The Most Frequent Word in Source Code on GitHub

Kenta Motomura

July 28, 2012

1 Introduction

1.1 Background and Objective

The most frequent alphabet in English is "E" and the most frequent word is "the". Subsequent words are "of", "and", "in", "to" and more. The distribution of words is accordance with Zipf's law, that is reported in the reference[1].

How about programming languages? What is the most frequent word in program source code? The objective of this research is to clarify the most frequent word in programming languages and word distributions.

The original motivation was to make the data that will be helpful in order to decide the variable name. But those words appear in the lower-ranking frequently used words. If you want to get them, please refer to the author's web site*1. Due to space limitations in this paper, printed in only top 20 ranked words.

1.2 Analysis target

In reference[1] paper, analysis target was all articles of Wikipedia. We selected GitHub *2 for analysis target in this research. GitHub doesn't provide a tarball hold all repositories like Wikimedia Downloads *3 that provides Wikipedia dumps. We tried to make a repositories list by recursive download from search result pages, but GitHub doesn't provide result pages of over 100 so that was impossible.

It is not necessarily required to analyze all repositories to clarify words using trend, so we decided the analysis target in this research is "Top 200 Most Watched repositories per language on GitHub Top 10 Languages *4 in July 2012."

Top 200 repositories multiplied 10 languages is 2000 repositories to be analyzed. Selected 10 languages

for analysis target are JavaScript(20%), Ruby(14%), Python(9%), Shell(8%), Java(8%), PHP(7%), C(7%), C++(4%), Perl(4%) and Objective-C(3%). (% means the share of the language in GitHub)

2 Preparation

2.1 Getting data

We got names of repositories from Most Watched ranking *5 of top languages and used GitHub API v3 *6 to get repository contents tarballs. In this stage, 7.2%, 144 repositories in 2000 repositories of analysis target didn't respond tarballs, so We removed such repositories from analysis targets.

2.2 Extract source code files

We extracted files with suffix extensions of programming languages from tarballs. The strings of suffix extensions are JavaScript(js), Ruby(rb), Python(py), Java(java), PHP(php php3), C(c h), C++(cc cxx cpp hxx hpp h) and Objective-C(m h). Some files of Shell and Perl cannot be determined the file type by finding suffix extensions, so we checked the contents of files. If the first line of the file is matched a following regular expression, that's Shell.

^\(#!/bin/[a-z]*sh\).*

If the first line of the file is matched a following regular expression, that's Perl.

^\(\#!/bin/[a-z/]*perl\).*\|^\(package \).*

We decided the file types under these rules. In this stage, the number of files was 714,959, and the total size of source code is 6.16 GB. The number of program source files, the total bytes of source files and the average file size of a file are a following list, Fig.1, Fig.2 and Fig.3.

language	filenum	<pre>size(byte)</pre>
C++	154273	1653647781
C	201646	2666746054
Java	163589	1125790663

^{*5} https://github.com/languages/C/most_watched, etc

^{*1} http://knt5lab.appspot.com/

^{*2} GitHub is a popular web-based hosting service for software development projects that use the Git revision control system. http://github.com/

^{*3} http://dumps.wikimedia.org/

^{*4} https://github.com/languages

^{*6} http://developer.github.com/v3/

JavaScript	16971	235269012
Objective-C	21114	95801465
PHP	70822	435027550
Perl	6925	38078855
Python	28697	207131755
Ruby	50104	154138671
Shell	818	2441018

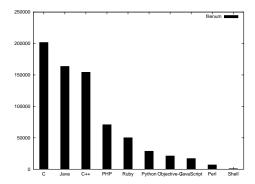


Fig. 1 The number of program source files

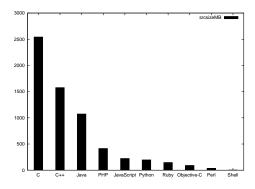


Fig. 2 The total bytes of source files (MB)

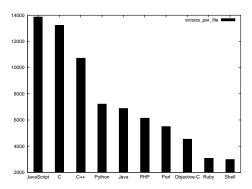


Fig. 3 The average size per file (byte)

2.3 Comments strip

We stripped program comments "// to line end" and "/* to */" from C, C++, Objective-C, Java, JavaScript and PHP, and removed "# to line end" from Python, Ruby, Perl, Shell and PHP. In this method, the comment strip of Ruby, Perl and Python is not enough. This is as a theme for the future in this research, we analyze mainly C, C++, Objective-C, Java, JavaScript, PHP and Shell. They were stripped comments perfectly. String literals are subject to this analysis so that the removal of the string literals are not performed.

2.4 Definition of "word"

We defined "word" to a set of alphabets and numbers. For example, int, bmp2png, 0x100 and 256 are words. We defined as "space" any other characters. For example, "/", "{", ")", "-" and NULL are spaces. We analyzed all source code based on this definition.

3 Method

The excellent point of my method is "insert a new item to the break point of binary search". First, we do a normal binary search. If the word is found in the word list, the number of the word will be counted up. Second, if the word is not found in the word list, "the minimum index" (=min) in binary search is the position to insert a new item. (Fig.4)

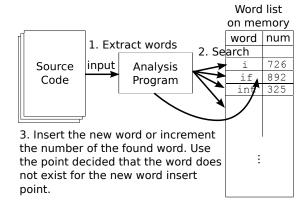


Fig. 4 The process flow

The implemented C code is following.

```
// binary search
while(max >= min) {
  mid = (min + max) / 2;
  if(strcmp(list[mid]->word, buf) < 0) {
      min = mid + 1;
}</pre>
```

```
} else if(strcmp(list[mid]->word, buf) > 0) {
      max = mid - 1;
  } else {
      found = 1;
      break;
 }
}
// found
if(found) {
  // [mid] is a found point
 list[mid] ->num ++;
 return;
}
// ...(omitted)...
// insert
// [min] is a point to insert
list[min] = item;
```

This is it. We used the "min" for the insert point. The time to search the insert point was 0. That is already done in searching to check whether the word already exists in the word list.

Could we use bsearch() in C stdlib? - No. It returns only a found pointer. If not found, it just returns NULL. If not found, we need to get the value of "min" on the end of search. But bsearch() doesn't provide such value, so we cannot use it.

We analyzed source code based on the method. First, extract words from input stream. Second, search the word whether it already exists in the word list. Finally, if found, increment the number of the word. If not found, insert the word to the "min" position of the word list. It's zero search. We just need memory shift.

4 Result

The analysis result is following.

rank	num	word
1	12536894	0
2	9174613	if
3	7211215	1
4	6344625	return
5	5777769	int
6	5284984	00x0
7	5210612	define
8	5084560	struct
9	3769638	void
10	3714602	i
11	3284344	const
12	3077231	static

```
13
    2882689 2
14
    2667224 t
15
    2661897 this
16
    2595296 type
17
    2403928 n
    2374491 dev
18
19
    2206270 typename
20
    2118588 public
```

The most frequent word in program source code on GitHub is "0". We use 0 to initialize variables every where. Among the magic number, 0 is allowed at many places, it's interesting.

No.2 is reserved word, "if". Every top 10 languages has it. No.19 "typename" is one of C++ reserved words. Java has No.20 "public". Most of No.18 "dev" was used as devices. The word comes from C, Linux kernel project.

The amount size of code is different depending on the language. The above result has a bias of large amount of languages such as C, C++ and Java. The results for each language are listed in the Appendix.

Finally, we show Fig.5 of a distribution of words. The straight dotted line is an ideal.

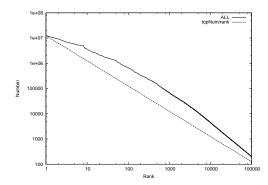


Fig. 5 the distribution of words

That is in accordance with Zipf's law. Programming languages has a characteristics of natural languages.

5 Conclusion

We research the word trend in source code with a normal binary search plus alpha. The most frequent word in program source code of popular 1856 repositories on GitHub is "0", and "if". It is proven with Zipf's law that programming language is one of natural languages. If you want to know results of each language, please continue reading Appendix A.

Appendix A The results for each language

A.1 C

C		
rank	num	word
1	6146035	0
2	4891254	0x00
3	4705643	struct
4	4664406	if
5	4338966	define
6	3490574	int
7	3331332	1
8	2677817	return
9	2362372	dev
10	2082935	static
11	1974376	i
12	1818830	void
13	1559201	n
14	1558874	t
15	1380114	unsigned
16	1279201	h
17	1259813	data
18	1200809	2
19	1132519	include
20	1120355	S

"0" is No.1. "1" and "2" are also ranked in. The reserved words of C are struct, if, define, int, return, static, void, unsigned and include. No.11 is "i". It's often used for indexer, especially in "for" statement. No.16 "h" is used for extensions of header files, like #include<stdio.h>. No.19 "include" is suited in combination with "h".

A.2 C++

C++		
rank	num	word
1	3092328	0
2	2407359	const
3	2185141	typenam
4	1980505	if
5	1977830	1
6	1575812	return
7	1538026	type
8	1458276	int
9	1236206	void
10	1116292	typedef
11	1053923	m
12	907114	i

13	879121	class
14	853601	BOOST
15	828639	define
16	825507	t
17	732219	2
18	712400	include
19	621149	ACE
20	589779	result

"0" is No.1 again. No.14 "BOOST" and No.19 "ACE" are interesting. They are library sets or frameworks for C++. We got a degree of popularity of the library in a target language through the research. That's interesting.

A.3 Java

Java		
rank	num	word
1	1548837	public
2	1189715	import
3	1036363	return
4	1028678	if
5	939549	String
6	835153	new
7	750503	int
8	722877	final
9	703868	0
10	670320	null
11	612922	com
12	608461	void
13	596945	org
14	578719	1
15	514793	private
16	497025	static
17	435057	this
18	413554	java
19	404779	class
20	351926	i

No.1 is "public". No.11 "com" and No.13 "org" are interesting. They are used for package names. The number of "com" and "org" is almost the same number of "import".

A.4 JavaScript

JavaScript			
rank	num	word	
1	1585184	0	
2	958519	var	
3	551508	1	
4	486903	this	

5	439424	if
6	417286	function
7	403660	2
8	373297	a
9	286398	return
10	271008	b
11	247881	i
12	245712	4
13	229418	С
14	213057	8
15	211056	3
16	194877	4294967295
17	190654	d
18	181739	е
19	176903	5
20	162685	6

 $^{\circ}0^{\circ}$ is No.1 again. There are many magic numbers and short variable names like a, b or c. No.16 $^{\circ}4294967295^{\circ}$ is interesting. It's used for a maximum number everywhere.

A.5 PHP

PHP		
rank	num	word
1	1222473	this
2	626742	array
3	490418	if
4	428536	function
5	376373	return
6	341948	x
7	297129	public
8	283763	0
9	224594	id
10	222114	1
11	179525	php
12	172359	new
13	171420	name
14	162234	1000
15	159836	null
16	158497	class
17	144447	false
18	140839	value
19	133731	lang
20	126800	true

No.1 is "this".

A.6 Objective-C

Objec	ctive-C	
${\tt rank}$	num	word

1	163304	self
2	120977	if
3	98350	void
4	94333	0
5	93318	return
6	86468	NSString
7	61358	id
8	59469	nil
9	51831	h
10	48686	import
11	43553	BOOL
12	38670	1
13	33035	end
14	27715	else
15	26222	int
16	26023	YES
17	24849	alloc
18	23217	NO
19	22273	size
20	22047	property

No.1 is "self". No.16 "YES" and No.18 "NO" are "BOOL" values.

A.7 Shell

Shell rank	num	word
1	6418	1
2	5289	_
3	4041	
4	4033	
5	3958	
6	3623	
7	2384	
8		
9	2367 1818	•
10	1751	
11	1741	
12	1669	
13	1602	
14	1498	name
15	1478	the
16	1447	to
17	1390	file
18	1364	d
19	1317	s
20	1309	f

 $\rm ``1"$ is No.1. No.8 is "git" because GitHub has many git utility written in Shell.

A.8 Python, Perl and Ruby

Python, Perl and Ruby are not stripped comments perfectly.

100013.				
Python				
rank	num	word		
	1053604			
2	461005	0		
3	320134	def		
4	283445	1		
5	272219	if		
6	214872	the		
7	211776	return		
8	187084	name		
9	178753	None		
10	178226	a		
11	178022	in		
12	151475	is		
13	147688	import		
14	147212	for		
15	142698	2		
16	142281	s		
17	137890	u		
18	135869	to		
19	128236	from		
20	115005	get		
Perl				
		,		
rank		word		

Perl		
rank	num	word
1	123877	my
2	97577	self
3	70622	the
4	47578	if
5	46695	1
6	46631	sub
	44611	
8	39681	return
9	37262	name
10	36779	use
11	36725	is
12	36063	id
13	34084	a
14	33909	0
15	27235	for
16	26912	get
17	26781	of
18	25772	shift
19	21982	С
20	21976	and

rank	num	word
1	739641	end
2	263685	do
3	260162	def
4	228260	assert
5	206872	should
6	174484	1
7	147050	new
8	141169	equal
9	136831	a
10	133595	to
11	132846	0
12	130953	name
13	126375	if
14	108420	test
15	108344	it
16	103338	nil
17	94199	class
18	80565	2
19	75019	path
20	74016	the

Every language does not have "the" for No.1. Noise caused by the natural language in comments is not strong enough to counteract fully the characteristics of programming languages.

A.9 The distribution of words for each language

The straight dotted line is an ideal. JavaScript has the strongest characteristics of a natural language. Objective-C has the best balance of the characteristics of programming language and natural language.

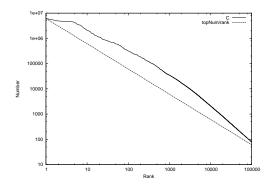


Fig. 6 C

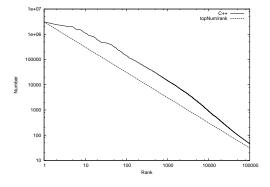


Fig. 7 C++

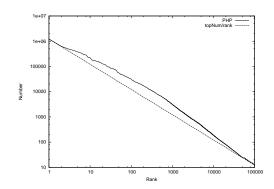


Fig. 10 PHP

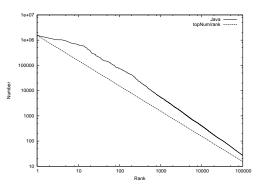


Fig. 8 Java

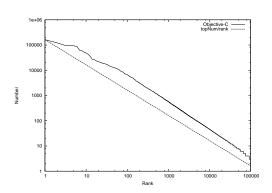


Fig. 11 Objective-C

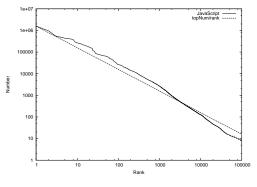


Fig. 9 JavaScript

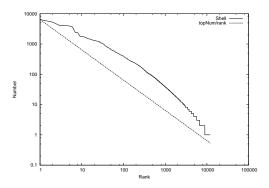


Fig. 12 Shell

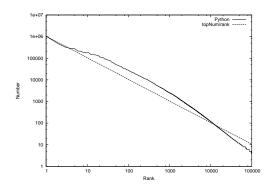


Fig. 13 Python

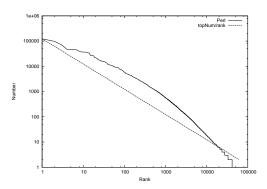


Fig. 14 Perl

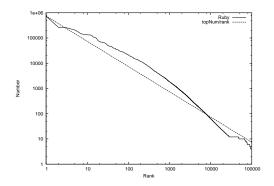


Fig. 15 Ruby

Appendix B The analysis program source code

All source code is in GitHub. https://github.com/knt5/githubwords

B.1 Extraction and count up of words

```
1 #include<stdio.h>
 2 #include<stdlib.h>
 3 #include<string.h>
 4 #define BUF_LEN 4096
 5 #define ITEM_MAX 100000000
 7 struct ITEM {
 8
       char *word;
9
       unsigned int num;
10 };
11
12 int length = 0;
14 void add(char *buf, struct ITEM **list) {
       int min=0, max=length-1, found=0, mid=0, 1;
16
       struct ITEM *item;
17
       // binary search
18
19
       while(max >= min) {
20
           mid = (min + max) / 2;
21
           if( strcmp(list[mid]->word, buf) < 0 ) {</pre>
22
               min = mid + 1;
23
           } else if(strcmp(list[mid]->word, buf) > 0) {
24
               max = mid - 1;
25
           } else {
26
               found = 1;
27
               break;
28
29
       }
30
31
       // found
32
       if(found) {
33
           list[mid]->num ++;
34
           return;
35
36
37
       // check length
38
       if(length >= ITEM_MAX) {
39
           fprintf(stderr, "No more memory, cannot add: %s\n", buf);
40
           return;
41
       }
42
43
44
       memmove(list+min+1, list+min, (length-min) * sizeof(struct ITEM*));
45
46
       // copy
47
       item = (struct ITEM*)malloc(sizeof(struct ITEM));
48
       l = strlen(buf) + 1;
```

```
49
        item->word = (char*)malloc(1);
50
        item->num = 1;
        memcpy(item->word, buf, 1);
51
 52
53
        // add
        list[min] = item;
54
55
        length++;
56
 57
        return;
58 }
59
 60 int main(int ac, char *av[]) {
61
        FILE *fp;
        char buf[BUF_LEN+64], *s;
 62
 63
        int c, 1, b, i;
        int isfile = 0;
 64
        struct ITEM** list;
 65
 66
 67
        // check args
 68
        if(ac != 1 && ac != 2) {
 69
            fprintf(stderr, "Usage: %s filename\n", av[0]);
 70
            return 1;
 71
        }
 72
 73
        // open stream
 74
        if(ac == 2) {
 75
            fp = fopen(av[1], "rb");
 76
            if(fp == NULL) {
                 \label{eq:connot_open_file: $$\n^*, av[1]);} fprintf(stderr, "Cannot open file: $$\n^*, av[1]);
 77
78
                 return 2;
79
            }
80
            isfile = 1;
 81
        } else {
 82
            fp = stdin;
 83
84
85
        // memory alloc
        1 = sizeof(struct ITEM*) * ITEM_MAX;
86
87
        list = (struct ITEM**)malloc(1);
88
        memset(list, 0, 1);
89
90
        // scan
        s = buf;
91
        b = 0;
92
93
        1 = 0;
        while(1) {
 94
95
            c = fgetc(fp);
 96
            if(c == EOF) {
 97
                 if(ferror(fp)) fprintf(stderr, "Read error: %s\n", av[1]);
98
                 break;
            }
99
            if( ('a' <= c && c <= 'z')
100
101
             || ('A' <= c && c <= 'Z')
             || ('0' <= c && c <= '9')
102
103
            ) {
104
                 if(b == 0) b = 1;
```

```
105
                *s = c;
106
                s++;
107
                1++;
108
                if(1 > BUF_LEN) { // search cancel
109
                    s = buf;
110
                    b = 0;
                    1 = 0;
111
                }
112
            } else {
113
                if(b) { // break point
114
115
                    *s = 0;
116
                    if(1) add(buf, list);
117
                    s = buf;
                    b = 0;
118
                    1 = 0;
119
                }
120
            }
121
122
123
124
        // close file
125
        if(isfile) fclose(fp);
126
127
        // print
128
        for(i=0; i<ITEM_MAX && list[i]; i++) {</pre>
129
            printf("%d %s\n", list[i]->num, list[i]->word);
130
131
132
        // free : not required
133
        //free(list);
134
135
        return 0;
136 }
B.2 C/C++ style comment strip
  1 #include<stdio.h>
  2 #include<stdlib.h>
  3 #include<string.h>
  5 // skip comment
  6 int skip(FILE *fp) {
        int c = fgetc(fp);
 9
        if(c == '/') {
 10
            // single line comment
            while(c != EOF && (c=fgetc(fp)) != '\n') {
11
                if(c == '\\') {
12
13
                    c = fgetc(fp); // \\n = void
 14
 15
            }
 16
            if(c!=EOF) return '\n';
17
        } else if(c == '*') {
 18
            // multi line comment
 19
 20
            do {
```

```
21
               while( c != EOF && (c=fgetc(fp)) != '*') {
22
                   //none
23
               }
24
           } while(c != EOF && (c=fgetc(fp)) != '/');
25
           if(c!=EOF) return 0;
26
       } else {
27
           if(c != EOF) c = c | ('/' << 8);
28
       }
29
       return c;
30 }
31
32 // main
33 int main(int ac, char *av[]) {
       FILE *fp;
35
       int c;
36
       int isfile = 0;
37
       // check args
38
39
       if(ac != 1 && ac != 2) {
40
           fprintf(stderr, "Usage: %s filename\n", av[0]);
41
           return 1;
42
       }
43
44
       // open stream
45
       if(ac == 2) {
46
           fp = fopen(av[1], "rb");
47
           if(fp == NULL) {
48
               fprintf(stderr, "Cannot open file\n");
49
               return 2;
50
           }
51
           isfile = 1;
52
       } else {
53
           fp = stdin;
54
55
56
       // scan
57
       while(1) {
58
           c = fgetc(fp);
59
           if(c == EOF) break;
60
61
           // skip comment
           if(c == '/') {
62
               c = skip(fp);
63
               if(c == EOF) break;
64
               if(c == 0) continue;
65
66
               if(c > 0x0FF) {
67
                   putchar('/');
68
                   c = c \& 0x0FF;
69
               }
70
           }
71
72
           // output
73
           putchar(c);
       }
74
75
       // check error
```

```
77    if(ferror(fp)) fprintf(stderr, "Read error\n");
78
79    // close file
80    if(isfile) fclose(fp);
81
82    return 0;
83 }
```

References

[1] Kenta Motomura, Morikazu Nakamura, Joji M. Otaki, "Sequence decoding and design based on short constituent amino acid sequences in proteins", p.45. (2009) http://bio.ads.ie.u-ryukyu.ac.jp/