

# **Project Report – keyMeUp**

Rebecca Massey 20540609

## **1. Assumptions**

- 1.1. The “cursor” starts at the same position every time the keyboard is used, i.e. it has a designated resting/start position as shown in the image examples. Position on keyboard is always lowercase “a”.

## **2. User Guide**

### **2.1. Purpose**

The purpose of the keyMeUp program is to provide a user with a means to create/load a keyboard representation into a graph data structure to simulate the connections. The program then can be used to find the shortest path to enter a string into the keyboard, including the steps needed to type.

### **2.2. How to Use & Features**

To use the keyMeUp program, navigate to the directory it is in and compile all .java files. After that the user can run the program using one of three options:

“keyMeUp”

Running the program without any following lines will result in the output of usage information to the console.

“keyMeUp -s keyboardFileName strFileName outFileName”

Running the program with these arguments will result in the program reading a keyboard file in and storing a representation of it as a graph, using the representation to process a string from the next file. After processing the file, the resulting path will be output to the specified output file.

#### **Arguments:**

*keyboardFileName*: A correctly formatted .txt file representation of the keyboard to use for pathing.

*StrFileName*: A file containing a string to find the shortest entry path on the keyboard.

*OutFileName*: A file name to save the pathway generated.

“keyMeUp -i”

Opens a menu that users can select operations to manipulate and create keyboard graph representations. Options are numbered 1 to 11, the user selects an option by entering a corresponding number within this range. Option 2 and 3 open sub-menus for the node and edge operations (respectively) that can be used to manually add/edit the keyboard.

## **3. Justifications**

### **3.1. 2D Array**

The program reads in keyboard elements using a **2D array** and for reading in raw keyboard data and inserting it into a graph. I chose this array as it allowed for an easy to implement way of collecting relevant keyboard information. Due to the fact that the keyboards as displayed in the image files display a rid, retrieving the raw input data using a 2D array allows the “keys” to maintain their relative positions within the 2D array.

When inserting nodes and edges into the graphs, the position of elements in the array can be used to create the node (element I) and the edges can be made using the location of neighbouring nodes (left =  $i-1$ , right =  $i + 1$ , etc). This reduced complexity of reading in the keyboard and differentiating between wrapped and unwrapped keyboards.

### 3.2. Graph

I used a **graph** to represent for storing keyboard and tracking connections as the way the keyboards were structured could result in many unordered connections with no cascade. Other data structures covered in this unit were inadequate for these reasons, particularly binary trees. While binary trees are perhaps the most similar structure, they are too ordered and require a clear comparison of values to determine position. The data that needed to be manipulated by the program had directional connections but those were dictated by logical positions on a keyboard as opposed to sorted order. Furthermore, graphs are very flexible data structures, allowing nodes and edges to be created and removed as needed and they have no strict “size” limit (in theory).

### 3.3. Breadth First Algorithm

A breadth first algorithm was used to search for pathways as the structure of keyboards result in nodes having a relatively small number of edges that are wide spread and very connected.

In essence, the keys are connected in every direction (for the most part) and therefore the graph is relatively shallow and spanning. Due to this I felt that searching for pathways was more efficient through breadth as a node will check all directions for the target before moving on to the next node.

I felt that a depth search would take longer as there are lots of edges and possible nodes to visit, resulting in long unnecessary pathways being used.

### 3.4. Graph Edge

Edges are used to maintain the relative positions of keys to the source node. I felt this was the best approach as I could establish that due to the direction of edges being important the direction should be stored as an edge value.

The graph is not bi-directional so the source and destination are important information that is required for the driver program.

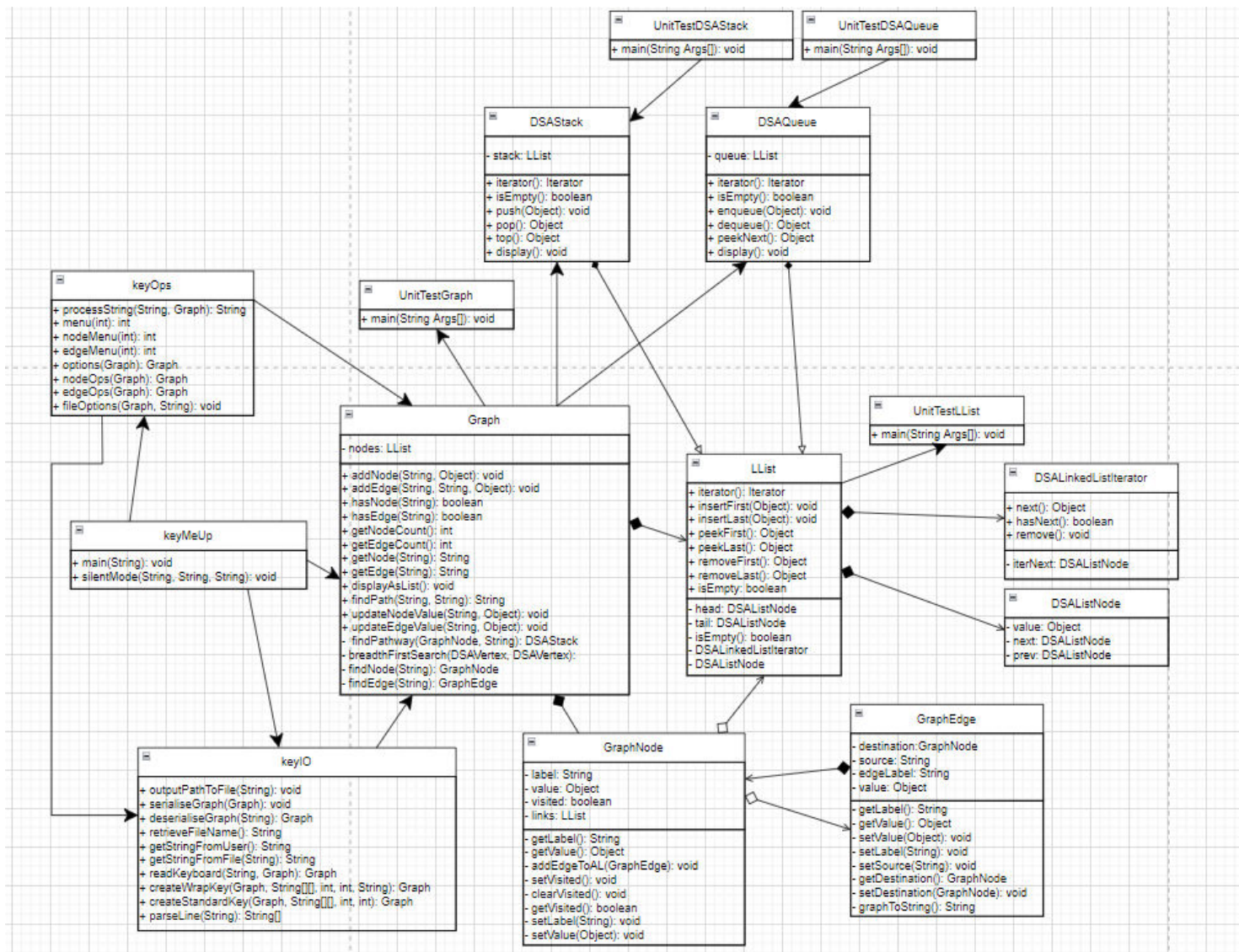
A graph wide adjacency list felt superfluous for the implementation as for complex keyboards such as the switch, the number of edges can be upwards of 200. This would make searching for an edge inefficient and poorly scalable.

Therefore the edges were implemented within a LList maintained by the graph's nodes, thus when searching for edges we first look for the node (a much shorter list with an average of 60) and then through the nodes eternal LList of graph edges which contain a maximum of 4-6 entries.

### 3.5. Graph Node

The graph node is used as a data structure to store the key name and the neighbouring key of which the current node is the source (as the graph is uni-directional). I used the graph node as a separate class so it could contain a list of edges which would allow for faster traversal and edge searching. Having a global list of edges would result in slow traversal as it could contain many elements. A node maintaining a vastly shorter list is much more manageable, especially as this implementation does not require a large amount of nodes.

#### 4. UML Class Diagram



## 5. Traceability Matrix

Requirements	Design/Code	Test
1.1. System displays use information when called w/o arguments	keyMeUp.main()	[PASSED] UI Test: Script 1.1
1.2. Processes files silently when user enters "-s" followed by 3 file names	keyMeUp.main()	[PASSED] UI Test: Script 1.2
1.3 System returns error upon wrong input	keyMeUp.main()	[PASSED] UI Test: Script 1.3
1.4. Displays interactive mode with "-i" argument.	keyMeUp.menu()	[PASSED] UI Test: Script 1.4
1.5 If user enters a command at menu, the system responds accordingly.	keyMeUp.menu()	[PASSED] UI Test: Script 1.5
2.1. Find optimal series of moves to enter a string into a key board	keyOps.processString()	[PASSED] UI Test: Script 2.1
2.2. Movement is shown in the pathway (up, down, left, right and select)	keyOps.processString()	[PASSED] UI Test: Script 2.2
2.3. Incorporates uppercase and lowercase letters	keyOps.processString()	[PASSED] UI Test: Script 2.3
2.4. Incorporates special character keyboard (switch keyboard 3)	keyOps.processString()	[FAILED] UI Test: Script 2.4
3.1. Switch keyboard 1 (lowercase) represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.1
3.2. Switch keyboard 2 (uppercase) represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.1
3.4. Switch keyboard 3 (special chars) represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.1
3.5. iView keyboard is represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.2
3.6. Stan keyboard is represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.3
3.7. Netflix keyboard is represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.4
3.8. Unknown keyboard is represented in .txt file	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.5
4.1. Builds a representation of the keyboards using graph	keyIO.readKeyboard()	[PASSED] UI Test: Script 3.1
4.2. Switch keyboard wraps around all edges	keyIO.createWrapKey()	[PASSED] UI Test: Script 3.1
4.3. iView keyboard wraps around all edges	keyIO.createWrapKey()	[PASSED] UI Test: Script 3.2
4.4. Netflix keyboard doesn't wrap	keyOps.createStandardKey()	[PASSED] UI Test: Script 3.3
4.5. Unknown keyboard wraps around left/right edges	keyIO.createWrapKey()	[PASSED] UI Test: Script 3.6
4.6. Stan keyboard wraps around left/right edges	keyIO.createWrapKey()	[PASSED] UI Test: Script 4.6
5.1. Load keyboard file from specific filename	keyIO.retrieveFileName()	[PASSED] UI Test: Script 5.1
5.2. Can launch a sub-menu of node operations	keyIO.readKeyboard()	[PASSED] UI Test: Script 5.1
5.3. Can launch a sub-menu of edge operations	keyOps.nodeMenu()	[PASSED] UI Test: Script 5.2
5.4. Display graph	keyOps.edgeMenu()	[PASSED] UI Test: Script 5.3
5.5. Display graph information	keyOps.menu()	[PASSED] UI Test: Script 5.4
5.6. Can enter string for pathfinding.	keyOps.menu()	[PASSED] UI Test: Script 5.5
5.7. Generate pathway	keyIO.getStringFromUser()	[PASSED] UI Test: Script 5.6
5.8. Display pathway	keyOps.processString()	[PASSED] UI Test: Script 5.7
5.9. Option to save displayed pathway to file	keyOps.menu()	[PASSED] UI Test: Script 5.8
5.10. Outputs pathway to .txt file	keyOps.menu()	[PASSED] UI Test: Script 5.9
5.11. Save current keyboard (Serialise)	keyIO.outputToFile()	[PASSED] UI Test: Script 5.9
5.12. Load edited keyboard (De-serialise)	keyIO.serialise()	[PASSED] UI Test: Script 5.10
	keyIO.deserialise()	[PASSED] UI Test: Script 5.11
6.1. Find specified edge	UnitTestGraph.java	[PASSED] UI Test: Script 6.1
6.2. Update specified edge	UnitTestGraph.java	[PASSED] UI Test: Script 6.2
6.3. Insert a new edge	UnitTestGraph.java	[PASSED] UI Test: Script 6.3
6.4. Delete an edge	NOT IMPLEMENTED	N/A
7.1. Find specified node	UnitTestGraph.java	[PASSED] UI Test: Script 7.1
7.2. Update specified node	NOT IMPLEMENTED	N/A
7.3. Insert a new node	UnitTestGraph.java	[PASSED] UI Test: Script 7.2
7.4. Delete a node	NOT IMPLEMENTED	N/A
8.1. Silent mode reads file representing keyboard from command args	keyIO.readKeyboard()	[PASSED] UI Test: Script 8.1
8.2. Silent mode reads file representing string from command args	keyIO.getStringFromFile()	[PASSED] UI Test: Script 8.1
8.3. Silent mode outputs generated path to file	keyIO.outputToFile()	[PASSED] UI Test: Script 8.1
9.1. Can read specified file name given by user.	keyIO.retrieveFileName()	[PASSED] UI Test: Script 9.1



## 6. Showcase

### 6.1. **Introduction:**

In the scenarios that follow, there will be a comparison between the way the keyMeUp program handles the input of different types of keyboards and how it attempts to path find around when dealing with the same string.

The test input string will be “Hello, it’s a dreary morning. It’s going to be sunny on Thursday though!”.

*In Scenario 1*, the program will read in a lowercase only non-wrapping array (key\_netflix.txt) keyboard.

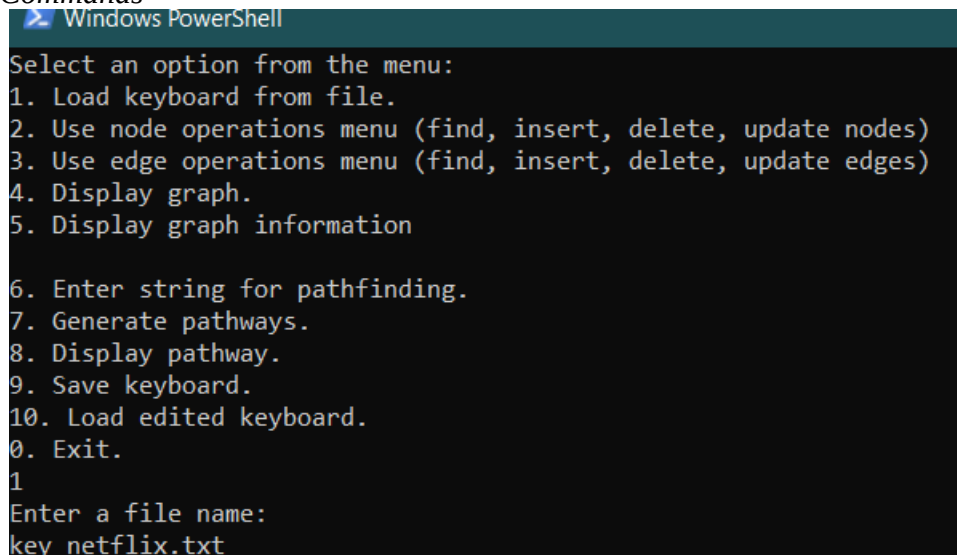
*In Scenario 2*, the program will read in a keyboard with a lowercase and uppercase keyboard that wraps in all directions (key\_stan.txt).

*In Scenario 3*, the program will read in an upper-lowercase keyboard that wraps only to the left and right.

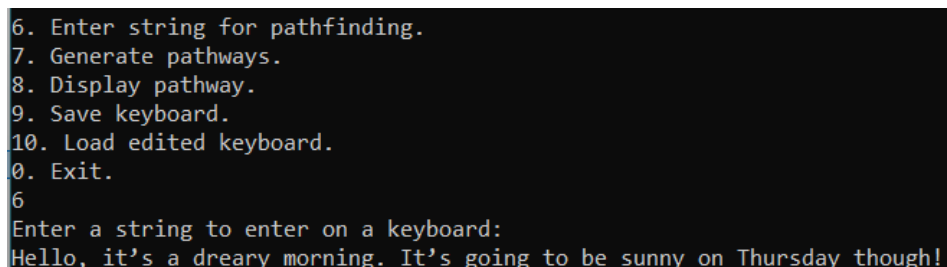
The scenarios will compare how the program handles different keyboard input files and the specification of wrapping and no wrapping.

#### a) **Scenario 1:**

- *Setup:* key\_netflix.txt has been setup to represent the keyboard image. I will be using the interactive keyboard to enter the string listed in section 6.1. I will then generate the pathways and output them to a file called “s1out.txt” I expect that this will result in the pathway including “node not found” strings as the netflix keyboard representation does not contain uppercase and special characters.
- *Commands*



```
Windows PowerShell
Select an option from the menu:
1. Load keyboard from file.
2. Use node operations menu (find, insert, delete, update nodes)
3. Use edge operations menu (find, insert, delete, update edges)
4. Display graph.
5. Display graph information
6. Enter string for pathfinding.
7. Generate pathways.
8. Display pathway.
9. Save keyboard.
10. Load edited keyboard.
0. Exit.
1
Enter a file name:
key_netflix.txt
```



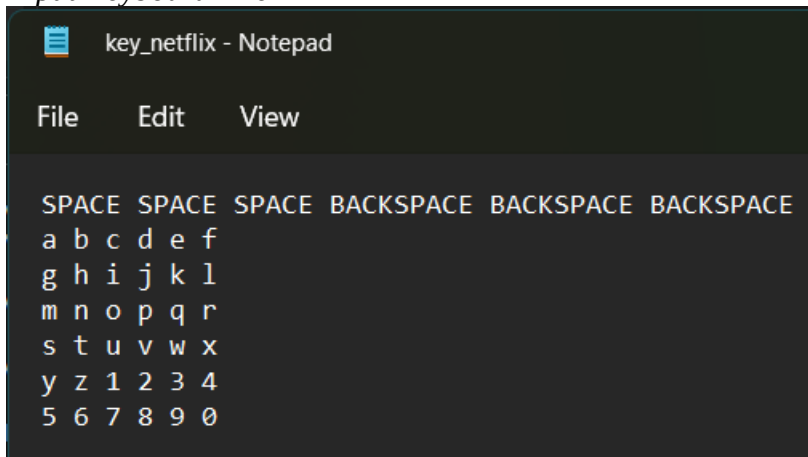
```
6. Enter string for pathfinding.
7. Generate pathways.
8. Display pathway.
9. Save keyboard.
10. Load edited keyboard.
0. Exit.
6
Enter a string to enter on a keyboard:
Hello, it's a dreary morning. It's going to be sunny on Thursday though!
```

```

6. Enter string for pathfinding.
7. Generate pathways.
8. Display pathway.
9. Save keyboard.
10. Load edited keyboard.
0. Exit.
7
Generating pathway...
Would you like to print them to the screen now?

```

- *Input Keyboard File*



```

key_netflix - Notepad

File Edit View

SPACE SPACE SPACE BACKSPACE BACKSPACE BACKSPACE
a b c d e f
g h i j k l
m n o p q r
s t u v w x
y z 1 2 3 4
5 6 7 8 9 0

```

- *Output File: See s1out.txt for full output. Included in zip file under documents section. Snippet included here.*

```

Path for String: Hello, it's a dreary morning. It's going to be sunny on Thursday though!
Couldn't find char: H on the keyboard.
a --> Up --> SPACE SPACE --> Right --> BACKSPACE BACKSPACE --> Down --> e [SELECT]
e --> Right --> f f --> Down --> l [SELECT]
l [SELECT]
l --> Left --> k k --> Left --> j j --> Left --> i i --> Down --> o [SELECT]
Couldn't find char: , on the keyboard.
o --> Up --> i i --> Up --> c c --> Up --> SPACE [SELECT]
SPACE --> Down --> c c --> Down --> i [SELECT]

```

- **Discussion**

As seen the input file reads in the keyboard in a 2D array and generates the pathway for the string. Due to the lack of symbols or capitals on the keyboard, the graph simply doesn't have them. There is no wrap around on this keyboard as reflected in the results and the keyboard includes " " by using the SPACE alias in the input file.

*The program reads in elements as strings delimited by " ". This means users can use aliases for keys that are difficult to represent in text files (such as the world symbol on the switch keyboard).*

## b) Scenario 2:

- *Setup:* key\_iview.txt has been setup to represent the keyboard image. I will be using the interactive keyboard to enter the string listed in section 6.1. I will then generate the pathways and output them to a file called "s2out.txt". This keyboard will require more edges than scenario 1 as key\_iview.txt requires wrapping around the left and right sides. Furthermore, many more edges and nodes will be added as there are upper and lowercase keys.

- *Commands*

```
Windows PowerShell
Select an option from the menu:
1. Load keyboard from file.
2. Use node operations menu (find, insert, delete, update nodes)
3. Use edge operations menu (find, insert, delete, update edges)
4. Display graph.
5. Display graph information
6. Enter string for pathfinding.
7. Generate pathways.
8. Display pathway.
9. Save keyboard.
10. Load edited keyboard.
0. Exit.
1
Enter a file name:
key_iview.txt
```

```
6. Enter string for pathfinding.
7. Generate pathways.
8. Display pathway.
9. Save keyboard.
10. Load edited keyboard.
0. Exit.
6
Enter a string to enter on a keyboard:
Hello, it's a dreary morning. It's going to be sunny on Thursday though!
```

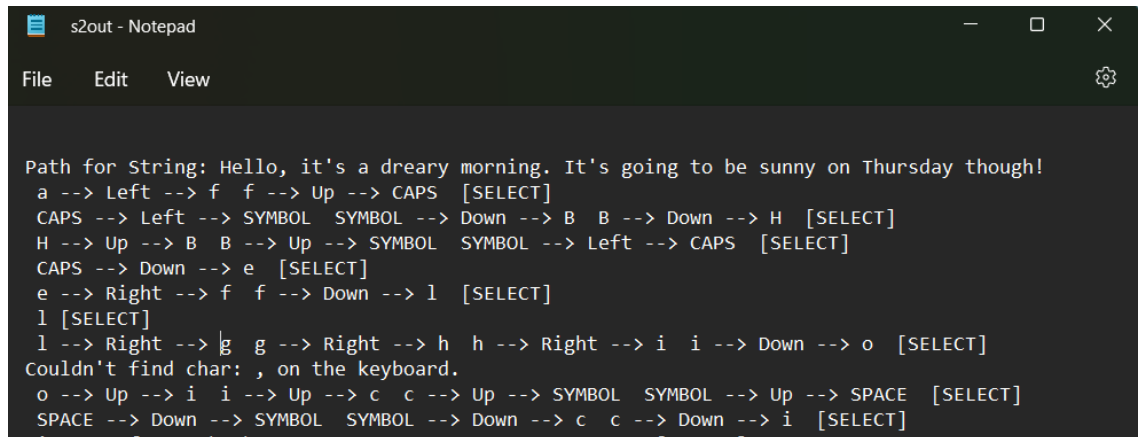
```
Would you like to save path to a file?
1. Yes
2. No
1
Enter a file name:
s2out.txt
```

- *Input File*

```
key_iview - Notepad
File Edit View

w
t b l r
a b c d e f
g h i j k l
m n o p q r
s t u v w x
y z 1 2 3 4
5 6 7 8 9 0
SPACE SPACE BACKSPACE BACKSPACE CLEAR CLEAR
SYMBOL SYMBOL SYMBOL CAPS CAPS CAPS
A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Y Z 1 2 3 4
5 6 7 8 9 0
SPACE SPACE BACKSPACE BACKSPACE CLEAR CLEAR
SYMBOL SYMBOL SYMBOL CAPS CAPS CAPS
```

- *Output File: See s2out.txt for full output. Included in zip file under documents section.*



```

s2out - Notepad
File Edit View

Path for String: Hello, it's a dreary morning. It's going to be sunny on Thursday though!
a --> Left --> f f --> Up --> CAPS [SELECT]
CAPS --> Left --> SYMBOL SYMBOL --> Down --> B B --> Down --> H [SELECT]
H --> Up --> B B --> Up --> SYMBOL SYMBOL --> Left --> CAPS [SELECT]
CAPS --> Down --> e [SELECT]
e --> Right --> f f --> Down --> l [SELECT]
l [SELECT]
l --> Right --> g g --> Right --> h h --> Right --> i i --> Down --> o [SELECT]
Couldn't find char: , on the keyboard.
o --> Up --> i i --> Up --> c c --> Up --> SYMBOL SYMBOL --> Up --> SPACE [SELECT]
SPACE --> Down --> SYMBOL SYMBOL --> Down --> c c --> Down --> i [SELECT]
i --> Left --> h h --> Down --> n n --> Down --> t t [SELECT]

```

- **Discussion**

Unlike scenario 1, key\_iview.txt contains 2 additional lines at the top. The first line uses w as a marker to tell the program that this keyboard wraps around. The program will then read the next line and use that to determine the direction of wrapping (t = top, etc).

This is reflected in the output file where the keyboard moves off the left side of the keyboards and off the top in order to get to the CAPS key. However the program cannot path find for symbols as this file doesn't include them (as it wasn't given in the images).

## 7. Conclusion and Future Work:

Implement a method to establish a gateway key between keyboards. The graph is split into 2 or 3 large sections where the CAPS or SPECIAL CHAR keys act as a pathway connecting the two. E.g. they are an edge/node that must be passed through to access other sections of the keyboard.

Implementation felt clumsy in sections in the graph as I expanded on it. I would refine the use of private/public methods and refine the functions in the Graph to be more concise as some functions feel like their 2 or 3 lines of code away from being redundant. Probably would have clearer and streamlined wrapper methods. More robust exception handling.

Extensions could see a comparison of multiple pathways and their efficiency, a counter of the number of steps and a comparison

Diagonal movement would be interesting to see as the pathway output would be smaller but an increased number of edges would test the scalability of searching and pathing functions.

Limitations and requirements weren't met.

Noted that there will be a limit on the program for the number of nodes and edges and the overall size of the graph. Due to serialisation and recursion implementation the program will stack overflow if input files are too large.



## 8. Test Scripts

### 1. *Driver/Menu & Modes*

- 1.1. [CLI] "Java keyMeUp"
- 1.2. [CLI] "Java keyMeUp -s strFile.txt key\_netflix.txt outFile.txt"
  - Copies of the files can be found under test\_files folder.
- 1.3. [CLI] "Java keyMeUp -w as.txt"
- 1.4. [CLI] "Java keyMeUp -i"
- 1.5. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_stan.txt"
  - [Menu] "0"

### 2. *Processing*

- 2.1. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_stan.txt"
  - [Menu] "6"
  - [File Input] "electric boogaloo"
  - [Menu] "7"
- 2.2. CONT from above
  - [Menu] "8"
- 2.3. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_iview.txt"
  - [Menu] "6"
  - [File Input] "Electric Boogaloo 1z27"
  - [Menu] "7"
  - [Menu] "8"
- 2.4. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_switch.txt"
  - [Menu] "6"
  - [File Input] "eLeCtrlc~bOOgaloo`"
  - [Menu] "7"
  - [FAILED]: Output is erroneous, does not move correctly for some elements,

### 3. *Input Files & 4. Keyboard Representation*

- 3.1. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_switch.txt"
  - [Menu] "4"
  - [Menu] "0"
- 3.2. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_iview.txt"
  - [Menu] "4"
  - [Menu] "0"
- 3.3. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_stan.txt"
  - [Menu] "4"
  - [Menu] "0"

- 3.4. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_netflix.txt"
  - [Menu] "4"
  - [Menu] "0"
- 3.5. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_unknown.txt"
  - [Menu] "4"
  - [Menu] "0"

4.

#### 5. **Menu Operations**

- 5.1. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_iview.txt"
  - [Menu] "4"
  - [Menu] "0"
- 5.2. [CLI] "Java keyMeUp -i"
  - [Menu] "2"
  - [Menu] "0"
  - [Menu] "0"
- 5.3. [CLI] "Java keyMeUp -i"
  - [Menu] "2"
  - [Menu] "0"
  - [Menu] "0"
- 5.4. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_iview.txt"
  - [Menu] "4"
  - [Menu] "0"
- 5.5. CONTINUED FROM 5.4.
  - [Menu] "5"
- 5.6. CONTINUED FROM 5.5.
  - [Menu] "6"
  - [User Input] "sssdE12 fEw3"
- 5.7. CONTINUED FROM 5.6
  - [Menu] "7"
- 5.8. CONTINUED FROM 5.7
  - [Menu] "8"
- 5.9. CONTINUED FROM 5.8
  - [Sub-Menu Option to output to file] "1"
  - [User Input file name] "testOut59.txt"
- 5.10. [CLI] "Java keyMeUp -i"
  - [Menu] "1"
  - [Menu] "key\_netflix.txt"
  - [Menu] "2"
  - [Sub-Menu] "2"
  - [Insert] "peter"
  - [Insert] "value1"
  - [Sub-Menu] "2"
  - [Insert] "lilah"
  - [Insert] "value2"

- [Menu] “10”
  - [User Input file name] “serKey.txt”
- 5.11. [CLI] “Java keyMeUp -i”
- [Menu] “11”
  - [User Input file name] “serKey.txt”
  - [Menu] “4”
  - [Menu] “0”

## 6. *Sub-Menu Edge Operations*

- 6.1. [CLI] “Java keyMeUp -i”
- [Menu] “1”
  - [Menu] “key\_netflix.txt”
  - [Menu] “3”
  - [Sub-Menu] “1”
  - [Sub-Menu] “a”
  - [Sub-Menu] “b”
  - [Sub-Menu] “0”
  - [Menu] “0”
- 6.2. [CLI] “Java keyMeUp -i”
- [Menu] “3”
  - [Sub-Menu] “2”
  - [Sub-Menu] “Peter”
  - [Sub-Menu] “Dickens”
  - [Sub-Menu] “Lee”
  - [Sub-Menu] “0”
  - [Menu] “4”
  - [Menu] “0”
- 6.3. [CLI] “Java keyMeUp -i”
- [Menu] “1”
  - [Menu] “key\_netflix.txt”
  - [Menu] “3”
  - [Sub-Menu] “1”
  - [Sub-Menu] “a”
  - [Sub-Menu] “b”
  - [Sub-Menu] “ThisIsTheUpdatedValue”
  - [Sub-Menu] “0”
  - [Menu] “4”
  - [Menu] “0”

## 7. *Sub-Menu Node Operations*

- 7.1. [CLI] “Java keyMeUp -i”
- [Menu] “1”
  - [Menu] “key\_netflix.txt”
  - [Menu] “2”
  - [Sub-Menu] “1”
  - [Sub-Menu] “5”
  - [Menu] “0”
- 7.2. [CLI] “Java keyMeUp -i”
- [Menu] “1”
  - [Menu] “2”
  - [Sub-Menu] “2”
  - [Sub-Menu] “NewKey”
  - [Sub-Menu] “just\_a\_big\_key”
  - [Sub-Menu] “0”

- [Menu] “4”
- [Menu] “0”

## **8. *Silent Mode***

8.1. [CLI] “Java keyMeUp -s key\_netflix.txt str\_file.txt out.txt”

## **9. *Retrieve file name***

9.1. [CLI] “Java keyMeUp -i”

- [Menu] “1”
- [Menu] “key\_netflix.txt”
- [Menu] “0”