End User Documentation

This documentation is about using and running the source code for the project *2D String Matching and Pattern Identification*.

The entire source code is written in c++ and your system must have g++ installed. The User Interface is designed to run in Linux's terminal.

The project involves implementation of four 2D string matching algorithms, they are:

1. Naive 2D pattern matching
2. Baker Bird 2D matching algorithm
3. Baeza-Yates Régnier algorithm
4. An algorithm using Hash values

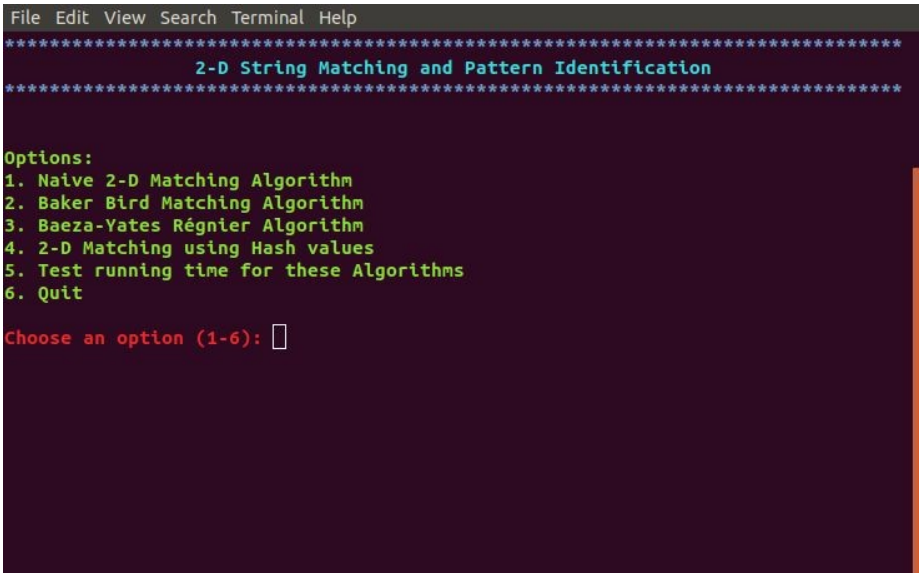
Setting Up

A compiled file '*2Dmatching*' is already present, but if you need to recompile, there is a Makefile to do the job. Just go to the folder where you saved the project in the terminal and run the command 'make'. This will compile all the source code files.

To run the program, write the command './2Dmatching' .

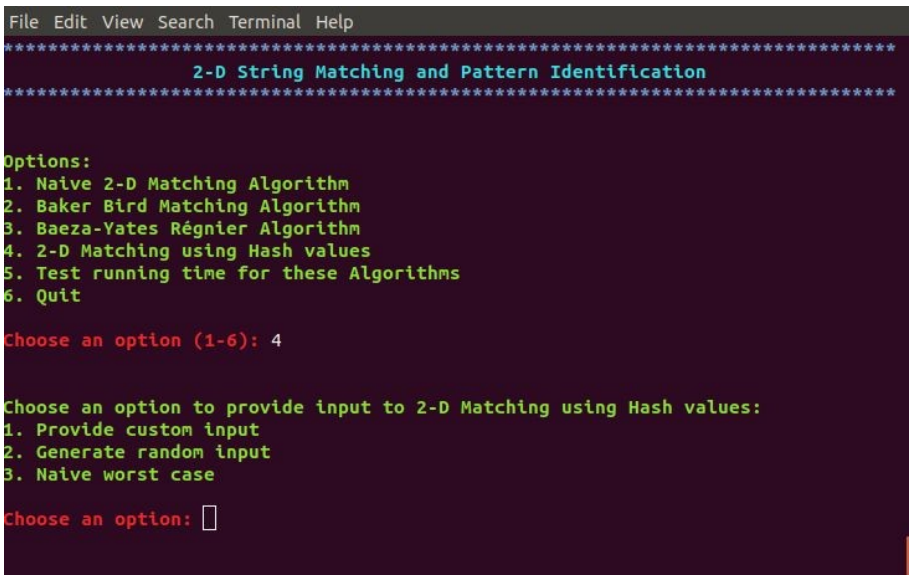
The User Interface

After you run the program, you will see a window with a home screen like this:



You can choose options 1-4 to run the corresponding algorithms.

After you choose an option from 1-4, you will be asked you you would like to provide the input to the algorithm. The input options are as shown in the figure below.



If you choose input option 1, you will need to provide text array dimension, then its content in row major order, then pattern array dimension, and finally the content of pattern array in row major order. The pattern array size for the '*Baeza-Yates Régnier algorithm*' and '*algorithm using Hash values*' must be less that 32 and for '*Baeza-Yates Régnier algorithm*' it should be greater than 2. The text array size must not exceed 1024 because of memory allocation issues and the pattern array size must be less than or equal to the text array size.

The input option 2 will generate random inputs. You just need to specify the dimensions of text array and pattern array. Then it will automatically generate the text array and pattern array will be a random sub-matrix of the text array. So, it is sure that we will have a match at some location.

The input option 3 will automatically provide the input case in which running time of the Naive algorithm is the worst.

After the input is provided, the expected and the actual output will be displayed. Also, the clock cycles taken by the algorithm will be displayed. The matches will be displayed as location *(i, j)* which refers to location *(i, j)* of the text array, *T[0 .. n-1] [0 .. n-1]*, where n is the size of text array. *(i, j)* denotes that the match was found starting at this location.

If you choose option 4: '*test running time for these algorithms*', all the four algorithms will run on different text array sizes and pattern array sizes and clock cycles taken will be stored on files inside the folder '*test-results*'.

If now you want to go back to the home screen, you will be asked to Enter any Input. You need to type anything and press enter. It will take you back to the home screen and you can choose any option again.