String Matching

Madhavan Mukund

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

Programming, Data Structures and Algorithms using Python

Week 10

String matching

- Searching for a pattern is a fundamental problem when dealing with text
  - Editing a document
  - Answering an internet search query
  - Looking for a match in a gene sequence

String matching

- Searching for a pattern is a fundamental problem when dealing with text
  - Editing a document
  - Answering an internet search query
  - Looking for a match in a gene sequence
- Example
  - `an` occurs in `banana` at two positions

String matching

- Searching for a pattern is a fundamental problem when dealing with text
  - Editing a document
  - Answering an internet search query
  - Looking for a match in a gene sequence
- Example
  - `an` occurs in `banana` at two positions
- Formally
  - A text string `t` of length n
  - A pattern string `p` of length m
  - Both `t` and `p` are drawn from an alphabet of valid letters, denoted Σ
  - Find every position `i` in `t` such that `t[i:i+m] == p`

Brute force

- Nested loop
  - For each starting position `i` in `t`, compare `t[i:i+m]` with `p`

```
def stringmatch(t,p):
  poslist = []
  for i in range(len(t)-len(p)+1):
    matched = True
    j = 0
    while j < len(p) and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j+1
    if matched:
      poslist.append(i)
  return(poslist)
```

Brute force

- Nested loop
  - For each starting position `i` in `t`, compare `t[i:i+m]` with `p`
- Nested search bails out at first mismatch

```
def stringmatch(t,p):
  poslist = []
  for i in range(len(t)-len(p)+1):
    matched = True
    j = 0
    while j < len(p) and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j+1
    if matched:
      poslist.append(i)
  return(poslist)
```

Brute force

- Nested loop
  - For each starting position `i` in `t`, compare `t[i:i+m]` with `p`
- Nested search bails out at first mismatch
- Worst case is $O(nm)$, for example
  - `t = aaa...a`, `p = aaab`

```
def stringmatch(t,p):
  poslist = []
  for i in range(len(t)-len(p)+1):
    matched = True
    j = 0
    while j < len(p) and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j+1
    if matched:
      poslist.append(i)
  return(poslist)
```

Brute force

- Nested loop
  - For each starting position `i` in `t`, compare `t[i:i+m]` with `p`
- Nested search bails out at first mismatch
- Worst case is $O(nm)$, for example
  - `t = aaa...a`, `p = aaab`
- Can also do nested scan from right to left
  - Worst case still $O(nm)$, `t = aaa...a`, `p = baaa`
  - Can reversing the scan help?

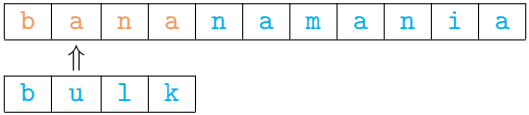
```
def stringmatchrev(t,p):
  poslist = []
  for i in range(len(t)-len(p)+1):
    matched = True
    j = len(p)-1
    while j >= 0 and matched:
      if t[i+j] != p[j]:
        matched = False
      j = j-1
    if matched:
      poslist.append(i)
  return(poslist)
```

Speeding things up

- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`

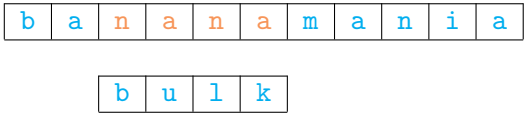
Speeding things up

- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`
- When we see `a` in `t`, we can shift the next scan to after `a`



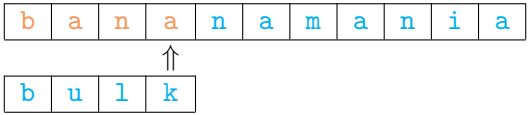
Speeding things up

- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`
- When we see `a` in `t`, we can shift the next scan to after `a`
  - If we scan from the left, we skip one position



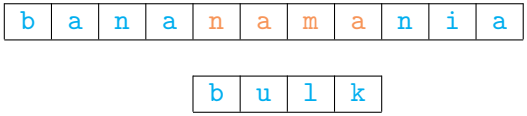
Speeding things up

- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`
- When we see `a` in `t`, we can shift the next scan to after `a`
  - If we scan from the left, we skip one position
  - If we scan from the right, we skip three positions



Speeding things up

- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`
- When we see `a` in `t`, we can shift the next scan to after `a`
  - If we scan from the left, we skip one position
  - If we scan from the right, we skip three positions



Speeding things up

- While matching, we find a letter in `t` that does not appear in `p`
  - `t = bananamania`, `p = bulk`
- When we see `a` in `t`, we can shift the next scan to after `a`
  - If we scan from the left, we skip one position
  - If we scan from the right, we skip three positions
- Don't need to check all of `t` to search for all occurrences of `p`!
- Formalized in Boyer-Moore algorithm