UiT Norges arktiske universitet

DTE-2602
Introduksjon til maskinlæring og AI

Introduksjon til trær

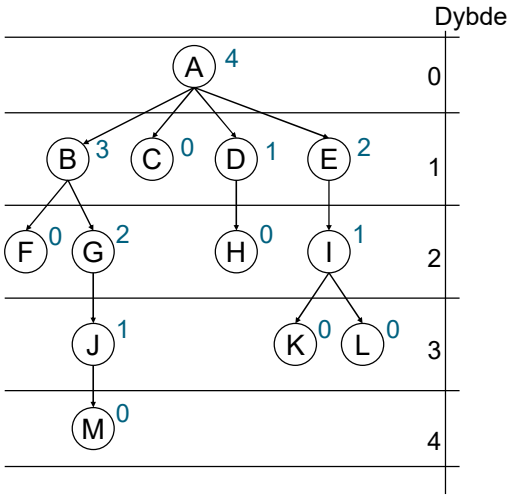
Asbjørn Danielsen
Førsteamanuensis
Rom: D2260
Epost: asbjorn.danielsen@uit.no

Innhold …

- Vi skal se på forskjellige varianter av trær med fokus på:
  - Trær generelt
  - Rekursjon
  - Binære trær
  - Traversering av trær
    - pre order
    - in order
    - post order
    - level order

Hva er et tre?

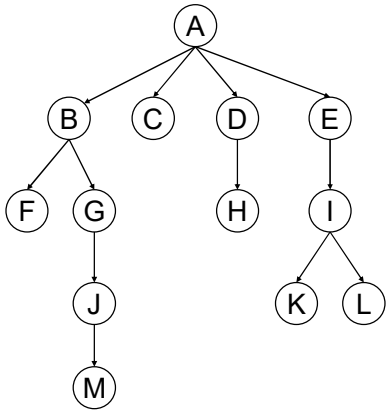
- En datastruktur av noder med rettede kanter mellom nodene
- En startnode
- Hver node har en, ingen eller flere barn-noder
- Kantene går fra en foreldrenode til barnnoder



Tallene ved nodene angir relativ høyde på hver enkelt node

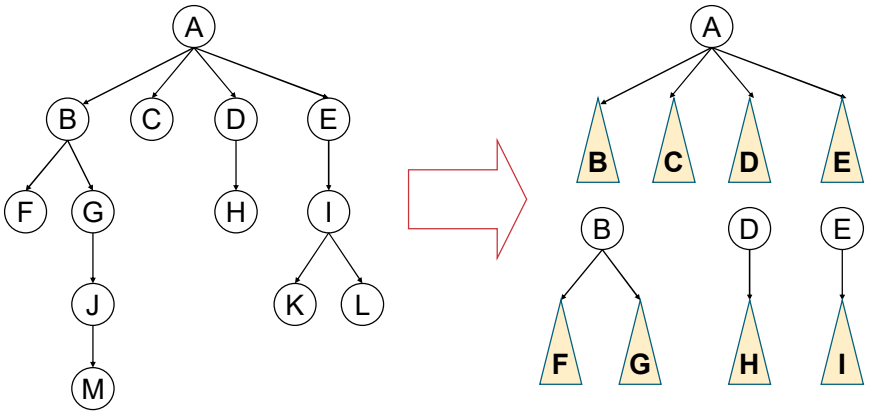
Begreper

- root-node
  - startnoden
- Foreldre
  - realtivt sett foreldre
- Barn
  - underordnet en foreldre



Prinsipp - datastruktur

- Et tre kan representeres rekursivt



Rekursiv definisjon av et tre

- Et tre består av:
  - En root-node
  - root-noden består av:
    - null ett eller flere trær

      … og …

    - objekt/verdi for noden som utgjør root-noden

Tre → Node
Node → Value ^ ([n • Tre] | null)

Trær - Brukes til hva da?

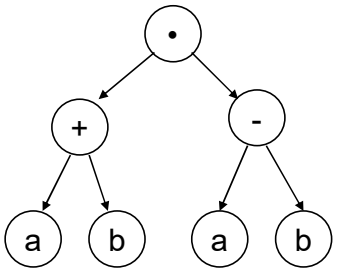
- Representere Filstrukturer
- Evaluering av uttrykk i matematikken
- Optimaliserte søk (søketrær)
- Databaser (gjenfinning og oppdatering)
- Avhengigheter
- Hierarkiske strukturer
- Komprimering (Huffman)

Binære trær

- En undertype av trær
  - En viktig forskjell fra andre trær
    - Maksimalt to barnenoder; venstre og høyre
- Brukes til:
  - uttrykkstre
  - søketrær
  - Huffman encoding (komprimeringsalgoritme)

Uttrykkstre

- Representerer matematiske uttrykk
  Eksempel: ((a + b) • (a - b))
- Kan traverseres på flere måter:
  - preorder:
    - noden behandles på tur inn i noden
      • + a b - a b
  - inorder: ((a + b) • (a - b))
  - postorder: a b + a b - •
  - levelorder:
    - nodene behandles ut fra dybden fra root-noden
      • + - a b a b



Kan traverseres på flere måter:

preorder:

noden behandles på tur inn i noden

• + a b - a b

inorder:

noden behandles inne i noden

((a + b) • (a - b))

postorder:

noden behandles på tur tilbake

a b + a b - •

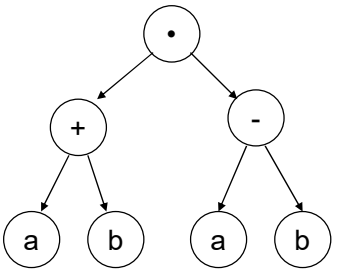
levelorder:

nodene behandles ut fra dybden fra root-noden

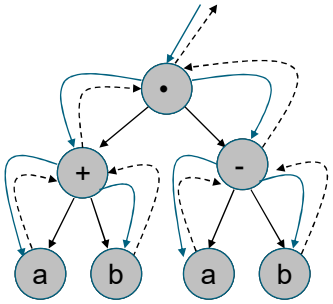
• + - a b a b

```python
class MyBinaryNode:
    def __init__(self, value,
                 lefttree = None,
                 righttree = None):
        self.value = value
        self.left = lefttree
        self.right = righttree

    # Bruker setter/getters
    @property
    def value(self):
        return self.__value

    @value.setter
    def value(self, value):
        self.__value = value

    @property
    def left(self):
        return self.__left

    @left.setter
    def left(self, lefttree):
        self.__left = lefttree

    @property
    def right(self):
        return self.__right

    @right.setter
    def right(self, righttree):
        self.__right = righttree

    def __eq__(self, node):
        if node == None:
            return False
        elif not isinstance(node, MyBinaryNode):
            raise Exception("Equality are only for object of equal types")
        else:
            return self.value == node.value

    def __str__(self):
        return self.value

    def hasRight(self):
        return self.right != None

    def hasLeft(self):
        return self.left != None

    def info(self):
        retval = self.value + " : ( "
        if isinstance(self.left, MyBinaryNode):
            retval += self.left.value
        else:
            retval += "none"
        retval += " , "
        if isinstance(self.right, MyBinaryNode):
            retval += self.right.value +" )"
        else:
            retval += "none )"
        return retval
```

```python
leftNode = MyBinaryNode('+', MyBinaryNode('a'), MyBinaryNode('b'))
rightNode = MyBinaryNode('-', MyBinaryNode('a'),MyBinaryNode('b'))
rootNode = MyBinaryNode('•', leftNode, rightNode)
```



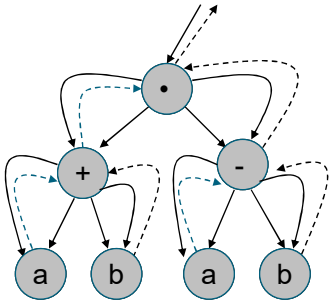
Preorder traversering



- + a b - a b

```
    def prefixOrder(self):
        print(self, ' ', end = '')
        if self.hasLeft():
            self.left.prefixOrder()
        if self.hasRight():
            self.right.prefixOrder()
```

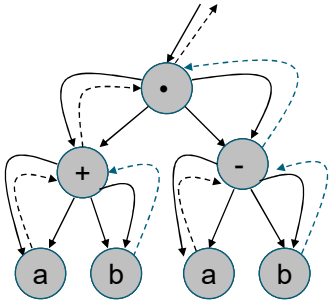
Inorder traversering



a + b • a - b

```
57
58      def infixOrder(self):
59          if self.hasLeft():
60              self.left.infixOrder()
61          print(self, ' ', end = '')
62          if self.hasRight():
63              self.right.infixOrder()
64
```

Postorder traversering

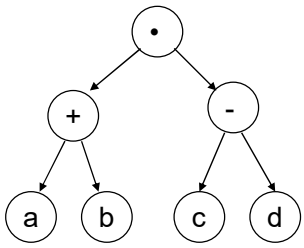


a b + a b - •

```
64
65      def postfixOrder(self):
66          if self.hasLeft():
67              self.left.postfixOrder()
68          if self.hasRight():
69              self.right.postfixOrder()
70          print(self, ' ', end = '')
71
```

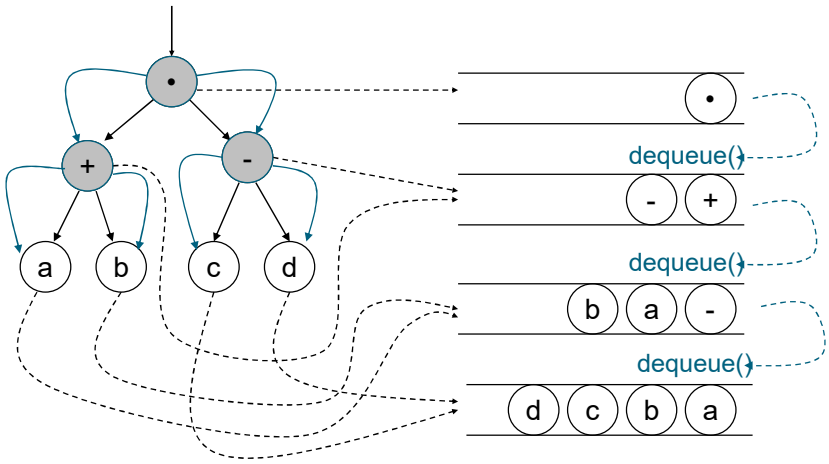
Levelorder traversering

- Siden man skal behandle nodene ut fra dybden på root-noden, kan man ikke traversere basert på enkel rekursjon (som for de andre traverseringene)
- Nodene traverseres basert på nivå, fra venstre mot høyre



Levelorder: • + - a b c d

Levelorder traversering med FIFO-kø




```
72      def levelOrder(self):
73          from queue import SimpleQueue
74          FIFOQueue = SimpleQueue()
75          FIFOQueue.put(self)
76          self.levelOrderEntry(FIFOQueue)
77          while not FIFOQueue.empty():
78              node = FIFOQueue.get()
79              print(node, " ", end='')
80
81      def levelOrderEntry(self, queue):
82          if queue.empty():
83              return
84          node = queue.get()
85          print(node, " ", end='')
86          if node.hasLeft():
87              queue.put(node.left)
88          if node.hasRight():
89              queue.put(node.right)
90          if node.hasLeft() or node.hasRight:
91              self.levelOrderEntry(queue)
```