String Matching: Boyer-Moore algorithm

Madhavan Mukund

`https://www.cmi.ac.in/~madhavan`

Programming, Data Structures and Algorithms using Python

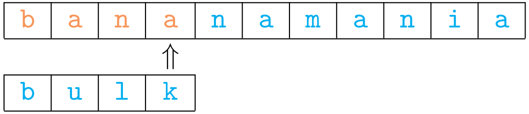
Week 10

Speeding up the brute force algorithm

- Text `t`, pattern `p` of of lengths n, m
- For each starting position `i` in `t`, compare `t[i:i+m]` with `p`
  - Scan `t[i:i+m]` right to left

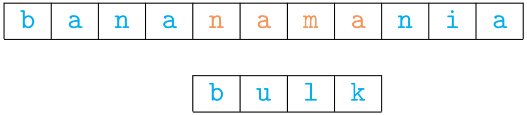
Speeding up the brute force algorithm

- Text `t`, pattern `p` of of lengths n, m
- For each starting position `i` in `t`, compare `t[i:i+m]` with `p`
    - Scan `t[i:i+m]` right to left
- While matching, we find a letter in `t` that does not appear in `p`
    - `t = bananamania`, `p = bulk`



Speeding up the brute force algorithm

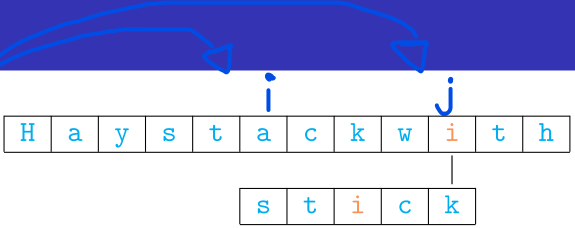
- Text `t`, pattern `p` of of lengths *n*, *m*
- For each starting position `i` in `t`, compare `t[i:i+m]` with `p`
  - Scan `t[i:i+m]` right to left
- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`
- Shift the next scan to position after mismatched letter
- What if the mismatched letter does appear in `p`?



Sliding the search

- Suppose `c = t[i+j] != p[j]`, but `c` does occur somewhere in `p[j]`

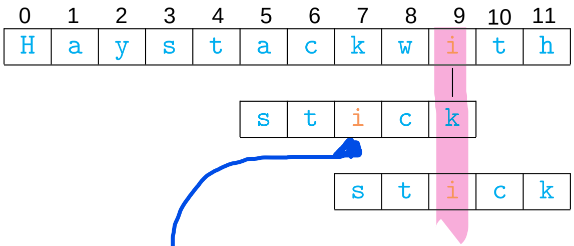
i - index of text
j - index of pattern



1. We find "i" != "k"
2. But "i" is present in pattern (p)
3. So, shifting slice by one does not make sense as we want to get "i" (in p) aligned with "i" (in t)

Sliding the search

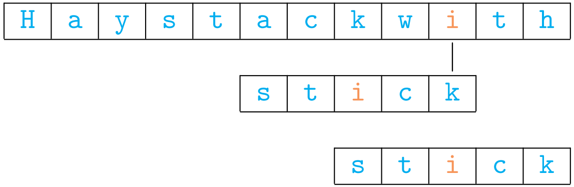
- Suppose c = t[i+j] != p[j], but c does occur somewhere in p[j]
- Align rightmost occurrence of c in p with t[i+j]



1. Here t[i+j] = "i"
2. Rightmost occurrence of "i" in p
3. i = 5, j = 4

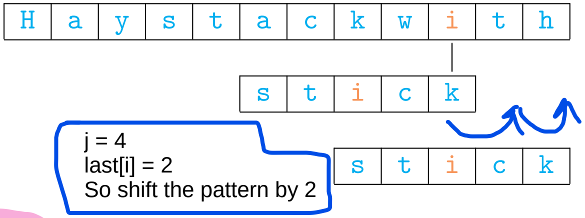
Sliding the search

- Suppose `c = t[i+j] != p[j]`, but `c` does occur somewhere in `p[j]`
- Align rightmost occurrence of `c` in `p` with `t[i+j]`
- Scan this substring of `t` next



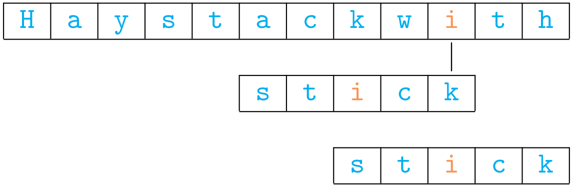
Sliding the search

- Suppose `c = t[i+j] != p[j]`, but `c` does occur somewhere in `p[j]`
- Align rightmost occurrence of `c` in `p` with `t[i+j]`
- Scan this substring of `t` next
- Use a dictionary `last`
  - For each `c` in `p`, `last[c]` records right most position of `c` in `p`
  - Shift pattern by `j - last[c]`



Sliding the search

- Suppose `c = t[i+j] != p[j]`, but `c` does occur somewhere in `p[j]`
- Align rightmost occurrence of `c` in `p` with `t[i+j]`
- Scan this substring of `t` next
- Use a dictionary `last`
  - For each `c` in `p`, `last[c]` records right most position of `c` in `p`
  - Shift pattern by `j - last[c]`
- If `c` not in `p`, shift pattern by `j+1`



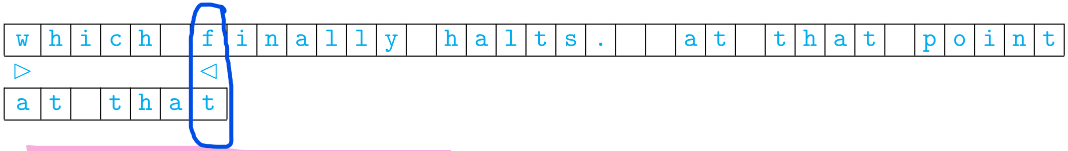
We saw this case in last lecture where "a" was not present in p "bulk" so we shifted the pattern by 4

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`



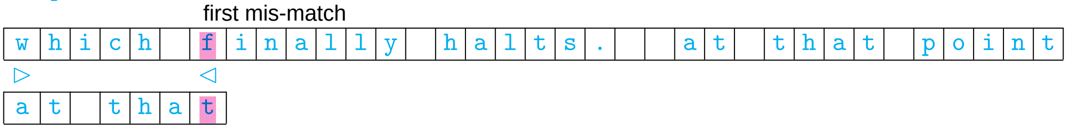
w h i c h   f i n a l l y   h a l t s .   a t   t h a t   p o i n t
▷ ◁
a t   t h a t

We start comparing from RIGHT to LEFT

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`

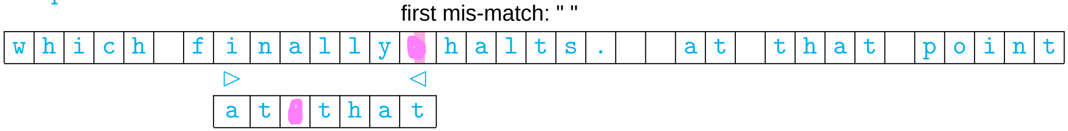
  `p = "at that"`


first mis-match
w h i c h   f i n a l l y   h a l t s .   a t   t h a t   p o i n t
▷ ◁
a t   t h a t

- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`

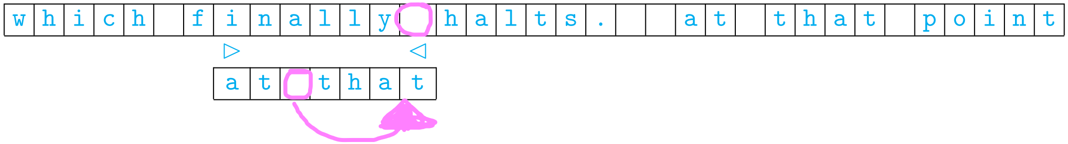


- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`

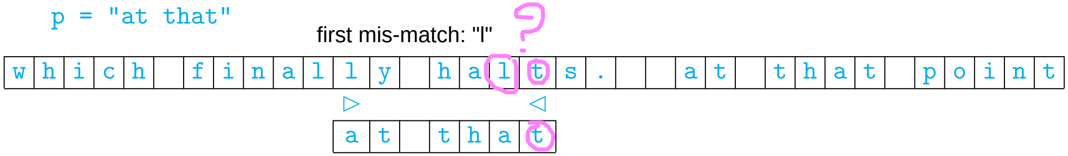
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11

Example [Boyer, Moore 1977]

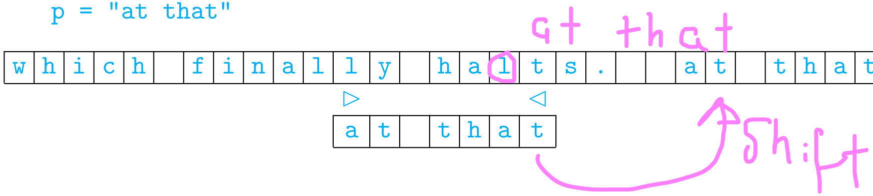
- `t = "which finally halts.  at that point"`
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`



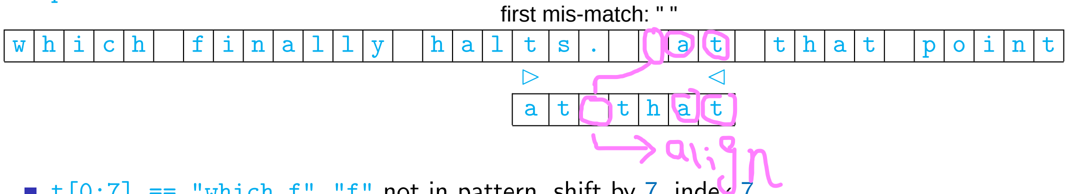
w h i c h   f i n a l l y   h a l t s .   a t   t h a t   p o i n t
a t   t h a t
at that
shift

- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`

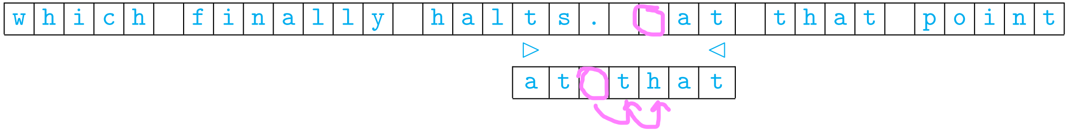
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`

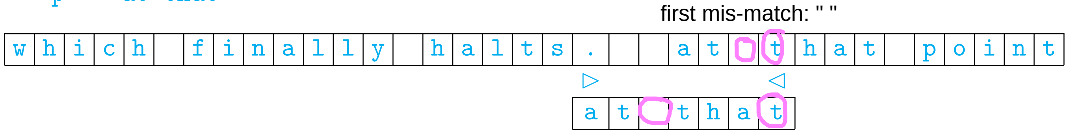


- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`

  `p = "at that"`

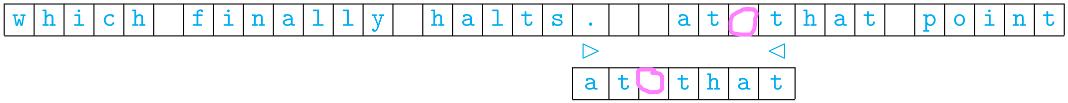


- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`

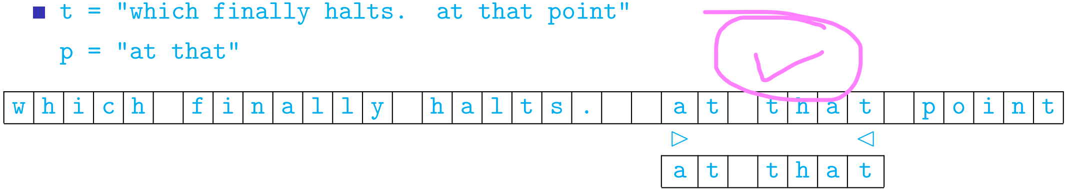
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`



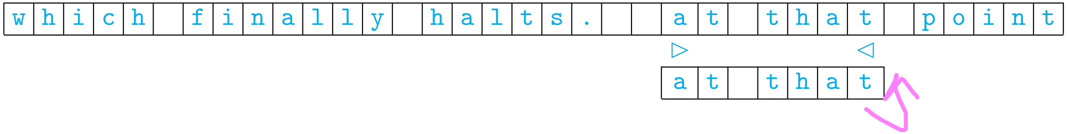
w h i c h   f i n a l l y   h a l t s .   a t   t h a t   p o i n t
a t   t h a t

- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22

NOTE: We cannot stop checking from here, as there could be more matches in the text.
NOTE: We shift by one as there can be OVERLAPPING pattern. Example t = "aaa" p = "aa"

Example [Boyer, Moore 1977]

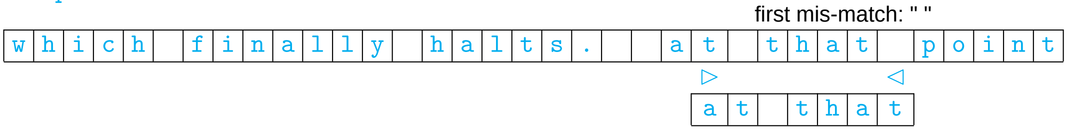
- `t = "which finally halts.  at that point"`
  `p = "at that"`


w h i c h   f i n a l l y   h a l t s .   a t   t h a t   p o i n t
▷ ◁
a t   t h a t

- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22
- `t[22:29] == "at that"`, report match at index 22, shift by 1, index 23

Example [Boyer, Moore 1977]

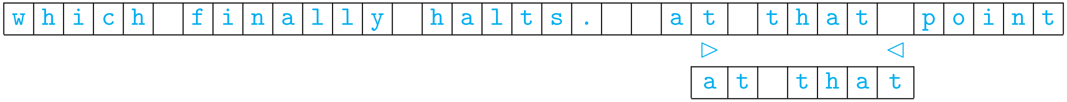
- `t = "which finally halts.  at that point"`
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22
- `t[22:29] == "at that"`, report match at index 22, shift by 1, index 23

Example [Boyer, Moore 1977]

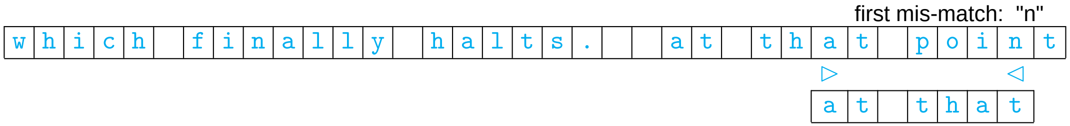
- `t = "which finally halts.  at that point"`
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22
- `t[22:29] == "at that"`, report match at index 22, shift by 1, index 23
- `t[23:30] == "t that "`, `" "` in pattern, shift by 4, index 27

Example [Boyer, Moore 1977]

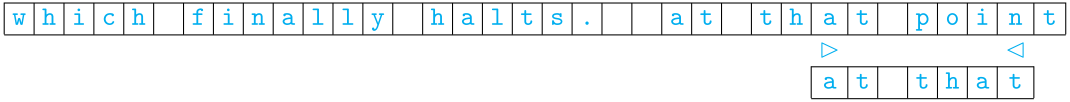
- `t = "which finally halts.  at that point"`
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22
- `t[22:29] == "at that"`, report match at index 22, shift by 1, index 23
- `t[23:30] == "t that "`, `" "` in pattern, shift by 4, index 27

Example [Boyer, Moore 1977]

- `t = "which finally halts.  at that point"`
  `p = "at that"`



- `t[0:7] == "which f"`, `"f"` not in pattern, shift by 7, index 7
- `t[7:14] == "inally "`, `" "` in pattern, shift by 4 to align `" "`, index 11
- `t[11:18] == "ly halt"`, `"l"` not in pattern, shift by 6, index 17
- `t[17:24] == "ts.  at"`, `" "` in pattern, shift by 2, index 19
- `t[19:26] == ".  at t"`, `" "` in pattern, shift by 3, index 22
- `t[22:29] == "at that"`, report match at index 22, shift by 1, index 23
- `t[23:30] == "t that "`, `" "` in pattern, shift by 4, index 27
- `t[27:34] == "at poin"`, `"n"` not in pattern, shift by 7, index 34, stop

Implementation

- Initialize `last[c]` for each `c` in `p`
    - Single scan, rightmost value is recorded

```
def boyermoore(t,p):
  last = {}                    # Preprocess
  for i in range(len(p)):
    last[p[i]] = i

  poslist,i = [],0             # Loop
  while i <= (len(t)-len(p)):
    matched,j = True,len(p)-1
    while j >= 0 and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j - 1
    if matched:
      poslist.append(i)
      i = i + 1
    else:
      j = j + 1
      if t[i+j] in last.keys():
        i = i + max(j-last[t[i+j]],1)
      else:
        i = i + j + 1
  return(poslist)
```

Implementation

- Initialize `last[c]` for each `c` in `p`
  - Single scan, rightmost value is recorded
- Nested loop, compare each segment `t[i:i+len(p)]` with `p`

```
def boyermoore(t,p):
  last = {}                    # Preprocess
  for i in range(len(p)):
    last[p[i]] = i

  poslist,i = [],0             # Loop
  while i <= (len(t)-len(p)):
    matched,j = True,len(p)-1
    while j >= 0 and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j - 1
    if matched:
      poslist.append(i)
      i = i + 1
    else:
      j = j + 1
      if t[i+j] in last.keys():
        i = i + max(j-last[t[i+j]],1)
      else:
        i = i + j + 1
  return(poslist)
```

Implementation

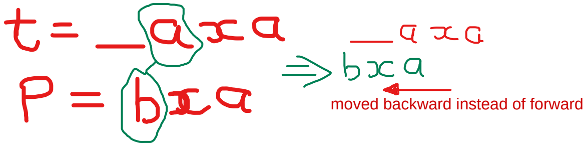
- Initialize `last[c]` for each `c` in `p`
  - Single scan, rightmost value is recorded
- Nested loop, compare each segment `t[i:i+len(p)]` with `p`
- If `p` matches, record and shift by `1`

As we decremented j after mismatch was found. So, we need to restore it, so increment j again by 1

```
def boyermoore(t,p):
  last = {}                    # Preprocess
  for i in range(len(p)):
    last[p[i]] = i

  poslist,i = [],0             # Loop
  while i <= (len(t)-len(p)):
    matched,j = True,len(p)-1
    while j >= 0 and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j - 1
    if matched:
      poslist.append(i)
      i = i + 1
    else:
      j = j + 1
      if t[i+j] in last.keys():
        i = i + max(j-last[t[i+j]],1)
      else:
        i = i + j + 1
  return(poslist)
```

Implementation



1. The first mis-match occurs at "a", not matching with "b"
2."a" is there in p but it is in front
3. In this case aligning p[j] and t[i+j] would move p backwards
4. We don't want this because "a" in "bxa" was already aligned in the past

- We find a mismatch at `t[i+j]`
  - If `j > last[t[i+j]]`, shift by `j - last[t[i+j]]`
  - If `last[t[i+j]] > j`, shift by `1`
    - Should not shift `p` to left!
  - If `t[i+j]` not in `p`, shift by `j+1`

```
def boyermoore(t,p):
  last = {}                    # Preprocess
  for i in range(len(p)):
    last[p[i]] = i

  poslist,i = [],0             # Loop
  while i <= (len(t)-len(p)):
    matched,j = True,len(p)-1
    while j >= 0 and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j - 1
    if matched:
      poslist.append(i)
      i = i + 1
    else:
      j = j + 1
      if t[i+j] in last.keys():
        i = i + max(j-last[t[i+j]],1)
      else:
        i = i + j + 1
  return(poslist)
```

Summary

- Worst case remains $O(nm)$
  - `t = aaa...a`, `p = baaa`

Summary

- Worst case remains $O(nm)$
    - `t = aaa...a`, `p = baaa`
- Without dictionary, computing `last` is a bottleneck, complexity is $O(|\Sigma|)$

Summary

- Worst case remains $O(nm)$
  - `t = aaa...a`, `p = baaa`
- Without dictionary, computing `last` is a bottleneck, complexity is $O(|\Sigma|)$
- Boyer-Moore works well, in practice
  - "Sublinear"
  - Experimentally — English text, 5 character pattern, average number of comparisons is 0.24 per character
  - Performance improves as pattern length grows — more characters skipped

Summary

- Worst case remains $O(nm)$
  - `t = aaa...a`, `p = baaa`
- Without dictionary, computing `last` is a bottleneck, complexity is $O(|\Sigma|)$
- Boyer-Moore works well, in practice
  - "Sublinear"
  - Experimentally — English text, 5 character pattern, average number of comparisons is 0.24 per character
  - Performance improves as pattern length grows — more characters skipped
- Often used in practice — `grep` in Unix