Linked Structures
Songs, Games, Movies

Fall 2013
Carola Wenk
The Big Picture (So Far)

- **Hardware**
  - Von Neumann architecture, logic, gates, circuits, binary numbers, machine instructions

- **Software**
  - Python: variables, loops, if-then, functions, lists, recursion

- **Algorithms**
  - Worst-case analysis of running time, simple linear-time algorithms, and efficient searching and sorting.
Algorithms So Far

Interestingly, these performance trends comprise a large fraction of algorithms - why?
Roadmap

• Application areas that seem drastically different are often tied together by algorithms:

- Making a phone call
- Listening to music
- Surfing the web
- Playing a game
- Looking for aliens
- Curing cancer
- Playing Jeopardy
- Telepathic control

Embedded Systems
Multimedia, Networking
Search and Optimization
Artificial Intelligence and Machine Learning

Algorithm development also brings up the question: what abstract problems are efficiently solvable?
Overview

• Media and games must organize data in order to provide a unique user experience (to make $$).

1. How are thousands of songs managed on a mobile device?

2. How do we actually “play against the computer” in a game?
An iPod can store up to 40K songs.

Are media “objects” just stored as a list? How can we quickly find/add/remove items?
An iPod can store up to 40K songs.

Are media “objects” just stored as a list? How can we quickly find/add/remove items?
Array-based Lists

• Recall that we were able to access any element of a list in constant time. How is this possible?

Lists are actually stored contiguously as an array of memory locations; we can access any element using simple arithmetic.

The physical location of the \(i\)-th element is just \(i\) memory locations from the beginning of the list.
Modifying Large Arrays

- How is a list of songs actually structured into an array?
Modifying Large Arrays

- If we store the list of songs as an array, then what do we do when we add content to our library?
Modifying Large Arrays

- To insert a song into this list, we have to restructure the contiguous storage to be larger, and shift elements down.
Modifying Large Arrays

- In the worst case, we have to move the entire list to add or remove entries. Syncing would be a nightmare!
Creating New Types

- Fundamentally, the problem of adding new data is that there is no room in an array.

- We need a different **data structure** that is more “spaced out”.

- What we really need is a way to consider items separately and link/string them together like “beads”.

- Using **indirection**, we can create our own “type” in Python that allows us to dynamically grow a collection of items.
In the dynamic list (= linked list), each element has a value, as well as what is “next” to it in the list.
Dynamic Lists (Linked List)

How do we add an item to a dynamic list?
How do we add an item to a dynamic list?

Static

Dynamic (Linked)

somewhere in memory

shift items
Because each list element explicitly stores where it’s neighbor is, to add an element to the dynamic list, we just need to assign a new neighbor.
In contrast to the static list, we only need to perform a constant amount of work to add an item to the dynamic list.
This data structure is often referred to as a **linked list**.
To implement a linked list, we need to create a new type that carries both data and a reference to the “next” item.
A List Node

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def __str__(self):
        return str(self.data)

x = Node('hello')
y = Node('world!')
x.next = y
print x
print x.next

how will a variable of this type be initialized?

how can we print the contents of this variable?

'hello'

'world!'
Linked Structures

Dynamic data structures allow us to specifically design how information is “laid out”. This is one of the keys to enabling the efficient storage and retrieval of media content on mobile and embedded devices.