Lecture 10

Algorithms & Data Structures

Goldsmiths Computing

December 10, 2018

Outline

Outline

Lecture

- String matching
  - Naïve: $\Theta(nm)$

Lecture

- String matching
  - Naïve: $\Theta(nm)$
- More string matching
  - Rabin-Karp: $\Theta(n+m)$ usually, $\Theta(nm)$ worst case

Lecture

- String matching
  - Naïve: $\Theta(nm)$
- More string matching
  - Rabin-Karp: $\Theta(n+m)$ usually, $\Theta(nm)$ worst case
- Even more string matching
  - Knuth-Morris-Pratt: $\Theta(n+m)$
  - Boyer-Moore: $\Theta(n+m)$ sometimes, $\Theta(n/m)$ best case

Lecture

- String matching
  - Naïve: $\Theta(nm)$
- More string matching
  - Rabin-Karp: $\Theta(n+m)$ usually, $\Theta(nm)$ worst case
- Even more string matching
  - Knuth-Morris-Pratt: $\Theta(n+m)$
  - Boyer-Moore: $\Theta(n+m)$ sometimes, $\Theta(n/m)$ best case
- (and some string matching that I didn't get to!)
  - next term...

Labs

1. Implement string-matching algorithms:
    1.1 naïve string matching
        - nested loops
    1.2 Rabin-Karp matching
        - rolling hash
    1.3 Knuth-Morris-Pratt
        - prefix table

Labs

1. Implement string-matching algorithms:
   1.1 naïve string matching
      - nested loops
   1.2 Rabin-Karp matching
      - rolling hash
   1.3 Knuth-Morris-Pratt
      - prefix table
2. measure performance
   - how many character reads?

VLE activities

Binary trees quiz

Statistics so far:

- 163 attempts: average mark 4.61
- 72 students: average mark 4.56
  - 34 under 4.00, 14 over 6.99, 6 at 10.00

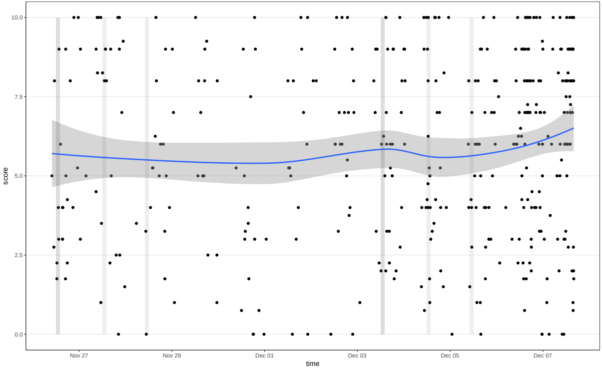
Quiz closes at 16:00 on Friday 14th December

- no extensions
- grade is
  - 0 (for no attempt)
  - $30 + 70 \times (\text{score}/10)^2$

VLE activities (cont'd)

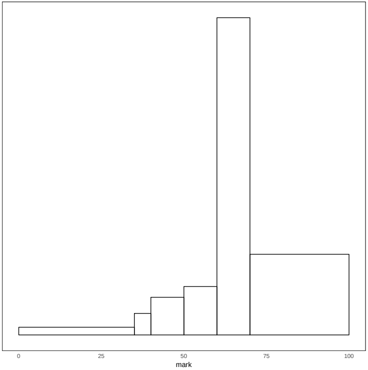
Recursive functions quiz

- 405 attempts: average mark 5.84
- 131 students: average mark 7.88
  - 13 under 4.00, 103 above 6.99, 45 at 10



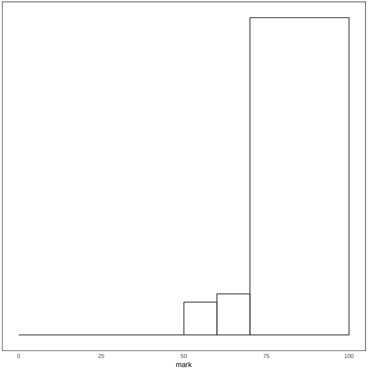
VLE activities (cont'd)

List visualiser



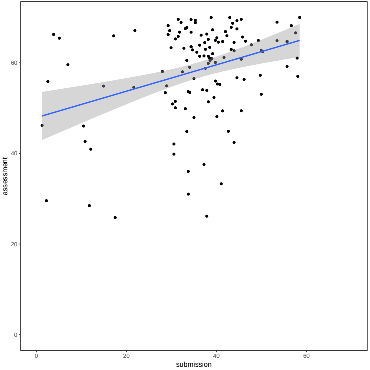
VLE activities (cont'd)

List visualiser



VLE activities (cont’d)

List visualiser



VLE activities (cont'd)

First-term questionnaire

Thank you for your answers!

VLE activities (cont'd)

First-term questionnaire

Thank you for your answers!

- slides are not meant to be the complete knowledge repository
  - textbooks
  - online courseware, MOOCs
  - wikipedia (and links therefrom)
  - infinite youtube videos

VLE activities (cont'd)

First-term questionnaire

Thank you for your answers!

- slides are not meant to be the complete knowledge repository
  - textbooks
  - online courseware, MOOCs
  - wikipedia (and links therefrom)
  - infinite youtube videos
- quizzes/labs are meant to contain things not necessarily in lectures
  - try implementing
  - can you solve it?

VLE activities (cont'd)

First-term questionnaire

Thank you for your answers!

- slides are not meant to be the complete knowledge repository
  - textbooks
  - online courseware, MOOCs
  - wikipedia (and links therefrom)
  - infinite youtube videos
- quizzes/labs are meant to contain things not necessarily in lectures
  - try implementing
  - can you solve it?
- people learn in different ways
  - (some people even like kahoots)

VLE activities (cont'd)

Module evaluation

Module evaluation is open at this link (also from module page on learn.gold)

- answers held anonymously
- please take evaluation survey by Friday 14th December
- (also for your other modules!)

Outline

Motivation

- simplest form of tree data structure
- algorithms straightforward to understand
  - and (reasonably) simple to analyse
- generalise to practical applications
  - *e.g.* B-Trees for disk storage

Definition

A binary tree is an ordered collection of data

Operations

left	return the left-child of the tree
right	return the right-child of the tree
key	return the data stored at this node of a tree
parent	return the parent of the node

(and associated setters)

Operations

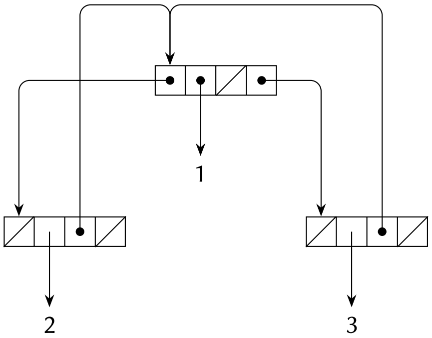
left	return the left-child of the tree
right	return the right-child of the tree
key	return the data stored at this node of a tree
parent	return the parent of the node

(and associated setters)

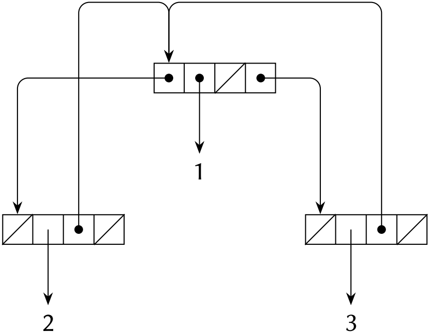
Collection operations

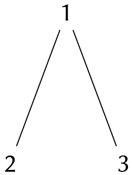
search[o]	return `true` if o is in the collection
max	return the maximum element (with respect to some ordering) of the collection
...	

Implementation



Implementation





Complexity analysis

left, right, key, parent

single pointer reads (or writes for setters)

$$\implies \Theta(1)$$

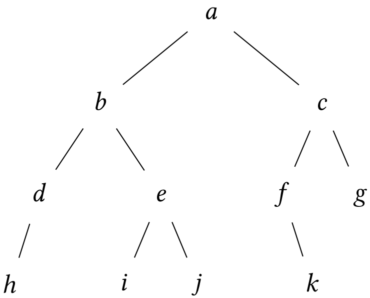
Traversal

vector	start at index zero, and visit elements in order of index until you reach the end
dynamic array	as vector
linked list	start at the head of the list, and visit the FIRST of each successive REST
binary tree	multiple possibilities!

Depth-first traversal

pre-order

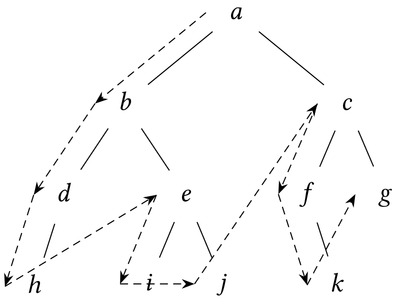
function PRE-ORDER(T)
  if ¬NULL?(T) **then**
    VISIT(T)
    PRE-ORDER(LEFT(T))
    PRE-ORDER(RIGHT(T))
  end if
end function



Depth-first traversal

pre-order

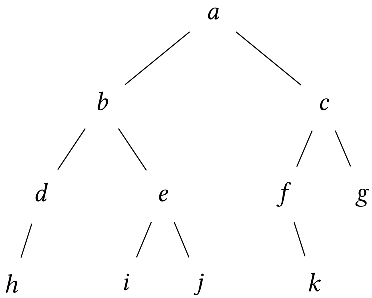
function PRE-ORDER(T)
  if ¬NULL?(T) **then**
    VISIT(T)
    PRE-ORDER(LEFT(T))
    PRE-ORDER(RIGHT(T))
  end if
end function



Depth-first traversal

post-order

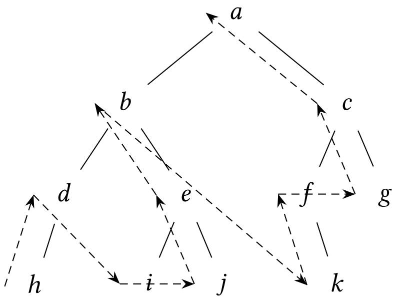
```
function POST-ORDER(T)
    if ¬NULL?(T) then
        POST-ORDER(LEFT(T))
        POST-ORDER(RIGHT(T))
        VISIT(T)
    end if
end function
```



Depth-first traversal

post-order

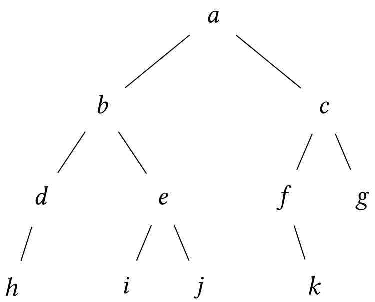
function POST-ORDER(T)
  if ¬NULL?(T) **then**
    POST-ORDER(LEFT(T))
    POST-ORDER(RIGHT(T))
    VISIT(T)
  end if
end function



Depth-first traversal

in-order

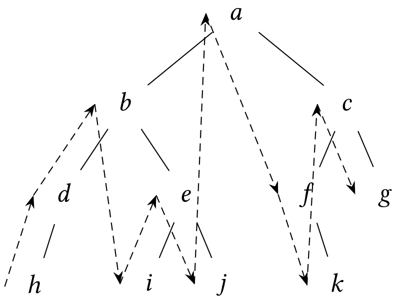
```
function IN-ORDER(T)
    if ¬NULL?(T) then
        IN-ORDER(LEFT(T))
        VISIT(T)
        IN-ORDER(RIGHT(T))
    end if
end function
```



Depth-first traversal

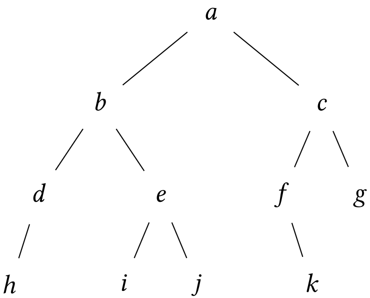
in-order

function IN-ORDER(T)
    if ¬NULL?(T) **then**
        IN-ORDER(LEFT(T))
        VISIT(T)
        IN-ORDER(RIGHT(T))
    end if
end function



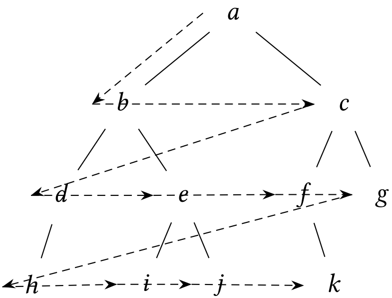
Breadth-first traversal

```
function ENQUEUE-IF!(Q,T)
    if ¬NULL?(T) then
        ENQUEUE!(Q,T)
    end if
end function
function BREADTH-FIRST(T)
    Q ← new Queue()
    ENQUEUE-IF!(Q,T)
    while ¬EMPTY?(Q) do
        t ← DEQUEUE!(Q)
        VISIT(t)
        ENQUEUE-IF!(Q,LEFT(t))
        ENQUEUE-IF!(Q,RIGHT(t))
    end while
end function
```



Breadth-first traversal

```
function ENQUEUE-IF!(Q,T)
    if ¬NULL?(T) then
        ENQUEUE!(Q,T)
    end if
end function
function BREADTH-FIRST(T)
    Q ← new Queue()
    ENQUEUE-IF!(Q,T)
    while ¬EMPTY?(Q) do
        t ← DEQUEUE!(Q)
        VISIT(t)
        ENQUEUE-IF!(Q,LEFT(t))
        ENQUEUE-IF!(Q,RIGHT(t))
    end while
end function
```



Outline

Heap property

Let x be a node in a max-heap. If y is a (generalised) parent of x, then y.key $\geq$ x.key.

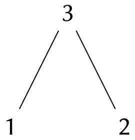
Motivation

An unordered collection for ordered keys which supports efficient construction and efficient extraction of the maximum key.

Definition

A heap is a tree data structure which both satisfies the heap contents property, and also satisfies the (nearly-)complete shape property.

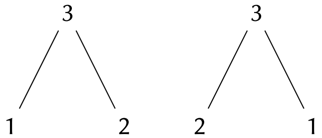
Example heaps



Definition

A heap is a tree data structure which both satisfies the heap contents property, and also satisfies the (nearly-)complete shape property.

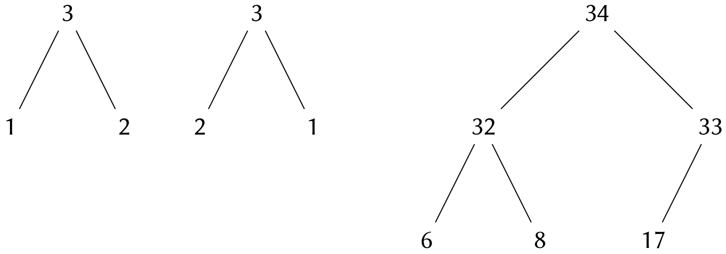
Example heaps



Definition

A heap is a tree data structure which both satisfies the heap contents property, and also satisfies the (nearly-)complete shape property.

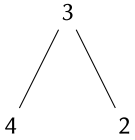
Example heaps



Definition

A heap is a tree data structure which both satisfies the heap contents property, and also satisfies the (nearly-)complete shape property.

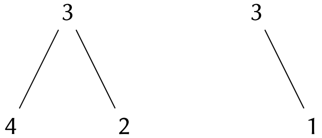
Example non-heaps



Definition

A heap is a tree data structure which both satisfies the heap contents property, and also satisfies the (nearly-)complete shape property.

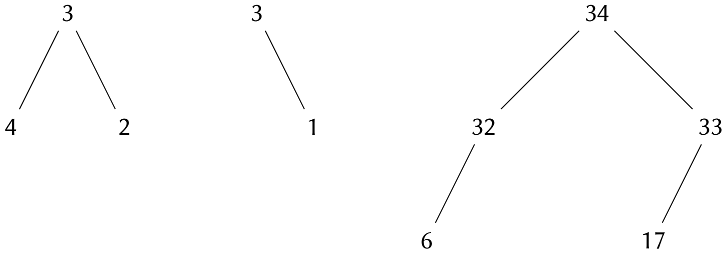
Example non-heaps



Definition

A heap is a tree data structure which both satisfies the heap contents property, and also satisfies the (nearly-)complete shape property.

Example non-heaps



Collection operations

find

```
Require: heap :: max Heap
  function FIND(heap,object)
     if NULL?(heap) then
        return false
     end if
     if heap.key = object then
        return true
     else if heap.key < object then
        return false
     else
        return FIND(heap.left,object) ∨ FIND(heap.right,object)
     end if
  end function
```

Collection operations

max

Require: heap :: non-empty max Heap
  function MAX(heap)
    return heap.key
  end function

Complexity analysis

find

must in principle go down both branches (*e.g.* to find object smaller than minimum element)

$$\implies \Theta(N)$$

Complexity analysis

find

must in principle go down both branches (*e.g.* to find object smaller than minimum element)

$$\Longrightarrow \Theta(N)$$

max

read key of root node

$$\Longrightarrow \Theta(1)$$

Work

1. Reading
   - CLRS, section 6.1
2. Questions from CLRS:

   Exercises 6.1-1, 6.1-2, 6.1-3, 6.1-4

Outline

Motivation

- interesting non-trivial data structure
- asymptotically efficient support for many operations:
  - comparison sort
  - priority queues
- component of efficient algorithms for
  - graph traversal
  - selection of k^{th} largest element

Operations

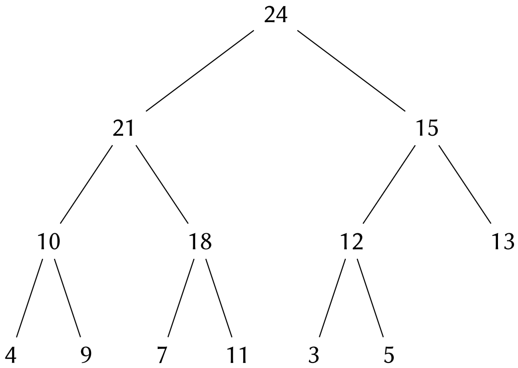
maximum	return the maximum element
extract-max!	remove and return the maximum element
insert![o]	insert the object o into the heap
size	how many elements are currently stored?

Insert

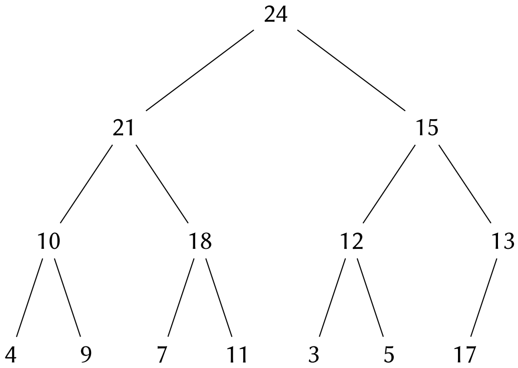
Require: heap :: Heap
  function INSERT!(heap,object)
    s ← NEXT(heap)
    p ← PARENT(s)
    while p ≠ NIL ∧ p.key < object **do**
      s.key ← p.key
      s ← p; p ← PARENT(p)
    end while
    s.key ← object
  end function

insert!

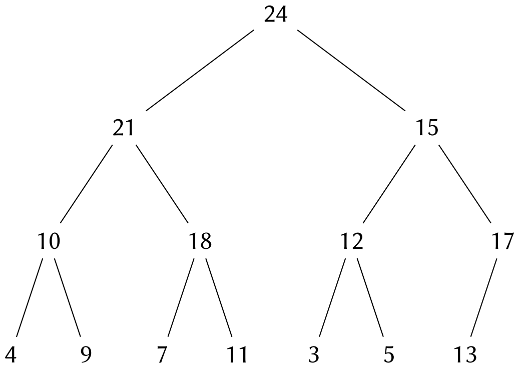
Inserting 17 to:



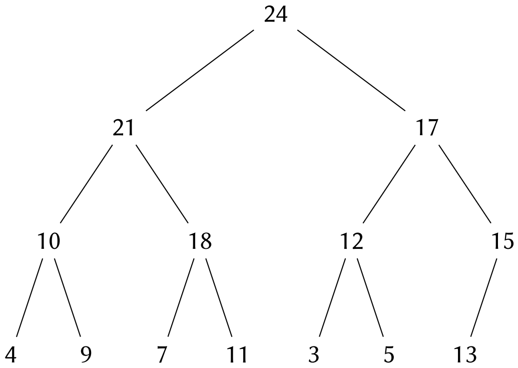
insert!



insert!



insert!



Complexity analysis

insert!

- new element goes at the bottom of the tree
- in principle could be moved up h times, with constant work each time

$$\implies \Theta(h) = \Theta(\log(N))$$

Constructing a heap incrementally

```
function MAKE-HEAP(S)
    H ← new Heap()
    for 0 ≤ i < LENGTH(S) do
        INSERT!(H,S[i])
    end for
    return H
end function
```

Complexity analysis

to build a heap with N elements, incrementally:

- each incremental addition takes $\Omega(h)$ time (h is the *current* height of the tree)
- in the worst case, there are $\frac{N}{2}$ nodes with height $\log(N)$

$$\Longrightarrow \Omega(N\log(N)), \text{ and in fact } \Theta(N\log(N)))$$

Other operations

maximum	trivial
extract-max!	see next term

Outline

Data structures

- Collections
    - Linear collections
        - Vector
        - Dynamic array
        - Linked list
        - Stack
        - Queue
        - Binary search tree
    - Hash table
    - Binary tree
    - Heap

Algorithms

- Select maximum
- Select second biggest
- Sorting
    - insertion sort, merge sort
- Recursive list algorithms
    - length, sum, remove
    - reverse
- Collision resolution
- String matching
    - naïve, Rabin-Karp, Knuth-Morris-Pratt

Theoretical techniques

- random access model
- pseudocode description of algorithms
- recursion
  - recursive expression of solutions
- recurrence relations and their solutions
- complexity analysis and big-O notation

Practical techniques

- translation of pseudocode
- algorithm measurement
- command-line practice using MinGW
- version control using git
- building software using make
- test-driven development using JUnit/CppUnit
- reading and running other people's code

Yet to come

1. more data structures!
   - priority queues
   - graphs
   - tries
   - suffix trees
2. more algorithms!
   - searching
   - sorting
   - numbers

Yet to come

1. more data structures!
    - priority queues
    - graphs
    - tries
    - suffix trees
2. more algorithms!
    - searching
    - sorting
    - numbers
3. more techniques
    - dynamic programming
    - higher-order functions
    - abstract data types
    - right tool for the job

Work

1. By 16:00 on Friday 14th December:
    - Hash table lab submission
    - String matching lab submission
    - Binary trees quiz
    - Module evaluation survey

Work

1. By 16:00 on Friday 14th December:
    - Hash table lab submission
    - String matching lab submission
    - Binary trees quiz
    - Module evaluation survey
2. Over the Christmas break:
    - any reading (CLRS, DPV, Drozdek) given in lectures that you haven't yet done
    - any exercises from textbooks that you haven't yet attempted
    - go over quizzes and lab exercises. Have another go at any labs you didn't finish
    - read feedback on ListVisualiser exercise. If you didn't implement cycle detection, give it a go.

Work

1. By 16:00 on Friday 14th December:
    - Hash table lab submission
    - String matching lab submission
    - Binary trees quiz
    - Module evaluation survey
2. Over the Christmas break:
    - any reading (CLRS, DPV, Drozdek) given in lectures that you haven't yet done
    - any exercises from textbooks that you haven't yet attempted
    - go over quizzes and lab exercises. Have another go at any labs you didn't finish
    - read feedback on ListVisualiser exercise. If you didn't implement cycle detection, give it a go.
3. Monday 14th January 2019, 10:00-12:00
    - off we go again!