

Hash Tables: String Search

Michael Levin

Higher School of Economics

Data Structures
Data Structures and Algorithms

Outline

- 1 Search Pattern in Text
- 2 Rabin-Karp's Algorithm
- 3 Improving Running Time

Searching for Patterns

Given a text T (book, website, facebook profile) and a pattern P (word, phrase, sentence), find all occurrences of P in T .

Searching for Patterns

Given a text T (book, website, facebook profile) and a pattern P (word, phrase, sentence), find all occurrences of P in T .

Examples

- Your name on a website

Searching for Patterns

Given a text T (book, website, facebook profile) and a pattern P (word, phrase, sentence), find all occurrences of P in T .

Examples

- Your name on a website
- Twitter messages about your company

Searching for Patterns

Given a text T (book, website, facebook profile) and a pattern P (word, phrase, sentence), find all occurrences of P in T .

Examples

- Your name on a website
- Twitter messages about your company
- Detect files infected by virus — code patterns

Substring Notation

Definition

Denote by $S[i..j]$ the substring of string S starting in position i and ending in position j .

Examples

If $S = \text{"abcde"}$, then

$S[0..4] = \text{"abcde"}$,

$S[1..3] = \text{"bcd"}$,

$S[2..2] = \text{"c"}$.

Find Pattern in Text

Input: Strings T and P .

Output: All such positions i in T ,
 $0 \leq i \leq |T| - |P|$ that
 $T[i..i + |P| - 1] = P$.

Naive Algorithm

For each position i from 0 to $|T| - |P|$,
check character-by-character whether
 $T[i..i + |P| - 1] = P$ or not.
If yes, append i to the result.

AreEqual(S_1, S_2)

```
if  $|S_1| \neq |S_2|$ :  
    return False  
for  $i$  from 0 to  $|S_1| - 1$ :  
    if  $S_1[i] \neq S_2[i]$ :  
        return False  
return True
```

FindPatternNaive(T, P)

```
result  $\leftarrow$  empty list
for  $i$  from 0 to  $|T| - |P|$ :
    if AreEqual( $T[i..i + |P| - 1], P$ ):
        result.Append( $i$ )
return result
```

Running Time

Lemma

Running time of `FindPatternNaive(T, P)` is $O(|T||P|)$.

Running Time

Lemma

Running time of `FindPatternNaive(T, P)` is $O(|T||P|)$.

Proof

- Each `AreEqual` call is $O(|P|)$

Running Time

Lemma

Running time of `FindPatternNaive(T, P)` is $O(|T||P|)$.



Proof

- Each `AreEqual` call is $O(|P|)$
- $|T| - |P| + 1$ calls of `AreEqual` total to $O((|T| - |P| + 1)|P|) = O(|T||P|)$ \square

Bad Example

If $T = \text{“aaa...aa”}$ and $P = \text{“aaa...ab”}$, and $|T| \gg |P|$, then for each position i in T from 0 to $|T| - |P|$ the call to `AreEqual` has to make all $|P|$ comparisons.

This is because $T[i..i + |P| - 1]$ and P differ only in the last character.

Thus, in this case the naive algorithm runs in time $\Theta(|T||P|)$.

Outline

- 1 Search Pattern in Text
- 2 Rabin-Karp's Algorithm
- 3 Improving Running Time

Rabin-Karp's Algorithm

- Need to compare P with all substrings S of T of length $|P|$

Rabin-Karp's Algorithm

- Need to compare P with all substrings S of T of length $|P|$
- Idea: use hashing to quickly compare P with substrings of T

Rabin-Karp's Algorithm

- If $h(P) \neq h(S)$, then definitely $P \neq S$

Rabin-Karp's Algorithm

- If $h(P) \neq h(S)$, then definitely $P \neq S$
- If $h(P) = h(S)$, call `AreEqual(P, S)`

Rabin-Karp's Algorithm


- If $h(P) \neq h(S)$, then definitely $P \neq S$
- If $h(P) = h(S)$, call `AreEqual(P, S)`
- Use polynomial hash family \mathcal{P}_p with prime p

Rabin-Karp's Algorithm

- If $h(P) \neq h(S)$, then definitely $P \neq S$
- If $h(P) = h(S)$, call `AreEqual(P, S)`
- Use polynomial hash family \mathcal{P}_p with



prime p


- If $P \neq S$, the probability $Pr[h(P) = h(S)]$ is at most $\frac{|P|}{p}$ for  polynomial hashing


RabinKarp(T, P)



```
 $p \leftarrow$  big prime,  $x \leftarrow \text{random}(1, p - 1)$   
result  $\leftarrow$  empty list  
pHash  $\leftarrow$  PolyHash( $P, p, x$ )  
for  $i$  from 0 to  $|T| - |P|$ :  
    tHash  $\leftarrow$  PolyHash( $T[i..i + |P| - 1], p, x$ )  
    if pHash  $\neq$  tHash:  
        continue  
    if AreEqual( $T[i..i + |P| - 1], P$ ):  
        result.Append( $i$ )  
return result
```

False Alarms

 “False alarm” is the event when P is compared with $T[i..i + |P| - 1]$, but $P \neq T[i..i + |P| - 1]$.

 The probability of “false alarm” is at most $\frac{|P|}{p}$

On average, the total number of “false alarms” will be $(|T| - |P| + 1) \frac{|P|}{p}$, which can be made small by selecting $p \gg |T||P|$.



Running Time without AreEqual

- $h(P)$ is computed in $O(|P|)$

Running Time without AreEqual

- $h(P)$ is computed in $O(|P|)$
- $h(T[i..i + |P| - 1])$ is computed in $O(|P|)$, $|T| - |P| + 1$ times

Running Time without AreEqual

- $h(P)$ is computed in $O(|P|)$
- $h(T[i..i + |P| - 1])$ is computed in $O(|P|)$, $|T| - |P| + 1$ times
- $O(|P|) + O((|T| - |P| + 1)|P|) = O(|T||P|)$

AreEqual Running Time

- AreEqual is computed in $O(|P|)$

AreEqual Running Time

- AreEqual is computed in $O(|P|)$
- AreEqual is called only when $h(P) = h(T[i..i + |P| - 1])$, meaning that either an occurrence of P is found or a “false alarm” happened

AreEqual Running Time

- AreEqual is computed in $O(|P|)$
- AreEqual is called only when $h(P) = h(T[i..i + |P| - 1])$, meaning that either an occurrence of P is found or a “false alarm” happened
- By selecting $p \gg |T||P|$ we make the number of “false alarms” negligible

Total Running Time

- If P is found q times in T , then total time spent in `AreEqual` is
$$O\left(\left(q + \frac{(|T| - |P| + 1)|P|}{p}\right)|P|\right) = O(q|P|) \text{ for } p \gg |T||P|$$

Total Running Time

- If P is found q times in T , then total time spent in `AreEqual` is
$$O\left(\left(q + \frac{(|T| - |P| + 1)|P|}{p}\right)|P|\right) = O(q|P|) \text{ for } p \gg |T||P|$$
- Total running time is
$$O(|T||P|) + O(q|P|) = O(|T||P|) \text{ as } q \leq |T|$$

Total Running Time

- If P is found q times in T , then total time spent in `AreEqual` is
$$O\left(\left(q + \frac{(|T| - |P| + 1)|P|}{p}\right)|P|\right) = O(q|P|) \text{ for } p \gg |T||P|$$
- Total running time is
$$O(|T||P|) + O(q|P|) = O(|T||P|) \text{ as } q \leq |T|$$
- Same as naive algorithm, but can be improved!

Outline

- ① Search Pattern in Text
- ② Rabin-Karp's Algorithm
- ③ Improving Running Time

Improving Running Time

$$h(S) = \sum_{i=0}^{|S|-1} S[i]x^i \bmod p$$

Improving Running Time

$$h(S) = \sum_{i=0}^{|S|-1} S[i]x^i \bmod p$$

$$h(T[i..i + |P| - 1]) = \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p$$

Improving Running Time

$$h(S) = \sum_{i=0}^{|S|-1} S[i]x^i \bmod p$$

$$h(T[i..i + |P| - 1]) = \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p$$

Idea: polynomial hashes of two consecutive substrings of T are very similar

Improving Running Time

$$h(S) = \sum_{i=0}^{|S|-1} S[i]x^i \bmod p$$

$$h(T[i..i + |P| - 1]) = \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p$$

Idea: polynomial hashes of two consecutive substrings of T are very similar

For each i denote $h(T[i..i + |P| - 1])$ by $H[i]$

Consecutive substrings

$$\begin{array}{l} T = \quad a \quad b \quad c \quad b \quad d \\ T' = \end{array} \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array}$$



$$|P| = 3$$

Consecutive substrings

$$\begin{array}{l} T = \quad a \quad b \quad c \quad b \quad d \\ T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3 \\ h(\text{"cbd"}) = \end{array}$$

Consecutive substrings

$$\begin{array}{l} T = \quad a \quad b \quad c \quad b \quad d \\ T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3 \\ h(\text{"cbd"}) = 1 \quad x \quad x^2 \end{array}$$

Consecutive substrings

$$\begin{array}{l} T = \quad a \quad b \quad c \quad b \quad d \\ T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3 \\ h(\text{"cbd"}) = 2 \quad x \quad 3x^2 \end{array}$$

Consecutive substrings

$$\begin{array}{l} T = \quad a \quad b \quad c \quad b \quad d \\ T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3 \\ h(\text{"cbd"}) = 2 + x + 3x^2 \end{array}$$

Consecutive substrings

$$\begin{array}{ccccc} T = & a & b & c & b & d \\ T' = & \boxed{0} & \boxed{1} & \boxed{2} & \boxed{1} & \boxed{3} \end{array} \quad |P| = 3$$
$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$$h(\text{"bcb"}) =$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$$h(\text{"bcb"}) = 1 + x + x^2$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$\downarrow \times x \quad \downarrow \times x$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$\downarrow_{xx} \quad \downarrow_{xx}$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

$$H[2] = h(\text{"cbd"}) = 2 + x + 3x^2$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$\downarrow \times x \quad \downarrow \times x$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

$$H[2] = h(\text{"cbd"}) = 2 + x + 3x^2$$

$$H[1] = h(\text{"bcb"}) = 1 + 2x + x^2 =$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$$\begin{array}{cc} \downarrow \times x & \downarrow \times x \end{array}$$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

$$H[2] = h(\text{"cbd"}) = 2 + x + 3x^2$$

$$\begin{aligned} H[1] &= h(\text{"bcb"}) = 1 + 2x + x^2 = \\ &= 1 + x(2 + x) = \end{aligned}$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$$\begin{array}{cc} \downarrow \times x & \downarrow \times x \end{array}$$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

$$H[2] = h(\text{"cbd"}) = 2 + x + 3x^2$$

$$H[1] = h(\text{"bcb"}) = 1 + 2x + x^2 =$$

$$= 1 + x(2 + x) =$$

$$= 1 + x(2 + x + 3x^2) - 3x^3 =$$

Consecutive substrings

$$T = \begin{array}{ccccc} a & b & c & b & d \end{array}$$
$$T' = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 1 & 3 \\ \hline \end{array} \quad |P| = 3$$

$$h(\text{"cbd"}) = 2 + x + 3x^2$$

$\downarrow \times x \quad \downarrow \times x$

$$h(\text{"bcb"}) = 1 + 2x + x^2$$

$$H[2] = h(\text{"cbd"}) = 2 + x + 3x^2$$

$$H[1] = h(\text{"bcb"}) = 1 + 2x + x^2 =$$

$$= 1 + x(2 + x) =$$

$$= 1 + x(2 + x + 3x^2) - 3x^3 =$$

$$= xH[2] + 1 - 3x^3$$



Recurrence of Hashes

$$H[i + 1] = \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} \bmod p$$

Recurrence of Hashes

$$H[i + 1] = \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} \bmod p$$

$$H[i] = \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p =$$

Recurrence of Hashes

$$H[i+1] = \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} \bmod p$$

$$H[i] = \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p =$$

$$= \sum_{j=i+1}^{i+|P|} T[j]x^{j-i} + T[i] - T[i+|P|]x^{|P|} \bmod p =$$

Recurrence of Hashes

$$H[i+1] = \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} \bmod p$$

$$H[i] = \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p =$$

$$= \sum_{j=i+1}^{i+|P|} T[j]x^{j-i} + T[i] - T[i+|P|]x^{|P|} \bmod p =$$

$$= x \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} + (T[i] - T[i+|P|]x^{|P|}) \bmod p$$

Recurrence of Hashes

$$H[i+1] = \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} \bmod p$$

$$\begin{aligned} H[i] &= \sum_{j=i}^{i+|P|-1} T[j]x^{j-i} \bmod p = \\ &= \sum_{j=i+1}^{i+|P|} T[j]x^{j-i} + T[i] - T[i+|P|]x^{|P|} \bmod p = \\ &= x \sum_{j=i+1}^{i+|P|} T[j]x^{j-i-1} + (T[i] - T[i+|P|]x^{|P|}) \bmod p \end{aligned}$$

$$H[i] = xH[i+1] + (T[i] - T[i+|P|]x^{|P|}) \bmod p$$

PrecomputeHashes($T, |P|, p, x$)

```
 $H \leftarrow$  array of length  $|T| - |P| + 1$   
 $S \leftarrow T[|T| - |P| .. |T| - 1]$   
 $H[|T| - |P|] \leftarrow \text{PolyHash}(S, p, x)$   
 $y \leftarrow 1$   
for  $i$  from 1 to  $|P|$ :  
     $y \leftarrow (y \times x) \bmod p$   
for  $i$  from  $|T| - |P| - 1$  down to 0:  
     $H[i] \leftarrow (xH[i + 1] + T[i] - yT[i + |P|]) \bmod p$   
return  $H$ 
```

PrecomputeHashes($T, |P|, p, x$)

```
 $H \leftarrow$  array of length  $|T| - |P| + 1$   
 $S \leftarrow T[|T| - |P| .. |T| - 1]$   
 $H[|T| - |P|] \leftarrow \text{PolyHash}(S, p, x)$   
 $y \leftarrow 1$   
for  $i$  from 1 to  $|P|$ :  
     $y \leftarrow (y \times x) \bmod p$   
for  $i$  from  $|T| - |P| - 1$  down to 0:  
     $H[i] \leftarrow (xH[i + 1] + T[i] - yT[i + |P|]) \bmod p$   
return  $H$ 
```

$O(|P|)$

PrecomputeHashes($T, |P|, p, x$)

```
 $H \leftarrow$  array of length  $|T| - |P| + 1$   
 $S \leftarrow T[|T| - |P| .. |T| - 1]$   
 $H[|T| - |P|] \leftarrow \text{PolyHash}(S, p, x)$   
 $y \leftarrow 1$   
for  $i$  from 1 to  $|P|$ :  
     $y \leftarrow (y \times x) \bmod p$   
for  $i$  from  $|T| - |P| - 1$  down to 0:  
     $H[i] \leftarrow (xH[i + 1] + T[i] - yT[i + |P|]) \bmod p$   
return  $H$ 
```

$$O(|P| + |P|)$$

PrecomputeHashes($T, |P|, p, x$)

```
 $H \leftarrow$  array of length  $|T| - |P| + 1$   
 $S \leftarrow T[|T| - |P| .. |T| - 1]$   
 $H[|T| - |P|] \leftarrow \text{PolyHash}(S, p, x)$   
 $y \leftarrow 1$   
for  $i$  from 1 to  $|P|$ :  
     $y \leftarrow (y \times x) \bmod p$   
for  $i$  from  $|T| - |P| - 1$  down to 0:  
     $H[i] \leftarrow (xH[i + 1] + T[i] - yT[i + |P|]) \bmod p$   
return  $H$ 
```

$$O(|P| + |P| + |T| - |P|) = O(|T| + |P|)$$

Precomputing H

- PolyHash is called once — $O(|P|)$
- First for loop runs in $O(|P|)$
- Second for loop runs in $O(|T| - |P|)$
- Total precomputation time $O(|T| + |P|)$

RabinKarp(T, P)

```
 $p \leftarrow$  big prime,  $x \leftarrow \text{random}(1, p - 1)$   
result  $\leftarrow$  empty list  
pHash  $\leftarrow$  PolyHash( $P, p, x$ )  
 $H \leftarrow$  PrecomputeHashes( $T, |P|, p, x$ )  
for  $i$  from 0 to  $|T| - |P|$ :  
    if pHash  $\neq H[i]$ :  
        continue  
    if AreEqual( $T[i..i + |P| - 1], P$ ):  
        result.Append( $i$ )  
return result
```


Improved Running Time

- $h(P)$ is computed in $O(|P|)$

Improved Running Time

- $h(P)$ is computed in $O(|P|)$
- `PrecomputeHashes` runs in $O(|T| + |P|)$

Improved Running Time

- $h(P)$ is computed in $O(|P|)$
- `PrecomputeHashes` runs in $O(|T| + |P|)$
- Total time spent in `AreEqual` is $O(q|P|)$ on average where q is the number of occurrences of P in T

Improved Running Time

- $h(P)$ is computed in $O(|P|)$
- `PrecomputeHashes` runs in $O(|T| + |P|)$
- Total time spent in `AreEqual` is $O(q|P|)$ on average where q is the number of occurrences of P in T
- Average running time $O(|T| + (q + 1)|P|)$

Improved Running Time

- $h(P)$ is computed in $O(|P|)$
- `PrecomputeHashes` runs in $O(|T| + |P|)$
- Total time spent in `AreEqual` is $O(q|P|)$ on average where q is the number of occurrences of P in T
- Average running time $O(|T| + (q + 1)|P|)$
- Usually q is small, so this is much less than $O(|T||P|)$

Conclusion

- Hash tables are useful for storing Sets and Maps

Conclusion

- Hash tables are useful for storing Sets and Maps
- Possible to search and modify hash tables in $O(1)$ on average!

Conclusion

- Hash tables are useful for storing Sets and Maps
- Possible to search and modify hash tables in $O(1)$ on average!
- Must use good hash families and randomization

Conclusion

- Hash tables are useful for storing Sets and Maps
- Possible to search and modify hash tables in $O(1)$ on average!
- Must use good hash families and randomization
- Hashes are also useful while working with strings and texts

Conclusion

- Hash tables are useful for storing Sets and Maps
- Possible to search and modify hash tables in $O(1)$ on average!
- Must use good hash families and randomization
- Hashes are also useful while working with strings and texts
- There are many more applications in distributed systems and data science