Optimization: Logic Optimization

CPSC 501: Advanced Programming Techniques
Fall 2020

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Tuesday, September 15, 2020

UNIVERSITY OF CALGARY

Code Tuning Logic

Code Tuning

- Guidelines:
  - Save each version of your code using version control
  - Use the profiler to find a bottleneck
  - Tune the bottleneck, using just one technique
  - Measure the improvement
    - If none, revert to the prior version
  - Repeat until desired performance is achieved

Stop when found

Logic Techniques - Found

- Logic techniques
  - Stop testing when answer found
    - E.g.

```
boolean negFound = false;
for (int i = 0; i < input.length; i++) {
    if (input[i] < 0) {
        negFound = true;
    }
}
```

Logic Techniques – Found (cont'd)

- Is better as:

```
boolean negFound = false;
for (int i = 0; i < input.length; i++) {
    if (input[i] < 0) {
        negFound = true;
        break;
    }
}
```

Frequency

Logig Techniques - Frequency

- Order tests by frequency in switch and if-else structures
  - E.g.

```
if ((c == '+') || (c == '-')) {
    processMath(c);
} else if ((c >= '0') && (c <= '9')) {
    processDigit(c);
} else if ((c >= 'a') && (c <= 'z')) {
    processLetter(c);
}
```

Logig Techniques – Frequency (cont'd)

- Since letters are more common, is better as:

```
if ((c >= 'a') && (c <= 'z')) {
    processLetter(c);
} else if ((c >= '0') && (c <= '9')) {
    processDigit(c);
} else if ((c == '+') || (c == '-')) {
    processMath(c);
}
```

Switch

Logic Techniques - Switch

- Substitute switch statement for if-else construct, or vice-versa
  - In Java, an if-else construct is about 6 times faster than a switch
    - Sometimes…
  - But in Visual Basic, is 4 times slower
  - Beware of assumptions (compiler optimizations, caching, predictive code execution mean this is variable sometimes)

Logic Techniques – Switch (cont'd)

```
int i = r.nextInt(10);
switch (i) {
    case 0:
        return r.nextInt();
    case 1:
        return r.nextInt();
    case 2:
        return r.nextInt();
    case 3:
        return r.nextInt();
    case 4:
        return r.nextInt();
    case 5:
        return r.nextInt();
    case 6:
        return r.nextInt();
    case 7:
        return r.nextInt();
    case 8:
        return r.nextInt();
    default:
        return r.nextInt();
}
```

```
int i = r.nextInt();
if (i == 0) {
    return r.nextInt();
} else if (i == 1) {
    return r.nextInt();
} else if (i == 2) {
    return r.nextInt();
} else if (i == 3) {
    return r.nextInt();
} else if (i == 4) {
    return r.nextInt();
} else if (i == 5) {
    return r.nextInt();
} else if (i == 6) {
    return r.nextInt();
} else if (i == 7) {
    return r.nextInt();
} else if (i == 8) {
    return r.nextInt();
} else {
    return r.nextInt();
}
```

Logic Techniques – Switch (cont'd)

```
long total1 = 0;
long total2 = 0;
Thread.sleep(2000);
for (long i = 0; i < times; i++) {
    long time = System.nanoTime();
    switcher();
    total1 += System.nanoTime() - time;
    time = System.nanoTime();
    ifelse();
    total2 += System.nanoTime() - time;
}
System.out.println(total1 / times);
System.out.println(total2 / times);
```

Logic Techniques – Switch (cont'd) [average]

Times	switcher	ifelse
1	3800	1800
10	2530	2420
100	1652	1919
1,000	526.4	552.6
10,000	209.99	204.86
100,000	88.525	94.928

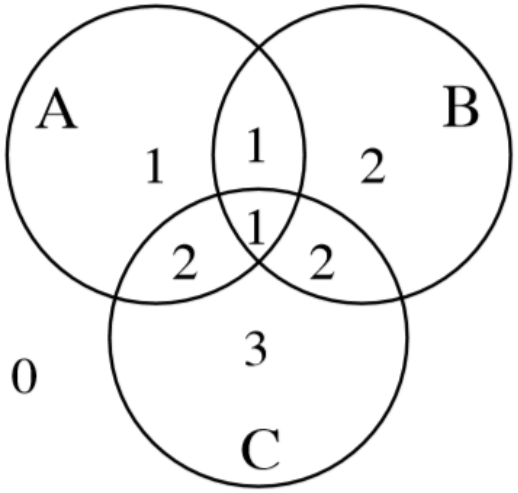
Logic Techniques – Switch (cont'd) [median]

Times	switcher	ifelse
1	3800	1800
10	2250	2250
100	1200	1350
1,000	300	300
10,000	200	100
100,000	100	100

Lookup Table

Logic Techniques – Lookup Table

- Substitute table lookups for complicated expressions
  - E.g.



Logic Techniques – Lookup Table (cont'd)

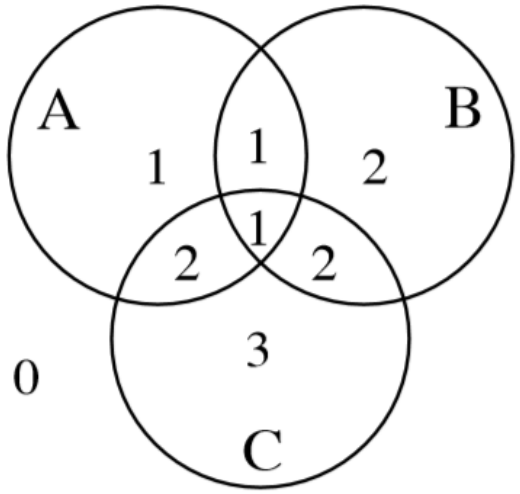
- Can be implemented using complicated logic:

```
if ((a && !c) || (a && b && c)) {
    category = 1;
} else if ((b && !a) || (a && c && !b)) {
    category = 2;
} else if (c && !a && !b) {
    category = 3;
} else {
    category = 0;
}
```

Logic Techniques – Lookup Table (cont'd)

- But is faster with a lookup table:

```
static int[][][] categoryTable = new int[2][2][2];
static {
    categoryTable[0][0][0] = 0;
    categoryTable[1][0][0] = 1;
    categoryTable[0][1][0] = 2;
    categoryTable[0][0][1] = 3;
    categoryTable[1][1][0] = 1;
    categoryTable[0][1][1] = 2;
    categoryTable[1][0][1] = 2;
    categoryTable[1][1][1] = 1;
}
```




```
category = categoryTable[a][b][c];
```

Lazy evaluation

Logic Techniques – Lazy Evaluation

- Use lazy evaluation
  - E.g. A 5000-entry table could be generated when the program starts
    - But if only a few entries are ever used, it may be better to compute values as needed, and then store them in the table
    - i.e. Cache them for further use

Onward to …
loop optimization.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY