CS 241 — Introduction to Problem Solving and Programming

Fundamentals of Programming

More Recursion

Feb 11, 2005

Review: Binary search

Remember binary search.

```
static int binarySearch(String text, char item) {
    int length = text.length();
    if (length == 0)
        return  -1;
    int mid = length / 2;
    char current = text.charAt(mid);
    if (current == item)
        return mid;
    else if (current < item)
        return binarySearch(text.substring(mid + 1, length), item);
    else
        return binarySearch(text.substring(0, mid), item);
}
```

Review: Binary search

Check if the string is empty, return sentinel.

```
static int binarySearch(String text, char item) {
    int length = text.length();
    if (length == 0)
        return  -1;
    int mid = length / 2;
    char current = text.charAt(mid);
    if (current == item)
        return mid;
    else if (current < item)
        return binarySearch(text.substring(mid + 1, length), item);
    else
        return binarySearch(text.substring(0, mid), item);
}
```

Review: Binary search

Look in the middle first.

```
static int binarySearch(String text, char item) {
    int length = text.length();
    if (length == 0)
        return  -1;
    int mid = length / 2;
    char current = text.charAt(mid);
    if (current == item)
        return mid;
    else if (current < item)
        return binarySearch(text.substring(mid + 1, length), item);
    else
        return binarySearch(text.substring(0, mid), item);
}
```

Review: Binary search

Call the method on the right half if item is bigger.

```
static int binarySearch(String text, char item) {
    int length = text.length();
    if (length == 0)
        return  -1;
    int mid = length / 2;
    char current = text.charAt(mid);
    if (current == item)
        return mid;
    else if (current < item)
        return binarySearch(text.substring(mid + 1, length), item);
    else
        return binarySearch(text.substring(0, mid), item);
}
```

Review: Binary search

Otherwise, call the method on the left half.

```
static int binarySearch(String text, char item) {
    int length = text.length();
    if (length == 0)
        return  -1;
    int mid = length / 2;
    char current = text.charAt(mid);
    if (current == item)
        return mid;
    else if (current < item)
        return binarySearch(text.substring(mid + 1, length), item);
    else
        return binarySearch(text.substring(0, mid), item);
}
```

Elements of recursion

A recursive method/algorithm is structured like

Conditional

- *Base (or Stopping) Case:*
  - Process without a recursive call
  - Return
- *Recursive (or Inductive) Case:*
  - Pre-process
  - Make recursive call
  - Post-process
  - Return

Factorial

Base case.

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

Factorial

Recursive case.

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

Factorial

Recursive case: Pre-processing.

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

Factorial

Recursive case: Recursive call.

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

Factorial

Recursive case: Post-processing call.

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

Mutual recursion

A number is *even* if it is one greater than an odd number.

A number is *odd* if is one greater than an even number.

Mutual recursion

A number is *even* if it is one greater than an odd number.

A number is *odd* if is one greater than an even number.

0 is an even number.

Mutual recursion

```
static boolean isEven(int n) {
    if (n == 0) return true;
    else return isOdd(n - 1);
}
static boolean isOdd(int n) {
    if (n == 0) return false;
    else return isEven(n - 1);
}
```

A hole in the bucket

There's a hole in the bucket,
Dear Liza, dear Liza

Then fix it, dear Henry,
Dear Henry, dear Henry

The stick is too long,
Dear Liza, dear Liza

Then cut it, dear Henry,
Dear Henry, dear Henry

The axe is too dull,
Dear Liza, dear Liza

Then sharpen it, dear Henry,
Dear Henry, dear Henry

The stone is too dry,
Dear Liza, dear Liza

Then wet it, dear Henry,
Dear Henry, dear Henry

How shall I get water?
Dear Liza, dear Liza

In the bucket, dear Henry,
Dear Henry, dear Henry

There's a hole in the bucket. . .

A hole in the bucket

```
static void fillBucket() {
    if (holeInTheBucket())
        fixBucket();
    ...
}

static void fixBucket() {
    if (stickTooLong())
        cutStick();
    ...
}

static void cutStick() {
    if (axeTooDull())
        sharpenAxe();
    ...
}
```

```
static void sharpenAxe() {
    if (stoneTooDry())
        wetStone();
    ...
}

static void wetStone() {
    fillBucket();
    ...
}
```

Infinite recursion

```
    static int factorial(int n) {
        if (n == 0)
            return 1;
        else
            return n * factorial(n + 1);
    }

...

Exception in thread "main" java.lang.StackOverflowError
```