The Art of Data Structures

Deques

Alan Beadle
CSC 162: The Art of Data Structures
• What Is a Deque?
• The Deque Abstract Data Type
• Implementing a Deque in Python
• Palindrome-Checker
Deques
Deques

- Not as widely used as queues, but still handy for some important applications
Deques
A Deque of Python Data Objects
Implementation
Queue Operations

• Deque() creates a new deque that is empty; it needs no parameters and returns an empty sequence

• add_front(item) adds a new item to the front of the deque; it needs the item and returns nothing

• add_rear(item) adds a new item to the rear of the deque; it needs the item and returns nothing
Implementation

Queue Operations

• `remove_front()` removes the front item from the deque; it needs no parameters and returns the item; the deque is modified

• `remove_rear()` removes the rear item from the deque; it needs no parameters and returns the item; the deque is modified
Implementation
Queue Operations

- `is_empty()` tests to see whether the deque is empty; it needs no parameters and returns a boolean value.

- `size()` returns the number of items in the deque; it needs no parameters and returns an integer.
### Deque Operations

<table>
<thead>
<tr>
<th>Deque Operation</th>
<th>Deque Contents</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>d.is_empty()</td>
<td>[]</td>
<td>TRUE</td>
</tr>
<tr>
<td>d.add_rear(4)</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>d.add_rear('dog')</td>
<td>['dog',4,]</td>
<td></td>
</tr>
<tr>
<td>d.add_front('cat')</td>
<td>['dog',4,'cat']</td>
<td></td>
</tr>
<tr>
<td>d.add_front(True)</td>
<td>['dog',4,'cat',True]</td>
<td></td>
</tr>
<tr>
<td>d.size()</td>
<td>['dog',4,'cat',True]</td>
<td>4</td>
</tr>
<tr>
<td>d.is_empty()</td>
<td>['dog',4,'cat',True]</td>
<td>FALSE</td>
</tr>
<tr>
<td>d.add_rear(8.4)</td>
<td>[8.4,'dog',4,'cat',True]</td>
<td></td>
</tr>
<tr>
<td>d.remove_rear()</td>
<td>['dog',4,'cat',True]</td>
<td>8.4</td>
</tr>
</tbody>
</table>
Implementation

Queue Implementation in Python

class Deque:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def add_front(self, item):
        self.items.append(item)

    def add_rear(self, item):
        self.items.insert(0, item)

    def remove_front(self):
        return self.items.pop()

    def remove_rear(self):
        return self.items.pop(0)

    def size(self):
        return len(self.items)
Palindromic-Checker

A Deque

Add "radar" to the rear

Remove from front and rear
Palindrome-Checker
A Python Implementation

def pal_checker(a_str):
    char_deque = Deque()

    for ch in a_str:
        char_deque.add_rear(ch)

    still_equal = True

    while char_deque.size() > 1 and still_equal:
        first = char_deque.remove_front()
        last = char_deque.remove_rear()
        if first != last:
            still_equal = False

    return still_equal

print(pal_checker("Isdkjfskf"))
print(pal_checker("radar"))
The palindrome checker is cute, but not really compelling.

We could simply start indices at the start and end of the string and read toward the middle.

If we were reading from stdin instead of a string, we'd need to store the characters somewhere, but we could use a string just as easily as a deque.
A slightly more compelling example: browser history

We mostly work at the front of the deque, but when it gets too long we delete off the back

(We never insert at the back, though)
Palindromic-Checker

Discussion: Cilk

• A more compelling example: work queues in the Cilk parallel programming language

• Every thread has a deque containing work to do

• A thread pushes new tasks onto, and pulls tasks off, the front of its own deque

• This LIFO strategy maximizes locality, which makes caches work well
Palindrome-Checker

Discussion: Cilk

- If a thread runs out of work, it steals from the tail of some other thread's deque
- Using the other end avoids interfering with the other thread whenever possible
- NB: this is the same access pattern as the browser history: no insertions at the back
Questions?
Next class, linked data structures!

• Stacks, queues, deques, and lists can all be implemented by linking together one node for each entry

• This has some advantages and is a very common and important technique
• We have a workshop today, but no new lab

• Use the time to work on finishing the labs you already have, read the book, etc

• Friday will have two new labs, both due after 1 week.

• One will be to implement queues/deques as a linked data structure and the other will be about linked lists