CMSC 330: Organization of Programming Languages

Project 1 – Maze Solver

Overview

- Write Ruby program to process maze files
  - Parse & store simple maze file
  - Analyze & process maze
  - Parse standard maze file
- Goals
  - Learn Ruby data structures
  - Learn basic text processing

Maze

- 2D square maze with walls separating cells
- Start & end located anywhere in maze
- Examples

```
+-+-+-+-+
|e|   | |
+ + + +-+
|       |
+-+ + + +
|   |   |
+ +-+ +-+
| |s    |
+-+-+-+-+
```

Solvable

```
+-+-+-+-+
|e|   | |
+ + +-+-+
|   |   |
+-+ + + +
|   |   |
+ +-+ +-+
| |s    |
+-+-+-+-+
```

Unsolvable

Parts of Project

- Read in maze from simple maze file, then
  1. Find maze properties
  2. Pretty-print maze
  3. Process paths & sort by cost
  4. List maze locations by distance from start
  5. Decide whether maze is solvable
- Parse standard maze file
  1. Find & report invalid lines, if any
  2. Else translate into simple maze file

Maze Specification

- Maze
  - An n × n array of cells
  - With specified starting & end cell
  - May include paths
- Cells
  - May have openings in walls
    - Specified as udlr → up, down, left, right
  - Each opening has a weight
- Paths
  - Sequence of directions (udlr) from starting cell

Simple Maze File Format

- Comprised of
  - Header (size, start, end)
  - Cell (location, openings, weights)
  - Path (pathname, start, directions)
- Items separated by spaces
- Example

16 0 2 13 11	16 × 16, start(0,2), end(13,11)
0 1 uldr 4.3 5.1 2.0 5.0	(0,1), openings→uldr, wts → 4.3,…
path p 0 2 drlr	Path p, start(0,2), directions →drlr

- May assume always valid & in correct format

Simple Parser Using String.split

```
line = file.gets
sz, sx, sy, ex, ey = line.split(/\s/)          # maze header
while line = file.gets do
  if line[0...4] == "path"
    p, name, x, y, ds = line.split(/\s/)       # path spec
  else
    x, y, ds, w = line.split(/\s/,4)           # cell spec
    ws = w.split(/\s/)
    ws.each {|w| … }
  end
end
```

Parser is provided, but still need maze data structures

Analyze & Process Maze

- Find maze properties
  - Number of closed cells
  - Number of openings of each type (udlr)
- Pretty-print maze
- Process paths & rank by cost
  - Path invalid if does not pass through opening
  - Cost of path = sum of weight of each opening on path
- List maze locations by distance from start
- Decide whether maze is solvable

Standard Maze File Format

- Comprised of header, cells, paths
- Items separated by spaces, commas, colons, etc.
- Example

maze: 16 0:2 -> 13:11	16 × 16, start(0,2), end(13,11)
0,1: uldr 4.3,5.1,2.0,5.0	(0,1), openings→uldr, wts → 4.3,…
“p:(0,2),d,r,l,r”,“q…”	Path p, start(0,2), dirs →drlr, Path q

Parse Standard Maze File

- Parse standard maze file
- Use Ruby regular expressions
- Invalid maze file if any lines not in proper format
  - Helpful to use ^ and $ to match entire line
  - line =~ /^expr$/ # true only if entire line matches expr
- If invalid maze file
  - Output “invalid maze”
  - Followed by list of invalid lines
- Else
  - Output maze in simple format

Ruby Program

- Code in maze.rb
- Invoked with two arguments
  - Mode command to execute
  - Filename name of maze file
- Print output to console
  - Using puts, print, etc…
- Examples
  - ruby maze.rb print <name of simple maze file>
  - ruby maze.rb solve <name of simple maze file>
  - ruby maze.rb parse <name of standard maze file>

Project Tasks

- Handle command line arguments
  - ARGV[0], ARGV[1]
- Open & read text files
  - file = File.new(<filename>, "r")
- Processing data
  - Count, compare, add, sort…
- Recognizing text patterns
  - Regular expressions (e.g., /\s/)
- Etc…

Administration

- Project description & files
  - Download from class web page
- Due midnight Wed, Sep 24th
  - 10% penalty for 1 day late
- Submit maze.rb to submit server
  - submit.cs.umd.edu
- Public test cases provided
  - Sample inputs & outputs available