Data Structures

Augustin Cosse.

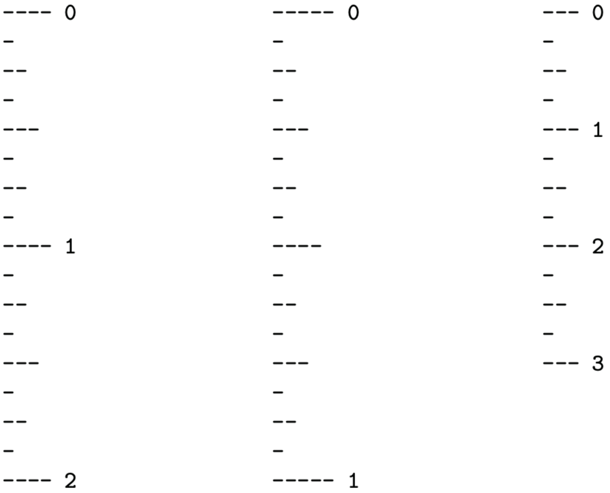
Summer 2025

June 10, 2025

Recursive algorithms: English Ruler

- In the case of the factorial function there is no compelling reason for preferring recursion over a direct iteration with a loop
- As a third example of recursion, we consider the drawing of a typical English ruler
- We denote the length of the tick designating a whole inch as a major tick length
- Between the marks of whole inches, we add a series of minor ticks corresponding to $1/2$ inch, $1/4$ inch and so on

Recursive algorithms: English Ruler



```
---- 0
-
--
-
---
-
--
-
---- 1
-
--
-
---
-
--
-
---- 2
```

```
----- 0
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
----- 1
```

```
--- 0
-
--
-
--- 1
-
--
-
--- 2
-
--
-
--- 3
```

Recursive algorithms: English Ruler

- The English ruler is a simple example of fractal that is a shape that has a self recursive structure at various levels of magnification

- In general, an interval with central tick length $L \geq 1$ is composed of :

    1. An interval with central tick length $L - 1$
    2. A single tick of length L
    3. An interval with central tick length $L - 1$

- Although it is possible to draw such the ruler using an iterative process, the task is considerably easier to acomplish with recursion

Recursive algorithms: File systems

- We first consider a main method `drawRuler` which manages the construction of the entire ruler. Its argument specifies the total number of inches in the ruler and the major tick length.

```
public static void drawRuler
          (int nInches, int majorLength) {
  drawLine(majorLength, 0); // draw inch 0 line
  for (int j = 1; j <= nInches; j++) {
    drawInterval(majorLength -1); // draw  ticks
    drawLine(majorLength, j); // draw inch j line
}}
```

Recursive algorithms: File systems

- The interesting work is done by the recursive `drawInterval` method. This method draws the sequence of minor ticks within some some interval based on the length of the interval central tick.

- The method relies on a base case when $L = 0$ that draws nothing.

- For $L \geq 1$ the first and last steps are performed by recursively calling `drawInterval(L-1)`

```java
private static void drawInterval(int centralLength) {
  if (centralLength >= 1) { // otherwise, do nothing
    drawInterval(centralLength -1); //  top interval
    drawLine(centralLength); // draw center tick
    drawInterval(centralLength -1); //  bottom interval
}}
```

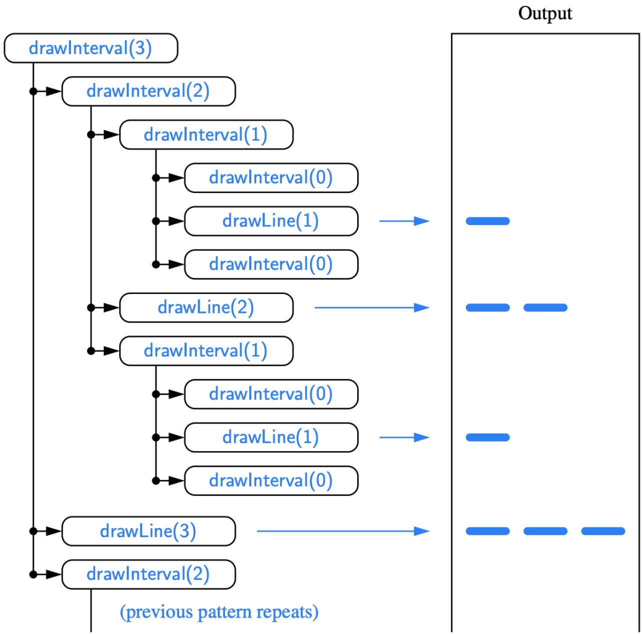
Recursive algorithms: File systems

- Each line is drawn using the `drawLine` method with or without tick

```java
private static void drawLine
                 (int tickLength, int tickLabel) {
  for (int j = 0; j < tickLength; j++)
    System.out.print("-");
  if (tickLabel >= 0)
    System.out.print(" " + tickLabel);
  System.out.print("\n");}
```

```java
private static void drawLine(int tickLength) {
  drawLine(tickLength, -1);
}
```

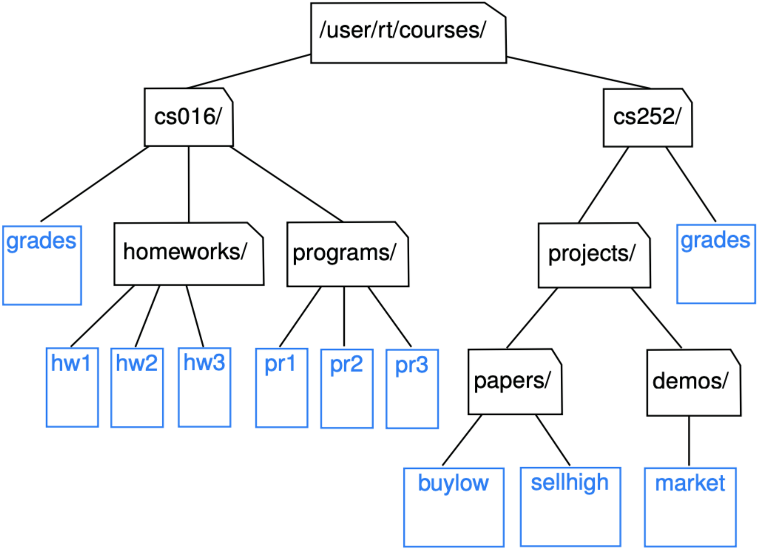
Recursive algorithms: English Ruler



Recursive algorithms: File System

- Modern operating systems define file-system directories (also called "folders") in a recursive way.
- A file system consists of a top-level directory and the content of this directory consists of files and other directories which in turn can contain files and other directories and so on
- The operating system allows directories to be nested arbitrarily deeply although there will always be some base directory that contains only files

Recursive algorithms: File System

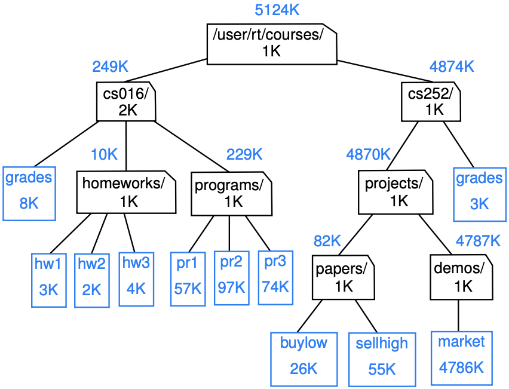


Recursive algorithms: File System

- Given the recursive nature of the file-system representation, it should not come as a surprise that many common behaviors of an operating system such as copying or deleting a directory are implemented with a recursive algorithm.

Recursive algorithms: File System

- For illustration, we display below the disk space being used by all entries in our sample file system. We make the difference between the immediate disk space used by each entry and the cumulative disk space used by that entry and all the nested features. For example cs016 directory uses only 2K of immediate space but 249K of cumulative space.



Recursive algorithms: File systems

- The cumulative disk space can be computed with a simple recursive algorithm. It is equal to the immediate disk space used by the entry plus the sum of the cumulative disk space of any entries that are stored directly within the entry

```
/* Input: A string designating a
              path to a file-system entry */
/* Output: The cumulative disk space used by that
                  entry and any nested entries */
total = size( path)
if path represents a directory then
  for each child entry stored within directory path do
    total = total + DiskUsage( child) {recursive call}
return total
```

Recursive algorithms: File systems

- To implement a recursive algorithm to compute disk usage in Java, we rely on the `java.io.File` class.
- An instance of that class represents an abstract path name in the operating system and allows for properties of that operating system entry to be queried

Recursive algorithms: File systems

- We will consider several methods from this class
    - new File(pathString) or new File(parentFile, childString). A new file instance can be constructed either by providing the full path name as a string, or by providing an existing File instance that represents a directory and a string that designates the name of child entry within that directory
    - file.length() returns the immediate disk usage for the OS entry represented by file
    - file.isDirectory() Returns true if the file instance represents a directory and false otherwise
    - file.list() returns an array of strings designating the names of all entries within the given directory

Recursive algorithms: File systems

- With the use of the `File` class we can now provide a formal implementation that return the disk usage

```
public static long diskUsage(File root) {
  long total = root.length( ); // direct disk usage
  if (root.isDirectory( )) { // if directory,
    for (String childname : root.list( )) {
      File child = new File(root, childname);
      total += diskUsage(child); // add child’s usage}
  }
  System.out.println(total + "\t" + root);
  return total; // return the grand total
}
```

Recursive algorithms: File systems

- The last line prints the total amount of disk space used by a particular directory and all content nested within

```
public static long diskUsage(File root) {
  long total = root.length( ); // direct disk usage
  if (root.isDirectory( )) { // if directory,
    for (String childname : root.list( )) {
      File child = new File(root, childname);
      total += diskUsage(child); // add child's usage}
  }
  System.out.println(total + "\t" + root);
  return total; // return the grand total
}
```