# Interface for module Kmp

**1. Knuth-Morris-Pratt string search algorithm.**

**2.** *search* $p$ $t$ searches the first occurrence of pattern $p$ in text $t$. It raises *Not_found* if $p$ does not occur in $t$. Strings are 0-based. Both arguments can be empty. If $p$ is to be searched many times in different texts then a partial application *search* $p$ is more efficient (the shift table is computed only once).

Complexity: if $m$ is the length of $p$ and $n$ the length of $t$ then *search* $p$ $t$ runs in time $O(m + n)$ and uses $O(m)$ space.

```
val search : string → string → int
```

**3.** Functorial interface. Knuth-Morris-Pratt algorithm can be applied to patterns and texts of any type using the following functor. Patterns and texts may be of different types, as soon as the type of characters is common. *length* $s$ must return the length of $s$, and *get* $s$ $i$ must return the $i$-th character of $s$ (starting from 0). Equality on characters is supposed to be structural equality (=).

```
module type STRING = sig
  type t
  type char
  val length : t → int
  val get : t → int → char
end

module Make(P : STRING)(T : STRING with type char = P.char) : sig

  val search : P.t → T.t → int

end
```

**4.** Debugging: setting the following boolean reference to `true` will dump the shift table on error output.

```
val debug : bool ref
```

# Module Kmp

**5.** Knuth-Morris-Pratt implementation. The shift table is called *next* here and stored in an array.

```
open Printf
```

```
let debug = ref false

let search p =
  let m = String.length p in
  let next = Array.create m 0 in
  (* initialization of next *)
  let i = ref 1 in
  let j = ref 0 in
  if m > 1 then begin
    while !i < m - 1 do
      if p.[!i] = p.[!j] then begin
        i := !i + 1; j := !j + 1; next.(!i) ← !j
      end else
        if !j = 0 then i := !i + 1 (* nexti ¡- 0 *) else j := next.(!j)
    done
  end;
  (* debug: dump of the next table *)
  if !debug then
    for i = 0 to m - 1 do eprintf "next[%d]=%d\n" i next.(i) done;
  fun t →
    (* search in t *)
    let n = String.length t in
    i := 0;
    j := 0;
    while !j < m ∧ !i < n do
      if t.[!i] = p.[!j] then begin
        i := !i + 1; j := !j + 1
      end else
        if !j = 0 then i := !i + 1 else j := next.(!j)
    done;
    if !j = m then !i - m else raise Not_found
```

**6.** Functorial interface.

```
module type STRING = sig
  type t
  type char
  val length : t → int
  val get : t → int → char
end

module Make(P : STRING)(T : STRING with type char = P.char) = struct
```

```
let search p =
  let m = P.length p in
  let next = Array.create m 0 in
  (* initialization of next *)
  let i = ref 1 in
  let j = ref 0 in
  if m > 1 then begin
    while !i < m - 1 do
      if P.get p !i = P.get p !j then begin
        i := !i + 1; j := !j + 1; next.(!i) ← !j
      end else
        if !j = 0 then i := !i + 1 else j := next.(!j)
    done
  end;
  fun t →
    (* search in t *)
    let n = T.length t in
    i := 0;
    j := 0;
    while !j < m ∧ !i < n do
      if T.get t !i = P.get p !j then begin
        i := !i + 1; j := !j + 1
      end else
        if !j = 0 then i := !i + 1 else j := next.(!j)
    done;
    if !j = m then !i - m else raise Not_found

end
```

# Index