Unit VII: Pattern matching and Tries: Pattern matching algorithms— the Boyer–Moore algorithm, the Knuth-Morris-Pratt algorithm, Tries: Definitions and concepts of digital search tree, Binary trie, Patricia, Multi-way trie

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Give a brief description of pattern matching problem and explain the Boyer-Moore algorithm with an example.</td>
<td>What are the advantages and disadvantages of digital search trees?</td>
</tr>
<tr>
<td>2</td>
<td>Describe the KMP pattern matching algorithm with relevant example.</td>
<td>Compute the prefix function PI for the pattern (P=ababaca) against a text (T)</td>
</tr>
<tr>
<td>3</td>
<td>What are the advantages and disadvantages of tries with respect to binary search tree</td>
<td>What are the applications of tries</td>
</tr>
<tr>
<td>4</td>
<td>With an example, describe Binary trie.</td>
<td>Write the analysis of time complexity of Boyer-Moore pattern matching algorithm</td>
</tr>
</tbody>
</table>

**Semester Regular and Supplementary Examinations, December - 2013**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is multi-way trie? What for it is used? Explain with suitable example</td>
<td>Discuss the KMP pattern matching algorithm with example.</td>
</tr>
<tr>
<td>2</td>
<td>What is binary tree? What for it is used? Explain with suitable example</td>
<td>Discuss the boyer-moore algorithm with example.</td>
</tr>
<tr>
<td>3</td>
<td>Draw a simple, connected, weighted graph with 8 vertices and 16 edges. Each with unique edge weight. Identify one vertex as a “start” vertex and illustrate a running of Dijkstra’s algorithm on this graph.</td>
<td>With examples, briefly describe graph storage representations.</td>
</tr>
<tr>
<td>4</td>
<td>What for the prefix function is used in KMP algorithm? Compute the prefix function for the pattern (abababababababb) when the alphabet is (a, b).</td>
<td>Explain Prim’s algorithm and illustrate with suitable example.</td>
</tr>
</tbody>
</table>

**Semester Supplementary Examinations, May 2013**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Explain about multi-way trie?</td>
<td>Explain about Knuth-Morris-Pratt algorithm with one example.</td>
</tr>
<tr>
<td>2</td>
<td>Write a program for pattern matching using Boyer Moore algorithm</td>
<td>What is mean by Binary tree</td>
</tr>
<tr>
<td>3</td>
<td>Explain about different pattern matching algorithms.</td>
<td>Explain about the concept of digital search tree.</td>
</tr>
</tbody>
</table>

**Semester Regular Examinations, November/December - 2012**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Explain about multi-way trie?</td>
<td>Explain about Knuth-Morris-Pratt algorithm with one example.</td>
</tr>
<tr>
<td>2</td>
<td>What is meant by trie?</td>
<td>Explain about Patricia</td>
</tr>
<tr>
<td>3</td>
<td>Explain about Boyer Moore algorithm with an example</td>
<td>What is meant by trie?</td>
</tr>
<tr>
<td>4</td>
<td>Explain about Patricia</td>
<td>Explain about Patricia</td>
</tr>
</tbody>
</table>

bphanikrishna.wordpress.com
UNIT VII
PATTERN MATCHING & TRIES

Pattern matching:

Pattern matching is the act of checking a given sequence of tokens (alphabets or symbols etc.) for the presence of constituents of some patterns.

Means, pattern matching is to find a pattern which is relatively small, in a text which is to be very large.

Pattern & text can be one-dimensional or two-dimensional.

1-dimensional example: text editor & DNA

Text editor: have a 26 characters & some special symbols.

DNA: DNA has 4 characters.

2-dimensional eg.: computer vision.

Either 1-dimensional or 2-dimensional the text is very large & therefore, a fast algorithm to find the occurrence of pattern in it needed.

In the classic string pattern matching problem, we are given text string T of length n and pattern string P of length m.
To find whether \( P \) is a substring of \( T \): The notation of a match is that there is substring of \( T \) starting at some index \( i \). That matches, \( P \) character by character so that such as,
\[
P[0] = T[i], P[1] = T[i+1], P[2] = T[i+2], \ldots, P[m-1] = T[i+n-1].
\]

- The output of a pattern matching alg could either be some indication that the pattern \( P \) does not exist in \( T \) as an integer indicating the starting index in \( T \) of a substring matching \( P \).

Applications of pattern matching:

* **Text editors:** In text editors we have no. of lines of text data for finding required string from the editor we use string pattern match.

* **Search engine:** The query submitted by the user in search engine uses the pattern matching.

* **Biological search:** E.g: DNA

* **Pattern matching** is used to find the common personalities of different research things.
Pattern matching algorithm:

The popular pattern matching algorithms are:

- Naive pattern matching algorithm.
- Brute force algorithm.
- Boyer-Moore algorithm.
- Knuth-Morris-Pratt algorithm [KMP].

Boyer-Moore algorithm:

Introduction: It is a pattern matching algorithm. It was developed by Robert S. Boyer and J. S. Moore in 1977.

Definition: The boyer-moore algorithm is an efficient string search algorithm. This algorithm preprocesses the string being searched for the pattern but not the string being searched in the text.

It is well suited for applications in which pattern is much shorter than the text, does persist (carry on) across multiple searches.

Purpose of Boyer-Moore algorithm:

It is usually used in text editors & commands substitution.

Text editor: - In text editors for search & substitute commands implementation we use this alg.
Working Procedure:

This alg. compares the pattern P with the sub string of sequence T within a sliding window in the right to left order. Means it can scan the characters of patterns from right to left beginning with the right most one.

This alg. uses two heuristics (set of rules).

1. Bad character rule (BCR).
2. Good suffix rule

BCR:

Suppose the P, is aligned to Ts. Now we perform a pair wise comparing b/w text ‘T’ & pattern ‘P’ from right to left.

Assume that the first mis-match occurs when comparing Ts+j−1 with Pj.

Since, Ts+j−1 ≠ Pj we moves the pattern P to the right. Such that the largest position ‘C’ in the left of Pj is equal to Ts+j−1.

We shift the pattern at least j−C positions.
BCR uses rule 2-1, [character matching rule].
* For any character \( x \) in \( T \), find the nearest \( x \) in \( P \) which is to be left as \( x \) in \( T \).

Move \( P \) so that the two \( x \)'s are match.

If no such a \( x \) in \( P \), consider the partial window defined by \( x \). In \( T \), shift to the left of it.

2) Good Subfix rule: — Good Subfix rule used when BCR failed.
The good suffix rule proposed to move the pattern to the right by the least amount so that a group of characters in the pattern will mismatch with the good suffix found in the text.

```
S
T
P
```

Example:
```
0 1 2 3 4 5 6 7 8 9 10
T → a b b b a d a b a c b a
P → b a b a c
```

Here 'c' does not match with 'd' then compare the left part of c with left part with d if 'd' don't match with left part of c then shift the P to right part of c.

```
0 1 2 3 4 5 6 7 8 9 10
T → a b b b a b a b a c b a
P → b a b a c
```

Mismatch.
Pattern matching and Tries

Time analysis for boyer-moore algorithm:

A string matching algorithm pre-process a pattern $P [|P| = n]$ size of pattern.

For a text $T [|T| = m]$ no of characters in text.
Find all the occurrence of $P$ in $T$.

Time complexity is $O(n+m)$.

Right to left pattern matching:

Worst case complexity is $O(n \cdot m)$

Best case complexity is $O(n/m)$

(Time complexity is calculated by the no. of lines executed).

Knuth-Morris-Pratt alg. (KMP):

KMP is a pattern matching algorithm. It was conceived by Donald Knuth, Vaughan Pratt in 1974, and independently J.H. Morris.
The three published jointly in 1977.

Definition:

KMP alg., searches for occurrence a word $w$ within a main text $s'$ by employing the observation that when a mismatch occurs the word itself embodies sufficient information to determine where
the next match could begin, that by passing re-examination of previously matched characters.

Working process:

It is a tight analysis of naive algorithm. KMP algorithm keeps the information that naive approach wasted gathering during the scan of the text. By avoiding the waste of information it achieves a running time of $O(m+n)$.

Eg. \[ W = \text{ABCDABD} \]
\[ S = \text{ABC' ABDAB, ABCDABCDABDE}. \]

At given time, the algorithm is in a stage determined by two integers:

1st: $m \rightarrow$ which denotes the position in text which is the beginning of a perspective (future/potential) match for $W$.

2nd: $i \rightarrow$ The index in $W$ denoting the character currently under consideration.

In each step we compare $S[i+m]$ with $W[i]$ advance if they are equal.

Eg. $M: 01234567890123456789012$
\[ W = \text{ABC' ABDAB, ABCDABCDABDE}. \]
\textbf{W: ABCDABD} \\
\textbf{i: 0123456} \\
\textbf{S[3] = Space\} mismatch.} \\
\textbf{w[2] = D} \\

we start at \textit{S[0]} but it fail. Now we start from \textit{S[1]}. But we note that no 'A' occurs below positions 0 & 3 expect of having checked all those characters previously we know that there is no chance of finding the beginning of a match. \\

We move onto the next character. \textbf{M=4, i=0} \\

\textbf{II) M: 01234567890123456789012} \\
\textbf{S: ABC ABCDAB ABCDABD ABDE} \\
\textbf{w: ABCDABD} \\
\textbf{i: 0123456.} \\
\textbf{S[10] = Space\} mismatch.} \\
\textbf{w[6] = D} \\

\textbf{III) M: 01234567890123456789012} \\
\textbf{S: ABC ABCDAB ABCDABD ABCDABDE} \\
\textbf{w: ABCDABD} \\
\textbf{i: 0123456.} \\
\textbf{S[10] = Space\} mismatch.} \\
\textbf{w[2] = DC} \\

\textbf{IV) M: 01234567890123456789012} \\
\textbf{S: ABC ABCDAB ABCDABD ABDE} \\
\textbf{w: ABCDABD} \\
\textbf{i: 0123456.}
This time we are able to complete the match.

Choose 1st character is $S[15]$.

Algorithm:


// I/p: - $S$ -> array of characters (The text. to be searched)

$W$ -> array of characters. (The text to be sought (looking for))

// O/p: - an integer.

The @Zero placed position in $S$

at which $W$ is found.

Integer: $M \leftarrow 0$ // The beginning of the current match in $S$.

Integer: $i \leftarrow 0$ // The position of the current

character in $W$.

Array Integer : $T$ // the table compute else where.

while ($m + i < \text{length}(S)$) do

if $W[i] = S[m + i]$ then

if $i = \text{length}(W) - 1$ then
\texttt{return m} \\
\texttt{i \leftarrow i + 1} \\
\text{else} \\
\texttt{m \leftarrow m + i - T[i]} \\
\text{if } T[i] > -1 \text{ then} \\
\texttt{i \leftarrow T[i]} \\
\text{else} \\
\texttt{i \leftarrow 0} \\
\text{end while}

If above alg. return m value \ldots \omega found in s at position m. If above alg. did not return m we say we have searched all of s' unsuccessful.

KMP - Time analysis:

For finding p of size n (|P| = n).
In text T of size \(m\) (|T| = m).
By using naive alg., time complexity \(O(n \cdot m)\).
The KMP makes use of information gained by previous symbol comparisons.

It never compares a text symbol that has matched a pattern symbol. Time complexity for text is \(O(m)\) & pattern is \(O(n)\).

Overall time complexity is \(O(m + n)\).
Trie:

Trie is an efficient information retrieval ordered tree data structure.

- It is also called multi-way tree ds.

[Multiway tree: It is a tree ds. In this if the root node contains 'k' elements then the subtrees are (k+1)]

- The name trie comes from its use of retrieval.
- We use trie to store piece of data that have a key 'k' (used to identify a data) & possibly a value.

In trie we use data whose keys are strings.

Ex:- Name & age for set of people:

<table>
<thead>
<tr>
<th>Name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>amy</td>
<td>56</td>
</tr>
<tr>
<td>ann</td>
<td>15</td>
</tr>
<tr>
<td>emma</td>
<td>30</td>
</tr>
<tr>
<td>bob</td>
<td>27</td>
</tr>
<tr>
<td>doge</td>
<td>52</td>
</tr>
</tbody>
</table>
Ex: A trie for keys A, to, tea, ted, i

- in
- inn

Another representation:

Ex: "allot", alone, ant, and, ace, bat, bad.
Unlike a binary search tree, no node in the tree stores the key associated with that node. Instead, its position in the tree defines the key with which it is associated.

- All the descendants of a node have a common prefix of the string associated with that node.
- Root is associated with empty string.
- Values are not associated with every node.
- Only with leaf nodes & some inner nodes that corresponds to keys at insert.

**Advantages**:

* The pattern matching can be done efficiently.
* In tries, the keys are searched using common prefix.
  
  It takes $O(K)$ lookup time, where $k$ is the size of a key.
* Lookup can takes less than ($\leq K$) times if it is not there.
* Comparing with hash table:
  
  - Look up can be faster in time in worst case time complexity as to compared with hash table.
  - There is no collision in toie.
  - There is no hash function in toie.
* Comparing with BST:
Disadvantages:

* Some times data retrieval of tree is very much slower than hash table.
* Representation of keys a string is complex.
  Eg: Representation of floating point numbers using string is really complicated in trees.
* It always takes more space.
* It is not available in programming tool.

Applications:

- Tries has an ability to insert, delete, or search for the entries. Hence, they are used in building dictionaries. Such as, English words, Telephone numbers.
- These are also used in spelling check software.
- These are well suited for approximate matching algorithm.

Digital Search: tree (DST):

- A DST is a binary tree in which each node contains one element.
- The element to node assignment is
determined by the binary representation of element key.

- DST represents one possible data structure which allow us to store, search, & delete data using the key.
- DST is similar to BST but, the main difference is instead of comparing key values, they make use of the digital representation of the keys.
- If the key can be represented as binary number, it makes sense to refer the $b$th bit of a key, where the bits are numbered from left to right.
- To insert a record $(k, data)$ with key $k$ into a DST, we said ‘$x$’ to point to root and “$b$” to $1$.

Steps for DST:

*Step 1:* If $x$ is null, then store $(k, data)$ in new node and terminate.

*Step 2:* If $key(x) = k$, then terminate ($k$ is already stored in the tree).

*Step 3:* Otherwise,

  - If $b$th bit of $k$ is $0$ then set $x$ to left($x$).
  - If $b$th bit of $k$ is $1$ then set $x$ to right($x$).
Step 4: Set b to (b+1)

Step 5: Go to step 1

Example:
Create a DST by using keys:
1000, 0010, 1001, 0001, 1100, 0000

Ex. 2)

insert 0010
insert 0001
insert 1001
insert 1011
insert c000

EG-3) create DST,
A-000001, S-10011, E-00101, R-10010
C-00011, H-01000, I-01001, N-01110
G-00111, X-11000, M-01101, P-10000

Advantages of DST:
In DST we compare keys bitwise by this we avoid full length of key which is not suitable for searched element key. By this we eliminate time for searching full length of key.

Drawbacks of DST:
* When we dealing with very large key, the cost of a key comparison is very high.
To solving this problem, we have to reduce no.of comparisons.

For reducing no.of comparisons we using another data structure called "PATRICIA".

This PATRICIA structure is developed in three steps:

**Step-1:** First we introduce a structure called binary trie.

**Step-2:** Then we transform binary trie into compressed binary tries.

**Step-3:** Finally from compressed binary tries we obtain PATRICIA.

**Binary Trie:**

A binary trie is a binary tree that has two kinds of nodes:

1. Branch Node
2. Element Node

**Branch Node:** Branch node has two data members: *Left child* and *Right child*.

**Element Node:** Element node has single data member called *Data*.
Branch nodes are used to build a binary tree search structure. Similarly to the BST, this leads to element nodes.

Example binary trie:

```
+--------+
|        |
|        |
| 0010   |
|        |
| 0000   |
|        |
|        |
| 0001   |
|        |
|        |
| 1100   |
|        |
| 1000   |
|        |
| 1001   |
```

Compressed binary trie:

* The binary trie's contains branch whose degree is one.
* By adding another data member bit-number, to each branch node, we can eliminate all degree one branch nodes from the trie.
* The bit-member data members of a branch nodes gives the bit-number of the key, i.e., to be used add at this node.
Introduction:

It is related to trie & developed by D.R. Morrison in 1968.

Purpose:

- This is for solving the comparison problem in DST.
- It is also called radix tree.

Definition:

Patricia is a space optimized trie ds where each node with only one child is merged with its parent. This obtained from compressed binary trie in the following way.
Step 1: Replace each branch node by a argumented branch node.

Step 2: Eliminate the element nodes.

Step 3: Store the data previously in the element node, data members of argumented branch node since every non-empty compressed binary tries has one less branch node then it has element nodes. It is necessary to add 1 'argument branch node. This node is called head node.

Step 4: Replace the original pointers to the element node by pointers to the respective argumented branch nodes

Multiway Tries:

Multiway tries is a one of the trie ds for fast data retrieving. It is for overcome the drawbacks of binary trie.
* Binary trie uses radix search with radial search.

* Coming to multiway tries, we uses radix search with \( R > 2 \).

* Multiway tries sometimes called as \( R \)-ary tries.

* In each digit in a key has \( r \)-bits, then the radix is \( R = 2^r \).

* If the key have almost \( B \)-bits, the cost case time for the no. of comparisons would be \( B/8 \).

Ex-1) Keys are words made up of lower case letters in English. There are 26 different lower case letters in English. So, \( R \)-ary tries with \( R = 2^{26} \) could hold these keys. This type of tries sometimes referred as alphabet tries.

Properties:

* The structure of multiway tries depends only on keys in it, not on the order in which they are inserted.

* Multiway tries have strong key ordering property.
  - At a node \( x \), all keys in \( x \) left most sub tree are smaller than keys in \( x \).
  - At a node \( x \), all keys in \( x \) right most
Subtrees are larger than keys in X.

- So, tree order traversal of a multiway trie visits keys in sorted order.

* The worst case time for the no. of comparisons would be $B/8$.

- Worst case height in BST contains $N$.
- Worst case height in DST contains $\log N$.

Drawbacks:

There is a space cost disadvantage in multiway trie. So, to solve
To solve this problem we go for another data structure ternary trie.