OCAML KMP

...But Functional

What is KMP?

Two parts:

- 1) The runtime (no backtracking)
- 2) The “partial match” Table (len
  o proper prefix and suffix)
- Allows “skipping” when backtracking

```
1  char:  | a | b | a | b | a | b | c | a |
2  index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
3  value: | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 1 |
```

Ocaml KMP Table

...But Bad

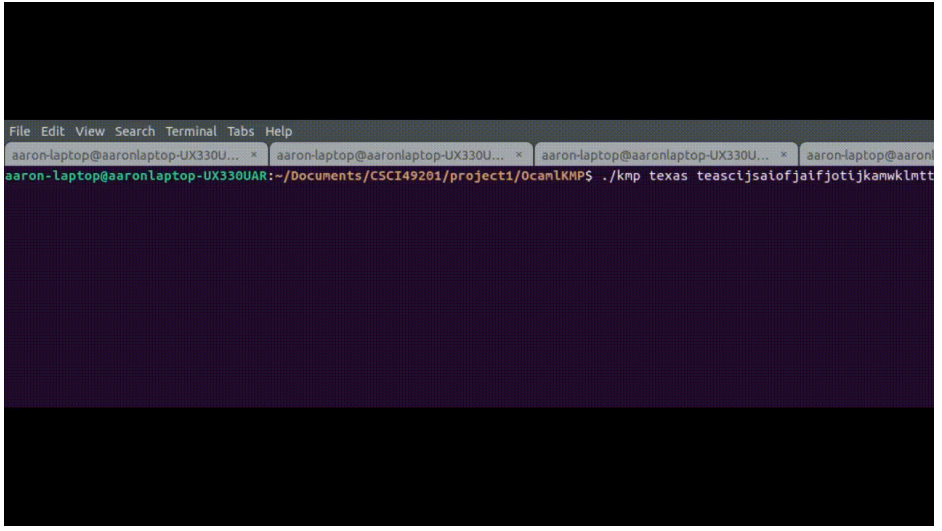
```
let init pattern m =
  let table = Array.create m 0 in
  for j = 2 to m - 1 do
    let s = assertInterrupted (
      loop pattern m pattern j table table.(j - 1) (j - 1)
    ) in
    table.(j) <-
      if s > 0 && pattern.[s] = pattern.[j] then table.(s)
      else s
  done;
  table
```

The True Way

```
open Lazy


(*Creates pattern structure that will keep track of next step in pattern *)
type pattern = { is_match : bool; step : char -> pattern}
let is_match = function
        |{is_match = true; step = _} -> true
        | _ -> false

let mk b f = {is_match = b; step = f}
let next c p = p.step c
let rec const b = {is_match = b; step = fun _ -> const b}

let rec to_list_ch = function
        |""-> []
        | ch -> (String.get ch 0 ) :: (to_list_ch ( String.sub ch 1 ( (String.length ch) -1 )))

(*checks against pattern *)
let run ( pattern: pattern) (text : string) (ln_pattern : int) : int list =
        let accum ( pattern, ix, acc) c =
                let pattern = next c pattern in
                let acc = if is_match pattern then ((ix - ln_pattern) :: acc) else acc in
                (pattern, ix+1, acc)
        in
        let (_,_,acc) = List.fold_left accum (pattern, 0, []) (to_list_ch text) in
        List.rev acc

let generate_pattern (cs: char list) : pattern =
        let rec pattern = lazy (gen pattern cs)
        and gen curr_pattern = function
                | [] -> const true
                | [c] -> mk false @@ fun x ->
                                let next_pattern = force curr_pattern in
                                if x = c then mk true (fun _ ->next_pattern)
                                else next_pattern
                |c :: cs ->
                                let next_pattern = lazy (next c @@ force curr_pattern) in
                                let cont_pattern = lazy (gen next_pattern cs) in
                                mk false @@ fun x -> force @@ if x = c then cont_pattern
                                                              else curr_pattern
        in
        force pattern

let search cs text= run ( generate_pattern cs) text
```

The power of lazy

- Runs on CMD args & files
- Lazy Module suspends computation
- https://imgur.com/hjTipPr



Fight!

Functional

```
aaron-laptop@aaronlaptop-UX330UAR:~/Documents/CSCI49201/project1/temp$ ./kmp pattern.txt text.txt

Execution time: 0.000004s
```

Non-functional

```
aaron-laptop@aaronlaptop-UX330UAR:~/Documents/CSCI49201/project1/temp$ ./kmp pattern.txt text.txt
90
Execution time: 0.000003s
```

Short-comings and Thanks

- Pattern.step’s char uneeded.
- Lazy not thread-safe
- File-reading limit (192)

- Thanks:
- Joel Björnson
- Ager’s, Danvy’s, Rohde’s *On Obtaining KMP String Matcher by Partial Evaluation*
- Cieslak’s *A Functional Version of the KMP algorithm*
- Alexey Nikolaev