CompSci 201, L7: Runtime Efficiency
Logistics, Coming up

• Today
  • Project 1 Nbody due today
  • Runtime efficiency
  • Project 2 Markov releasing later (due in 2 weeks)

• Wednesday 9/21
  • APT 3 due
  • Big O / Asymptotic Analysis

• Friday 9/23
  • Discussion: Maps, Big O, hashCode
Runtime Efficiency, an Empirical Look at String Concatenation
Two methods for repeated concatenation

```java
public static String repeatConcatA(int reps, String toConcat) {
    String result = new String();
    for (int i=0; i<reps; i++) {
        result += toConcat;
    }
    return result;
}
```

Method A: Using String object and basic + operator

```java
public static String repeatConcatB(int reps, String toConcat) {
    StringBuilder result = new StringBuilder();
    for (int i=0; i<reps; i++) {
        result.append(toConcat);
    }
    return result.toString();
}
```

Method B: Using StringBuilder object and append method
Empirical timing experiment

```java
public class StringConcatTiming {
    static final int NUM_TRIALS = 100;
    static final int REPS_PER_TRIAL = 1024;
    static final String TO_CONCAT = "201";

    public static void main(String[] args) {
        long totalTime = 0;
        for (int trial=0; trial<NUM_TRIALS; trial++) {
            long startTime = System.nanoTime();
            //repeatConcatA(REPS_PER_TRIAL, TO_CONCAT);
            repeatConcatB(REPS_PER_TRIAL, TO_CONCAT);
            long endTime = System.nanoTime();
            totalTime += (endTime - startTime);
        }
        double avgTime = (double)totalTime / NUM_TRIALS;
        System.out.printf("Avg time per trial is %f ms", avgTime*1E-6);
    }
}
```

static final used for constants here

Going to time both methods separately.
Empirical results

![Graph showing comparison between MethodA [String] and MethodB [StringBuilder] for average runtime in ms against number of String concat reps. The graph indicates MethodB is more efficient for larger numbers of concatenations.]
### Empirical results in more detail

<table>
<thead>
<tr>
<th>Reps</th>
<th>MethodA (ms)</th>
<th>MethodB (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.384</td>
<td>0.050</td>
</tr>
<tr>
<td>2048</td>
<td>1.136</td>
<td>0.061</td>
</tr>
<tr>
<td>4096</td>
<td>3.443</td>
<td>0.077</td>
</tr>
<tr>
<td>8192</td>
<td>12.244</td>
<td>0.099</td>
</tr>
<tr>
<td>16384</td>
<td>41.754</td>
<td>0.143</td>
</tr>
<tr>
<td>32768</td>
<td>147.719</td>
<td>0.207</td>
</tr>
</tbody>
</table>

Multiply reps by 2 multiplies runtime by 4. Quadratic complexity.

Multiply reps by 2 multiplies runtime by ~2. Linear complexity.
### Empirical results in more detail

<table>
<thead>
<tr>
<th>Reps</th>
<th>MethodA ns/rep</th>
<th>MethodB ns/rep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.375</td>
<td>0.048</td>
</tr>
<tr>
<td>2048</td>
<td>0.555</td>
<td>0.030</td>
</tr>
<tr>
<td>4096</td>
<td>0.841</td>
<td>0.019</td>
</tr>
<tr>
<td>8192</td>
<td>1.495</td>
<td>0.012</td>
</tr>
<tr>
<td>16384</td>
<td>2.548</td>
<td>0.009</td>
</tr>
<tr>
<td>32768</td>
<td>4.508</td>
<td>0.006</td>
</tr>
</tbody>
</table>

Runtime / rep increasing, *greater than linear* complexity.

Runtime / rep not increasing, at most linear complexity.
What’s going on? Documentation?

docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String

Class String

java.lang.Object
  java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc

public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.
methodA revisited

```java
19  public static String repeatConcatA(int reps, String toConcat) {
20    String result = new String();
21    for (int i=0; i<reps; i++) {
22      result += toConcat;
23    }
24    return result;
25  }
```

String is immutable, line 22 creates a new string and copies result then toConcat.

How many characters will be copied per iteration if toConcat == “201”?

• i=0: 3
• i=1: 6
• i=2: 9
• ...
• On iteration i, need to copy 3*(i+1) characters!
How many total characters are copied? Algebra!

methodA: i goes from 0 to reps-1, copy 3*(i+1) characters per iteration.

\[
\sum_{i=0}^{reps-1} 3(i + 1) = 3(reps) + 3 \left( \sum_{i=0}^{reps-1} i \right)
\]

\[
= 3(reps) + 3 \left( \frac{reps}{2} \right) (0 + reps - 1)
\]

\[
\approx \frac{3}{2} reps^2 + reps
\]
Abstracting, Intro to Big O Notation (Preview for next time)

• The $3/2$ in $\frac{3}{2}\text{reps}^2$ doesn’t tell us much about how the performance scales with the size of reps.

• Often, we use asymptotic notation, especially Big O notation to abstract away constants.

• For example: let $N = \text{reps}$, then we say that the asymptotic runtime complexity is $O(N^2)$.
  • If you ~double $N$, you ~quadruple the runtime
What’s the real difference between methodA and methodB?

• methodA: Copies roughly $\frac{3}{2} \text{reps}^2$ characters.

• methodB: i goes from 0 to reps-1, copy 3 characters per iteration $\rightarrow$ copies roughly $3 \times \text{reps}$ characters.

<table>
<thead>
<tr>
<th>Reps</th>
<th>MethodA char copies</th>
<th>MethodB char copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1572864</td>
<td>3072</td>
</tr>
<tr>
<td>2048</td>
<td>6291456</td>
<td>6144</td>
</tr>
<tr>
<td>4096</td>
<td>25165824</td>
<td>12288</td>
</tr>
<tr>
<td>8192</td>
<td>100663296</td>
<td>24576</td>
</tr>
<tr>
<td>16384</td>
<td>402653184</td>
<td>49152</td>
</tr>
<tr>
<td>32768</td>
<td>1610612736</td>
<td>98304</td>
</tr>
</tbody>
</table>
Memory/Runtime Tradeoff

```java
27    public static String repeatConcatB(int reps, String toConcat) {
28       StringBuilder result = new StringBuilder();
29       for (int i=0; i<reps; i++) {
30          result.append(toConcat);
31       }
32       System.out.printf("String builder capacity is %d characters\n", result.capacity());
33       System.out.printf("Result length is %d characters\n", result.length());
34       return result.toString();
35    }
```

Final StringBuilder is using about 146k / 98k ~= 1.5 times as much memory as necessary. Very common tradeoff in data structures!
How does StringBuilder work?

“Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger.” - StringBuilder JDK 17 documentation.

• But how does it grow?

• Geometrically! Like ArrayList, HashMap, ...
  • Still linear amortized complexity, for same reasons
WOTO

Go to duke.is/57dsn

Not graded for correctness, just participation.

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

But do talk to your neighbors!
Designing more efficient algorithms: Examples with HashMaps
CounterAttack APT

- **CounterAttack APT**
  - Count the number occurrences in `str` of each string in `words`.
  - Idea from discussion 3? Use `Collections.frequency()`

```java
str = "one two one two one two vorpal blade"
words = {"snicker", "one", "blade", "runner"}
Returns {0, 3, 1, 0}
```
Efficiency of current solution

- Suppose `String[] words` has N strings
- Suppose `str` has M Strings

Current algorithm:
- For each of the N strings in `words`:
  - count # occurrences in `str`: compare to M strings

$M \times N$ total comparisons, algorithm has $O(MN)$ complexity. Can we decrease this?
Using a Map for M+N complexity

• Instead, use a Map to keep track, loop through words in `str` just once.

```python
str = "one two one two one two vorpal blade"
words = {"snicker", "one", "blade", "runner"}
Returns {0, 3, 1, 0}
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>3</td>
</tr>
<tr>
<td>two</td>
<td>3</td>
</tr>
<tr>
<td>vorpal</td>
<td>1</td>
</tr>
<tr>
<td>blade</td>
<td>1</td>
</tr>
</tbody>
</table>
Using a Map for M+N complexity

- HashMap\langle String,Integer\rangle map stores counts
- Avoid putIfAbsent/getOrDefault?
  - Guard with if statements

```java
public int[] analyze(String str, String[] words) {
    int[] ret = new int[words.length];
    HashMap<String,Integer> map = new HashMap<>();
    for (String s : str.split(" ")) {
        map.putIfAbsent(s, 0);
        map.put(s, map.get(s) + 1);
    }
    for (int k=0; k < words.length; k++) {
        ret[k] = map.getOrElse(words[k], 0);
    }
    return ret;
}
```
NM vs .N+M Complexity

**O(NM)**
If we double N and double M?
• Runtime increases by a factor of 4.

What if N >> M and we double M?
• Doubles runtime, M still relevant

**O(N+M)**
If we double N and double M?
• Runtime increases by a factor of 2.

What if N >> M and we double M?
• Little difference in runtime, N dominates
Leetcode Isomorphic Strings

leetcoding.com/problems/isomorphic-strings

<LiveCoding>