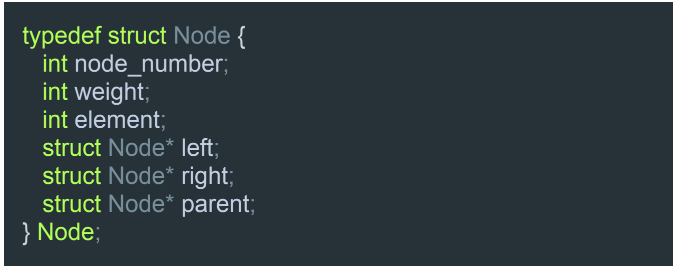
VIZ

Adaptive Huffman (FGK) compressor

VItali Denys & Spo**Z**io Cristian

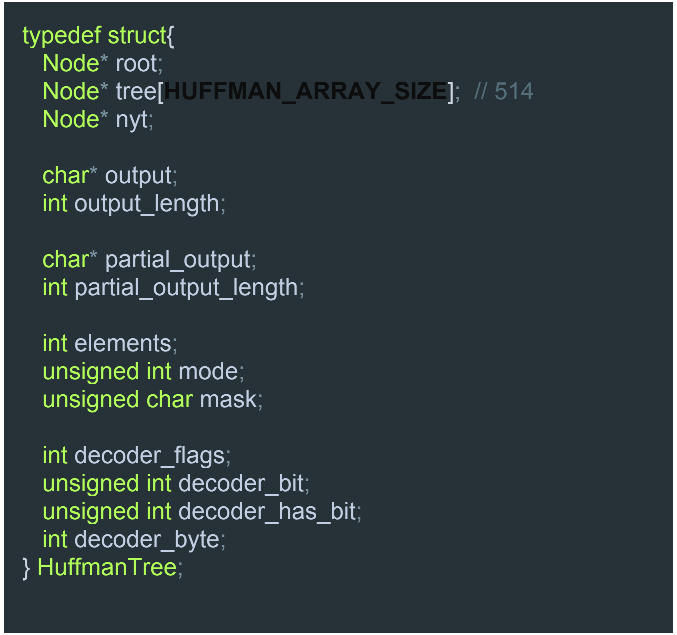
Strutture dati

Node ed Huffman Tree



```
typedef struct Node {
   int node_number;
   int weight;
   int element;
   struct Node* left;
   struct Node* right;
   struct Node* parent;
} Node;
```

Node

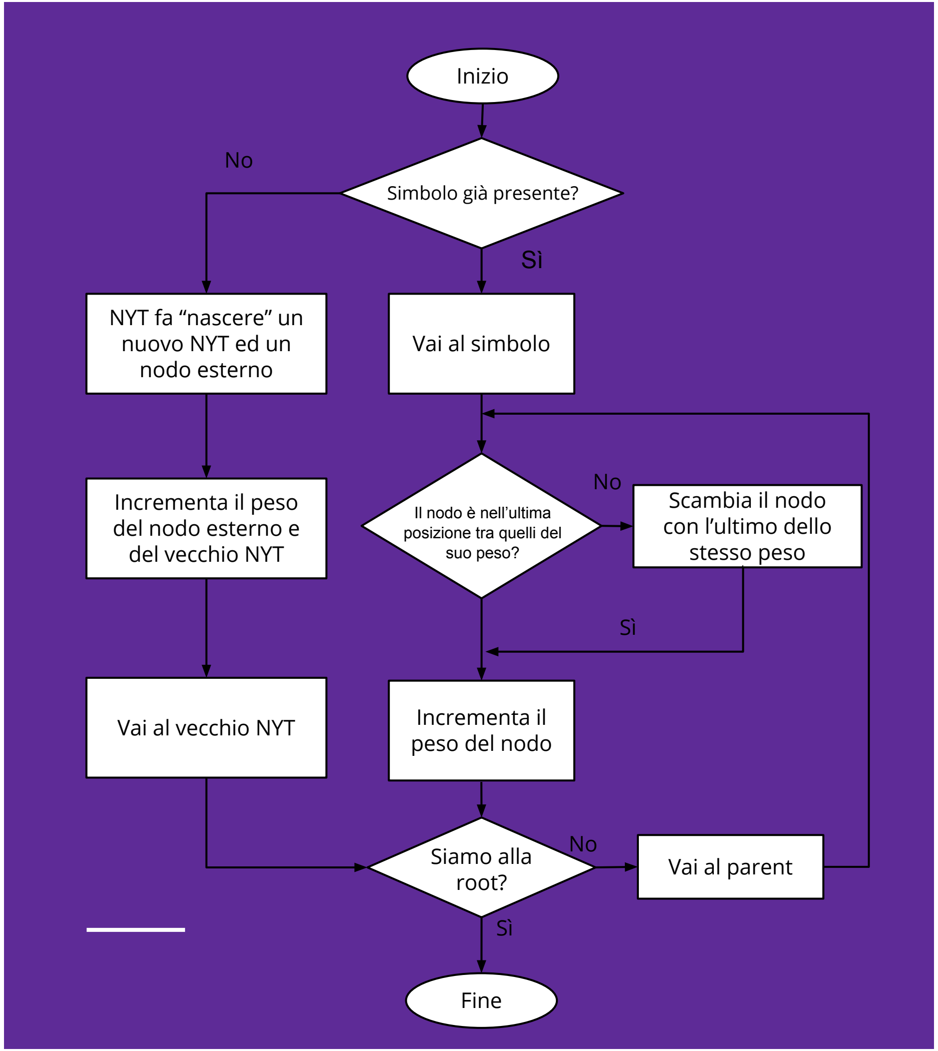


```
typedef struct{
   Node* root;
   Node* tree[HUFFMAN_ARRAY_SIZE];  // 514
   Node* nyt;

   char* output;
   int output_length;

   char* partial_output;
   int partial_output_length;

   int elements;
   unsigned int mode;
   unsigned char mask;

   int decoder_flags;
   unsigned int decoder_bit;
   unsigned int decoder_has_bit;
   int decoder_byte;
} HuffmanTree;
```

Huffman Tree

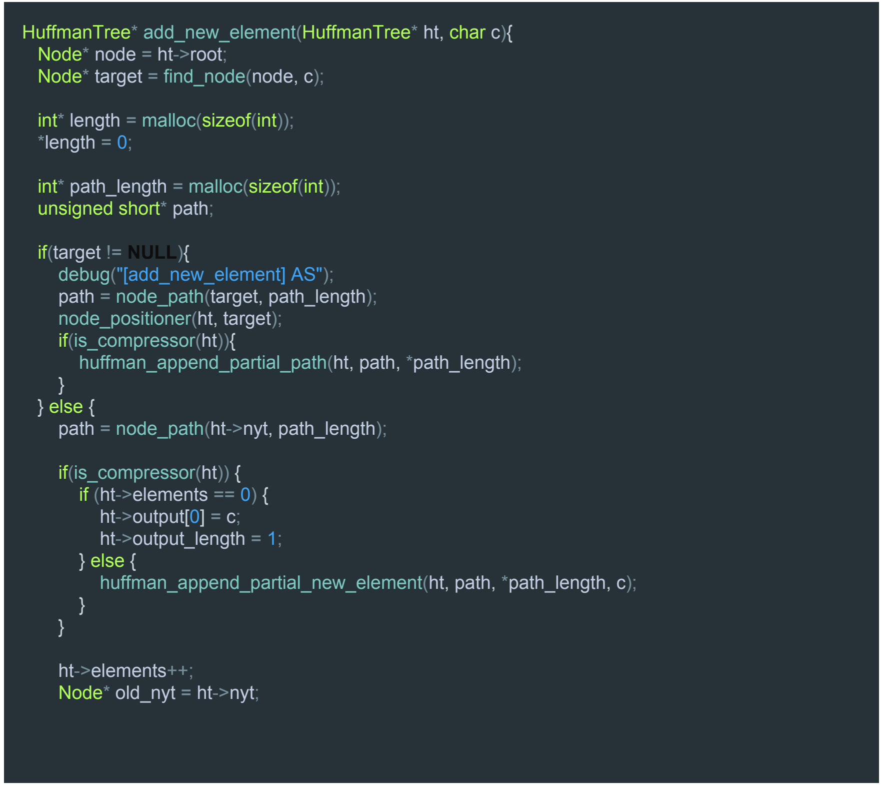
Aggiornamento dei nodi

Diagramma di flusso

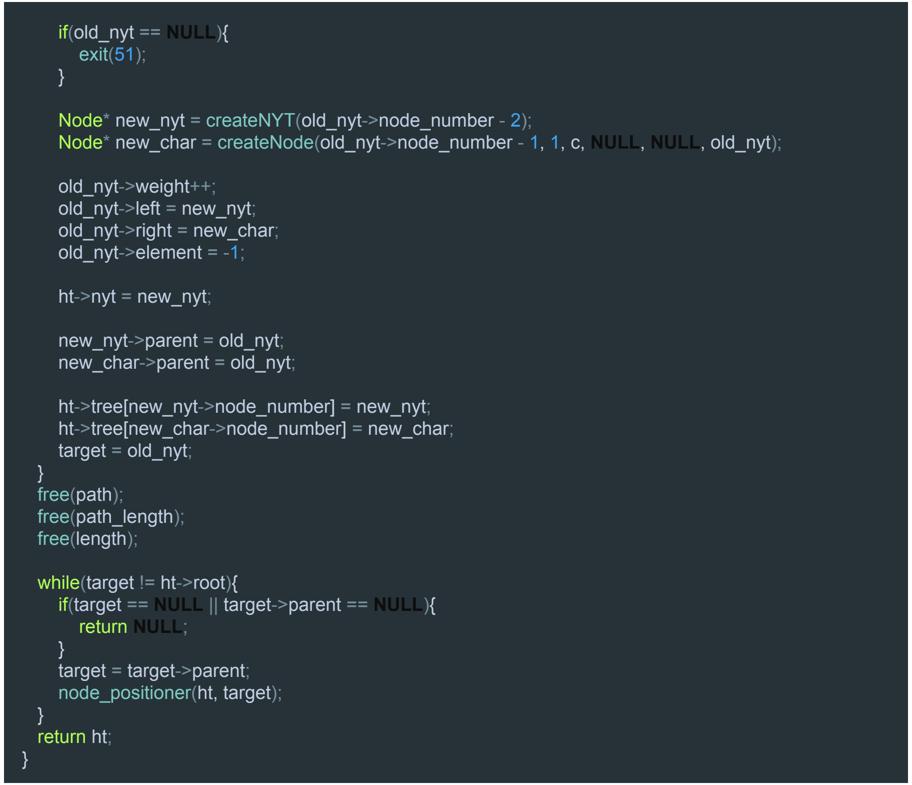


Inizio
Simbolo già presente?
No
Sì
NYT fa “nascere” un nuovo NYT ed un nodo esterno
Vai al simbolo
Incrementa il peso del nodo esterno e del vecchio NYT
Il nodo è nell’ultima posizione tra quelli del suo peso?
No
Scambia il nodo con l’ultimo dello stesso peso
Sì
Vai al vecchio NYT
Incrementa il peso del nodo
Siamo alla root?
No
Vai al parent
Sì
Fine

Implementazione dell’aggiornamento

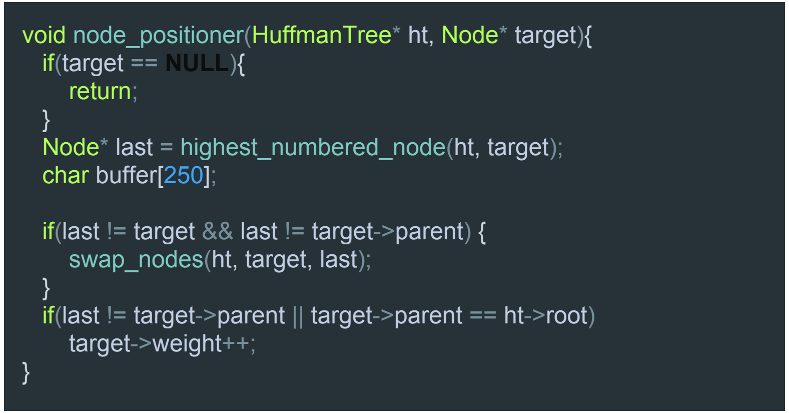


```
HuffmanTree* add_new_element(HuffmanTree* ht, char c){
  Node* node = ht->root;
  Node* target = find_node(node, c);

  int* length = malloc(sizeof(int));
  *length = 0;

  int* path_length = malloc(sizeof(int));
  unsigned short* path;

  if(target != NULL){
    debug("[add_new_element] AS");
    path = node_path(target, path_length);
    node_positioner(ht, target);
    if(is_compressor(ht)){
      huffman_append_partial_path(ht, path, *path_length);
    }
  } else {
    path = node_path(ht->nyt, path_length);

    if(is_compressor(ht)) {
      if (ht->elements == 0) {
        ht->output[0] = c;
        ht->output_length = 1;
      } else {
        huffman_append_partial_new_element(ht, path, *path_length, c);
      }
    }

    ht->elements++;
    Node* old_nyt = ht->nyt;
```



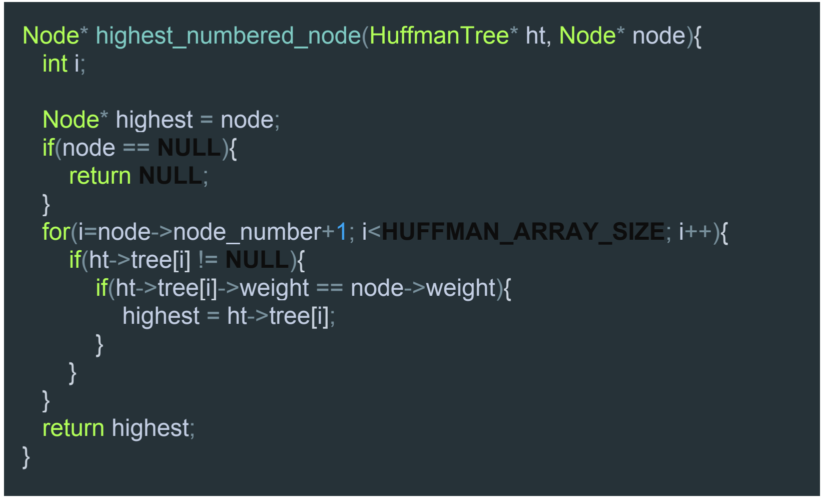
```
    if(old_nyt == NULL){
      exit(51);
    }

    Node* new_nyt = createNYT(old_nyt->node_number - 2);
    Node* new_char = createNode(old_nyt->node_number - 1, 1, c, NULL, NULL, old_nyt);

    old_nyt->weight++;
    old_nyt->left = new_nyt;
    old_nyt->right = new_char;
    old_nyt->element = -1;

    ht->nyt = new_nyt;

    new_nyt->parent = old_nyt;
    new_char->parent = old_nyt;

    ht->tree[new_nyt->node_number] = new_nyt;
    ht->tree[new_char->node_number] = new_char;
    target = old_nyt;
  }
  free(path);
  free(path_length);
  free(length);

  while(target != ht->root){
    if(target == NULL || target->parent == NULL){
      return NULL;
    }
    target = target->parent;
    node_positioner(ht, target);
  }
  return ht;
}
```

Node Positioner e Highest Numbered Node



```
void node_positioner(HuffmanTree* ht, Node* target){
  if(target == NULL){
    return;
  }
  Node* last = highest_numbered_node(ht, target);
  char buffer[250];

  if(last != target && last != target->parent) {
    swap_nodes(ht, target, last);
  }
  if(last != target->parent || target->parent == ht->root)
    target->weight++;
}
```

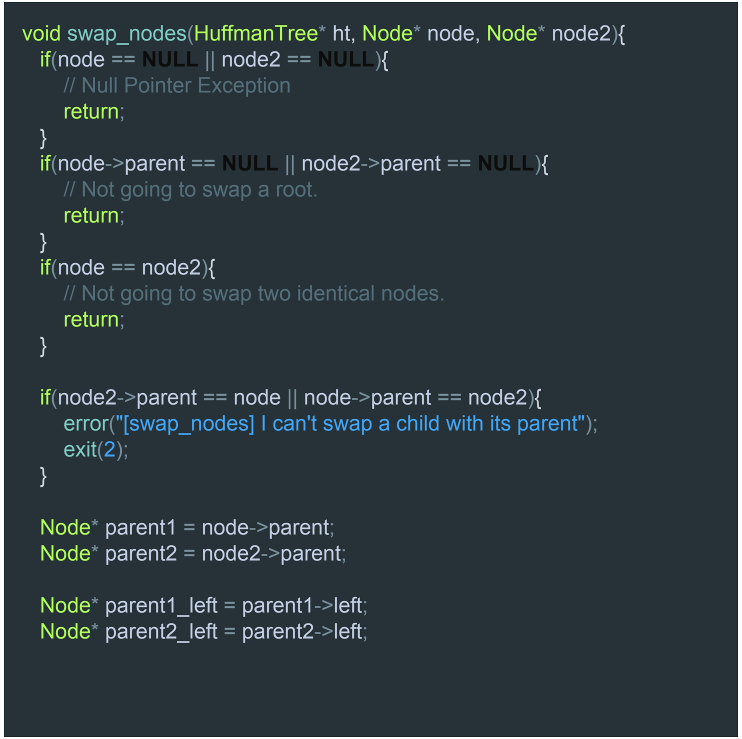
Node Positioner



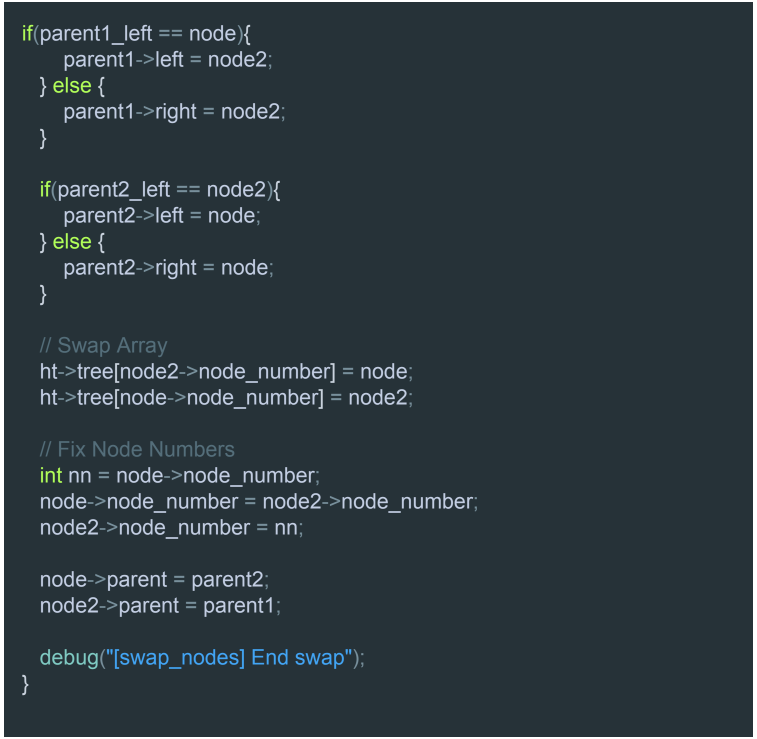
```
Node* highest_numbered_node(HuffmanTree* ht, Node* node){
  int i;

  Node* highest = node;
  if(node == NULL){
    return NULL;
  }
  for(i=node->node_number+1; i<HUFFMAN_ARRAY_SIZE; i++){
    if(ht->tree[i] != NULL){
      if(ht->tree[i]->weight == node->weight){
        highest = ht->tree[i];
      }
    }
  }
  return highest;
}
```

Highest Numbered Node

Swap nodes

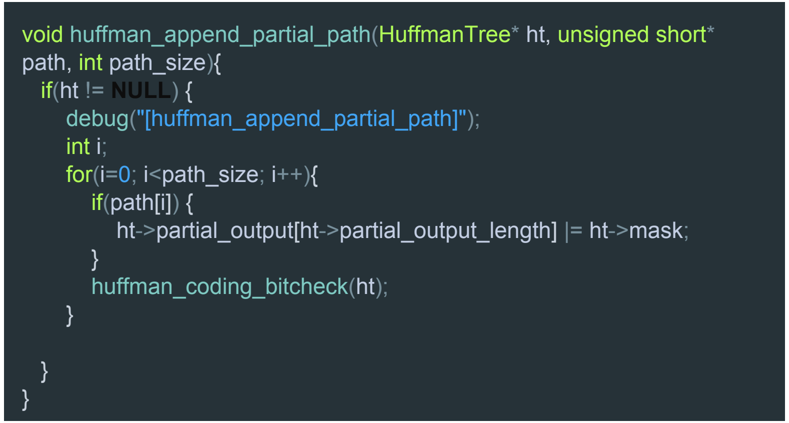


```
void swap_nodes(HuffmanTree* ht, Node* node, Node* node2){
  if(node == NULL || node2 == NULL){
    // Null Pointer Exception
    return;
  }
  if(node->parent == NULL || node2->parent == NULL){
    // Not going to swap a root.
    return;
  }
  if(node == node2){
    // Not going to swap two identical nodes.
    return;
  }

  if(node2->parent == node || node->parent == node2){
    error("[swap_nodes] I can't swap a child with its parent");
    exit(2);
  }

  Node* parent1 = node->parent;
  Node* parent2 = node2->parent;

  Node* parent1_left = parent1->left;
  Node* parent2_left = parent2->left;
```

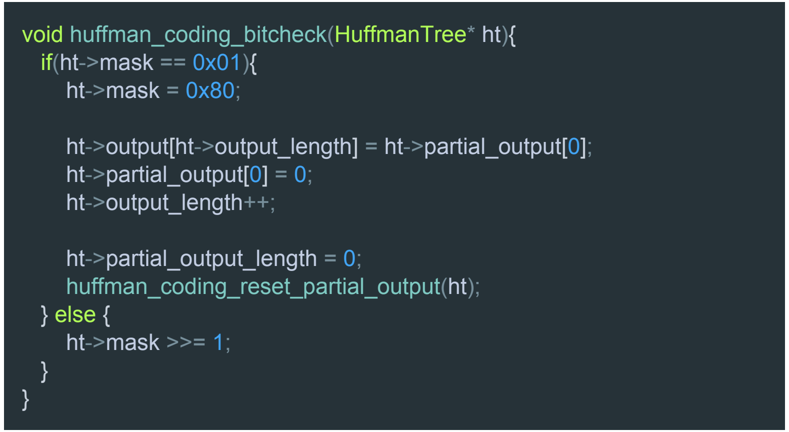


```
if(parent1_left == node){
    parent1->left = node2;
  } else {
    parent1->right = node2;
  }

  if(parent2_left == node2){
    parent2->left = node;
  } else {
    parent2->right = node;
  }

  // Swap Array
  ht->tree[node2->node_number] = node;
  ht->tree[node->node_number] = node2;

  // Fix Node Numbers
  int nn = node->node_number;
  node->node_number = node2->node_number;
  node2->node_number = nn;

  node->parent = parent2;
  node2->parent = parent1;

  debug("[swap_nodes] End swap");
}
```

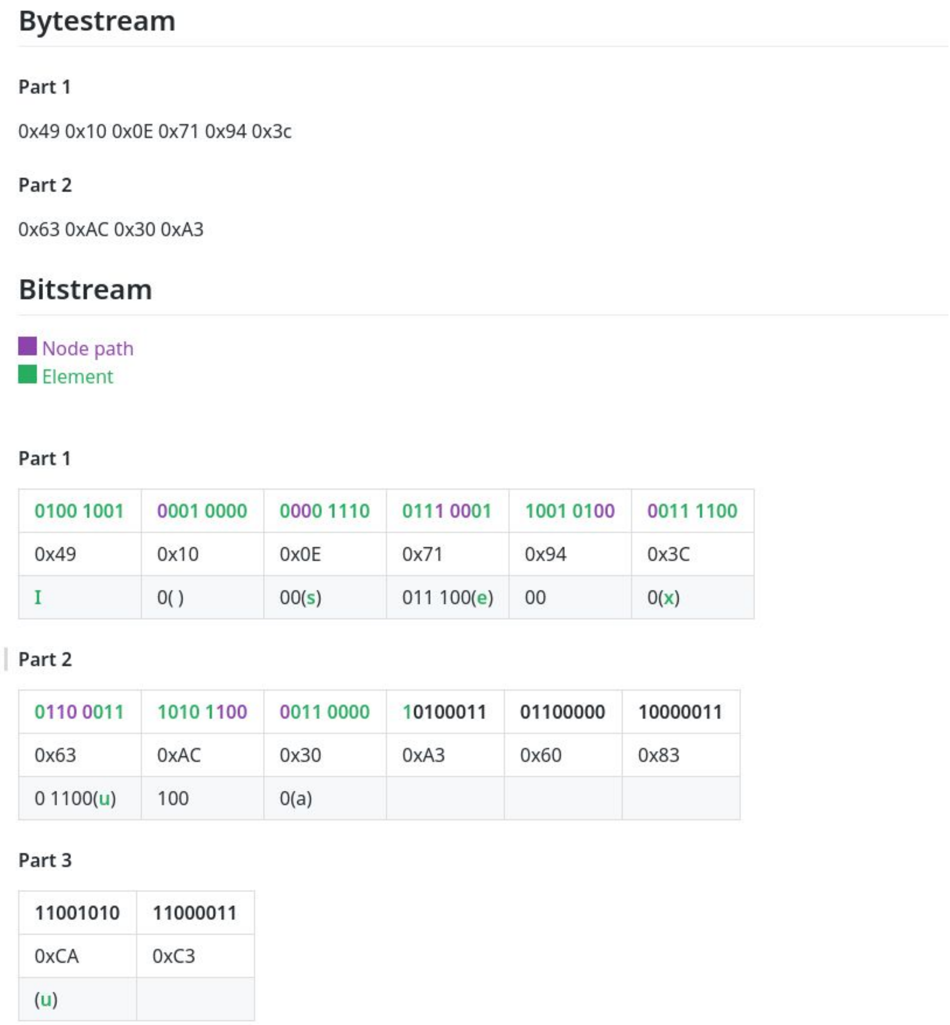
Split dei byte



```
void huffman_append_partial_path(HuffmanTree* ht, unsigned short*
path, int path_size){
  if(ht != NULL) {
      debug("[huffman_append_partial_path]");
      int i;
      for(i=0; i<path_size; i++){
         if(path[i]) {
            ht->partial_output[ht->partial_output_length] |= ht->mask;
         }
         huffman_coding_bitcheck(ht);
      }

  }
}
```



```
void huffman_coding_bitcheck(HuffmanTree* ht){
  if(ht->mask == 0x01){
      ht->mask = 0x80;

      ht->output[ht->output_length] = ht->partial_output[0];
      ht->partial_output[0] = 0;
      ht->output_length++;

      ht->partial_output_length = 0;
      huffman_coding_reset_partial_output(ht);
  } else {
      ht->mask >>= 1;
  }
}
```



Bytestream

Part 1

0x49 0x10 0x0E 0x71 0x94 0x3c

Part 2

0x63 0xAC 0x30 0xA3

Bitstream

🟪 Node path
🟩 Element

Part 1

0100 1001	0001 0000	0000 1110	0111 0001	1001 0100	0011 1100
0x49	0x10	0x0E	0x71	0x94	0x3C
I	0( )	00(s)	011 100(e)	00	0(x)

Part 2

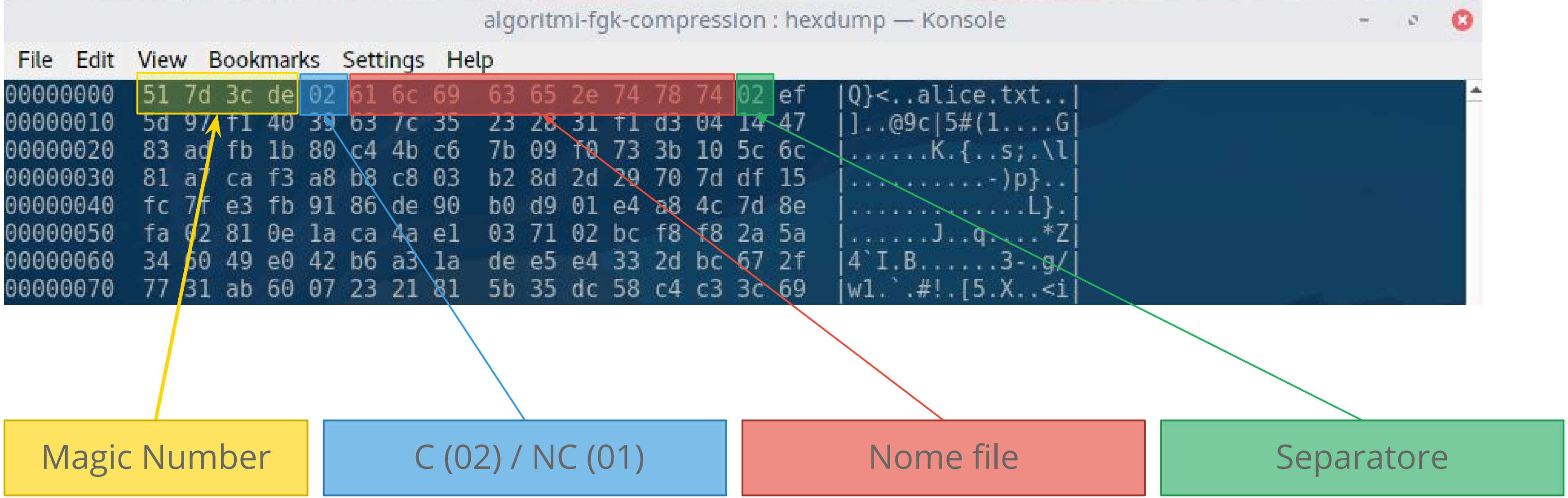
0110 0011	1010 1100	0011 0000	10100011	01100000	10000011
0x63	0xAC	0x30	0xA3	0x60	0x83
0 1100(u)	100	0(a)			

Part 3

11001010	11000011
0xCA	0xC3
(u)	

Struttura file .viz

Struttura del file



```
00000000  51 7d 3c de 02 61 6c 69  63 65 2e 74 78 74 02 ef  |Q}<..alice.txt..|
00000010  5d 97 f1 40 39 63 7c 35  23 28 31 f1 d3 04 14 47  |]..@9c|5#(1....G|
00000020  83 ad fb 1b 80 c4 4b c6  7b 09 f0 73 3b 10 5c 6c  |......K.{..s;.\l|
00000030  81 a7 ca f3 a8 b8 c8 03  b2 8d 2d 29 70 7d df 15  |..........-)p}..|
00000040  fc 7f e3 fb 91 86 de 90  b0 d9 01 e4 a8 4c 7d 8e  |.............L}.|
00000050  fa 02 81 0e 1a ca 4a e1  03 71 02 bc f8 f8 2a 5a  |......J..q....*Z|
00000060  34 60 49 e0 42 b6 a3 1a  de e5 e4 33 2d bc 67 2f  |4`I.B......3-.g/|
00000070  77 31 ab 60 07 23 21 81  5b 35 dc 58 c4 c3 3c 69  |w1.`.#!.[5.X..<i|
```

Magic Number

C (02) / NC (01)

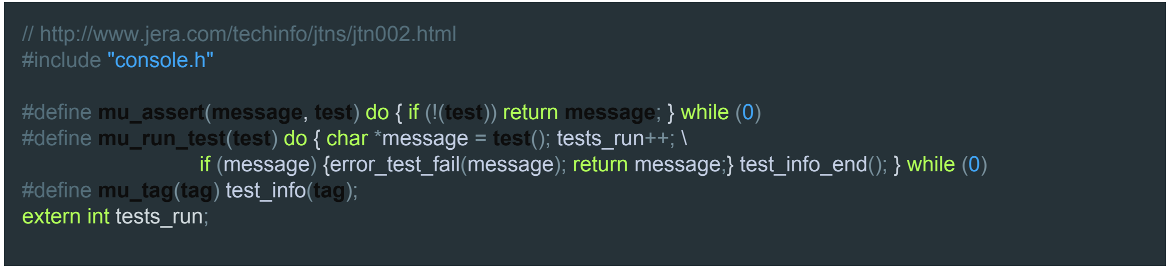
Nome file

Separatore

Testing

Framework

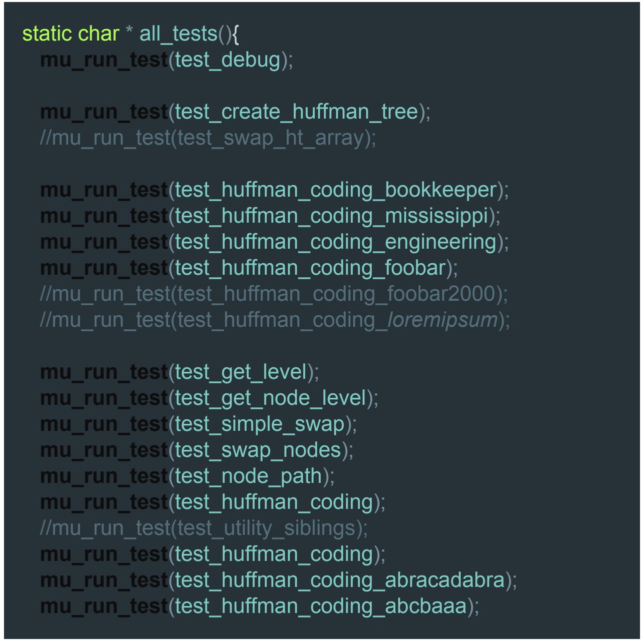
minunit.h - a minimal unit testing framework for C



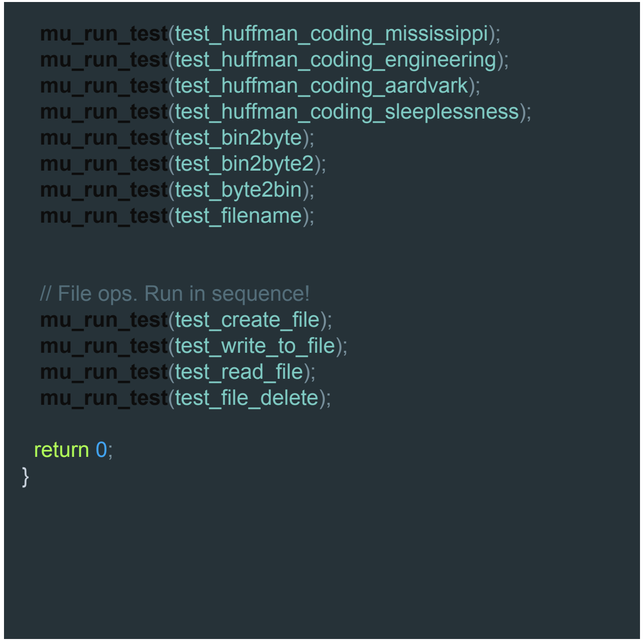
```
// http://www.jera.com/techinfo/jtns/jtn002.html
#include "console.h"

#define mu_assert(message, test) do { if (!(test)) return message; } while (0)
#define mu_run_test(test) do { char *message = test(); tests_run++; \
                    if (message) {error_test_fail(message); return message;} test_info_end(); } while (0)
#define mu_tag(tag) test_info(tag);
extern int tests_run;
```

Versione modificata di minunit.h

I nostri test

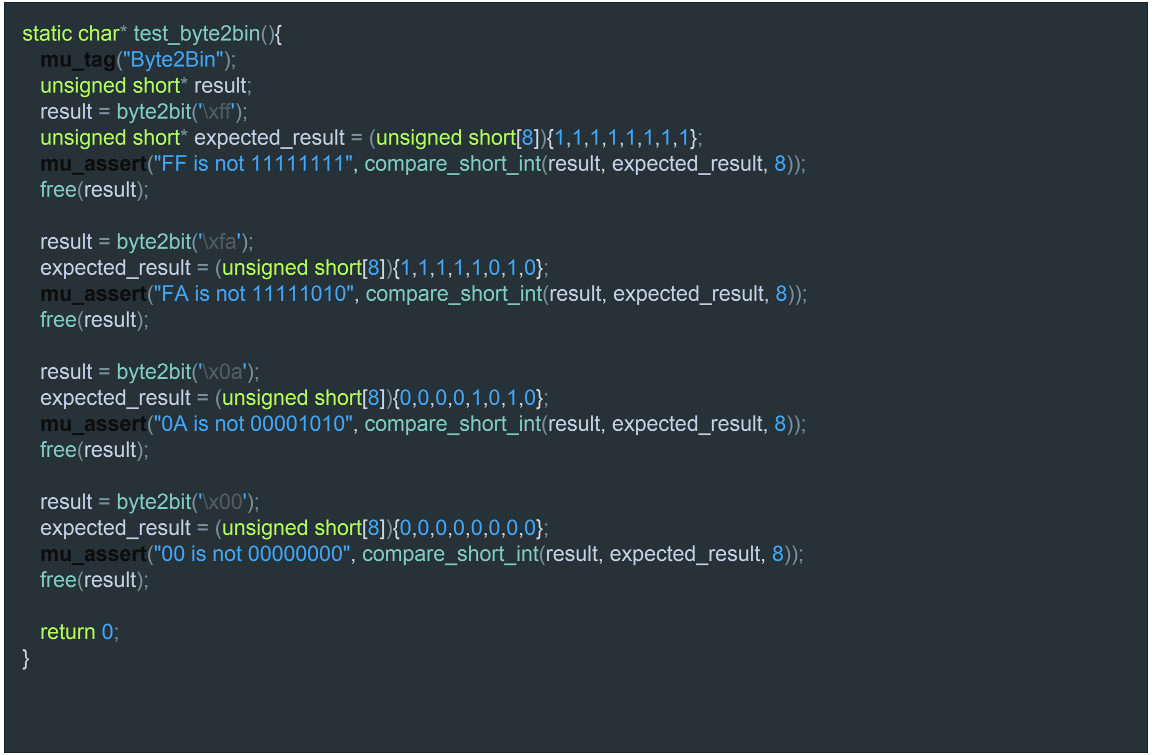


```
static char * all_tests(){
  mu_run_test(test_debug);

  mu_run_test(test_create_huffman_tree);
  //mu_run_test(test_swap_ht_array);

  mu_run_test(test_huffman_coding_bookkeeper);
  mu_run_test(test_huffman_coding_mississippi);
  mu_run_test(test_huffman_coding_engineering);
  mu_run_test(test_huffman_coding_foobar);
  //mu_run_test(test_huffman_coding_foobar2000);
  //mu_run_test(test_huffman_coding_loremipsum);

  mu_run_test(test_get_level);
  mu_run_test(test_get_node_level);
  mu_run_test(test_simple_swap);
  mu_run_test(test_swap_nodes);
  mu_run_test(test_node_path);
  mu_run_test(test_huffman_coding);
  //mu_run_test(test_utility_siblings);
  mu_run_test(test_huffman_coding);
  mu_run_test(test_huffman_coding_abracadabra);
  mu_run_test(test_huffman_coding_abcbaaa);
```

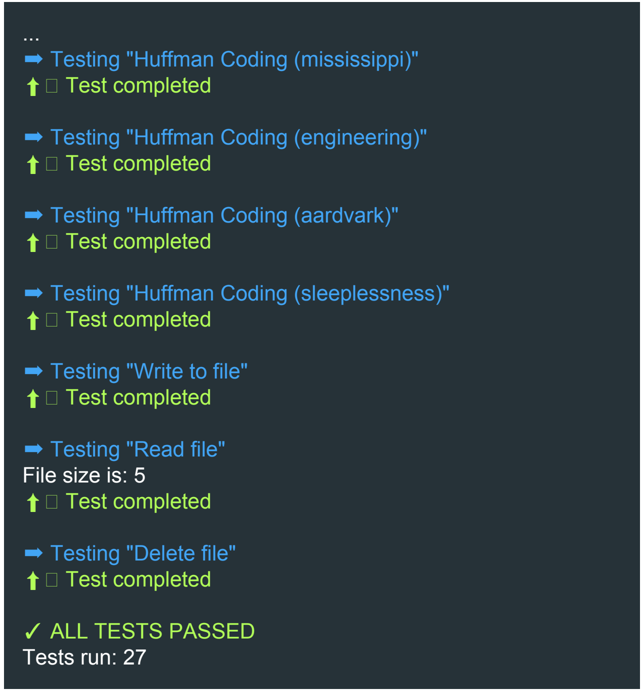


```
  mu_run_test(test_huffman_coding_mississippi);
  mu_run_test(test_huffman_coding_engineering);
  mu_run_test(test_huffman_coding_aardvark);
  mu_run_test(test_huffman_coding_sleeplessness);
  mu_run_test(test_bin2byte);
  mu_run_test(test_bin2byte2);
  mu_run_test(test_byte2bin);
  mu_run_test(test_filename);


  // File ops. Run in sequence!
  mu_run_test(test_create_file);
  mu_run_test(test_write_to_file);
  mu_run_test(test_read_file);
  mu_run_test(test_file_delete);

  return 0;
}
```

Esempio di test (test_byte2bin)



```
static char* test_byte2bin(){
  mu_tag("Byte2Bin");
  unsigned short* result;
  result = byte2bit('\xff');
  unsigned short* expected_result = (unsigned short[8]){1,1,1,1,1,1,1,1};
  mu_assert("FF is not 11111111", compare_short_int(result, expected_result, 8));
  free(result);

  result = byte2bit('\xfa');
  expected_result = (unsigned short[8]){1,1,1,1,1,0,1,0};
  mu_assert("FA is not 11111010", compare_short_int(result, expected_result, 8));
  free(result);

  result = byte2bit('\x0a');
  expected_result = (unsigned short[8]){0,0,0,0,1,0,1,0};
  mu_assert("0A is not 00001010", compare_short_int(result, expected_result, 8));
  free(result);

  result = byte2bit('\x00');
  expected_result = (unsigned short[8]){0,0,0,0,0,0,0,0};
  mu_assert("00 is not 00000000", compare_short_int(result, expected_result, 8));
  free(result);

  return 0;
}
```



```
...
➡ Testing "Huffman Coding (mississippi)"
⬆ □ Test completed

➡ Testing "Huffman Coding (engineering)"
⬆ □ Test completed

➡ Testing "Huffman Coding (aardvark)"
⬆ □ Test completed

➡ Testing "Huffman Coding (sleeplessness)"
⬆ □ Test completed

➡ Testing "Write to file"
⬆ □ Test completed

➡ Testing "Read file"
File size is: 5
⬆ □ Test completed

➡ Testing "Delete file"
⬆ □ Test completed

✓ ALL TESTS PASSED
Tests run: 27
```

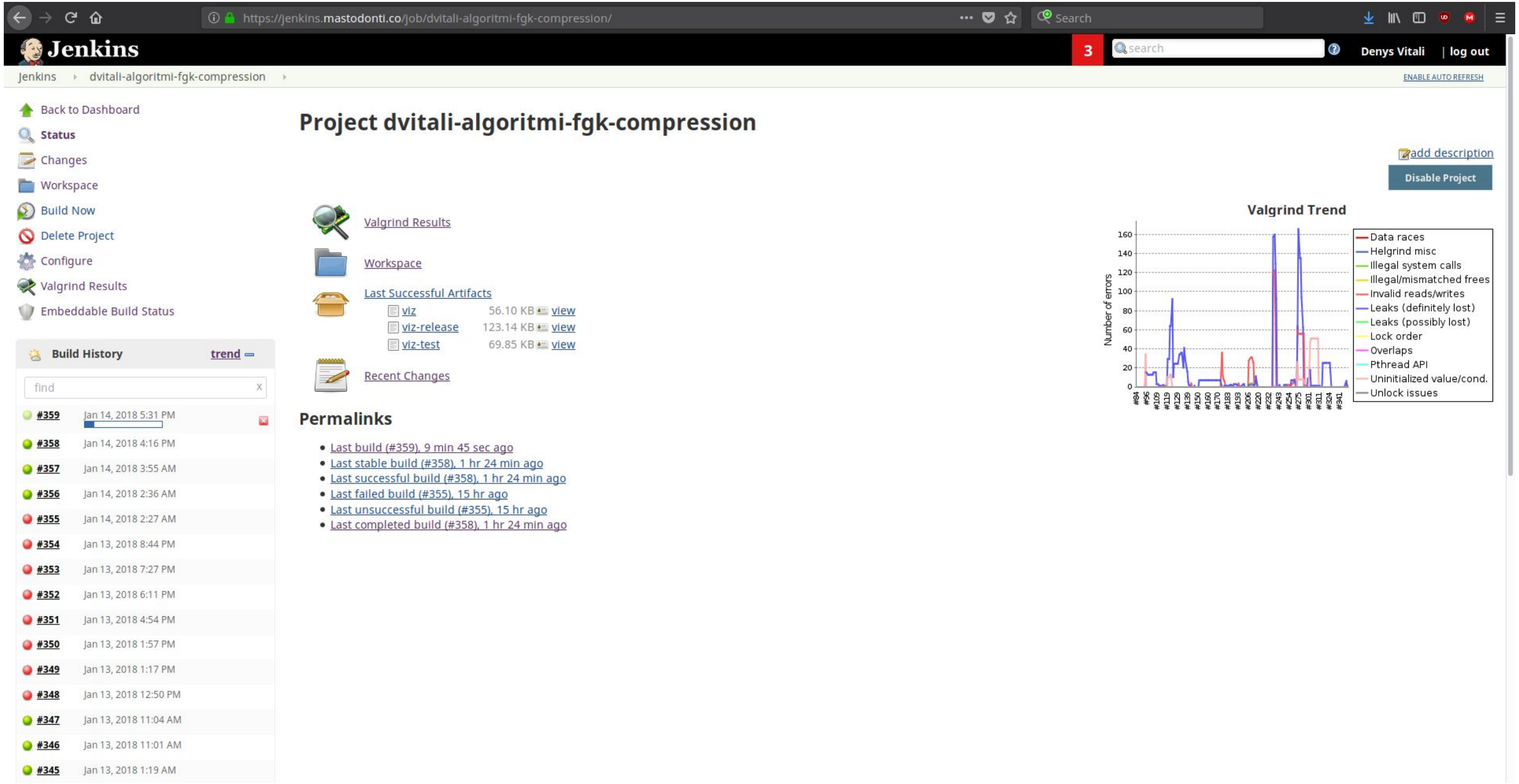
Problemi riscontrati

- Swap dei nodi nell’ HT Array
- Split dei byte
- Decompressione (WIP)

Jenkins

Continuous Integration
&
Automation

Automatic Builds



Project dvitali-algoritmi-fgk-compression

Valgrind Results

Workspace

Last Successful Artifacts

viz	56.10 KB	view
viz-release	123.14 KB	view
viz-test	69.85 KB	view

Recent Changes

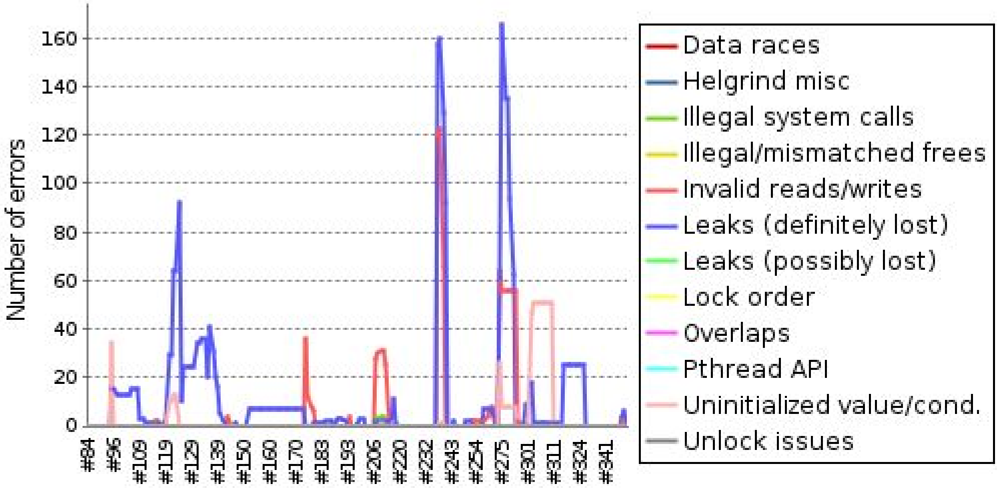
Permalinks

- Last build (#359), 9 min 45 sec ago
- Last stable build (#358), 1 hr 24 min ago
- Last successful build (#358), 1 hr 24 min ago
- Last failed build (#355), 15 hr ago
- Last unsuccessful build (#355), 15 hr ago
- Last completed build (#358), 1 hr 24 min ago

Build History trend

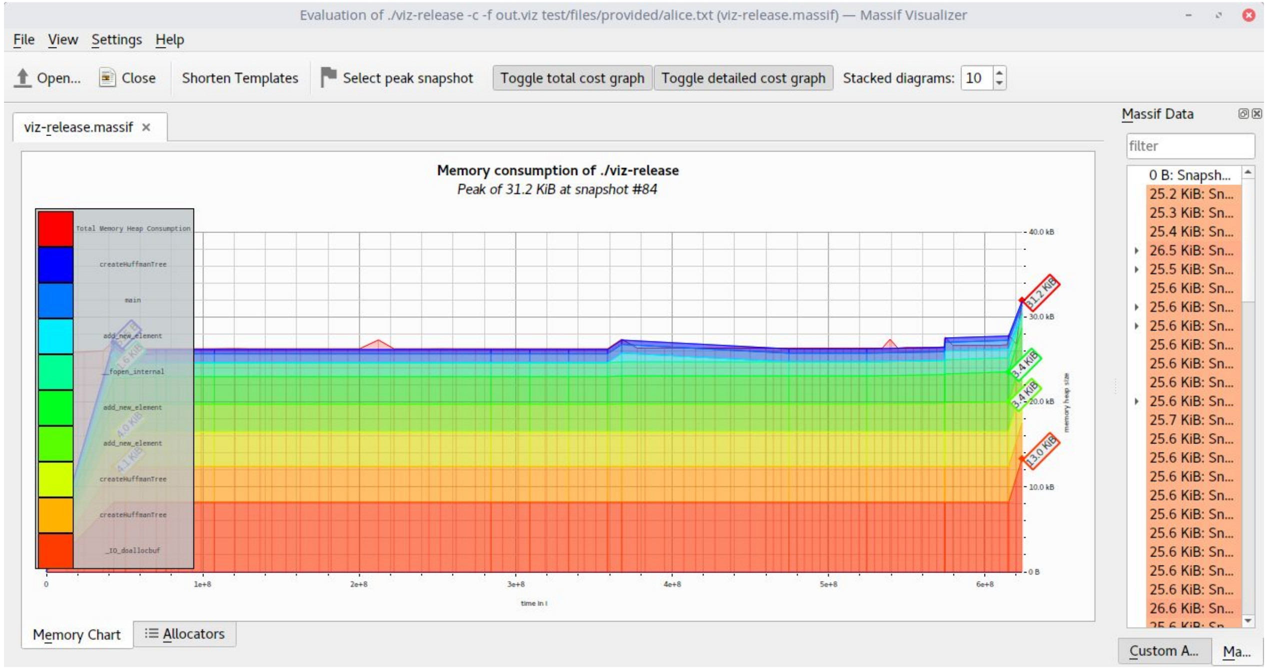
○	#359	Jan 14, 2018 5:31 PM
🟢	#358	Jan 14, 2018 4:16 PM
🟢	#357	Jan 14, 2018 3:55 AM
🟢	#356	Jan 14, 2018 2:36 AM
🔴	#355	Jan 14, 2018 2:27 AM
🔴	#354	Jan 13, 2018 8:44 PM
🔴	#353	Jan 13, 2018 7:27 PM
🔴	#352	Jan 13, 2018 6:11 PM
🔴	#351	Jan 13, 2018 4:54 PM
🔴	#350	Jan 13, 2018 1:57 PM
🔴	#349	Jan 13, 2018 1:17 PM
🔴	#348	Jan 13, 2018 12:50 PM
🟢	#347	Jan 13, 2018 11:04 AM
🟢	#346	Jan 13, 2018 11:01 AM
🟢	#345	Jan 13, 2018 1:19 AM

Valgrind


Number of errors
160
140
120
100
80
60
40
20
0
#84
#96
#109
#119
#129
#139
#150
#160
#170
#183
#193
#206
#220
#232
#243
#254
#275
#301
#311
#324
#341
Data races
Helgrind misc
Illegal system calls
Illegal/mismatched frees
Invalid reads/writes
Leaks (definitely lost)
Leaks (possibly lost)
Lock order
Overlaps
Pthread API
Uninitialized value/cond.
Unlock issues

Massif - Memory Visualizer

make massif_release (make massif_release_c)



Evaluation of ./viz-release -c -f out.viz test/files/provided/alice.txt (viz-release.massif) — Massif Visualizer

Stacked diagrams: 10

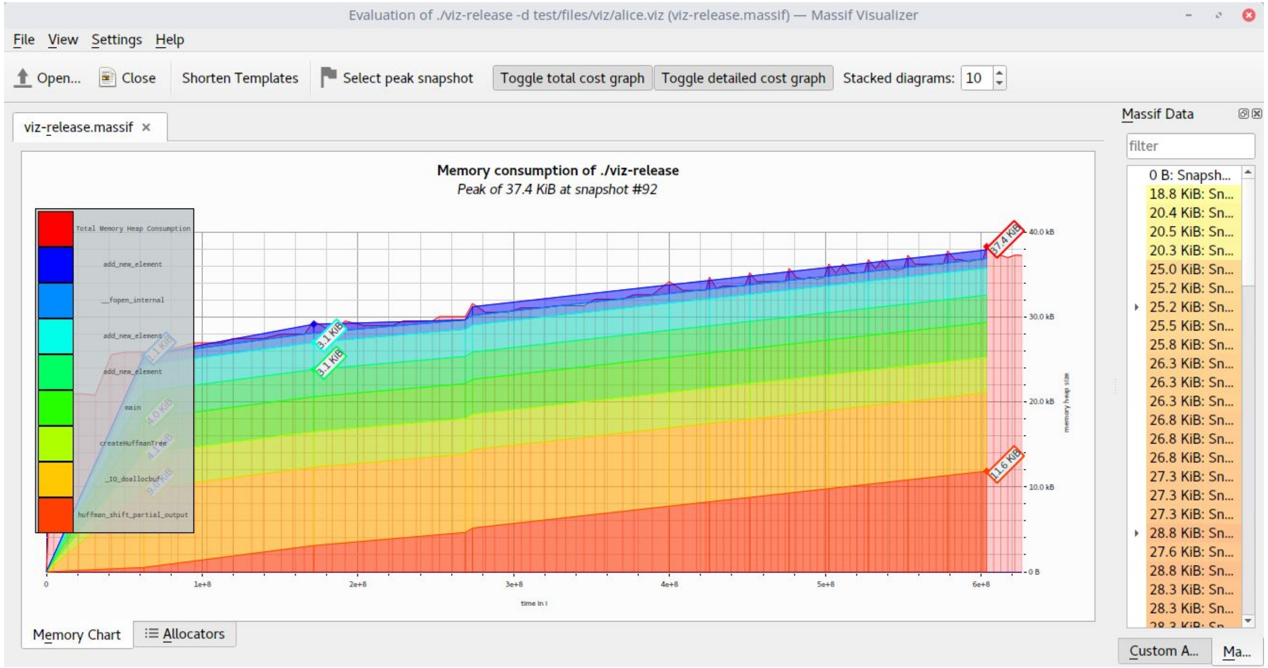
Memory consumption of ./viz-release

Peak of 31.2 KiB at snapshot #84

0 B: Snapsh...
25.2 KiB: Sn...
25.3 KiB: Sn...
25.4 KiB: Sn...
26.5 KiB: Sn...
25.5 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.7 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
25.6 KiB: Sn...
26.6 KiB: Sn...

Massif - Memory Visualizer

make massif_release_d

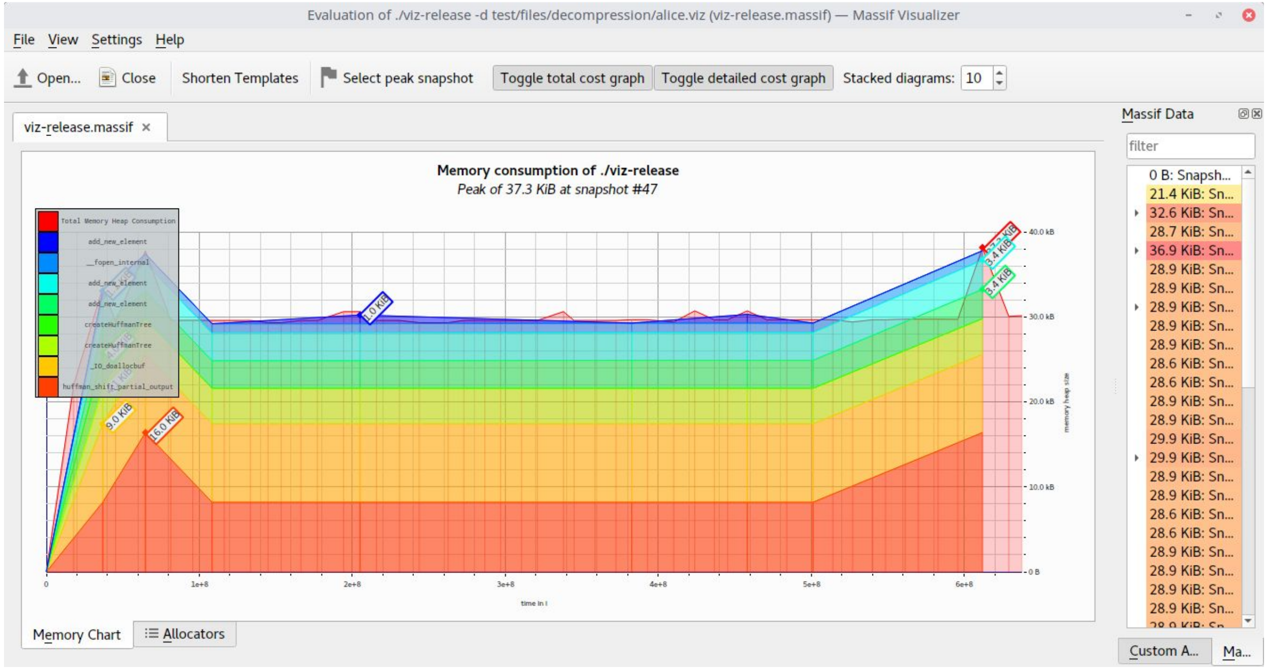


Evaluation of ./viz-release -d test/files/viz/alice.viz (viz-release.massif) — Massif Visualizer

```
0 B: Snapsh...
18.8 KiB: Sn...
20.4 KiB: Sn...
20.5 KiB: Sn...
20.3 KiB: Sn...
25.0 KiB: Sn...
25.2 KiB: Sn...
25.2 KiB: Sn...
25.5 KiB: Sn...
25.8 KiB: Sn...
26.3 KiB: Sn...
26.3 KiB: Sn...
26.3 KiB: Sn...
26.8 KiB: Sn...
26.8 KiB: Sn...
26.8 KiB: Sn...
27.3 KiB: Sn...
27.3 KiB: Sn...
27.3 KiB: Sn...
28.8 KiB: Sn...
27.6 KiB: Sn...
28.8 KiB: Sn...
28.3 KiB: Sn...
28.3 KiB: Sn...
28.3 KiB: Sn...
```

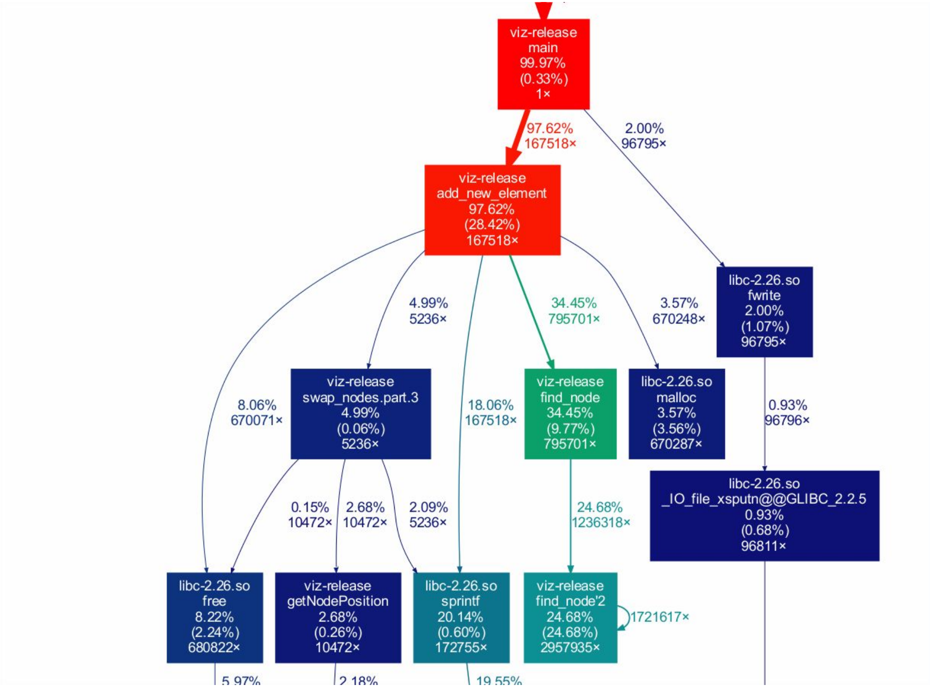
Massif - Memory Visualizer

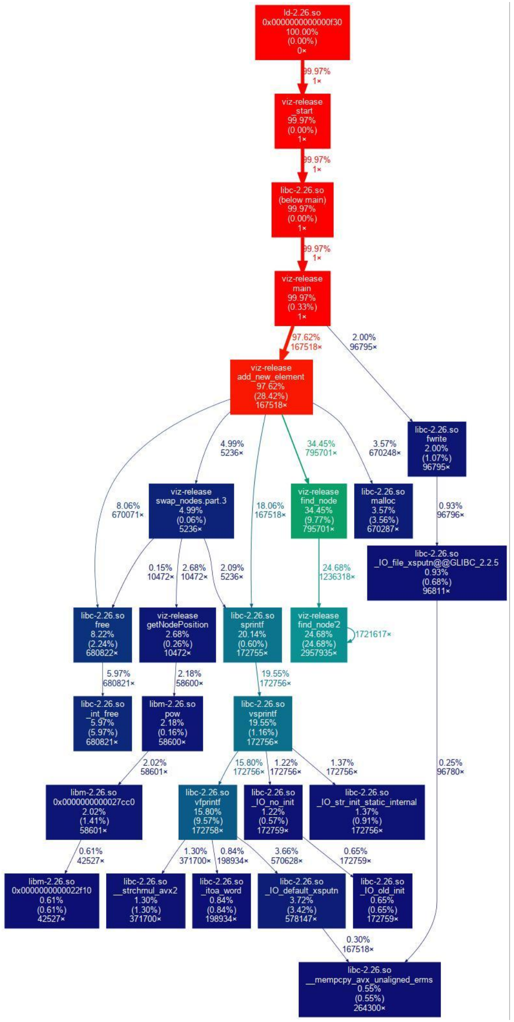
make massif_release_d - after optimization



Callgrind - Call graph

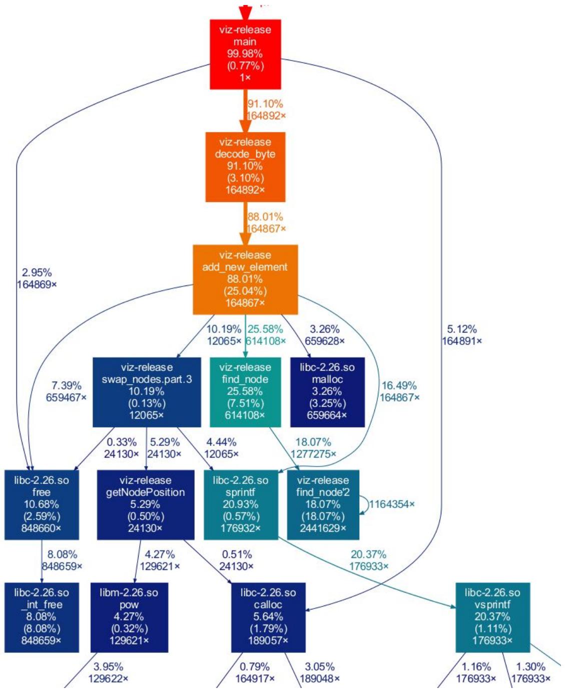
make callgrind_release (make callgrind_release_c)

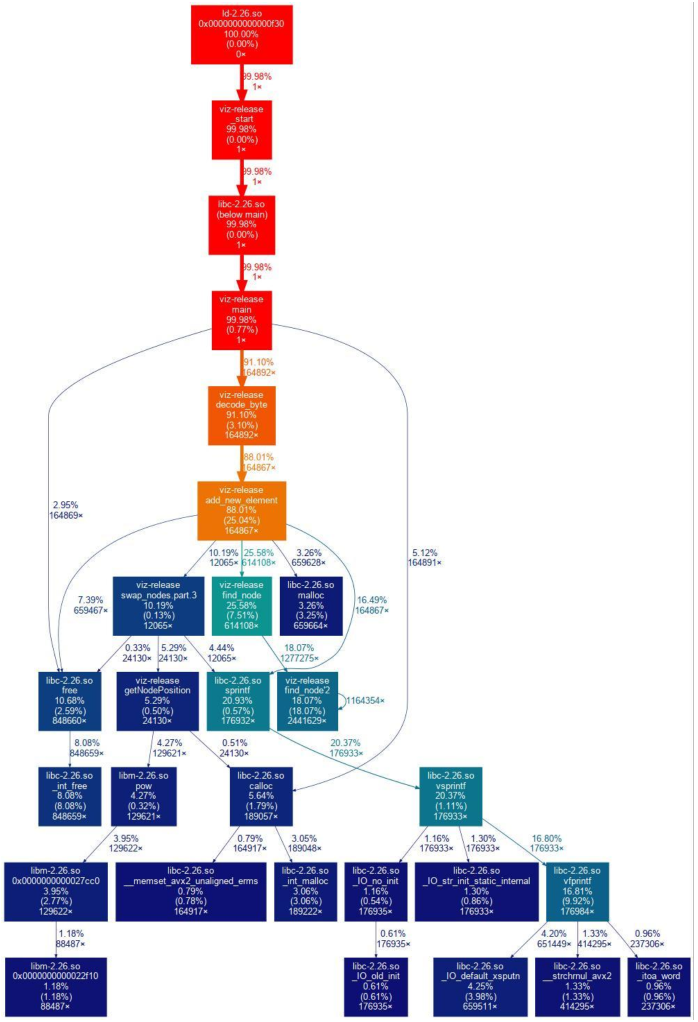

viz-release main 99.97% (0.33%) 1×
97.62% 167518×
2.00% 96795×
viz-release add_new_element 97.62% (28.42%) 167518×
8.06% 670071×
4.99% 5236×
18.06% 167518×
34.45% 795701×
3.57% 670248×
libc-2.26.so fwrite 2.00% (1.07%) 96795×
viz-release swap_nodes.part.3 4.99% (0.06%) 5236×
viz-release find_node 34.45% (9.77%) 795701×
libc-2.26.so malloc 3.57% (3.56%) 670287×
0.93% 96796×
0.15% 10472×
2.68% 10472×
2.09% 5236×
24.68% 1236318×
libc-2.26.so _IO_file_xsputn@@GLIBC_2.2.5 0.93% (0.68%) 96811×
libc-2.26.so free 8.22% (2.24%) 680822×
viz-release getNodePosition 2.68% (0.26%) 10472×
libc-2.26.so sprintf 20.14% (0.60%) 172755×
viz-release find_node'2 24.68% (24.68%) 2957935×
1721617×
5.97%
2.18%
19.55%


ld-2.26.so 0x0000000000000f30 100.00% (0.00%) 0×
99.97% 1×
viz-release _start 99.97% (0.00%) 1×
99.97% 1×
libc-2.26.so (below main) 99.97% (0.00%) 1×
99.97% 1×
viz-release main 99.97% (0.33%) 1×
97.62% 167518×
2.00% 96795×
viz-release add_new_element 97.62% (28.42%) 167518×
4.99% 5236×
34.45% 795701×
3.57% 670248×
libc-2.26.so fwrite 2.00% (1.07%) 96795×
8.06% 670071×
viz-release swap_nodes.part.3 4.99% (0.06%) 5236×
18.06% 167518×
viz-release find_node 34.45% (9.77%) 795701×
libc-2.26.so malloc 3.57% (3.56%) 670287×
0.93% 96796×
libc-2.26.so _IO_file_xsputn@@GLIBC_2.2.5 0.93% (0.68%) 96811×
0.15% 10472×
2.68% 10472×
2.09% 5236×
24.68% 1236318×
libc-2.26.so free 8.22% (2.24%) 680822×
viz-release getNodePosition 2.68% (0.26%) 10472×
libc-2.26.so sprintf 20.14% (0.60%) 172755×
viz-release find_node'2 24.68% (24.68%) 2957935×
1721617×
5.97% 680821×
2.18% 58600×
19.55% 172756×
libc-2.26.so _int_free 5.97% (5.97%) 680821×
libm-2.26.so pow 2.18% (0.16%) 58600×
libc-2.26.so vsprintf 19.55% (1.16%) 172756×
2.02% 58601×
15.80% 172756×
1.22% 172756×
1.37% 172756×
0.25% 96780×
libm-2.26.so 0x0000000000027cc0 2.02% (1.41%) 58601×
libc-2.26.so vfprintf 15.80% (9.57%) 172758×
libc-2.26.so _IO_no_init 1.22% (0.57%) 172759×
libc-2.26.so _IO_str_init_static_internal 1.37% (0.91%) 172756×
0.61% 42527×
1.30% 371700×
0.84% 198934×
3.66% 570628×
0.65% 172759×
libm-2.26.so 0x0000000000022f10 0.61% (0.61%) 42527×
libc-2.26.so __strchrnul_avx2 1.30% (1.30%) 371700×
libc-2.26.so _itoa_word 0.84% (0.84%) 198934×
libc-2.26.so _IO_default_xsputn 3.72% (3.42%) 578147×
libc-2.26.so _IO_old_init 0.65% (0.65%) 172759×
0.30% 167518×
libc-2.26.so __mempcpy_avx_unaligned_erms 0.55% (0.55%) 264300×

Callgrind - Call graph

make callgrind_release_d


viz-release main 99.98% (0.77%) 1×
91.10% 164892×
viz-release decode_byte 91.10% (3.10%) 164892×
88.01% 164867×
viz-release add_new_element 88.01% (25.04%) 164867×
2.95% 164869×
5.12% 164891×
7.39% 659467×
10.19% 12065×
25.58% 614108×
3.26% 659628×
16.49% 164867×
viz-release swap_nodes.part.3 10.19% (0.13%) 12065×
viz-release find_node 25.58% (7.51%) 614108×
libc-2.26.so malloc 3.26% (3.25%) 659664×
0.33% 24130×
5.29% 24130×
4.44% 12065×
18.07% 1277275×
libc-2.26.so free 10.68% (2.59%) 848660×
viz-release getNodePosition 5.29% (0.50%) 24130×
libc-2.26.so sprintf 20.93% (0.57%) 176932×
viz-release find_node'2 18.07% (18.07%) 2441629×
1164354×
8.08% 848659×
4.27% 129621×
0.51% 24130×
20.37% 176933×
libc-2.26.so _int_free 8.08% (8.08%) 848659×
libm-2.26.so pow 4.27% (0.32%) 129621×
libc-2.26.so calloc 5.64% (1.79%) 189057×
libc-2.26.so vsprintf 20.37% (1.11%) 176933×
3.95% 129622×
0.79% 164917×
3.05% 189048×
1.16% 176933×
1.30% 176933×


ld-2.26.so 0x0000000000000f30 100.00% (0.00%) 0×
99.98% 1×
viz-release _start 99.98% (0.00%) 1×
99.98% 1×
libc-2.26.so (below main) 99.98% (0.00%) 1×
99.98% 1×
viz-release main 99.98% (0.77%) 1×
91.10% 164892×
viz-release decode_byte 91.10% (3.10%) 164892×
88.01% 164867×
viz-release add_new_element 88.01% (25.04%) 164867×
2.95% 164869×
5.12% 164891×
10.19% 12065×
25.58% 614108×
3.26% 659628×
7.39% 659467×
16.49% 164867×
viz-release swap_nodes.part.3 10.19% (0.13%) 12065×
viz-release find_node 25.58% (7.51%) 614108×
libc-2.26.so malloc 3.26% (3.25%) 659664×
0.33% 24130×
5.29% 24130×
4.44% 12065×
18.07% 1277275×
libc-2.26.so free 10.68% (2.59%) 848660×
viz-release getNodePosition 5.29% (0.50%) 24130×
libc-2.26.so sprintf 20.93% (0.57%) 176932×
viz-release find_node'2 18.07% (18.07%) 2441629×
1164354×
8.08% 848659×
4.27% 129621×
0.51% 24130×
20.37% 176933×
libc-2.26.so _int_free 8.08% (8.08%) 848659×
libm-2.26.so pow 4.27% (0.32%) 129621×
libc-2.26.so calloc 5.64% (1.79%) 189057×
libc-2.26.so vsprintf 20.37% (1.11%) 176933×
3.95% 129622×
0.79% 164917×
3.05% 189048×
1.16% 176933×
1.30% 176933×
16.80% 176933×
libm-2.26.so 0x0000000000027cc0 3.95% (2.77%) 129622×
libc-2.26.so __memset_avx2_unaligned_erms 0.79% (0.78%) 164917×
libc-2.26.so _int_malloc 3.06% (3.06%) 189222×
libc-2.26.so _IO_no_init 1.16% (0.54%) 176935×
libc-2.26.so _IO_str_init_static_internal 1.30% (0.86%) 176933×
libc-2.26.so vfprintf 16.81% (9.92%) 176984×
1.18% 88487×
0.61% 176935×
4.20% 651449×
1.33% 414295×
0.96% 237306×
libm-2.26.so 0x0000000000022f10 1.18% (1.18%) 88487×
libc-2.26.so _IO_old_init 0.61% (0.61%) 176935×
libc-2.26.so _IO_default_xsputn 4.25% (3.98%) 659511×
libc-2.26.so __strchrnul_avx2 1.33% (1.33%) 414295×
libc-2.26.so _itoa_word 0.96% (0.96%) 237306×

Benchmark

800 kB/s

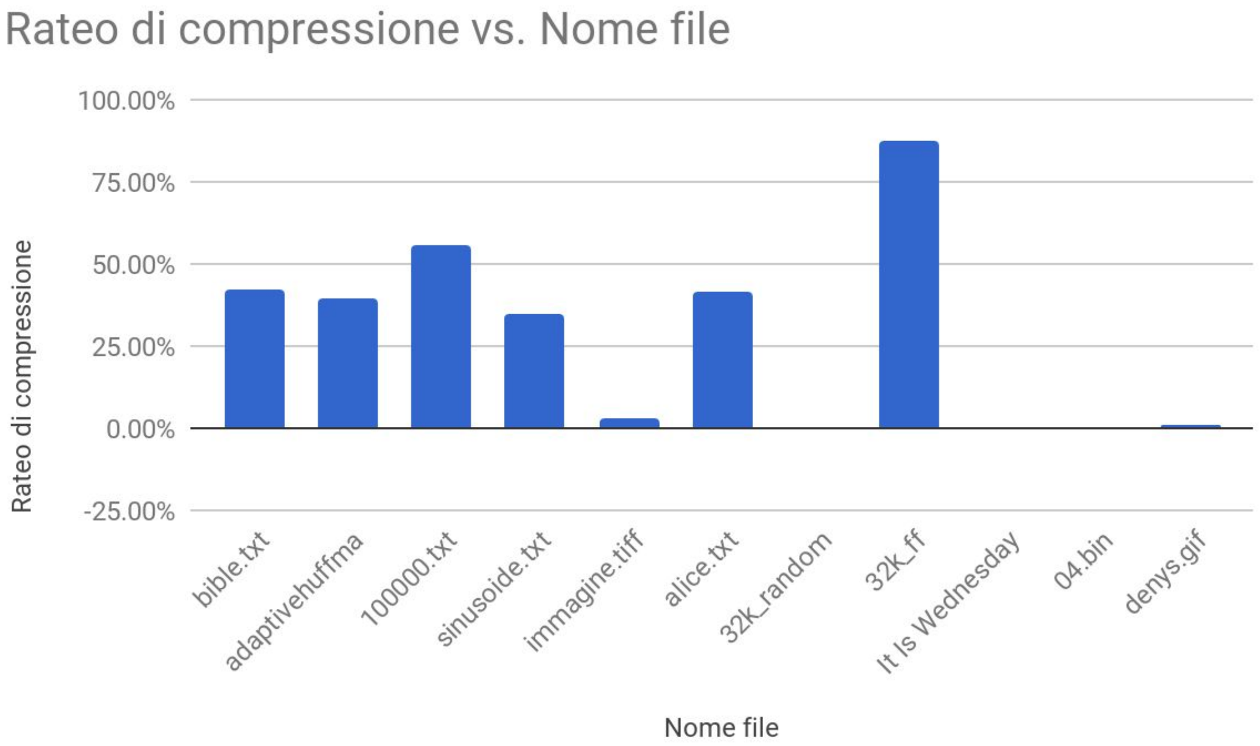
Velocità media di compressione

147MB compressi in 189.42s (153875545/189.42/1024 = 793.311 kB/s)
Intel Core i7-6700HQ @ 8x 3.5GHz, 32GB RAM, x86_64 Linux 4.14.13-1

35%

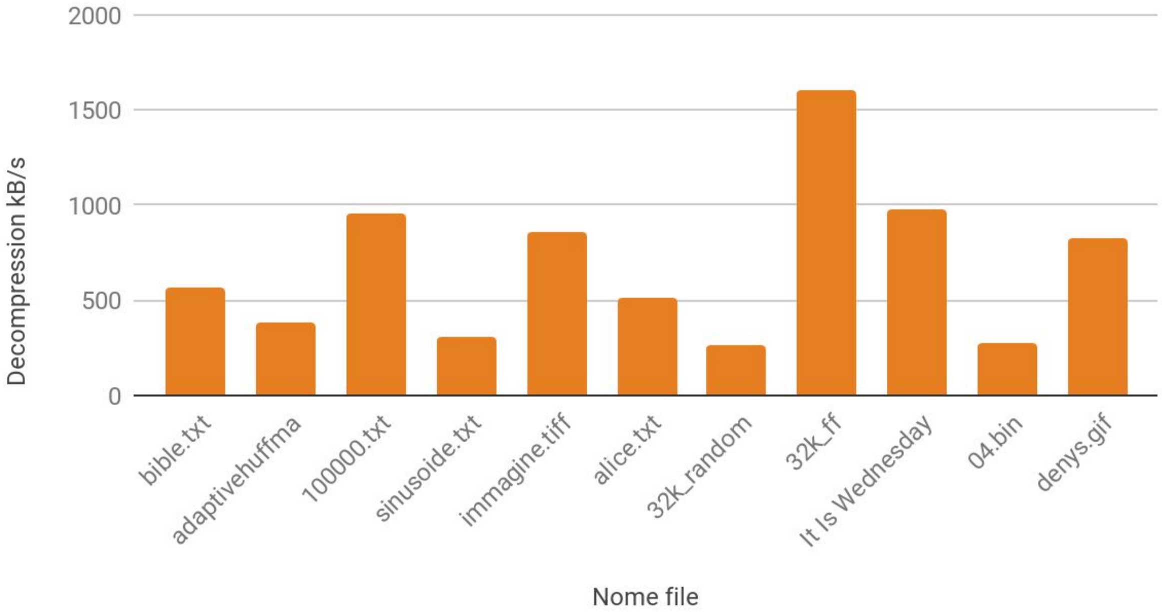
Fattore di compressione medio

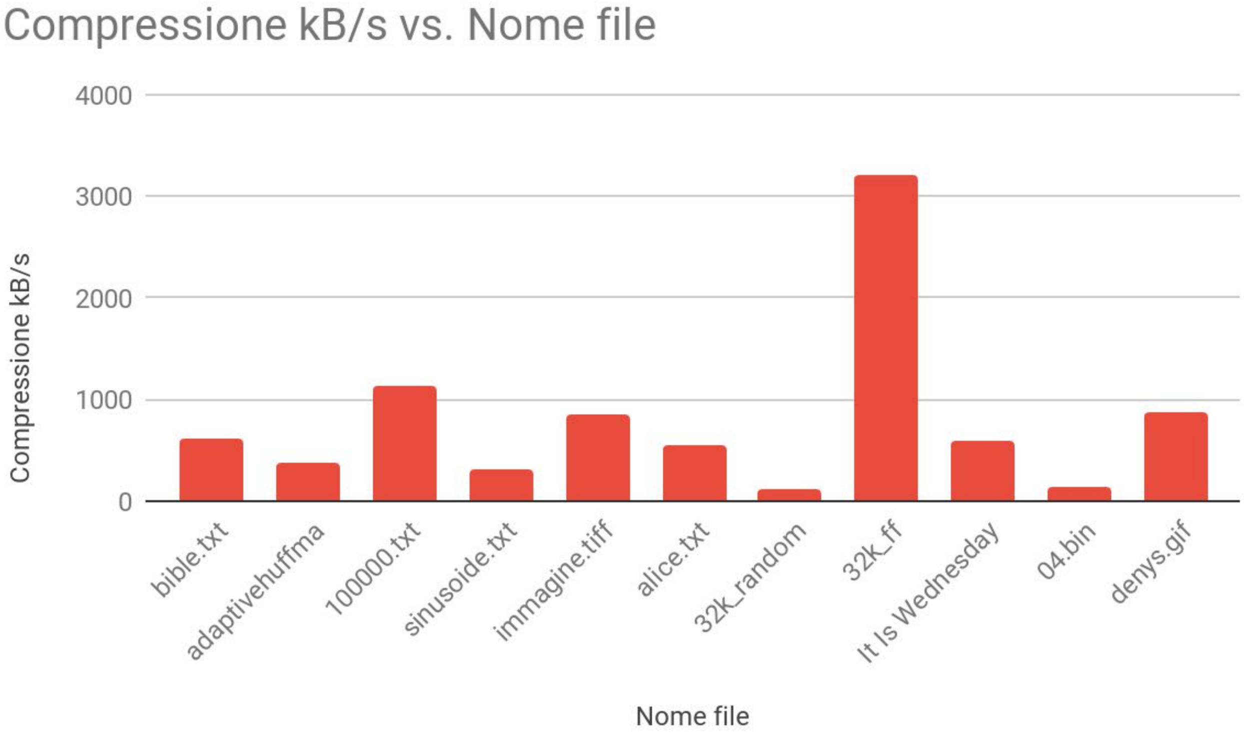
*Ottenuto con una media del rapporto di compressione dei file:
immagine.tiff, fitnessgram.txt, adaptivehuffman.txt, alice.txt, bible.txt, 32k_ff



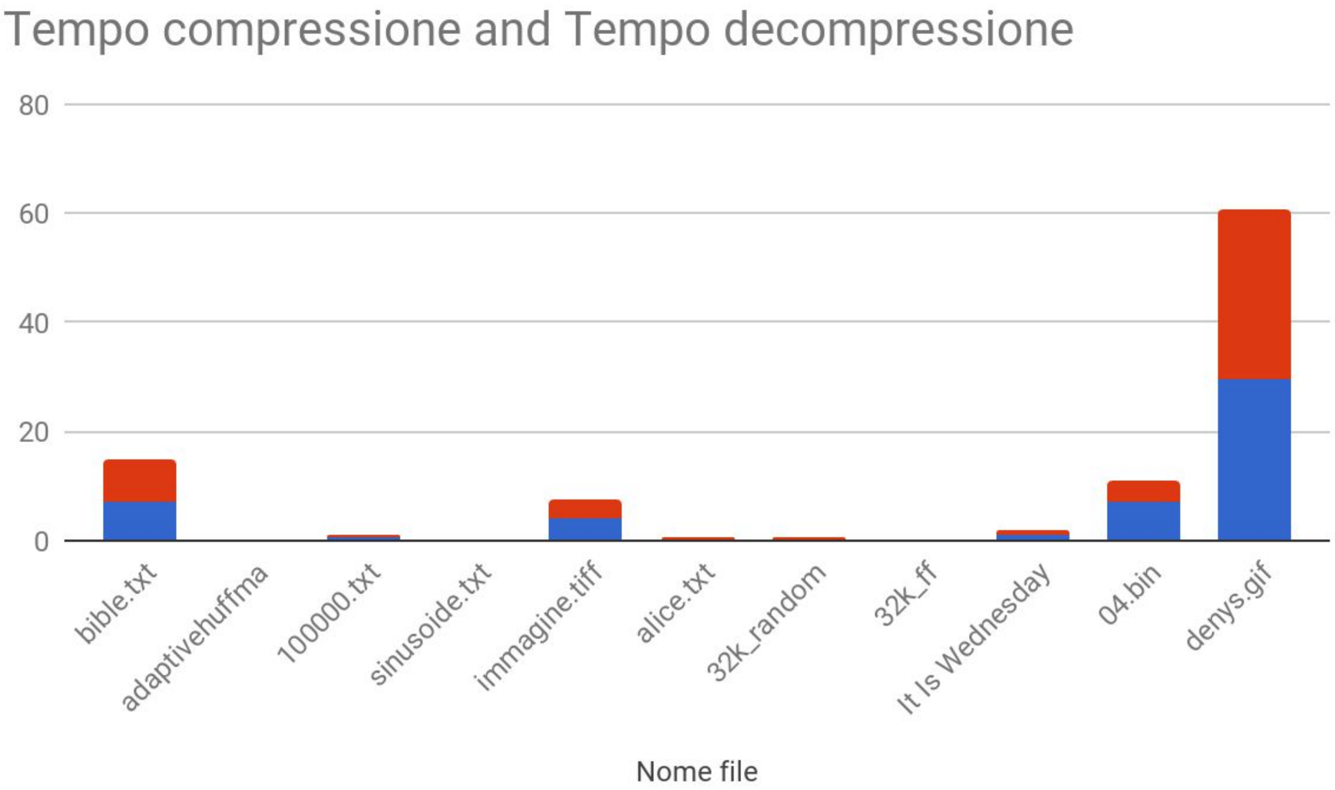
Rateo di compressione vs. Nome file

Decompression kB/s vs. Nome file





Compressione kB/s vs. Nome file


Tempo compressione and Tempo decompressione
80
60
40
20
0
bible.txt
adaptivehuffma
100000.txt
sinusoide.txt
immagine.tiff
alice.txt
32k_random
32k_ff
It Is Wednesday
04.bin
denys.gif
Nome file

60% of the time,
it works every time.