# MA/CSSE 473 Day 24

**Student questions**

**Quadratic probing proof**

**String search**

## FINISH THE HASHING DISCUSSION: QUADRATIC PROBING

# Collision Resolution: Quadratic probing

- With linear probing, if there is a collision at H, we try H, H+1, H+2, H+3, ... (all modulo the table size) until we find an empty spot.
  - Causes (primary) clustering
- With quadratic probing, we try H, $H+1^2$. $H+2^2$, $H+3^2$, ...
  - Eliminates primary clustering, but can cause secondary clustering.
  - Is it possible that it misses some available array positions?
  - I.e it repeats the same positions over and over, while never probing some other positions?

# Hints for quadratic probing

- **Choose a prime number for the array size, then …**
  - If the array is not more than half full, finding a place to do an insertion is guaranteed , and no cell is probed twice before finding it
  - Suppose the array size is P, a prime number greater than 3
  - Show by contradiction that if i and j are ≤ ⌊P/2⌋, and i≠j, then $H + i^2 \pmod P \not\equiv H + j^2 \pmod P$.
- **Use an algebraic trick to calculate next index**
  - Replaces mod and general multiplication with subtraction and a bit shift
  - Difference between successive probes:
    - $H + (i+1)^2 = H + i^2 + (2i+1)$ [can use a bit-shift for the multiplication].
    - nextProbe = nextProbe + (2i+1);
      if (nextProbe >= P) nextProbe -= P;

# Quadratic probing analysis

- No one has been able to analyze it
- Experimental data shows that it works well
  - Provided that the array size is prime, and is the table is less than half full

# Brute Force String Search Example

The problem: Search for the first occurrence of a **pattern** of length m in a **text** of length n.
Usually, m is much smaller than n.

- What makes brute force so slow?
- When we find a mismatch, we can shift the *pattern* by only one character position in the *text*.

```
Text:    abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
Pattern: abracadabra
          abracadabra
           abracadabra
            abracadabra
             abracadabra
              abracadabra
```

# Faster String Searching

- Brute force: **worst case m(n-m+1)** **Was a HW problem**
- A little better: **but still Θ(mn) on average**
  - **Short-circuit the inner loop**

```
def search(pattern, text):
    n, m = len(text),len(pattern)
    for i in range(n-m+1):
        j = 0
        while j < m and text[i+j] == pattern[j]:
            j +=1
        if j==m:
            return i
    return False
```

# What we want to do

- When we find a character mismatch
  - Shift the pattern as far right as we can
  - Without the possibility of skipping over a match.

# Horspool's Algorithm

- A simplified version of the Boyer-Moore algorithm
- A good bridge to understanding Boyer-Moore
- Published in 1980
- Recall: What makes brute force so slow?
  - When we find a mismatch, we can only shift the pattern to the right by one character position in the text.
  - ```
    Text:    abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
    Pattern: abracadabra
              abracadabra
               abracadabra
                abracadabra
    ```
- Can we sometimes shift farther?
  Like Boyer-Moore, Horspool does the comparisons in a counter-intuitive order (moves right-to-left through the pattern)

# Horspool's Main Question

- If there is a character mismatch, how far can we shift the pattern, with no possibility of missing a match within the text?
- What if the last character in the pattern is compared to a character in the text that does not occur anywhere in the pattern?
- ```
  Text:     ... ABCDEFG ...
  Pattern:      CSSE473
  ```

# How Far to Shift?

- Look at first (rightmost) character in the part of the text that is compared to the pattern:
- The character is not in the pattern

  .....**C**..... {*C* not in pattern)

  **BAOBAB**
- The character is in the pattern (but not the rightmost)

  .....**O**.....(O occurs once in pattern)

  **BAOBAB**

  .....**A**.....(A occurs twice in pattern)

  **BAOBAB**
- The rightmost characters do match

  .....**B**.....

  **BAOBAB**

# Shift Table Example

- Shift table is indexed by text and pattern alphabet
  E.g., for BAOBAB:

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

# Example of Horspool's Algorithm

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | - |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

BARD LOVED BANANAS (this is the text)

BAOBAB (this is the pattern)

BAOBAB

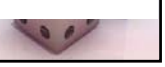
BAOBAB

BAOBAB (unsuccessful search)

# Horspool Code

```
def populateShiftTable(table, pattern, mMinusOne):
    for i in range(mMinusOne):
        table[ord(pattern[i])] = mMinusOne - i
```

```
def search(pattern, text):
    """ return index of first occurrence of pattern in text;
        return -1 if no match """
    n, m = len(text), len(pattern)
    shiftTable = [m]*128 # if char not in pattern, shift by full amount
    populateShiftTable(shiftTable, pattern, m-1)

    i = m - 1 # i is position in text that corresponds to end of pattern

    while i < n:  # while not past end of text
        k = 0 # k is number of pattern characters compared so far

        while k < m and pattern[m-1-k]==text[i-k]:
            k += 1;  # loop stops if mismatch or complete match

        if k==m:  # found a match
            return i - m + 1

        i = i + shiftTable[ord(text[i])] # ready to begin next comparison
    return -1
```



# Horspool Example

```
pattern = abracadabra
text =
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
shiftTable:  a3 b2 r1 a3 c6 a3 d4 a3 b2 r1 a3 x11
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
   abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
      abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
        abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
           abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
               abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                  abracadabra
```

Continued on next slide

# Horspool Example Continued

```
pattern = abracadabra
text =
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
shiftTable:  a3 b2 r1 a3 c6 a3 d4 a3 b2 r1 a3 x11

abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                  abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                             abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                   abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                                abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                                   abracadabra
49
```

Using brute force, we would have to compare the pattern to 50 different positions in the text before we find it; with Horspool, only 13 positions are tried.