Design and Analysis of Algorithms

Lecture 07 – Strings and String Algorithms

Strings

Every letter, digit, and punctuation mark is a character.

There are also many characters that are invisible on screen, such as the space, tab, and carriage-return characters.

Most programming languages provide a data type called 'character' or 'char'.

In a particular character set, each value represents a single character from a predefined set, such as ASCII or Unicode. Each character has its own binary pattern.

Strings

Most programming languages have a data type called a string, which is used for data values that are made up of ordered sequences of characters, such as "hello world".

When we discuss strings, string operations, and string algorithms we are discussing storage of and algorithms related to these character sequences.

These should not be related to any particular character set. Regarding string algorithms, we should simply assume that in a given character set, (1) there are finite characters, and (2) each character has only one representation (ie. binary data value) so that no two characters are equal.

However, case sensitivity, and dictionary ordering of characters (Example. is the Turkish ‘ı’ before or after the universal ‘I’) may become problems in their own due to implementation choices regarding the character set.

Strings

Considering implementations of string data type in common programming languages.

A string can contain any sequence of characters, visible or invisible, and characters may be repeated.

The number of characters in the string is called its length. Invisible characters such as leading or trailing spaces are also counted.

Example: "hello world" has length 11, 10 letters and 1 space, and " hello world " has length 13, 10 letters and 3 spaces.

There is usually a practical restriction on the maximum length of a string, but you may never observe that restriction using the typical contents of strings.

For example in Java, the size of a String is held in a 32-bit integer. That means the maximum (positive) value possible for the length is 2^{31}-1, which is 2147483647 (2 billion) characters.

If you try to process large contents such as contents of a large file as a single string, you might hit that restriction. Note that, a 2 billion character file would take up 4 gigabytes of space.

Trying to allocate 4GB of consecutive space in memory is in itself not easy unless your computer has much more than that available.

There is also such a thing as an empty string, which contains no characters - length 0.

Strings

Considering implementations of string data type in common programming languages.

A string can be a constant or variable. If it is a constant, it is usually written as a sequence of characters enclosed in single or double quotation marks, ie 'hello' or "hello".

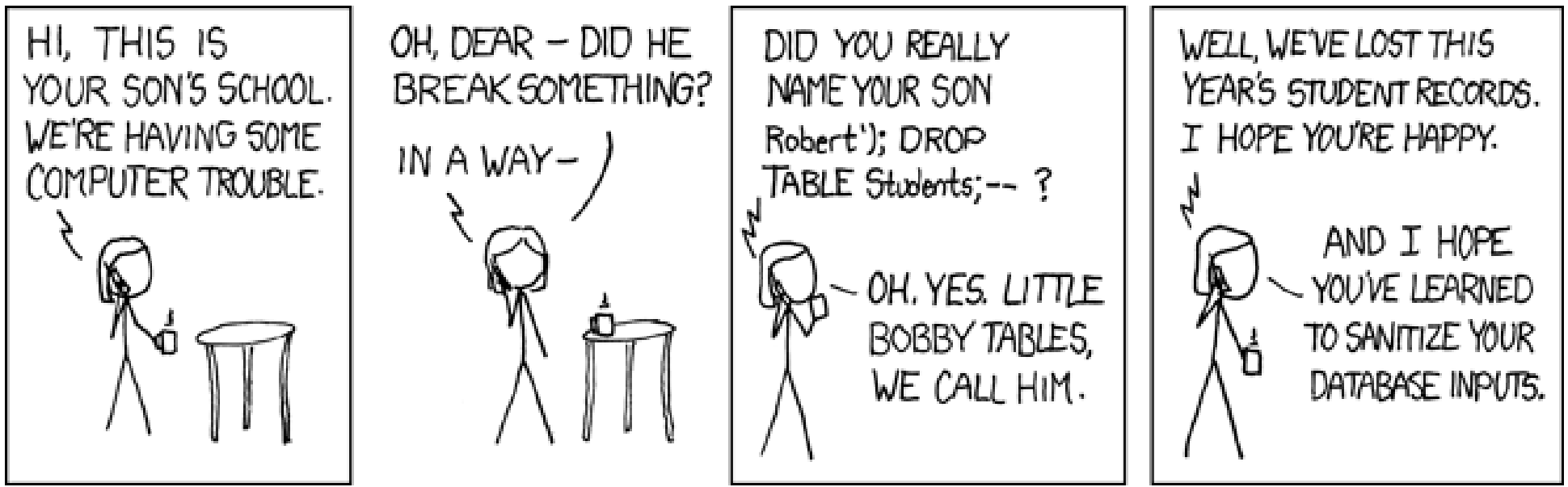
A **string literal** or anonymous string is a type of literal for the representation of a string value in the source code of a computer program.

- When your code includes hard-coded strings such as print(“Hello”), those strings are collected and stored in a particular area in your program’s executable.
- Many times string literals repeat themselves, or share parts.

Variable interpolation is the process of evaluating an expression containing one or more variables, and returning output where the variables are replaced with their corresponding values in memory.

- This becomes important in many programming languages because the input/output routines contain string literals with encoded parts (starting with %, $, ?, etc) which will be replaced with user input, or content produced from user input.
- SQL injection is a popular type of attack which injects harmful SQL code into the variable interpolated string literals. This attack uses the input fields in computer programs, with which the program is supposed to generate a valid SQL commands.
- **Input sanitization** is the process in which user inputs are pre-processed so that no user input which contains malicious string content.

Strings


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Strings

How would you sanitize little “Bobby Tables”?

Check that a “name” does not include characters other than characters in the alphabet. Would any name normally contain “;” which is a special character in SQL?

If you are interested please visit. – https://en.wikipedia.org/wiki/SQL_injection

Strings

When we consider String algorithms we can broadly classify them into two categories.

Algorithms that work with a single string, processing that string. An example would be to find a particular sub-string within that string.

Algorithms that work with multiple strings. An example would be sorting strings.

These algorithms are really important because strings make up of most data processed by computers. When you have efficient string processing, you have efficient computing.

The decision on choosing memory consumption over computation or vice-versa is very visible in string processing.

Languages and their design decisions on string implementations are also very important. You should know how your language implements some fundamental operations so that you can analyze their complexity.

An example – http://java-performance.info/changes-to-string-java-1-7-0_06/ - a design decision on Java string was reversed very quickly.

Strings

Regarding Java strings

- A String is a sequence of characters. Characters are of type char and take up 2 bytes.
- All strings are represented by objects.
  - Low-level implementations based on arrays of characters in other languages allow a bit faster processing at the risk of allowing more programmer's error.
- String objects are immutable.
  - This means once they are set, their content does not change.
  - This is useful so that we can use them in assignment statements and as arguments and return values from methods without having to worry about their values changing.
  - If you want to change the value of a string variable, you re-assign the reference to another string object.
- There are some core embedded features.
  - The charAt() method extracts a specified character from a string in **constant time**.
  - The length() method returns the length of a string in **constant time**.
  - The substring() method typically extracts a specified sub-string in **linear time**.

Strings

Regarding Java strings

- There are some core embedded features (cnt'd).
  - The + operator performs string concatenation.
    - However, we avoid forming a string by appending one character at a time because that is O(n^2) in Java.
    - If you need to concatenate multiple strings to produce another string, there is a utility class called StringBuilder to assist. – https://docs.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html Please note that, StringBuilder class also sacrifices thread safety for performance improvement.
- Please review – https://docs.oracle.com/javase/7/docs/api/java/lang/String.html – and see what the class methods offer out of box.

Strings

Before going into more complex string algorithms, one can decide for some exercises on strings.

- These exercises are easier to discuss on arrays of characters.
- Adding leading “0”s to a string that represents a number.
- Removing a particular character from a string.
  - Special case: Removing whitespace.
  - Special case: Remove leading “0”s from a string representing a number.
- Locate a sub-string.
  - Derivation: Count the sub-string.
  - Derivation: Remove the sub-string.
  - Derivation: Remove duplicates of a sub-string (ie. keep the first one).

Strings

Before going into more complex string algorithms, one can decide for some exercises on strings. (cnt'd)

- Tokenize a string.
- Complement a string.
- Convert from hexadecimal to decimal.
- Use a binary tree to represent a string.
  - Variation: Nodes contain individual characters.
  - Variation: Nodes contain sub-strings tokenized by a particular separator character.

String Sorting

Sorting an array of strings is a classic problem.

In Java, because String class implements Comparable, and we have an array based implementation of List interface, ArrayList, this is actually **too easy** to do.

```
String[] array = {"a", "b", "c", "d", "e"};
List<String> list = Arrays.asList(array);
// populate through list.add()
// Ascending order
Collections.sort(list);
// Descending order
Collections.sort(list, Collections.reverseOrder());
```

String Sorting

Sorting an array of strings is a classic problem.

If we would like to do it by processing arrays of characters, then Radix Sort is the most popular algorithm for this type of operation.

Radix sort is a sorting algorithm that **sorts numbers based on the positions of their digits**. It does not utilize a comparison of the whole number. It is a **stable sorting algorithm**.

Note that we assume all strings are made up of characters, and all characters are unique.

Then we can assume a number system, where we have 2^8 (for ASCII) or 2^{16} (for UNICODE) digits instead of the usual decimal system with 10 digits.

Strings of same size are much easier to compare this way.

Radix Sort takes O(d*(n+b)) time where

b is the base for representing numbers, ie for the decimal system, b is 10.

If k is the maximum possible value, then d would be $O(\log_b(k))$.

If b is large, as in strings ($b=2^{16}=65.536$) then radix sort is comparable to O(n logn)

String Sorting

Sorting an array of strings is a classic problem.

- How does radix sort work on numbers?
- There are two variations (based on which end of the sequence you start).
  - Least Significant Digit (LSD) Radix Sort starts on the rightmost digit (character)
  - Most Significant Digit (MSD) Radix Sort starts on the leftmost digit (character)
- There is an example on –
  https://github.com/eugenp/tutorials/tree/master/algorithms-sorting
- Another example (very creatively using a queue!) –
  https://eddmann.com/posts/least-significant-digit-lsd-radix-sort-in-java/

String Sorting

Why is a sorted string array useful?

Implementing search on such an array will be efficient.

Can we not use a tree?

Of course we can.

- Take the elements input in an array.
- Create a Binary search tree by inserting data items from the array into the binary search tree.
- Perform in-order traversal on the tree to get the elements in sorted order.

In worst case we get o(n^2) and on average case we get o(n logn).

However, many BST implementations are designed to be used as sets. Those would not repeat the same string multiple times.

Tries

Getting fast and space-efficient string searching done is not easy.

If we store keys in binary search tree (BST), a well balanced BST will need time proportional to M * log N, where M is maximum string length and N is number of keys in tree.

Can we do better?

With tries we can.

A Trie is an information re**Tri**eval data structure.

Tries can provide linear time search on keys. For a set of strings, with at most M characters, a trie would take O(M) time.

Caveat – **Huge** memory use.

So how do we implement tries?

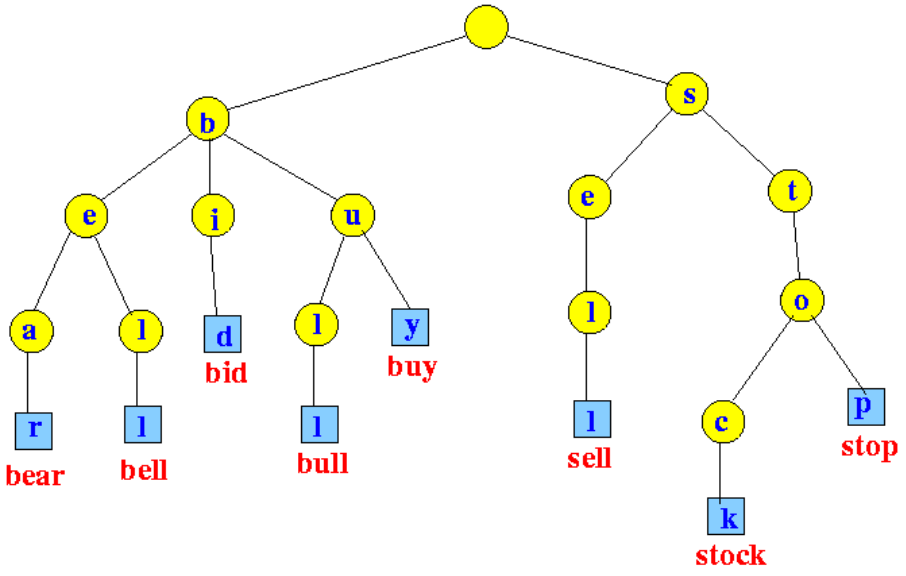
Tries

In a trie

A node's position in the tree defines the key with which that node is associated, which makes tries different in comparison to binary search trees, in which a node stores a key that corresponds only to that node.

All descendants of a node have a common prefix of a String associated with that node, whereas the root is associated with an empty String.

For this reason a trie is also known as a **prefix tree**.



Tries

To create a trie,

You should have **a fixed alphabet** and therefore know **the size of that alphabet**.

Each TrieNode is an s-ary tree node where s is the alphabet size.

There is also the concept of being “end of word” which corresponds to **a sequence ending there**. It does not have to mean being a tree-node.

Consider “bear” and “bearish”.

```
public class TrieNode {
    public static final int ALPHABET_SIZE = 26;
    TrieNode[] children =
                        new TrieNode[ALPHABET_SIZE];
    boolean endOfWord;
    TrieNode(){
        endOfWord = false;
        for(int i = 0; i < ALPHABET_SIZE; i++){
            children[i] = null;
        }
    }
}
```

Tries

To create a trie

Adding a string to Trie requires parsing through all character.

Let's practice inserting

"big", "bear", "bull", "bearish", "bullish", "babka", "babel", "baboon", "babooneries", "bachelor", "backhand"

and then

"wag".

Does the tree re-balance itself when we add "wag"? Why or why not?

```
class Trie {
    TrieNode root;
    //...
   /** Inserts a word into the trie. */
   public void insert(String word) {
      int index;
      TrieNode crawler = root;
      for(int i = 0; i < word.length(); i++){
         index = word.charAt(i) - 'a';
         if(crawler.children[index] == null){
            crawler.children[index] = new TrieNode();
         } // if
         crawler = crawler.children[index];
      } // for
      crawler.endOfWord = true;
   } // insert()
}
```

Tries

To search within a trie

We start iterating through TrieNodes and if we end up with the word and at that exact node there is a marking of “end of node” then the word is in the trie.

For example, in the previous slide

We added “backhand” to the trie, but not “back”.

The path b-a-c-k-h-a-n-d exists, but only on the ending “d” we have the end of word set to true.

```
class Trie {
    TrieNode root;
    //...
     /** Returns if the word is in the trie. */
     public boolean search(String word) {
       int index;
       TrieNode crawler = root;

       for(int i = 0; i < word.length(); i++){
          index = word.charAt(i) - 'a';
          if(crawler.children[index] == null){
             return false;
          } // if
          crawler = crawler.children[index];
       } //for
       return (crawler != null && crawler.endOfWord);
      } // search()
}
```

Tries

To search within a trie

We start iterating through TrieNodes and if we end up with the word and at that exact node there is a marking of “end of node” then the word is in the trie.

For example, in the previous slide

We added “backhand” to the trie, but not “back”.

The path b-a-c-k-h-a-n-d exists, but only on the ending “d” we have the end of word set to true.

```
class Trie {
    TrieNode root;
    //...
     /** Returns if the word is in the trie. */
     public boolean search(String word) {
       int index;
       TrieNode crawler = root;

       for(int i = 0; i < word.length(); i++){
          index = word.charAt(i) - 'a';
          if(crawler.children[index] == null){
             return false;
          } // if
          crawler = crawler.children[index];
       } //for
       return (crawler != null && crawler.endOfWord);
      } // search()
}
```

Tries

When to use a trie?

As you see, a Trie is a data structure optimized for a specific type of search operation.

You use a Trie when you want to take a partial value and return a set of possible complete values which start with the partial value.

- The classic example for this is the autocomplete function. Another example is a spell checker.
- So many advanced text editors load dictionaries of particular languages as a trie.
- Because a try takes up huge memory, such editors prefer to work with **one dictionary at a time**.
- An interesting case is the case of online text editors, where the autocomplete and spell checker run on servers. In this case, because the server can support huge memory, they can load all dictionaries at the same time.

Tries

When to use a trie?

Another use is to design filters based on structured text.

IPv4 addresses. For example 193.140.83.251 is the IP address of ULAKBİM's DNS sever. It accepts connections from a large number of computers, particularly those from Turkish universities.

Suppose we'd like to create an IP-address filter so that only IP addresses with certain prefixes can connect to this server. However, the filter should work very efficiently because even 1 mili second is important for network latency.

The prefixes for filters can be held in a Trie. Time to check the prefix is constant, for example O(6) for a 6-digit prefix.

Question

Could you do the same for a **phonebook**, for example a blacklisted numbers feature?

That is your next homework assignment. Implement a phone number blacklist using a trie.