Binære trær:

Noen algoritmer og anvendelser

Algoritmer / anvendelser:

- Søking i usortert binært tre
- Telling av antall noder og nivåer i treet
- Traversering av binære trær
- Binære uttrykkstrær
- Kunstig intelligens(?): «Guess the Animal»

Søking i et *usortert* binært tre

- Hvis treet ikke er sortert/ordnet, må vi sjekke en og en node inntil:
  - Verdien vi leter etter er funnet, eller
  - Vi har vært innom alle nodene i treet uten å finne søkt verdi
- Programmeres relativt elegant med to rekursive kall
- Eksempel: Binært tre med enkle tegn som data →

Java: Rekursiv søking i et *usortert* binært tre*

```
class treeNode
{
    char data;
    treeNode left, right;
}

boolean contains(treeNode root, char x)
{
    if (root == null)
        return false;

    if (root.data == x)
        return true;

    return (contains(root.left, x) ||
            contains(root.right, x));
}
```

*: Komplett kode: `BinCharTree.java`

Søking i usortert tre: Effektivitet

- Søking er $O(n)$ for et usortert tre med n noder, fordi worst-case er at alle nodene må oppsøkes
- Men: Hvis vi klarer å holde et binært tre *sortert* og *balansert*, vil søking i treet bli $O(\log n)$ *
- I tillegg blir da også både innsetting og fjerning av verdier $O(\log n)$

*: Søketrær og AVL-trær: Kapittel 11 i læreboken

Oppgave: «Telling» i et binært tre

- Ta utgangspunkt i metoden `contains` for søking i usortert tre
- Programmer følgende fire *rekursive* funksjoner:

1. `int numNodes(treeNode root)`

   Returnerer antall noder i treet med rot i noden `root`

2. `int numLeaves(treeNode root)`

   Returnerer antall bladnoder i treet med rot i roten `root`

3. `int numTwoChildren(treeNode root)`

   Returnerer antall noder i treet som har to barn

4. `int numLevels(treeNode root)`

   Returnerer antall nivåer (høyden) i treet med rot i roten `root`

1. Antall noder i et binært tre

```
int numNodes(treeNode root)
{
   // Returns number of nodes in binary tree

   if (root == null)
       return 0;

   return 1 + numNodes(root.left) + numNodes(root.right);
}
```

2. Antall bladnoder i et binært tre

```
int numLeaves(treeNode root)
{
   // Returns number of leaf nodes in binary tree

   if (root == null)
       return 0;

   if (root.left == null && root.right == null)
       return 1;

   return (numLeaves(root.left) + numLeaves(root.right));
}
```

3. Antall noder i et binært tre som har to barn

```
int numTwoChildren(treeNode root)
{
  // Returns number of binary nodes with two children

  if (root == null)
      return 0;

  int add = 0;

  if (root.left != null && root.right != null)
      add = 1;

  return add + numTwoChildren(root.left) +
               numTwoChildren(root.right);
}
```

4. Antall nivåer (høyden) i et binært tre *

```
int numLevels(treeNode root)
{
   // Returns number of levels (height) in binary tree

   if (root == null)
       return 0;

   int nLeft  = numLevels(root.left);
   int nRight = numLevels(root.right);
   int max = 0;

   if (nLeft > nRight)
      max = nLeft;
   else
      max = nRight;

   return 1 + max;
}
```

*: Komplett kode med testprogram for oppgavene 1-4: BinCharTree.java

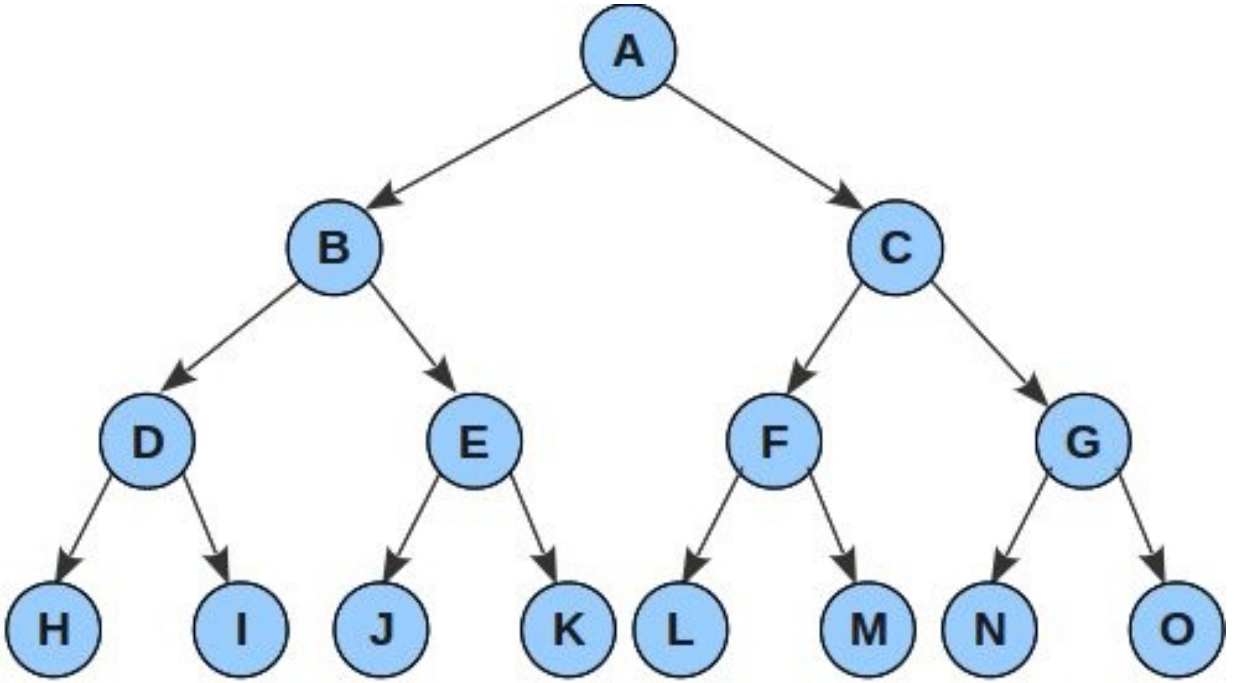
Traversering av binære trær

- Traversering:
  - Oppsøk hver node i treet én (og bare en) gang, på en eller annen systematisk måte
- Fire ulike standard traverseringer av binære trær:
  - Preorder
  - Inorder
  - Postorder
  - Bredde-først (level order)

Preorder traversering

Rekkefølge:

1. Roten
2. Venstre subtre
3. Høyre subtre

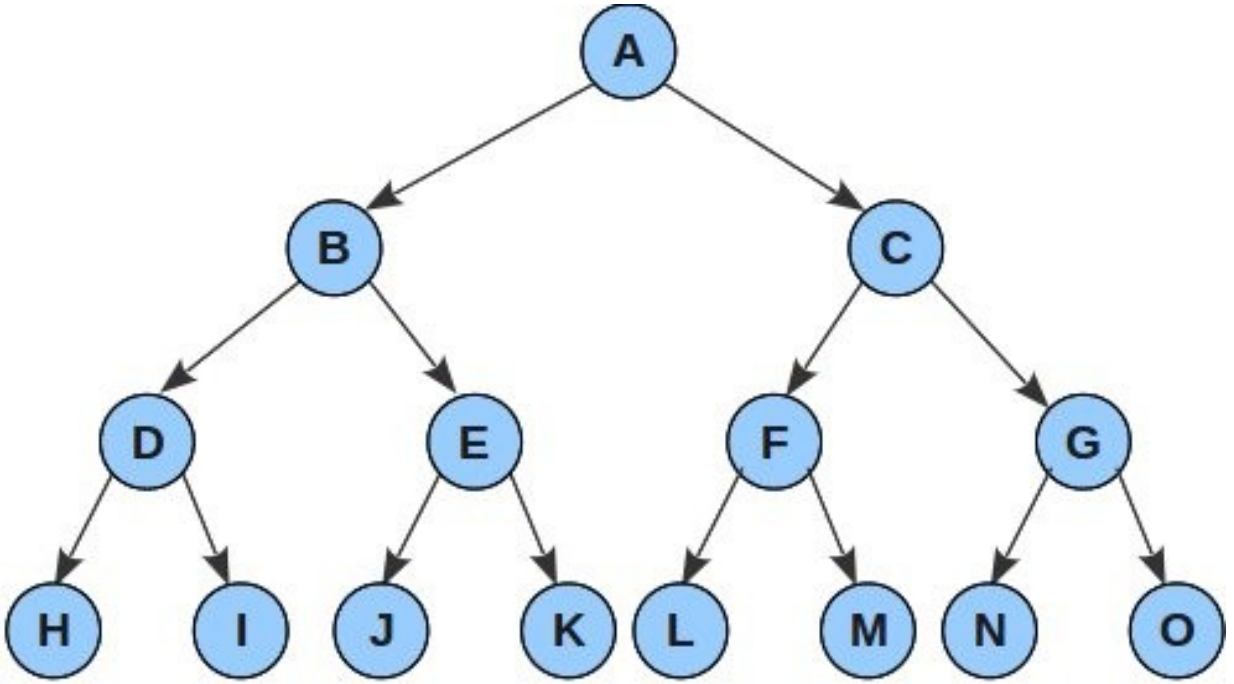


A B D H I E J K C F L M G N O

Inorder traversering

Rekkefølge:

1. Venstre subtre
2. Roten
3. Høyre subtre

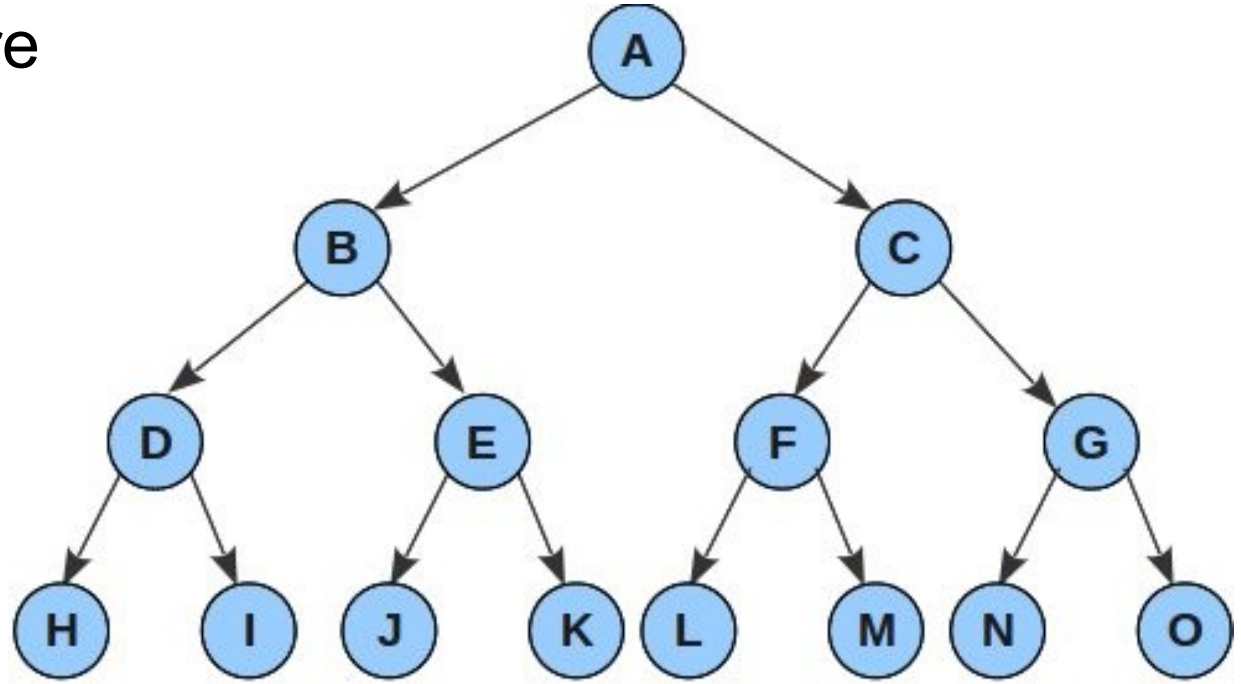


H D I B J E K A L F M C N G O

Postorder traversering

Rekkefølge:

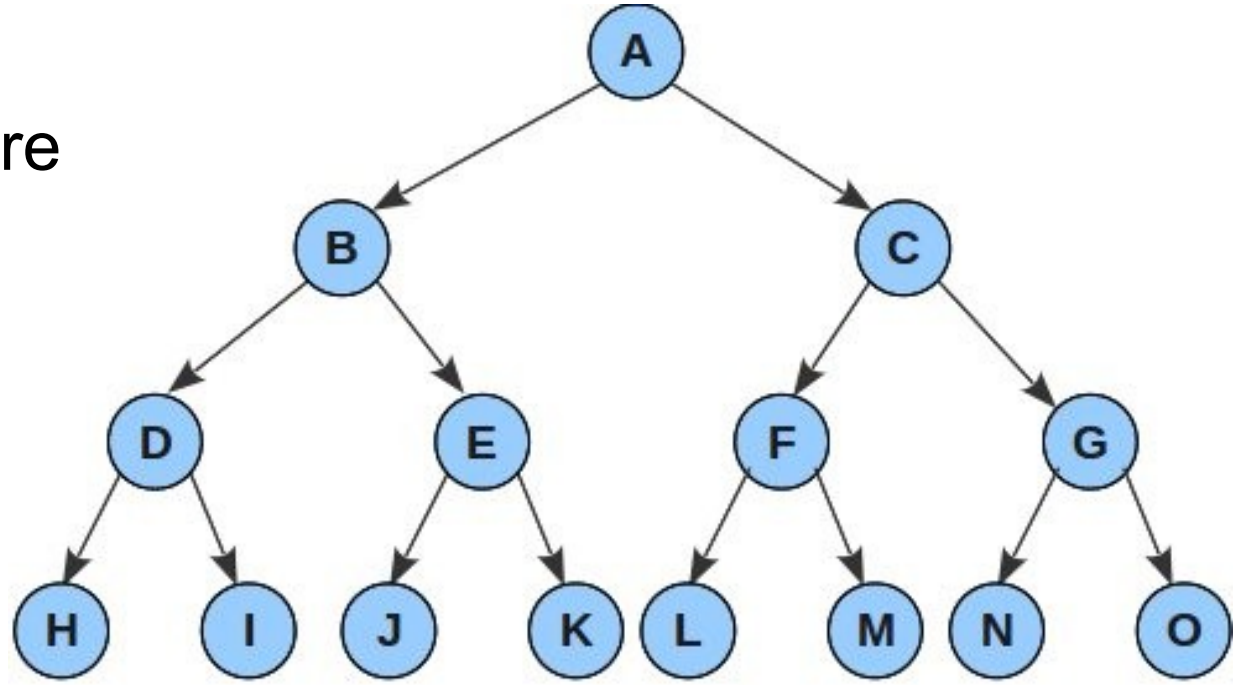
1. Venstre subtre
2. Høyre subtre
3. Roten



H I D J K E B L M F N O G C A

Bredde-først traversering

- Rekkefølge:
  - Nivå for nivå
  - Venstre mot høyre
  - Ovenfra og ned



A B C D E F G H I J K L M N O

Implementasjon av traverseringer

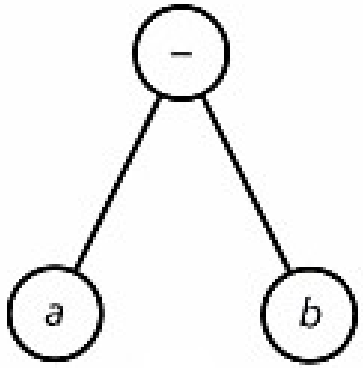
- Pre-, in- og postorder er alle 'dybde-først' traverseringer:
  - Programmeres enkelt med to rekursive kall
  - Men: Hvis det skal lages en standard Java-iterator må vi enten kopiere hele treet over i en liste (læreboka) eller simulere rekursjonen med en stack
- Bredde-først traversering:
  - Kan ikke implementeres rekursivt
  - Er mer fiklete fordi vi må lagre unna barna til en node samtidig som vi oppsøker noden
  - Bruker en kø til å lagre barn som ikke er oppsøkt
- Enkel demo: `treeTraversals.java`

En anvendelse: Uttrykkstrær

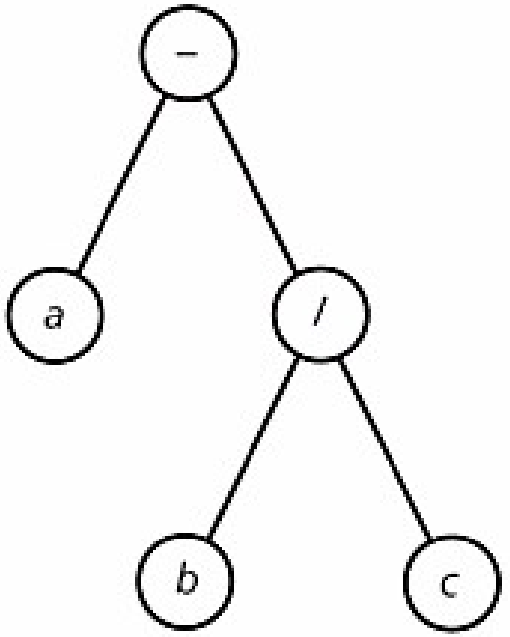
- Binære trær passer utmerket til lagre regneuttrykk med binære operatorer: `+` `-` `*` `/`
- Trenger ikke paranteser eller presedensregler
- Uttrykkene lagres slik:
  - Alle bladene i treet er operander (tall eller variabler)
  - Alle indre noder er binære operatorer
  - Venstre operand ligger i operatorens venstre subtre
  - Høyre operand ligger i operatorens høyre subtre

Uttrykkstrær: Eksempler

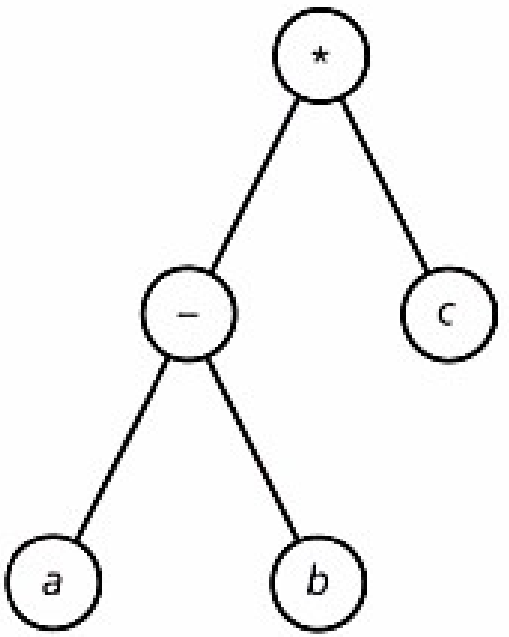
a – b



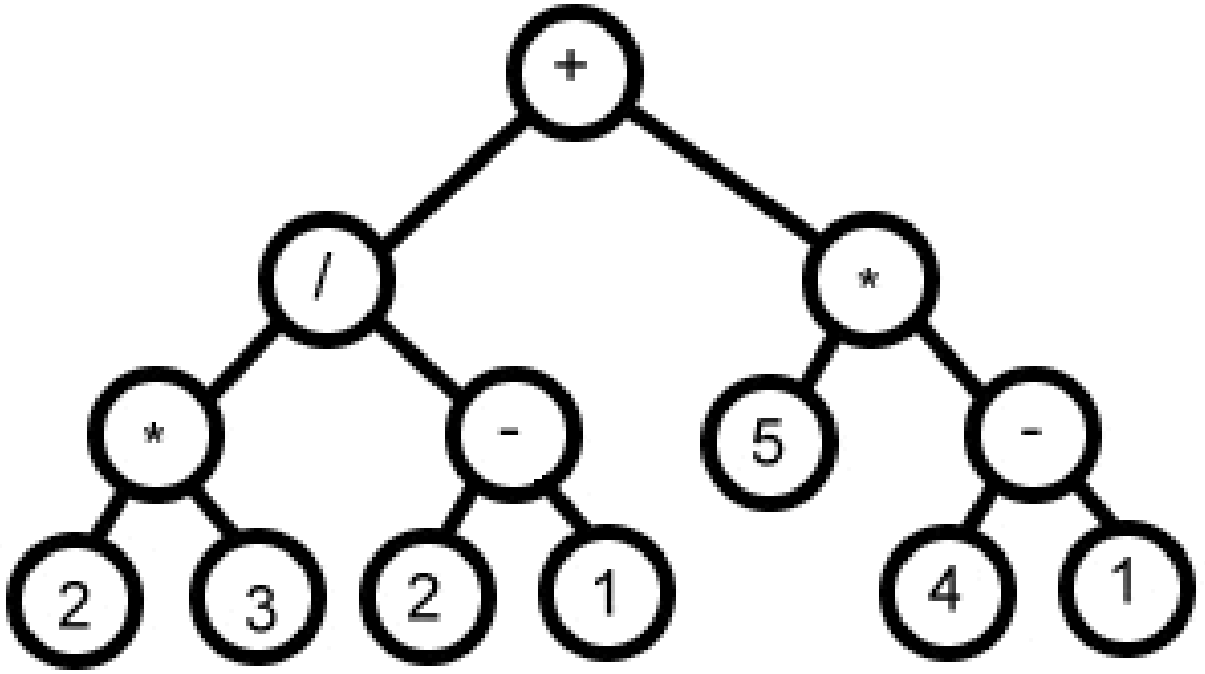
a – b / c



*(a – b) * c*



Uttrykkstrær: Eksempel



Expression tree for 2*3/(2-1)+5*(4-1)

- Læreboka implementerer en evaluator for å beregne verdien av regneuttrykk i uttrykkstrær der operandene er heltall, se avsnitt 10.5 og Java-koden

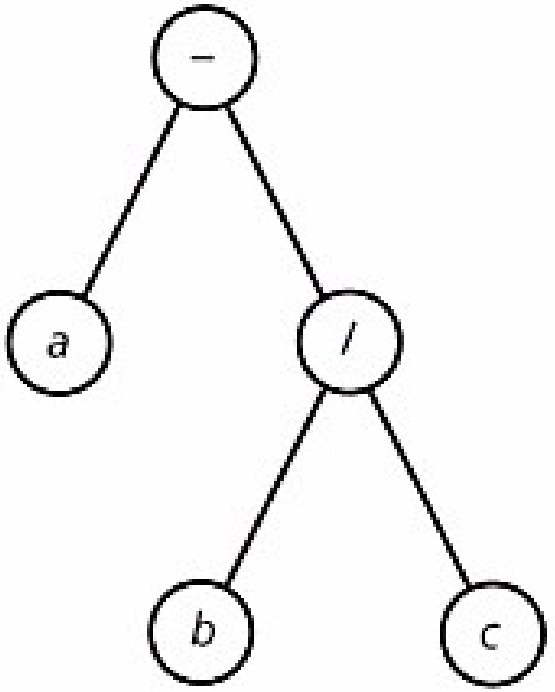
Traversering av uttrykkstrær

- Inorder traversering av treet skriver ut regne-uttrykkene i infix (vanlig) form:

  a – b / c

- Postorder traversering skriver ut regneuttrykkene i postfix form:

  a *b c / –*



En klassiker: «Guess the Animal»

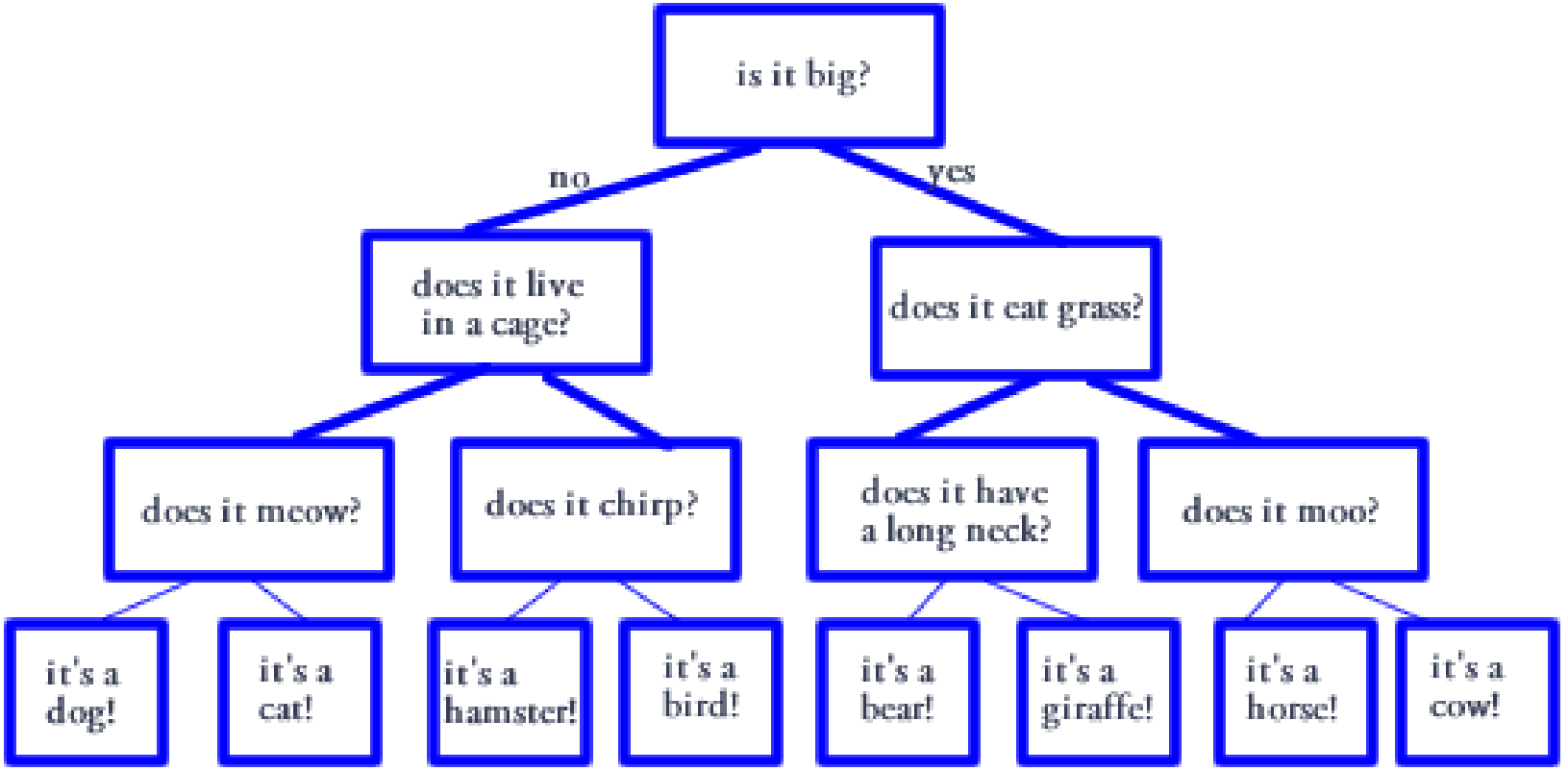
- Tidlig eksempel på maskinlæring og «kunstig intelligens», fra ca. 1960
- Programmet forsøker å gjette hvilket dyr brukeren tenker på, ved å stille spørsmål som alltid har svar «ja» eller «nei»
- Hvis programmet gjetter på feil dyr, bes brukeren om å skrive inn:
  - Hvilket dyr han/hun tenkte på
  - Et ja/nei spørsmål som kan brukes til å skille de to dyrene (det vi gjettet og det riktige) fra hverandre

«Maskinlæring» i GtA

- Programmet starter med et lite antall dyr som det kjenner til, og noen spørsmål som kan brukes til å skille dem fra hverandre
- Hver gang programmet gjetter feil, legges det inn et nytt dyr i «kunnskapsbasen» til programmet
- Det legges også inn et spørsmål som kan brukes til å skille det nye dyret fra et annet dyr
- Programmet blir derfor «smartere» jo mer det brukes – hvis bruker legger inn riktige data

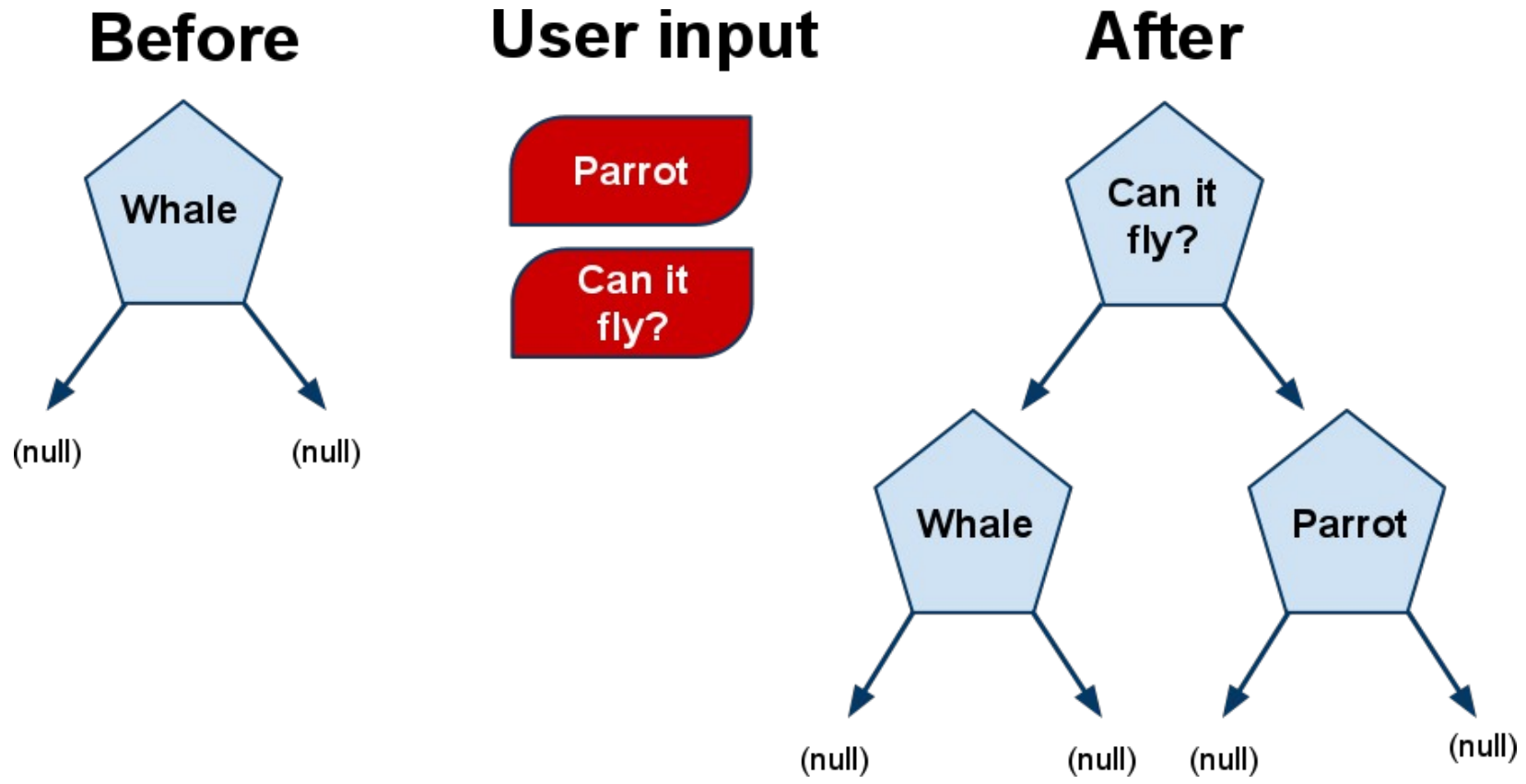
Lagring av kunnskapen i GtA

- Bruker et binært tre:
  - Alle bladnodene lagrer navnet på et dyr
  - Alle indre noder lagrer et ja/nei spørsmål



Oppdatering av kunnskapen i GtA

- Når vi gjetter feil:
  - Legger inn en ny bladnode med nytt dyr
  - Legger inn ny indre node med nytt spørsmål



Implementasjon

- Spiller GtA ved å gå gjennom kunnskapstreet fra roten og langs en vei helt ut til en bladnode
- I hver indre node vi er innom skriver vi ut spørsmålet og leser ja/nei fra bruker
- Går videre til venstre (nei) eller høyre (ja) barn
- Når vi kommer til en bladnode, gjetter vi på dyret som er lagret i bladnoden
- Hvis vi gjetter feil, leser vi riktig svar og nytt spørsmål fra bruker, og legger dette inn i treet
- Java-kode: `guessTheAnimal.java`