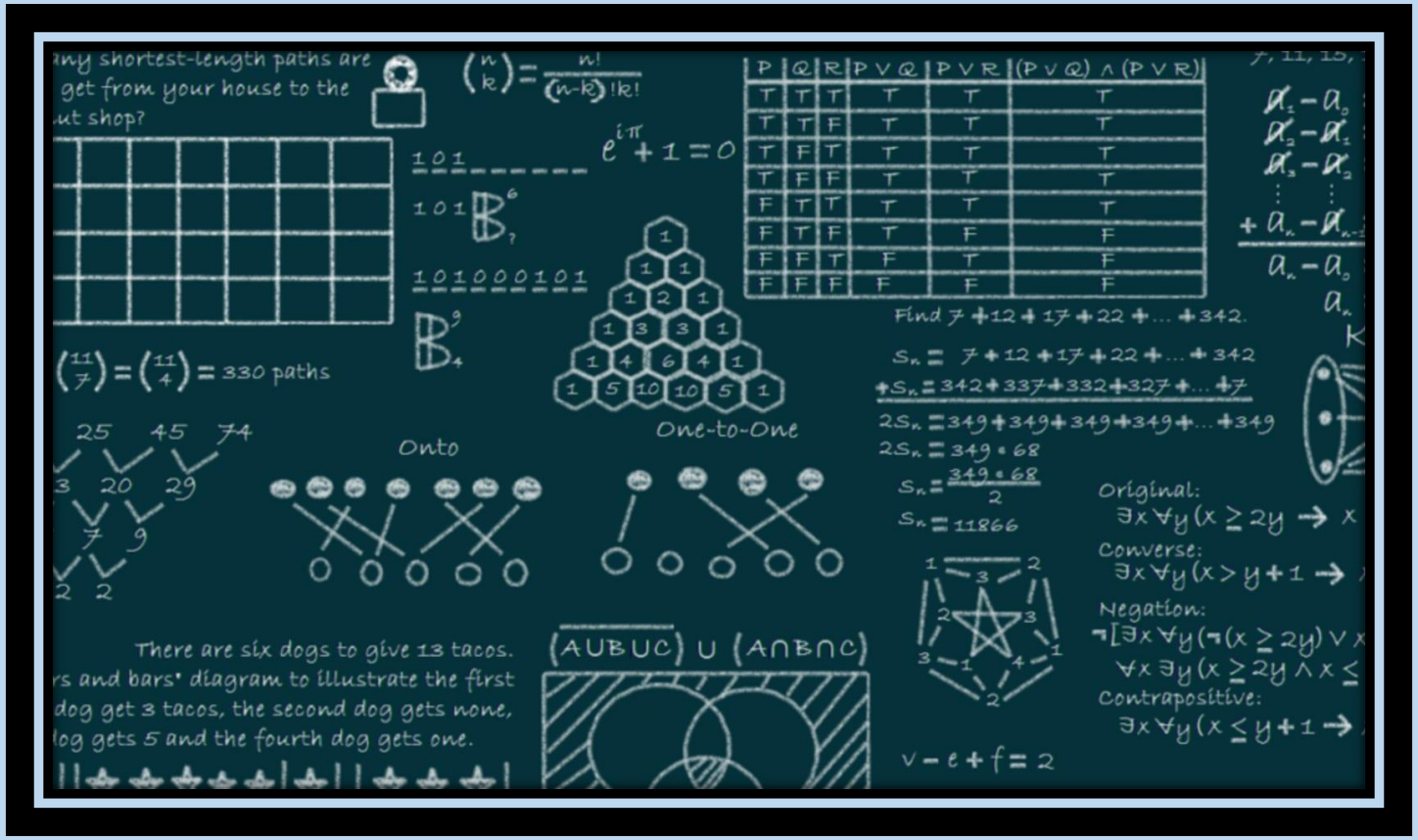


FÖRELÄSNING 7

Datastrukturer och algoritmer
KYH – 2022 HT

Andreas Nilsson Ström

Agenda

- Träd
- Grafer

Repetition

Uppvärmning!

Algoritm: Fibinacci-nummer

- En Fibonacci-sekvens definieras genom att man adderar ett tal med föregående tal för att få nästa tal i en serie.
- Serien börjar med 0 och 1
  - 1 + 1 = 2
  - 1 + 2 = 3
  - 2 + 3 = 5
  - 3 + 5 = 8
  - 5 + 8 = 13
  - Osv.
- Vad är indexet på första talet som har 1000 siffror?
- Källa: ProjectEuler.net problem 25

Träd

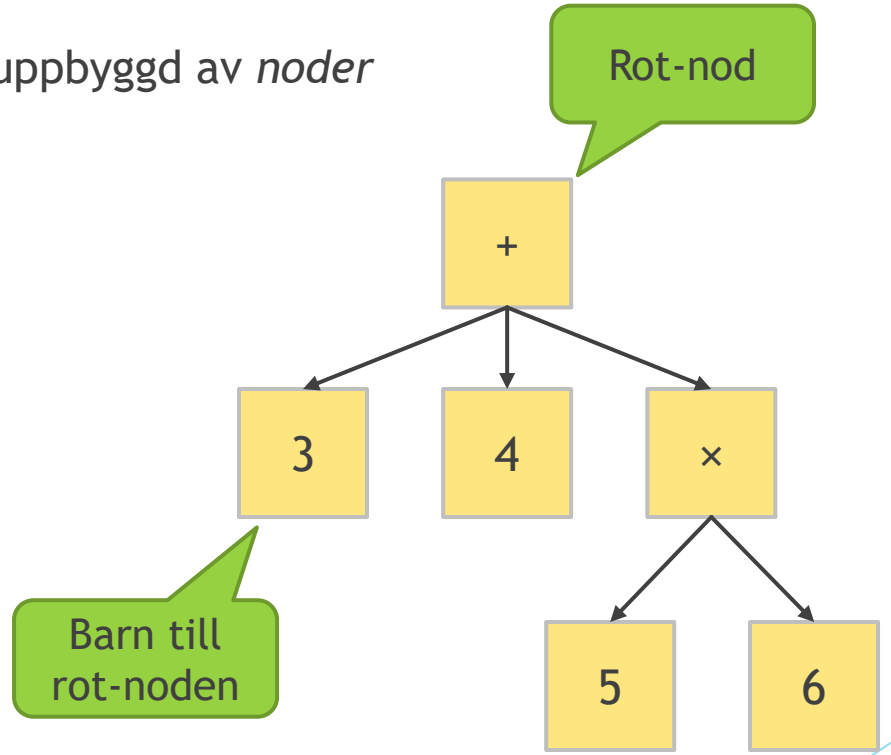
Träd

Ett *träd* är en hierarkisk datastruktur som är uppbyggd av *noder*

- I toppen har vi vår *rotnod*
- Varje nod kan ha *barn-noder*
- Som i sin tur kan ha *barn-noder*
- … Och så vidare
- Varje nod utom rot-noden har en *förälder*

Exempel: Ett uttrycks-träd

- Nod: Ett aritmetiskt uttryck
- Barn: Operationens argument
- Träder till höger betyder
  3 + 4 + (5 × 6)

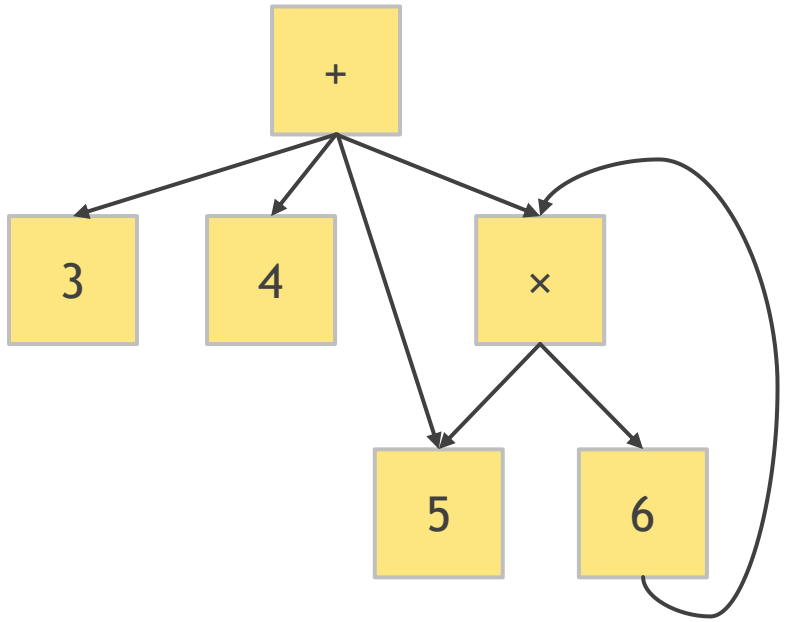


Träd

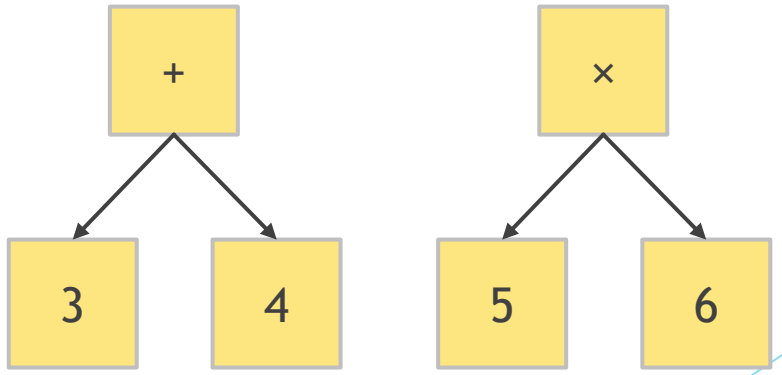
För att en samling noder ska vara ett träd:

- Varje nod måste ha **en unik förälder**, utom roten som **inte har en förälder**
- Det måste finnas exakt en rot-nod – alla andra noder nås från roten genom att följa länkar från förälder till barn
- (Dessa regler förbjuder också **cykler**, som × -> 6 -> × nedan)

Inte ett träd: 5 och × har två föräldrar



Inte ett träd: flera rot-noder



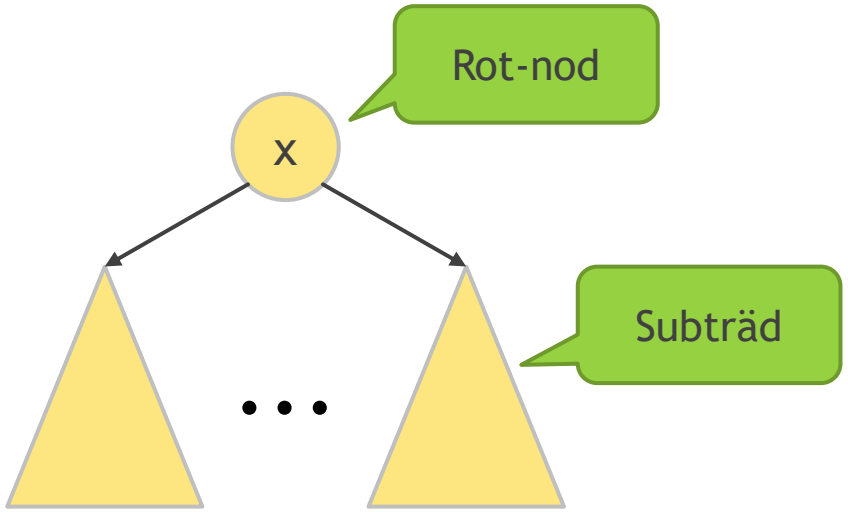
Rekursiva träd

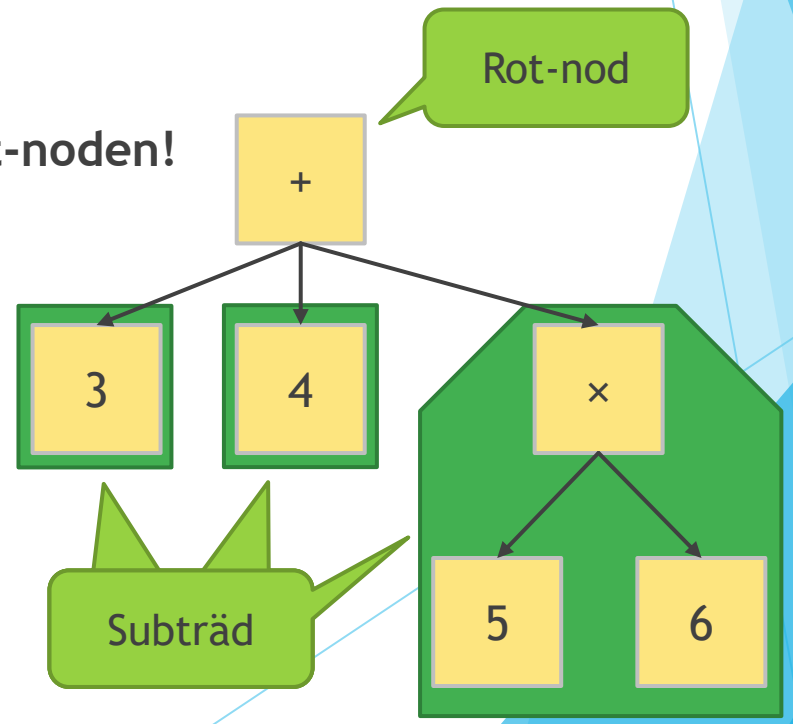
Träd kan definieras rekursivt:

- Eftersom länkar bara får gå nedåt och barn inte får ha fler än en förälder:
- Man kan se en barn-nod som början på ett nytt träd
- Träd består en rot-nod som länkar till subträd

Användbart för programmering!

För att lagra ett träd så behöver vi bara komma ihåg rot-noden!

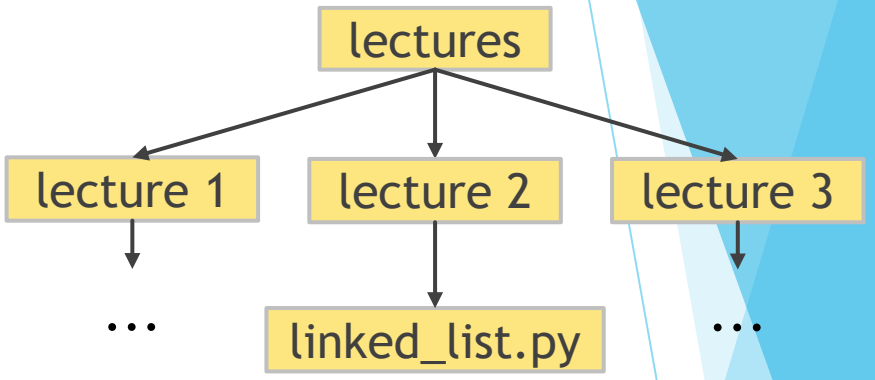




Programmera med träd

Exempel: Katalogstruktur för denna kursen

Hur kan vi representera detta träd med kod?



Data: Varje nod är ett objekt, med dess subträd som referenser. Vi representerar trädet med rot-noden.

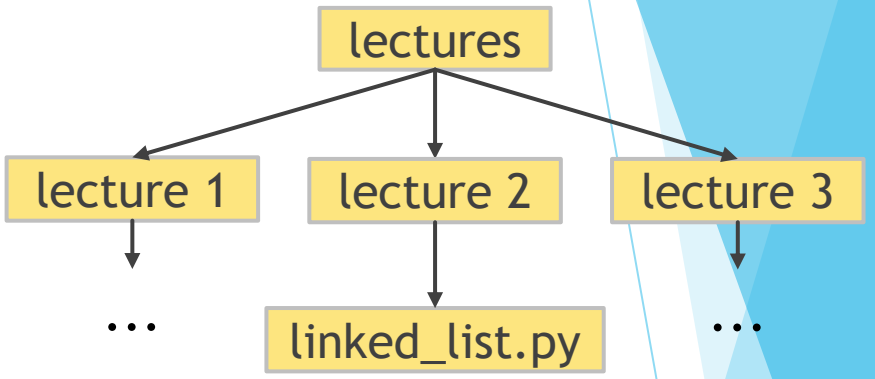
```
class Node:
    def __init__(self, name: str):
        self.name = name  # Namn på fil / mapp

        self.children = []  # Länkar till noder
```

Kod: För att programmera träd-algoritmer, använd rekursion!
Exempel: Räkna antalet noder

```
def size(self) -> int:
    # Börja med 1 och lägg på 1 för varje barn
    s = 1

    for child in self.children:
        s += child.size()

    return s
```

Programmera med träd

```python
class Node:
    def __init__(self, name: str):
        self.name = name  # Namn på fil / mapp

        self.children = []  # Länkar till noder
```



Pre-order traversal. Rekursiva algoritmer som behandlar noder **innan** de besöker dess barn-noder

```python
def list_nodes(self, n=0: int):
    # Lista namnet på alla filer/mappar
    indent = ' ' * n

    print(indent, self.name)

    for child in self.children:
        child.list_nodes(n+2)
```

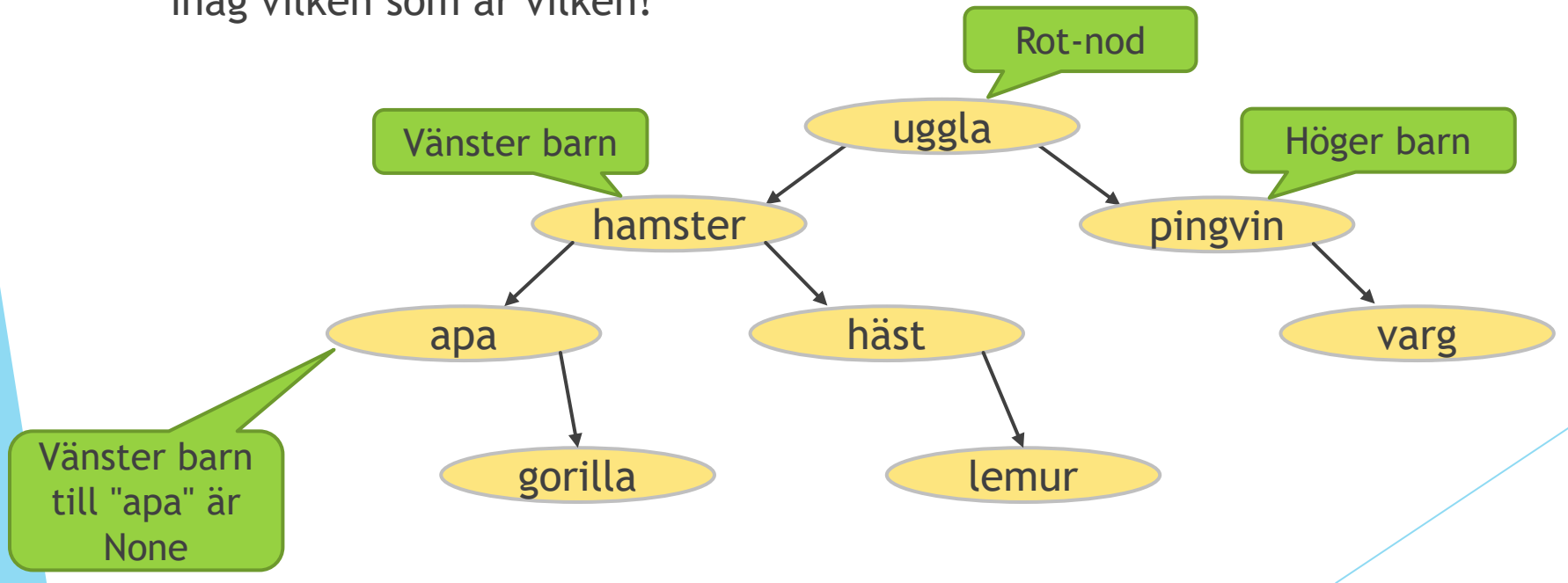
Post-order traversal: Rekursiva algoritmer som behandlar noder **efter** att ha besökt dess barn-noder

```python
def size(self) -> int:
    # Börja med 1 och lägg på 1 för varje barn
    s = 1

    for child in self.children:
        s += child.size()

    return s
```

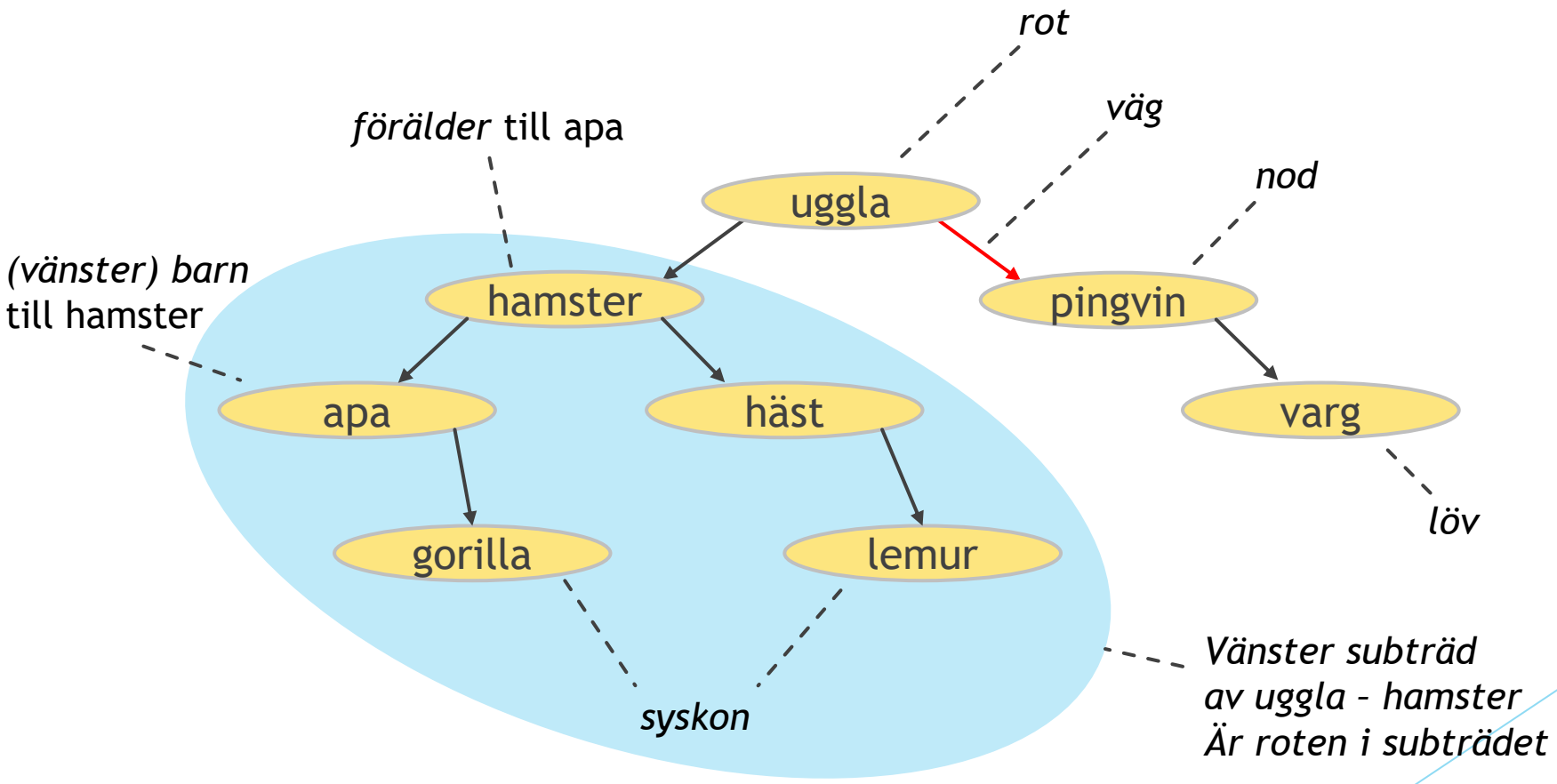
Binära träd

Ett binärt träd är ett träd där varje nod har exakt två barn-noder

- Ett binärt träd kan också vara tomt (None)! Om inte tomt, har root-noden två barn-noder (som själva kan vara None)
- Vi kallar barn-noderna vänster och höger barn (eller subträd) och kommer ihåg vilken som är vilken!



Terminologi för träd (tänk familjetärd, plantor)

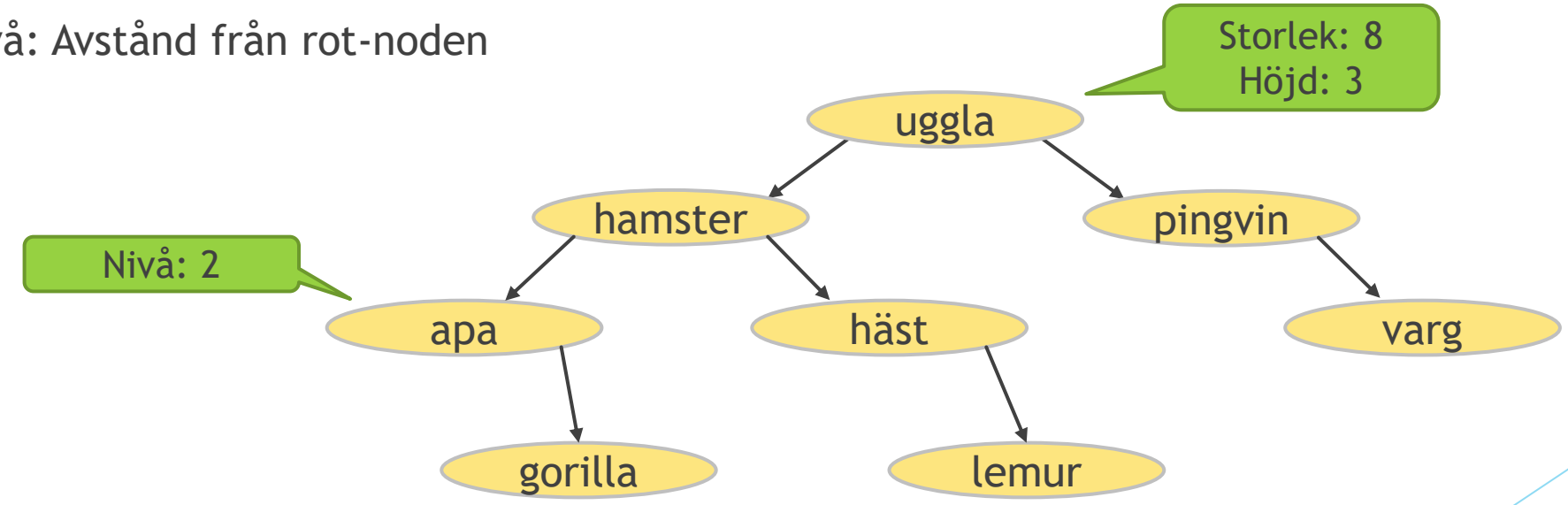


Mer terminologi

Storlek: antal noder i träd eller subträd

Höjd: antal nivåer i trädet

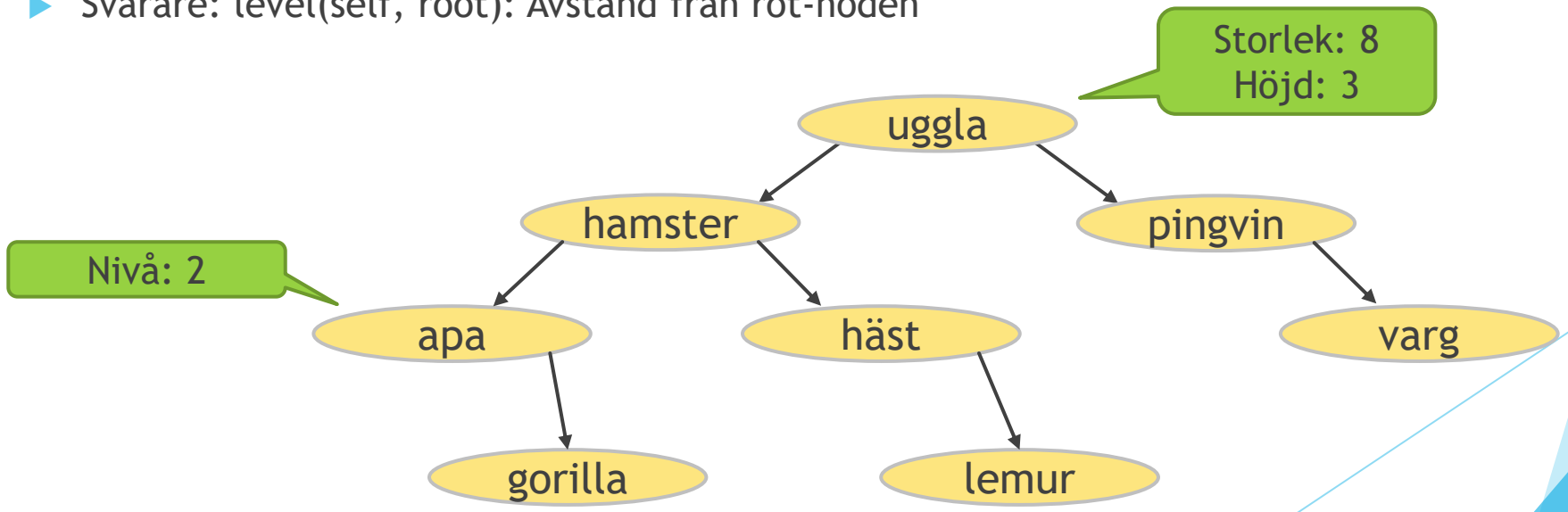
Nivå: Avstånd från rot-noden



Övning! Implementera

Implementera detta träd.

- Klass Node: Har namn, och har left och right som pekar på andra noder.
- Metoder i klassen:
  - Lätt: *size(self)*: antal noder i träd eller subträd
  - Medium: height(self): antal nivåer i trädet
  - Svårare: level(self, root): Avstånd från rot-noden

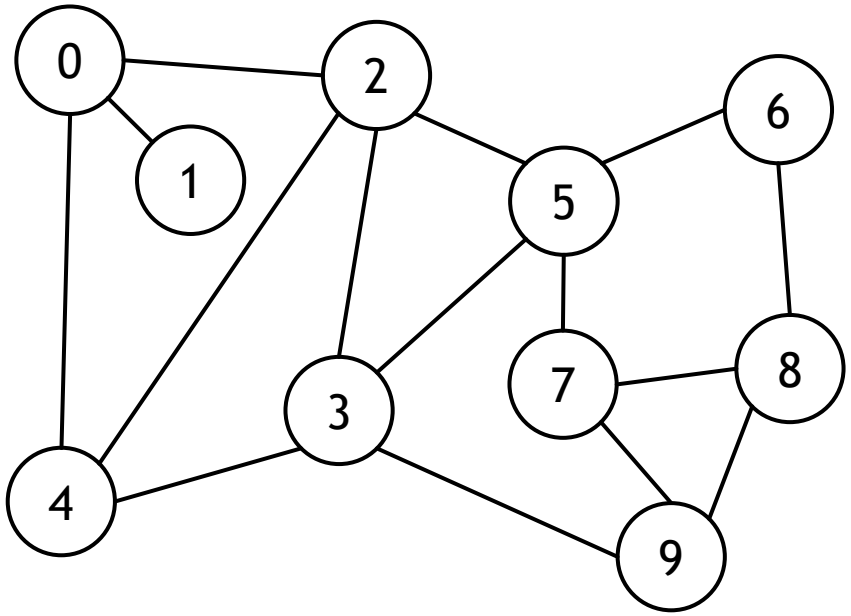


Grafer

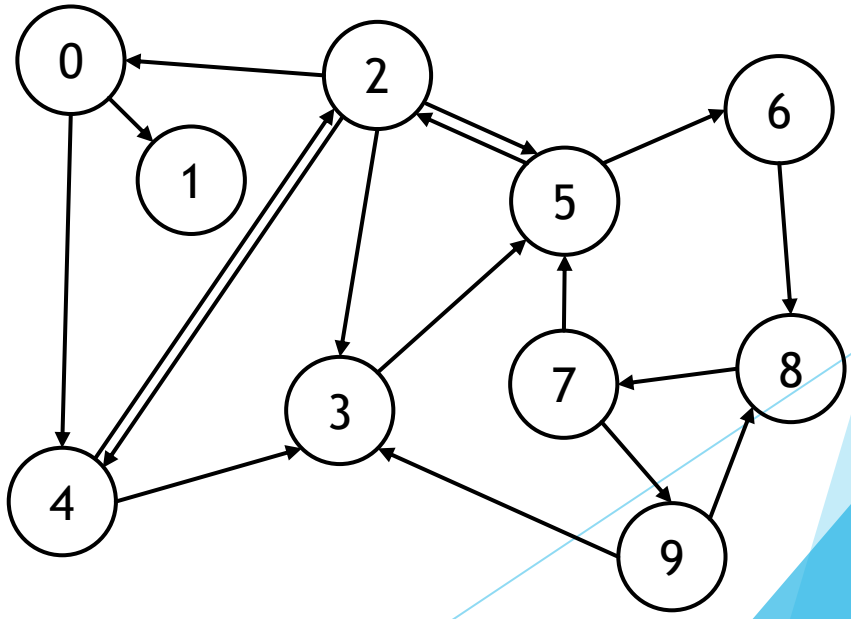
Grafer

- En graf är ett set *noder/hörn* anslutna parvis med *kanter/vägar*
- Det finns *oriktade* grafer och *riktade* grafer

Oriktad graf



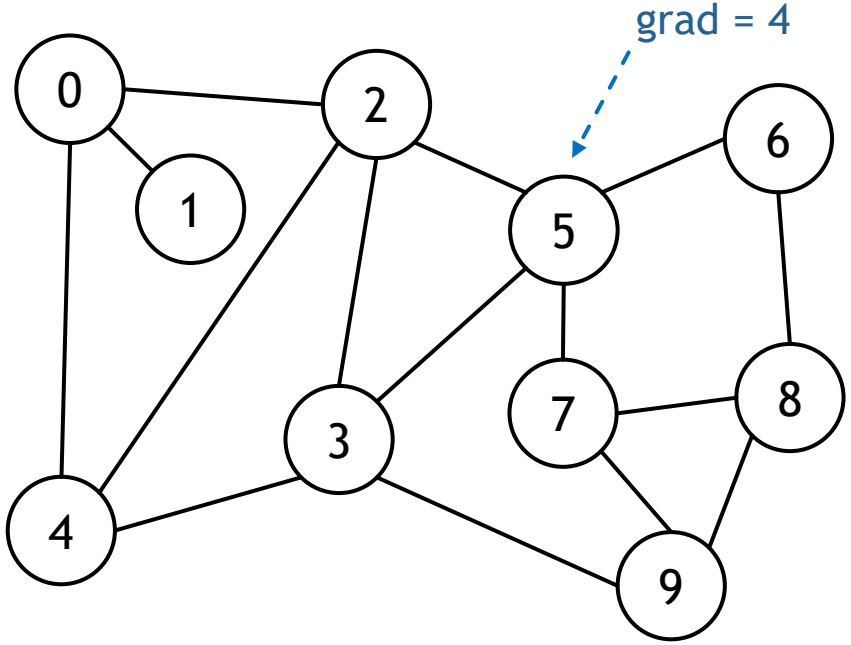
Riktad graf



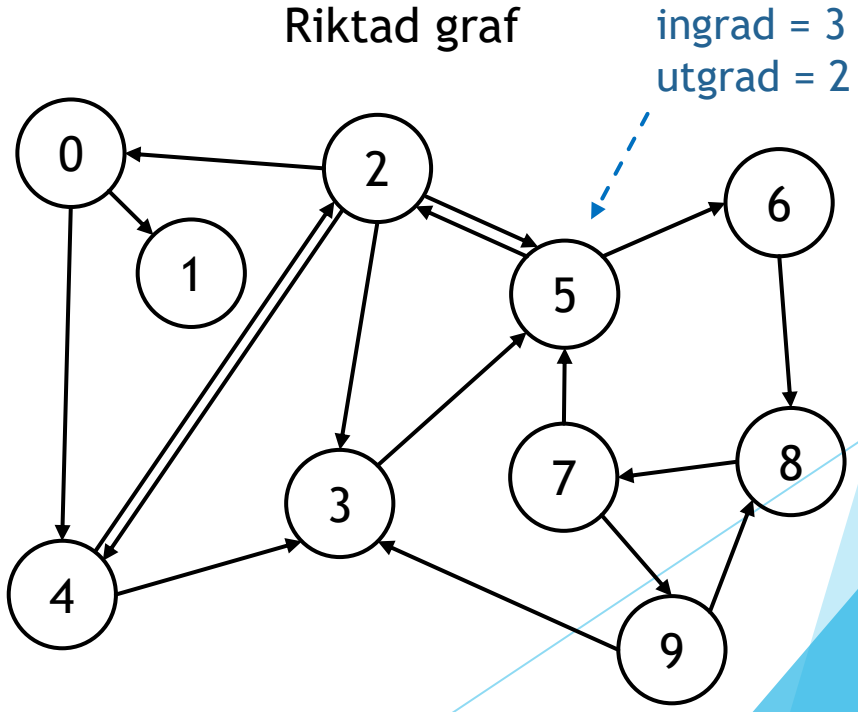
Grafer

- En graf är ett set *noder/hörn* anslutna parvis med *kanter/vägar*
- Det finns *oriktade* grafer och *riktade* grafer

Oriktad graf

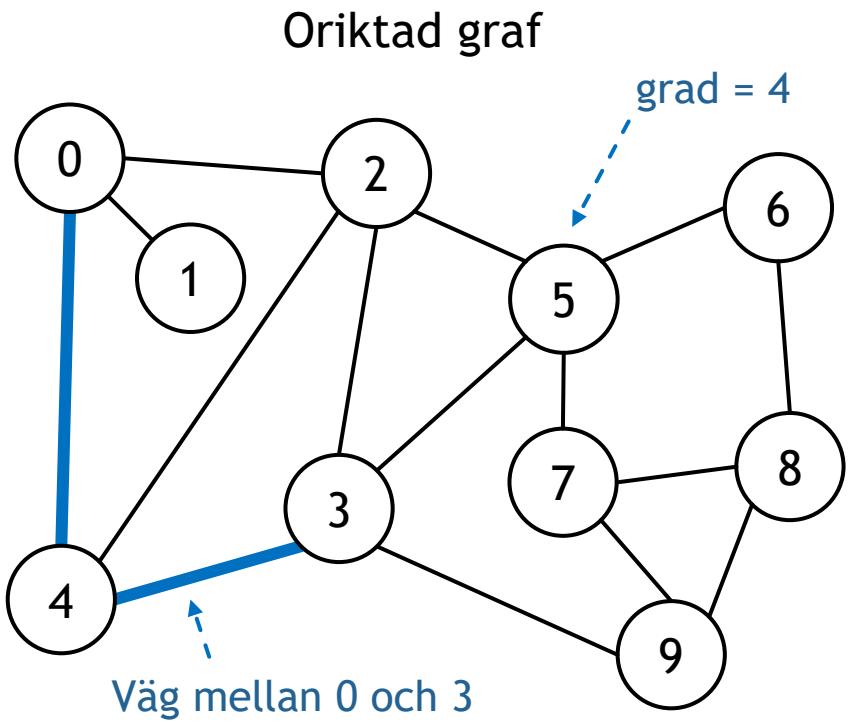


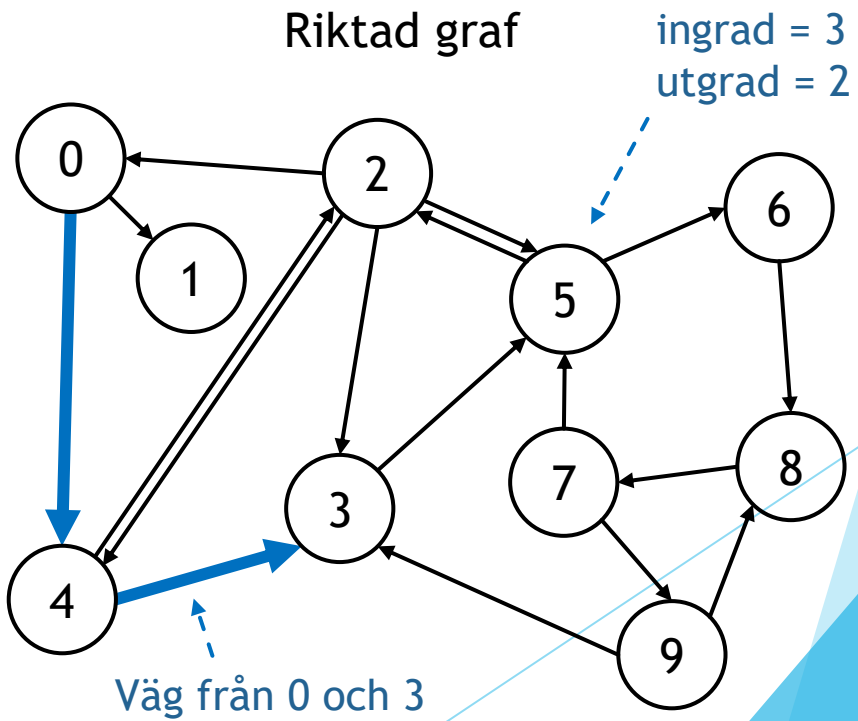
Riktad graf



Grafer

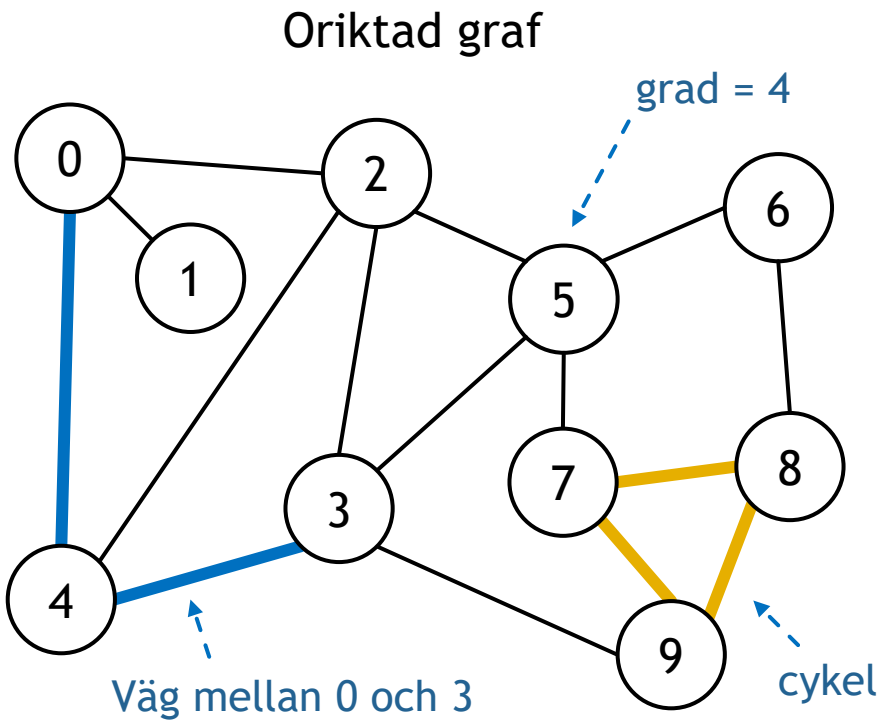
- En graf är ett set *noder/hörn* anslutna parvis med *kanter/vägar*
- Det finns *oriktade* grafer och *riktade* grafer

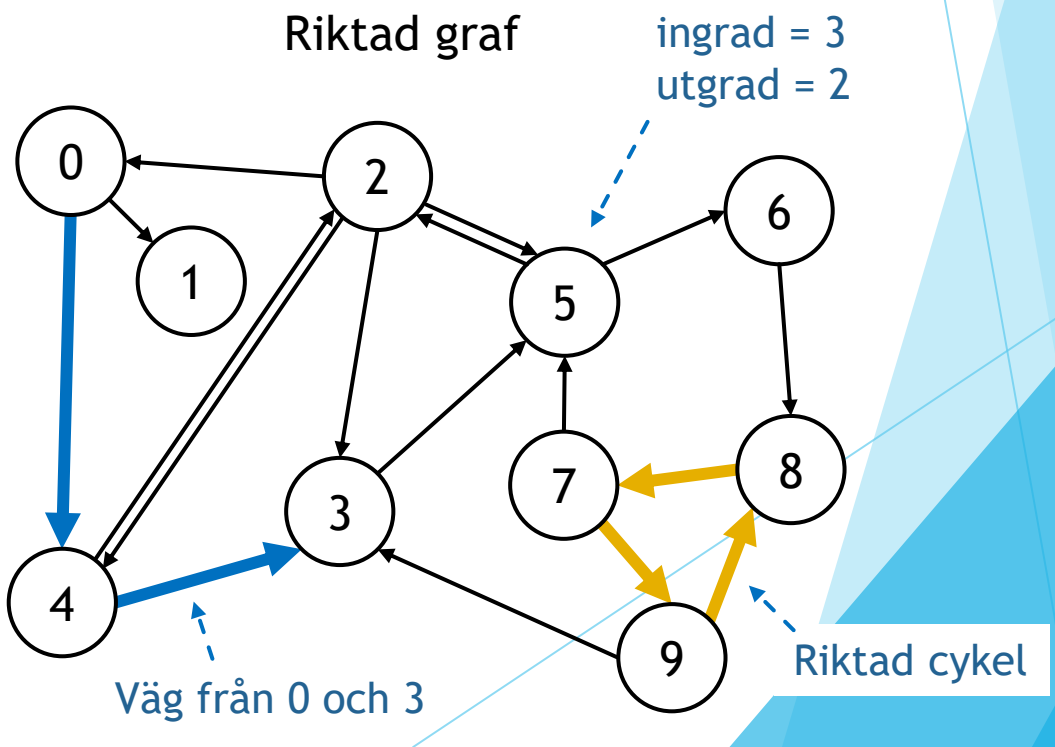




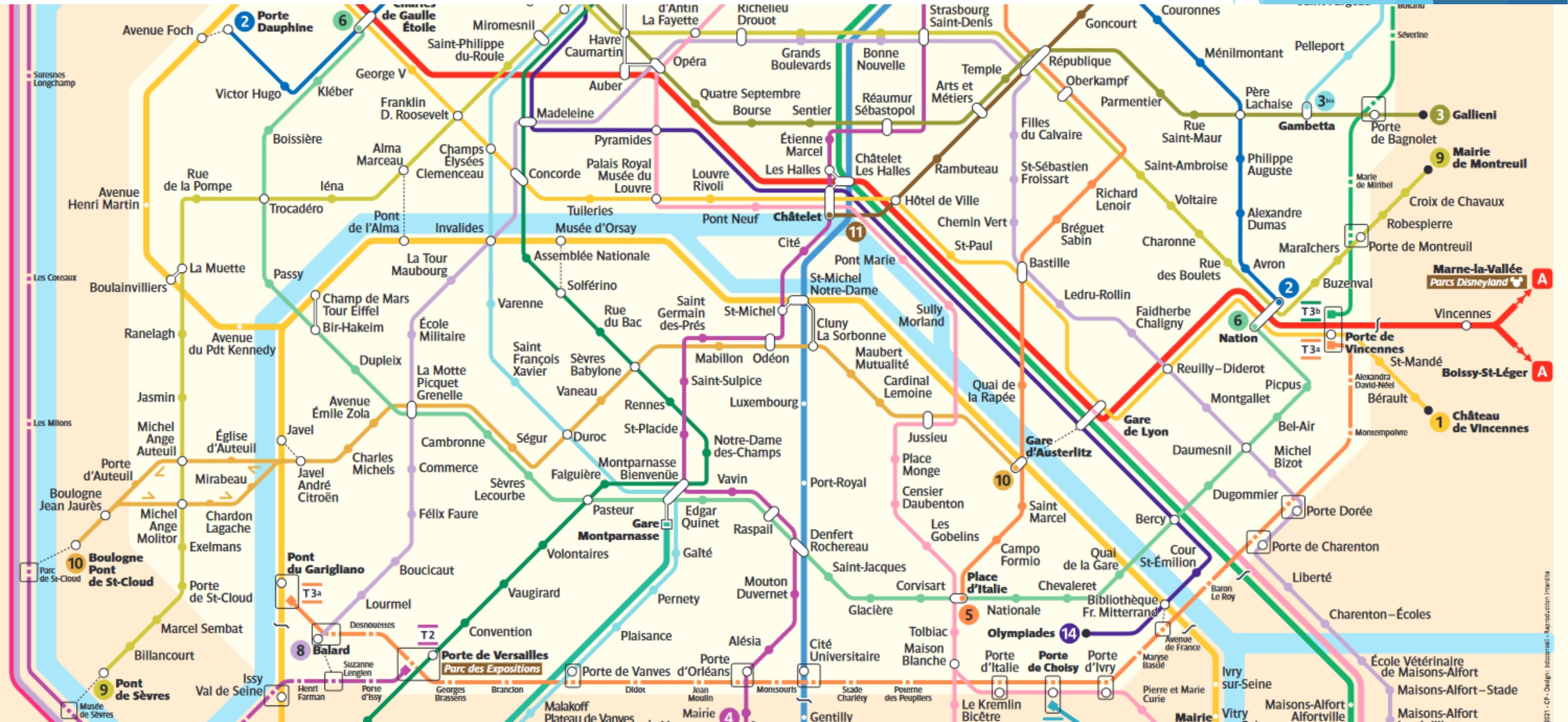
Grafer

- En graf är ett set *noder/hörn* anslutna parvis med *kanter/vägar*
- Det finns *oriktade* grafer och *riktade* grafer



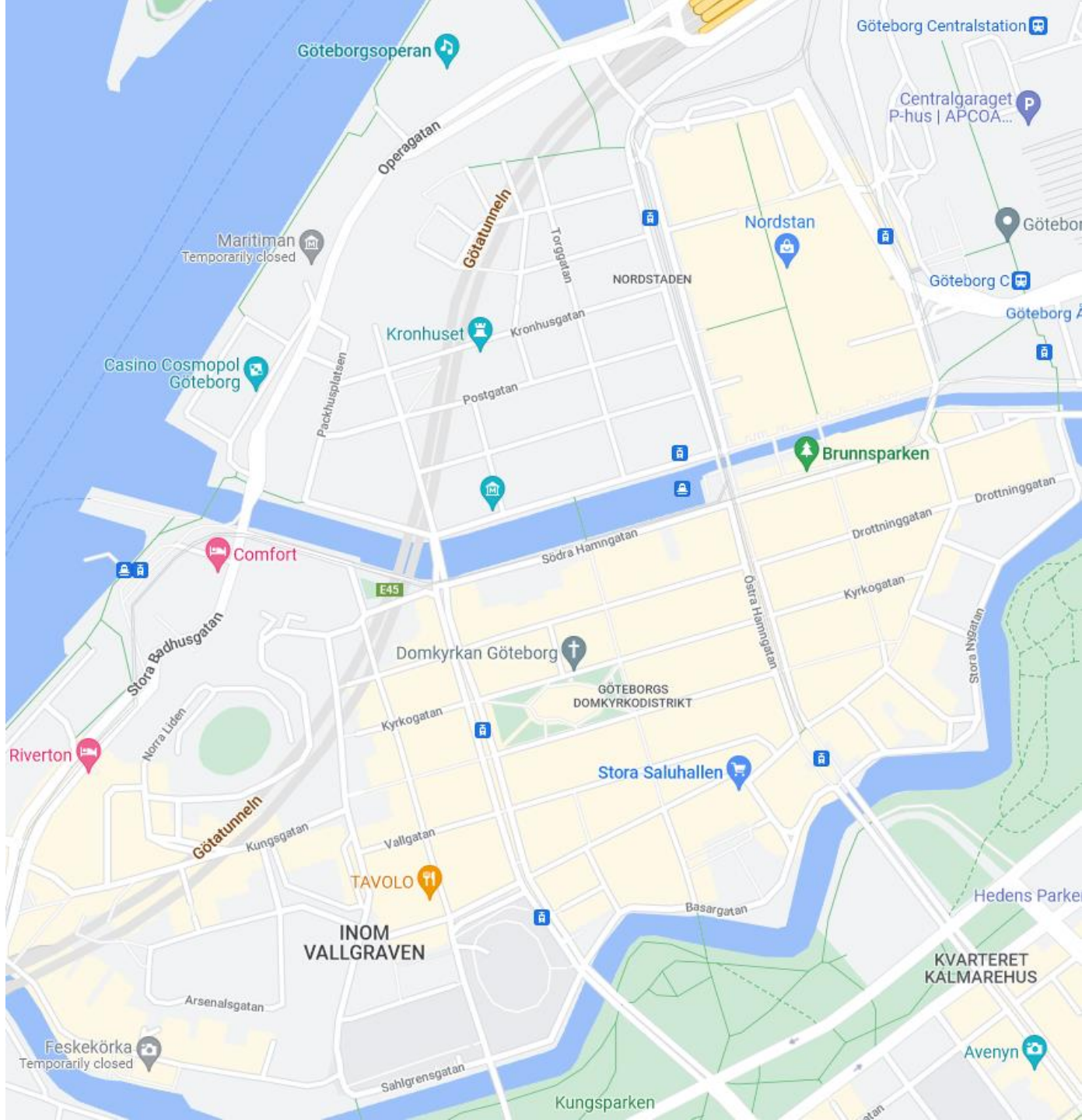


Exempel: Paris Metro

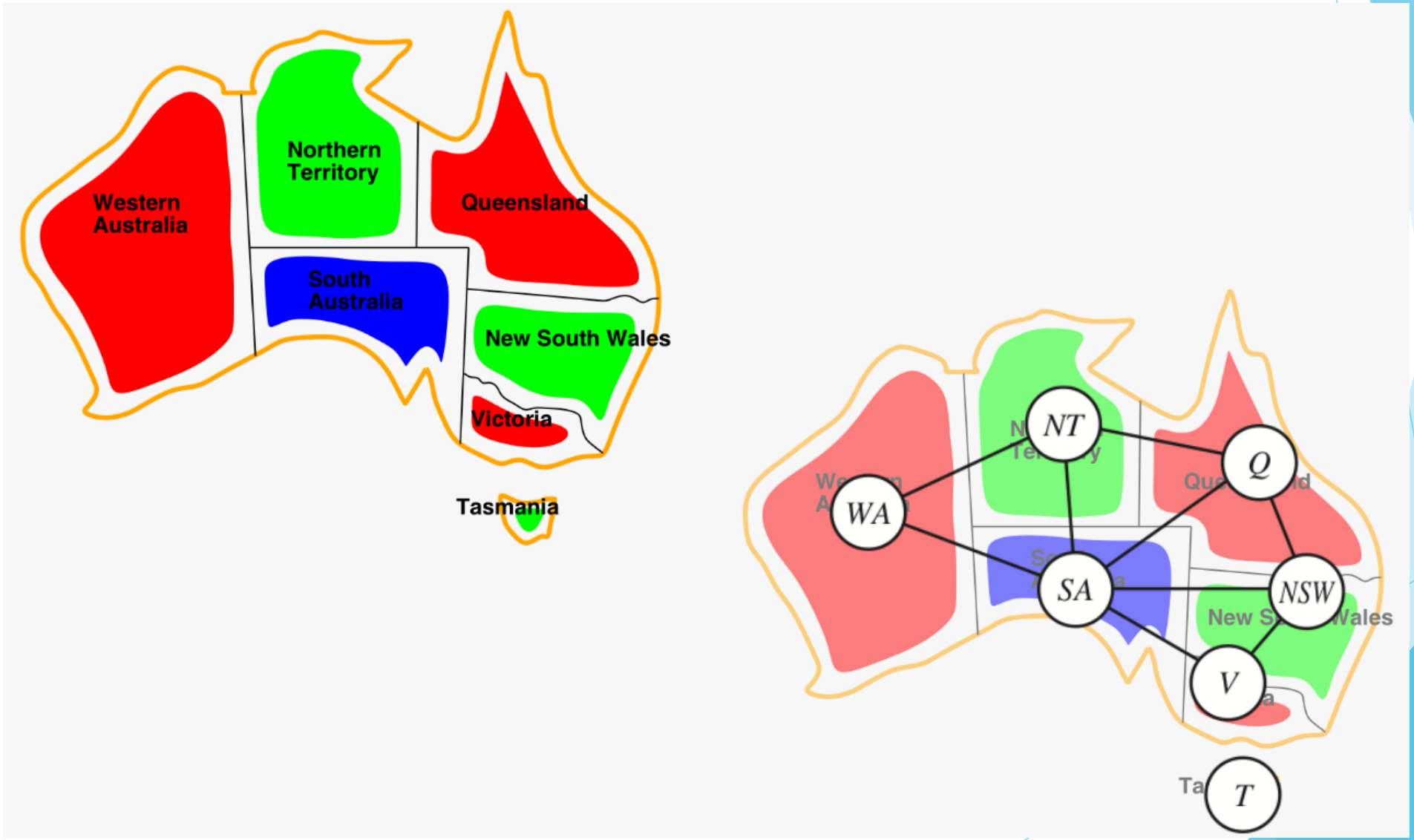


Exempel: Karta

- Korsningar = Noder
- Vägar = Kanter



Exempel: Australien - Gränskarta



Applikationer av graf-teori

directed graph	node	directed edge
transportation	intersection	one-way street
www	web page	hyperlink
food chains	species	predator-prey relationship
WordNet	synset (synonym set)	hypernym
scheduling	task	precedence constraint
financial	bank account	transaction
phone calls	phone number	placed call
chemical reactions	molecules	reaction
infectious disease	person	infection
neural network	neuron	synapse
board game	board position	legal move
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump / branch

Interface för grafer

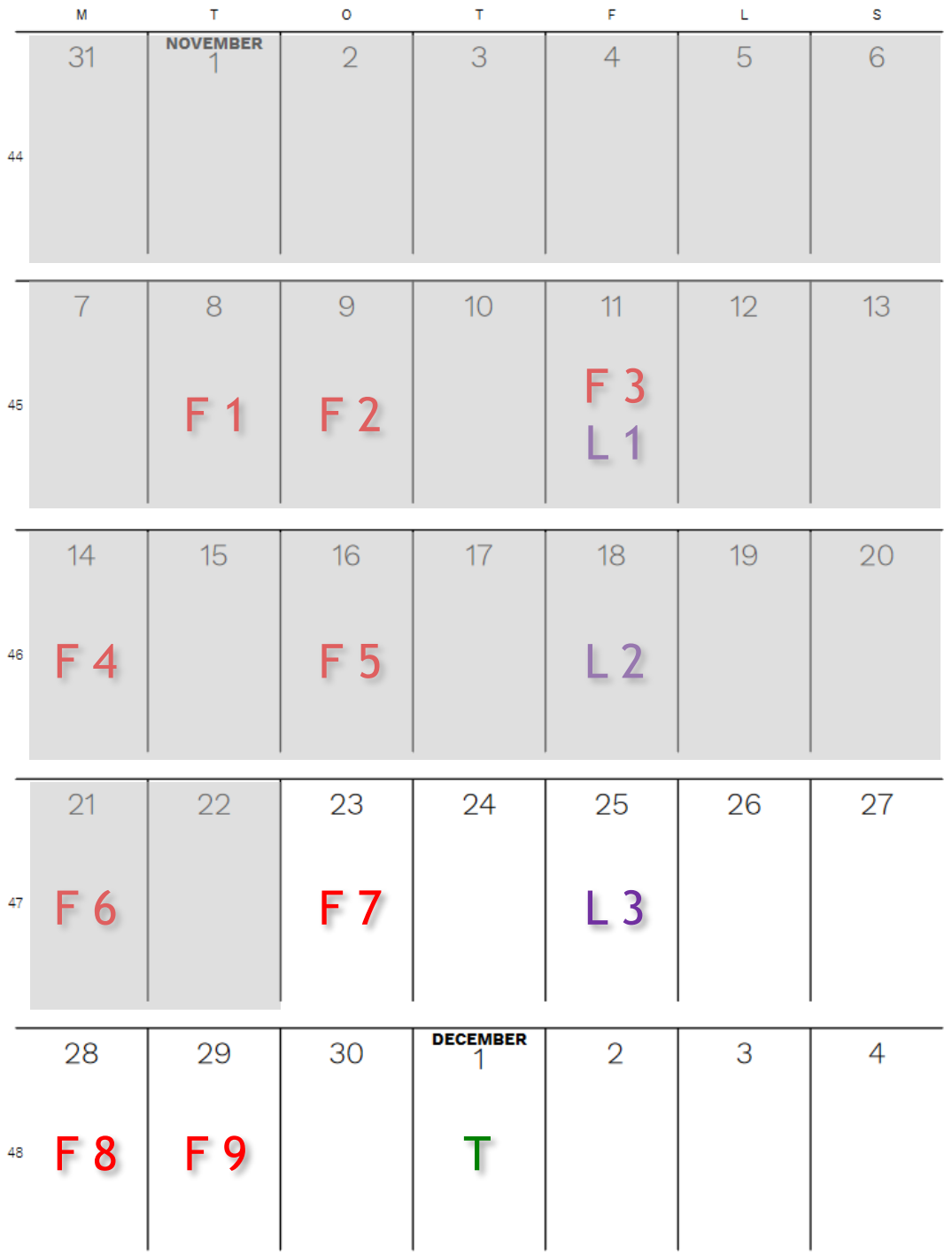
- Här kommer vi till ett litet problem.. Alla böcker och föreläsare verkar ha sitt eget sätt att beskriva hur en graf ska översättas i programkod

- Se kap. 7.6 i boken för en bra variant

Slut om grafer och träd för idag

Översikt

- F7: Träd och Grafer
- F8-F9: Repetition



Mer läsning

Bok

- **Problem Solving with Algorithms and Data Structures Using Python**
- Länk

Dagens material: Kapitel 6 och 7