Data Structures

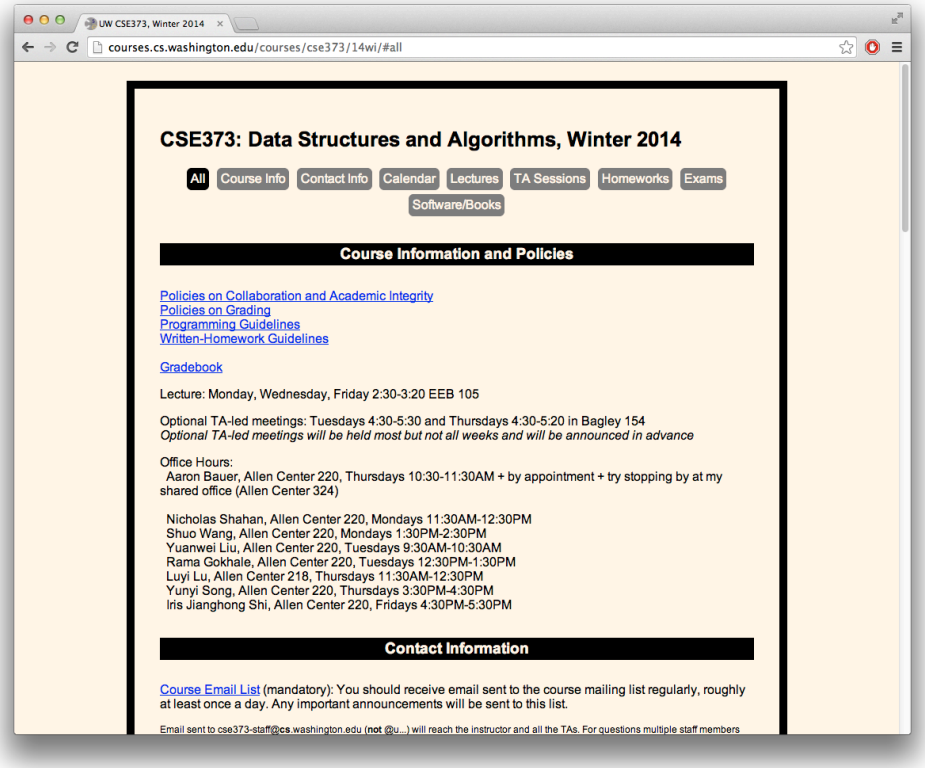
Kathleen Tuite

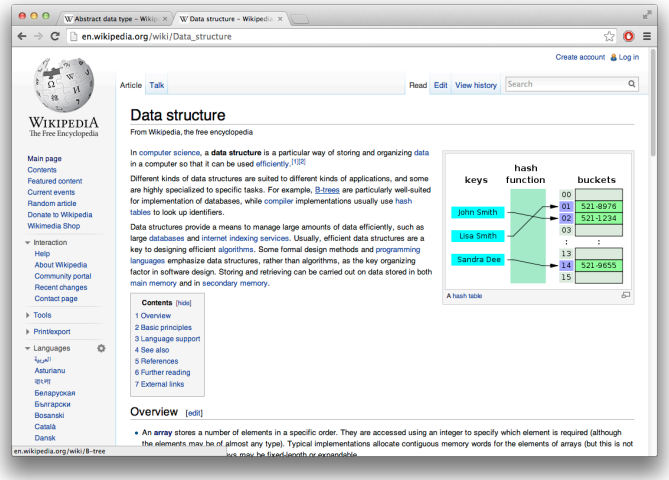
March 27, 2014

Ada Developers Academy

Data structures in the classroom

- 1 course → 1 hour
- You've used a bunch before
- Can learn more on Wikipedia





http://courses.cs.washington.edu/courses/cse373/14wi/

Goals of the talk

- Familiarity with more data structures
  - What they're called
  - How they work
  - What they're used for
- How to figure out more
- Inspiration for new ways to work with data

Things to know about data structures

- How it works
- Why it matters
- Example use cases

- Running times / algorithmic complexity of specific operations
  - E.g. Hash tables have constant time O(1) insertion/lookup! They're so cool!

Records / Structs

- Simple data structures with fields
- Fields can be:
  - Primitives (ints, strings, floats, etc.)
  - References to other records
- Use a class for a struct
  - Classes have methods and structs don't
  - Structs have fixed set of attributes while you can add new keys to a hash


```
Cat = Struct.new(:name, :color, :disposition)
```

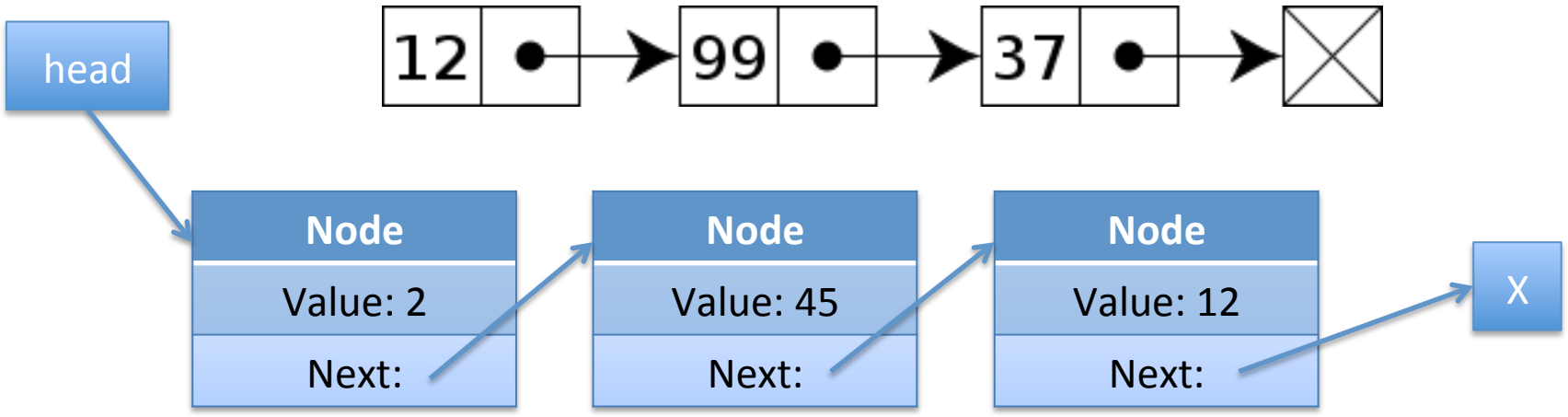
```
Cat.new("JPEG", "Orange Tabby", "Snuggly")
Cat.new("MPEG", "Calico", "Awkward cuddler")
```

http://en.wikipedia.org/wiki/Record_(computer_science)

Linked Lists

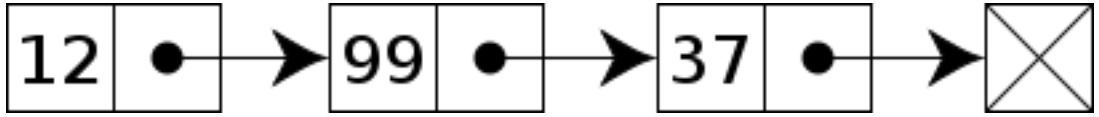
Interview alert:
- Get the K-th node from the end of a linked list
- Reverse a linked list
- Etc.

- An easy thing to build out of records
- A useful building block for other data structures
- Stack overflow has a bunch of good reasons why they're useful

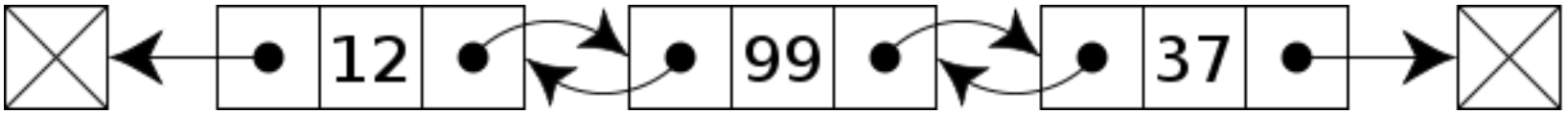


http://stackoverflow.com/questions/2429217/under-what-circumstances-are-linked-lists-useful

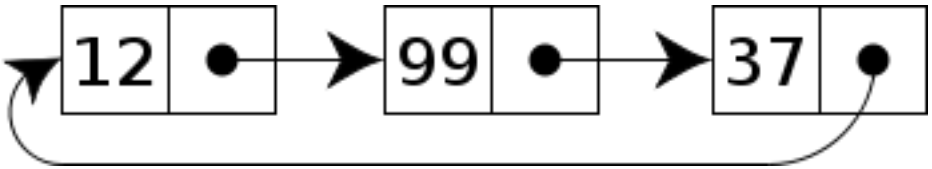
Different Flavors of Linked Lists



Linked list



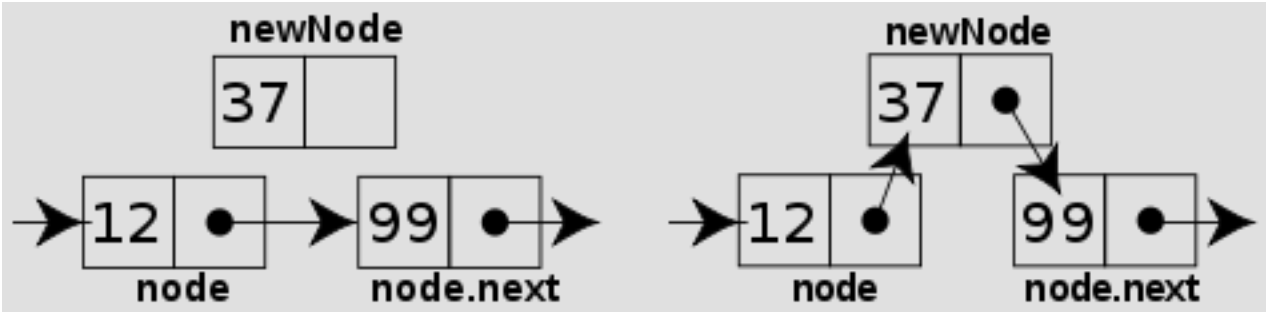
Doubly linked list

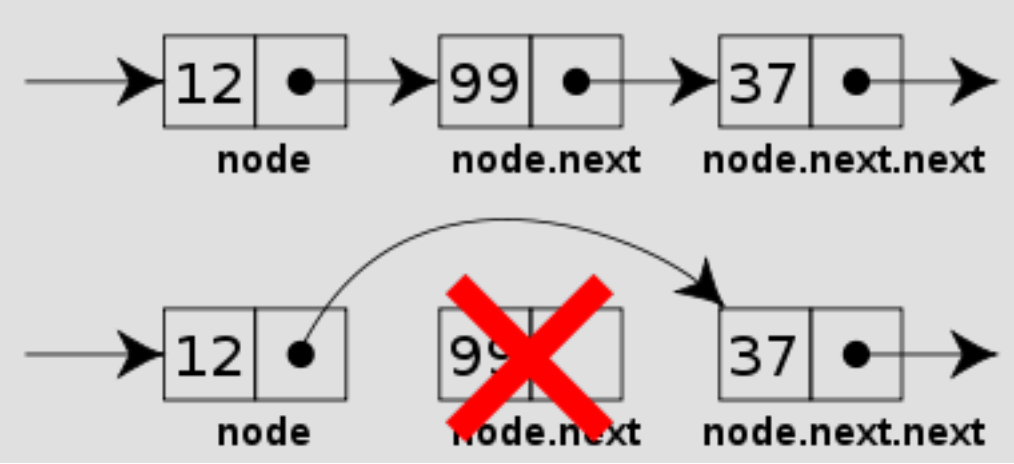


Circularly linked list

Stuff you can do to a Linked List

(wikipedia says it best)





Linked List Ruby Code

```ruby
# Quick Example of a Self Referential Data Structure in Ruby
# NODE -> contains a value and a pointer to (next_node)
# LinkedList -> This class holds the linked list functions - adding a node, traversing and displaying the linked list

class Node

    attr_accessor :value, :next_node

    def initialize val,next_in_line
        @value = val
        @next_nodex = next_in_line
        puts "Initialized a Node with value:  " + value.to_s
    end
end
```

http://www.thelearningpoint.net/computer-science/basic-data-structures-in-ruby---linked-list---a-simple-singly-linked-list

Linked List Ruby Code (cont.)

```ruby
class LinkedList

    def initialize val
        # Initialize a new node at the head
        @head = Node.new(val,nil)
    end

    def add(value)
        # Traverse to the end of the list
        # And insert a new node over there with the specified value
        current = @head
        while current.next_node != nil
            current = current.next_node
        end
        current.next_node = Node.new(value,nil)
        self
    end
```

... Also has methods to display the list and delete a specific value from the list

```ruby
# Initializing a Linked List with a node containing value (5)
ll = LinkedList.new(5)

# Adding Elements to Linked List
ll.add(10)
ll.add(20)

# Display the Linked List
puts "Displaying Linked List:"
ll.display

puts "Delete 10 and then display the linked list:"
ll.delete(10)
ll.display

=begin
Output:
Initialized a Node with value:  5
Initialized a Node with value:  10
Initialized a Node with value:  20
Displaying Linked List:
5->10->20
Delete 10 and then display the linked list:
5->20
=end
```

Linked List Ruby Code (output)

Stacks and Queues

Abstract stacks and queues: list of elements + operations

Stacks

Last in first out (LIFO)

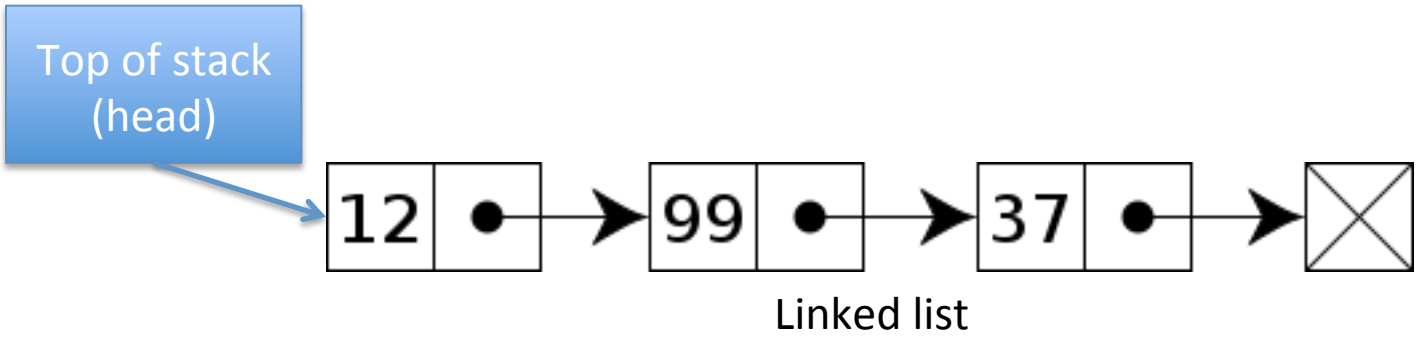
- Push
- Pop
- isEmpty

Queues

First in first out (FIFO)

- Enqueue
- Dequeue
- isEmpty

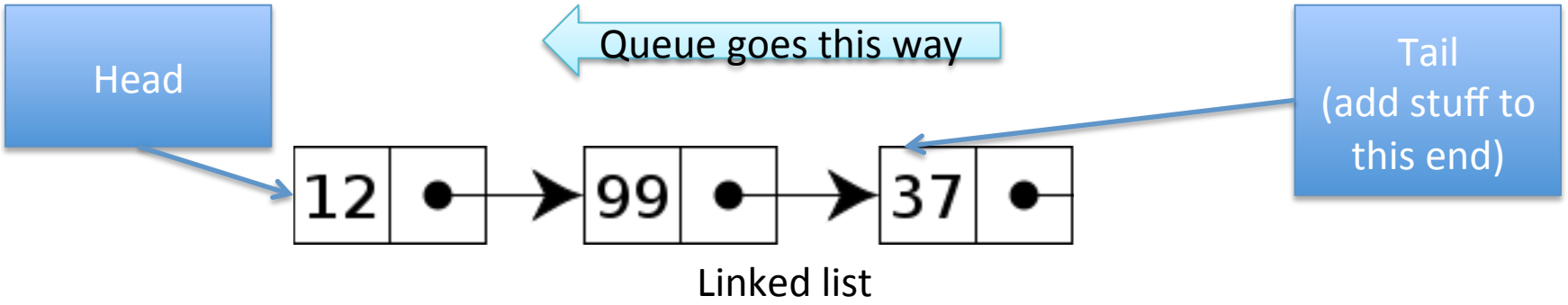
Making stacks with linked lists



Linked list

- push(new_node)
  new_node.next ← head
  head ← new_node
- pop()
  node_to_return ← head
  head ← head.next

Making queues with linked lists



- enqueue(new_node)
  new_node.next ← tail
  tail ← new_node
- dequeue()
  node_to_return ← head
  head ← head.next

Lots of different ways to implement stacks and queues

- Built into many languages
  - Stack and queue operations available on arrays
- Always the same operations
  - Push/pop
  - Enqueue/dequeue

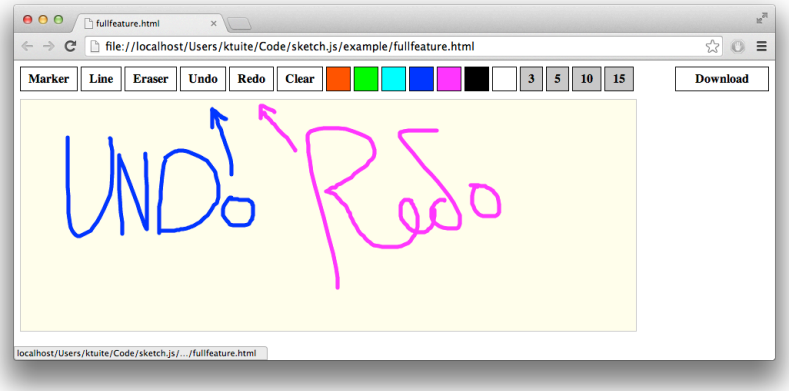
```
// JavaScript code

var stack = [];
stack.push(2);       // stack is now [2]
stack.push(5);       // stack is now [2, 5]
var i = stack.pop(); // stack is now [2]
console.log(i);            // displays 5

var queue = [];
queue.push(2);         // queue is now [2]
queue.push(5);         // queue is now [2, 5]
var i = queue.shift(); // queue is now [5]
console.log(i);              // displays 2
```

Stack/queue running times:
Enqueue, dequeue, push, pop should all run in O(1) constant time

Using stacks for drawing history undo/ redo



https://github.com/ktuite/sketch.js

Action: list of line segments and pen style (color + width)

Undo/Redo: Pop off one stack and push onto the other

```
// coffeescript code

@actions = [] // stuff that has been drawn
@undone = []  // stuff that has been undone
              // that we might want to redo

# ### sketch.operation(mode)
#
# mode="undo" Pop one action off the actions array
and put it in undone array
# mode="redo" Put undone action back into actions
# mode="clear" Clear by emptying actions

operation: (mode)->
  if mode is "undo" and @actions
    @undone.push @actions.pop()

  else if mode is "redo" and @undone
    @actions.push @undone.pop()

  else if mode is "clear"
    @undone = []
    @actions = []

  @redraw()
```

Using a queue for face detection worker queue

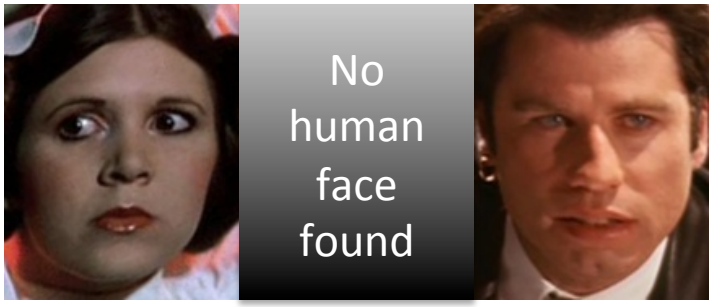
...

Queue of commands that my web server sends to face detection service

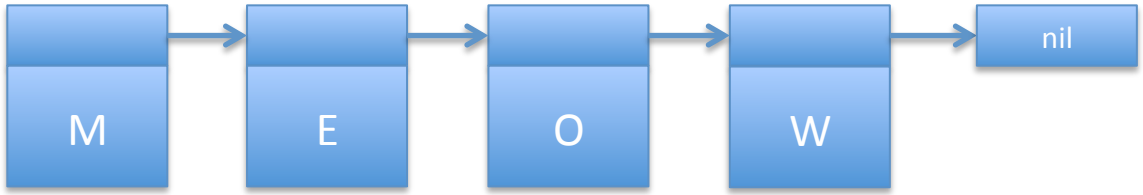
Face detection service

Command: find face in file /path/123.jpg

...



Arrays (versus linked lists)



Linked List

- Easier for some types of manipulation
- Variable size (made of small pieces)
- Takes up more space in memory

M	E	O	W	_	_

Array

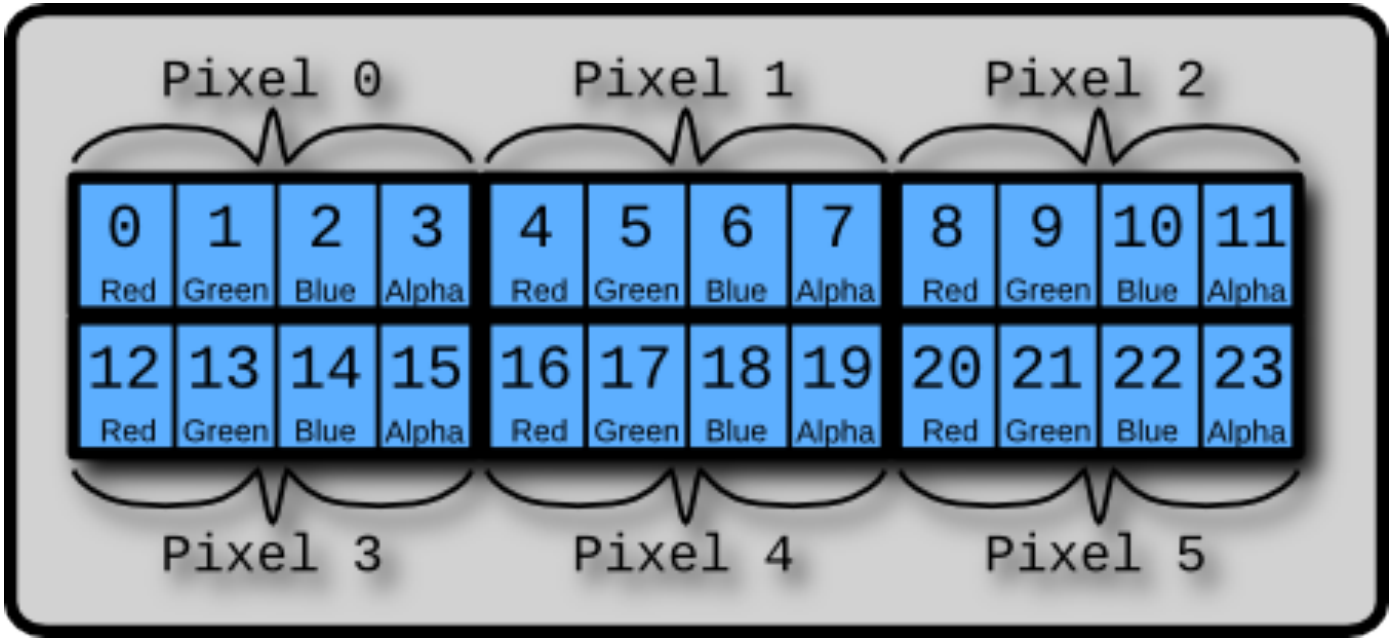
- Harder for some types of manipulation
- Direct access with integer index
- Fixed size
- Takes up less space in memory

```
my_list[3] == "w"
```

Images are arrays of pixels

- A pixel is a record/struct of colors

```
pixels[i,j].red == 255
```



Images are arrays of pixels

```
stroopwafel:Desktop ktuite$ convert  -size 100x100 gradient:blue-red  gradient.bmp
stroopwafel:Desktop ktuite$ hexdump -C gradient.bmp
00000000  42 4d ba 75 00 00 00 00  00 00 8a 00 00 00 7c 00  |BM.u..........|.|
00000010  00 00 64 00 00 00 64 00  00 00 01 00 18 00 00 00  |..d...d.......|
00000020  00 00 30 75 00 00 00 00  00 00 00 00 00 00 00 00  |..0u............|
00000030  00 00 00 00 00 00 00 00  ff 00 00 ff 00 00 ff 00  |................|
00000040  00 00 00 00 00 ff 42 47  52 73 80 c2 f5 28 60 b8  |......BGRs...(`.|
00000050  1e 15 20 85 eb 01 40 33  33 13 80 66 66 26 40 66  |.. ...@33..ff&@f|
00000060  66 06 a0 99 99 09 3c 0a  d7 03 24 5c 8f 32 00 00  |f.....<...$\.2..|
00000070  00 00 00 00 00 00 00 00  00 00 04 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 ff 00 00 ff  |................|
00000090  00 00 ff 00 00 ff 00 00  ff 00 00 ff 00 00 ff 00  |................|
000000a0  00 ff 00 00 ff 00 00 ff  00 00 ff 00 00 ff 00 00  |................|
000000b0  ff 00 00 ff 00 00 ff 00  00 ff 00 00 ff 00 00 ff  |................|
000000c0  00 00 ff 00 00 ff 00 00  ff 00 00 ff 00 00 ff 00  |................|
000000d0  00 ff 00 00 ff 00 00 ff  00 00 ff 00 00 ff 00 00  |................|
000000e0  ff 00 00 ff 00 00 ff 00  00 ff 00 00 ff 00 00 ff  |................|
000000f0  00 00 ff 00 00 ff 00 00  ff 00 00 ff 00 00 ff 00  |................|
00000100  00 ff 00 00 ff 00 00 ff  00 00 ff 00 00 ff 00 00  |................|
00000110  ff 00 00 ff 00 00 ff 00  00 ff 00 00 ff 00 00 ff  |................|
00000120  00 00 ff 00 00 ff 00 00  ff 00 00 ff 00 00 ff 00  |................|
00000130  00 ff 00 00 ff 00 00 ff  00 00 ff 00 00 ff 00 00  |................|
00000140  ff 00 00 ff 00 00 ff 00  00 ff 00 00 ff 00 00 ff  |................|
00000150  00 00 ff 00 00 ff 00 00  ff 00 00 ff 00 00 ff 00  |................|
00000160  00 ff 00 00 ff 00 00 ff  00 00 ff 00 00 ff 00 00  |................|
00000170  ff 00 00 ff 00 00 ff 00  00 ff 00 00 ff 00 00 ff  |................|
00000180  00 00 ff 00 00 ff 00 00  ff 00 00 ff 00 00 ff 00  |................|
```

red green blue

Mappings

Maps are another abstract data type with different ways of being implemented

```
my_list[0] == “m”
my_list[1] == “e”
my_list[2] == “o”
my_list[3] == “w”
```

Mapping integers to strings

```
ages[“keiko”] == 10
ages[“mpeg”] == 3
```

Mapping strings to integers


```
my_list[3] == “w”
```

How do you go over “keiko” steps into an array or list???

Need a different data structure

- One that maps arbitrary things onto other arbitrary things

- Ruby hashes

```
options = { :font_size => 10, :font_family => "Arial" }
options[:font_size]  # => 10
```

- How do you make one?

Hash Functions

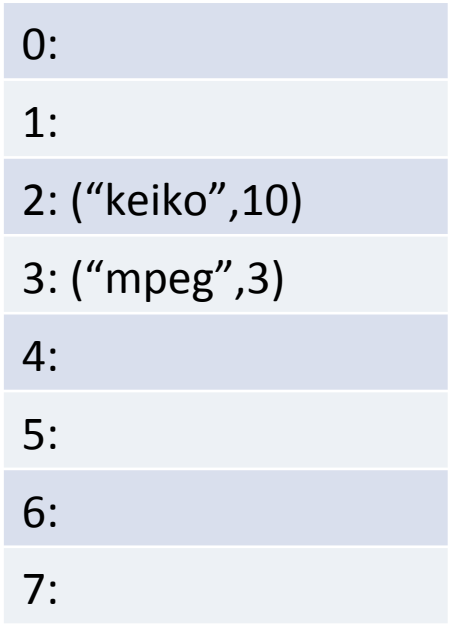
used in to hash passwords so the password isn't saved/sent in plain text

```
stroopwafel:server ktuite$ irb
1.9.3-p484 :001 > "keiko".hash
 => 3638129426109475698
1.9.3-p484 :002 > "keiko".hash % 8
 => 2
1.9.3-p484 :003 > "mpeg".hash % 8
 => 3
```

Hash Tables

Use hash function to store arbitrary kinds of mappings

```
stroopwafel:server ktuite$ irb
1.9.3-p484 :001 > "keiko".hash
 => 3638129426109475698
1.9.3-p484 :002 > "keiko".hash % 8
 => 2
1.9.3-p484 :003 > "mpeg".hash % 8
 => 3
```

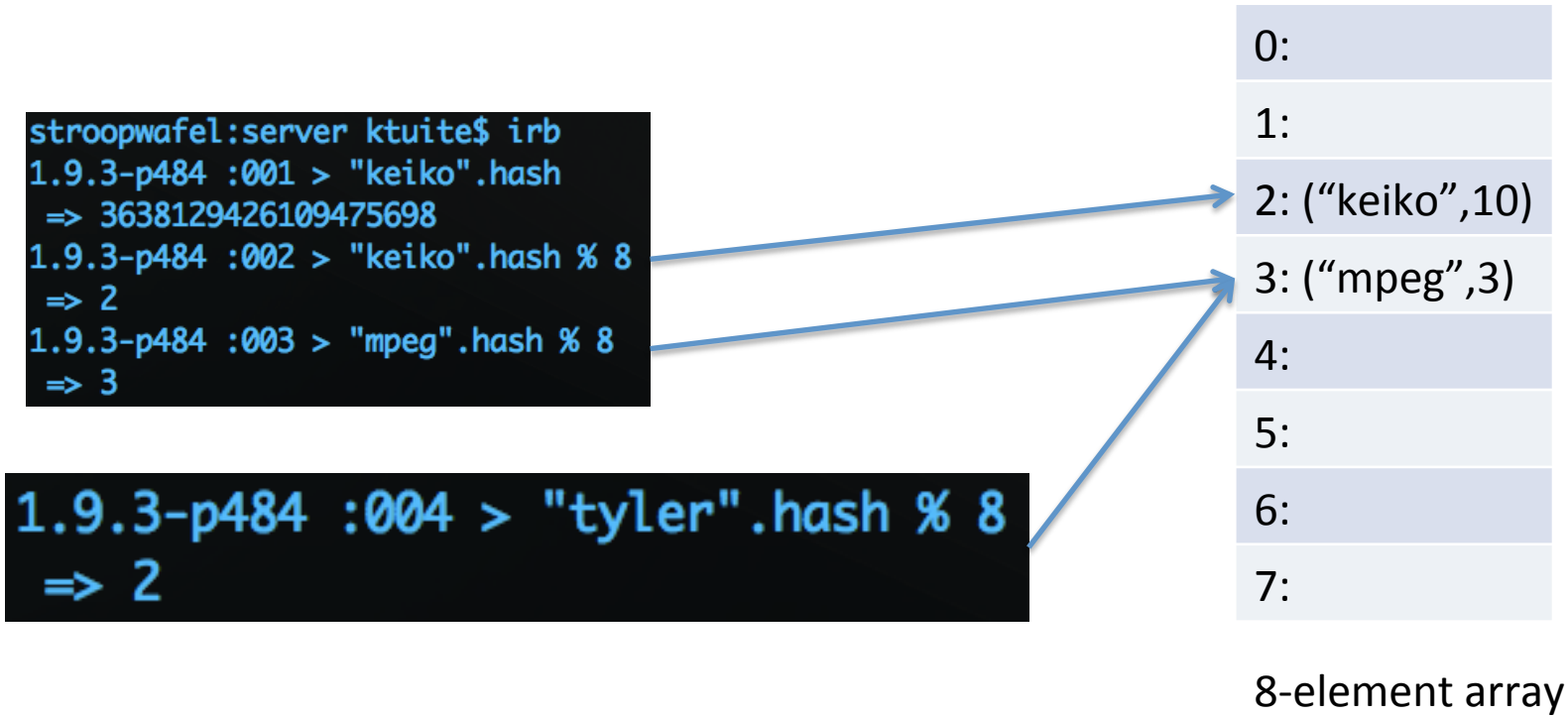


8-element array

```
ages[“keiko”] == 10
ages[“mpeg”] == 3
```

Important: Constant-time lookup/insert – O(1)

Hash Table Collisions

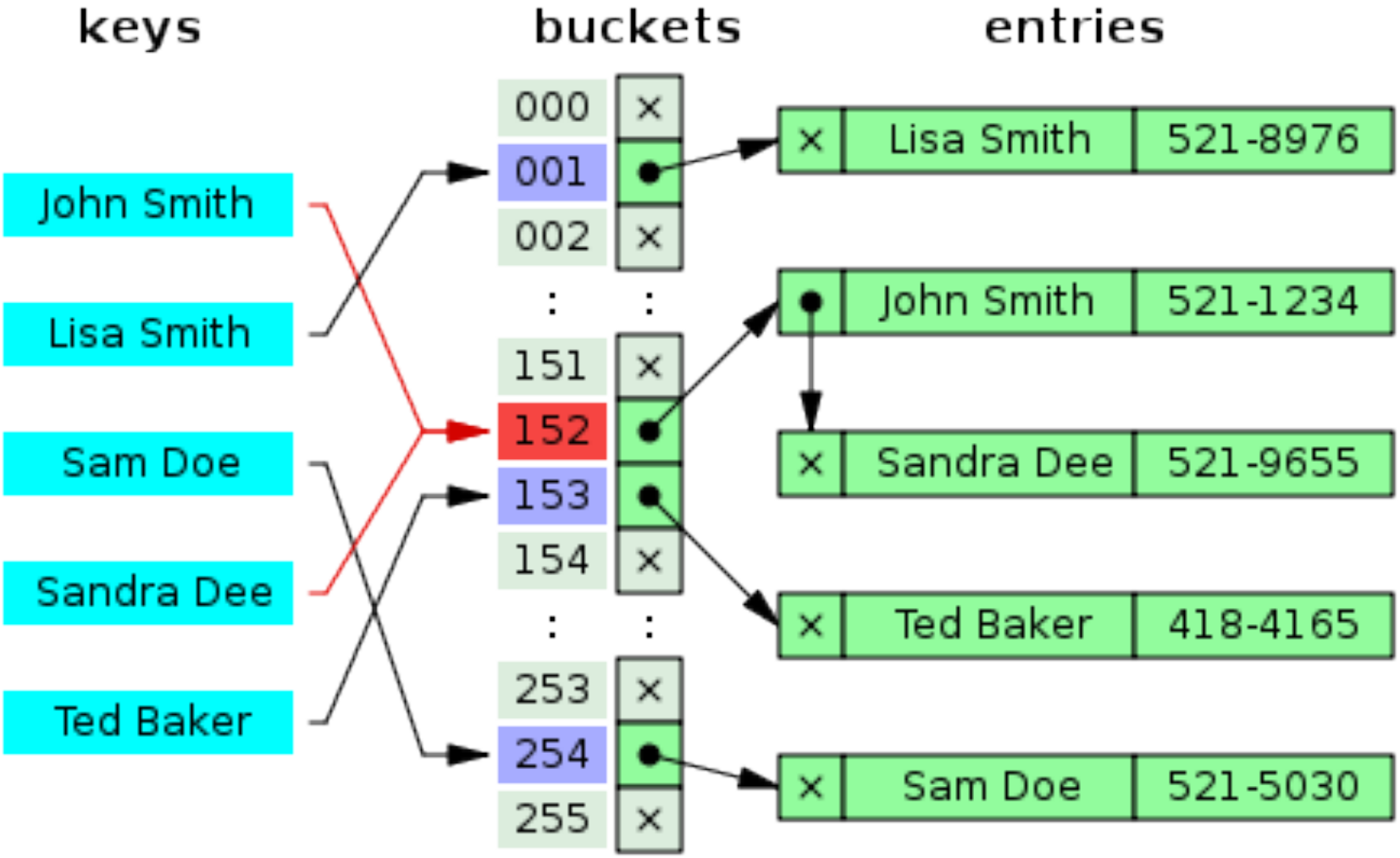


What if two strings map to the same location in the array???

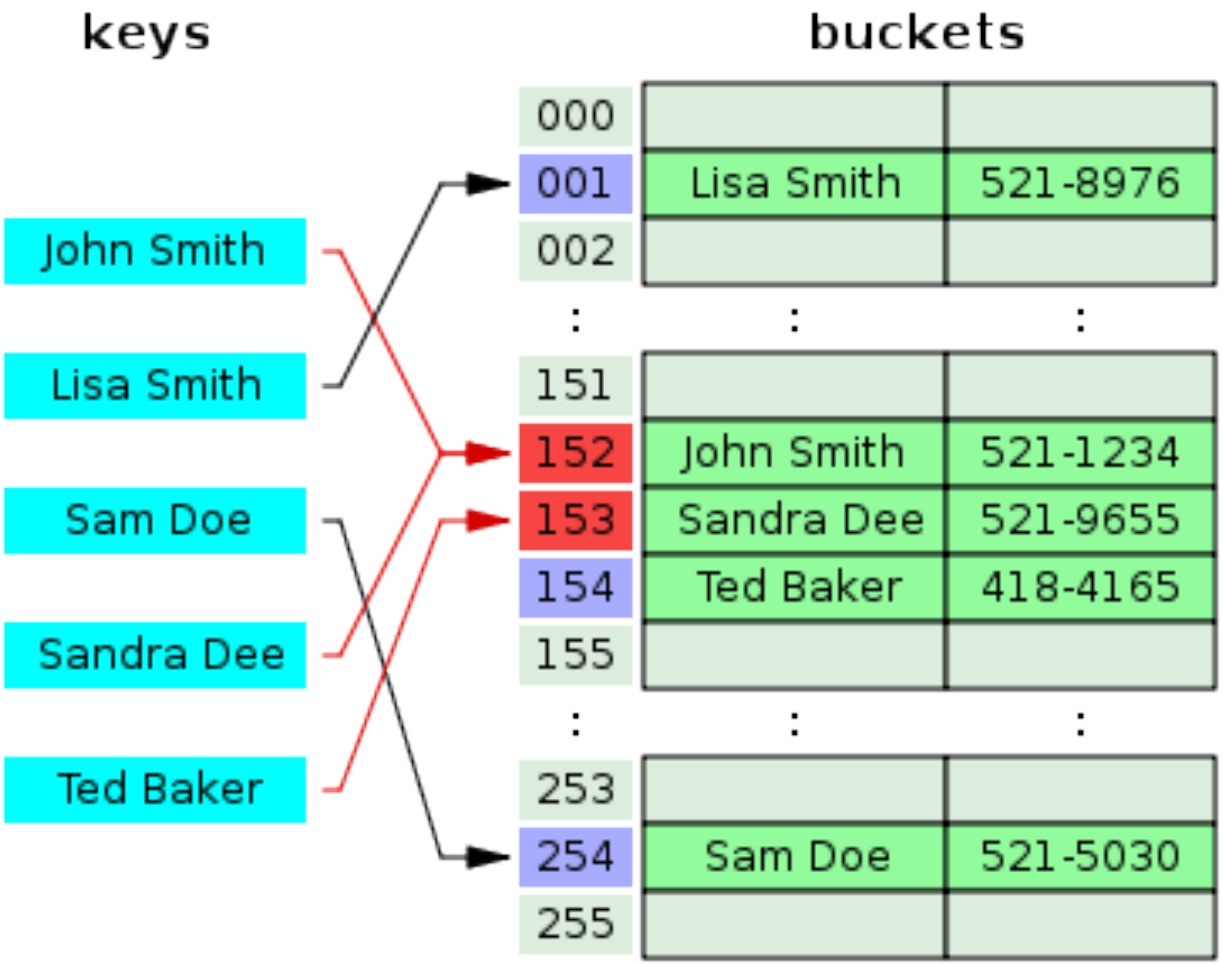
Collision Resolution Strategies

- "Separate chaining"
  - Each bucket contains a list (linked list?)
- "Open addressing"
  - Linear probing (look for next free space)
  - Double hashing (hash something else to get next place to look)

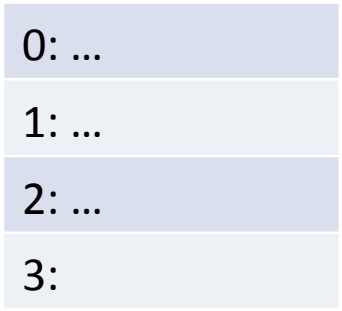
Separate Chaining



Open Addressing



Resizing a hash table



4-element array

8-element array

When occupancy approaches 75%,
build a new hash table of double size.
Insert all old mappings into new table.

When are hash tables used?

- Any time you used a “hash” in Ruby
- Any time you used an object in Ruby

- Ruby hashes come from Perl hashes where they were implemented with hash tables.

- In Python, they are called “dicts” (dictionaries)

Jets and Jet Planes

Has propellers, not a “jet”.

The actual jet of air.

Other Mapping Structures

Interface Map<K,V> (in Java)

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Known Subinterfaces:

Bindings, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>, LogicalMessageContext, MessageContext, NavigableMap<K,V>, SOAPMessageContext, SortedMap<K,V>

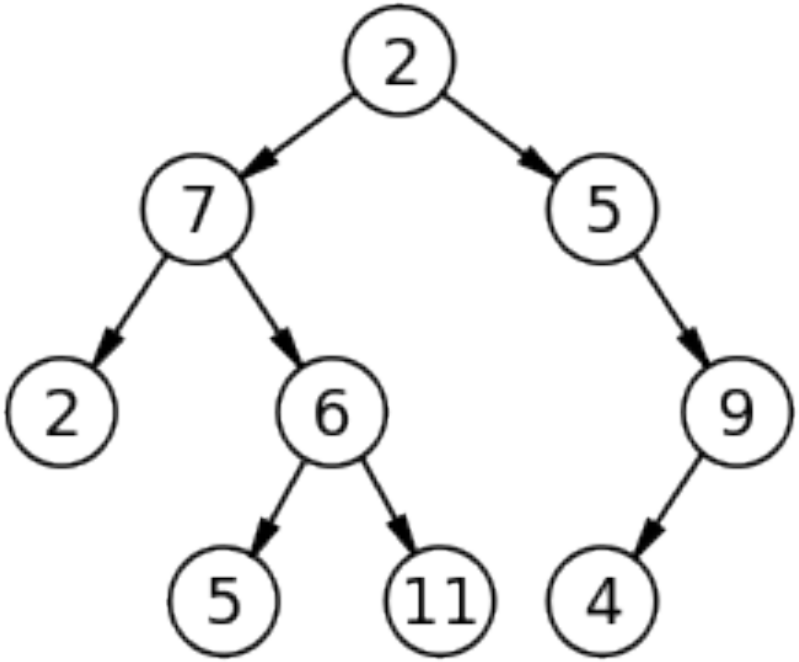
All Known Implementing Classes:

AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap

Abstract data type versus **data structures**

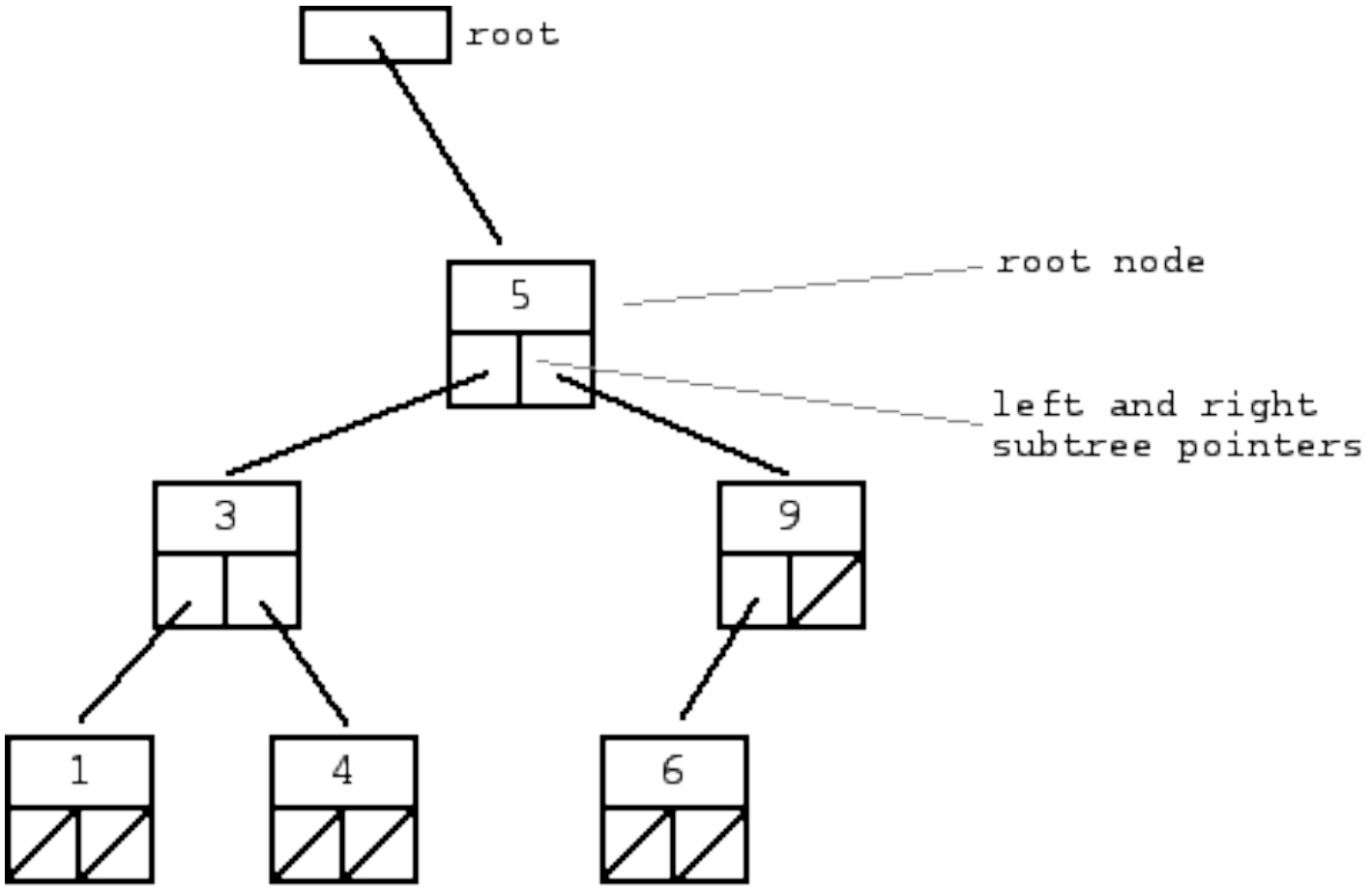
Trees

The root

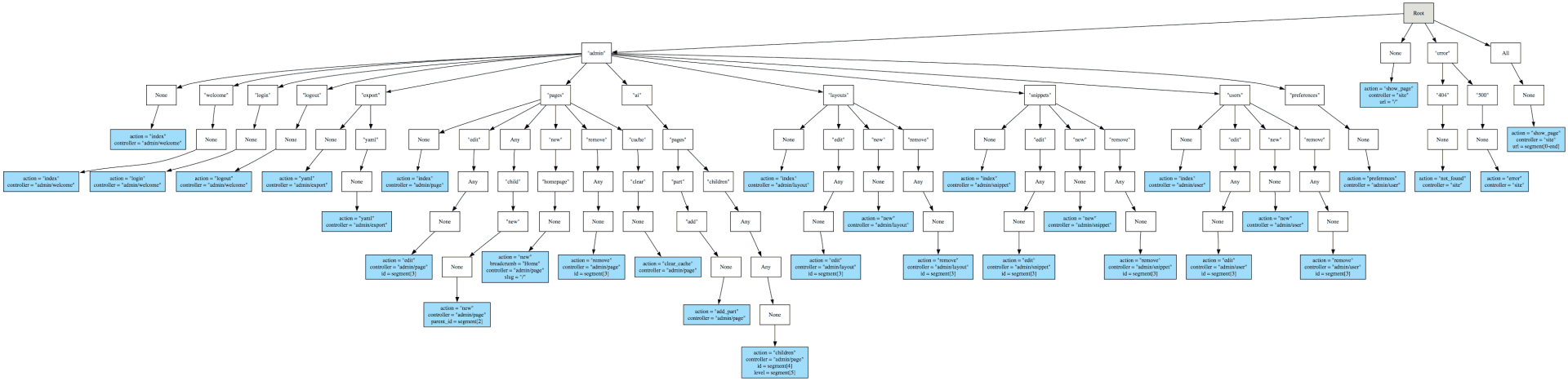


The leaves

Binary Trees

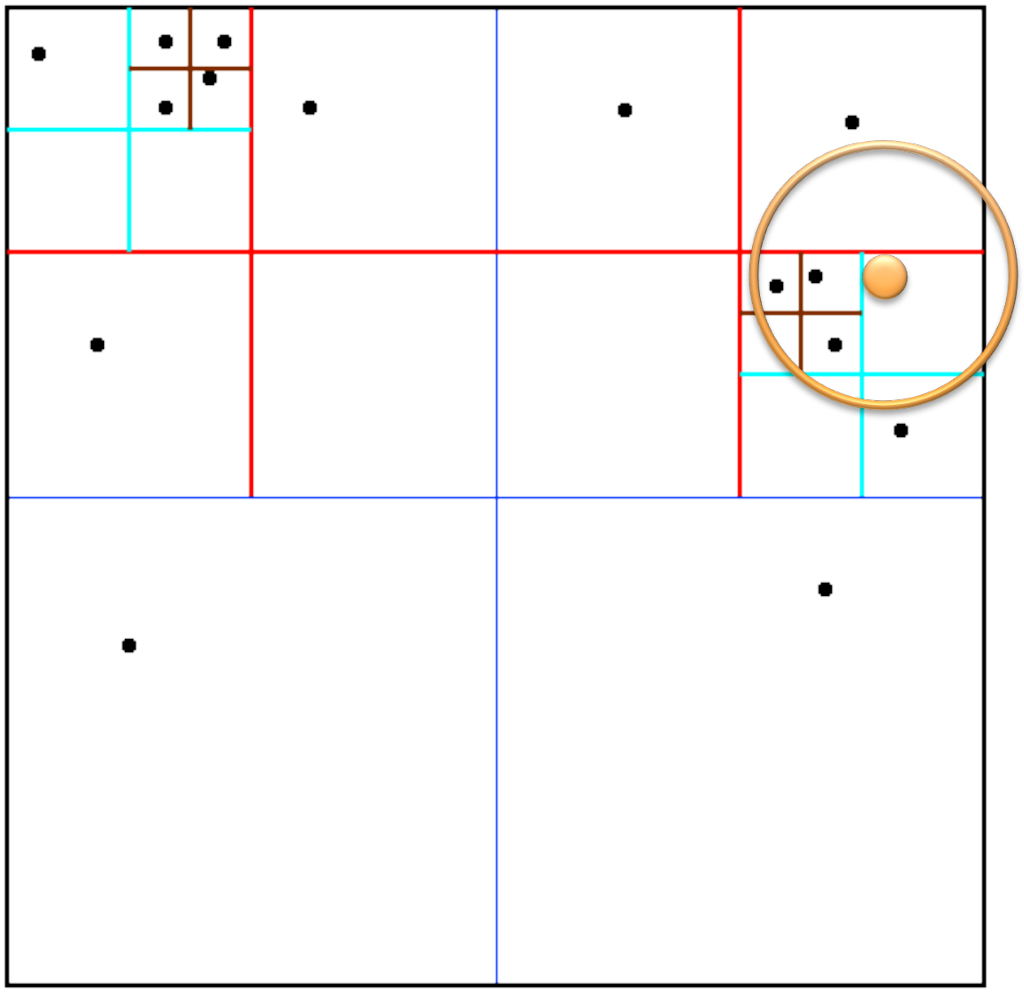


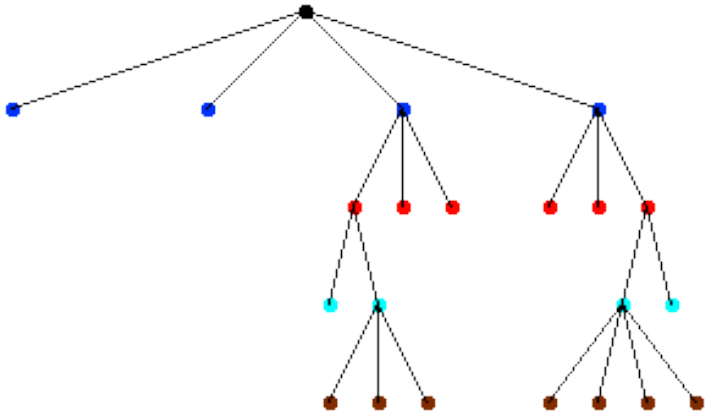
General Trees



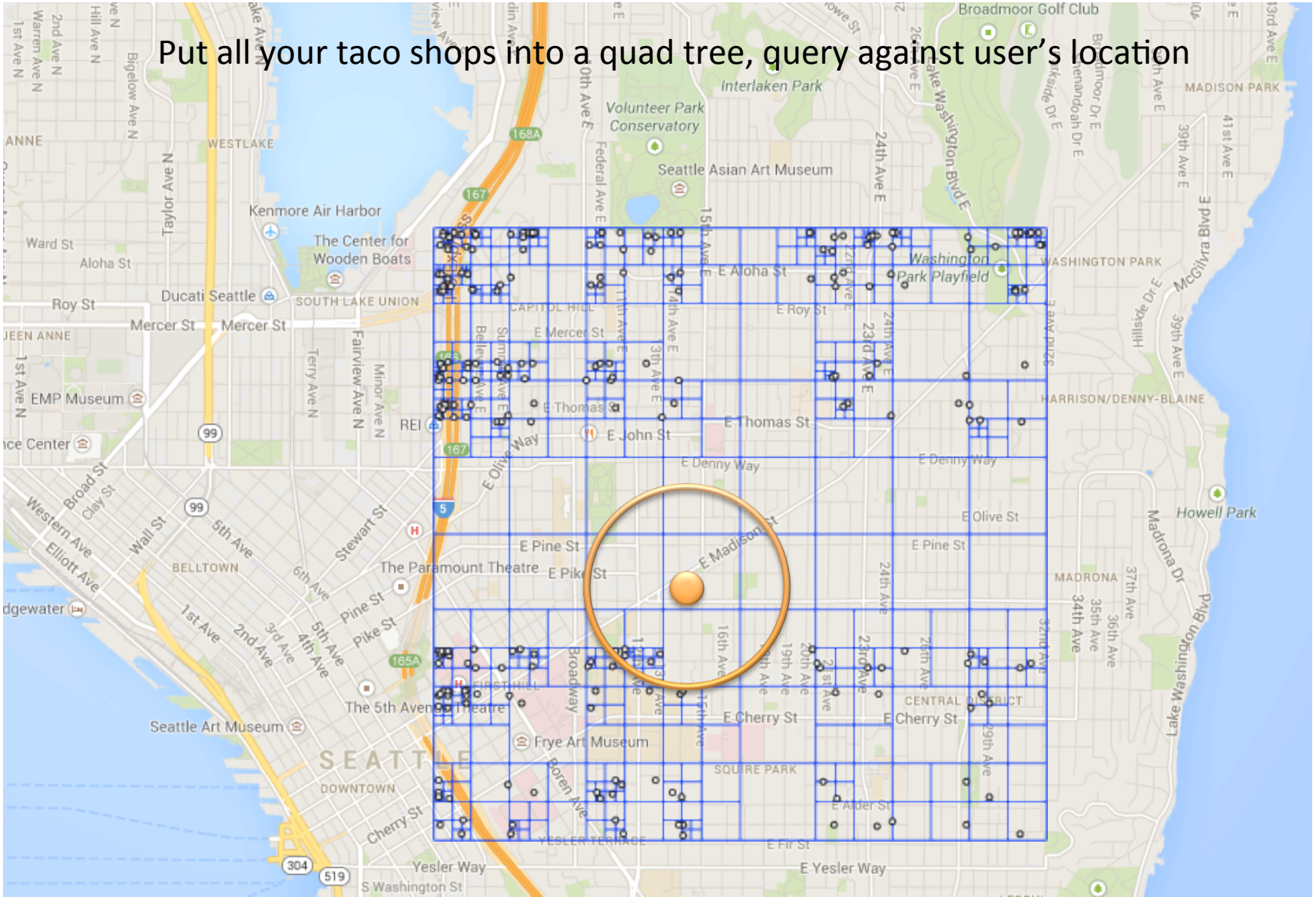
Variable number of children

Quadtrees for Spatial Partitioning



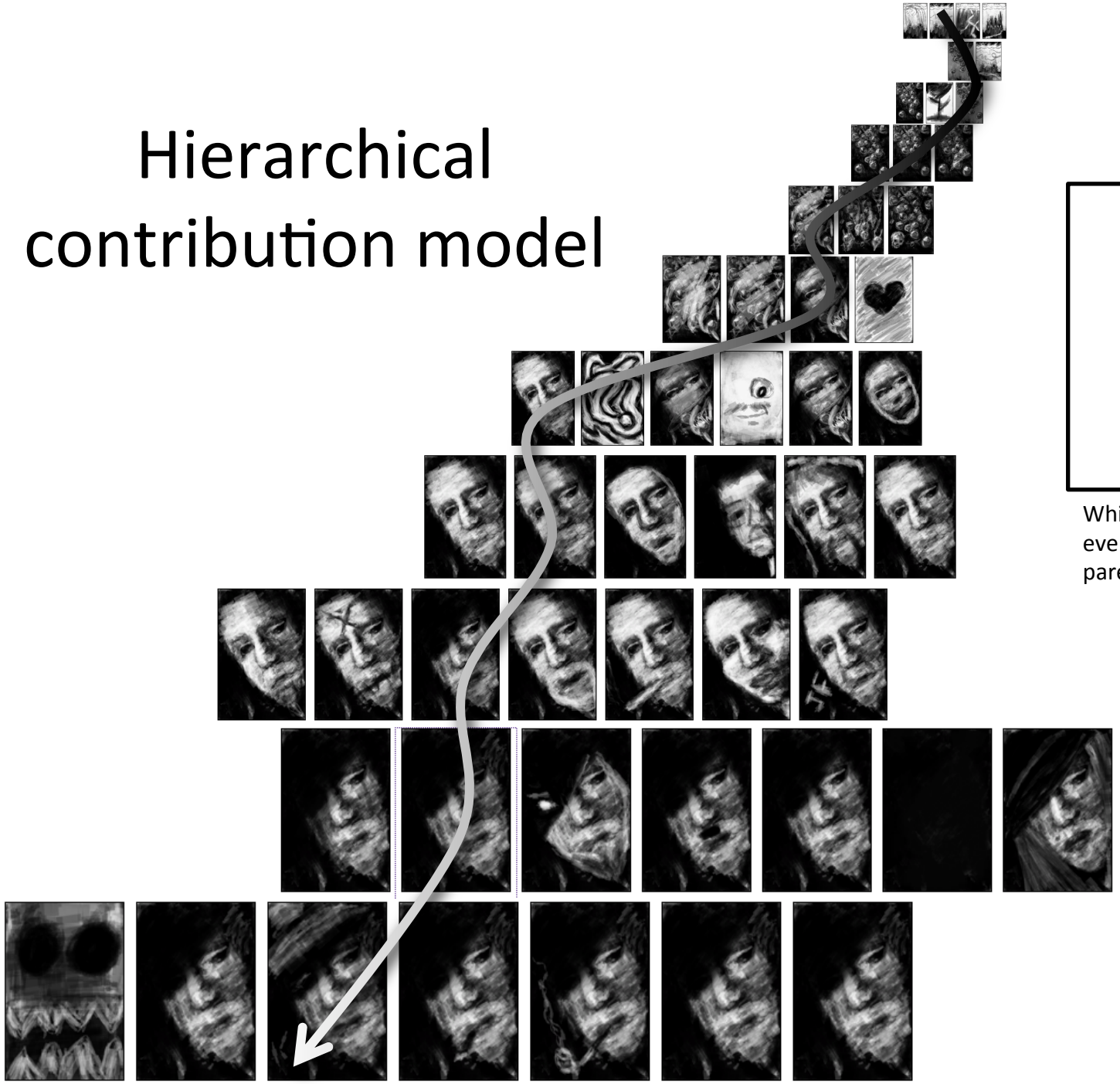


Taco 911 without Google APIs



http://sythirra.deviantart.com/art/What-should-I-draw-today-180310724

Hierarchical contribution model



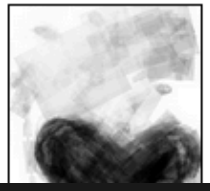
White canvas: every sketch's parent

Sketch-a-bit

Instant Inspiration

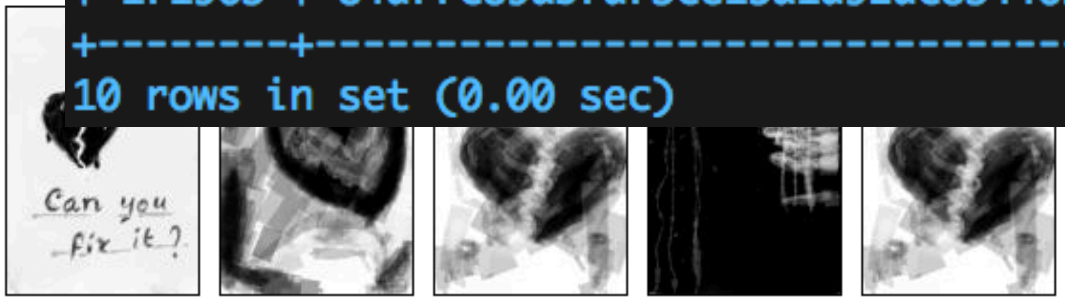
Sketch ID: 2120

Evolved from...



```
mysql> select id, name, parent from sketches order by id desc limit 10;
+--------+----------------------------------+--------+
| id     | name                             | parent |
+--------+----------------------------------+--------+
| 171972 | 209f8dffeea2bb94bad40cd80684443d |      0 |
| 171971 | 30622e8808798a0bb8292fabb50aeae7 |  10181 |
| 171970 | 36db876b6ac4d94f01e553d244b8b358 |  86766 |
| 171969 | a4dbe0b89393df9e886b67eb4b6de770 |  26131 |
| 171968 | 1cbaebb33ea6bdc73e9ac77032ac39fb |  73248 |
| 171967 | 5c5cf34263eca7634db66fd938396afb | 169893 |
| 171966 | 117c2362b2ed77b4ef3c5e7066e7240f | 143843 |
| 171965 | 90cbc9a19e78fcd8eb2cc6dc34eb851d | 158719 |
| 171964 | 27c6419dede9f6812df8e5d4aa5551d4 |  41974 |
| 171963 | 84d77c89ab7d73ce15a1d92dc8344692 | 123456 |
+--------+----------------------------------+--------+
10 rows in set (0.00 sec)
```

Child

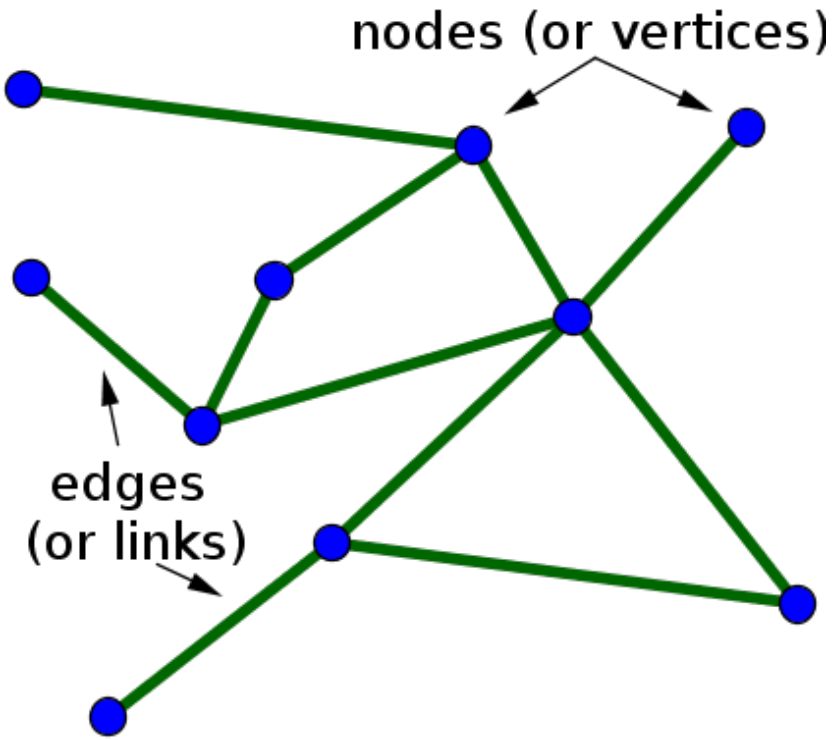


Sketch-a-bit Android App: Made by Kathleen and Adam.

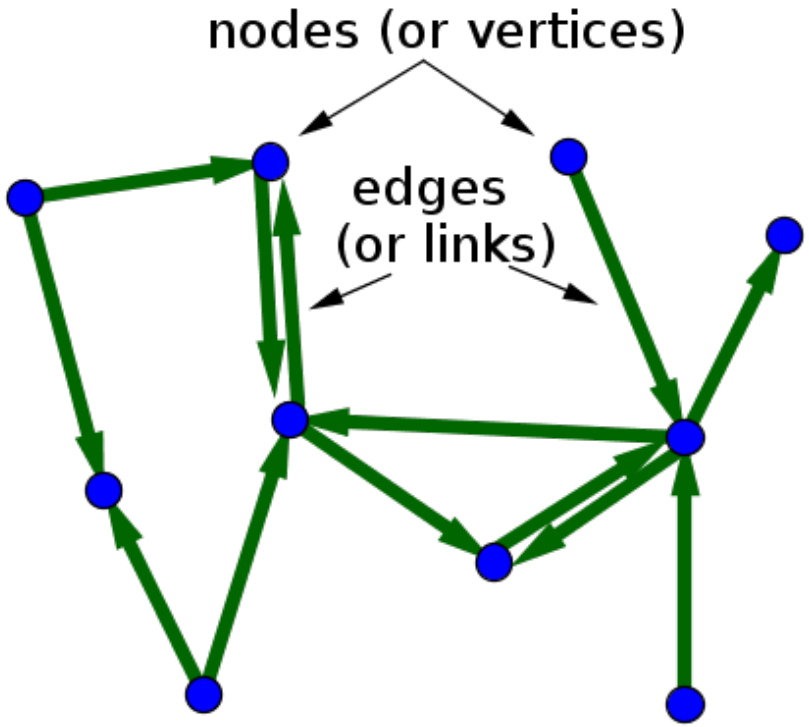
Graphs

It’s a chart!

Graphs (networks)



undirected



directed

Trees and lists are just special kinds of graphs!

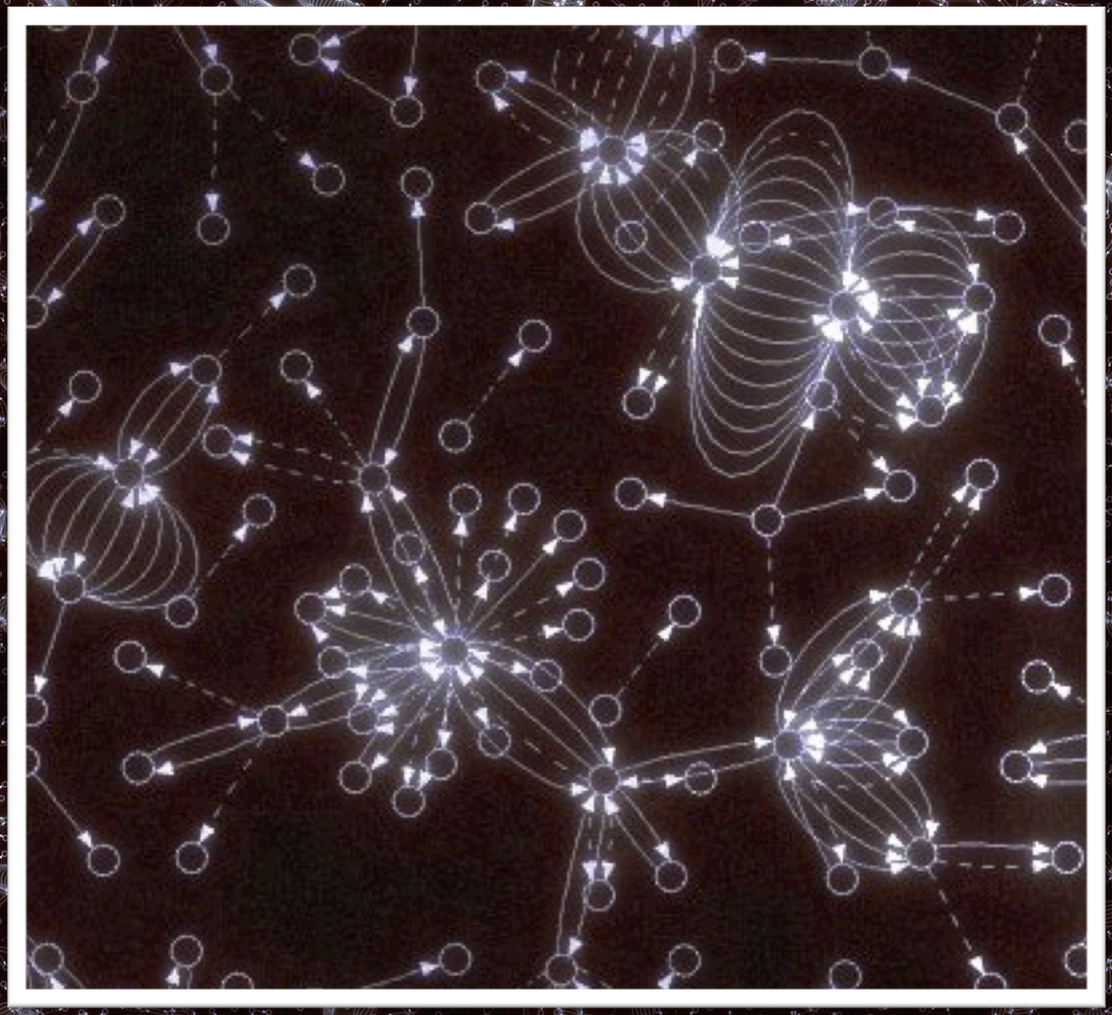
Graphs

- List of nodes & list of edges between nodes

```
ply
format ascii 1.0           { ascii/binary, format version number }
comment made by Greg Turk  { comments keyword specified, like all lines }
comment this file is a cube
element vertex 8           { define "vertex" element, 8 of them in file }
property float x           { vertex contains float "x" coordinate }
property float y           { y coordinate is also a vertex property }
property float z           { z coordinate, too }
element face 6             { there are 6 "face" elements in the file }
property list uchar int vertex_index { "vertex_indices" is a list of ints }
end_header                 { delimits the end of the header }
0 0 0                      { start of vertex list }
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3                  { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

Ply file: one way of representing a 3D mesh
Vertices + triangles connecting vertices

Poke Graph

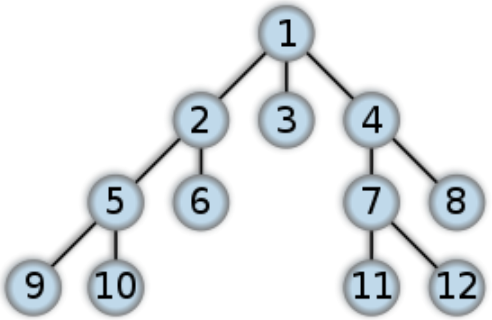


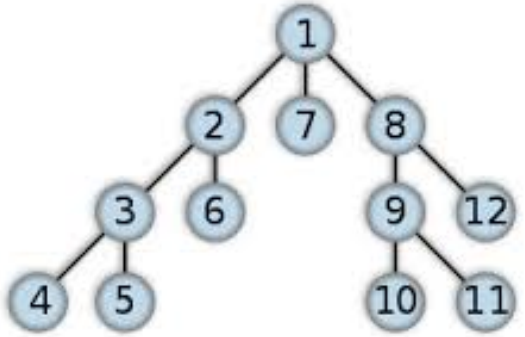
Poke Graph

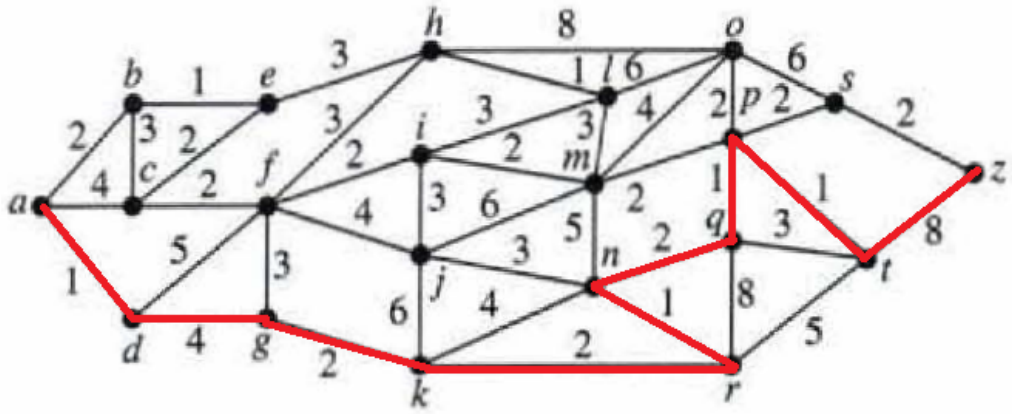
- Node: person
- Edge: message from one person to another
  - Message body
  - Date
  - Has been read yet?

Graph and Tree Traversal

- Breadth first search
- Depth first search
- Shortest path
- Topological sort







Recap

- Abstract data types vs. data structures
- Records/structs
- Linked lists
- Stacks and queues
- Arrays
- Maps / mappings / dictionaries
- Hash tables
- Trees
- Graphs

Links

- http://codercareer.blogspot.com/p/list-interview-questions.html
- http://stackoverflow.com/questions/2429217/under-what-circumstances-are-linked-lists-useful
- http://www.thelearningpoint.net/computer-science/basic-data-structures-in-ruby---linked-list---a-simple-singly-linked-list
- http://matt.weppler.me/2013/08/14/implementing-a-linked-list-in-ruby.html

- http://en.wikipedia.org/wiki/Data_structure
- http://en.wikipedia.org/wiki/Record_(computer_science)
- http://en.wikipedia.org/wiki/Linked_list
- http://en.wikipedia.org/wiki/Graph_traversal
- Moar wikipedia