CS 173 Lecture 13

Recursion, induction proofs, context free grammars

13th October, 2016

Why recursion?

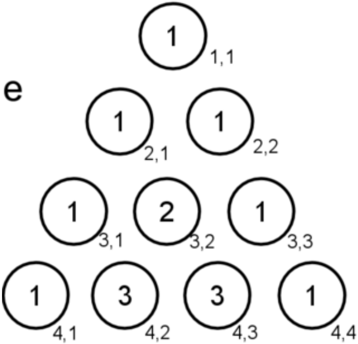
- Till now we saw representations of objects or equations in semi-formal way, where patterns could easily be seen. E.g. summation

$$\sum_{i=1}^{n} i = 1 + 2 + 3...(n-1) + n \tag{1}$$

- What if those patterns are less obvious? Recursive definitions to the rescue! Similar to recursive procedures in programming
- Defines objects in terms of smaller objects of the same type, with an end at some point. Includes
  - Base case
  - Recursive formula
- Fibonacci sequence as recursion:
  $\mathrm{F}_0 = 1, F_1 = 1$
  $\mathrm{F}_i = F_{i-1} + F_{i-2}, \forall i \geq 2$

More recursion examples

- $f : Z^+ \to N, f(n) = \sum_{k=1}^{n} k(k-1)$
- Pascal's triangle
- Series: 1, 2, 2, 4, 8, 32, 256,...



Closed form expressions

- Mathematical expression that can be evaluated in finite number of operations. E.g. Solutions to a quadratic equation

$$ax^2 + bx + c = 0 \tag{2}$$

- Many recursive formulas have closed form. Can do so by
  - Guessing pattern
  - Unrolling

Closed form - unrolling

- Definition:

$$T(1) = 1$$
$$T(n) = 2T(n-1) + 3, \forall n \geq 2$$

- Unroll:

$$T(n) = 2(2T(n-2) + 3) + 3$$
$$T(n) = 2(2(2T(n-3) + 3) + 3) + 3$$
$$T(n) = 2^3 T(n-3) + 2^2 * 3 + 2 * 3 + 3$$
$$T(n) = 2^4 T(n-4) + 2^3 * 3 + 2^2 * 3 + 2 * 3 + 3$$
$$\ldots$$
$$T(n) = 2^k T(n-k) + 2^{(k-1)} * 3 + \ldots + 2^2 * 3 + 2 * 3 + 3$$
$$T(n) = 2^k T(n-k) + 3 * \sum_{i=0}^{k-1} 2^i$$

Closed form - unrolling (continued)

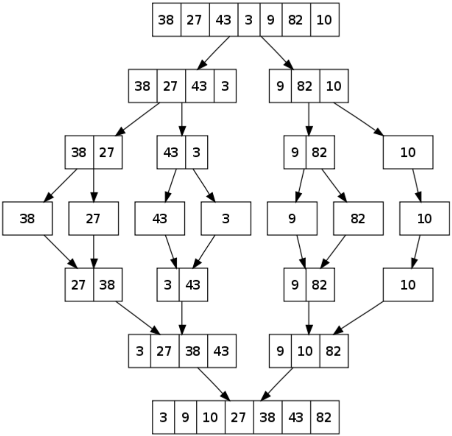
- When do you hit the base case? When n - k = 1

$$T(n) = 2^{n-1}T(1) + 3 * \sum_{i=0}^{n-2} 2^i$$
$$T(n) = 2^{n-1} + 3 * (2^{n-1} - 1)$$
$$T(n) = 2^{n+1} - 3$$

Divide and conquer

- Merge sort is an example

Figure: Merge sort



Divide and conquer

- Divide a big problem of size n into a sub-problems
- Analyze such problems by looking at recursive definition, account for dividing big problem and merging solutions for smaller problems
- Example: Unrolling a cost function expressed recursively

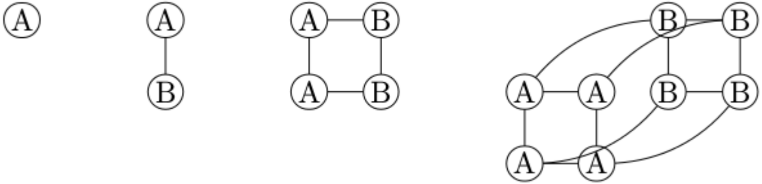
$$S(1) = c$$
$$S(n) = 2S(n/2) + n, \forall n \geq 2 \ (n \ is \ a \ power \ of \ 2)$$

gives

$$S(n) = cn + nlogn$$

Recursive definition of non-numerical objects

- Recursively define hypercube Q_n in terms of Q_{n-1}



- Recursive definition for computing number of edges E(n) in Q_n?

Proofs with recursive definitions

- Inductive proofs are well suited for recursive functions
- Example: For any n ≥ 0, $(3n)^{th}$ Fibonacci term F_{3n} is even.
- How would we prove it?

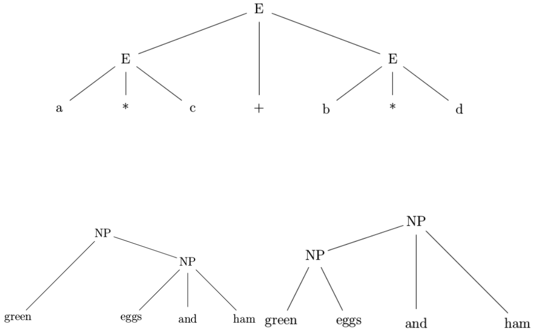
Strong induction on recursive definition proofs

- Prove closed form is correct for a recursive definition
- Prove $\forall n \epsilon N$, f(n) $= 2^n + 1$, where f : N $\rightarrow$ Z and

$$f(0) = 2$$
$$f(1) = 3$$
$$f(n+1) = 3f(n) - 2f(n-1) \ \forall n \geq 1$$

Trees and induction

- Trees are data structures for organizing/storing data such as:
  - Efficient storage/retrieval (binary search trees)
  - Parse trees, used for structure of a program/expression such as a*c + b*d
  - Store sentence structures in NLP, "green eggs and ham"
  - Decision trees, to classify data



Trees examples

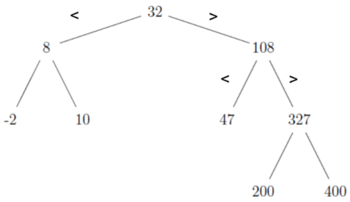


Figure: Trees for sorting

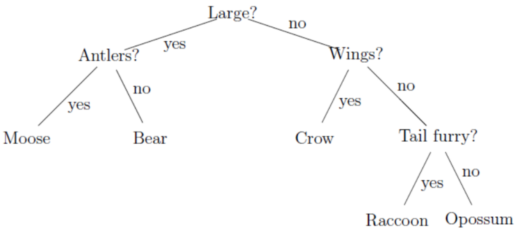
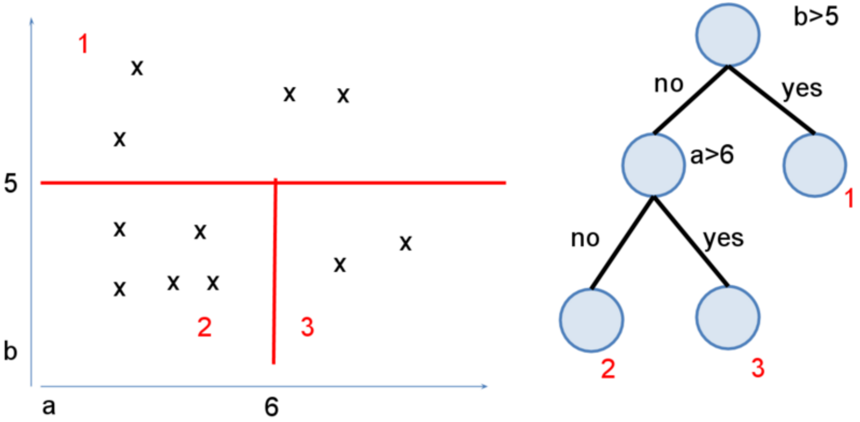


Figure: Decision trees

Trees for clustering

- Goal: Create a function that maps from R^N to Z^+ such that nearby N-dimensional points are mapped to the same integer



Properties of trees

- Trees are undirected graphs with special node called root (to which every other node is connected by one path)
- m-ary trees allow each node to have upto m children. Full: 0 or m children, Complete: all leaves at same height
- Properties
  - A full and complete m-ary tree with i internal nodes has $mi + 1$ nodes in total
  - Bounds for leaves and total nodes in binary tree?

Trees and induction

- In a binary tree of height h, the number of nodes $n \leq 2^{h+1} - 1$
- Use induction to prove the claim?

Context free grammars

- Set of rules that defines a set of possible parse trees
- Tells us what sorts of children are possible for a parent node with each type of label
- Specifies set of rules, with valid start symbols and valid terminals
- Example: $S \rightarrow Sa$, $S \rightarrow a|b|c$
  Start symbols: S
  Terminals: a, b, c

Context free grammars example

- $S \rightarrow bSa$
- $S \rightarrow a|b|ca$
- Which of these strings can be generated by grammar above: a, bba, ba, abbcaa, bca, bbbcaaaa

Next class

- More real world examples on parse trees and CFG's
- Induction proofs on CFG's
- Recursion trees and more proofs with trees