Tu-An Nguyen
tunhnguy@ucsc.edu
03/14/2021

CSE 13S Winter 2021
Assignment 7: Lempel-Ziv Compression
Design Document

Purpose

The purpose of this assignment is to create two programs, called `encode` and `decode`, which perform LZ78 compression and decompression, respectively. The `encode` program can compress any file, text, or binary, and the `decode` program can decompress any file, text, or binary that was compressed with `encode`.

Both programs operate on both little and big endian systems, use variable bit-length codes, and perform read and writes in efficient blocks of 4KB.

Pseudocode

encode.c

See Assignment 7 specifications.

decode.c

See Assignment 7 specifications.

trie.c

```
struct TrieNode:
      TrieNode *children[ALPHABET]
      u16 code

TrieNode *trie_node_create(u16 code):
      TrieNode tn = allocate memory of size(TrieNode)

      if tn:

            tn->code = code
```

```
            for i in [0, ALPHABET):
                  tn->children[i] = NULL

      else:
            free(tn)
            tn = NULL

      return tn

void trie_node_delete(TrieNode *n):
      free(tn)
      tn = NULL

TrieNode *trie_create(void):
      TrieNode *tn =  trie_node_create(EMPTY_CODE)

      if tn:
            return tn

      return NULL

void trie_reset(TrieNode *root):
      for i in [0, ALPHABET):
            if root->children[i]:
                  trie_delete(root->children[i])

void trie_delete(TrieNode *n):
      for i in [0, ALPHABET):
            if n->children[i]:
                  trie_delete(n->children[i])
                  n->children[i] = NULL

      trie_node_delete(n)

TrieNode *trie_step(TrieNode *n, u8 sym):
      return n->children[sym]
```

word.c

```
struct Word:
      u8 *syms
      u32 len
```

```
typedef Word *WordTable

Word *word_create(u8 *syms, u32 len):
      Word *w = allocate memory of size(Word)

      if w:
            w->syms = syms
            w->len = len

      else:
            free(w)
            w = NULL

      return w

Word *word_append_sym(Word *w, u8 sym):
      Word *x = word_create(w->syms, w->len + 1)

      x->syms[w->len] = sym   // appends sym at the end of the array

void word_delete(Word *w):
      free(w)
      w = NULL

WordTable *wt_create(void):
      u8 *syms = 0
      Word *w = word_create(syms, 0)

      WordTable *wt[MAX_CODE] = w

void wt_reset(WordTable *wt):
      for i in [1, MAX_CODE):
            word_delete(wt[i])
```

io.c

```
static u8 bit_buffer[BLOCK]
static int bit_index = 0
static u8 sym_buffer[BLOCK]
static int sym_index = 0

struct FileHeader:
      u32 magic
```

```
      u16 protection

int read_bytes(int infile, uint8_t *buf, int to_read):
      int total = 0
      int bytes = 0

      while to_read !=  0 and bytes = read(infile, buf, to_read) > 0:
            total += bytes
            to_read -= bytes
            buf += bytes

      return total

int write_bytes(int outfile, uint8_t *buf, int to_write):
      int total = 0
      int bytes = 0

      while to_write != 0 and bytes = write(outfile, buf, to_write) > 0:
            total += bytes
            to_write -= bytes
            buf += bytes

      return total

void read_header(int infile, FileHeader *header):
      if header->magic = MAGIC_NUM:

            read_bytes(infile, header, sizeof(FileHeader))

            if big endian:
                  swap endianness of header->magic
                  swap endianness of header->protection

      else:
            print error

void write_header(int outfile, FileHeader *header):
      write_bytes(outfile, header, sizeof(FileHeader))

      if big endian:
            swap endianness of header->magic
            swap endianness of header->protection
```

```
bool read_sym(int infile, u8 *sym):
      int index = 0
      int n = 0
      int buffer_end = 0

      if index == 0:
            n = read_bytes(infile, sym, BLOCK)

      *sym = sym[index]
      index += 1

      if index == BLOCK:
            index = 0

      if n < BLOCK:
            int buffer_end = n

      if buffer_end == index:
            return false

      return true

void write_pair(int outfile, u16 code, u8 sym, int bitlen):
      for i in [0, bitlen):
            if ith bit of code is set:
                  set bit_buffer[bit_index / 8]’s (bit_index % 8 - 1)’s bit

            else:
                  clear bit_buffer[bit_index / 8]’s (bit_index % 8 - 1)’s bit

            if bit_index == BLOCK * 8:
                  write_bytes(outfile, bit_buffer, BLOCK)

            bit_index += 1

      for i in [0, 8):
            if ith bit of sym is set:
                  set sym_buffer[sym_index / 8]’s (sym_index % 8 - 1)’s bit

            else:
                  clear sym_buffer[sym_index / 8]’s (sym_index % 8 - 1)’s bit

            if sym_index == BLOCK * 8:
```

```
                write_bytes(outfile, sym_buffer, BLOCK)

            sym_index += 1

int bytes(int bits):
      if bits / 8 == 0:
            returns bits / 8
      return bits / 8 + 1

void flush_pairs(int outfile):
      if bit_index != 0: // if there are still bits in the bit_buffer
            write_bytes(outfile, bit_buffer, bytes(bit_index))

      bit_index = 0

      if sym_index != 0: // if there are still bits in the sym_buffer
            write_bytes(outfile, sym_buffer, bytes(bit_index))

      sym_index = 0

bool read_pair(int infile, u16 *code, u8 *sym, int bitlen):
      for i in [0, bitlen):
            if bit_index == 0: // if bit_buffer is empty
                  read_bytes(infile, bit_buffer, BLOCK)

            if bit at bit_buffer[bit_index / 8]'s (bit_index % 8 - 1) == 1:
                  set bit at i in code

            else:
                  clear bit at i in code

            bit_index += 1

            if sym_index == BLOCK * 8:
                  sym_index = 0

      for i in [0, 8):
            if sym_index == 0: // if sym_buffer is empty
                  read_bytes(infile, sym_buffer, BLOCK)

            if bit at sym_buffer[sym_index / 8]'s (sym_index % 8 - 1) == 1:
                  set bit at i in sym
```

```
            else:
                  clear bit at i in sym

            sym_index += 1

            if sym_index == BLOCK * 8:
                  sym_index = 0

      if code != STOP_CODE:
            return true

      return false

void write_word(int outfile, Word *w):
      for sym in word:
            sym_buffer[sym_index] = sym
            sym_index += 1

            if sym_index == BLOCK * 8:
                  write_bytes(outfile, sym_buffer, BLOCK)

void flush_words(int outfile):
      if sym_index != 0:
            write_bytes(sym_buffer, sym_index)
```