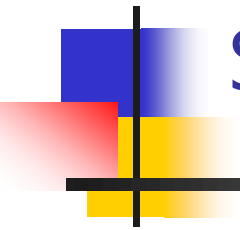
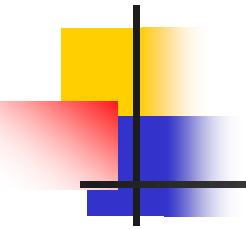


Program Patterns:
Search Patterns

J.W. Choi

2024

Program Patterns

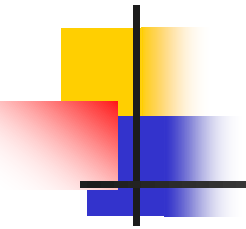


Programming Patterns

- Computers are used for
  - computation and
  - data processing
- Computers were created to “computing”.
- But today computers are used predominantly for “data processing”.

Data Processing Patterns

- There are several frequently used patterns in the design and development of data processing software.
- In this course, we will learn study and think about six of them.
  - Data Search
  - Data Update/Change
  - Data Copying & Moving
  - Data Derivation
  - Data Transformation
  - Data Reorganization



Data Search

- search for max/min
- element count
- Boolean predicate-based search
- string match
  - exact match
  - partial match
  - approximate match

Data Update

- insert
- update/replace
- delete
- append, concatenate
- filling in missing data

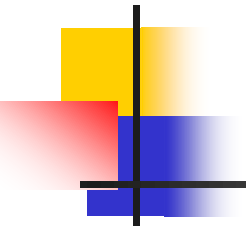
- multiple qualifying data
- constraints on data update
  - unique, null, data typing, value range
  - aggregate

Data Copying & Moving

- data copying/replication
- data subsetting
- data appending
- data moving
- data compaction

Data Derivation

- data aggregation
  - total, average
- data versioning
- data differential
- data sampling
- data lineage (data lifecycle)

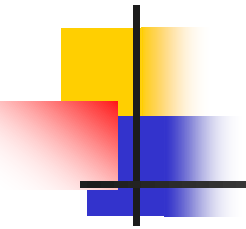


Data Transformation

- data conversion (low level)
  - integer to string, string to integer
  - integer/float to categorical data
  - date/time, money, measurement units, checksum computation
- data compression and decompression
- data encryption and decryption

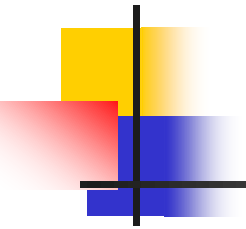
Data Reorganization

- sorting
- grouping
- vertical decomposition
- horizontal decomposition
- recomposition
- data structure change



Data Processing Patterns

- Data search
- Data update
- Data copying & moving
- Data derivation
- Data transformation
- Data reorganization

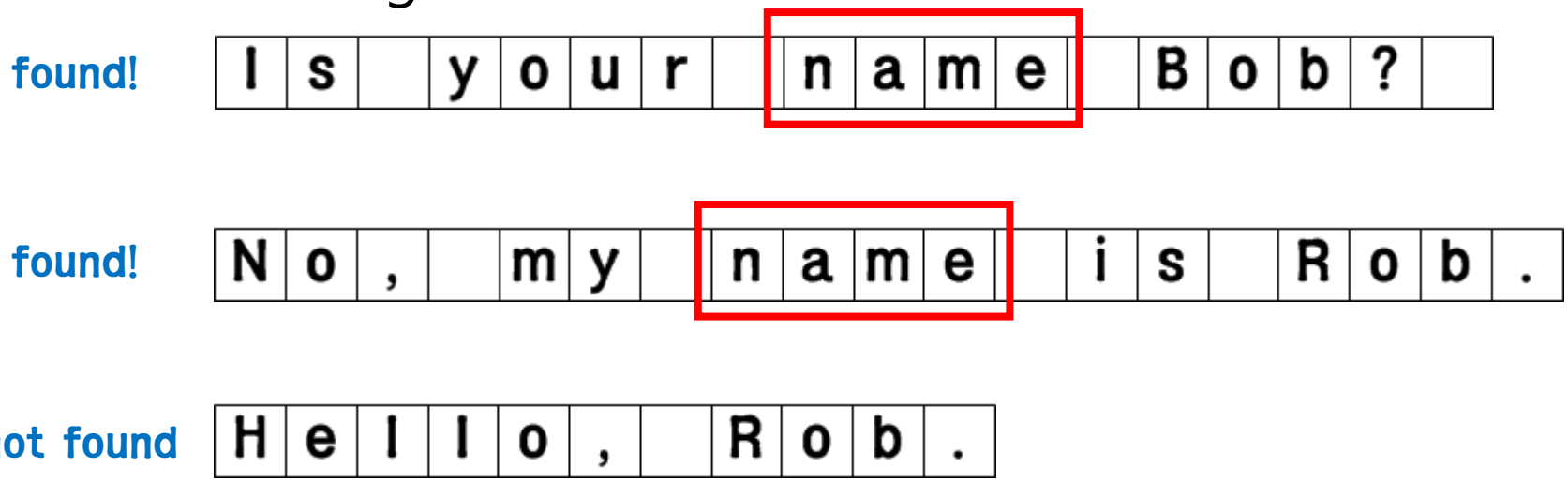


Data Search

- search for max/min
- element count
- Boolean predicate-based search
- string match
  - exact match
  - partial match
  - approximate match

Search for Exact Match

- Given a string, and a search string, determine if the string contains the search string.
- string
  - Is your name Bob?
  - No, my name is Rob.
  - Hello, Rob.
- search string: "name"



Example 1: Problem

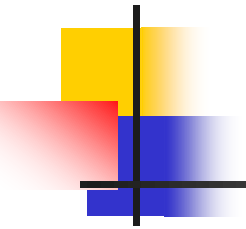
- Given a string, find all sub-strings that match a search string.
- string
  - "A thief named hong gil dong lived with friends named hong gil don and hong gil ja in a village named hong gil dong village."
- search string
  - "hong gil dong"

(Reminder) Software Development Steps

- Follow the steps of program development
  - Step 1: Understand the problem
  - Step 2: Outline a solution (including logic sketch)
  - Step 3: Form a program structure
  - Step 4: Write a pseudo code (including logic sketch)
  - Step 5: Write the program
  - Step 6: Inspect the program
  - Step 7: Compile the program
  - Step 8: Test the program
  - Step 9: Document the source code
  - Step 10: Maintain the source and test cases

Step 1: Understand the problem

- Make up a search string
  - "hong gil dong"
- Think about various match cases
  - no match, partial match, exact match
- Make up an example string (with various cases)
  - " **A thief named hong gil dong lived with friends named hong gil don and hong gil ja and hhhong gil dong and kong gil dong and honggil dong and hong gil donggg in a village named hong gil dong village.**"



Step 2: Outline a Solution

- Loop through the characters in the string
  - outline of the match logic

    find a character that matches the first character of the search string
    - If found, use a loop to find out if the characters that start from the first character exactly match the search string
      - If a match is found, increment count by 1

Using a “Sliding Window”

A thief named hong gil dong lived with friends

hong gil dong ← search string

named hhhong gil dong and hong gil don in a village named hong gil dong village.

Using a "Sliding Window" (cont'd)

A thief named hong gil dong lived with friends

named hhhong gil dong and hong gil don in

hong gil dong

a village named hong gil dong village.

Using a "Sliding Window" (cont'd)

A thief named hong gil dong lived with friends

named hhhong gil dong and hong gil don in

hong gil dong

hong gil dong

a village named hong gil dong village.

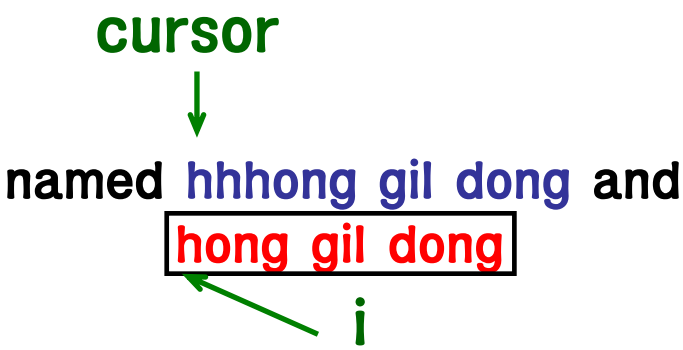
Using a "Sliding Window" (cont'd)

A thief named hong gil dong lived with friends

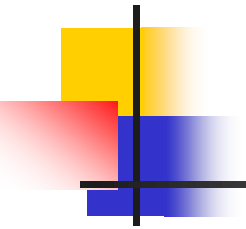
named hhhong gil dong and hong gil don in

a village named hong gil dong village.

Match Logic




```
/* use a cursor to advance the string by one char */
/* use i to advance the search string  */
/* check for  "\0"  in string */

if string[cursor] == search_string[i]
   if i == strlen(search_string) {  /* exact match  */
      match found
      i=0 }       /* for next match  */
   else i++
cursor++         /* move the cursor on the string  */
```



Step 3: Form a Program Structure

read string
read search-string

search for match /* the heart of the program */

print count

Step 4: Write a Program Outline

```
read string
read search-string

/* search for match */
for loop (1 through string length)
   if first character of search-string found
      for loop (1 through length of search-string)
         if substring matches the search-string, found
      if found
         increment count

print count    /* print number of sub-strings that match
                  the search string */
```

Search for Partial Match

- Search using wildcard *
  - search key: "Hong*Dong"
  - any string that starts with "Hong" and ends with "Dong"
  - * (wildcard) means any string of any length

Example (1/2)

- string: “hello mister monkey”

- search string: “money”
- result: match not found

- search string: “mon*ey”
- result: match found

- search string: “m*y”
- result: match found

Example (2/2)

- string: "my name is lee jongho"
- search string: "lee *ho"
- result: match found

- search string: "lee *ha"
- result: match not found

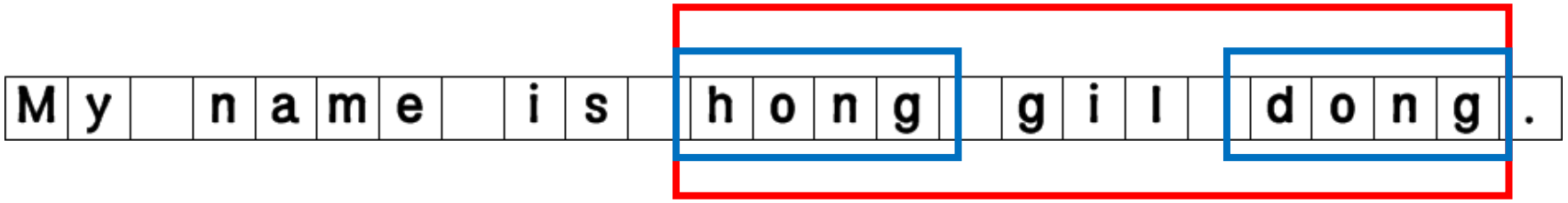
Exercise 2: Problem

- Read a string, and store it.
- Read a search string.
- Determine the number of sub-strings that match the search string, allowing the use of a wildcard *

- Assumption: Wildcard is used only in the middle of a string, and only once (e.g. hong*dong)
  - *dong (x), dong* (x), * (x), a*b*c (x)

Step 1: Understand the problem

- Make up a search string: "hong*dong"
- Think about various match cases.
- Make up an example string (with various cases).
  - "My name is hong gil dong. My brother is hong je dong. My sister is hong gilja, and her friend is hongdong."



Step 2: Outline a Solution

- Divide the search string into two parts
  - hong*dong → first part: hong, second part: dong

- Loop through the characters in the string
  - Logic

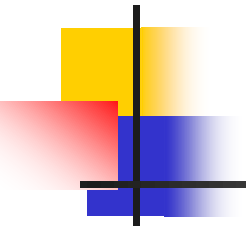
    Find substring that matches the first part of the search string

    My name is hong gil dong.

    - If found, start from the end of the first part and find substring that matches the second part of the search string

      My name is hong gil dong.

      - If second part of the search string is found, increment count by 1 and go to the next part of the string



Step 3: Form a Program Structure

read string
read search-string
divide the search string

search for match /* heart of the program */

print count

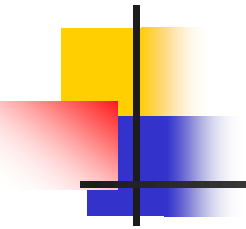
Step 4: Write a Program Outline

```
read string
read search-string
divide the search string

/* search for match */
for loop
    find the first part of the search string
    if found
       for loop
          find the second part of the search string
          if found
             increment count and go to the next part of the string

print count     /* print number of sub-strings that match
                        the search string  */
```

Homework

- Complete Examples 1 and 2 (Do steps 5-9)
- Notes
  - Be sure to document the code.
  - Run the program and screen capture the results.
  - Run the program with 5 different, carefully chosen test datasets (strings and search strings for each string).
  - * Note: Write a function and call it 5 times, each time with a different string and corresponding search strings.

Search Patterns (continued)

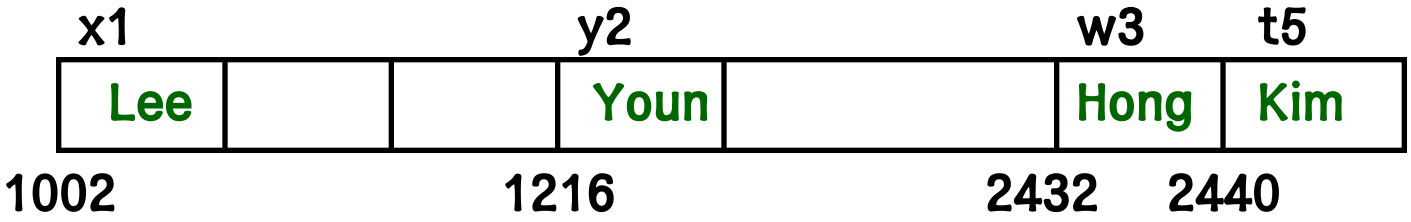


Search Scope

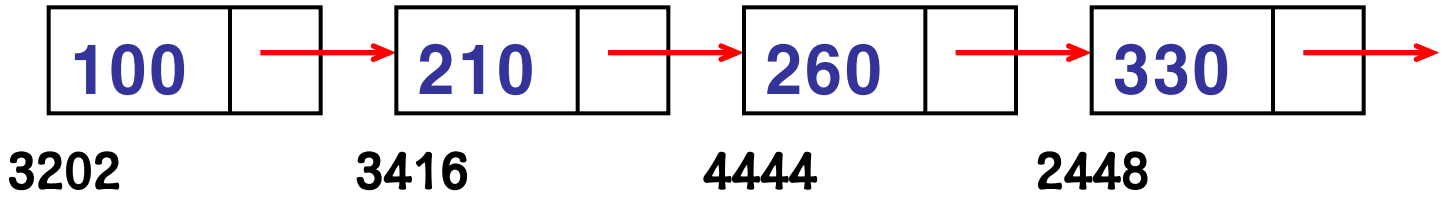
- Exhaustive search
  - look at all the data
- Index-based search
  - zero in on selected data

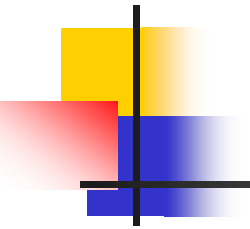
Exhaustive Search

Data in Array



Data in Linked List



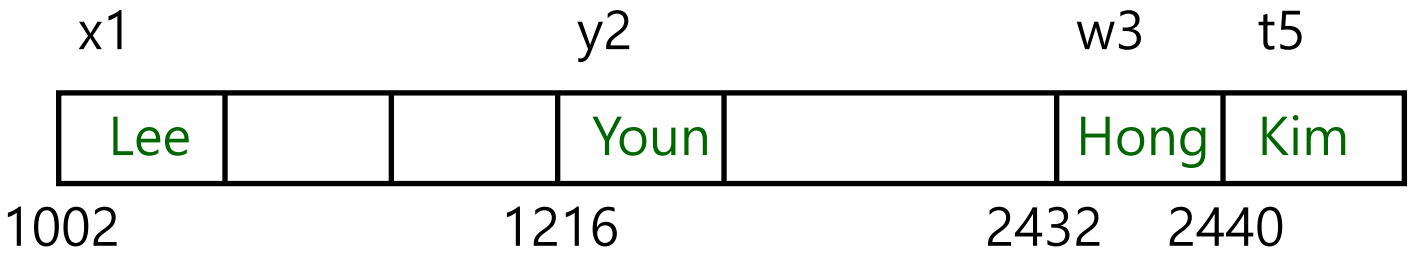


Index-Based Search

- Index
  - Data is stored in some data structure
    - e.g, struct array, linked list, heap, tree,…
  - Index is a separate data structure
  - It maintains (key, key address) pairs for the data.
  - It is useful to get direct access to the data using the key.

Index (for Data Stored in an Array)

Data in Array



Index

Key	Address
Hong:	2432
Kim:	2440
Lee:	1002
Youn:	1216

100	Hong gil dong	22	night worker

2432

Creating & Maintaining an Index

- Array of struct with 2 members
  - key and memory address of the key
  - (e.g.)
```
struct {
    int   key;
    int   *key_ptr;
} index[1000];
```
- Index needs to be changed when data is updated, inserted, or deleted.

Lab: Index Search

- Write the following C program:
  - Store the dataset (on the next page) in a struct array.
    - Each line contains name, age and hobby
  - Create an index (struct array) by name in the order in the dataset (shown on the next page)
  - (* In the index, the names are stored in a sort order; but for this Lab, there is no need to sort the names. *)

  - For names (Lee and Park), do the following:
  - (* You should write a function and call it twice, once for "Lee" and once for "Park".)
    - Search the index for a name, and find and print the (name) age and hobby in the dataset corresponding to the name.
  - (* To search the index, a technique named hashing is used. But for this Lab, do a sequential search of the index. *)

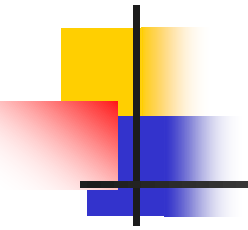
Lab Dataset, and Index

dataset

array index			
0	Kim	39	Tennis
1	Ko	15	Soccer
2	Lee	17	Soccer
3	Choi	21	Tennis
4	Park	10	Tennis

index

index	array index
Kim	0
Ko	1
Lee	2
Choi	3
Park	4



End of Class