European University of Lefke

COMP415: Artificial Intelligence

**Group 5**

**Instructor: Asst. Prof. Vesile Evrim**

**December 24, 2019**

# **Problem Definition**

To find the shortest distance between a start point and end point in a 12x12 grid using DFS algorithm.

# **Team Members and Responsibilities**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student No. | Name Surname | Work Assigned | Work Done? | Sign |
| 164202 | Naimul Hasan | Working with grid, files and parsing, assist in hardware programming and circuit. Pipelining software, hardware and documentation | YES |  |
| 164177 | Abbasagha Babayev | Assembling hardware and writing software for it. Pipelining software, hardware and documentation | YES |  |
| 164163 | Tafadzwa Imbikani | Creating DFS algorithm, and grid. | NO |  |
| 164253 | Baboucarr Sallah | Creating DFS algorithm and parsing and creating files | NO |  |
| 164182 | Churchill Makate | Creating graph and assisting with DFS algorithm | NO |  |
| 154228 | Muhammad Adil | Create a file parser. | NO |  |

# **Programming Languages**

The primary programming language used is C/C++ because the group had the most experience with C/C++. And the secondary programming language used is Python 3 since it is very intuitive and easy to automate programs.

# **Software Logic**

Firstly, a problem creator was made using the given template, and it was coded in a way so that number of available cells were generated randomly along with randomly selected start and end cells.

Given template,

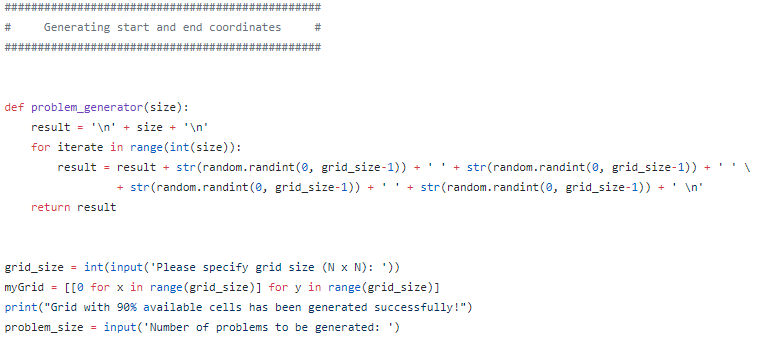
A screenshot of a social media post

Description automatically generated

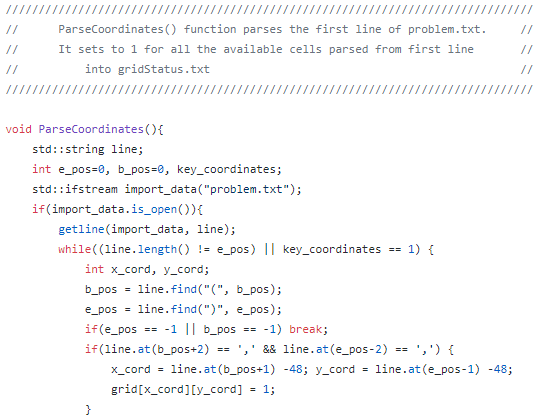
Problem Generator Code Snippet,



Problem Generator Snippet 2



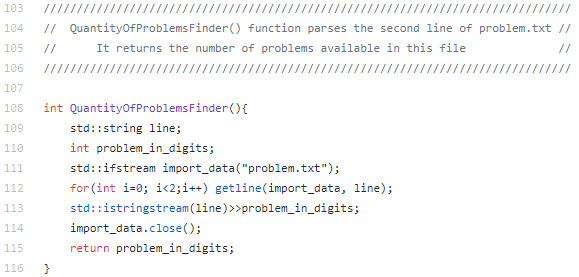
Secondly, **Problem Parser** was created, the first line is read for the number of available cells and applied into the grid variable. The grid variable is initialized with all cells unavailable. Code snippet is provided below: (File name: *problem\_file\_parser.cpp*)



**Problem Parser Snippet 2:**



The second line of problem contains the number of problems, we will use this number to create a loop and record the coordinates into line below it. **Quantity of Problems finder** code snippet:



From third line onwards in *problem.txt*, it contains start and end coordinates separated by a space.

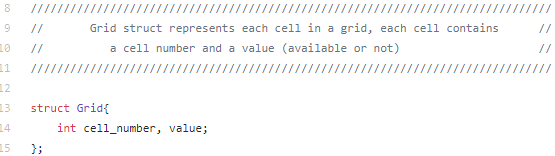
Using the **Quantity of Problems Finder** function we will look at each line for start and end coordinates. These coordinates are then appended to ‘coordinates.txt’. Code snippet is found below,

(File name: *problem\_file\_parser.cpp*)



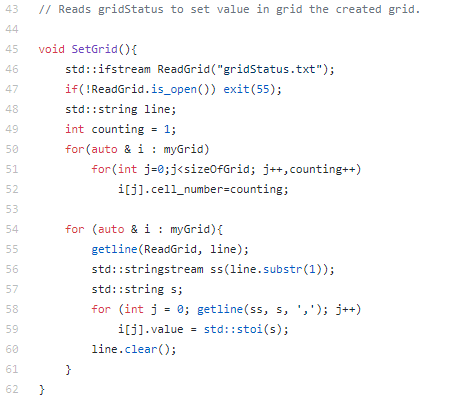
After *problem.txt* is parsed, solution creator will now be created from the data provided by *problem\_file\_parser.cpp*

In file ‘solution\_creator.cpp’, first each gird cell will be defined by cell number and cell value, it is implemented by creating a struct as shown below (File Name: solution\_creator.cpp)

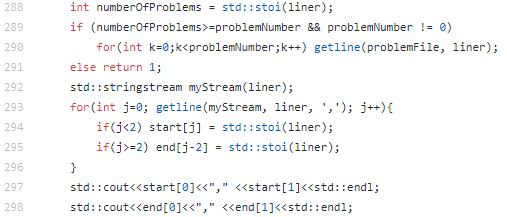


This file reads the *coordinates.txt*, *gridStatus.txt*. Grid array is initialized using ‘*gridStatus.txt*’, start and end coordinates are stored into a variable which is read from ‘*coordinates.txt*’.

(Storing Grid using **SetGrid()** function code snippet) (File Name: *solution\_creator.cpp*)

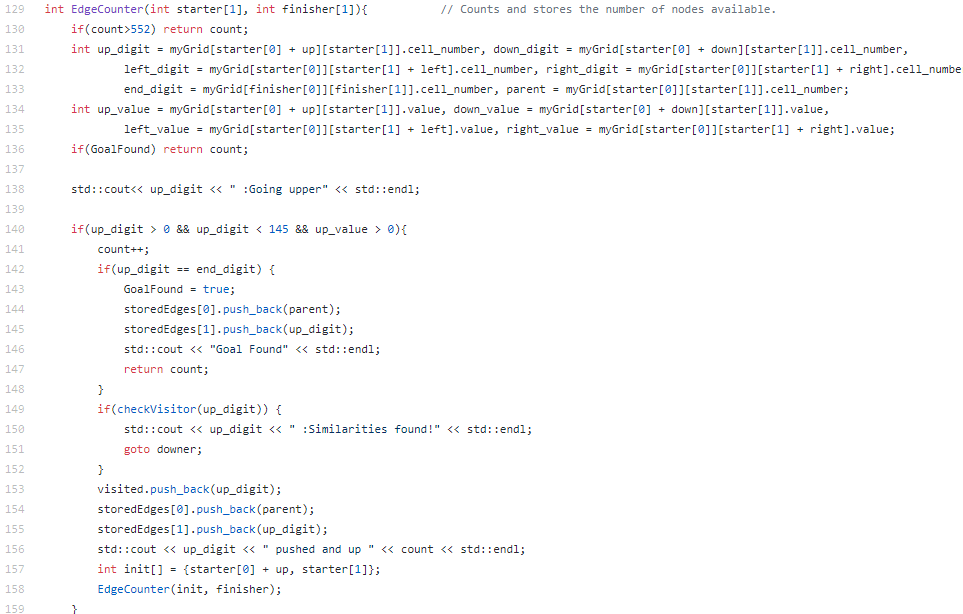


(Storing Start and End Coordinates from **EntryPoint()** function Code Snippet)(File Name: *solution\_creator.cpp*)



After both files are parsed and stored into its respective variables it initializes the DFS algorithm object by passing the number of nodes, these nodes are calculated using **EdgeCounter()** function).

As shown in the code snippet below (File name: *solution\_creator.cpp*)



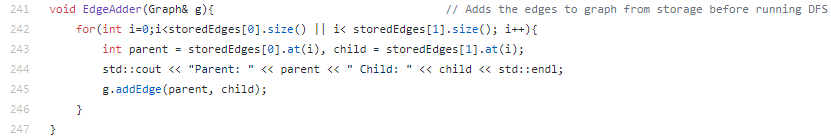


After DFS Graph object is initialized using the number of nodes created the **EdgeCounter()** also creates a graph and stores it into a vector array by parent and child relation using this vector array this graph is inserted into the DFS algorithm using **addEdge()** function that is present in the DFS Graph class.

Code snippet of **addEdge()** function (File Name: *solution\_creator.cpp*)

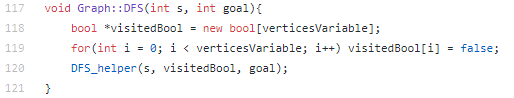


The vector array is traversed using **EdgeAdder()** function, which loops thru and adds all the nodes to the ***DFS Graph object***.

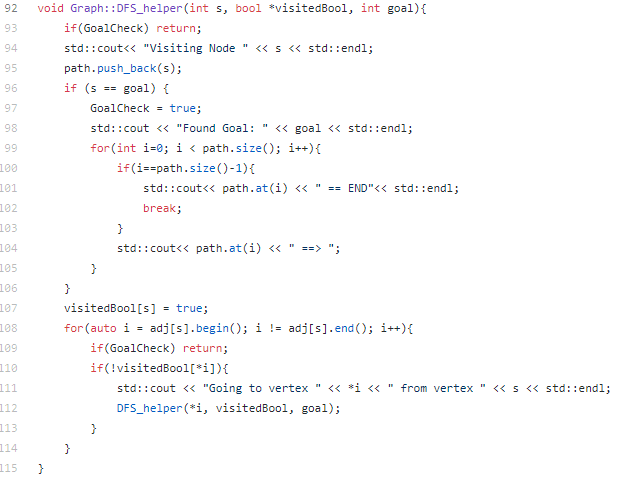


After all the nodes are inserted successfully, the DFS Object runs its algorithm using the **DFS()** function, start and end nodes are passed as its parameters, the function then call another function **DFS\_helper()** function to run the algorithm recursively, until goal is found.

**DFS()** function code snippet as shown below (File Name: *solution\_creator.cpp*)

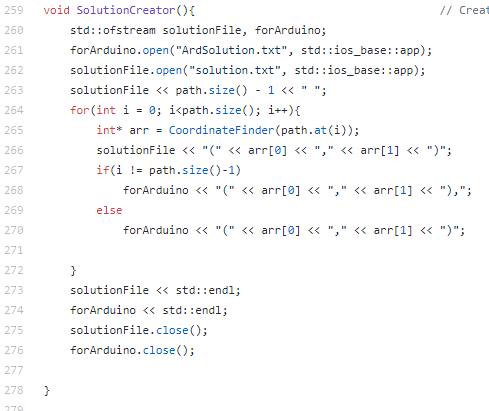


**DFS\_helper()** function code snippet as shown below (File Name: *solution\_creator.cpp*)



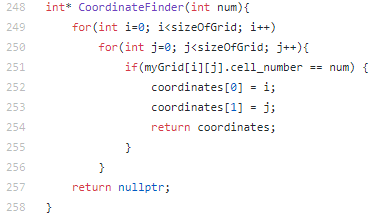
Once ***DFS Class Object*** algorithm finds the shortest distance between start and end coordinate, it will then stores it into path vector variable, which is then used by the **SolutionCreator()** function to create solution.txt and ArdSolution.txt. Code Snippet of **SolutionCreator()** is shown below:

(File Name: *solution\_creator.cpp*)



The vector variable path stores the path by cell number of grid, conversion of cell number to coordinates take place in **CoordinatesFinder()** function which then returns an array of coordinates, which is then stored into the open files.

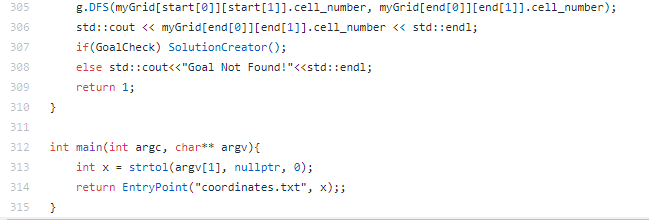
Code snippet of **CoordinatesFinder()** shown below (File Name: *solution\_creator.cpp*)



**CoordinatesFinder()** function takes a number as a parameter which is assumed to be a cell number. It the runs a loop to through the grid to look for a matching cell number, once it matches it then returns the result as an array.

If Goal is not found it will give an output to let the user know. Code snippet is shown below;

(File Name: *solution\_creator.cpp*)



# **Hardware**

We decided to use 8 by 8 matrix for showing output results which is connected to Arduino microcontroller. We divided this problem into the steps which is described below:

## Software Part

## Software Architecture for Arduino

## 

### Serial Communication (Arduino Part)

We used Serial Communication library for Arduino Microcontroller and for Python in order to be able to communicate between controller and Laptop using USB port, which gives us ability to make most important work, such as reading .txt document, creating greed and filling Solution.txt file using programming language in linux environment and then send us final solution inside of Arduino’s environment with ready code inside, waiting for upcoming data from PC. For this part I use:

#include "SoftwareSerial.h"

### Void Setup – Beginning

**void setup() {**

**/\***

**The MAX72XX is in power-saving mode on startup,**

**we have to do a wakeup call**

**\*/**

**lc.shutdown(0,false);**

**/\* Set the brightness to a medium values \*/**

**lc.setIntensity(0,8);**

**/\* and clear the display \*/**

**lc.clearDisplay(0);**

**Serial.begin(115200);**

**}**

this function is needed to initialize all our data in Arduino. Any code that lives inside setup() 's curly brackets ( { and } ) runs once at the very beginning of your program and then never again -- at least not until you reset the Arduino, or upload new code.

### Void Loop – Main Part of Process

In void loop mechanism we can iterate the functions to get the result needed.

if (Serial.available()){

Serial.println("Connected");

**/\* Creating an info string which receives until #\*/**

info = Serial.readStringUntil('#');

Serial.println(info);

} else

Serial.println("Error: Serial Connection is not available");

then as we notice before in (1.2) in Arduino, we are waiting if connection is available and receiving coming from Python string, where after we print it to see if its fine.

### Parsing String

**for(int i = 0; i < info.length(); i++) {**

**if(info[i] == '#'){**

**lc.clearDisplay(0);**

**}**

**if(info[i] == '\n'){**

**lc.clearDisplay(0);**

**delay(3000);**

**}**

**if(info[i] == ' ') {**

**++i;**

**continue;**

**}**

**if(info[i] == '(') {**

**++i;**

**if(isDigit(info[i])) {**

**row = info[i] - '0';**

**Coordinates += row;**

**}**

**++i;**

**if(info[i] == ',') {**

**++i;**

**if(isDigit(info[i])) {**

**col = info[i] - '0';**

**Coordinates += col;**

**}**

**++i;**

**if(info[i] == ')') {**

**++i;**

**}**

**}**

**}**

**}**

In this part we are parsing string and find out integers which will be our coordinates, given as Row and Column. Now after we found LEDs we are ready to light up them.

### LED’s

We create an object lc to control leds from library for Matrices

**LedControl lc=LedControl(12,11,10,4);**

**boolean ON = true;**

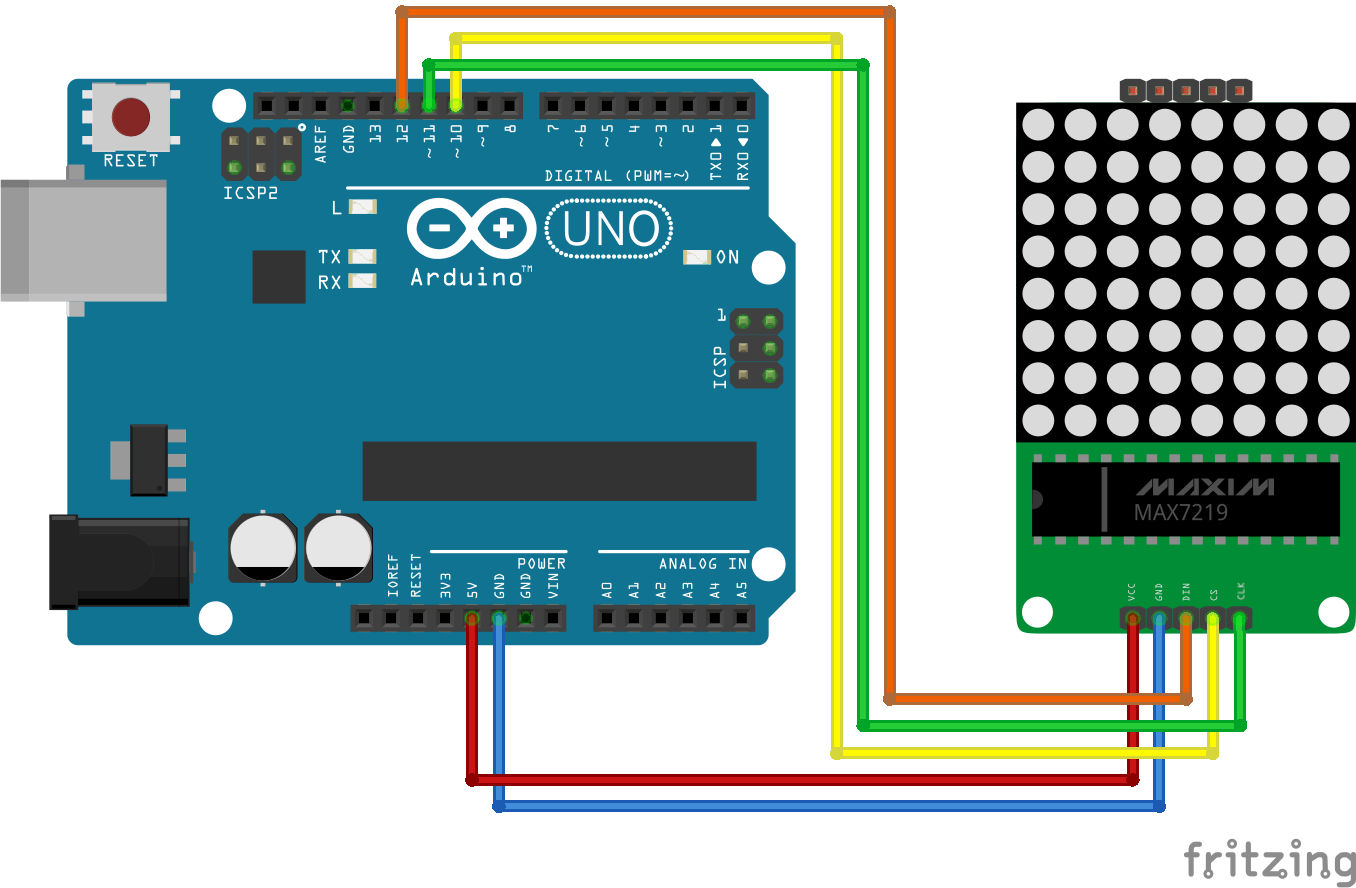
**boolean OFF = false;**

With this part of code we light up each LED which comes in loop each time we find Column and Row in my String

**ls.setLed(0, row, col, ON);**

## Circuit Diagram:

For the connections we simply used Led Matrix and Arduino and connected them in such way:



// Pins: DIN,CLK,CS, # of Display connected

LedControl lc=LedControl(12,11,10,4);

In this part of code we see for which purposes we use Pins 12,11,10.

4 is number of Display connected to arduino. In this case We have 4 screens and use only one, so we divide 4 screens by 4 and get 1 screen as output, rest are off. If we needed we can use them by manipulating this value