Algoritmos e Estruturas de Dados

2020/2021

Trabalho 3

Descodificação recursiva de um número binário não instantâneo

David Ferreira, 98608 (33%);

Pedro Ferreira, 98620 (34%);

Helio Filho, 93390 (33%)

Objetivo e solução computacional

Este trabalho consiste em, dado uma mensagem codificada e um conjunto de símbolos com as respetivas codewords (símbolos codificados), descobrir a mensagem original.

Para tal, foi feita uma procura em profundidade por todas as combinações de codewords encontradas na mensagem codificada. Como todas as codewords são diferentes, então só existe 1 mensagem encontrada válida, que será encontrada quando o programa encontrar a codeword do último símbolo da mensagem.

Resultados (tabela em apêndice)

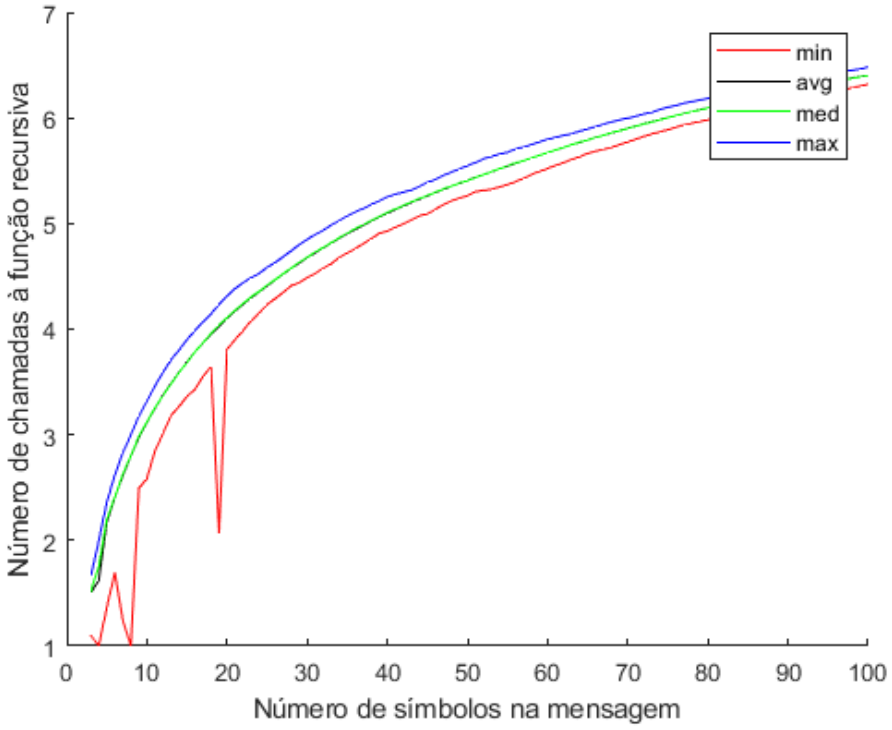

Gráfico 1

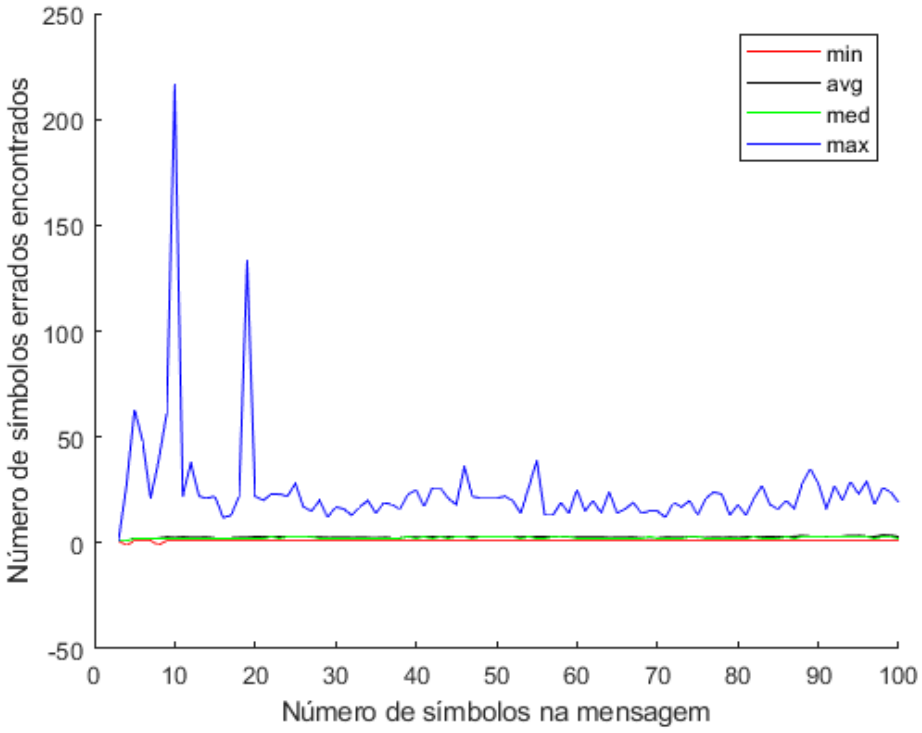

Gráfico 2

Conclusões

No gráfico 1, número de chamadas à função recursiva aumenta exponencialmente. Na função dos os valores mínimos, no mesmo gráfico, verificamos que existem algumas anomalias, o que quer dizer que houve um menor número de chamadas à função do que era esperado, ou seja, a mensagem original foi encontrada em menos iterações nesses pontos. Isto é normal especialmente para um número de símbolos na mensagem mais pequeno, visto que a probabilidade de tal acontecer é maior.

Podemos então concluir que o melhor, pior e caso médio têm uma complexidade de **log(n)**.

Relativamente ao gráfico 2, a azul, podemos observar o número de símbolos errados para o caminho com mais símbolos errados encontrados na mensagem codificada, ou seja, o número máximo de possibilidades que teríamos de considerar se quiséssemos descodificar a mensagem em tempo real.

Apêndice 1 – Código de resolução do problema

```
static int get_next_encoded_idx(int encoded_idx, int symbol_idx)

{
 int j;

 for(j=0; j<_c_->max_bits && _c_->data[symbol_idx].codeword[j] != '\0'; j++)
 {
  if(j + encoded_idx >= _max_encoded_message_size_ ||
    _encoded_message_[encoded_idx + j] == '\0' ||
    _c_->data[symbol_idx].codeword[j] != _encoded_message_[encoded_idx + j])
  {
   return -1;
  }
 }
 return j + encoded_idx;

}


static void recursive_decoder(int encoded_idx,int decoded_idx,int good_decoded_size)
{
 // Update number of calls
 _number_of_calls_++;

 int is_dead_end = 1;
 for(int symbol=0; symbol < _c_->n_symbols; symbol++)
 {
  int next_encoded_idx = get_next_encoded_idx(encoded_idx, symbol);
  if(next_encoded_idx != -1)
  {
   is_dead_end = 0;
   _decoded_message_[decoded_idx] = symbol;

   if(next_encoded_idx >= _max_encoded_message_size_ ||
        _encoded_message_[next_encoded_idx] == '\0')
   {
    _number_of_solutions_++;
   }
   else
   {

    if (symbol == _original_message_[decoded_idx])
    {
     recursive_decoder(next_encoded_idx, decoded_idx+1, good_decoded_size+1);
    }
    else
    {
     recursive_decoder(next_encoded_idx, decoded_idx+1, good_decoded_size);
    }
   }
  }
 }
 if(is_dead_end)
 {
  _max_extra_symbols_ = decoded_idx - good_decoded_size;
 }
}
```

Apêndice 2 – Programa em python para automizar a recolha de dados e escrita em ficheiro excel

```python
import subprocess
import xlwt

book = xlwt.Workbook()
sheet1 = book.add_sheet("Data")
sheet1.write(0, 0, 'ns')
sheet1.write(0, 1, 'min_calls')
sheet1.write(0, 2, 'avg_calls')
sheet1.write(0, 3, 'med_calls')
sheet1.write(0, 4, 'max_calls')
sheet1.write(0, 5, 'min_lookahead')
sheet1.write(0, 6, 'avg_lookahead')
sheet1.write(0, 7, 'med_lookahead')
sheet1.write(0, 8, 'max_lookahead')

num = 1

for n in range(num+2, 101):  # start at 3
    execSc = ["./a", "-x", str(n)]
    proc = subprocess.Popen(execSc, stdout=subprocess.PIPE)
    # catch the output from terminal
    output, error = proc.communicate()

    output = output.decode('utf-8').split()

    ns = int(output[0].strip())
    min_calls = float(output[1].strip())
    avg_calls = float(output[2].strip())
    med_calls = float(output[3].strip())
    max_calls = float(output[4].strip())
    min_lookahead = float(output[5].strip())
    avg_lookahead = float(output[6].strip())
    med_lookahead = float(output[7].strip())
    max_lookahead = float(output[8].strip())

    # write in table
    sheet1.write(num, 0, ns)
    sheet1.write(num, 1, min_calls)
    sheet1.write(num, 2, avg_calls)
    sheet1.write(num, 3, med_calls)
    sheet1.write(num, 4, max_calls)
    sheet1.write(num, 5, min_lookahead)
    sheet1.write(num, 6, avg_lookahead)
    sheet1.write(num, 7, med_lookahead)
    sheet1.write(num, 8, max_lookahead)

    num += 1


book.save('Results_continue.xlw')
```

Apêndice 3 – Programa de Matlab para criar os gráficos apresentados

```
1
2       figure(1)
3       hold on
4       plot(ns,min_calls, 'r')
5       plot(ns, avg_calls, 'k')
6       plot(ns, med_calls, 'g')
7       plot(ns, max_calls, 'b')
8       legend('min','avg','med','max')
9       xlabel('Número de símbolos na mensagem')
10      ylabel('Número de chamadas à função recursiva')
11
12      figure(2)
13      hold on
14      plot(ns,min_lookahead, 'r')
15      plot(ns, avg_lookahead, 'k')
16      plot(ns, med_lookahead, 'g')
17      plot(ns, max_lookahead, 'b')
18      legend('min','avg','med','max')
19      xlabel('Número de símbolos na mensagem')
20      ylabel('Número de símbolos errados encontrados')
```

Apêndice 4 – Tabela de dados

ns	min_calls	avg_calls	med_calls	max_calls	min_lookahead	avg_lookahead	med_lookahead	max_lookahead
3	1,1	1,501	1,516	1,66	1	1	1	1
4	1	1,624	1,756	1,999	-1	0,9	1	27
5	1,367	2,172	2,201	2,362	1	2,3	2	63
6	1,695	2,41	2,41	2,62	1	2	2	48
7	1,239	2,613	2,629	2,83	1	2,2	2	21
8	1	2,799	2,801	2,999	-1	2,2	2	39
9	2,493	2,973	2,986	3,176	1	2,8	2	61
10	2,585	3,118	3,121	3,318	1	2,5	2	216
11	2,845	3,251	3,254	3,462	1	2,8	2	22
12	3,011	3,373	3,371	3,59	1	2,6	2	38
13	3,182	3,486	3,481	3,709	1	2,6	2	22
14	3,267	3,589	3,588	3,801	1	2,7	2	21
15	3,363	3,688	3,685	3,9	1	2,3	2	22
16	3,43	3,784	3,784	3,988	1	2,4	2	12
17	3,552	3,87	3,871	4,069	1	2,3	2	13
18	3,645	3,952	3,959	4,145	1	2,6	2	22
19	2,067	4,029	4,04	4,235	1	2,6	2	133
20	3,808	4,103	4,11	4,314	1	2,9	2	22
21	3,901	4,171	4,178	4,384	1	3	2	20
22	3,986	4,236	4,242	4,439	1	3,1	3	23
23	4,079	4,297	4,304	4,488	1	2,9	2	23
24	4,153	4,354	4,357	4,526	1	3,2	3	22
25	4,233	4,411	4,415	4,585	1	3,1	3	28
26	4,289	4,467	4,468	4,63	1	3,1	3	17
27	4,346	4,522	4,525	4,686	1	2,9	3	15
28	4,411	4,575	4,577	4,74	1	2,7	2	20

29	4,441	4,626	4,631	4,8	1	2,5	2	12
30	4,487	4,676	4,68	4,849	1	2,6	2	17
31	4,527	4,723	4,728	4,896	1	2,6	2	16
32	4,577	4,77	4,772	4,936	1	2,5	2	13
33	4,618	4,815	4,818	4,986	1	2,6	2	17
34	4,679	4,86	4,865	5,028	1	2,5	2	20
35	4,72	4,903	4,908	5,071	1	2,3	2	14
36	4,762	4,944	4,951	5,108	1	2,6	2	19
37	4,809	4,984	4,988	5,142	1	2,4	2	18
38	4,856	5,024	5,027	5,181	1	2,3	2	16
39	4,908	5,062	5,068	5,215	1	3	3	23
40	4,927	5,098	5,104	5,251	1	2,7	2	25
41	4,963	5,132	5,137	5,276	1	2,9	3	17
42	4,998	5,166	5,17	5,296	1	2,9	2	26
43	5,035	5,199	5,204	5,318	1	3	3	26
44	5,074	5,232	5,236	5,352	1	2,8	2	21
45	5,092	5,263	5,266	5,392	1	3,3	3	18
46	5,134	5,294	5,296	5,416	1	3,3	3	36
47	5,18	5,324	5,328	5,454	1	2,8	2	22
48	5,214	5,353	5,353	5,488	1	3,3	3	21
49	5,24	5,381	5,384	5,516	1	3,2	3	21
50	5,262	5,408	5,406	5,548	1	3,1	3	21
51	5,301	5,436	5,435	5,576	1	3,1	3	22
52	5,314	5,464	5,462	5,613	1	3,3	3	20
53	5,322	5,49	5,491	5,635	1	3,1	2	14
54	5,346	5,517	5,519	5,656	1	3,1	3	26
55	5,369	5,543	5,547	5,674	1	2,9	2	39
56	5,397	5,57	5,572	5,706	1	2,9	2	13
57	5,428	5,595	5,595	5,723	1	2,9	3	13
58	5,463	5,622	5,622	5,747	1	3	3	19
59	5,494	5,647	5,647	5,773	1	2,8	3	14
60	5,521	5,671	5,672	5,797	1	2,7	2	25
61	5,549	5,696	5,697	5,816	1	2,7	2	15
62	5,577	5,72	5,721	5,835	1	2,7	2	20
63	5,605	5,745	5,747	5,848	1	2,7	2	14
64	5,633	5,768	5,77	5,874	1	2,5	2	24
65	5,664	5,791	5,791	5,896	1	2,6	2	14
66	5,685	5,814	5,813	5,917	1	2,6	2	16
67	5,704	5,836	5,837	5,939	1	2,7	2	19
68	5,722	5,859	5,861	5,963	1	2,3	2	14
69	5,749	5,879	5,878	5,982	1	2,7	3	15
70	5,773	5,901	5,9	5,996	1	2,2	2	15
71	5,799	5,922	5,921	6,016	1	2,7	2	12
72	5,825	5,944	5,942	6,037	1	2,6	2	19
73	5,849	5,963	5,961	6,053	1	2,6	2	17
74	5,866	5,983	5,982	6,08	1	2,9	3	20
75	5,888	6,003	6,002	6,101	1	2,9	3	13
76	5,911	6,021	6,021	6,116	1	2,4	2	21
77	5,933	6,04	6,039	6,139	1	2,6	2	24
78	5,95	6,058	6,058	6,153	1	2,5	2	23
79	5,965	6,077	6,075	6,169	1	2,7	2	13
80	5,982	6,094	6,093	6,184	1	2,5	2	18
81	5,993	6,112	6,113	6,198	1	2,8	2	13
82	6,011	6,129	6,128	6,212	1	3,2	3	21
83	6,018	6,146	6,144	6,226	1	2,8	2	27
84	6,033	6,162	6,16	6,244	1	2,9	2	18
85	6,054	6,179	6,18	6,257	1	3	2	16
86	6,068	6,196	6,196	6,275	1	3,2	3	20
87	6,096	6,211	6,21	6,293	1	3	2	16
88	6,118	6,228	6,225	6,314	1	3,4	3	28
89	6,135	6,243	6,242	6,332	1	3,1	3	35
90	6,155	6,258	6,258	6,338	1	3,3	3	28
91	6,169	6,274	6,274	6,355	1	2,9	3	16
92	6,193	6,289	6,289	6,37	1	3,3	3	27
93	6,201	6,304	6,304	6,383	1	3,1	3	20
94	6,221	6,318	6,317	6,397	1	3,4	3	29
95	6,239	6,332	6,33	6,417	1	3,4	3	23
96	6,25	6,348	6,345	6,425	1	3,2	3	29
97	6,268	6,362	6,359	6,443	1	2,9	2	18
98	6,284	6,376	6,375	6,45	1	3,7	3	26
99	6,301	6,39	6,388	6,463	1	3,5	3	24

100	6,32	6,404	6,401	6,481	1	2,9	2	19