Recursive problems

Prototypical recursive problems: mergesort, quicksort, language generation from grammars, fractal terrain generation, quad trees ...

Divide and Conquer  //  Solve simpler problems

MergeSort, sort an array by halves, by halves, by halves ....

divide array in half
sort first half
sort second half
merge the halves

Sort halves with MergeSort
when stop?

How to merge?
Divide by halving. Until when?

Conquer by merging. How?

mergesort notes continue in sortingBigO.pdf notes
Grammars

Languages consist of words (strings) containing vocabulary symbols (alphabet, characters).

Grammar describes the "rules" for making syntactically correct sentences of words.

Grammar can be written given a set of rules:

\[
x | y \quad \text{means either } x \text{ or } y
\]

\[
xy \quad \text{means } x \text{ followed by } y, \quad xy
\]

\[
<\text{word}> \quad \text{means any instance of } \text{word} \text{ is defined}
\]

Consider the following productions (rules):

\[
<integer> = <integer> | <digit>
\]

\[
<digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
\]

What productions generate 321

\[
321 \leftarrow <integer> <integer> <digit> \quad \text{// 3 calls to } <integer>
\]

\[
<digit> <digit> 1 \quad \text{// each resolves to a digit}
\]

\[
3 \quad 2 \quad \text{// the actual digit}
\]
Consider

productions 1 2 3
<word> = <dot> | <dash> <word> | <word> <dot>
<dot> = *
<dash> = −

What are all the 3 character strings that are in this language?

Is ****-- in this language? What are the productions

Design an algorithm to write all strings of length n
what design observations can we make?
what are the recursive calls – the divide?
what is the halting condition?
Assertions, invariants

Invariant: statement about "condition" or "state" that is always true at a position in an algorithm, or for all the algorithm.

1. Must represent the correctness of the algorithm
2. Must be true initially
3. Statements must preserve (! violate) the invariant
4. Must be true at completion (algorithm must halt)

Assertion: statement about "condition" or "state" at a position in an algorithm.

Assertions can enforce invariants.

Assertions can be written into a program and tested at run-time.
Java's assert statement
If statements: language has no assert
Terrain representation

Procedural terrain generation is a problem found in games, VR, scientific simulation.

A terrain is often 2D surface grid represented by nodes (x, y, z) separated by "spacing" that has been triangulated.

X and Z are the surface grid coordinates and Y is the height.

( Triangle surfaces always have a perpendicular, useful for lighting calculations. )

A terrain surface can be generated from:
- Height maps are int[X][Z] representing the node's height
- Color maps are Color[X][Z] representing the node's terrain color maps fn (height maps + random variation)
Terrain representation

Procedural terrain generation is a problem found in games, VR, scientific simulation.

A terrain is often 2D surface grid represented by nodes (x, y, z) separated by "spacing" that has been triangulated.

X and Z are the surface grid coordinates and Y is the height.

( Triangle surfaces always have a perpendicular, useful for lighting calculations. )

A terrain surface can be generated from:
- Height maps are int[X][Z] representing the node's height
- Color maps are Color[X][Z] representing the node's terrain color maps fn (height maps + random variation)
Fractal height generation

A fractal is "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole,“ – B. Mandelbrot, 1982

Recursively dividing a line and deforming height position creates a 1 D height range.

- deform with ± random value
- smooth value w/ "roughness" constant
- decrease range of random sample each "pass"

http://upload.wikimedia.org/wikipedia/commons/6/6d/Animated_fractal_mountain.gif
Fractal Map

Generalized to 2D with a Square / Diamond algorithm to create a height map and color map for a terrain.


Output is 2 512 by 512 png images: heightMap.png and colorMap.png

Height map square matrix $2^n + 1$ // 513

Set initial height, roughness parameters

- heightMap in grayscale 0..255 range of r,g,b color, white is highest
- roughness function $\text{smoother} = 2^{-\text{roughness}}$
- Initial scale $\text{scale} \leftarrow \text{height} \times \text{smoother}$
- initialize heightMap all cells empty except edges $\leftarrow \text{scale} / 2$
Random offset at any step:

\[
\text{randomValue()} \leftarrow \pm \text{random}(0..\text{scale})
\]

Recursively subdivide height map into quarters

on each pass of algorithm reduce scale

\[\text{scale} \leftarrow \text{scale} \times \text{smoother};\]

fill submatrix values with square / diamond averages

\text{fill2D}(x, y, \text{len}, \text{scale}) \quad \text{// 4 way recursive}
Square / Diamond algorithm

len ← map size    // 9
x ← y ← len / 2
height ← value    // 500
smoother ← 2^{-roughness}    // 2^{-0.8}, 0.57
scale ← height * smoother    // 287
seed edge ← scale / 2    // 143
fill2D(x, y, len, scale)    // ⇒ Terrain

fill2D(x, y, len, scale) {
  center ← rV() + square average    // rV() ← random 0..scale
  left ← rV() + diamond average
  top ← rV() + diamond average
  right ← rV() + diamond average
  bottom ← rV() + diamond average
  scale ← scale * smooth
  if (not halting condition) {
    len = len/2 + 1
    // set x, y for each quadrant from origin
    fill2D(len/2, len/2, len, scale)    // left near
    fill2D(len/2, y + len/2, len, scale)    // left far
    fill2D(x + len/2, y, len, scale)    // right near
    fill2D(x + len/2, y + len2, len, scale) } // right far

square average of 4 "matrix" corner values
diamond average "triangle variant" = (2*center + 2 corners) / 4
Recursive Problems

FractalTerrain

- heightMap : int []
- colorMap : Color []
- random : Random
- width, depth, length : int
- heightMapScale : float
- smoother : float
- terrainTitle : String

+ FractalTerrain(String title, side : int, initialScale : int, roughness : double)
  - menu() : void
  - clamp(v : int, min : int, max : int)
  - scaleHeight() : void
  - fractalRand(range : float) : int
  - fill2D(x : int, z : int, length : int, scale : float)
  - heightMapColor(height : int) : Color
  - makeHeightMap() : void
  - makeColorMap() : void
  - smoothMaps() : void
  - drawMap(Graphics g, grayscale : boolean)
  - saveImage(filename : String, grayscale : boolean)
  - main(args : String[]) : static void
FractalTerrain.java

FractalTerrain.java is a menu based program. Inherits from Comp182Window.java used in the first project.

Fractal Terrain options:
  m   make height and color maps
  h   display height map (grayscale)
  c   display color map (terrain)
  s   smooth maps
  g   save current image as heightMap.png
  t   save current image as colorMap.png
  q   quit menu input

enter command {m h c s g t q} m

Command sequence for following images:  m, s, g, t, q

must kill graphics window to quit program.
run of FractalTerrain
Recursive Problems

images

heightTerrain.png  colorTerrain.png
AGXNASKv4 -- Academic Graphics XNA Starter Kit for CSUN Comp 565 assignments.
Press keyboard for input (not case sensitive 'H' 'h')
Inspector toggles: 'H' help or info 'M' matrix or info 'I' displays next info pane.
Arrow keys move the player in, out, left, or right. 'R' resets player to initial orientation.
Stage toggles: 'B' bounding spheres, 'C' cameras, 'F' fog, 'T' updates, 'Y' yon
Quad Trees

A graphical scene is often populated with solid objects: buildings, trees, rocks, characters (players). Characters in the scene may need to navigate, find paths, and avoid collisions with other solids.

Collision testing can be computational expensive.

Collision testing with single bounding spheres \( O(n^2) \)

\[
\text{distance}(\text{Obj}_1, \text{Obj}_2) \leq (\text{Obj}_1.\text{radius} + \text{Obj}_2.\text{radius})
\]

Navigation graphs \{nodes, connecting edges\} can be used as "waypoints" to travel on, to find paths with \( \text{A}^* \) shortest path).

The construction of navigation graphs can be partially done by using a Quad Tree to partition the space.

QuadTree demo google code
Terrain w/ Quad NavNodes
public void quadTree(int range, int halt) {
    qt(0, 0, range -1, range - 1, halt, 1, "terrain");
}

private void qt(int x1, int z1, int x2, int z2, int halt, int level, String str) {
    int midX = x1 + (x2 - x1) / 2;
    int midZ = z1 + (z2 - z1) / 2;
    boolean sceneObject = sceneObjects(x1, z1, x2, z2); // is empty?
    if (! sceneObject) {
        // empty quad process and done
        addNodes(x1, midX, x2, z1, midZ, z2); // up to 9 nav nodes
        return;
    }
    else if ((x2 - x1) <= halt || (z2 - z1) <= halt) {
        // halt
        // add up to 9 navigation nodes (! duplicates)
        if (! sceneObject) addNodes( x1, midX, x2, z1, midZ, z2);
        return;
    }
    else {
        // recursively subdivide non-empty quad
        qt(x1, z1, midX, midZ, halt, level + 1, "upper left");
        qt(x1, midZ, midX, z2, halt, level + 1, "lower left");
        qt(midX, z1, x2, midZ, halt, level + 1, "upper right");
        qt(midX, midZ, x2, z2, halt, level + 1, "lower right");
    }
}
QuadTree

Green box are objects in scene.

Red box approximates the "bounding sphere" of objects in scene.

Black "nodes" are NavNodes for resulting navigation graph. Middle nodes larger

Adjacency edges are not shown or generated by a QuadTree.

How could we find all adjacency nodes?
Consider the grammar:

\[
\begin{align*}
\langle G \rangle &= \langle E \rangle \mid \langle V \rangle \langle E \rangle \mid \langle E \rangle \langle G \rangle \langle V \rangle \\
\langle E \rangle &= \& \mid \# \\
\langle V \rangle &= W \mid A
\end{align*}
\]

Is \&W#W in the language?
- if yes, what productions?
- if no, why?
- design algorithm to test if string is in language ...

What happens with adding an "blank" production

\[
\begin{align*}
\langle G \rangle &= \langle B \rangle \mid \langle E \rangle \mid \langle V \rangle \langle E \rangle \mid \langle E \rangle \langle G \rangle \langle V \rangle \\
\langle B \rangle &= "_" \quad // \text{blank, shown as underscore}
\end{align*}
\]
## Recursive Problems

<table>
<thead>
<tr>
<th>length</th>
<th>count</th>
<th>blanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>?</td>
</tr>
<tr>
<td>7</td>
<td>192</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>?</td>
</tr>
<tr>
<td>9</td>
<td>768</td>
<td>?</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>?</td>
</tr>
</tbody>
</table>

Any strings longer than 3 with blanks?
If yes how many blanks / string?

If yes, any even number length strings with blanks?

Any strings longer than 3 with blanks?
Any strings longer than 3 with blanks?