RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Recitation 4

Jae Woo Joo

Programming Assignment 2 (First)

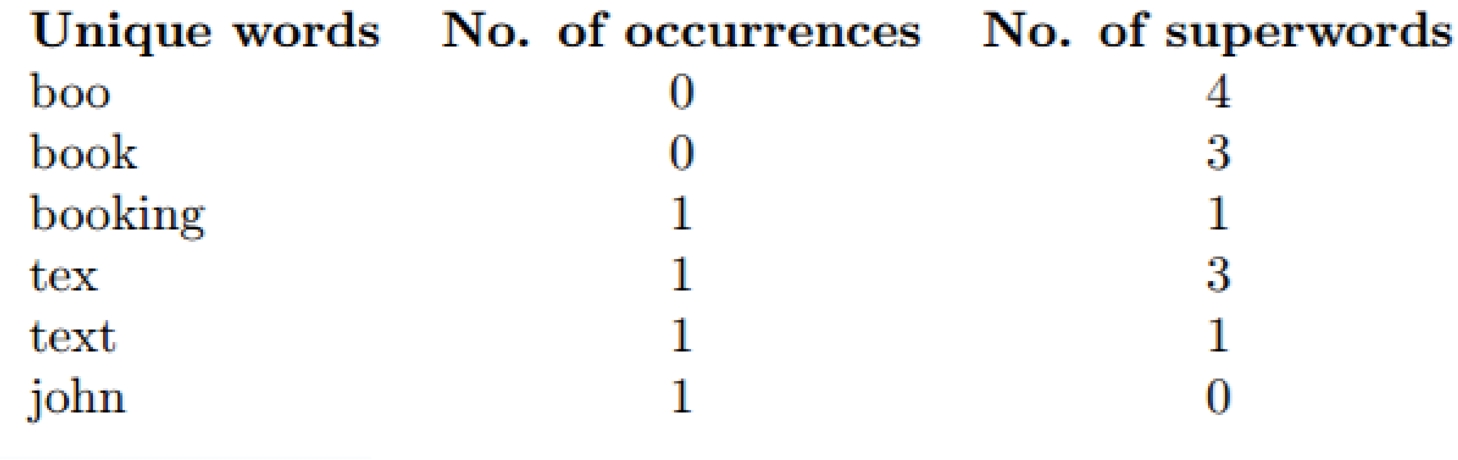
- For every word ***w*** in dic file, count the number of words ***w'*** that occur in the data file s.t. ***w' = w***

- Count the number of words ***w'*** that occur in the data file s.t. ***w*** is a ***proper prefix*** of ***w'*** (***w'*** is a ***superword*** of ***w***)

- Dictionary file
  - boo22$Book5555bOoKiNg#bOo#TeX123tEXT(JOHN)
- Data file
  - John1TEXAN4isa1BOoRiSH%whohasa2bo3KING BOOKING bOoKIngs$12for a TEX-Text(BOOKS(textBOOKS)

Programming Assignment 2 (First)

- The w
  - boo, book, booking, boo, tex, text, john
- The w'
  - john, texan, isa, boorish, wohasa, bo, king, booking, bookings, for, a, tex, text, books, textbooks



Unique words	No. of occurrences	No. of superwords
boo	0	4
book	0	3
booking	1	1
tex	1	3
text	1	1
john	1	0

Programming Assignment 2 (First)

- Usage interface

  -
    ```
    $ ./first <mapping file>
    ```

- Structure of Mapping file
  - map.txt

    ```
    dict_1 data_1
    dict_2 data_2
    …
    dict_m data_m
    dictm foom
    ```

Programming Assignment 2 (First)

- Output specification
  - You should generate several output files **out*i*.txt** (*i* is the num of mapping file)
  - Suppose line *j* in the mapping files is **dict_j data_j**. Then you should produce **out*j*.txt**
  - Format of **out*j*.txt**

    <Word1> <count11> <count12>

    <Word2> <count21> <count22>

    …

    <Wordn> <countn1> <countn2>

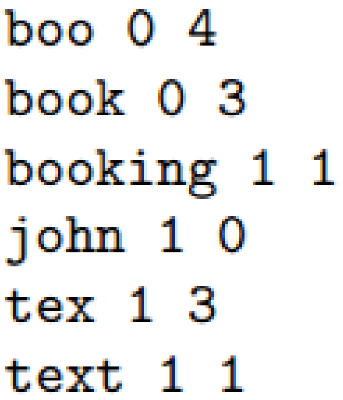
Programming Assignment 2 (First)

- (Example)
- Let’s assume you have dict_1.txt, data_1.txt, and map.txt

- dict_1.txt
  - boo22$Book5555bOoKiNg#bOo#TeX123tEXT(JOHN)
- data_1.txt
  - John1TEXAN4isa1BOoRiSH%whohasa2bo3KING BOOKING bOoKIngs$12for a TEX-Text(BOOKS(textBOOKS)
- map.txt
  - dict_1 data_1

- If you run your first part
  - $ ./first map.txt

out1.txt



```
boo 0 4
book 0 3
booking 1 1
john 1 0
tex 1 3
text 1 1
```

Programming Assignment 2 (Second)

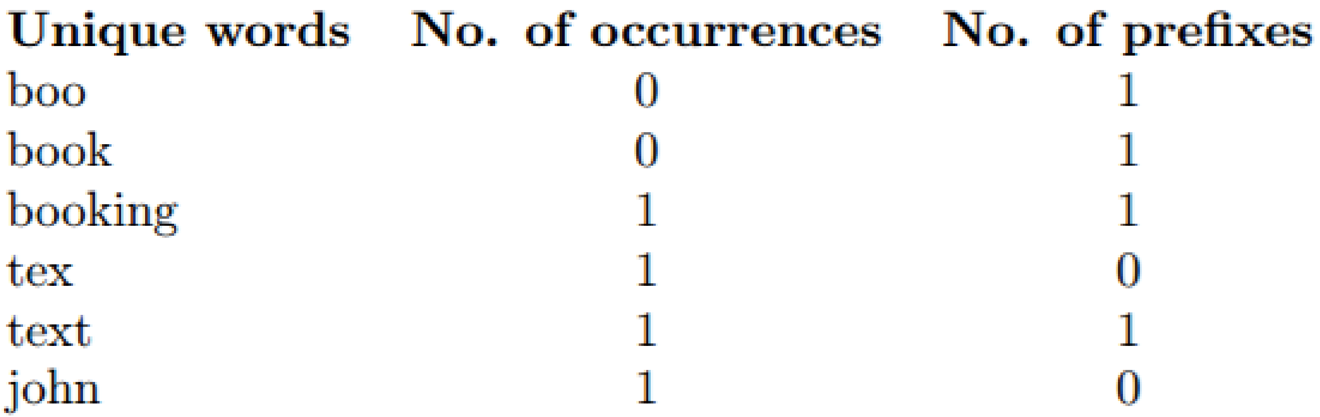
- For every word ***w*** in dic file, count the number of words ***w'*** that occur in the data file s.t. ***w' = w***

- For every word ***w*** in dic file, count the number of words ***w'*** that occur in the data file s.t. ***w'* is a proper prefix of *w***

- Dictionary file
  - boo22$Book5555bOoKiNg#bOo#TeX123tEXT(JOHN)
- Data file
  - John1TEXAN4isa1BOoRiSH%whohasa2bo3KING BOOKING bOoKIngs$12for a TEX-Text(BOOKS(textBOOKS)

Programming Assignment 2 (Second)

- The w
  - boo, book, booking, boo, tex, text, john
- The w'
  - john, texan, isa, boorish, wohasa, bo, king, booking, bookings, for, a, tex, text, books, textbooks



Unique words	No. of occurrences	No. of prefixes
boo	0	1
book	0	1
booking	1	1
tex	1	0
text	1	1
john	1	0

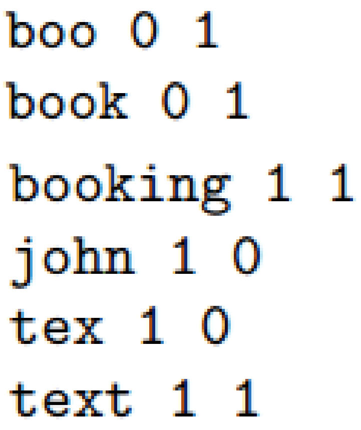
Programming Assignment 2 (Second)

- (Example)
- Let’s assume you have dict_1.txt, data_1.txt, and map.txt

- dict_1.txt
  - boo22$Book5555bOoKiNg#bOo#TeX123tEXT(JOHN)
- data_1.txt
  - John1TEXAN4isa1BOoRiSH%whohasa2bo3KING BOOKING bOoKIngs$12for a TEX-Text(BOOKS(textBOOKS)
- map.txt
  - dict_1 data_1

- If you run your first part
  - $ ./second map.txt

out1.txt

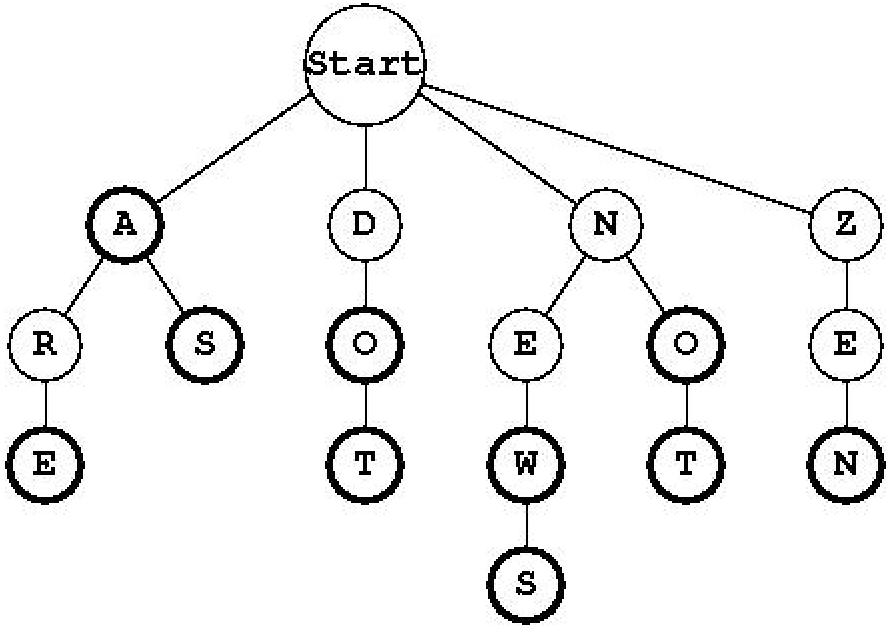


```
boo 0 1
book 0 1
booking 1 1
john 1 0
tex 1 0
text 1 1
```

Programming Assignment 2 (Design)

- Any data structure is fine, but we recommend using the data structure “trie”

- Trie is also called as “prefix tree”



Programming Assignment 2 (Design)

- Trie insert
  - Every character of input key is inserted as an individual trie node
  - The key character acts as an index
- Trie search
  - Similar to insert operation
  - Only compare the characters and move down
  - Search can terminate due to end of string or lack of key in trie

Programming Assignment 2 (Implementation)

- Void readDict(FILE *dict_file)
  - The function takes a file pointer to the dictionary file and reads the unique words from the dictionary file and store them
- Void matchStr(char* str)
  - The function will take a word and search the data structure that holds the unique dictionary words in order to find matches and update the counts
  - It should be used while scanning the data file for occurrences of dictionary words and their prefixes and superwords
- Void printResult()
  - Produce the output of the program
- Any other functions could be added

Programming Assignment 2 (Implementation)

- If you store all the dictionary words in an array, then matching the word by doing a linear search

  => Your program will exceed the time limit!!!

- Think of some data structures that could be efficient

- We recommend the running time and space complexity be O(mk) and O(nk) respectively
  - M: maximum number of words in either the dictionary or data files
  - K: Every word length
  - N: the number of unique words in dictionary file

Header file in C

- A header file is a file with extension **.h**
  - C function declarations
  - Macro functions
  - #define
  - Global variables

- It can be used in your .c file
  - For example, let’s say you have first.c, first.h
  - In your first.c, include
    - #include “first.h”

Header file in C

- Header file format

```
#ifndef _first_h
#define _first_h
….
- Global variables
- Function declaration
….
#endif
```

- If a header file happens to be included twice, the compiler will process its contents twice and it will occur an error

Programming Assignment 2 (Submission)

```
pa2
    | - first
          | -- first.c
          | -- first.h
          | -- Makefile
          | -- readme.pdf
    | - second (if you did)
           | -- second.c
           | -- second.h
           | -- Makefile
           | -- readme.pdf
```

Programming Assignment 2 (Submission)

- Source code
  - A c file and a header file
  - Ex) first.c & first.h
- Makefile
  - first: build your first executable
  - clean: prepare for rebuilding from scratch
- Readme file (pdf format)
  - Brief description of your data structures
  - Big O analysis of run time, and space requirements
  - Any challenges that you encountered

=> If you did second as well, the format should be the same

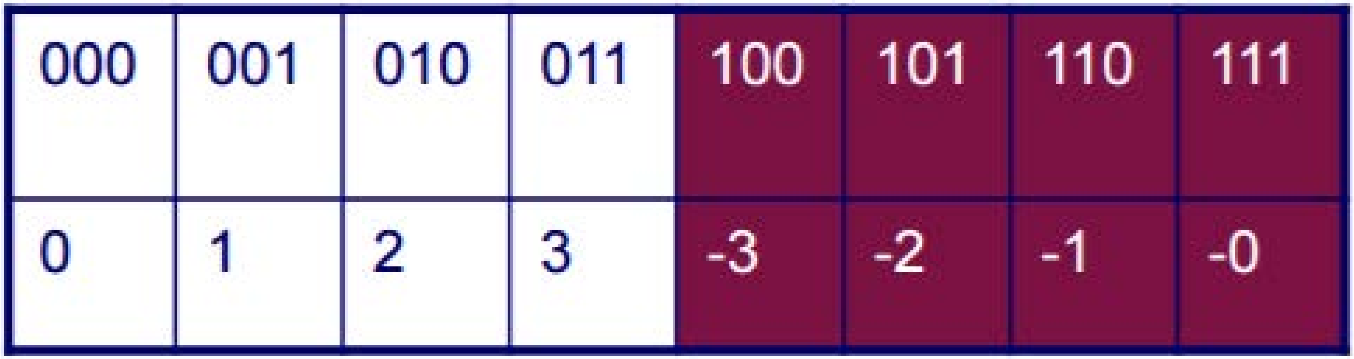
Programming Assignment 2 (Auto grader)

```
autograder
        pa2
            | - first
                 | -- first.c
                 | -- first.h
                 | -- Makefile
            | - second (if you did)
                 | -- second.c
                 | -- second.h
                 | -- Makefile
```

One’s Complement

- Represent negative numbers by complementing positive numbers

- It has two zero representation



000	001	010	011	100	101	110	111
0	1	2	3	-3	-2	-1	-0

Two’s Complement

- Advantages – only 1 zero & convenient for arithmetic computation

- Flip the bits and add 1 (One’s complement + 1)
- Ex)

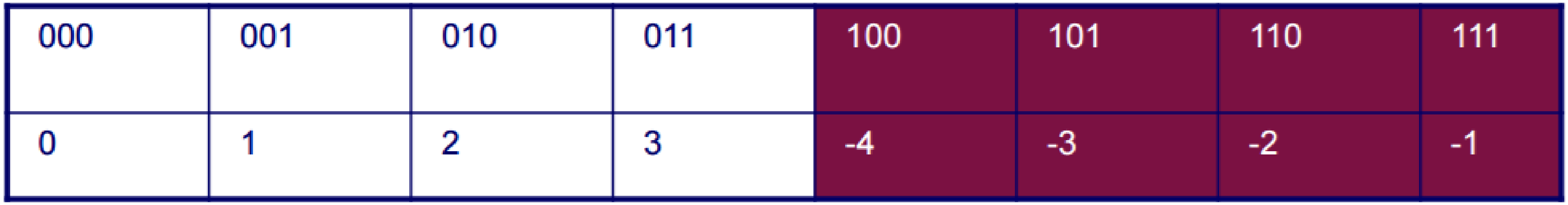
```
          40 = 0010 1000
  -> Flip    1101 0111
  -> Add1 1101 1000 (-40 in two’s complement form)

          -40 = 1101 1000
  -> Flip      0010 0111
  -> Add1   0010 1000 (two’s complement of -40)
```

Two's complement

- What is the range that can represent with n bits?

$$[-2^{n-1}, 2^{n-1} - 1]$$

- More negative numbers than positive numbers
  - Since we only have 1 zero



000	001	010	011	100	101	110	111
0	1	2	3	-4	-3	-2	-1

Arithmetic of two's complement

- Arithmetic addition

+ 6	0000 0110		- 6	1111 1010
+13	0000 1101		+13	0000 1101
+19	0001 0011		+ 7	0000 0111

Arithmetic of two’s complement

- Arithmetic subtraction

-5	1111 1011	-5	1111 1011
- 6	0000 0110	- -6	1111 1010
-11	1111 0101	+ 1	0000 0001

Two’s complement overflow

- It needs one extra bit but the sign bit will be wrong

- How to detect an overflow?
  - Adding 2 positive numbers -> But negative result
  - Adding 2 negative numbers -> But positive result

$$\begin{array}{rr} 6 & 0110 \\ \underline{+\ 5} & \underline{0101} \\ -5 & 1011 \end{array} \qquad\qquad \begin{array}{rr} -6 & 1010 \\ \underline{+\ -6} & \underline{1010} \\ 4 & 0100 \end{array}$$

Q & A

- Any questions?