Chapter 1

Linux基础命令

在终端用管理员权限添加用户的命令`$ sudo adduser [username]`, 创建用户名之后用命令`$ sudo passwd [username]`来分配用户密码.

1.1 Linux C

1.1.1 linux GCC/G++编译器与调试器

GCC是GNU的一个子项目, 最初作为C语言的编译器. 现在GCC能编译C, C++, Ada, Object C和Java等语言, 同时还可执行跨硬件平台的交叉编译.

安装GCC和G++的命令如下:

```
yum install make gcc gcc-c++
```

GCC/G++编译选项

Table 1.1: 常用GCC/G++编译选项

编译选项	说明
-c	只进行预处理, 编译和汇编, 生成.o文件
-S	只进行预处理和编译, 生成.s文件
-E	只进行预处理, 产生预处理后的结果到标准输出
-C	预处理时不删除注释信息, 常与-E同时使用
-o	指定目标名称, 常与-c, -S同时使用, 默认是.out
-include file	插入一个文件, 功能等同源代码中的#include
-Dmacro[=defval]	定义一个宏, 功能等同源代码中的#define macro [defval]
-Umacro	取消宏定义, 功能等同源代码中的#undef macro
-Idir	优先在选项后的目录中查找包含的头文件
-lname	链接后缀为.so的动态链接库来编译程序
-Ldir	指定编译搜索库的路径
-O[0-3]	编译器优化, 数值越大优化级别越高, 0没有优化
-g	编译器编译时加入编译信息
-pg	编译器加入信息给gprof
-share	使用动态库
-static	禁止使用动态库

1.1.2 程序和进程

程序是指一组指示计算机或其他具有信息处理能力设备每一步动作的指令. 进程是一个具有独立功能的程序关于某个数据集

合的一次可以并发执行的运行活动, 是处于活动状态的程序. 在Linux系统中, 用户创建进程时会先在系统的进程表中为进程创建独一无二的编码, 即PID.

1.1.3 GDB调试器

用`$ sudo yum install gdb`安装GDB调试器. GDB调试器调试的对象是可执行文件, 使用GCC或G++编译器编译源代码时, 必须加上选项-g才能使目标可执行文件包含被调试的信息. 如

```
gcc -g -o helloworld helloworld.c // 编译并连接程序, 使之包含可被调试信息
gdb helloworld // 使用GDB调试器打开 helloworld 可执行文件
```

Table 1.2: 常用GDB命令及解释

命令	解释
file [文件名]	在GDB中打开执行文件
break	设置断点, 支持的形式由break 行号, break 函数名, break 行号/函数名 if 条件
info	查看和可执行文件相关的各种信息
kill	终止正在调试的程序
print	显示变量或表达式的值
set args	设置调试程序的运行参数
delete	删除设置的断点或观测点
clear	删除设置在指定行号或函数上的断点
continue	从断点处继续执行程序
list	列出GDB中打开的可执行文件代码
watch	在程序中设置观测点
run	运行打开的可执行文件
next	单步执行程序
step	进入所调用的函数内部, 查看执行情况
whatis	查看变量或函数类型, 调用格式为“whatis 变量名/函数名”
ptype	显示数据结构定义情况
make	编译程序
quit	退出GDB

Chapter 2

BPNN

输入向量为$(x_1, x_2, \cdots, x_m)$, $m \to innode$为输入神经节点数. 训练数据为$X_1, X_2, \cdots, X_M$, 对于每个$X_i = (x_1^{(i)}, x_2^{(i)}, \cdots, x_m^{(i)})$是一个输入训练向量.

现设输入数据为$X = (x_1, x_2, \cdots, x_m)^T$, 隐藏层有$N = 1 \to hidelayer$层, 隐藏层的神经元数为$q \to hidenode$个. 设节点$i(1 \le i \le m)$和节点$j(1 \le j \le q)$之间的权值为$v_{ij}$, $v_j = (v_{1j}, v_{2j}, \cdots, v_{mj})$, $V = (v_1, v_2, \cdots, v_q)^T$, 节点$j$的阈值$b_j$, $b = (b_1, b_2, \cdots, b_q)$, 则隐藏层的第$j$个节点的纯输入值为

$$z_j' = \sum_{i=1}^{m} v_{ij} x_i + b_j = v_j X + b_j,$$

其实, 若设$Z' = (z_1', z_2', \cdots, z_q')^T$, 则上式可简写为$Z' = VX + b$. 而使用爱森斯坦因记号(作为张量分析的基本工具), 上式可简化为$z_j' = v_{ij} x_i + b_j$. 有时, 为方便表达, 可令$x_0 = 1$, $v_{0j} = b_j$, 而使上式简化为$z_j' = \sum_{i=0}^{m} v_{ij} x_i$. 从而$Z' = VX$.

激活函数用于将z_j'转化为隐藏层的输入数据$z_j = f_1(z_j')$, 记$Z = (z_1, z_2, \cdots, z_q)^T$, 并经过隐藏层到输出层的权矩阵得出隐藏层的正向纯输出$y_k' = w_{jk} z_j + c_k$, $(1 \le k \le n)$. 最后再用一个激活函数$f_2(\cdot)$将y_k'转化为输出数据

$$\hat{y}_k = f_2(y_k') = f_2\left(\sum_{j=1}^{q} w_{jk} z_j + c_k\right) = f_2(w_k Z + c_k).$$

最后根据输出数据$\hat{Y} = (\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n)^T$和训练数据的真实输出$Y = (y_1, y_2, \cdots, y_n)^T$比较误差, 使用平方误差

$$E = \frac{1}{2}\|\hat{Y} - Y\|_2^2 = \frac{1}{2}\sum_{k=1}^{n}(\hat{y}_k - y_k)^2.$$

现在假设输入的训练数据有P个, 分别为$X^{(1)}, X^{(2)}, \cdots, X^{(P)}$, 输入$X^{(p)}$对应输出$Y^{(p)}$, 神经网络的正向输出

$$\hat{Y}^{(p)} = f_2(WZ^{(p)} + c) = f_2(W f_1(VX^{(p)} + b) + c).$$

根据V, b的定义可类似的定义W和c, 记第p个样本的误差为E_p,

$$E_p = \frac{1}{2}\|Y^{(p)} - \hat{Y}^{(p)}\|_2^2 = \frac{1}{2}\sum_{k=1}^{n}\left(y_k^{(p)} - \hat{y}_k^{(p)}\right)^2.$$

于是对于每个给定的四元组(V, W, b, c), 都有一个对应的样本误差, $E_p = E_p(V, W, b, c)$, BPNN的主要思想是找出(V_0, W_0, b_0, c_0), 使

$$\begin{aligned} E(V_0, W_0, b_0, c_0) &= \min_{(V,W,b,c)} E(V, W, b, c) = \min_{(V,W,b,c)} \sum_{p=1}^{P} E_p(V, W, b, c) \\ &= \frac{1}{2}\min_{(V,W,b,c)} \sum_{p=1}^{P}\sum_{k=1}^{n}\left(y_k^{(p)} - \hat{y}_k^{(p)}\right)^2 = \frac{1}{2}\min_{(V,W,b,c)} \sum_{p=1}^{P}\sum_{k=1}^{n}\left(y_k^{(p)} - f_2(w_k f_1(VX^{(p)} + b) + c_k)\right)^2. \end{aligned}$$

由于$E(V, W, b, c)$为多维函数, 多元函数总是沿着负梯度方向递减, 所以可取(V, W, b, c)的修正项$\Delta(V, W, b, c)$为:

$$\Delta(V, W, b, c) = -\eta \frac{\nabla E}{\nabla(V, W, b, c)},$$

其中$\eta \to learningRate$, 为学习速率. 所以

$$\Delta v_{ij} = \eta \sum_{p=1}^{P} \sum_{k=1}^{n} (y_k{}^{(p)} - \hat{y}_k^{(p)}) f_2'(y_k'{}^{(p)}) w_{jk} \cdot f_1'(z_j'{}^{(p)}) \cdot x_i{}^{(p)}$$

$$\Delta w_{jk} = \eta \sum_{p=1}^{P} (y_k{}^{(p)} - \hat{y}_k^{(p)}) \cdot f_2'(y_k'{}^{(p)}) z_j{}^{(p)}$$

Chapter 3

加密

3.1 RSA加密

先选择两个大素数p, q, 并令$n = pq$, 则$\varphi(n) = (p-1)(q-1)$, 并取e使$(e, \varphi(n)) = 1$, 取d满足$ed \equiv 1 \pmod{\varphi(n)}$, 则对于任意的$m$, $m^{ed} \equiv m \pmod{n}$, 加密m为密文c的过程为

加密 $c \equiv m^e \pmod{n}$;

解密 $m \equiv c^d \pmod{n}$.

3.2 Okamoto-Uchiyama加密

取两个素数p, q, 让$n = p^2 q$, 取$g \in \mathbb{Z}_n^*$, 使得$g^{p-1} \not\equiv 1 \pmod{p^2}$, 让$h \equiv g^n \pmod{n}$, 则$(n, h, g)$为公钥, (p, q)为私钥. 则

加密 加密m为c, 任取$r \in \mathbb{Z}_n$, $c \equiv g^m h^r \pmod{n}$;

解密 定义$L(x) = \frac{x-1}{p}$, 其中$x \equiv 1 \pmod{p}$, 则$m = \frac{L(c^{p-1} \pmod{p^2})}{L(g^{p-1} \pmod{p^2})} \pmod{p}$.

解. 证明解密正确. $\mathbb{Z}_n^* \simeq \mathbb{Z}_{p^2}^* \times \mathbb{Z}_q^*$, $\mathbb{Z}_{p^2}^*$有唯一非平凡正规子群$H = \{x : x^p \equiv 1 \pmod{p}\}$, 然后证明

$$\{x^{p-1} \pmod{p^2} : x \in \mathbb{Z}_{p^2}^*\} = H,$$

$L : <H, \cdot> \to <\mathbb{Z}_p, +>$是同态映射, 由$c \equiv g^m h^r \pmod{n}$, 所以

$$c^{p-1} \equiv (g^{p-1})^m g^{p(p-1)rpq} \equiv (g^{p-1})^m \pmod{p^2}$$

所以$L((g^{p-1})^m) \equiv mL(g^{p-1}) \pmod{p}$, 所以$m = \frac{L(c^{p-1})}{L(g^{p-1})} \pmod{p}$. □

这是一个同态加密算法, 即若记$\varepsilon(m)$为明文m的密文, 则$\varepsilon(m_1)\varepsilon(m_2) = \varepsilon(m_1 + m_2)$.

Chapter 4

C++

4.1 C++画图

Linux下编译

code/plot0001.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

typedef struct {
    size_t width;
    size_t height;
    unsigned char *data;
} Image;

// 申请内存空间
static Image *image_new (size_t width, size_t height)
{
    Image *image;

    image = malloc (sizeof *image);
    image->width = width;
    image->height = height;
    image->data = malloc (width * height);

    return image;
}

// 释放内存
static void image_free (Image *image)
{
    free (image->data);
    free (image);
}

static void image_fill (Image *image, unsigned char value)
{
    memset (image->data, value, image->width * image->height);
}

/**
    * image_set_pixel:
    *
    * Sets a pixel passed in signed (x, y) coordinates, where (0,0) is at
```

```
41      * the center of the image.
42      **/
43  static void image_set_pixel (Image *image, ssize_t x, ssize_t y, unsigned char value)
44  {
45      size_t tx, ty;
46      unsigned char *p;
47
48      tx = (image->width / 2) + x;
49      ty = (image->height / 2) + y;
50
51      p = image->data + (ty * image->width) + tx;
52
53      *p = value;
54  }
55
56  static void image_save (const Image *image, const char *filename)
57  {
58      FILE *out;
59
60      out = fopen (filename, "wb");
61      if (!out)
62          return;
63
64      fprintf (out, "P5\n");
65      fprintf (out, "%zu %zu\n", image->width, image->height);
66      fprintf (out, "255\n");
67
68      fwrite (image->data, 1, image->width * image->height, out);
69
70      fclose (out);
71  }
72
73  static void draw_Taijitu(Image *image,int radius,int value)
74  {
75      int x,y;
76      int rlimit ,llimit;
77
78      int radius_2 = radius*radius;
79      for(y = -radius;y<radius;y++)
80          for(x= -radius;x<radius;x++)
81              if(x*x+y*y <= radius_2)
82                  image_set_pixel(image,x,y,0xff);
83
84      for(y = -radius;y<0;y++)
85          for(x = 0;x<radius;x++)
86              if((x*x)+(y*y) <= radius_2)
87                  image_set_pixel(image,x,y,value);
88
89      for(y = -radius;y<0;y++)
90          for(x = -(int)sqrt((double)(-radius*y-y*y));x<0;x++)
91              image_set_pixel(image,x,y,value);
92
93
94      for(y = 0;y<radius;y++)
95      {
96          llimit = (int)sqrt((double)(radius*y - y*y));
97          rlimit = (int)sqrt((double)(radius_2 - y*y));
98          for(x = llimit;x<rlimit;x++)
99              image_set_pixel(image,x,y,value);
100     }
101
102     for(y = 2*radius/6;y<4*radius/6;y++)
```

```
103     {
104         rlimit =(int) sqrt((double)(radius*y-y*y-2*radius_2/9));
105         llimit = -rlimit;
106 
107         for(x = llimit;x<rlimit;x++)
108             image_set_pixel(image,x,y,value);
109     }
110 
111     for(y = -4*radius/6;y<-2*radius/6;y++)
112     {
113         rlimit = sqrt(-radius*y-y*y-2*radius_2/9);
114         llimit = -rlimit;
115         for(x = llimit;x<rlimit;x++)
116             image_set_pixel(image,x,y,0xff);
117     }
118 
119     return ;
120 }
121 
122 int main (int argc, char *argv[])
123 {
124     Image *image;
125 
126     image = image_new (800, 800);
127 
128     image_fill (image, 0xaa);
129     draw_Taijitu (image, 300, 0);
130     image_save (image, "taiji_6.pgm");
131 
132     image_free (image);
133 
134     return 0;
135 }
```

报错`undefined reference to ‘sqrt’`, 编译时添加参数`$ gcc -lm src.cpp -o obj`.

4.2 cuckoo

Cuckoo算法

code/cuckoo/nvromTest.c

```c
/*
 * nvrom_test.c
 *
 *  Created on: Dec 10, 2016
 *      Author: math
 */


#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/stat.h>

#include "cuckoo_filter.h"
#include "mozilla-sha1/sha1.h"

int main(int argc, char **argv)
{
        SHA_CTX c;
        struct stat st;
        uint32_t key_num;
        uint8_t *keys;
        uint8_t **sha1_key;
        uint8_t value[DAT_LEN], *v;
        int bytes, i, j;
        FILE *f1, *f2;

        if (argc < 3) {
                fprintf(stderr, "usage: ./cuckoo_filter read_file write_file\n");
                exit(-1);
        }

        --argc;
        ++argv;

        f1 = fopen(argv[0], "rb");
        if (f1 == NULL) {
                fprintf(stderr, "Fail to open %s!\n", argv[0]);
                exit(-1);
        }
        stat(argv[0], &st);

        f2 = fopen(argv[1], "wb+");
        if (f2 == NULL) {
                fprintf(stderr, "Fail to open %s!\n", argv[1]);
                exit(-1);
        }

        /* Initialization */
        cuckoo_filter_init(st.st_size);

        /* Allocate SHA1 key space */
        key_num = next_pow_of_2(st.st_size) / DAT_LEN;
        keys = malloc(key_num * 20);
        sha1_key = malloc(key_num * sizeof(void *));
        if (!keys  !sha1_key) {
```

```
58                  fprintf(stderr, "Out of memory!\n");
59                  exit(-1);
60          }
61          for (i = 0; i < key_num; i++) {
62                  sha1_key[i] = keys + i * 20;
63          }
64
65          /* Put read_file into log on flash. */
66          i = 0;
67          do {
68                  memset(value, 0, DAT_LEN);
69                  bytes = fread(value, 1, DAT_LEN, f1);
70                  SHA1_Init(&c);
71                  SHA1_Update(&c, value, bytes);
72                  SHA1_Final(sha1_key[i], &c);
73                  cuckoo_filter_put(sha1_key[i], value);
74                  i++;
75          } while (bytes == DAT_LEN);
76
77          /* Real key number */
78          key_num = i;
79          printf("Total %u records.\n", key_num);
80
81          /* Deletion test */
82          for (i = 0; i < key_num; i += 2) {
83                  cuckoo_filter_put(sha1_key[i], NULL);
84          }
85
86          fseek(f1, 0, SEEK_SET);
87          for (i = 0; i < key_num; i++) {
88                  memset(value, 0, DAT_LEN);
89                  bytes = fread(value, 1, DAT_LEN, f1);
90                  if (!(i & 0x1)) {
91                          cuckoo_filter_put(sha1_key[i], value);
92                  }
93          }
94
95          /* Get logs on flash and write them into a new file. */
96          for (j = 0; j < key_num; j++) {
97                  v = cuckoo_filter_get(sha1_key[j]);
98                  if (v != NULL) {
99                          memcpy(value, v, DAT_LEN);
100                         fwrite(value, 1, DAT_LEN, f2);
101                 }
102         }
103
104         fclose(f1);
105         fclose(f2);
106
107         free(keys);
108         free(sha1_key);
109
110         return 0;
111 }
```

code/cuckoo/cuckooFilter.h

```
1  /*
2   * cuckoo_filter.h
3   *
4   *  Created on: Dec 10, 2016
```

```
 *      Author: math
 */
/*
 * Copyright (C) 2015, Leo Ma <begeekmyfriend@gmail.com>
 */
#ifndef SRC_CUCKOO_FILTER_H_
#define SRC_CUCKOO_FILTER_H_

//#define CUCKOO_DBG

/* Configuration */
#define SECTOR_SIZE    (1 << 5)
#define DAT_LEN        (SECTOR_SIZE - 20)  /* minus sha1 size */
#define ASSOC_WAY      (4)  /* 4-way association */
#define INVALID_OFFSET (-1)

/* Cuckoo hash */
#define force_align(addr, size)  ((void *)((((uintptr_t)(addr)) + (size) - 1) & ~((size) - 1)))
#define cuckoo_hash_lsb(key, count)  (((size_t *)(key))[0] & (count - 1))
#define cuckoo_hash_msb(key, count)  (((size_t *)(key))[1] & (count - 1))

/* Flash driver interfaces. */
#define flash_align(addr)  (!((uintptr_t)(addr) & (SECTOR_SIZE - 1)))
#define flash_read(addr)  (*(volatile uint8_t *)(addr))
#define flash_write(addr, byte)  (*(volatile uint8_t *)(addr) = (byte))
#define flash_sector_erase(addr) \
        do { \
                uint32_t __i; \
                volatile uint8_t *__addr = (volatile uint8_t *)(addr); \
                for (__i = 0; __i < SECTOR_SIZE; __i++) { \
                        *(volatile uint8_t *)__addr = 0xff; \
                        __addr++; \
                } \
        } while (0)

/* The log entries store key-value pairs on flash and
 * each entry is assumed just one sector size fit.
 */
struct log_entry {
        uint8_t sha1[20];
        uint8_t data[DAT_LEN];
};

enum { AVAILIBLE, OCCUPIED, DELETED, };

/* The in-memory hash buckets cache filter keys (which are assumed SHA1 values)
 * via cuckoo hashing function and map them to log entries stored on flash.
 */
struct hash_slot_cache {
        uint32_t tag : 30;  /* summary of key */
        uint32_t status : 2;  /* FSM */
        uint32_t offset;  /* offset on flash memory */
};

static inline int is_pow_of_2(uint32_t x)
{
        return !(x & (x-1));
}

static inline uint32_t next_pow_of_2(uint32_t x)
{
        if (is_pow_of_2(x))
```

```
67                  return x;
68          x = x>>1;
69          x = x>>2;
70          x = x>>4;
71          x = x>>8;
72          x |= x>>16;
73          return x + 1;
74  }
75
76  int cuckoo_filter_init(size_t size);
77  uint8_t *cuckoo_filter_get(uint8_t *key);
78  void cuckoo_filter_put(uint8_t *key, uint8_t *value);
79
80
81
82  #endif /* SRC_CUCKOO_FILTER_H_ */
```

code/cuckoo/cuckooFilter.c

```
1   /*
2    * cuckoo_filter.c
3    *
4    *  Created on: Dec 10, 2016
5    *      Author: math
6    */
7
8   /*
9    * Copyright (C) 2015, Leo Ma <begeekmyfriend@gmail.com>
10   */
11
12  #include <stdio.h>
13  #include <stdlib.h>
14  #include <stdint.h>
15  #include <string.h>
16  #include <assert.h>
17
18  #include "cuckoo_filter.h"
19
20  struct hash_table {
21          struct hash_slot_cache **buckets;
22          struct hash_slot_cache *slots;
23          uint32_t slot_num;
24          uint32_t bucket_num;
25  };
26
27  static uint8_t *nvrom_base_addr;
28  static uint32_t nvrom_size;
29  static uint32_t log_entries;
30  static struct hash_table hash_table;
31
32  static void dump_sha1_key(uint8_t *sha1)
33  {
34  #ifdef CUCKOO_DBG
35          int i;
36          static const char str[] = "0123456789abcdef";
37
38          printf("SHA1: ");
39          for (i = 19; i >= 0; i--) {
40                  putchar(str[sha1[i] >> 4]);
41                  putchar(str[sha1[i] & 0xf]);
42          }
```

```
43          putchar('\n');
44  #endif
45  }
46
47  static uint32_t next_entry_offset(void)
48  {
49          uint8_t *append_addr = nvrom_base_addr + log_entries * sizeof(struct log_entry);
50          assert(flash_align(append_addr));
51          if ((log_entries + 1) * sizeof(struct log_entry) >= nvrom_size) {
52                  return INVALID_OFFSET;
53          } else {
54                  return (uint32_t)(append_addr - nvrom_base_addr);
55          }
56  }
57
58  static void show_hash_slots(struct hash_table *table)
59  {
60  #ifdef CUCKOO_DBG
61          int i, j;
62
63          printf("List all keys in hash table (tag/status/offset):\n");
64          for (i = 0; i < table->bucket_num; i++) {
65                  printf("bucket[%04x]:", i);
66                  struct hash_slot_cache *slot = table->buckets[i];
67                  for (j = 0; j < ASSOC_WAY; j++) {
68                          printf("\t%04x/%x/%08x", slot[j].tag, slot[j].status, slot[j].offset);
69                  }
70                  printf("\n");
71          }
72  #endif
73  }
74
75  static uint8_t *key_verify(uint8_t *key, uint32_t offset)
76  {
77          int i;
78          uint8_t *read_addr = nvrom_base_addr + offset;
79          for (i = 0; i < 20; i++) {
80                  if (key[i] != flash_read(read_addr)) {
81                          return NULL;
82                  }
83                  read_addr++;
84          }
85          return read_addr;
86  }
87
88  static int cuckoo_hash_collide(struct hash_table *table, uint32_t *tag, uint32_t *p_offset)
89  {
90          int i, j, k, alt_cnt;
91          uint32_t old_tag[2], offset, old_offset;
92          struct hash_slot_cache *slot;
93
94          /* Kick out the old bucket and move it to the alternative bucket. */
95          offset = *p_offset;
96          slot = table->buckets[tag[0]];
97          old_tag[0] = tag[0];
98          old_tag[1] = slot[0].tag;
99          old_offset = slot[0].offset;
100         slot[0].tag = tag[1];
101         slot[0].offset = offset;
102         i = 0 ^ 1;
103         k = 0;
104         alt_cnt = 0;
```

```

KICK_OUT:
        slot = table->buckets[old_tag[i]];
        for (j = 0; j < ASSOC_WAY; j++) {
                if (offset == INVALID_OFFSET && slot[j].status == DELETED) {
                        slot[j].status = OCCUPIED;
                        slot[j].tag = old_tag[i ^ 1];
                        *p_offset = offset = slot[j].offset;
                        break;
                } else if (slot[j].status == AVAILIBLE) {
                        slot[j].status = OCCUPIED;
                        slot[j].tag = old_tag[i ^ 1];
                        slot[j].offset = old_offset;
                        break;
                }
        }

        if (j == ASSOC_WAY) {
                if (++alt_cnt > 512) {
                        if (k == ASSOC_WAY - 1) {
                                /* Hash table is almost full and needs to be resized */
                                return 1;
                        } else {
                                k++;
                        }
                }
                uint32_t tmp_tag = slot[k].tag;
                uint32_t tmp_offset = slot[k].offset;
                slot[k].tag = old_tag[i ^ 1];
                slot[k].offset = old_offset;
                old_tag[i ^ 1] = tmp_tag;
                old_offset = tmp_offset;
                i ^= 1;
                goto KICK_OUT;
        }

        return 0;
}

static int cuckoo_hash_get(struct hash_table *table, uint8_t *key, uint8_t **read_addr)
{
        int i, j;
        uint8_t *addr;
        uint32_t tag[2], offset;
        struct hash_slot_cache *slot;

        tag[0] = cuckoo_hash_lsb(key, table->bucket_num);
        tag[1] = cuckoo_hash_msb(key, table->bucket_num);

#ifdef CUCKOO_DBG
        printf("get t0:%x t1:%x\n", tag[0], tag[1]);
#endif
        dump_sha1_key(key);

        /* Filter the key and verify if it exists. */
        slot = table->buckets[tag[0]];
        for (i = 0; i < ASSOC_WAY; i++) {
                if (cuckoo_hash_msb(key, table->bucket_num) == slot[i].tag) {
                        if (slot[i].status == OCCUPIED) {
                                offset = slot[i].offset;
                                addr = key_verify(key, offset);
                                if (addr != NULL) {
```

```
                                        if (read_addr != NULL) {
                                                *read_addr = addr;
                                        }
                                        break;
                                }
                        } else if (slot[i].status == DELETED) {
#ifdef CUCKOO_DBG
                                printf("Key has been deleted!\n");
#endif
                                return DELETED;
                        }
                }
        }

        if (i == ASSOC_WAY) {
                slot = table->buckets[tag[1]];
                for (j = 0; j < ASSOC_WAY; j++) {
                        if (cuckoo_hash_lsb(key, table->bucket_num) == slot[j].tag) {
                                if (slot[j].status == OCCUPIED) {
                                        offset = slot[j].offset;
                                        addr = key_verify(key, offset);
                                        if (addr != NULL) {
                                                if (read_addr != NULL) {
                                                        *read_addr = addr;
                                                }
                                                break;
                                        }
                                } else if (slot[j].status == DELETED) {
#ifdef CUCKOO_DBG
                                        printf("Key has been deleted!\n");
#endif
                                        return DELETED;
                                }
                        }
                }
                if (j == ASSOC_WAY) {
#ifdef CUCKOO_DBG
                        printf("Key not exists!\n");
#endif
                        return AVAILIBLE;
                }
        }

        return OCCUPIED;
}

static int cuckoo_hash_put(struct hash_table *table, uint8_t *key, uint32_t *p_offset)
{
        int i, j;
        uint32_t tag[2], offset;
        struct hash_slot_cache *slot;

        tag[0] = cuckoo_hash_lsb(key, table->bucket_num);
        tag[1] = cuckoo_hash_msb(key, table->bucket_num);

#ifdef CUCKOO_DBG
        printf("put offset:%x t0:%x t1:%x\n", *p_offset, tag[0], tag[1]);
#endif

        /* Insert new key into hash buckets. */
        offset = *p_offset;
        slot = table->buckets[tag[0]];
```

```
229         for (i = 0; i < ASSOC_WAY; i++) {
230                 if (offset == INVALID_OFFSET && slot[i].status == DELETED) {
231                         slot[i].status = OCCUPIED;
232                         slot[i].tag = cuckoo_hash_msb(key, table->bucket_num);
233                         *p_offset = offset = slot[i].offset;
234                         break;
235                 } else if (slot[i].status == AVAILIBLE) {
236                         slot[i].status = OCCUPIED;
237                         slot[i].tag = cuckoo_hash_msb(key, table->bucket_num);
238                         slot[i].offset = offset;
239                         break;
240                 }
241         }
242 
243         if (i == ASSOC_WAY) {
244                 slot = table->buckets[tag[1]];
245                 for (j = 0; j < ASSOC_WAY; j++) {
246                         if (offset == INVALID_OFFSET && slot[j].status == DELETED) {
247                                 slot[j].status = OCCUPIED;
248                                 slot[j].tag = cuckoo_hash_lsb(key, table->bucket_num);
249                                 *p_offset = offset = slot[j].offset;
250                                 break;
251                         } else if (slot[j].status == AVAILIBLE) {
252                                 slot[j].status = OCCUPIED;
253                                 slot[j].tag = cuckoo_hash_lsb(key, table->bucket_num);
254                                 slot[j].offset = offset;
255                                 break;
256                         }
257                 }
258 
259                 if (j == ASSOC_WAY) {
260                         if (cuckoo_hash_collide(table, tag, p_offset)) {
261 #ifdef CUCKOO_DBG
262                                 printf("Hash table collision!\n");
263 #endif
264                                 return -1;
265                         }
266                 }
267         }
268 
269         show_hash_slots(table);
270 
271         return 0;
272 }
273 
274 static void cuckoo_hash_status_set(struct hash_table *table, uint8_t *key, int status)
275 {
276         uint32_t i, j, tag[2];
277         struct hash_slot_cache *slot;
278 
279         tag[0] = cuckoo_hash_lsb(key, table->bucket_num);
280         tag[1] = cuckoo_hash_msb(key, table->bucket_num);
281 
282 #ifdef CUCKOO_DBG
283         printf("set status:%d t0:%x t1:%x\n", status, tag[0], tag[1]);
284 #endif
285         dump_sha1_key(key);
286 
287         /* Insert new key into hash buckets. */
288         slot = table->buckets[tag[0]];
289         for (i = 0; i < ASSOC_WAY; i++) {
290                 if (cuckoo_hash_msb(key, table->bucket_num) == slot[i].tag) {
```

```
291                         slot[i].status = status;
292                         return;
293                 }
294         }
295
296         if (i == ASSOC_WAY) {
297                 slot = table->buckets[tag[1]];
298                 for (j = 0; j < ASSOC_WAY; j++) {
299                         if (cuckoo_hash_lsb(key, table->bucket_num) == slot[j].tag) {
300                                 slot[j].status = status;
301                                 return;
302                         }
303                 }
304
305                 if (j == ASSOC_WAY) {
306 #ifdef CUCKOO_DBG
307                         printf("Key not exists!\n");
308 #endif
309                 }
310         }
311 }
312
313 static void cuckoo_hash_delete(struct hash_table *table, uint8_t *key)
314 {
315         cuckoo_hash_status_set(table, key, DELETED);
316 }
317
318 static void cuckoo_hash_recover(struct hash_table *table, uint8_t *key)
319 {
320         cuckoo_hash_status_set(table, key, OCCUPIED);
321 }
322
323 static void cuckoo_rehash(struct hash_table *table)
324 {
325         int i;
326         struct hash_table old_table;
327
328         /* Reallocate hash slots */
329         old_table.slots = table->slots;
330         old_table.slot_num = table->slot_num;
331         table->slot_num *= 2;
332         table->slots = calloc(table->slot_num, sizeof(struct hash_slot_cache));
333         if (table->slots == NULL) {
334                 table->slots = old_table.slots;
335                 return;
336         }
337
338         /* Reallocate hash buckets associated with slots */
339         old_table.buckets = table->buckets;
340         old_table.bucket_num = table->bucket_num;
341         table->bucket_num *= 2;
342         table->buckets = malloc(table->bucket_num * sizeof(struct hash_slot_cache *));
343         if (table->buckets == NULL) {
344                 free(table->slots);
345                 table->slots = old_table.slots;
346                 table->buckets = old_table.buckets;
347                 return;
348         }
349         for (i = 0; i < table->bucket_num; i++) {
350                 table->buckets[i] = &table->slots[i * ASSOC_WAY];
351         }
352
```

```
353         /* Rehash all hash slots */
354         uint8_t *read_addr = nvrom_base_addr;
355         uint32_t entries = log_entries;
356         while (entries--) {
357                 uint8_t key[20];
358                 uint32_t offset = read_addr - nvrom_base_addr;
359                 for (i = 0; i < 20; i++) {
360                         key[i] = flash_read(read_addr);
361                         read_addr++;
362                 }
363                 /* Duplicated keys in hash table which can cause eternal
364                  * hashing collision! Be careful of that!
365                  */
366                 assert(!cuckoo_hash_put(table, key, &offset));
367                 if (cuckoo_hash_get(&old_table, key, NULL) == DELETED) {
368                         cuckoo_hash_delete(table, key);
369                 }
370                 read_addr += DAT_LEN;
371         }
372
373        free(old_table.slots);
374        free(old_table.buckets);
375 }
376
377 uint8_t *cuckoo_filter_get(uint8_t *key)
378 {
379         int i;
380         uint8_t *read_addr;
381         static uint8_t value[DAT_LEN];
382
383         /* Read data from the log entry on flash. */
384         if (cuckoo_hash_get(&hash_table, key, &read_addr) != OCCUPIED) {
385                 return NULL;
386         }
387
388         for (i = 0; i < DAT_LEN; i++) {
389                 value[i] = flash_read(read_addr);
390                 read_addr++;
391         }
392
393         return value;
394 }
395
396 void cuckoo_filter_put(uint8_t *key, uint8_t *value)
397 {
398         if (value != NULL) {
399                 /* Important: Reject duplicated keys keeping from eternal collision */
400                 int status = cuckoo_hash_get(&hash_table, key, NULL);
401                 if (status == OCCUPIED) {
402                         return;
403                 } else if (status == DELETED) {
404                         cuckoo_hash_recover(&hash_table, key);
405                 } else {
406                         /* Find new log entry offset on flash. */
407                         uint32_t offset = next_entry_offset();
408
409                         /* Insert into hash slots */
410                         if (cuckoo_hash_put(&hash_table, key, &offset) == -1) {
411                                 cuckoo_rehash(&hash_table);
412                                 cuckoo_hash_put(&hash_table, key, &offset);
413                         }
414                         if (offset == -1) {
```

```
415                                 fprintf(stderr, "Not enough capacity!\n");
416                                 return;
417                         }
418
419                         /* Add new entry of key-value pair on flash. */
420                         int i;
421                         uint8_t *append_addr = nvrom_base_addr + offset;
422                         assert(flash_align(append_addr));
423                         flash_sector_erase(append_addr);
424                         for (i = 0; i < 20; i++) {
425                                 flash_write(append_addr, key[i]);
426                                 append_addr++;
427                         }
428                         for (i = 0; i < DAT_LEN; i++) {
429                                 flash_write(append_addr, value[i]);
430                                 append_addr++;
431                         }
432                         log_entries++;
433                 }
434         } else {
435                 /* Delete at the hash slot */
436                 cuckoo_hash_delete(&hash_table, key);
437         }
438 }
439
440 int cuckoo_filter_init(size_t size)
441 {
442         int i;
443
444         /* Make whole memory space large enough(but not always predictable...) */
445         nvrom_size = next_pow_of_2((size / DAT_LEN + 1) * SECTOR_SIZE);
446         nvrom_base_addr = malloc(nvrom_size + SECTOR_SIZE);
447         if (nvrom_base_addr == NULL) {
448                 return -1;
449         }
450         nvrom_base_addr = force_align(nvrom_base_addr, SECTOR_SIZE);
451
452         /* Allocate hash slots */
453         hash_table.slot_num = nvrom_size / SECTOR_SIZE;
454         /* Make rehashing happen */
455         hash_table.slot_num /= 4;
456         hash_table.slots = calloc(hash_table.slot_num, sizeof(struct hash_slot_cache));
457         if (hash_table.slots == NULL) {
458                 return -1;
459         }
460
461         /* Allocate hash buckets associated with slots */
462         hash_table.bucket_num = hash_table.slot_num / ASSOC_WAY;
463         hash_table.buckets = malloc(hash_table.bucket_num * sizeof(struct hash_slot_cache *));
464         if (hash_table.buckets == NULL) {
465                 free(hash_table.slots);
466                 return -1;
467         }
468         for (i = 0; i < hash_table.bucket_num; i++) {
469                 hash_table.buckets[i] = &hash_table.slots[i * ASSOC_WAY];
470         }
471
472         return 0;
473 }
```

code/cuckoo/mozilla-sha1/sha1.h

```
/*
 * sha1.h
 *
 *  Created on: Dec 10, 2016
 *      Author: math
 */
/*
 * The contents of this file are subject to the Mozilla Public
 * License Version 1.1 (the "License"); you may not use this file
 * except in compliance with the License. You may obtain a copy of
 * the License at http://www.mozilla.org/MPL/
 *
 * Software distributed under the License is distributed on an "AS
 * IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * rights and limitations under the License.
 *
 * The Original Code is SHA 180-1 Header File
 *
 * The Initial Developer of the Original Code is Paul Kocher of
 * Cryptography Research.  Portions created by Paul Kocher are
 * Copyright (C) 1995-9 by Cryptography Research, Inc.  All
 * Rights Reserved.
 *
 * Contributor(s):
 *
 *     Paul Kocher
 *
 * Alternatively, the contents of this file may be used under the
 * terms of the GNU General Public License Version 2 or later (the
 * "GPL"), in which case the provisions of the GPL are applicable
 * instead of those above.  If you wish to allow use of your
 * version of this file only under the terms of the GPL and not to
 * allow others to use your version of this file under the MPL,
 * indicate your decision by deleting the provisions above and
 * replace them with the notice and other provisions required by
 * the GPL.  If you do not delete the provisions above, a recipient
 * may use your version of this file under either the MPL or the
 * GPL.
 */
#ifndef SRC_MOZILLA_SHA1_SHA1_H_
#define SRC_MOZILLA_SHA1_SHA1_H_

typedef struct {
  unsigned int H[5];
  unsigned int W[80];
  int lenW;
  unsigned int sizeHi,sizeLo;
} SHA_CTX;

void SHA1_Init(SHA_CTX *ctx);
void SHA1_Update(SHA_CTX *ctx, void *dataIn, int len);
void SHA1_Final(unsigned char hashout[20], SHA_CTX *ctx);




#endif /* SRC_MOZILLA_SHA1_SHA1_H_ */
```

code/cuckoo/mozilla-sha1/sha1.c

```c
/*
 * sha1.c
 *
 *  Created on: Dec 10, 2016
 *      Author: math
 */

/*
 * The contents of this file are subject to the Mozilla Public
 * License Version 1.1 (the "License"); you may not use this file
 * except in compliance with the License. You may obtain a copy of
 * the License at http://www.mozilla.org/MPL/
 *
 * Software distributed under the License is distributed on an "AS
 * IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * rights and limitations under the License.
 *
 * The Original Code is SHA 180-1 Reference Implementation (Compact version)
 *
 * The Initial Developer of the Original Code is Paul Kocher of
 * Cryptography Research.  Portions created by Paul Kocher are
 * Copyright (C) 1995-9 by Cryptography Research, Inc.  All
 * Rights Reserved.
 *
 * Contributor(s):
 *
 *     Paul Kocher
 *
 * Alternatively, the contents of this file may be used under the
 * terms of the GNU General Public License Version 2 or later (the
 * "GPL"), in which case the provisions of the GPL are applicable
 * instead of those above.  If you wish to allow use of your
 * version of this file only under the terms of the GPL and not to
 * allow others to use your version of this file under the MPL,
 * indicate your decision by deleting the provisions above and
 * replace them with the notice and other provisions required by
 * the GPL.  If you do not delete the provisions above, a recipient
 * may use your version of this file under either the MPL or the
 * GPL.
 */

#include "sha1.h"

static void shaHashBlock(SHA_CTX *ctx);

void SHA1_Init(SHA_CTX *ctx) {
  int i;

  ctx->lenW = 0;
  ctx->sizeHi = ctx->sizeLo = 0;

  /* Initialize H with the magic constants (see FIPS180 for constants)
   */
  ctx->H[0] = 0x67452301;
  ctx->H[1] = 0xefcdab89;
  ctx->H[2] = 0x98badcfe;
  ctx->H[3] = 0x10325476;
  ctx->H[4] = 0xc3d2e1f0;

```

```
  for (i = 0; i < 80; i++)
    ctx->W[i] = 0;
}


void SHA1_Update(SHA_CTX *ctx, void *_dataIn, int len) {
  unsigned char *dataIn = _dataIn;
  int i;

  /* Read the data into W and process blocks as they get full
   */
  for (i = 0; i < len; i++) {
    ctx->W[ctx->lenW / 4] <<= 8;
    ctx->W[ctx->lenW / 4] |= (unsigned int)dataIn[i];
    if ((++ctx->lenW) % 64 == 0) {
      shaHashBlock(ctx);
      ctx->lenW = 0;
    }
    ctx->sizeLo += 8;
    ctx->sizeHi += (ctx->sizeLo < 8);
  }
}


void SHA1_Final(unsigned char hashout[20], SHA_CTX *ctx) {
  unsigned char pad0x80 = 0x80;
  unsigned char pad0x00 = 0x00;
  unsigned char padlen[8];
  int i;

  /* Pad with a binary 1 (e.g. 0x80), then zeroes, then length
   */
  padlen[0] = (unsigned char)((ctx->sizeHi >> 24) & 255);
  padlen[1] = (unsigned char)((ctx->sizeHi >> 16) & 255);
  padlen[2] = (unsigned char)((ctx->sizeHi >> 8) & 255);
  padlen[3] = (unsigned char)((ctx->sizeHi >> 0) & 255);
  padlen[4] = (unsigned char)((ctx->sizeLo >> 24) & 255);
  padlen[5] = (unsigned char)((ctx->sizeLo >> 16) & 255);
  padlen[6] = (unsigned char)((ctx->sizeLo >> 8) & 255);
  padlen[7] = (unsigned char)((ctx->sizeLo >> 0) & 255);
  SHA1_Update(ctx, &pad0x80, 1);
  while (ctx->lenW != 56)
    SHA1_Update(ctx, &pad0x00, 1);
  SHA1_Update(ctx, padlen, 8);

  /* Output hash
   */
  for (i = 0; i < 20; i++) {
    hashout[i] = (unsigned char)(ctx->H[i / 4] >> 24);
    ctx->H[i / 4] <<= 8;
  }

  /*
   *  Re-initialize the context (also zeroizes contents)
   */
  SHA1_Init(ctx);
}


#define SHA_ROT(X,n) (((X) << (n)) | ((X) >> (32-(n))))

static void shaHashBlock(SHA_CTX *ctx) {
```

```
123   int t;
124   unsigned int A,B,C,D,E,TEMP;
125
126   for (t = 16; t <= 79; t++)
127     ctx->W[t] =
128       SHA_ROT(ctx->W[t-3] ^ ctx->W[t-8] ^ ctx->W[t-14] ^ ctx->W[t-16], 1);
129
130   A = ctx->H[0];
131   B = ctx->H[1];
132   C = ctx->H[2];
133   D = ctx->H[3];
134   E = ctx->H[4];
135
136   for (t = 0; t <= 19; t++) {
137     TEMP = SHA_ROT(A,5) + (((C^D)&B)^D)     + E + ctx->W[t] + 0x5a827999;
138     E = D; D = C; C = SHA_ROT(B, 30); B = A; A = TEMP;
139   }
140   for (t = 20; t <= 39; t++) {
141     TEMP = SHA_ROT(A,5) + (B^C^D)           + E + ctx->W[t] + 0x6ed9eba1;
142     E = D; D = C; C = SHA_ROT(B, 30); B = A; A = TEMP;
143   }
144   for (t = 40; t <= 59; t++) {
145     TEMP = SHA_ROT(A,5) + ((B&C)(D&(BC))) + E + ctx->W[t] + 0x8f1bbcdc;
146     E = D; D = C; C = SHA_ROT(B, 30); B = A; A = TEMP;
147   }
148   for (t = 60; t <= 79; t++) {
149     TEMP = SHA_ROT(A,5) + (B^C^D)           + E + ctx->W[t] + 0xca62c1d6;
150     E = D; D = C; C = SHA_ROT(B, 30); B = A; A = TEMP;
151   }
152
153   ctx->H[0] += A;
154   ctx->H[1] += B;
155   ctx->H[2] += C;
156   ctx->H[3] += D;
157   ctx->H[4] += E;
158 }
```

Chapter 5

Java

5.1 命令行java

编辑文件

code/test0001.java

```
1 public class test{
2         public static void main(String args[]){
3                 System.out.println("A new jdk test!");
4         }
5 }
```

然后在终端执行`$ javac test0001.java`生成class文件test0001.class. 然后在终端执行`$ java test0001`便可执行class文件了.

5.2 Java计算日期间的天数

code/date0001.java

```java
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class test16 {

    /**
     * @param args
     * @throws ParseException
     */
    public static void main(String[] args) throws ParseException {
        // TODO Auto-generated method stub
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date d1=sdf.parse("2012-09-08 10:10:10");
        Date d2=sdf.parse("2012-09-15 00:00:00");
        System.out.println(daysBetween(d1,d2));

        System.out.println(daysBetween("2012-09-08 10:10:10","2012-09-15 00:00:00"));
    }

    /**
     * 计算两个日期之间相差的天数
     * @param smdate 较小的时间
     * @param bdate  较大的时间
     * @return 相差天数
     * @throws ParseException
     */
    public static int daysBetween(Date smdate,Date bdate) throws ParseException
    {
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        smdate=sdf.parse(sdf.format(smdate));
        bdate=sdf.parse(sdf.format(bdate));
        Calendar cal = Calendar.getInstance();
        cal.setTime(smdate);
        long time1 = cal.getTimeInMillis();
        cal.setTime(bdate);
        long time2 = cal.getTimeInMillis();
        long between_days=(time2-time1)/(1000*3600*24);

        return Integer.parseInt(String.valueOf(between_days));
    }

    /**
    *字符串的日期格式的计算
    */
    public static int daysBetween(String smdate,String bdate) throws ParseException{
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        Calendar cal = Calendar.getInstance();
        cal.setTime(sdf.parse(smdate));
        long time1 = cal.getTimeInMillis();
        cal.setTime(sdf.parse(bdate));
        long time2 = cal.getTimeInMillis();
        long between_days=(time2-time1)/(1000*3600*24);

        return Integer.parseInt(String.valueOf(between_days));
    }

```

```
59 }
```

5.3 Bloom Filter

There's a whole theory on good hash functions that are close to random in suitable ways. what is random hash function? Hashing themes repeat? Why Bloom Filters Are Not Taught in Algorithms 101?

Index