# Bilkent University
# CS481: Bioinformatics Algorithms
## Homework Assignment #3
## Fall 2020

---

### INSTRUCTIONS

- Solve the following problems.
- You must write your code yourself. Sufficient evidence of plagiarism will be treated the same as for plagiarism or cheating.
- Non-compiling submissions will not be evaluated.
- Your code must be complete.
- Do not submit the program binary. You must submit the following items:
  - All of the source files
  - A script to compile the source code and produce the binary (`Makefile`).
  - A `README.txt` file that describes how the compilation process works.
- Submit your answers **ONLY** through the Moodle page.
- **Zip** your files and send them in only one zipped file. File name format `surname_name_hw#.zip`
- C / C++, Python 3, Java will be used as programming language. STL is allowed. The use of `getopt` function is **compulsory** for C/C++ programs. Python programs **MUST** use `argparse` module. Java programs **MUST** use an argument parser such as `ArgParser`
- All submissions will be compiled and tested on **Dijkstra server**.
- All submissions must be made strictly before the stipulated deadline.
- The overall fastest implementation wins. **Bonus** will be given for the fastest code.

## 1) AHO-CORASICK ALGORITHM

**Aim:** In this assignment, given some patterns and a long text, you must implement the Aho-Corasick algorithm and search for the patterns in the text. You can assume that the patterns and text are given properly. The index starts from 0 and space will be considered as a character.

Input will be taken from the commandline as a filename using the `-i flag`. Said file will contain 2 lines. The first line will hold patterns that are separated by space and the text is given in the second line. Output will be printed in the standard output.

In the build tree part of output, each row will contain a **char** that is the character the node represents, **next states** that is a list of the IDs of the child nodes, **fail state** that is the ID of the fail state, and **output** that is a list of all the complete keywords encountered so far as we have gone through the input text. In the search part of the same output file, each row will contain a keyword and its index.

## 2) EXAMPLE

*Input file*:

```
carrot cool parrot car carpet rotate
once upon a time there was a cool parrot who loves eating carrot on the
   carpet
```

*Output*:

```
Build tree
-----------------------------------------------------------------------
char:   next states: [1, 10, 19] fail state: 0 output: []
char: c next states: [2, 7] fail state: 0 output: []
char: a next states: [3] fail state: 0 output: []
char: r next states: [4, 16] fail state: 19 output: ['car']
char: r next states: [5] fail state: 19 output: []
char: o next states: [6] fail state: 20 output: []
char: t next states: [] fail state: 21 output: ['carrot']
char: o next states: [8] fail state: 0 output: []
char: o next states: [9] fail state: 0 output: []
char: l next states: [] fail state: 0 output: ['cool']
char: p next states: [11] fail state: 0 output: []
char: a next states: [12] fail state: 0 output: []
char: r next states: [13] fail state: 19 output: []
char: r next states: [14] fail state: 19 output: []
char: o next states: [15] fail state: 20 output: []
char: t next states: [] fail state: 21 output: ['parrot']
char: p next states: [17] fail state: 10 output: []
char: e next states: [18] fail state: 0 output: []
char: t next states: [] fail state: 0 output: ['carpet']
char: r next states: [20] fail state: 0 output: []
char: o next states: [21] fail state: 0 output: []
char: t next states: [22] fail state: 0 output: []
char: a next states: [23] fail state: 0 output: []
char: t next states: [24] fail state: 0 output: []
char: e next states: [] fail state: 0 output: ['rotate']

Search
-----------------------------------------------------------------------
keyword: cool index: 29
keyword: parrot index: 34
keyword: car index: 58
keyword: carrot index: 58
keyword: car index: 72
keyword: carpet index: 72
```