Advanced Computer Contest Preparation

Lecture 32

STRING ALGORITHMS (II)

Contents

- Tries
- Manacher's Algorithm

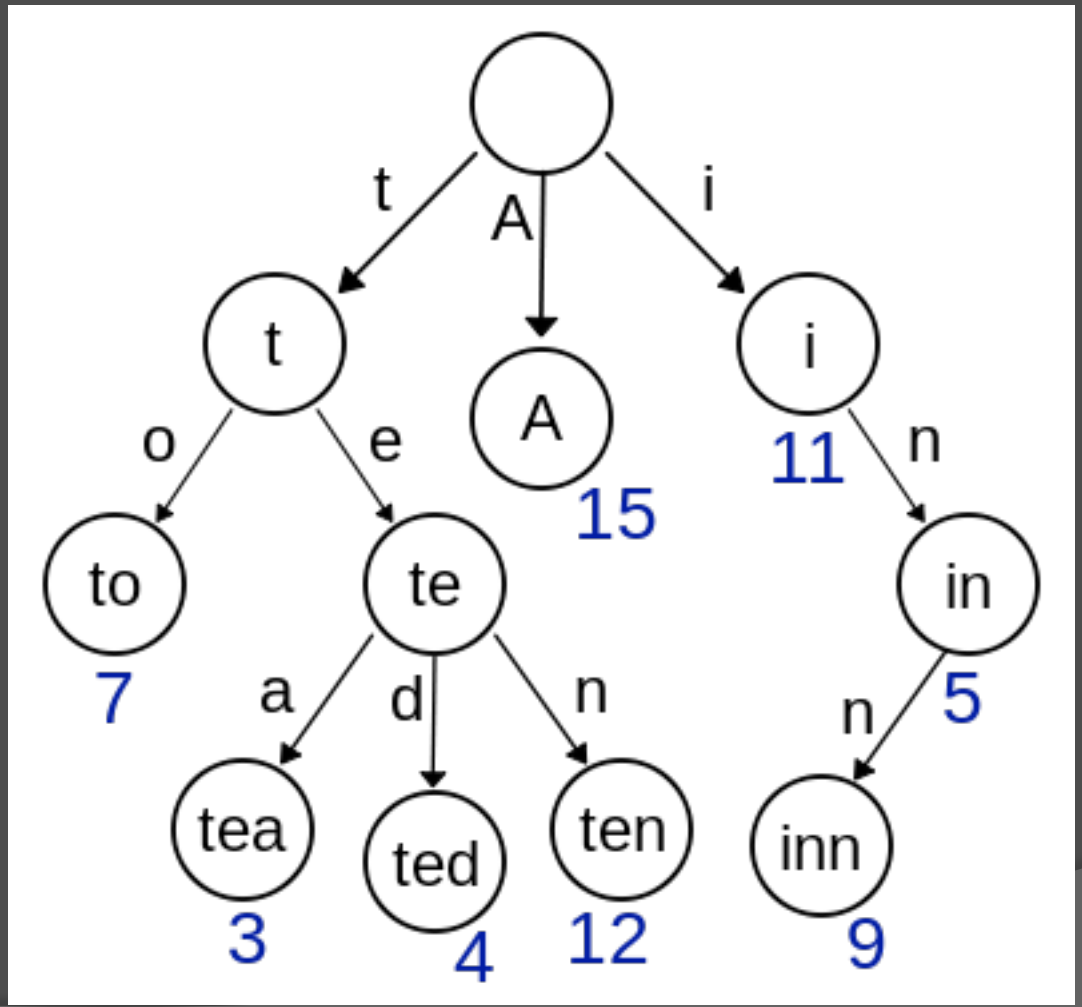
Trie

- Pronounced “try”
- A tree data structure that supports the searching of a set of strings and their prefixes
- Also called a prefix tree

Trie – Construction

- Each node will contain a prefix of one or more of the strings inserted
- Root node is an empty string
- For each edge, a letter is assigned to it
- When we go down an edge, add the character of the edge to the current string

Trie



The integers are arbitrary, they are there to signify that the node's string is in the set

Trie – Additional Features

- A boolean flag signaling if a node's string is a string in the set
- An integer for each node signifying the number of strings in the set with the node's prefix

Trie – Implementation

- Pointer implementation
  - Each node is a **`struct`**
  - For each possible character that an edge can be, have a pointer to the next node
    - Pointer acts as an edge to the next node
    - Typically, use an array of pointers
  - Do not store the string of the node
  - More variables can be added to the **`struct`** if more features are needed

Trie – Implementation

- Array implementation
  - Instead of using random places in memory, use an array of nodes as your “memory”
  - When a node is declared, its “memory address” is the next empty index in the array
  - Instead of using pointers, use integers signifying the indexes in the array
  - Advantages: integers can use less memory than pointers

Trie – Analysis

- Time complexity to insert a string S
  - $O(|S|)$
- Time complexity to search for a string S
  - $O(|S|)$
- Memory complexity
  - $O(num\_unique\_chars \times \sum|S|)$

Manacher's Algorithm

- Given a string S, what is the longest palindrome that is a substring of S?
  - A palindrome is a string that is equal to its reverse (e.g. *"racecar"*)
- Given a string S, what is the number of palindromic substrings of S?
- Manacher's algorithm can solve these problems in $O(|S|)$ time each

Naive Solution

- A simple way of solving the problem is:
  - Choose a character to be the middle of the palindromic string
  - Go outward until the string is no longer a palindrome
  - Time complexity: $O(|S|^2)$
- Issue: Even-length palindromes (e.g. *"abba"*) have no "middle character"
  - Solution: Surround characters with "dummy characters"
  - *"abba"* → *"#a#b#b#a#"*

Manacher Idea

- Manacher's algorithm uses a similar idea
- The left side of a palindrome is a mirror of the right side
- Therefore, from the center of a palindrome, it is enough to keep track of the length of one side
- Manacher's generates an array, M, that finds these values for each character in the modified string, T
- $M(i)$ is the largest integer such that $T_{i-M(i)} \ldots T_i \ldots T_{i+M(i)}$ is a palindrome

Manacher Idea

- Recall: the left side of a palindrome is a mirror of the right side
- Manacher's idea takes advantage of the fact that due to palindromic symmetry, sometimes, $M(i+x) = M(i-x)$
- Suppose the i^{th} character is a center of a palindrome
- For each x, $1 \leq x \leq M(i)$, $M(i+x) = M(i-x)$ if $i-x-M(i-x) > i-M(i)$ (case 1)
  - The largest palindrome centered at $i-x$ is contained entirely within the palindrome centered at i
  - The leftmost character of the $i-x$ palindrome is at least one to the right of the leftmost character of the i palindrome

Manacher – Case 1

Suppose T = "#c#a#b#a#b#a#c#", $i = 7$
$M(0) \ldots M(7)$ is

0	1	2	3	4	5	6	7
#	c	#	a	#	b	#	a
0	1	0	1	0	3	0	7

Suppose $x = 2$, then $i-x = 5$, $i+x = 9$

"#c#a#b#a#b#a#c#"

Using case 1, since $i-x-M(i-x) > i-M(i)$, $5-3 > 7-7$
Therefore, $M(9) = M(5) = 3$

Manacher Idea

- Recall that case 1 is:
  - $M(i+x) = M(i-x)$ if $i-x-M(i-x) > i-M(i)$
- Case 2 is when the condition is not satisfied, i.e. $i-x-M(i-x) \leq i-M(i)$
- In case 2, $M(i+x)$ is at least $M(i-x)$
- To check if $M(i+x)$ can be larger, continue to check outward until characters don't match or end of string is reached

Manacher – Case 2

Suppose T = "#a#b#a#c#a#b#a#c#", $i = 7$

$M(0) \ldots M(7)$ is

0	1	2	3	4	5	6	7
#	a	#	b	#	a	#	c
0	1	0	3	0	1	0	7

Suppose $x = 4$, then $i-x = 3$, $i+x = 11$

"#a#b#a#c#a#b#a#c#"

Using case 2, since $i-x-M(i-x) \leq i-M(i)$, $3-3 \leq 7-7$

Therefore, $M(11)$ is at least $M(3)$

If we try checking outward, we find that $M(11) = 5$

Manacher Idea

- If some $y > i + M(i)$, no information associated with i will be able to determine $M(y)$ (case 3)
- Choose a new palindrome so that we can get case 1 or case 2
- This suggests that we should keep track of the palindrome with the rightmost boundary to avoid case 3 from happening at all

Getting Actual Length

- Note that in the explanation, $M(i)$ does not signify the length of the longest palindrome centered at i
  - $M(i)$ is the length of one side of the palindrome centered at i
- If T_i is not a dummy character
  - For example, T = "#a#b#a#", $i = 3$
  - We can see that the actual length $= M(i)$
- If T_i is a dummy character
  - For example, T = "#a#a#", $i = 2$
  - We can see that the actual length $= M(i)$
- Therefore, $M(i)$ is the actual length as well

Pseudocode

```
string S
string T ← pre-processed S //surround S with dummy characters
int palin ← 0 //palindrome center with furthest right boundary
int idx ← 0
int M[]
while idx < |T| do
        int lft ← palin-(idx-palin)
        if palin-M[palin] < lft-M[lft]
                M[i] ← M[lft]
        else
                int rht ← palin+M[palin]+1
                while rht < |T| and T[rht] = T[idx-(rht-idx)] do
                        rht ← rht+1
                M[idx] ← rht-1-idx
                palin ← idx
        idx ← idx + 1
//answers located in M
```

Analysis

- For each character, only case 1 or case 2 occurs
  - Case 3 should not occur if we always use the rightmost palindrome
- If case 1 occurs, no additional time added
- If case 2 occurs, we always move right from the previous check to continue checking characters
- Therefore, final time complexity is $O(|S|)$

THANK YOU!