CompSci 201, L15: Stacks, Queues, Priority Queues
Logistics, Coming up

• APT Quiz 1 due today, Wednesday 10/19
  • Takes 2 hours, finish by 11:59
  • No regular APTs this week, just the quiz

• Project 4: Autocomplete available, due 10/31.

• Midterm Exam 2 next Wednesday 10/26
  • Same format as first midterm
  • Topics: References, nodes, linked list, recursion, sorting
Today’s Agenda

1. Review Comparator

2. Review Binary Search

3. Introduce Stack, Queue, PriorityQueue
Defining a Comparator: Revisited

• Last time showed “convenient” ways to create a Comparator quickly:
  • Static methods from Comparator: comparing, thenComparing, reversed, etc.
  • Lambda expressions.

• Can also always just create a Comparator class and implement the interface.
  • Must implement compare method.
Suppose you want to sort an Array of Person201 objects...

By building, then floor, then name.

Could create a Comparator class.

```java
public class Person201 {
    private String myName;
    public String myBuilding;
    public int myFloor;
}
```

```java
import java.util.Comparator;

public class PersonComparator implements Comparator<Person201> {
    @Override
    public int compare(Person201 p1, Person201 p2) {
        // Compare methods...
    }
}
```
Live Coding a Comparator for Person201 Objects

Live coding
Runtime Complexity of Sort and Comparator?

• Arrays.sort, Collections.sort, call either compareTo (default) or compare (if you give a Comparator)...

• O(N log(N)) times, on an Array/List of N elements.

• Theoretical proof that this many comparisons is necessary for any comparison-based sorting.
Where are the comparisons in Mergesort?

```java
public static List<String> merge(List<String> listA, List<String> listB) {
    List<String> merged = new ArrayList<>();
    int indexA = 0;
    int indexB = 0;
    while (indexA < listA.size() && indexB < listB.size()) {
        if (listA.get(indexA).compareTo(listB.get(indexB)) <= 0) {
            merged.add(listA.get(indexA));
            indexA++;
        } else {
            merged.add(listB.get(indexB));
            indexB++;
        }
    }
    return merged;
}
```

Overall runtime complexity to sort (quicksort, mergesort, Java API sort) is $O(CN \log(N))$, where $C$ is the runtime complexity of `compareTo` or `compare`.

One comparison per loop iteration, that is, per element merged.
When would C not be constant?

```java
public class ListComp implements Comparator<List<Integer>> {
    @Override
    public int compare(List<Integer> list1, List<Integer> list2) {
        int minLength = Math.min(list1.size(), list2.size());
        for (int i=0; i<minLength; i++) {
            int diff = list1.get(i) - list2.get(i);
            if (diff != 0) {
                return diff;
            }
        }
        return 0;
    }
}
```

Overall runtime complexity to sort N ArrayLists, each with M elements, is $O(MN \log(N))$ in the worst case with this Comparator.

Runtime complexity of this Comparator may depend on the length of the two Lists being compared.
Binary Search
Binary Search

• Given a sorted list of N elements and a target, in just $O(\log(N))$ time, return:
  • Index $i$ such that `list.get(i)` equals target, or
  • -1 if target not in list

• Example:
  • If we search for ‘h’, should return 4
  • If we search for ‘c’, should return -1

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
How is Binary Search $O(\log(N))$?

- How to find something in a list of $N$ elements without looping over the list?
- Let $\text{low}$ (initially 0) and $\text{high}$ (initially $N-1$) mark the limits of the active search space.
- Want to cut down the search space by half at each step:

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

$\log_2(N)$ steps!
Binary Search in Pictures

• Searching for ‘d’ in

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Mid = (low + high) / 2

• ‘h’ > ‘d’, so need to keep searching in the lower half.
• Set high = mid - 1;
Binary Search in Pictures

- Searching for ‘d’ in

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

\[ \text{Mid} = \left( \frac{\text{low} + \text{high}}{2} \right) \]

- ‘b’ < ‘d’, so need to keep searching in the *upper* half.
- Set low = mid+1;
Binary Search in Pictures

• Searching for ‘d’ in

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Mid = (low + high) / 2

• ‘d’ equals ‘d’, return mid (2)
Reasoning about Coding Binary Search

• Going to loop while \((\text{low} <= \text{high})\)
  • Looping while there is anything left to search

• For correctness, want to maintain the following loop invariant:
  • If the target is in the array/list, it is in \([\text{low}, \text{high}]\)

• At each step, either find the target and return, or...cut \([\text{low}, \text{high}]\) in half without losing the target
  • Needs sortedness
public static <T> int binarySearch(List<T> list, T target, Comparator<T> comp) {
    int low = 0;
    int high = list.size()-1;
    while (low <= high) {
        int mid = (low + high)/2;
        T midval = list.get(mid);

        int cmp = comp.compare(midval,target);
        if (cmp < 0)
            low = mid + 1;
        else if (cmp > 0)
            high = mid - 1;
        else
        {
            return mid; // target found
        }
    }
    return -1; // target not found
}
WOTO

Go to duke.is/4d3z2

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
Finding the first or last?

- Algorithm we have shown does *not* guarantee to find the first or last such index if there are multiple.

- You will reason about developing these modifications of binary search in P5 Autocomplete.

- Speaking of...
P5 Autocomplete

• How to create something like:

• All about two things:
  • Searching for all words that match on a prefix, and...
  • Sorting them by how common they are,
  • Return these words to show in the GUI above
Stacks and Queues
Stack Abstract Data Structure: LIFO List

route = new Stack
Push(route, Tokyo)
Push(route, Osaka)
Push(route, Nara)
print Pop(route)
print Pop(route)

Print result: Nara Osaka

LIFO = Last In First Out
Push: Add element to stack
Pop: Get last element in

Popping an item removes and returns the item from the top of the stack.

Zybook
java.util.Stack class

• both push and pop are O(1)
  • Adds and removes from end of ArrayList
  • Could also use LinkedList

```java
public static void demo() {
    String[] strs = {"compsci", "is", "wonderful"};
    Stack<String> st = new Stack<>();
    for (String s : strs) {
        st.push(s);
    }
    while (!st.isEmpty()) {
        System.out.println(st.pop());
    }
}
```
DIY Stack with Plain Old ListNodes

Class

```java
public class myStack<Integer> {
    private class ListNode {
        ...
    }

    private ListNode myFirst;
    private int mySize;

    myStack() {
        myFirst = null;
        mySize = 0;
    }
}
```

Push

```java
public void push(Integer val) {
    if (myFirst == null) {
        myFirst = new ListNode(val);
    } else {
        ListNode temp = myFirst;
        myFirst = new ListNode(val, temp);
    }
    mySize++;
}
```

Add new nodes to front of list and advance last node pointer
DIY Stack with Plain Old ListNodes

**Peek**

```
public Integer peek() {
    if (myFirst != null) {
        return myFirst.info;
    }
    return null;
}
```

Peek returns the next value (last in, at front of list), no mutation

**Pop**

```
public Integer pop() {
    if (myFirst != null) {
        Integer ret = myFirst.info;
        myFirst = myFirst.next;
        mySize--;
        return ret;
    }
    return null;
}
```

Pop returns the next value (last in, at front of list), AND removes
Applications? Stack in the real world?

- Remember the call stack?
- History on your web browser / back button?
- Depth-first search in a graph (more coming soon!)
Queue Abstract Data Structure:
FIFO List

Enqueue(wQueue, Mel)
Enqueue(wQueue, Nina)
Enqueue(wQueue, Ruth)
print Dequeue(wQueue)

FIFO = First In First Out

Enqueue: Add element to queue
Dequeue: Remove first in element

Items are dequeued from the front of the queue.

Zybook
java.util.Queue interface

• Both add and remove are O(1)
  • Add at end of LinkedList
  • Remove from front of LinkedList

```java
public static void qdemo() {
    String[] strs = {"compsci", "is", "wonderful"};
    Queue<String> q = new LinkedList<>();
    for (String s : strs) {
        q.add(s);
    }
    while (!q.isEmpty()) {
        System.out.println(q.remove());
    }
}
```
DIY Queue with Plain Old ListNode

### Class

```java
public class myQueue<Integer> {
    private class ListNode {...}
    private ListNode myFirst;
    private ListNode myLast;
    private int mySize;

    myQueue() {
        myFirst = null;
        myLast = null;
        mySize = 0;
    }
}
```

### Add

```java
public void add(Integer val) {
    if (myLast == null) {
        myLast = new ListNode(val);
        myFirst = myLast;
    } else {
        myLast.next = new ListNode(val);
        myLast = myLast.next;
    }
    mySize++;
}
```

- Storing a reference to the end of list as well as front
- Add to END of list instead of front as we did with stack
DIY Queue with Plain Old ListNodes

**Peek**

```java
public Integer peek() {
    if (myFirst != null) {
        return myFirst.info;
    }
    return null;
}
```

**Remove**

```java
public Integer remove() {
    if (myFirst != null) {
        Integer ret = myFirst.info;
        myFirst = myFirst.next;
        mySize--;
        return ret;
    }
    return null;
}
```

EXACT same code as for Stack peek and pop
Applications? Queue in the real world?

• Operating system keeps track of which program should get processor time next.

• Waitlist for class registration on Dukehub?

• Many “shortest way to get from X to Y” problems, e.g., breadth-first search in a graph (more coming soon!)
Priority Queue in the Abstract

Queue sorted by priority instead of insertion order.

Dequeue removes from the front of the queue, which is always the highest priority item.

Zybook
java.util.PriorityQueue Class

- Kept in sorted order, smallest out first
  - Objects must be Comparable OR provide Comparator to priority queue

```java
PriorityQueue<String> pq = new PriorityQueue<>();
pq.add("is");
pq.add("Compsci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
```

```java
PriorityQueue<String> pq = new PriorityQueue<>((
    Comparator.comparing(String::length));
pq.add("is");
pq.add("Compsci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
```

Compsci 201
is
wonderful