Programming Eliot

Antoine Fraboulet

December 28, 2005

1 The Dictionary

The dictionary is a directed tree. Compression is achieved by sharing word prefix. Search is NOT case sensitive.

considering this 3 words dictionary:

```
ABC
ADA
EDAA
```

The tree will look like this:

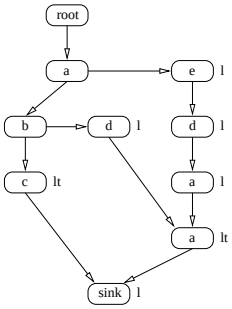

Figure 1: Dictionary Directed Acyclic Word Graph

1.1 Binary Structure

The tree is saved using an array of 32 bits words. A cell is a binary structure

- ptr : index in the array of the first child
- term : is it the last letter of a word (*)
- last : is it the last child of its local root (!)

- fill : currently unused
- chr : the letter

There is no pointer from a cell to its brother, it is simply the next cell in the array (you know you are on the last brother when the flag "last" is set).

The way it is stored in a file is different thing! The tree is stored bottom-up. The sink (offset 0) is the first cell of the array.

Using compdict (which you can found in the eliot/dic directory), the compiled dictionary will look like this:

compdict's console output:

```
keyword length 21 bytes
keyword size   22 bytes
header size    48 bytes

3 words

root :       9 (edge)
root :      36 (byte)

nodes : 7+1
edges : 9+1
```

binary view of the dictionary:

```
          0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
00000000: 5f43 4f4d 5049 4c45 445f 4449 4354 494f  _COMPILED_DICTIO
00000010: 4e41 5259 5f00 0000 0900 0000 0300 0000  NARY_............
00000020: 0900 0000 0700 0000 0100 0000 0100 0000  ................
00000030: 0000 0002 0000 001b 0000 000b 0100 0010  ................
00000040: 0200 0022 0200 000a 0500 0022 0300 0008  ..."......."....
00000050: 0600 002a 0700 0000                      ...*....
```

The header structure is the following:

```
#define _COMPIL_KEYWORD_ "_COMPILED_DICTIONARY_"

typedef struct _Dict_header {
  char ident[sizeof(_COMPIL_KEYWORD_)];    // 0x00
  char unused_1;                           // 0x16
  char unused_2;                           // 0x17
  int root;                                // 0x18
  int nwords;                              // 0x1c
  unsigned int edgesused;                  // 0x20
  unsigned int nodesused;                  // 0x24
  unsigned int nodessaved;                 // 0x2c
  unsigned int edgessaved;                 // 0x28
} Dict_header;
```

binary output of the header:

```
0x00 ident       : _COMPILED_DICTIONARY_
0x16 unused 1    :      0 00000000
0x17 unused 2    :      0 00000000
0x18 root        :      9 00000009
0x1c words       :      3 00000003
0x20 edges used  :      9 00000009
0x24 nodes used  :      7 00000007
0x28 nodes saved :      1 00000001
0x2c edges saved :      1 00000001
```

The real array of data begins at offset 0x30. Integer are stored in a machine dependent way. This dictionary was compiled on an i386 and is not readable on a machine with a different endianess. The array is stored 'as is' right after the header. Each array cell is a bit-structure:

```
typedef struct _Dawg_edge {
   unsigned int ptr  : 24;
   unsigned int term : 1;
   unsigned int last : 1;
   unsigned int fill : 1;  // reserved (currently unused)
   unsigned int chr  : 5;
} Dawg_edge;
```

Characters are not stored in ASCII. The order is preserved but we changed the values: A=1, B=2, ... This is very easy to do with the ASCII table as `('A' & 0x1f) == ('a' & 0x1f) == 1`. This may not work on machines that are not using ASCII.

```
offs binary       structure
---- -------- |   ------------------
0x00 02000000 | 0 ptr= 0 t=0 l=1 f=0 chr=0 (‘)
0x04 1b000000 | 1 ptr= 0 t=1 l=1 f=0 chr=3 (c)
0x08 0b000000 | 2 ptr= 0 t=1 l=1 f=0 chr=1 (a)
0x0c 10000001 | 3 ptr= 1 t=0 l=0 f=0 chr=2 (b)
0x10 22000002 | 4 ptr= 2 t=0 l=1 f=0 chr=4 (d)
0x14 0a000002 | 5 ptr= 2 t=0 l=1 f=0 chr=1 (a)
0x18 22000005 | 6 ptr= 5 t=0 l=1 f=0 chr=4 (d)
0x1c 08000003 | 7 ptr= 3 t=0 l=0 f=0 chr=1 (a)
0x20 2a000006 | 8 ptr= 6 t=0 l=1 f=0 chr=5 (e)
0x24 00000007 | 9 ptr= 7 t=0 l=0 f=0 chr=0 (‘)
```

Strictly speaking, there is no node in the graph, only labelled edges.