

## Data Structures and Algorithms II

### Lecture 2: Introducing String Algorithms and Data Structures

- String ADT?
- Introduction to the String Search.

1

### A String ADT

- Most languages either have a built in string datatype or a standard library, so rare to create own string ADT.
- A string datatype should have operations to:
  - Return the nth character in a string.
  - Set the nth character in a string to c.
  - Find the length of a string.
  - Concatenate two strings. “Alison” + “Cawsey” = “Alison Cawsey”
  - Copy a string.
  - Delete part of a string. (“Alison Cawsey” → “Alison”)
  - Modify and compare strings in other ways.

3

### String Processing Algorithms

- String processing algorithms are algorithms for processing sequences of characters or symbols e.g.,
  - File compression - take a sequence, encode it as a shorter sequence.
  - Cryptography - take a sequence, encode it so enemies can't read it!
  - String search - search for occurrences of one sequence within another.
  - Pattern matching - find out if sequence matches some pattern.
  - Parsing - Work out structure of a sequence, in terms of a grammar.
- Applications of more complex algorithms might include genome sequencing and analysis.
- We can start by looking at the relevant datatypes or classes for strings and sequences.

2

### String Implementations

There are two main ways that strings may be implemented:

- As a fixed length array, where the first element denotes the length of the string, e.g., [6,a,l,i,s,o,n,.....]. This is used as the standard string type in Pascal.
- As an array, but with the end of the string indicated using a special 'null' character (denoted '\0'), e.g., [a,l,i,s,o,n,\0,.....]. Memory can be dynamically allocated for the string once we know its length.

First implementation has disadvantages of all fixed length array implementations. But some operations are efficient (e.g., finding length).

Second implementation has advantages of dynamic allocation of space; modifying string also may be more efficient, as needn't recalculate size.

4

## Strings in C++

The `string.h` library provides many string manipulation functions. (Many listed in Friedman & Koffman) such as:

```
char *strcpy(trg, src)
char *strcat(trg, src)
int strcmp(trg, src)
char *strchr(trg, c) // returns pointer to
                    // first occ of c in trg
size_t strlen(src)
```

These all assume you've declared your strings to be pointers to characters (or arrays of characters).

We also have trivial operations to get and modify characters via array operations (e.g., `str[2]='c'`), and simple string input and output via `<<` and `>>`

5

Borland C++ has a string class library that has further useful operations and hides pointer implementation. Use `cstring.h` header.

```
#include <iostream.h>
#include <cstring.h>

void main()
{
    string s2 = "string";
    string s3 = "this";
    cout << s2.length() << s2+s3 << endl;
    s3 += s2;
    s3.insert(4, " is a ");
    cout << s3;
    s3.replace(0, 4, "that");
    cout << s3;
}
```

What do you think this will output?

7

## Examples

Using standard string library we might have examples like:

```
#include <string.h>

void main()
{
    char *s1 = "hello";
    char *s2 = " there";
    char *s3;
    int x = strlen(s1);
    s3 = strcat(s1, s2);
    cout << s3;
}
```

6

## Introduction to String Searching

- String search = find occurrence of string `s2` in document `s1`.  
e.g., find "cat" in "the dog ate the cat".
- Important in text editors, mailers, and any application that handles text.
- Typically `s2` is quite short (5-20 characters) while `s1` is very long (10,000s of characters).
- Very common operation, so important to have efficient methods of doing the search.

8

### Simple “Brute Force” Search

(Algorithms that try every possibility without doing anything clever are often called “brute force”).

Simplest approach is the following:

If S1=doc, s2=search string:

For each start position i in s1, from 1 to strlen(s1):

– Match characters of s1 with s2, starting with matching s1[i] with s2[0], until match fails.

```
int i=0, j=0;
while(i < s1.length() && j < s2.length())
{
    j=0; // start at beginning of s
    while(j < s2.length() &&
        s1[i+j] == s2[j])
        j++;
    i++;
}
if (j==s2.length())
    cout << "found at " << i-1 << endl;
else cout << "not found";
```

9

### Simple “Brute Force” Search

Example: (\* indicates comparison of chars!):

```
can cats i=0
cat
***      n<>t

can cats i=1
cat
*        c<>a

can cats i=2
cat
*        c<>n

can cats i=3
cat
*

can cats i=4
cat      j=3=length("cat")
***
return 4
```

11

### Common Bug!

What would happen if you had the following:

```
int i=0, j=0;
while(i < s1.length() && j < s2.length())
{
    j=0; // start at beginning of s
    while(s1[i+j] == s2[j]) &&
        j < s2.length())
        j++;
    i++;
}
if (j==s2.length())
    cout << "found at " << i-1 << endl;
else cout << "not found";
```

10

### Efficiency

Efficiency: If S1 of length N, S2 of length M

Worst case efficiency is  $M \times N$ . (e.g.,  
s2=0000001, s1=00000000000000001).

Average case efficiency for English texts is  
M: Usually s2[0] != s1[i] so move string  
up immediately.

But much better efficiency possible.

12

### More Efficient Algorithms

Next lecture will introduce two algorithms that are substantially more efficient:

**Knuth-Morris-Pratt (KMP) Algorithm:** Takes into account way strings repeat themselves, to avoid re-trying matches that are doomed to fail.

**Boyer-Moore Algorithm:** Boyer Moore pre-analyses string and stores where each possible character in alphabet occurs in string, so it can find it right away.

13

### Summary

- Various implementations used for strings in different programming languages - can implement your own if needed.
- String ADT should allow strings to be manipulated in various standard ways - length, replace, concatenate, etc.
- In C++ standard string library with string as character array - usually declared as `char*`; Borland also has string class library.
- String search involves finding one string in another - simple algorithm rather inefficient.

14