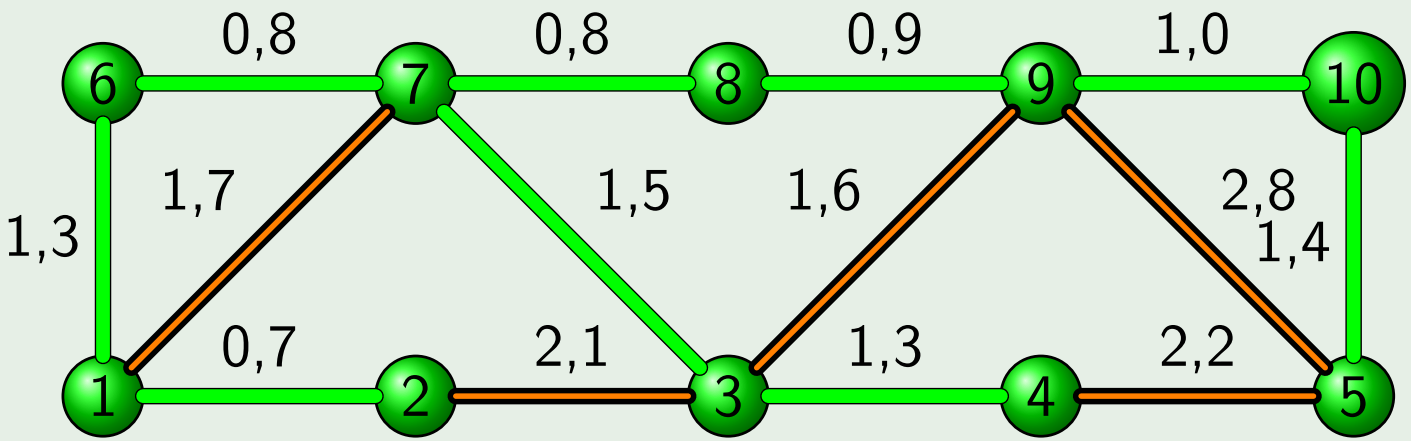
Vektede grafer

Eksempel (Fortsatt)



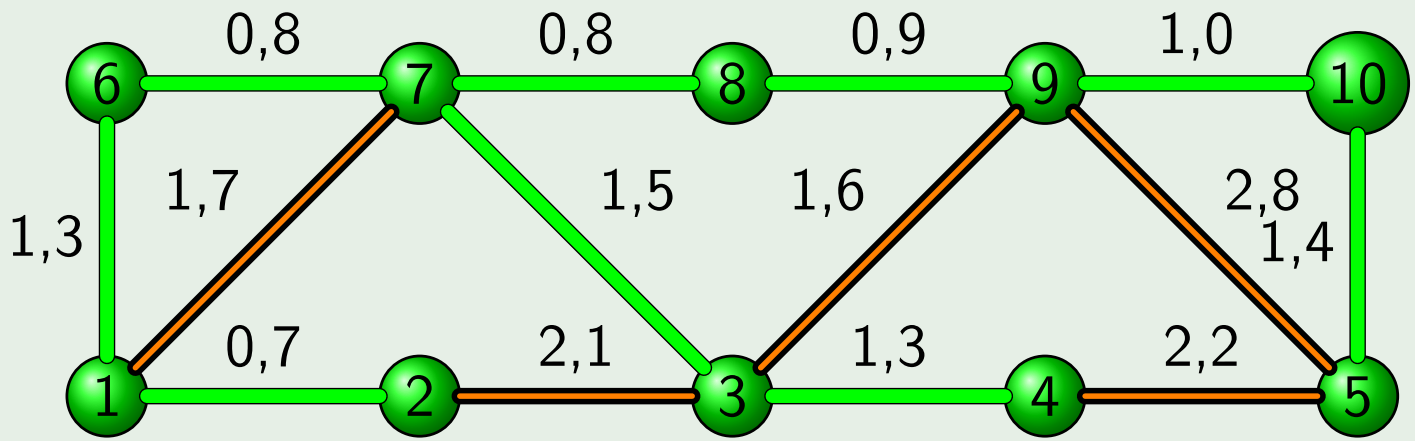
Vektede grafer

Eksempel (Fortsatt)



Vektede grafer

Eksempel (Fortsatt)



Vi får fortsatt det samme treet

Vektede grafer

- 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".

Vektede grafer

- 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.

Vektede grafer

- 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 finnes effektive algoritmer for dette også.

Vektede grafer

- 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 finnes effektive algoritmer for dette også.
- Vi skal gi en uformell beskrivelse av Dijkstras algoritme og vise hvordan den virker på eksemplet vårt.

Vektede grafer

- 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 finnes 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.

Vektede grafer

- 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 finnes 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".

Vektede grafer

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

Vektede grafer

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.

Vektede grafer

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 $i \leq n$ konstruerer vi et tre T_i med i noder og $i-1$ kanter fra G.

Vektede grafer

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 $i \leq n$ 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:

Vektede grafer

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 $i \leq n$ 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 en ny node til v_1 via e og T_i på denne måten.

Vektede grafer

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 $i \leq n$ 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 en ny node til v_1 via e og T_i på denne måten.
2. Finnes det flere like gode alternativer, velg den v'en og deretter den e'en som står først i listene.

Vektede grafer

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 $i \leq n$ 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 en ny node til v_1 via e og T_i på denne måten.
2. Finnes 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.

Vektede grafer

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

Vektede grafer

- 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.

Vektede grafer

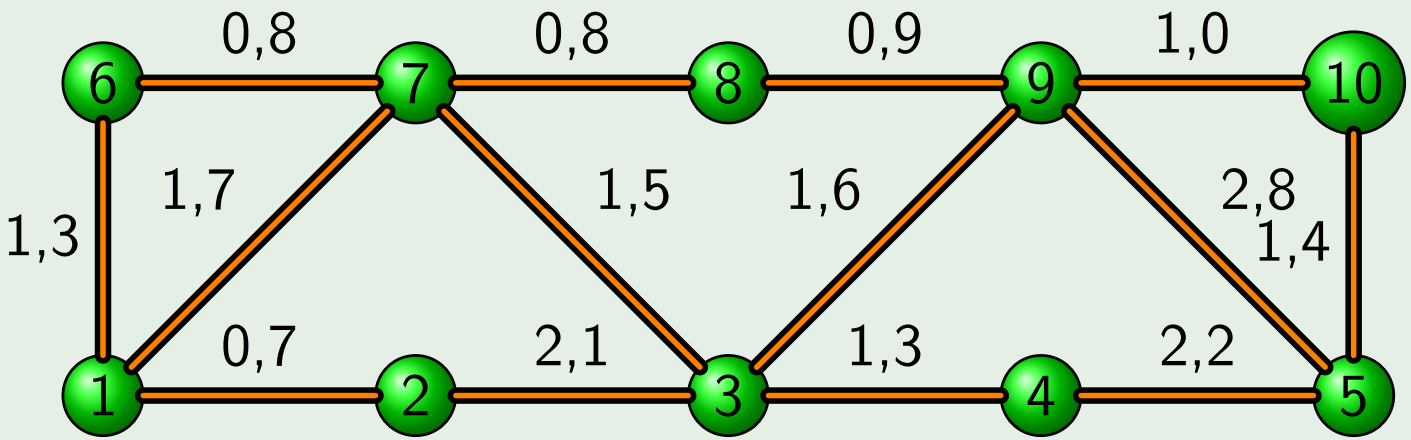
- Dijkstras algoritme er beskrevet i form av en pseudokode i boka.
- Det er meningen at dere skal kunne finne det utspennende treet som gir de korteste stiene fra provinsen til sentrum når dere har fått gitt en vektet, sammenhengende graf.
- La oss se på eksemplet vårt en gang til.

Vektede grafer

Eksempel

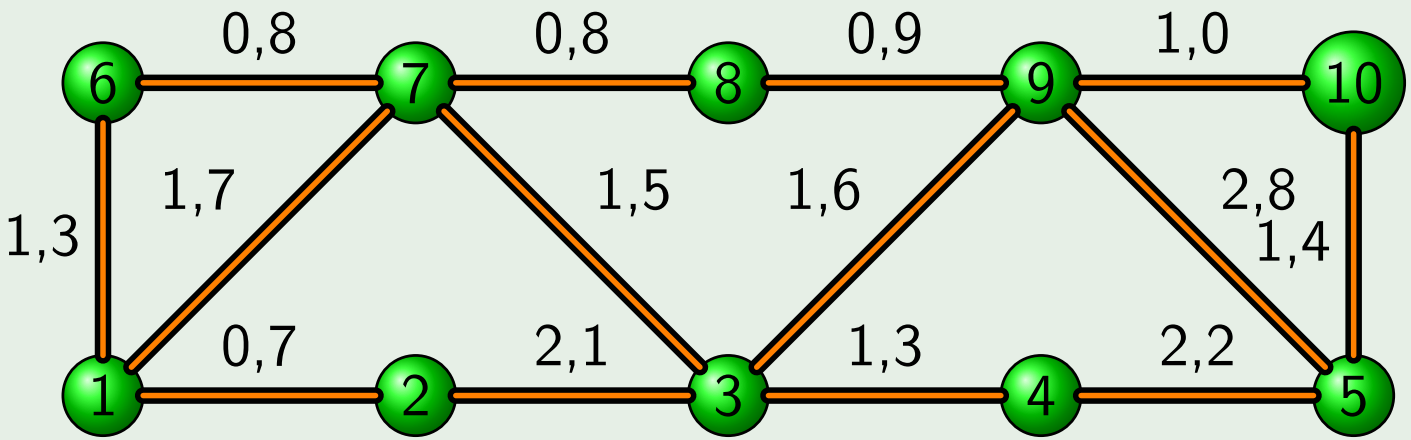
Vektede grafer

Eksempel



Vektede grafer

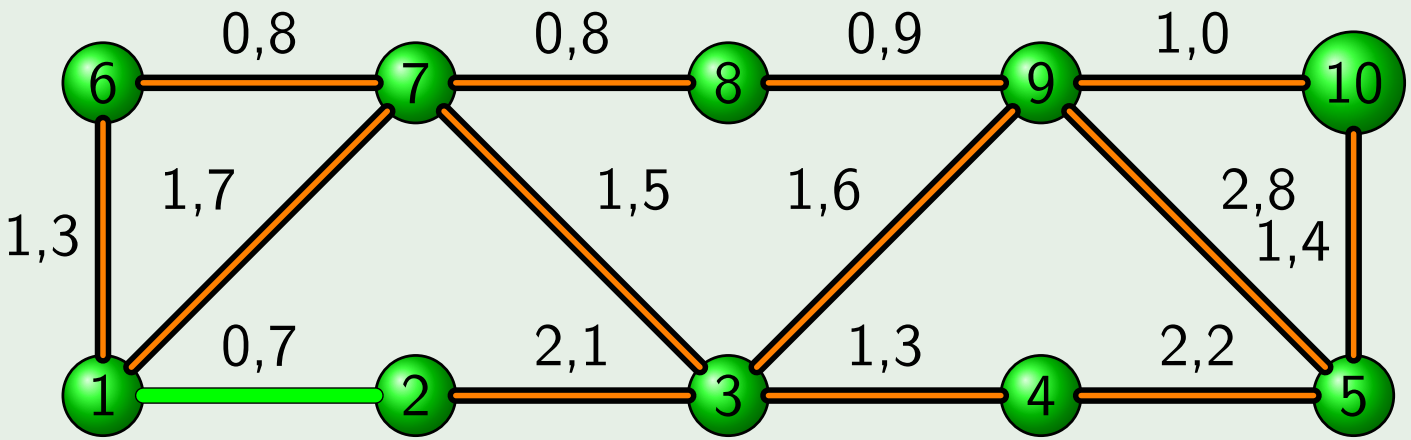
Eksempel



Vi starter i Node 1

Vektede grafer

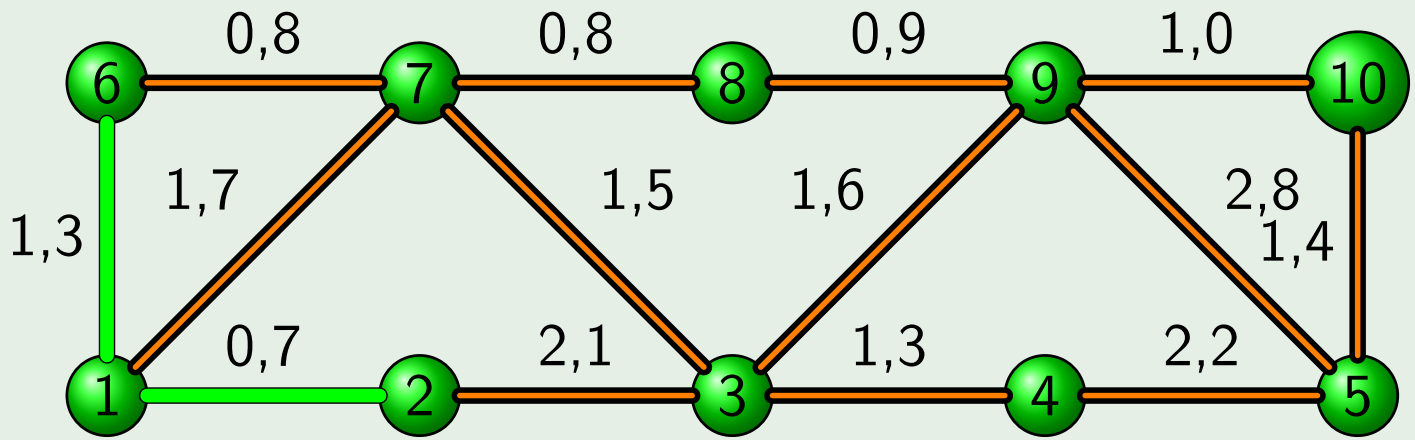
Eksempel



Vi starter i Node 1

Vektede grafer

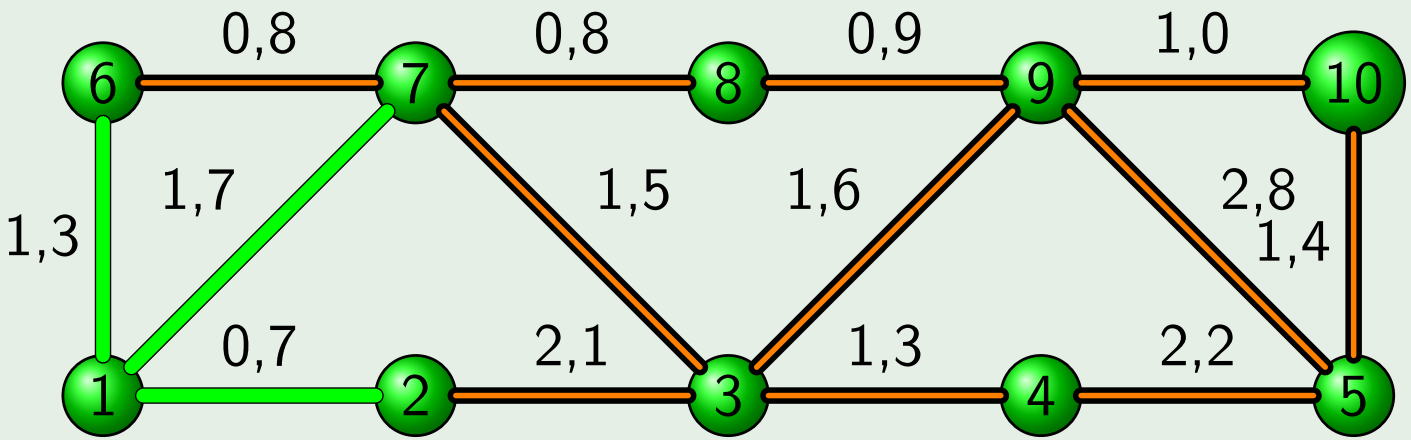
Eksempel



Vi starter i Node 1

Vektede grafer

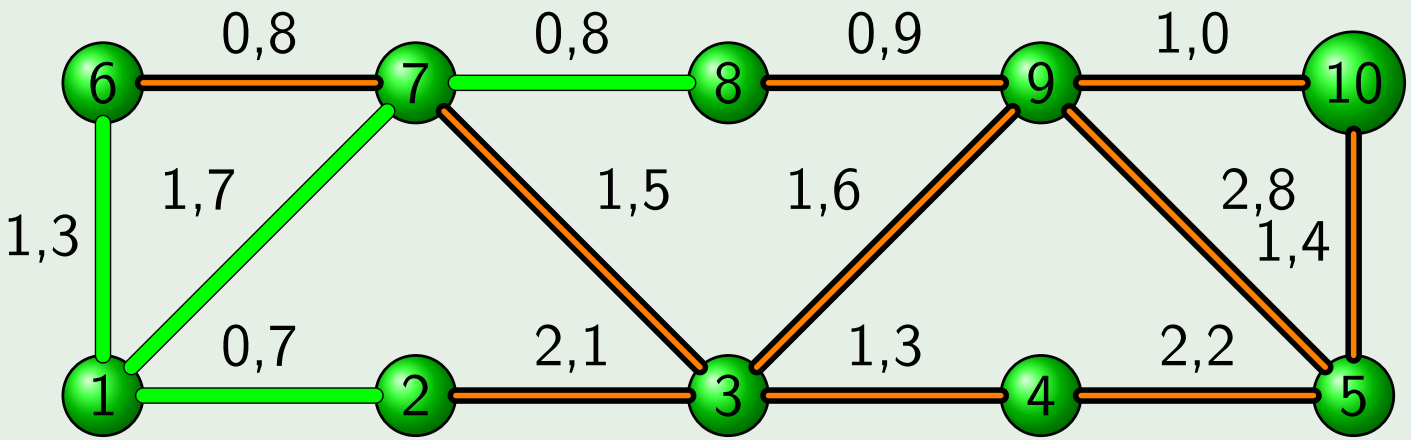
Eksempel



Vi starter i Node 1

Vektede grafer

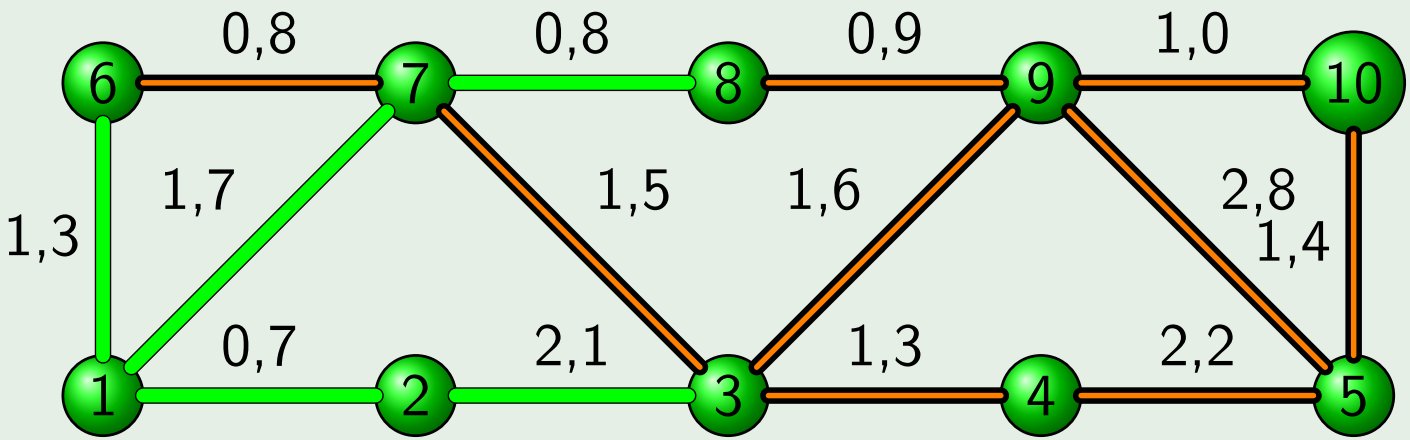
Eksempel



Vi starter i Node 1

Vektede grafer

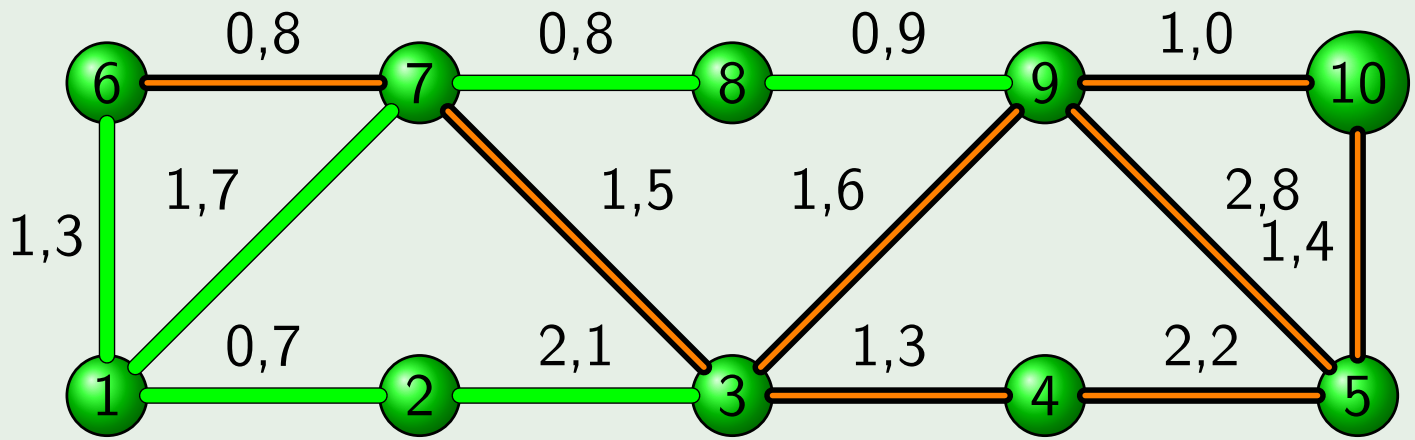
Eksempel



Vi starter i Node 1

Vektede grafer

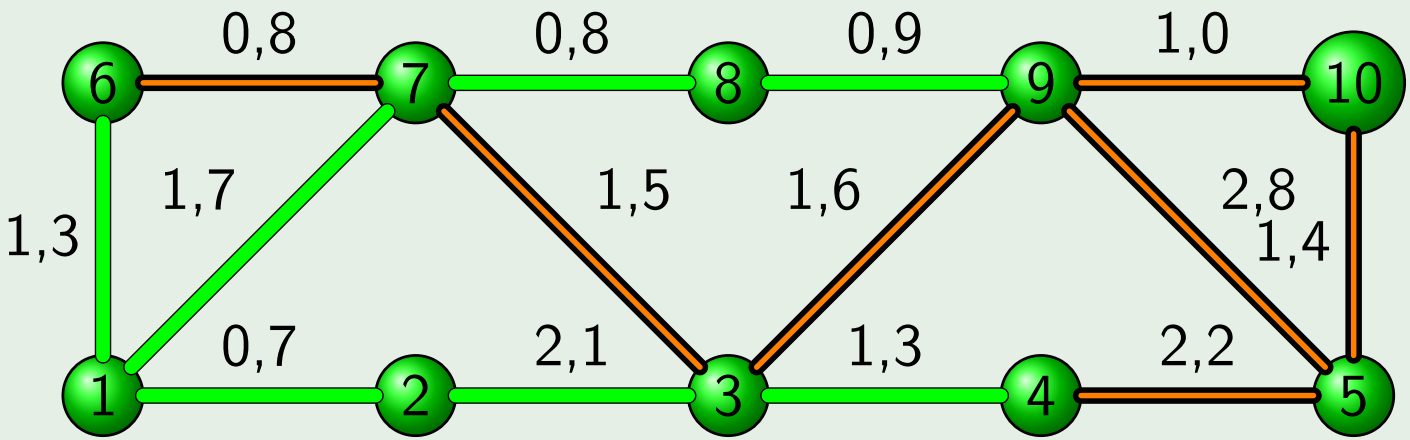
Eksempel



Vi starter i Node 1

Vektede grafer

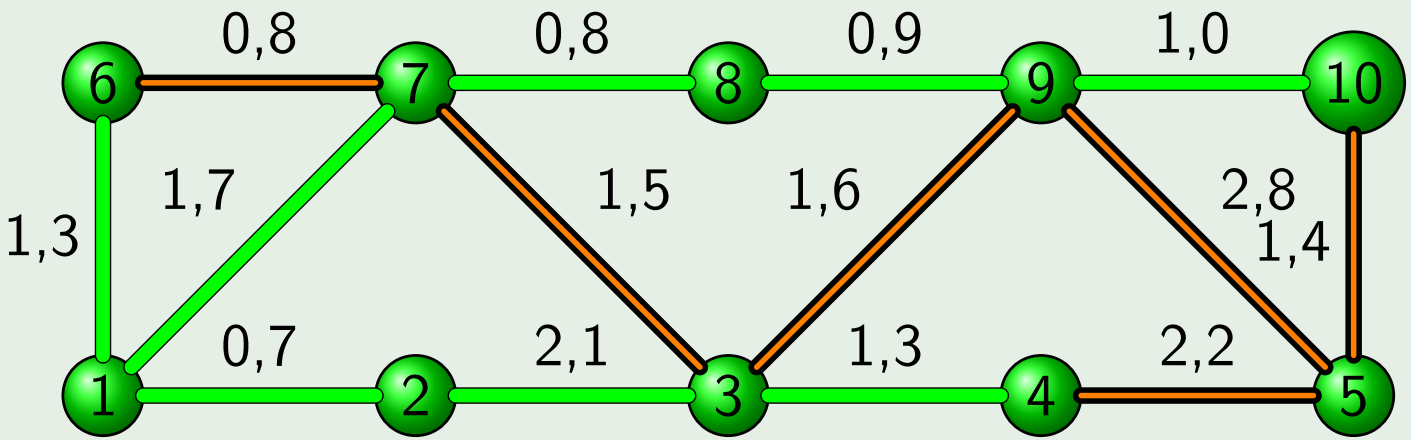
Eksempel



Vi starter i Node 1

Vektede grafer

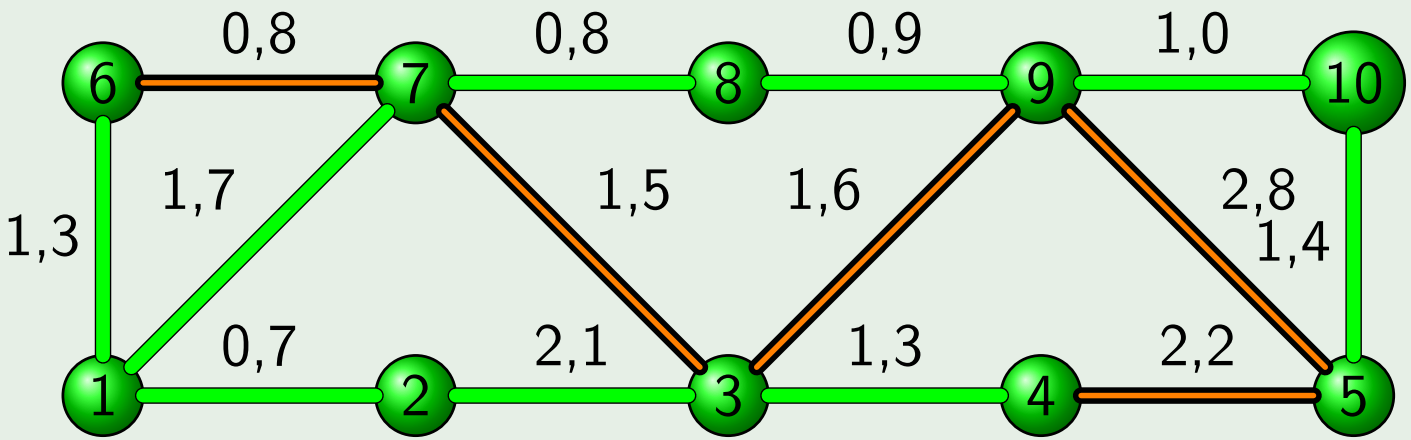
Eksempel



Vi starter i Node 1

Vektede grafer

Eksempel



Vi starter i Node 1

Vi får et nytt tre.

Vektede grafer

- 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.

Vektede grafer

- 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.

Vektede grafer

- 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.

Vektede grafer

- 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

- 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.

Vektede grafer

- 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 finnes noen kant mellom to noder.

Vektede grafer

- 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 finnes 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.

Vektede grafer

- 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 finnes 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 ett eksempel.

Vektede grafer

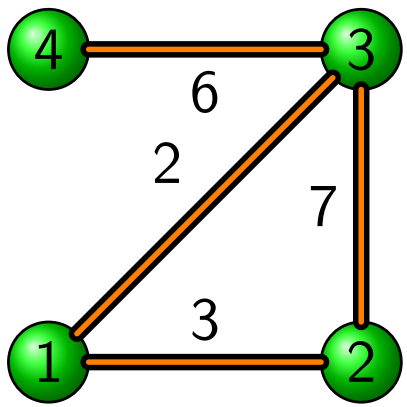
Vektede grafer

Vektede grafer

En vektet graf:

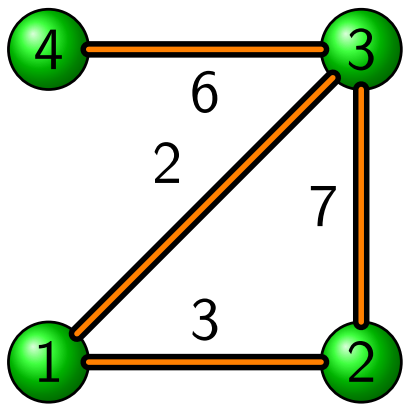
Vektede grafer

En vektet graf:



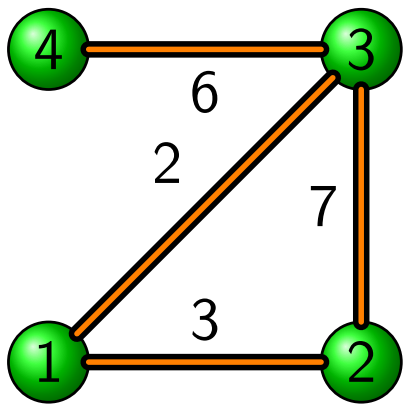
Vektede grafer

En vektet graf:



Vektede grafer

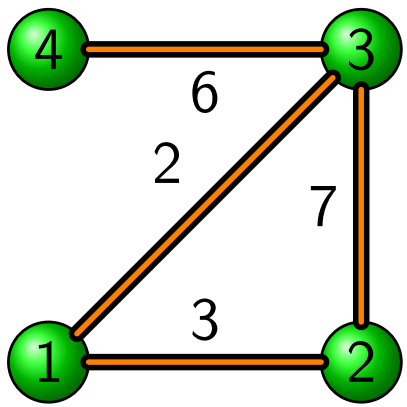
En vektet graf:



Matriserepresentasjonen:

Vektede grafer

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

- 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

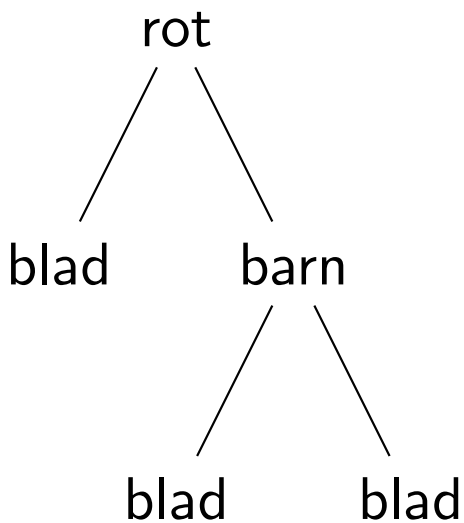
- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.
- For mange anvendelser av teorien for trær, er det nyttig å kunne betrakte den ene noden som roten til treet, den noden alt annet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

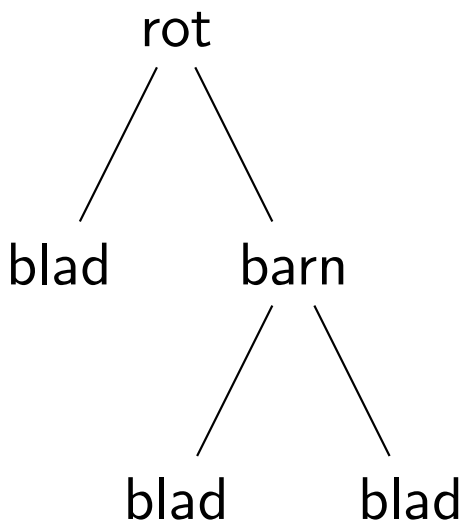
- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.
- For mange anvendelser av teorien for trær, er det nyttig å kunne betrakte den ene noden som roten til treet, den noden alt annet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

Trær med rot

Trær med rot

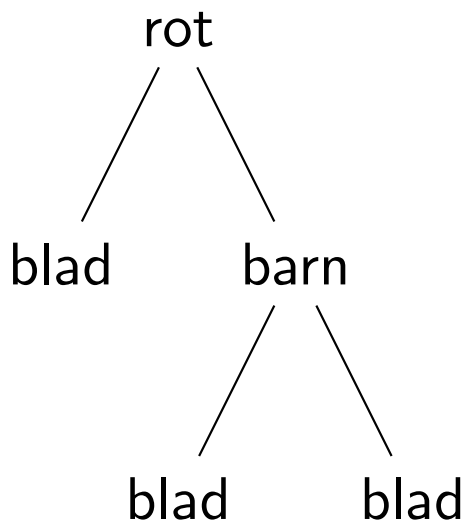


Trær med rot



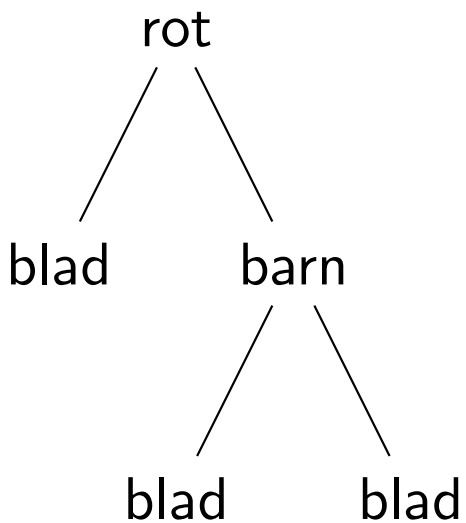
- Roten tegnes øverst, og treet "vokser" nedover.

Trær med rot



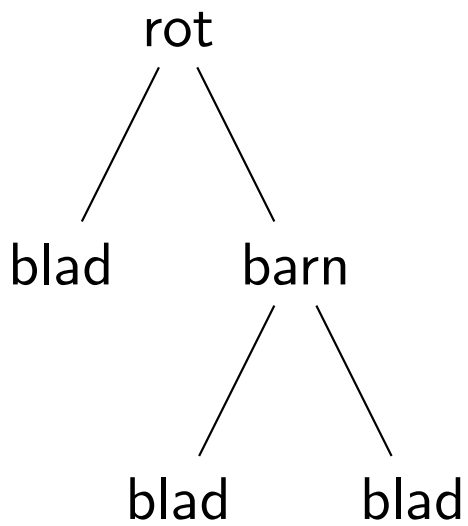
- Roten tegnes øverst, og treet "vokser" nedover.
- Nodene ligger i "lag" avhengig av avstanden til roten.

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

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

Trær med rot

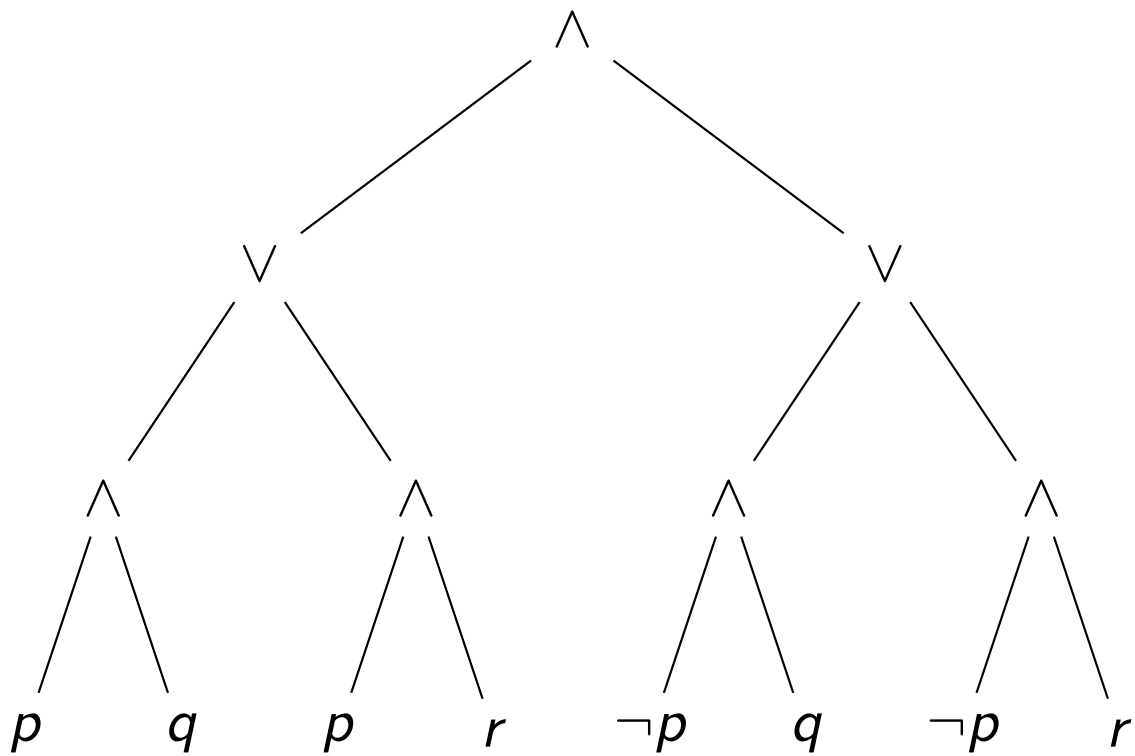
- Vi kan bruke trær til å gi en grafisk fremstilling av en utsagnslogisk formel.
- La $A = ((p \wedge q) \vee (p \wedge r)) \wedge ((\neg p \wedge q) \vee (\neg p \wedge r))$.

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

- 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 gramatikk-analyse 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 gramatikk-analyse av setninger i naturlige språk.
- Da setter man opp et tre som beskriver hvordan en setning 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 gramatikk-analyse av setninger i naturlige språk.
- Da setter man opp et tre som beskriver hvordan en setning 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

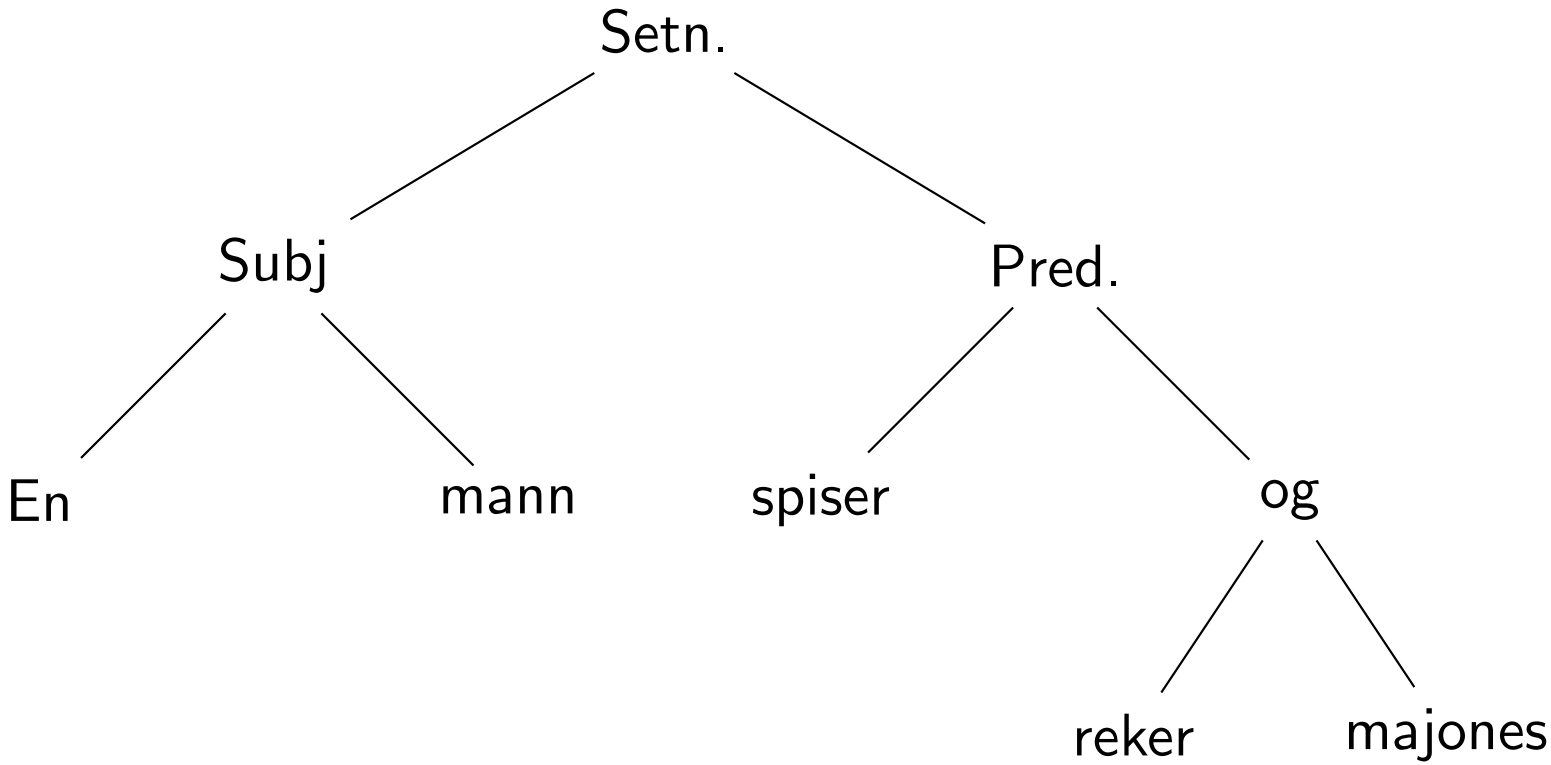
- *En mann spiser reker og majones*

Trær med rot

- *En mann spiser reker og majones*
- Denne setningen er bygget opp slik:

Trær med rot

- *En mann spiser reker og majones*
- Denne setningen er bygget opp slik:



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, d tilhører samme art.

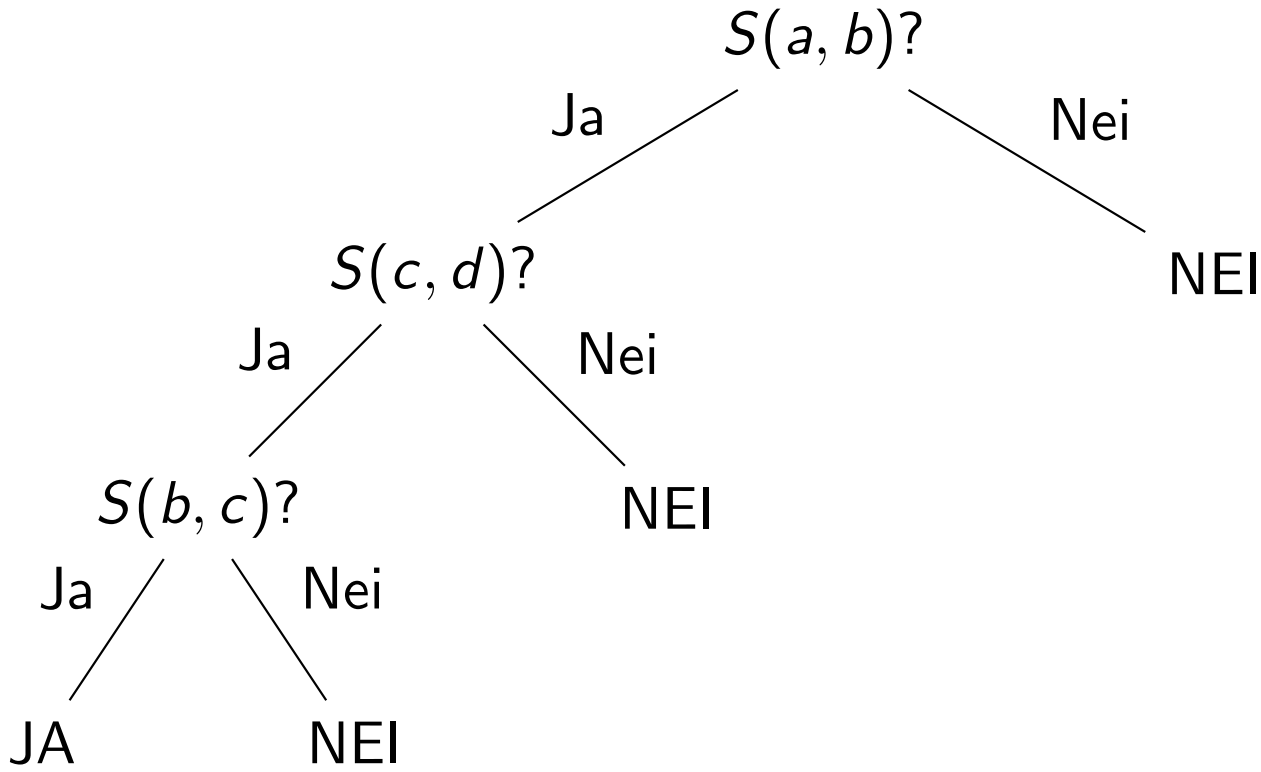
Trær med rot

- 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, 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, d tilhører samme art.
- Vi skriver $S(x, y)$ for at x og y tilhører samme art.
- Det å tihøre samme art er en ekvivalensrelasjon, og korrekthet av programmet bygger bare på det faktum (på samme måte som korrekthet av eksemplet i boka bygger på at vi har en total ordning).

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 etterhverandre i S-betydningen.

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 etterhverandre i S-betydningen.
- 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 etterhverandre i S-betydningen.
- 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.