Graph Theory
Lecture 10

ShanghaiTech University

29 May 2019

Constructing a binary search tree

- Start with an ordered list of items having **labels (keys)**.
  Want to store them in a **binary search tree**.
- Use an algorithm to locate items in the tree and/or add missing items.

Constructing a binary search tree

- Start with an ordered list of items having **labels (keys)**. Want to store them in a **binary search tree**.
- Use an algorithm to locate items in the tree and/or add missing items.

Constructing the tree:

1. First item in the list $\rightarrow$ label of the root,
2. To add a new item x in the tree, compare its label with labels of vertices already in the tree starting from the root:
   - move to the left child if label(x) is less than the label of the respective vertex in the tree,
   - move to the right child if label(x) is greater than the label of the respective vertex in the tree.
   - if no left or right child: create a new vertex

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Binary search tree:

- *mathematics* $\rightarrow$ label of the root,

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Binary search tree:

- *mathematics* $\rightarrow$ label of the root,
- *physics* $>$ *mathematics* $\Rightarrow$ add a right child of the root with label *physics*,

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Binary search tree:

- *mathematics* $\rightarrow$ label of the root,
- *physics* $>$ *mathematics* $\Rightarrow$ add a right child of the root with label *physics*,
- *geography* $<$ *mathematics* $\Rightarrow$ add a left child to the root with label *geography*,

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Binary search tree:

- *mathematics* $\rightarrow$ label of the root,
- *physics* $>$ *mathematics* $\Rightarrow$ add a right child of the root with label *physics*,
- *geography* $<$ *mathematics* $\Rightarrow$ add a left child to the root with label *geography*,
- *zoology* $>$ *mathematics* $\Rightarrow$ move to the right child
  zoology $>$ *physics* $\Rightarrow$ add a right child with label *zoology* to the vertex labeled *physics*,

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Binary search tree:

- *mathematics* $\to$ label of the root,
- *physics* $>$ *mathematics* $\Rightarrow$ add a right child of the root with label *physics*,
- *geography* $<$ *mathematics* $\Rightarrow$ add a left child to the root with label *geography*,
- *zoology* $>$ *mathematics* $\Rightarrow$ move to the right child
  zoology $>$ *physics* $\Rightarrow$ add a right child with label *zoology* to the vertex labeled *physics*,
- *meteorology* $>$ *mathematics* $\Rightarrow$ move to the right child
  meteorology $<$ *physics* $\Rightarrow$ add a left child with label *meteorology* to the vertex labeled *physics*,

Example

Ordered list: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry*. Label given by alphabetical order.

Binary search tree:

- *mathematics* $\rightarrow$ label of the root,
- *physics* $>$ *mathematics* $\Rightarrow$ add a right child of the root with label *physics*,
- *geography* $<$ *mathematics* $\Rightarrow$ add a left child to the root with label *geography*,
- *zoology* $>$ *mathematics* $\Rightarrow$ move to the right child
  zoology $>$ *physics* $\Rightarrow$ add a right child with label *zoology* to the vertex labeled *physics*,
- *meteorology* $>$ *mathematics* $\Rightarrow$ move to the right child
  meteorology $<$ *physics* $\Rightarrow$ add a left child with label *meteorology* to the vertex labeled *physics*,
- *geology* $<$ *mathematics* $\Rightarrow$ move to the leftt child
  geology $>$ *geography* $\Rightarrow$ add a right child with label *geology* to the vertex labeled *geography*,
- and so on until each item is in the tree.

Locate items in a binary search tree and add missing items

Insertion algorithm: take as input a binary search tree and an item x. Locate the item x if it is in the tree, otherwise add a vertex with label x.

input: T binary search tree, x item
$\nu :=$ root of the tree (a vertex not present in T has value *null*)
while $\nu \neq$ *null* and $label(\nu) \neq x$
  if $x < label(\nu)$ **then**
    if left child of $\nu \neq$ *null* **then** $\nu :=$ left child of ν
    else add *new vertex* as left child of ν
      $\nu :=$ *null*
  else $(x \geq label(\nu))$
    if right child of $\nu \neq$ *null* **then** $\nu :=$ right child of ν
    else add *new vertex* as right child of ν
      $\nu :=$ *null*
if root of $T =$ *null* **then** add a vertex ν with label x to the tree
else if $\nu =$ *null* **then** label *new vertex* with x
   $\nu :=$ *new vertex*
return ν

Complexity

- T binary search tree for a list of n items
- U full binary tree obtained from T by adding the minimum number of unlabeled vertices such that each vertex of T is an internal vertex of U.
- Most comparisons needed to add an item in the tree = height of U.

Complexity

- T binary search tree for a list of n items
- U full binary tree obtained from T by adding the minimum number of unlabeled vertices such that each vertex of T is an internal vertex of U.
- Most comparisons needed to add an item in the tree = height of U.

Theorem (Recall)

A full m-ary tree with i internal vertices has $mi + 1$ *vertices and* $\ell = (m-1)i + 1$ *leaves.*

$\Rightarrow$ U has $n+1$ leaves $\Rightarrow$ height of $U \geq \lceil \log_2(n+1) \rceil$

Theorem

In a binary search tree for a list of n items, to add an item requires at least $\lceil \log(n+1) \rceil$ *comparisons (worst-case).*

Remark: A balanced binary search tree gives optimal worst-case complexity for binary searching.

Definition

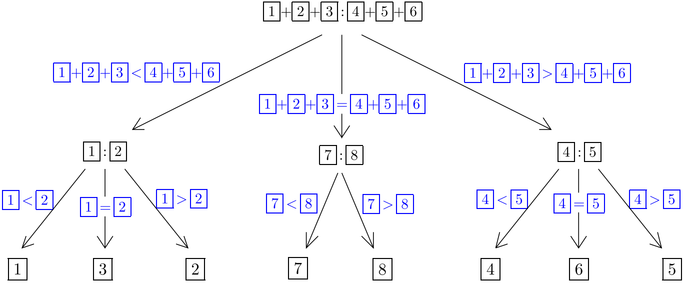
A rooted tree in which each internal vertex corresponds to a decision, with one subtree for each possible outcome of the decision, is called a **decision tree**.

Definition

A rooted tree in which each internal vertex corresponds to a decision, with one subtree for each possible outcome of the decision, is called a **decision tree**.

Examples:

- Find a minimum of 3 numbers a, b, c,
- Binary search tree: locate items based on a series of comparisons,
- Determining a counterfeit coin (less weight) among 8 coins:



Decision tree for sorting algorithm

- Use a decision tree to sort a list of n items: $n!$ possible orderings
- Sorting algo based on binary comparison $\rightsquigarrow$ Binary decision tree with:
  - internal vertices: comparison
  - leaf: possible ordering ($n!$ leaves)
- Find a lower bound for the worst case complexity of sorting algorithms based on binary comparisons.

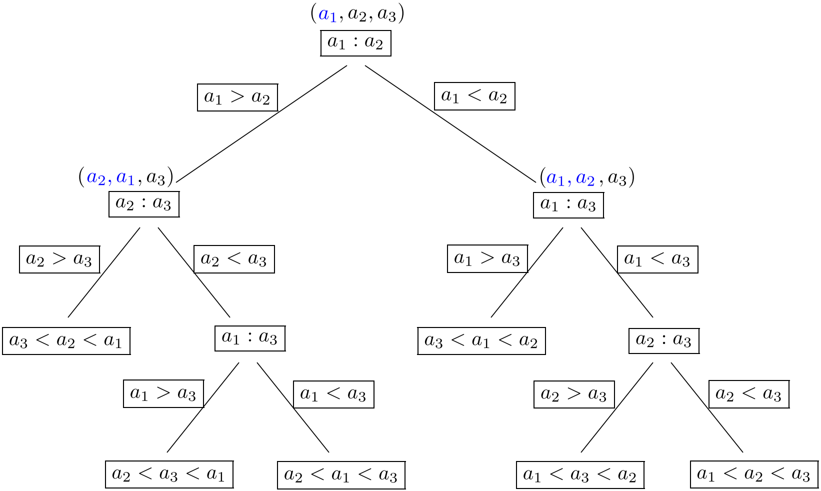
Decision tree for sorting algorithm

- Use a decision tree to sort a list of n items: $n!$ possible orderings
- Sorting algo based on binary comparison $\rightsquigarrow$ Binary decision tree with:
  - internal vertices: comparison
  - leaf: possible ordering ($n!$ leaves)
- Find a lower bound for the worst case complexity of sorting algorithms based on binary comparisons.

Algorithm: Insertion sort

input: $(a_1, \ldots, a_n)$ list of real numbers with $n \geq 2$
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i + 1$ the while loop stops at the first i such that $a_i > a_j$
    $m := a_j$
    for $k := 0$ **to** $j - i - 1$
        $a_{j-k} = a_{j-k-1}$ shift $a_j, a_{j-1}, \ldots, a_i$ by one position to the right
    $a_i = m$
return $(a_1, \ldots, a_n)$ list in increasing order

Example: Binary decision tree for sorting 3 distinct elements a_1, a_2, a_2.



Complexity

Worst-case $=$ largest number of binary comparisons needed to sort a list of n elements
$=$ longest path representing a sorting
$=$ height of the decision tree

Let h denote the height of a binary tree with $n!$ leaves

$$h \geq \lceil \log_2 n! \rceil$$

and we have

$$\lceil \log_2 n! \rceil \geq \log_2 n! \geq \frac{1}{2} n \log_2 n$$

Theorem

The number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$

Question: How to code letters of the English alphabet with bit strings?
26 letters $\rightsquigarrow$ can use bit strings of length 5 (there are 32)
But: costs too much.

Idea: Using bit strings of various lengths.

Example: Consider the alphabet $\{a, b, c, d\}$. If a is encoded with 0, b with 1, c with 01, d with 10.
The bit string 10111 can correspond to the words *babbb*, *dbbb*, *bcbb*.

Question: How to code letters of the English alphabet with bit strings?
26 letters $\rightsquigarrow$ can use bit strings of length 5 (there are 32)
But: costs too much.

Idea: Using bit strings of various lengths.

Example: Consider the alphabet $\{a, b, c, d\}$. If a is encoded with 0, b with 1, c with 01, d with 10.
The bit string 10111 can correspond to the words *babbb*, *dbbb*, *bcbb*.

Definition

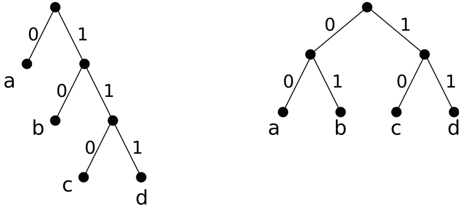
A **prefix code** is a code system using code words so that no code word is prefix of another code word.

Example: a is encoded with 0, b with 10, c with 110, d with 111.
Then the bit string 10111 corresponds to *bad* with no ambiguity.

Prefix codes and binary trees

Each binary tree gives a prefix code: the left edge at each internal vertex is labeled with 0 and the right edge with 1. Each leaf corresponds to a character encoded.

Example:



Prefix code given by the binary tree on the left: a:0, b:10, c:110, d:111
Prefix code given by the binary tree on the right: a:00, b:01, c:10, d:11

Huffman coding

Goal: Producing a prefix code with the fewest possible bits. The symbols which occur the most will be coded with bit strings of shortest length.

Algorithm: construct a binary tree whose leaves are characters we want to encode

input: symbols a_i with frequencies w_i, $i = 1, \ldots, n$
$F =$ forest of n rooted trees, each consisting of one vertex a_i and weighted with w_i.
while F is not a tree
  Let T' and T'' be the trees of less weights in F, with $w(T') > w(T'')$
  Replace T' and T'' by a rooted tree T such that the left subtree at its root is T' and the right subtree is T''
  $w(T) = w(T') + w(T'')$
return F

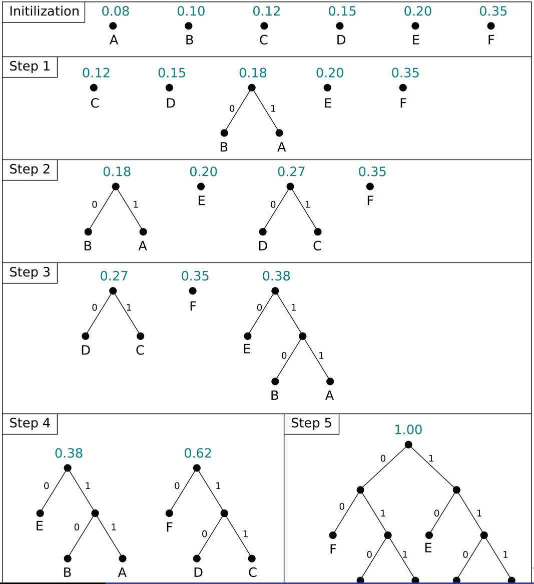
Theorem

Huffman coding algorithm is an optimal algorithm: no binary prefix code for these symbols can encode these symbols using fewer bits.

Example: Find an optimal prefix code for the following symbols and frequencies. (A, 0.08); (B, 0.10); (C, 0.12); (D, 0.15); (E, 0.20); (F, 0.35).

By Huffman coding we get: A:111; B:110; C:011; D:010; E:10; F:00.

The average number of bits used to encode a symbol is: $3\cdot 0.08+3\cdot 0.10+3\cdot 0.12+3\cdot 0.15+2\cdot 0.20+2\cdot 0.35=2.45$



Definition

A rooted tree in which vertices correspond to positions that a game for two players playing in turn can be in as it progresses, and edges represent legal moves between these positions, is called a **game tree**.

- root = starting position of the game.
- **Convention:** boxes for vertices at even level, and circles for vertices at odd levels. An edge from a box to a circle is a move of the first player, an edge from a circle to a box represent a move of the second player.
- **Leaves:** final positions of the game. Assign $+1$ the leaf if the first player wins at this final position, -1 if the second player wins, or 0 in case of draw.
- **Value of a leaf:** payoff to the first player
- **Value of an internal boxed vertex:** maximum of the values of its children
- **Value of an internal circled vertex:** minimum of the values of its children

Example: Game of Nim