Implement Day

Anna Baynes

CSC 130

What's going on...

- First class – Stacks, Queues, Analysis
- Tuesday – Lots of math and algorithm analysis
- **Today – Coding and Implementation**
  - **Java, project session**
- Next week
  - ***New Data Structures!!***

Add Codes

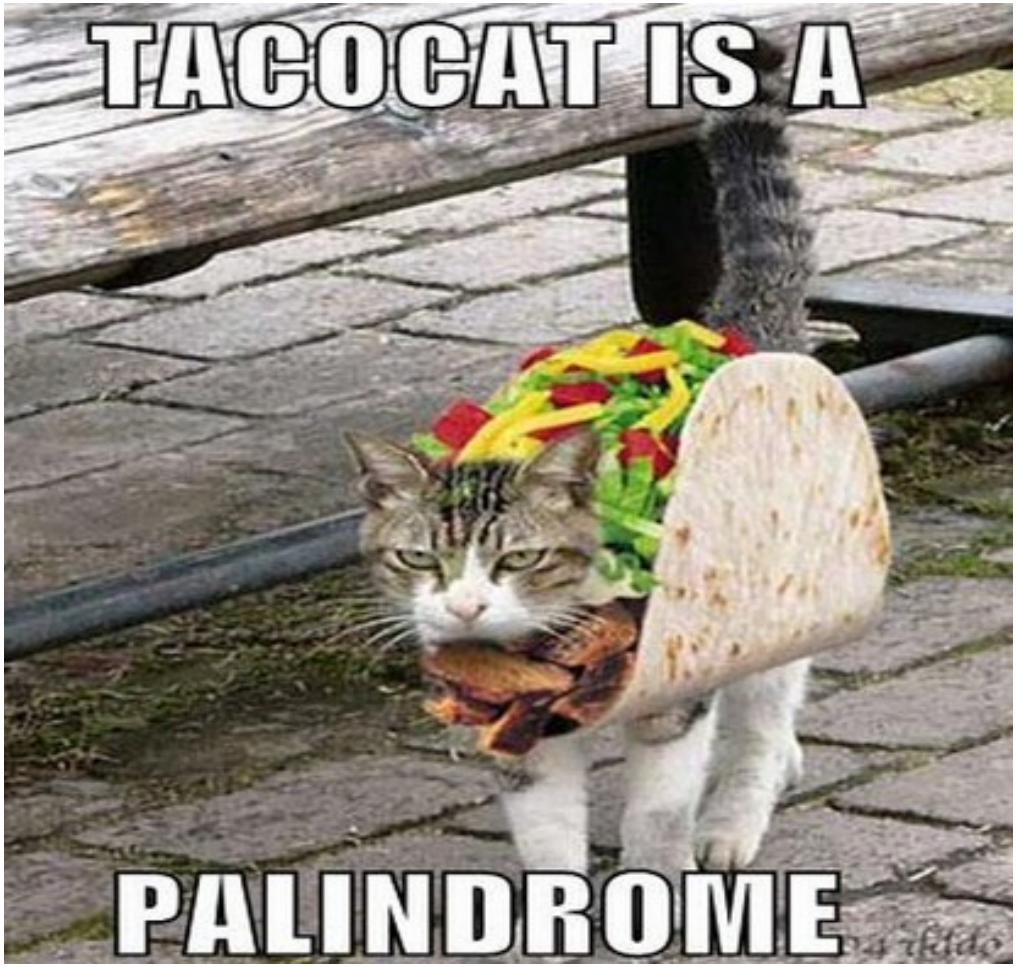
- Priority & Must have come to both previous classes
  - **Will give out today~!**

Java

- Which IDE do you use?
- Eclipse tutorial
- http://www.eclipse.org/downloads/
- http://www.oracle.com/technetwork/java/javase/downloads/index.html

Palindrome

- What is it?



Palindrome

WIKIPEDIA
The Free Encyclopedia

Article | Talk | Read | Edit | View history | Search

Palindrome

From Wikipedia, the free encyclopedia

"Palindromes" redirects here. For the film, see Palindromes (film).

A **palindrome** is a word, phrase, number, or other sequence of characters which reads the same backward or forward. Allowances may be made for adjustments to capital letters, punctuation, and word dividers. Famous examples include "A man, a plan, a canal, Panama!", "Amor, Roma", "race car", "taco cat", "Was it a car or a cat I saw?" and "No 'x' in Nixon".

Composing literature in palindromes is an example of constrained writing.

The word "palindrome" was coined by the English playwright Ben Jonson in the 17th century from the Greek roots *palin* (πάλιν; "again") and *dromos* (δρόμος; "way, direction").

Contents [hide]

1 History

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here

Testing if something is a palindrome?

- Pretty popular computer science problem
  - ex. Find the largest palindrome in X...
- How do we check if something is a palindrome?

TACO CAT

How to check if a linked list is a palindrome?

- Palindrome
  - $0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 0$
- Same backwards and forwards

- Try out!

1st Solution: Reverse and Compare

- Reverse the linked list and compare the reversed list to the original list
  - Equal?? Then lists are identical
- Only need to compare the first half of the list
  - Wasteful work... Can we do better?

2nd Solution: Iterative Approach

- We want to find lists where the front half of list is the reverse of the second half
  - How?
    - Reverse the front half of the list
  - Know any data structures which can do this?

2nd Solution: Iterative Approach

- Stack!
- Push the first half of the elements onto a stack
  - Know size of stack?
    - Iterate to first half of elements, careful on odd case
  - Don't know size of stack?

- How would we do this?

Fast Runner/ Slow Runner Trick

- Find first half of stack, when we don't know the size of the stack?
  - Iterate through the linked list with two pointers simultaneously
    - One ahead of the other
  - “Fast” node might be ahead by a fixed amount
- Pointer p1 moves every two elements for every one move that pointer p2 makes
  - When p1 is at N.... where is p2?
  - Draw a picture

Iterative Algorithm

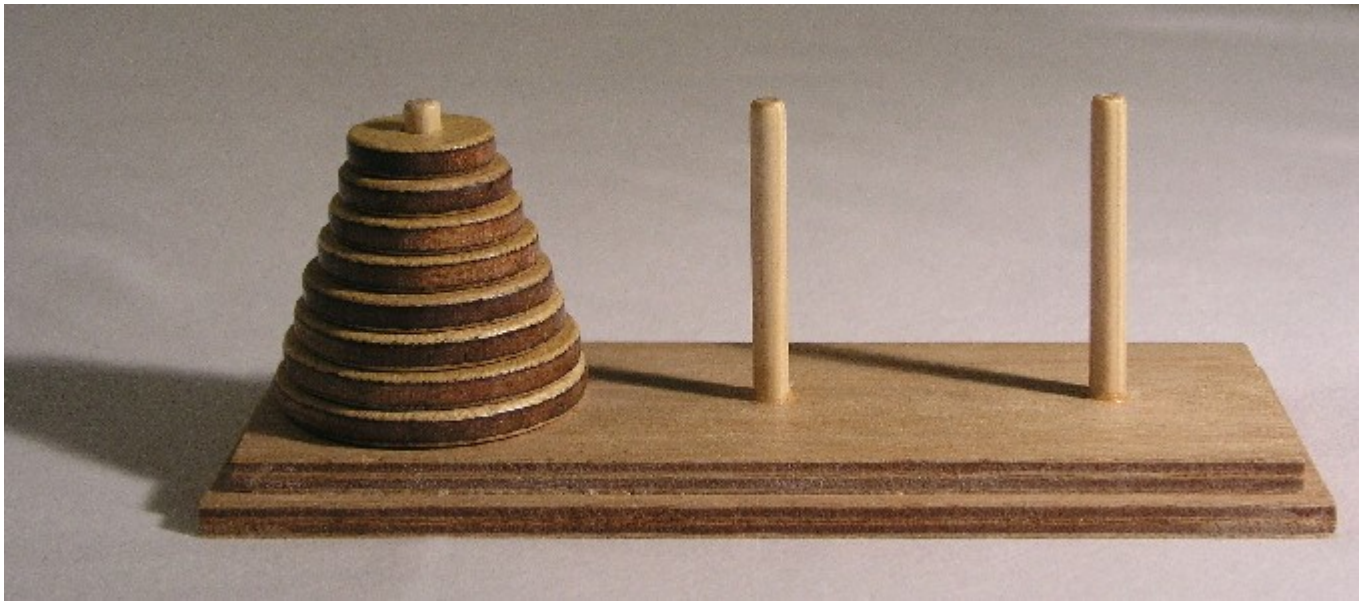
- We push elements from the slow runner onto a stack
- When fast runner hits the end, slow runner reaches the middle
- At the end, the stack will have all the elements from the front of the linked list, but in reverse order
- Now iterate through rest of the linked list, compare to top of stack
  - No difference at end, → palindrome!

Demo

```
boolean isPalindrome(LinkedListNode head){
    LinkedListNode fast = head;
    LinkedListNode slow = head;

    Stack<Integer> stack = new Stack<Integer>();

    //push elements from first half of linked list onto stack
    while(fast != null && fast.next != null){
        stack.push(slow.data);
        slow = slow.next;
        fast = fast.next.next;
    }


    //has odd num of elements, so skip the middle
    if (fast != null){
        slow = slow.next;
    }

    while(slow != null){
        int top = stack.pop().intValue();

        //if values are different, then it's not a palindrome
        if(top != slow.data)
            return false;

        slow = slow.next;
    }
    return true;
}
```

Towers of Hanoi

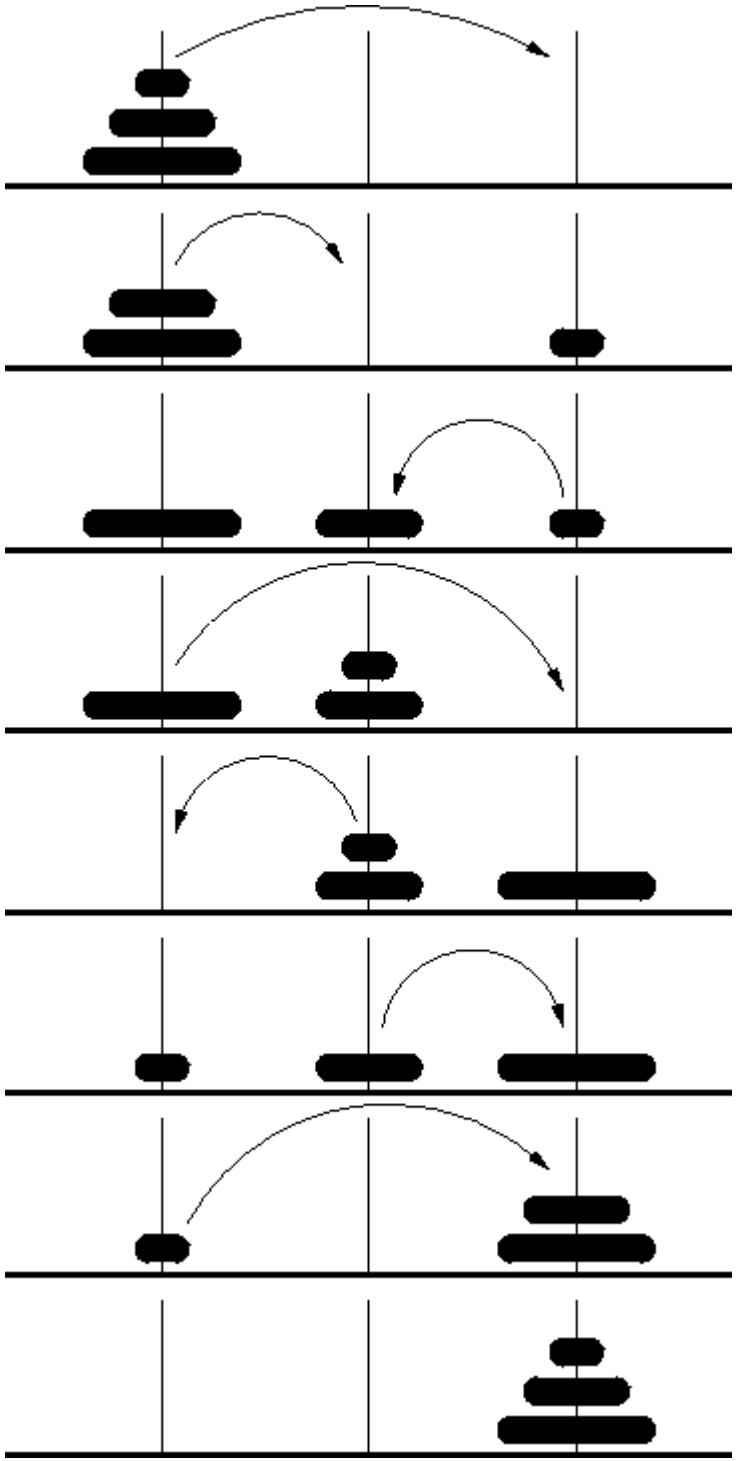
- Goal: Move the stacks of disks from the source pin to the destination pin



Rules

- Only one disk can be moved at a time
- A disk is slid off the top of one tower onto the next tower
- A disk can only be placed on top of a larger disk

- Write a program to move the disks from the first tower to the last – Using stacks!



Smallest possible example

- Case n = 1 (# of disks)
- Can we move Disk 1 from Tower 1 to Tower 3?
  - Yes,simple move it...

- Case n = 2. Can we move Disk 1 and Disk 2 from Tower 1 to Tower 3?
  - Yes, Move disk 1 from tower 1 to tower 2
  - Move disk 2 from tower 1 to tower 3
  - Move disk 1 from tower 2 to tower 3

Case n = 3

- We know we can move the top two disk from one tower to another (as shown earlier), so let's assume we've already done that. (But let's move them to tower 2)
- Move Disk 3 to Tower 3
- Move Disk 1 and Disk 2 to Tower 3 (We just did this for Step 1)

Case n = 4

- Move disks 1, 2, and 3 to tower 2. (We already know how to do this from earlier)
- Move disk 4 to tower 3
- Move disks 1, 2, 3, back to tower 3 (repeat step 1 basically)

Pseudocode

```
MoveDisks(int n, Tower origin, Tower destination,
Tower buffer) {

    /*base case*/

    if( n <= 0) return;

    /* move top n-1 disks from origin to buffer, using
destination as buffer */

     MoveDisks(n-1, origin, buffer, destination);

    /* move top from origin to destination

     MoveTop(origin,  destination)

    /* move top n -1 disks from buffer to destination,
using origin as a buffer */

    MoveDisks(n-1, buffer, destination, origin);
```

Analysis, … again

- How long will it take to solve the puzzle for n disks?
- What's a good predictor?

Analysis, … again

- How long will it take to solve the puzzle for n disks?
- What's a good predictor?
  - The # of times we move a disk from one pin to another
  - # of moves

Analysis, cont'd

- What is the pattern in the number of moves as n increases?
  - Let f(n) be the number of moves for n disks
  - Recurrence relation

Analysis, cont'd

- What is the pattern in the number of moves as n increases?
  - Let f(n) be the number of moves for n disks
  - Recurrence relation

$$f(n) = \begin{cases} 1 & n = 1 \\ 2f(n\text{-}1) + 1 & n > 1 \end{cases}$$

Now Implement

- Implement Towers of Hanoi
  - What data structure can you use to represent the towers?

Exponential Complexity

- What's the pattern?

$$f(n) = 2^n - 1$$

- How do we prove this?
  - Self-exercise

Disks	Moves
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

Project 1 Work Session

- After work session
  - Are you able to run SoX?
  - Do you understand the code changes you must make?
  - I will make “secret.wav” available next Tuesday...