Core data structures

It's all about relationships

Terence Parr
MSDS program
University of San Francisco

UNIVERSITY OF SAN FRANCISCO

Data structures organize data

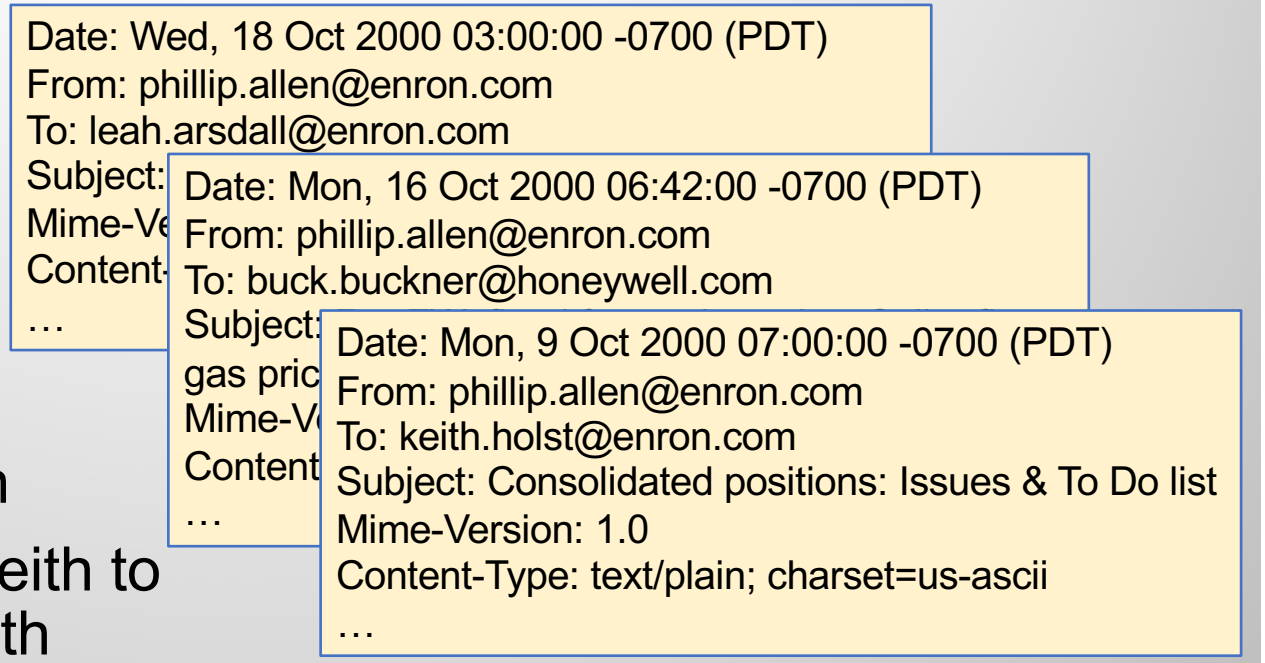
- Data structures group or encode relationships between data elements
- There's a difference between the *abstract data type* and the implementation (list vs array, dictionary vs hashtable, …)
- Two methods to organize data:
  - physical adjacency or relative position in RAM
  - *pointers*
- Algorithms operate on data structures; e.g., sorting algorithm operates on a list
- Often algorithms are needed to construct structures too

Advice on choosing data structures

- Use the simplest data structure you can initially because you never know if that code will survive very long
- Waste processor, memory power before brainpower (if possible)
- There is a trade off between time and space
  - We can often make faster algorithm using more memory
  - It's like driving to the other side of town to save 5% on beer; what are you trying to optimize? time or $$$
- Prep work or more sophisticated data structure can help
  - E.g., element lookup via: unordered list vs sorted list vs hash table
    O(n) O(log n) O(1)

Why you should know about DS/Alg

- Consider Enron emails
- Represent how?
- Depends on what?
- Depends on the info we want to extract
- Find all emails by Keith
- Find email path from Keith to Phillip or find path length
- Find all direct emailers to Keith

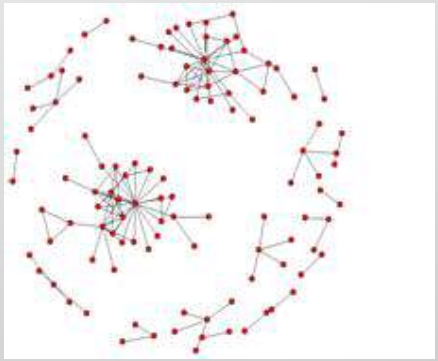


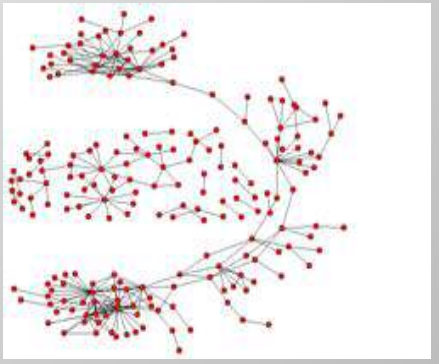
https://www.kaggle.com/wcukierski/enron-email-dataset

What can we learn, what alg's do we need

- Fast string search to find emails
- Compute edit distance to find similar or misspelled email addrs
- Shortest path analysis to discover company relationships not on org chart
- k-cliques (subcommunities) became more common as crisis built at Enron

Date: Mon, 9 Oct 2000 07:00:00 -0700 (PDT)
From: phillip.allen@enron.com
To: keith.holst@enron.com
Subject: Consolidated positions: Issues & To Do list
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
…





See *Social Network Analysis and Organizational Disintegration: The Case of Enron Corporation*

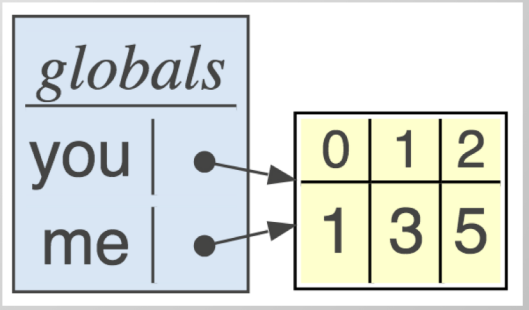
Elemental data in memory (RAM)

- (not disk formats, we covered in data acquisition MSDS692)
- What's the *type*?: typically int, float, string
- Numbers can be of different sizes; e.g., np.float32, np.float64
- Data *values*: an int can represent a number, signed or unsigned, but can also represent a categorical item such as US state
- We can also use strings for categorical but it's much less efficient in space, and often time
- You can even encode multiple things within a single number, such as 5 and 32005 could be a combined 32 and 5
- Data *properties*: e.g., can such values be ordered? Is there a notion of distance between values?

Pointer data type

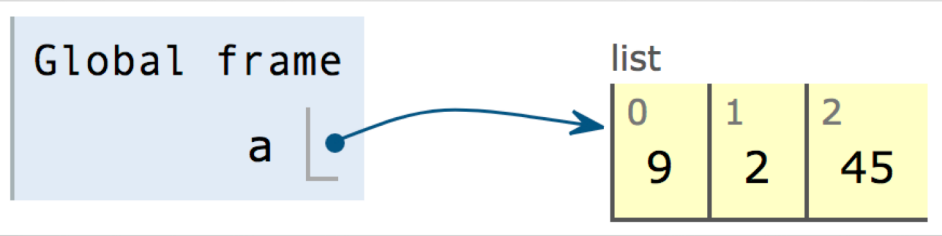
- A pointer p is implemented as an integer variable that holds a memory address, such as “p = Point(3,4)”
- Python knows variable is actually a reference to memory location
- Pointers are also called *references*

```
you = [1,3,5]
me  = you
```



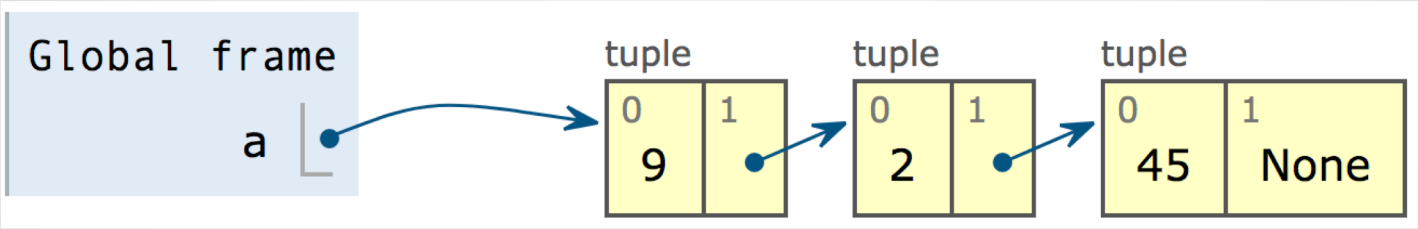
List abstract data type

- *Array* implementation is most common for *list* abstract data structure
- Lists are ordered but items aren't necessarily sortable
- Arrays use contiguous memory locations to associate items
- Code "a=[9,2,45]" yields a pointer to contiguous block of cells



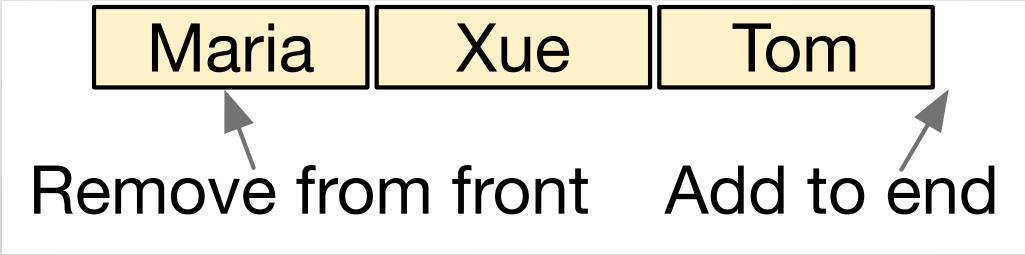
Non-contiguous lists: *linked lists*

- The other way to implement a list data type is with explicit pointers from one element to the next: “a = (9,(2,(45,None)))”



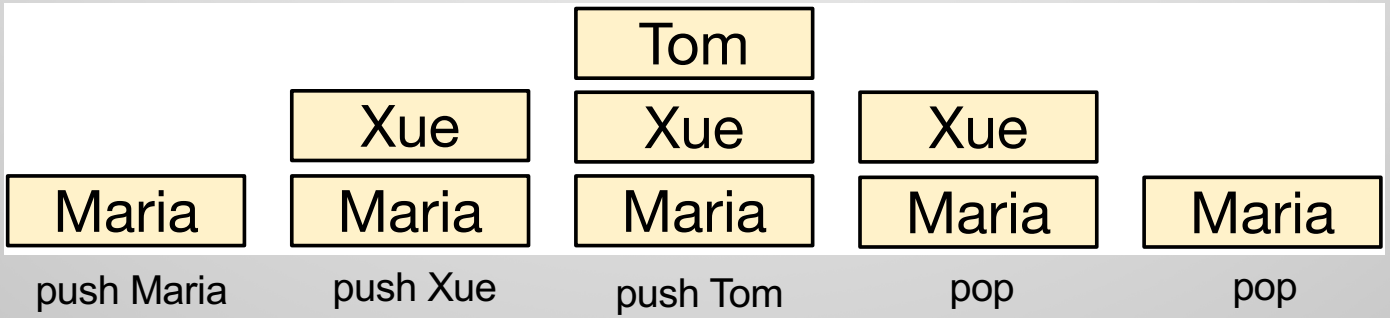
Queue: ordered list

- First In, First Out (**FIFO**); Key ops: ENQUEUE, DEQUEUE
- A list restricted to add to the end and delete from the front
- Most commonly an array implementation



Stacks: like stacks of plates

- Most commonly an array implementation
- First In Last Out (**FILO**); key ops: PUSH, POP
- Just a list restricted to add items to end and take from the end
- For us, possibly used as “work list” for non-recursive tree walking

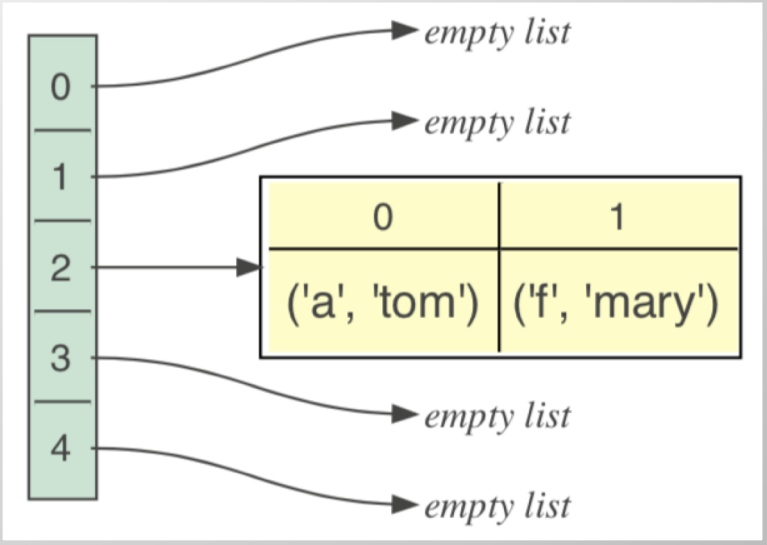


Set: unordered collection

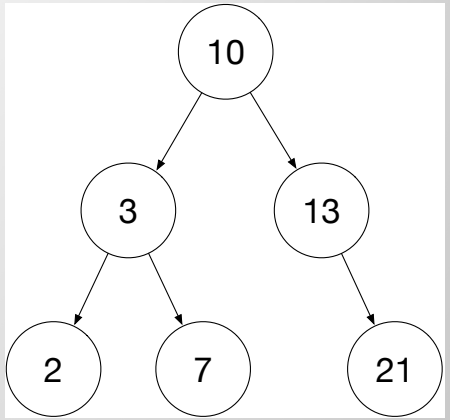
- Typical implementation is a hash table
- Operations are add, delete, contains, union, intersection, etc…
- “contains” operation takes constant time for hashtable implementation

Dictionary abstract data structure

- Maps key to value; i.e., d[key] = value
- Look up values by key; i.e., d[key]
- Hashtable is implementation of choice
- Recall hashtable is array of buckets, each bucket is array of (key,value) pairs

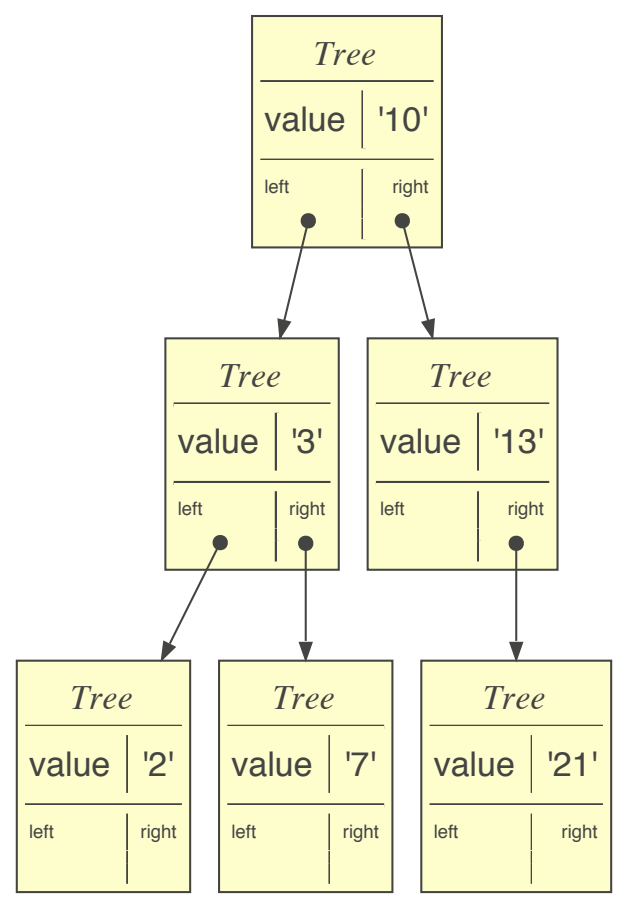


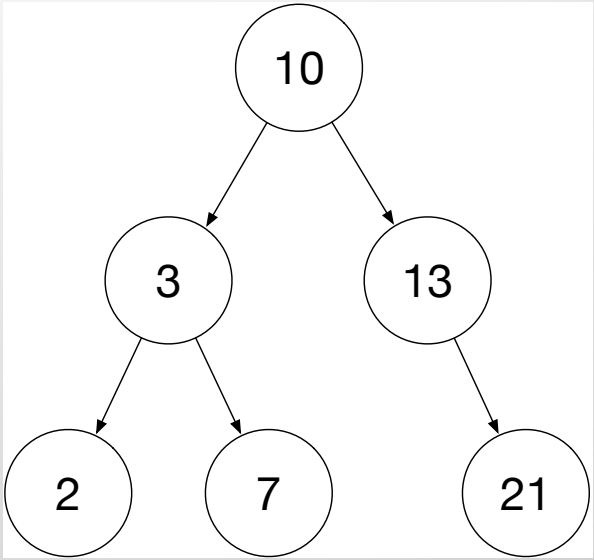
Binary tree abstract data structure



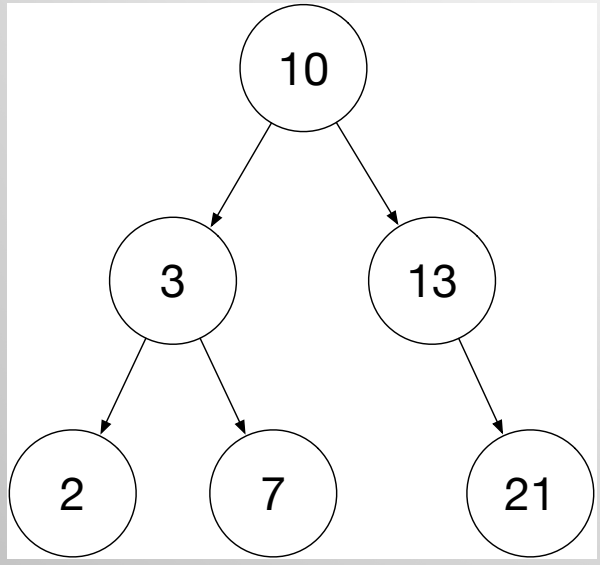
- A directed-graph with internal nodes and leaves
- No cycles and each node has at most one parent
- Each node has at most 2 child nodes
- For n nodes, there are n-1 edges
- A *full* binary tree: all internal nodes have 2 children
- Height of full tree with n internal nodes is about log2(n)
- Height defined as number of edges along path root->leaf
- Level 0 is root, level 1, …
- Note: binary tree doesn't imply *binary search tree*

Binary tree implementation using pointers





Binary tree implementation: contiguous array

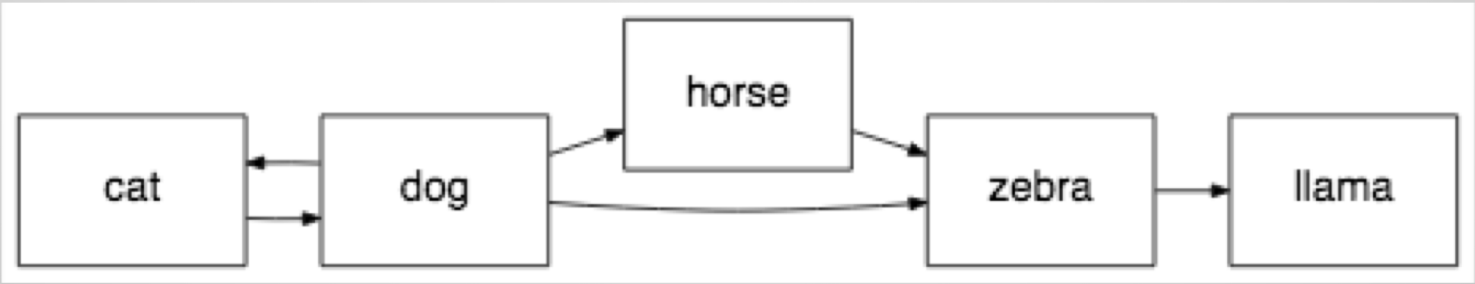


left child is 2i+1
right child is 2i+2

0	1	2	3	4	5	6
10	3	13	2	7	None	21

Graphs

- An arbitrary number of outgoing edges, not just 2 like binary trees
- Can be pointed at by any number of nodes
- Cycles are ok unless specified otherwise; e.g., directed acyclic graph (DAG) is a semi-common term



Basic node definitions (Tattoo these somewhere)

```
class LLNode:
  def __init__(self, value, next=None):
      self.value = value
      self.next = next
```

```
class TreeNode:
  def __init__(self, value, left=None, right=None):
    self.value = value
    self.left = left
    self.right = right
```

```
class Node:
  def __init__(self, value):
    self.value = value
    self.edges = [] # outgoing edges
```

only edges differ

Summary

- Abstract data types:
  List, Set, Queue, Stack, Dictionary, Binary tree, Graph
- Concrete implementations:
  arrays, linked lists, node object with 1+ outgoing edge pointers
- The questions you must ask of the data dictates the data structure and algorithms you need
- Waste processor, memory power before brainpower
  (start with simplest data structure that will work)