INF-2101 Algorithms
Assignment 2

Ida C. Rønningen and Jarl Fagerli
Department of Computer Science
University of Tromsø

September 27th, 2012

Abstract

The goal of this assignment is to search a labyrinth for an item before a timer runs out, where the timer keeps track of your steps and each step requires a certain amount of time. The final result is a working, but slow, algorithm that finds the item searched for.

1. Introduction

1.1 Overview

The assignment is based on a movie called "*Labyrinth*" from 1986. Sarah is searching for her brother Toby in a labyrinth, and the goal is to implement an algorithm so she can reach Toby before the time runs out. An algorithm to find Toby has been implemented and designed and the report will explain how the methods work and how it was designed.

1.2 Requirements

The only requirements for this assignment is to find Toby before the time runs out, and that Sarah must be able to find a path to somewhere she has not been if she gets stuck. The time is initially set to 780. Each step costs 3 units of time, which means you only have 260 steps available.

1.3 Technical background

The labyrinth is represented by a grid with m x n fields. Each field either contains open space, a wall or Toby. The precode provides a function to check, from the present field, the status of the four surrounding fields in the cardinal directions. Sarah is able to move north, south, east and west in the labyrinth, granted that there is open space on the tile in the given direction.

2. Design

The algorithm implemented to find Toby is a very simple one, basically a depth first search. Before taking a step, check if there is an open space in the given direction or not, and if so, go there and remember how to get back to the previous position. This test is done for all the cardinal directions, and Sarah will walk until she is surrounded by previously visited tiles or walls. If she gets stuck, she will move to the previously visited field and repeat the algorithm.

3. Implementation

The algorithm is implemented in a single class `Solver`, which tries to solve the maze problem (i.e. trying to find Toby in the labyrinth). It holds a list of all the visited coordinates in the labyrinth, a stack keeping track of the previously visited field and how to get there, and the coordinates of our current position. It also holds a constant initialized to False, to determine whether Toby is found or not.

The movement part of the algorithm is implemented by first adding the current coordinate to the list which holds the visited fields. Then proceeding with a while loop, the iterating condition being that the variable determining if Toby is found or not is False, and check each of the cardinal directions for an open space. If there is an open space in the given direction and the coordinates of that field is not in the visitlist, append the opposite direction to the stack and call `.direction()` (e.g. `.north()`). Before each step, a function is called to determine if Toby is in one of the surrounding fields, and break if this is the case. If the current coordinate is surrounded by visited fields or walls, a backtrack method is called. The backtrack method pops an item from the stack keeping the directions of how to get to the previously visited coordinates and does four checks to determine which cardinal direction it has to move to get to the previous position. It then moves to the previous position and updates the current coordinate to be the previous.

4. Discussion

The algorithm implemented to find Toby is quite inefficient and slow, as the backtracking function is not finding a short path to the next unvisited field. Another algorithm that could have been used in collaboration with the backtrack function is *Dijkstra's algorithm.* It is conceived by Edsger Dijkstra in 1956 and for a given start vertex in a graph, the algorithm finds the path with lowest cost (i.e the shortest path) between that vertex and every other vertex [1]. This algorithm would be more efficient since it gives Sarah the opportunity to find the shortest path to an unvisited field, instead of backtracking until she reaches an unvisited field. The number of steps required if Dijkstra's algorithm was applied could decrease.
An implementation of Dijkstra's algorithm would also require a function that builds a graph, by adding the current coordinates and the "open field" coordinates surrounding it to a dictionary for each step taken. Each field would be weighted with the value "1". If Dijkstra's was to be called with the graph as input as well as the current and target coordinates, the shortest path would be returned and the backtracking function could be called with a coordinate value for each sequential coordinate in the path. This function is not implemented in the source code. The implementation of the graph along with Dijkstra's algorithm proved to be a difficult task and because of lack of time it was not implemented.

5. Conclusion

The program satisfies the requirements as it finds Toby in less time than available and if Sarah is stuck, she finds a route to a new unvisited field. As of current, there are no known bugs and the program seems to be running fine.

6. Reference

[1] Wikipedia (2012) *Dijkstra's Algorithm.* Available at: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (Cited 27.09.12)