Algoritmos e Estruturas de Dados Ano Letivo 2022/2023

universidade de aveiro deti departamento de electrónica, telecomunicações e informática

.
.
.

Trabalho 1
Speed_Run

Pedro Rei 107463, P1 *33,3%*
Cristiano Nicolau 108536, P4 *33,3%*
Tiago Fonseca 108615, P4 *33,3%*

Índice

1.Introdução

Este trabalho pratico surgiu no contexto da cadeira de Algoritmos e Sistemas de Dados, onde nos foi fornecido um ficheiro com diversos scripts sendo um deles, **speed_run.c,** onde primeiramente o objetivo era tentar melhor o script para fornecer o tempo de execução o mais depressa possível, para fazer isso, poderíamos usar o código já fornecido, acrescentando o necessário, ou criar a nossa própria função. O script **speed_run** está feito em linguagem *c*.

Com este trabalho, esperamos ficar mais familiarizados com a programação em linguagem *c*, e em todos os processos necessários para a realização deste trabalho prático.

2. Apresentação do problema

O problema apresentado consiste numa estrada, dividida em segmentos, nas quais passa um carro. Esse carro vai a uma certa velocidade que corresponde ao número de segmentos que avança, sendo que não pode ultrapassar o limite de velocidade presente em cada segmento. A velocidade máxima possível para o carro é 9 e a mínima é 1. Para isso o carro tem 3 opções: aumentar em 1 a sua velocidade, manter a velocidade ou diminuir em 1 a sua velocidade. O objetivo será o carro atravessar a estrada no número mínimo de passos possível, sem ultrapassar os limites de velocidade estabelecidos, de forma a chegar ao último segmento com velocidade 1. Essa mesma estrada pode ter diversos tamanhos e deve-se encontrar o caminho mais rápido em cada um deles.

3. Explicação das Solução dadas e respetivos resultados

Nesta parte do documento, serão apresentadas as soluções já fornecidas pelo professor, as soluções realizadas pelo nosso grupo e ainda os resultados obtidos.

Serão ainda apresentados documentos PDF com a melhor solução encontrada, contendo variáveis relativas ao número de movimentos, tempo de execução e as posições percorridas.

3.1 Solução dada pelo professor

Para este trabalho foi fornecida um script, *speed_run.c*, que nos fornecia uma função, onde esta era ineficiente e apresentava uma velocidade de execução muito lenta e insuficiente para apresentar resultados de uma estrada com 800 segmentos.

Dentro do *script* estava a função *solution_1_recursion* que recebia como dados o número de movimentos,*move_number*, a posição onde estaria o carro,*position*, a velocidade a que ia, *speed,* e a posição final, *final_position*.

Esta função verificava para todas as velocidades em cada posição se esta era a mais eficiente, de modo que fizesse o caminho no menor número de passos possível.

Inicialmente ela irá verificar, para a posição definida, se irá aumentar, reduzir ou manter a velocidade. Para isso, ele verifica se, para cada um dos casos, a nova velocidade que ele vai ter, *new_speed,* cumpre os requisitos, sendo eles: não ultrapassar a velocidade máxima, *_max_road_speed_* e somando ele à posição não ultrapassar o valor da posição final, *final_position*. Por último verifica-se se os segmentos nos quais o carro vai passar têm valor igual ou superior ao da *new_speed*.

Se tudo isto se verificar ele irá efetuar de novo a função onde a velocidade passa a ser o *new_speed*, a posição passa a ser a *position* anterior, e adiciona ao *move_number* +1. A partir daí será sempre feito este processo até chegar ao último segmento, *final_position,* com velocidade 1. Sendo esta uma função recursiva caso exista um caso onde nenhuma das mudanças de velocidade é possível ele pode voltar para o segmento de onde partiu anteriormente e testar para outras velocidades até atingir o objetivo.

3.2 Solução dada pelo professor melhorada(solution_1_recursion)

Ao correr a solução dada pelo professor reparamos num crescimento acentuado do tempo de execução para posições finais maiores, devido ao esforço necessário para percorrer todas as soluções possíveis.

Para reduzir o tempo de execução primeiramente, priorizamos o aumento da velocidade, ou seja, como o objetivo principal é chegar a posição final, *final_position*, com o menor numero de movimentos possíveis, *move_number,* então, ao testar as velocidades, faz mais sentido, incrementar a velocidade ao invés de a manter ou reduzir.

De seguida, podemos poupar uma chamada da função recursiva, uma vez que sabemos sempre que o primeiro movimento, que será passar de velocidade 0 para velocidade 1 e da posição 0 para posição 1, assim sendo podemos logo começar na posição 1, com velocidade 1 e numero de movimentos 1, ou seja, *speed*=1, *move_number*=1 e *position*=1.

Por fim, uma vez que é uma função recursiva, é desnecessário continuar a executar se já encontramos uma solução melhor, ou seja, se com o mesmo numero de movimentos, *move_number,* nos encontramos em uma posição, *position*, mais recuada do que em uma melhor solução já encontrada, é desnecessário continuar a testar esta solução.

3.2 Solução sem Recursão (solution_2_Nonrecursion)

Nesta solução, o nosso objetivo era criar uma solução sem recursividade, mas após diversos erros, não foi possível dar-lhe continuidade, mesmo assim chegamos ao consenso de a colocar no relatório de forma a enriquece-lo mostrando todo o trabalho que foi realizado pelo grupo.

Primeiramente a nossa ideia, era chegar a uma solução sem ser através de uma função recursiva mas sim através de um ciclo *for*. Após decidirmos como iria ser feita a função, percebemos que a única entrada necessária seria o posição final, *final_position*, então de seguida, criamos as variáveis necessárias, *speed*, *move_number, position.*

Após isto, iniciamos o ciclo *for* que iria percorrer todos os i ate a posição final, dentro deste ciclo *for* temos ainda umas condições *if* onde caso a velocidade fosse 0, passaria a 1; caso o i fosse igual a final_position-1 ou igual a final_position-2 , e a velocidade fosse diferente de 1 então a velocidade diminuiria 1, caso nenhuma destas condições fossem reais então, irá verificar, para a posição definida, se irá aumentar, reduzir ou manter a velocidade. Para isso, ele verifica se, para cada um dos casos, a nova velocidade que ele vai ter, *new_speed,* cumpre os requisitos, sendo eles: não ultrapassar a velocidade máxima, *_max_road_speed_* e somando ele à posição não ultrapassar o valor da posição final, *final_position*. Por último verifica-se se os segmentos nos quais o carro vai passar têm valor igual ou superior ao da *new_speed*. Caso passe em todos os testes, o ciclo for volta a correr, guardando os valores da velocidade, a posição sendo a posição anterior mais a velocidade, e incrementa +1 no move_number. Caso a posição seja igual a posição final e o speed=1 e se o numero de movimentos for menor do que a melhor solução já encontrada, a solução é guardada e o numero de movimentos também.

Por fim encontra-se ainda um ciclo, *do While,* que foi desenvolvido com a intenção de melhorar o desenvolvimento da função e para a correção de erros. Em que o objetivo era sempre que chegássemos a uma posição onde não era possível decrementar a velocidade para que a velocidade fosse valida nos segmentos da estrada ou seja, por exemplo, se a velocidade fosse 4 mas o salto não fosse valido pois a position+speed percorressem casas onde a velocidade seria 2, entraria neste ciclo.

O objetivo seria ir ao salto anterior e decrementar a velocidade ou seja quando chegasse a este salto, a velocidade seria 3 sendo possível decrementar a velocidade para 2. Caso não fosse possível iria a outro salto atrás, ate que fosse possível fazer os saltos. Com isto dentro deste ciclo calculamos a velocidade anterior, old_speed, através das *positions* com o *move_number* anterior, e entrava numa condição *if* onde a velocidade atual teria de ser maior ou igual a velocidade anterior, caso isso acontecesse, guardava os valores da velocidade, a posição sendo a posição anterior mais a velocidade, e incrementava +1 no *move_number*.

A nossa função solve_2 ao contrario da já fornecida solve_1 que chama a função recursiva com as entradas *move_number, speed, final_position* e *position*; chama a função não recursiva unicamente com a entrada *final_position*.

4. Resultados

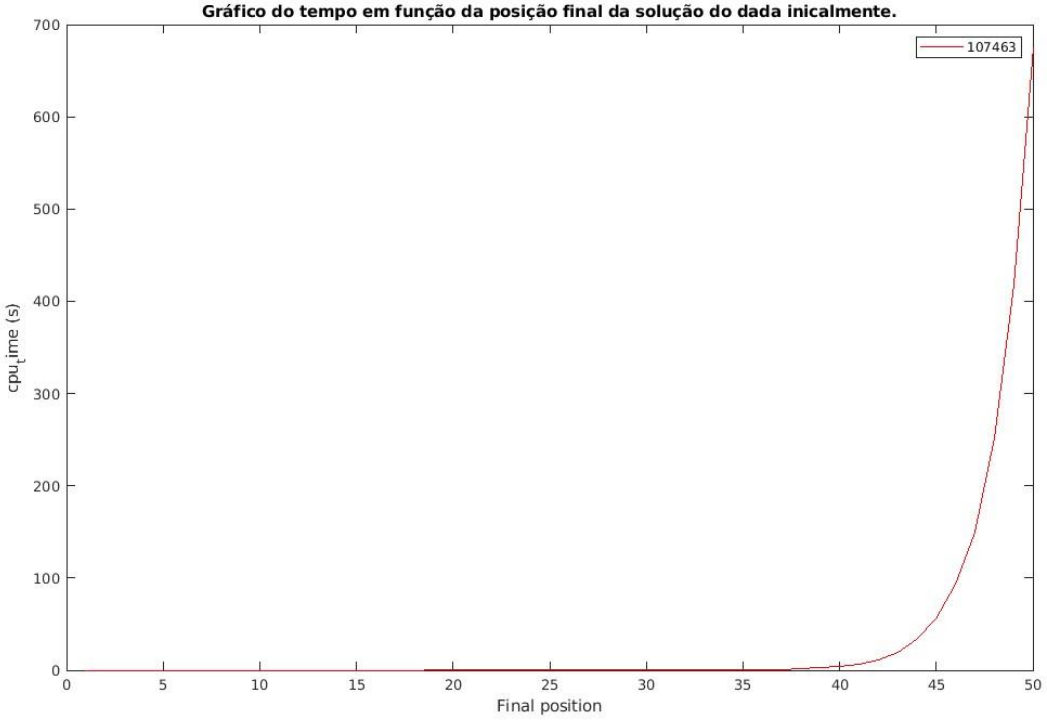
Apresentamos agora a parte com os resultados do script, tanto daquilo que aparece no terminal como de alguns dos PDFs que são criados.
Inicialmente temos o script original fornecido pelo stor que apresenta a seguinte tabela:

		plain recursion	
n	sol	count	cpu time
1	1	1	1.656e-06
2	2	2	1.115e-06
3	3	4	1.067e-06
4	3	7	1.186e-06
5	4	12	1.337e-06
6	4	21	1.692e-06
7	5	35	1.851e-06
8	5	59	2.474e-06
9	5	99	3.213e-06
10	6	166	4.993e-06
11	6	278	2.393e-06
12	6	464	3.047e-06
13	7	776	4.934e-06
14	7	1296	7.974e-06
15	7	2164	1.316e-05
16	7	3613	2.196e-05
17	8	6030	3.623e-05
18	8	10064	6.000e-05
19	8	16794	1.000e-04
20	8	28023	1.856e-04
21	9	46758	2.770e-04
22	9	78012	4.487e-04
23	9	130090	7.021e-04
24	9	216969	1.124e-03
25	9	359707	1.893e-03
26	10	597824	3.154e-03
27	10	995047	5.216e-03
28	10	1655499	9.279e-03
29	10	2757260	1.481e-02
30	10	4617502	2.409e-02
31	11	7716364	4.059e-02
32	11	12813314	6.703e-02
33	11	21329527	1.119e-01
34	12	35520858	1.846e-01
35	12	59124729	3.184e-01
36	12	98503441	5.132e-01

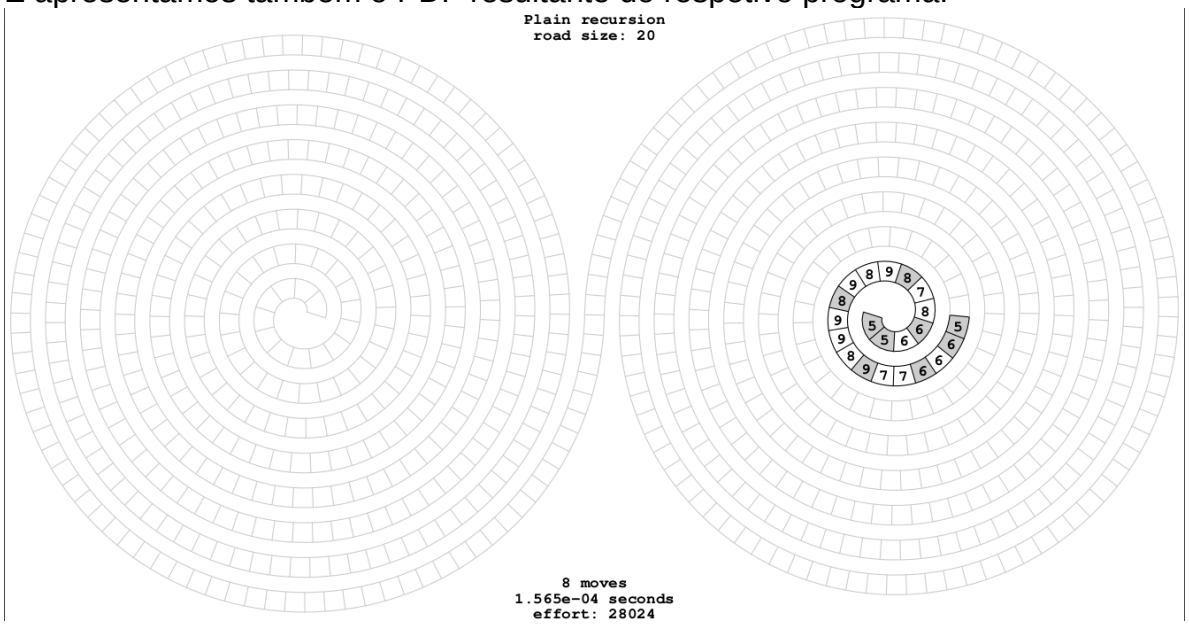
37	12	164103137	8.558e-01
38	13	274843839	1.445e+00
39	13	459339192	2.470e+00
40	13	766741706	4.061e+00
41	14	1280450790	6.817e+00
42	14	2137774080	1.147e+01
43	14	3567725220	1.946e+01
44	14	5919245371	3.404e+01
45	15	9841717643	5.630e+01
46	15	16384861943	9.362e+01
47	15	27264882096	1.496e+02
48	15	43602648571	2.521e+02
49	16	70820435199	4.180e+02
50	16	114375988302	6.761e+02

Através deste gráfico podemos observar também a relação entre o tempo de execução e o comprimento da estrada a percorrer.

Gráfico em Matlab para visualização dos resultados.



E apresentamos também o PDF resultante do respetivo programa:



Em seguida temos a nossa solução melhorada, em que inicialmente fomos apenas verificar para os diferentes números mecanográficos, mas fomos também comparar para os diferentes processadores as velocidades de execução.
Inicialmente temos a tabela para os números mecanográficos 107463, 108536 e 108615, respetivamente:

Para o 107463:

		plain recursion	
n	sol	count	cpu time
1	1	1	1.629e-06
2	2	2	1.016e-06
3	3	4	1.048e-06
4	3	4	1.016e-06
5	4	6	1.024e-06
6	4	7	1.117e-06
7	5	9	1.080e-06
8	5	11	1.278e-06
9	5	9	1.059e-06
10	6	12	1.277e-06
11	6	14	6.190e-07
12	6	12	2.690e-07
13	7	16	3.400e-07
14	7	19	3.460e-07
15	7	19	3.120e-07

16	7	15	2.880e-07
17	8	19	3.110e-07
18	8	22	3.130e-07
19	8	23	3.530e-07
20	8	20	3.290e-07
21	9	24	9.540e-07
22	9	28	7.340e-07
23	9	29	6.490e-07
24	9	26	5.820e-07
25	9	19	4.270e-07
26	10	22	4.700e-07
27	10	24	4.830e-07
28	10	23	4.220e-07
29	10	20	3.670e-07
30	11	22	3.860e-07
31	11	24	3.870e-07
32	11	22	3.500e-07
33	12	24	3.830e-07
34	12	26	3.820e-07
35	12	24	3.560e-07
36	13	26	3.740e-07
37	13	28	3.870e-07
38	13	26	3.600e-07
39	14	28	3.980e-07
40	14	30	4.150e-07
41	14	28	3.550e-07
42	15	31	4.250e-07
43	15	33	4.000e-07
44	15	31	4.110e-07
45	16	35	4.140e-07
46	16	37	4.630e-07
47	16	36	4.600e-07
48	16	32	4.100e-07
49	17	34	4.040e-07
50	17	35	4.510e-07
55	19	37	1.007e-06
60	22	43	7.130e-07
65	24	48	6.600e-07
70	25	51	6.150e-07
75	27	58	6.740e-07
80	28	68	7.830e-07
85	29	71	7.780e-07
90	30	64	9.290e-07
95	33	70	9.310e-07
100	35	74	8.460e-07
110	40	85	1.217e-06
120	43	95	1.053e-06
130	45	106	1.140e-06
140	47	111	1.172e-06
150	49	116	1.383e-06
160	52	109	1.334e-06

```
170 |  57          120 1.315e-06 |
180 |  60          132 1.338e-06 |
190 |  62          144 1.637e-06 |
200 |  64          150 1.679e-06 |
220 |  71          147 2.020e-06 |
240 |  78          169 1.801e-06 |
260 |  82          175 1.855e-06 |
280 |  89          186 1.671e-05 |
300 |  97          205 2.242e-06 |
320 | 102          216 2.375e-06 |
340 | 111          238 2.423e-06 |
360 | 116          248 2.481e-06 |
380 | 120          266 2.964e-06 |
400 | 129          276 2.831e-06 |
420 | 135          297 3.985e-06 |
440 | 139          296 4.203e-06 |
460 | 147          312 3.111e-06 |
480 | 152          336 3.359e-06 |
500 | 159          337 3.424e-06 |
520 | 167          357 3.597e-06 |
540 | 171          389 3.990e-06 |
560 | 178          378 3.917e-06 |
580 | 185          400 4.041e-06 |
600 | 189          418 5.036e-06 |
620 | 196          417 4.239e-06 |
640 | 204          438 4.352e-06 |
660 | 208          447 4.477e-06 |
680 | 215          458 4.526e-06 |
700 | 223          481 4.854e-06 |
720 | 227          484 4.748e-06 |
740 | 236          503 4.813e-06 |
760 | 241          528 6.058e-06 |
780 | 245          529 5.255e-06 |
800 | 253          539 5.304e-06 |
--- + --- ---------- --------- +
```

Para o 108536:

```
    + --- ---------- --------- +
    |          plain recursion |
--- + --- ---------- --------- +
  n | sol       count  cpu time |
--- + --- ---------- --------- +
  1 |   1           1 1.391e-06 |
  2 |   2           2 1.476e-06 |
  3 |   3           4 1.425e-06 |
  4 |   3           4 1.072e-06 |
  5 |   4           6 1.400e-06 |
  6 |   4           7 1.297e-06 |
  7 |   5           9 1.465e-06 |
  8 |   5          11 1.767e-06 |
  9 |   5           9 1.194e-06 |
```

10	6	12	1.214e-06
11	6	14	7.920e-07
12	6	12	4.220e-07
13	7	16	3.840e-07
14	7	19	4.240e-07
15	7	19	3.880e-07
16	7	15	3.630e-07
17	8	19	3.780e-07
18	8	22	3.700e-07
19	8	23	4.360e-07
20	8	19	4.060e-07
21	9	23	8.790e-07
22	9	27	4.990e-07
23	9	28	4.620e-07
24	9	25	4.410e-07
25	9	18	3.780e-07
26	10	22	3.700e-07
27	10	25	3.890e-07
28	10	24	4.080e-07
29	10	20	3.500e-07
30	11	24	3.550e-07
31	11	26	3.980e-07
32	11	25	3.890e-07
33	11	21	3.130e-07
34	12	23	3.470e-07
35	12	25	3.450e-07
36	12	23	3.270e-07
37	13	25	3.810e-07
38	13	27	3.740e-07
39	13	25	3.390e-07
40	14	27	3.780e-07
41	14	29	3.720e-07
42	14	27	3.480e-07
43	15	29	3.850e-07
44	15	31	4.070e-07
45	15	29	2.848e-06
46	16	31	4.010e-07
47	16	33	4.130e-07
48	16	31	3.710e-07
49	17	33	3.970e-07
50	17	34	4.580e-07
55	19	36	1.397e-06
60	22	42	7.570e-07
65	24	49	7.680e-07
70	26	54	7.010e-07
75	27	60	7.860e-07
80	28	69	9.020e-07
85	29	72	8.980e-07
90	30	66	1.048e-06
95	32	67	1.074e-06
100	35	73	9.280e-07

```
110 |  39             82 1.818e-06 |
120 |  42             96 1.132e-06 |
130 |  44            105 1.255e-06 |
140 |  46            109 1.154e-06 |
150 |  48            109 1.240e-06 |
160 |  53            111 1.279e-06 |
170 |  58            121 1.362e-06 |
180 |  61            134 1.397e-06 |
190 |  63            138 1.398e-06 |
200 |  65            150 1.729e-06 |
220 |  73            151 2.798e-06 |
240 |  81            172 1.975e-06 |
260 |  85            180 1.988e-06 |
280 |  91            187 2.038e-06 |
300 |  98            208 2.213e-06 |
320 | 102            211 2.299e-06 |
340 | 111            233 3.977e-06 |
360 | 116            243 2.498e-06 |
380 | 120            264 2.939e-06 |
400 | 129            267 2.753e-06 |
420 | 134            292 5.322e-06 |
440 | 139            294 3.243e-06 |
460 | 147            304 3.198e-06 |
480 | 152            329 3.212e-06 |
500 | 158            326 4.504e-06 |
520 | 165            342 3.487e-06 |
540 | 170            372 3.819e-06 |
560 | 176            363 3.680e-06 |
580 | 183            386 3.899e-06 |
600 | 187            403 5.880e-06 |
620 | 193            398 3.997e-06 |
640 | 201            422 4.105e-06 |
660 | 205            425 4.156e-06 |
680 | 212            437 6.765e-06 |
700 | 220            460 4.513e-06 |
720 | 225            466 5.538e-06 |
740 | 234            488 5.805e-06 |
760 | 239            502 4.843e-06 |
780 | 243            505 5.039e-06 |
800 | 252            524 5.138e-06 |
--- + --- ---------- --------- +
```

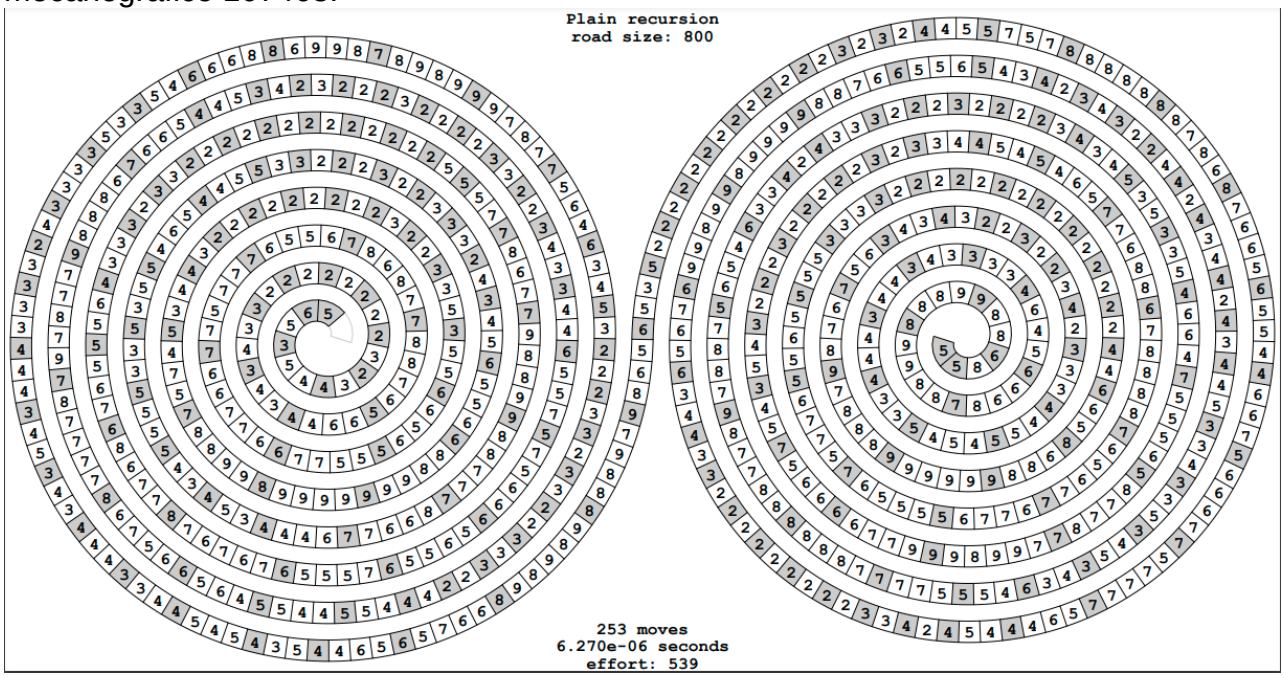
Para o 108615:

```
    + --- ---------- --------- +
    |              plain recursion |
--- + --- ---------- --------- +
  n | sol          count  cpu time |
--- + --- ---------- --------- +
  1 |   1              1 1.611e-06 |
  2 |   2              2 1.064e-06 |
  3 |   3              4 1.134e-06 |
```

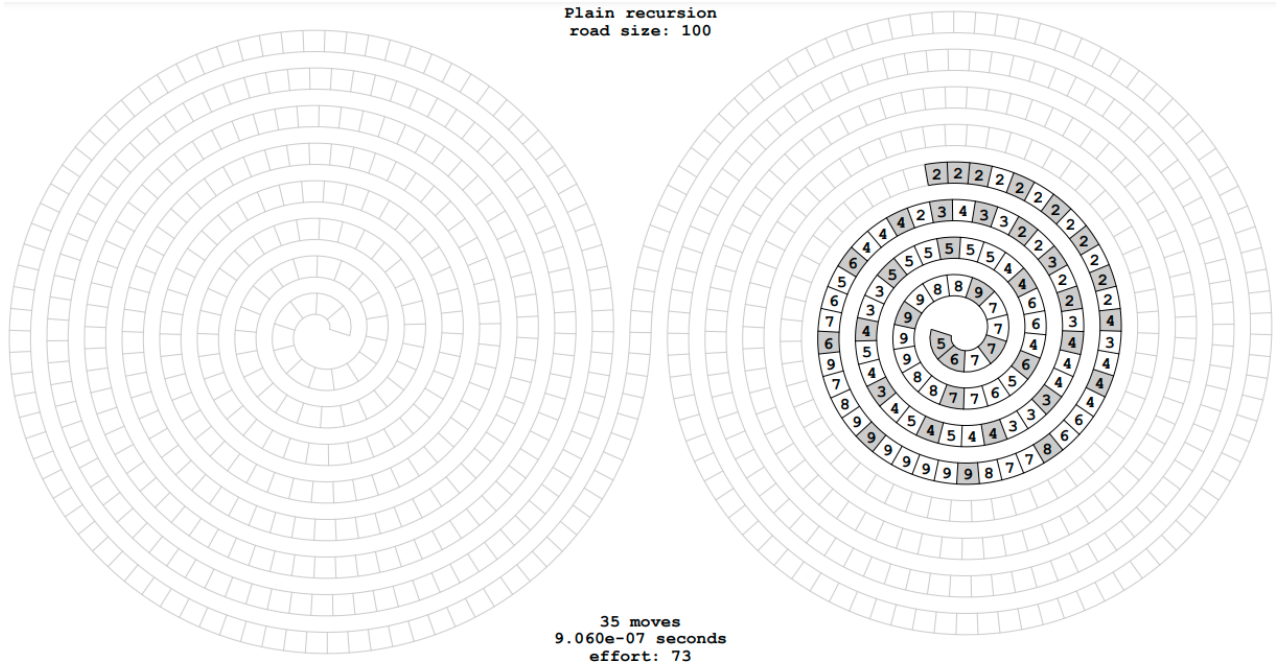
4	3	4	1.089e-06
5	4	6	1.083e-06
6	4	7	1.111e-06
7	5	9	1.125e-06
8	5	11	1.237e-06
9	5	9	1.123e-06
10	6	12	1.225e-06
11	6	14	6.580e-07
12	6	12	3.220e-07
13	7	16	4.100e-07
14	7	19	3.930e-07
15	7	19	3.750e-07
16	7	15	3.320e-07
17	8	19	3.940e-07
18	8	22	3.680e-07
19	8	23	4.720e-07
20	8	20	4.000e-07
21	9	25	6.810e-07
22	9	29	6.550e-07
23	9	30	5.710e-07
24	9	28	5.670e-07
25	9	21	4.720e-07
26	10	24	6.240e-07
27	10	27	6.180e-07
28	10	26	6.590e-07
29	10	24	6.070e-07
30	10	20	4.110e-07
31	11	23	5.240e-07
32	11	25	4.570e-07
33	11	23	6.970e-07
34	12	26	5.780e-07
35	12	28	6.190e-07
36	12	27	3.900e-07
37	12	24	3.600e-07
38	13	27	3.650e-07
39	13	29	3.770e-07
40	13	27	3.640e-07
41	14	31	3.870e-07
42	14	34	4.140e-07
43	14	33	4.240e-07
44	14	29	3.800e-07
45	15	32	3.970e-07
46	15	34	4.210e-07
47	15	33	4.190e-07
48	15	29	3.890e-07
49	16	31	4.340e-07
50	16	31	4.440e-07
55	19	37	1.271e-06
60	21	43	1.056e-06
65	23	48	6.870e-07
70	25	53	7.010e-07

```
 75 |  26            59 7.030e-07 |
 80 |  27            68 9.050e-07 |
 85 |  28            71 9.000e-07 |
 90 |  29            66 9.920e-07 |
 95 |  31            66 9.280e-07 |
100 |  34             72 9.030e-07 |
110 |  39             83 1.370e-06 |
120 |  42             93 1.036e-06 |
130 |  44            104 1.132e-06 |
140 |  46            109 1.118e-06 |
150 |  48            112 1.346e-06 |
160 |  51            106 1.258e-06 |
170 |  56            117 1.233e-06 |
180 |  59            129 1.326e-06 |
190 |  61            141 1.649e-06 |
200 |  63            149 1.692e-06 |
220 |  70            144 1.331e-05 |
240 |  78            168 1.979e-06 |
260 |  82            173 1.951e-06 |
280 |  88            180 2.057e-06 |
300 |  96            203 2.258e-06 |
320 | 100            207 2.301e-06 |
340 | 109            229 2.431e-06 |
360 | 114            239 2.529e-06 |
380 | 118            258 2.972e-06 |
400 | 127            262 3.767e-06 |
420 | 132            279 4.814e-06 |
440 | 137            283 3.159e-06 |
460 | 145            299 3.112e-06 |
480 | 151            321 3.280e-06 |
500 | 157            323 3.208e-06 |
520 | 165            341 3.409e-06 |
540 | 170            373 3.798e-06 |
560 | 177            368 5.015e-06 |
580 | 184            390 4.008e-06 |
600 | 187            399 3.982e-06 |
620 | 194            408 4.187e-06 |
640 | 202            432 4.322e-06 |
660 | 206            435 4.382e-06 |
680 | 213            446 4.378e-06 |
700 | 221            465 4.541e-06 |
720 | 226            480 5.779e-06 |
740 | 236            502 4.966e-06 |
760 | 240            514 4.940e-06 |
780 | 244            520 5.123e-06 |
800 | 254            543 6.373e-06 |
--- + --- ---------- --------- +
```

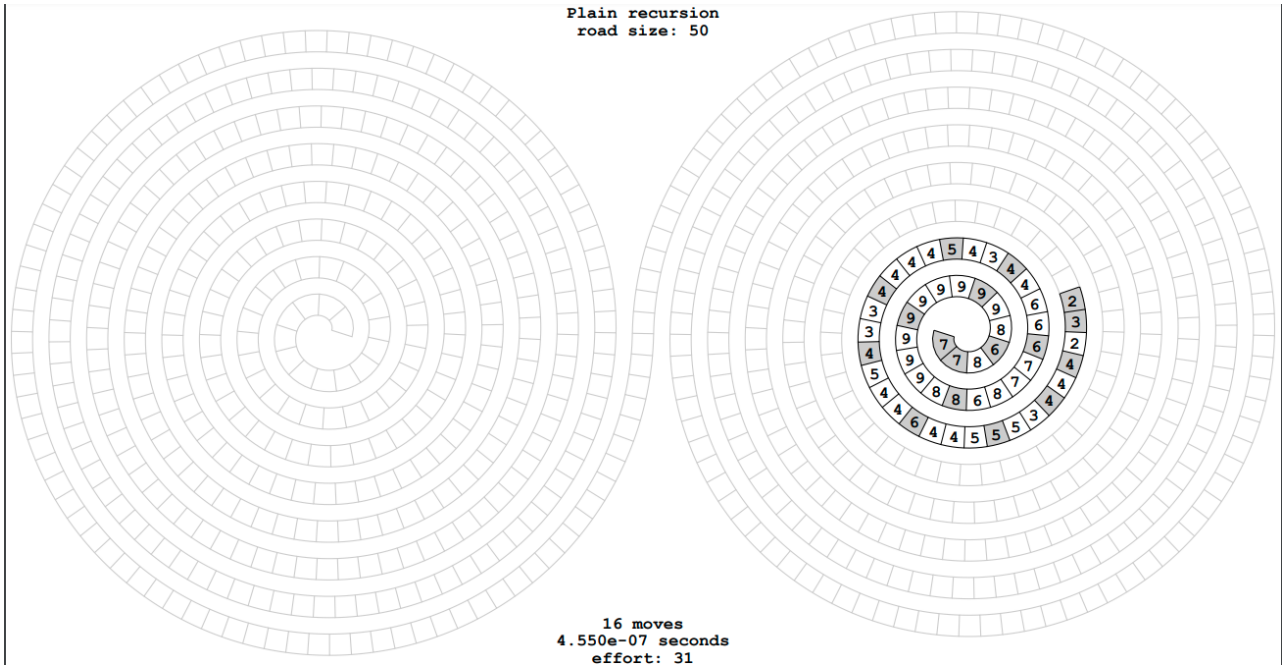
O resultado obtido em PDF foi bem sucedido em todos os casos, sendo que em baixo está apresentado um dos casos com o tamanho da estrada 800,neste caso para o número mecanográfico 107463:



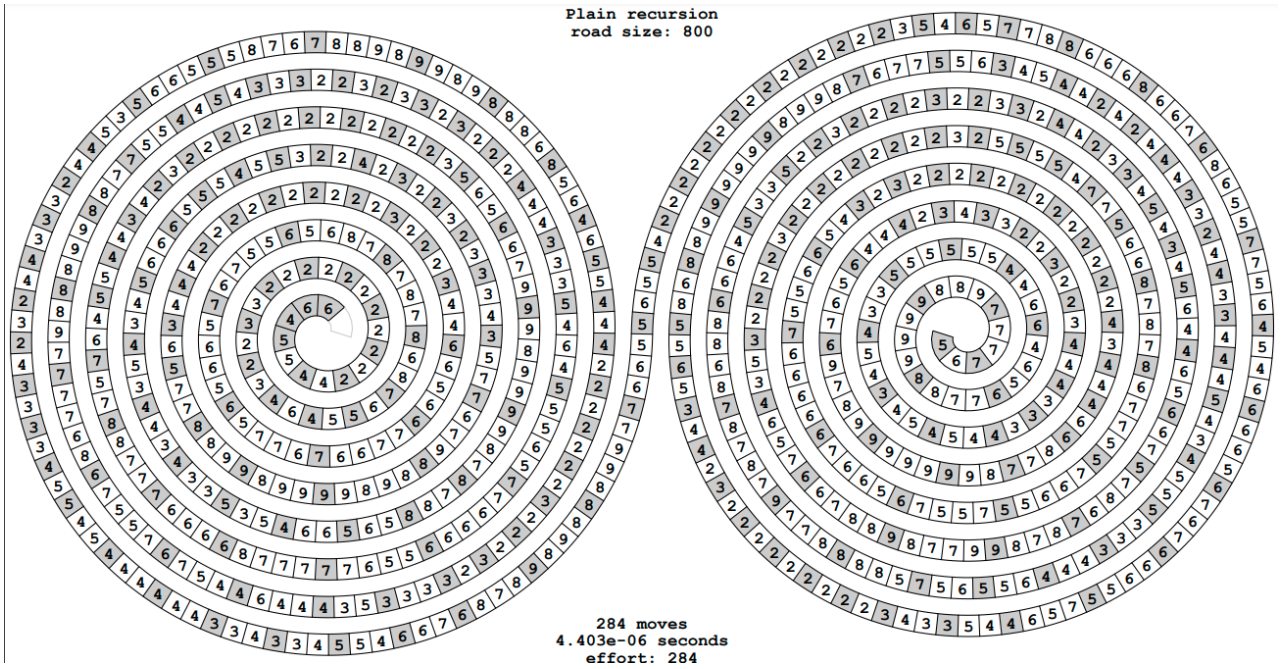
Outro exemplo, desta vez para o número mecanográfico 108536, é apresentado com tamanho da estrada 100:



Por último temos para o número mecanográfico 108615 um exemplo com tamanho da estrada de 50:



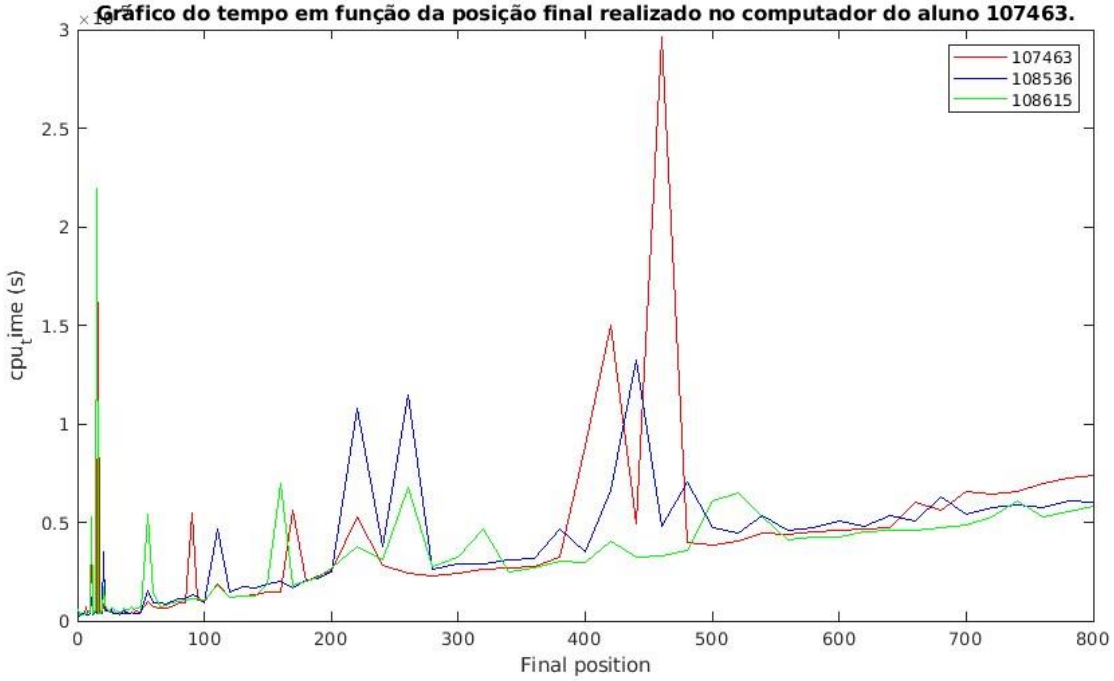
Para concluir temos a solução não recursiva referida anteriormente que, apesar de não ter sido concluída devido a diversos erros, teve resultados para um dos números mecanográficos, pelo que irá também ser apresentado o PDF resultante para uma estrada de tamanho 800:



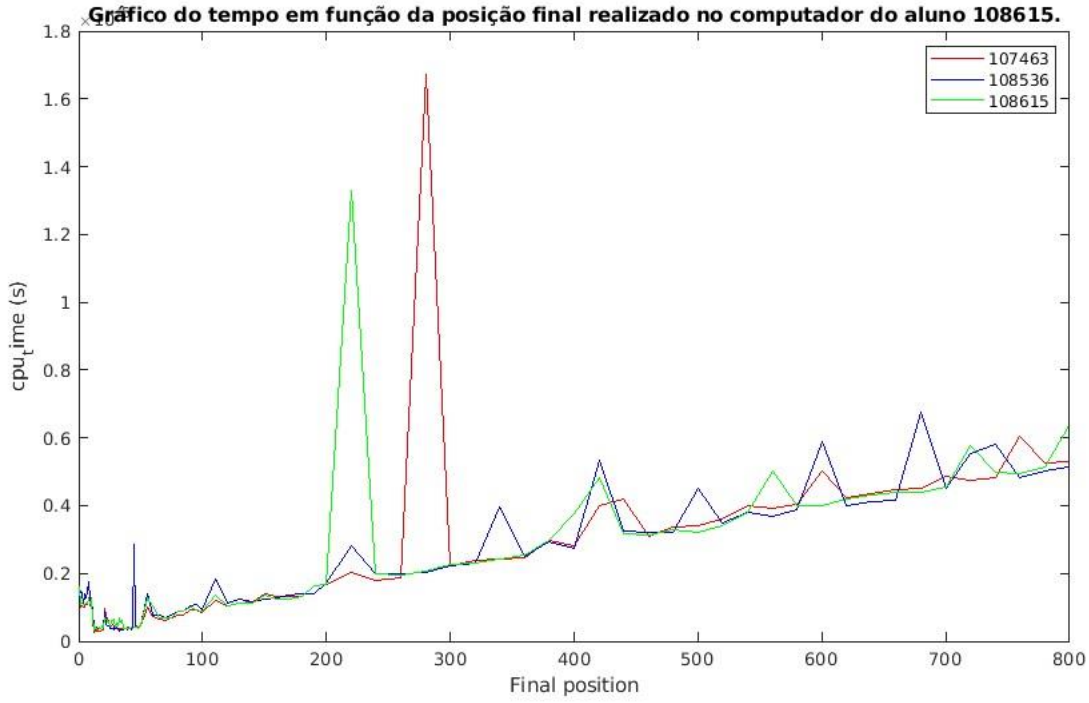
Resultados obtidos em Gráficos MatLab para a solução 1 (*solution_1_recursion*)

Inicialmente, testámos e corremos o script em cada um dos computadores dos constituintes do grupo com os diferentes números mecanográficos para assim podermos compara tempos de execução por número mecanográfico.

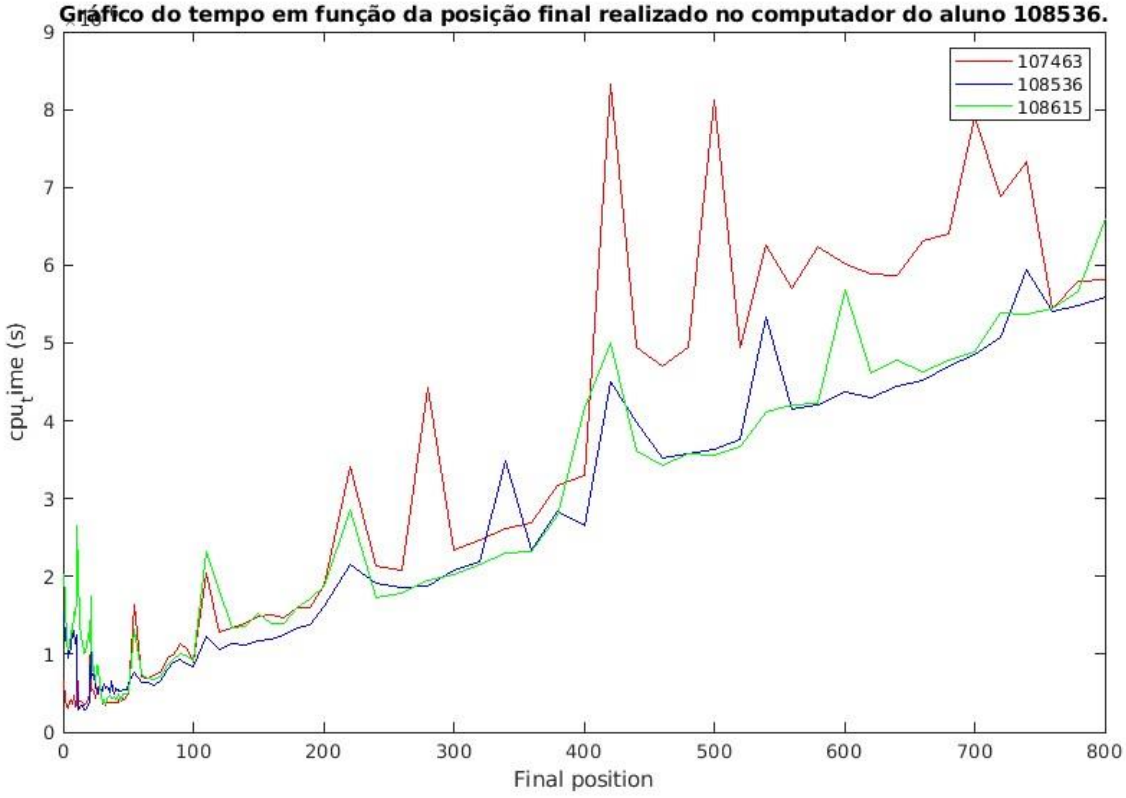
Resultado obtido para o computador do número mecanográfico 107463:



Resultado obtido para o computador do número mecanográfico 108615:

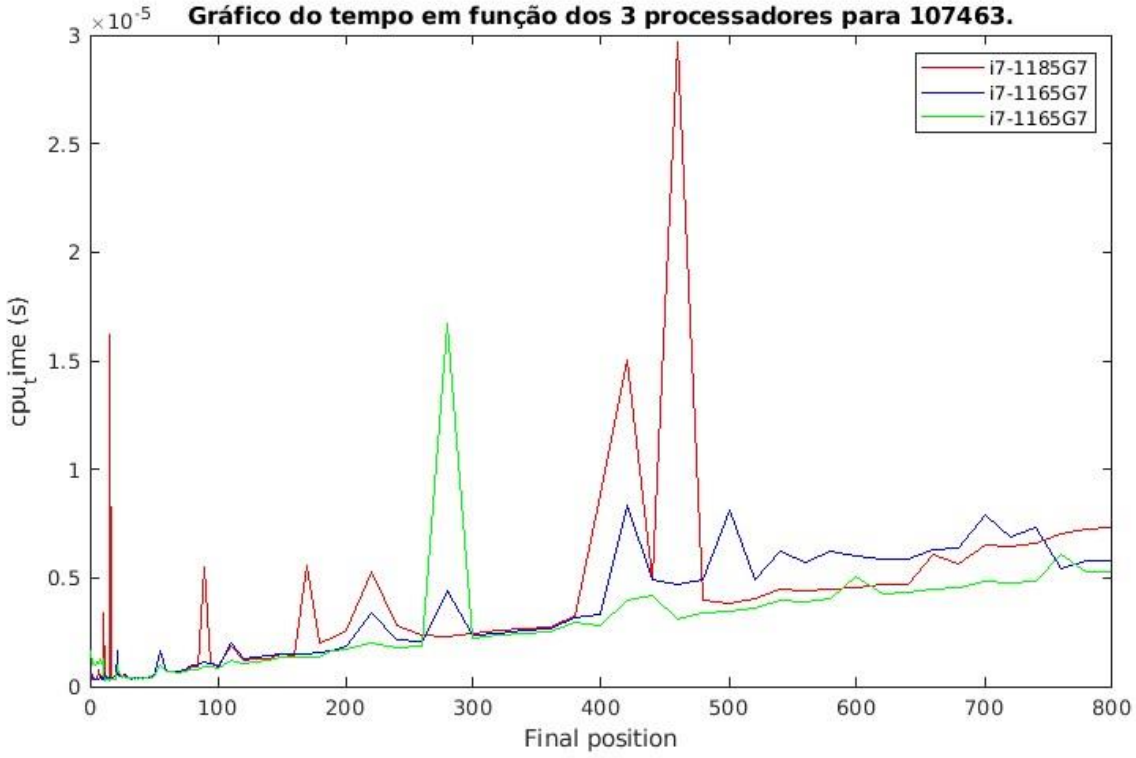


Resultado obtido para o computador do número mecanográfico 108536:

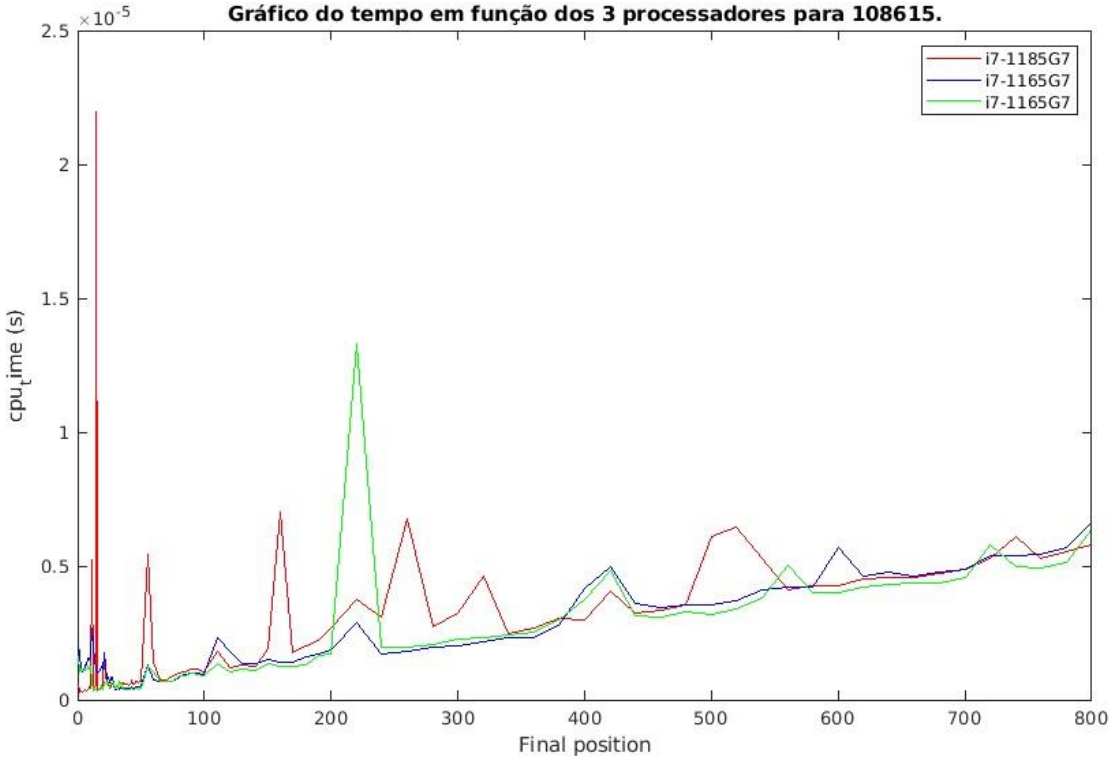


-Para comparar os diferentes tempos de execução para os mesmos processadores, juntámos os tempos de cada um dos computadores e fizemos um gráfico que nos permitiu comparar os tempos de execução para cada processador com o mesmo número mecanográfico.

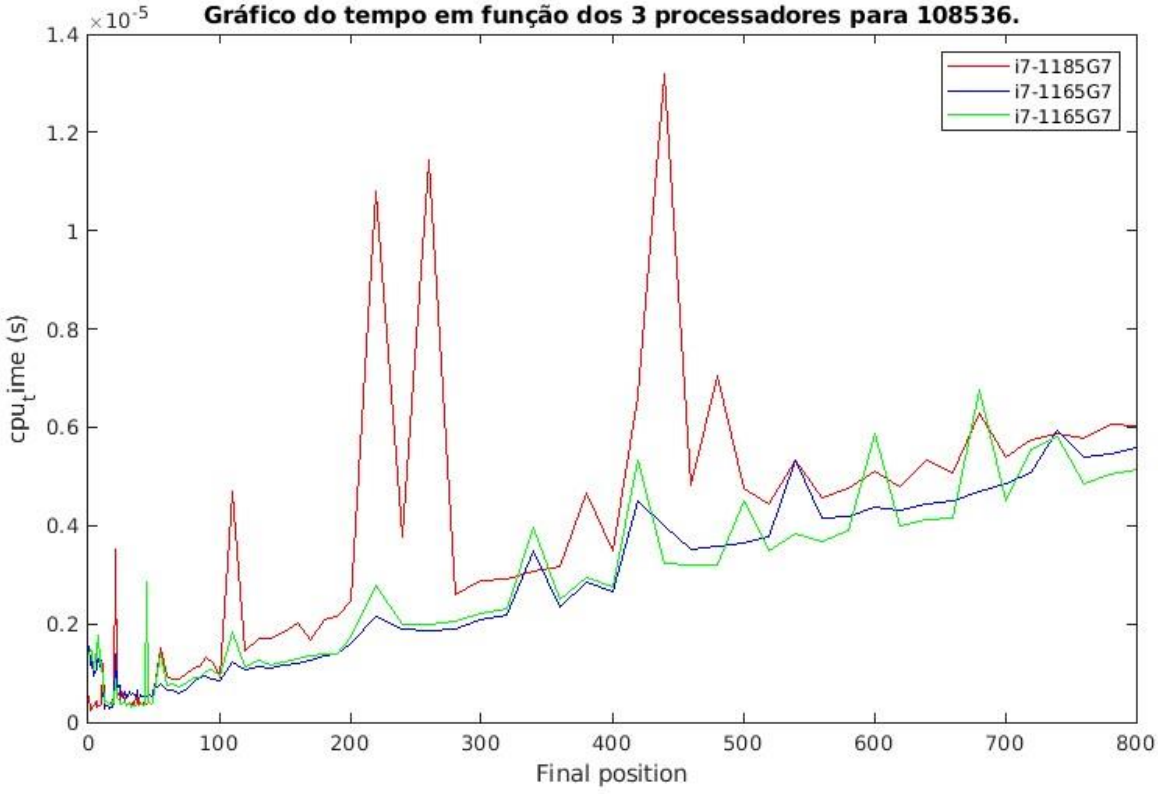
Para o processador "i7-1185G7" e para o número mecanográfico 107463:



Para o processador "i7-1165G7" e para o número mecanográfico 108615:



Para o processador "i7-1165G7" e para o número mecanográfico 108536:



Com estas comparações conseguimos comparar o desempenho dos vários processadores para a mesma solução encontrada, conseguindo assim ter uma noção da performance de cada um dadas as suas características.

5.Conclusão

Na nossa opinião o trabalho desenvolvido pelo grupo cumpriu os objetivos propostos num tempo de execução substancialmente inferior ao tempo original. Todo este trabalho mostrou em como não é só importante conseguir obter o programa funcional mas sim também torná-lo o mais eficiente possível porque em certos casos, como é o caso do script speed_run.c, a velocidade de execução pode não ser suficiente para o tempo que é pretendido.

Para além disso consideramos que a realização deste trabalho foi útil para compreender formas de otimização do nosso código e encontrar soluções alternativas mais eficientes. O facto de analisarmos durante bastante tempo o código fornecido ajudou também na compreensão cada vez mais completa da linguagem C e as alterações por nós feitas permitiu pôr na prática os nossos conhecimentos apreendidos nas aulas teóricas.

Durante a realização do trabalho enfrentámos algumas dúvidas e percalços, no entanto estas foram sempre esclarecidas pelo nosso professor durante as aulas práticas.

Concluindo, consideramos que alcançámos os objetivos propostos pelo professor e estamos bastante satisfeitos com os resultados finais obtidos.

Código speed_run

```
//
// AED, August 2022 (Tomás Oliveira e Silva)
//
// First practical assignement (speed run)
//
// Compile using either
//   cc -Wall -O2 -D_use_zlib_=0 solution_speed_run.c -lm
// or
//   cc -Wall -O2 -D_use_zlib_=1 solution_speed_run.c -lm -lz
//
// Place your student numbers and names here
//
//
//
// static configuration
//
#define _max_road_size_  800  // the maximum problem size
#define _min_road_speed_   2  // must not be smaller than 1, shouldnot be smaller than 2
#define _max_road_speed_   9  // must not be larger than 9 (only because of the PDF figure)
//
// include files --- as this is a small project, we include the PDF generation code directly from make_custom_pdf.c
//
#include <math.h>
#include <stdio.h>
#include "../P02/elapsed_time.h"
#include "make_custom_pdf.c"
#include <stdbool.h>
//
// road stuff
//
static int max_road_speed[1 + _max_road_size_]; // positions 0.._max_road_size_

static void init_road_speeds(void)
{
  double speed;
  int i;

  for(i = 0;i <= _max_road_size_;i++)
  {
    speed = (double)_max_road_speed_ * (0.55 + 0.30 * sin(0.11 * (double)i) + 0.10 * sin(0.17 * (double)i + 1.0) + 0.15 * sin(0.19 *
(double)i));
    max_road_speed[i] = (int)floor(0.5 + speed) + (int)((unsigned int)random() % 3u) - 1;
    if(max_road_speed[i] < _min_road_speed_)
      max_road_speed[i] = _min_road_speed_;
    if(max_road_speed[i] > _max_road_speed_)
      max_road_speed[i] = _max_road_speed_;
  }
}

//
// description of a solution
//

typedef struct
{
  int n_moves;                        // the number of moves (the number of positions is one more than the number of moves)
  int positions[1 + _max_road_size_];  // the positions (the first one must be zero)
}
solution_t;


//
// the (very inefficient) recursive solution given to the students
//

static solution_t solution_1,solution_1_best;
static double solution_1_elapsed_time; // time it took to solve the problem
static unsigned long solution_1_count; // effort dispended solving the problem

/* Nossa solucao nao recursiva contem  um erro onde so encontra resultado para um elemento do grupo 108536

static void solution_2_Nonrecursion( int final_position)
{
  int i;
```

```
  int flag;
  int move_number=0;
  int position=0;
  int speed=0;
  for (i = 0; i <=final_position +1  ; i++){

    int new_speed=speed;
    int j,w,q;
    flag = 0;
      if (speed == 0){
        speed=1;
        flag=1;
      } else if ( i == final_position - 1){
          if (speed!=1){
            speed=speed-1;
            flag=1;
          }
      } else if ( i == final_position - 2){
          if (speed!=1){
            speed=speed-1;
            flag=1;
          }
      } else{
        for(new_speed = speed+1;new_speed>=speed-1 && flag==0;new_speed--){\
          if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position ){
            for (j = 0; j <= new_speed && new_speed<=max_road_speed[position+j]; j++){
              ;
            }
            for (w= 0; w <= new_speed && new_speed<=max_road_speed[position+w+new_speed]; w++){
              ;
            }
            for (q = 0; q <= new_speed && new_speed<=max_road_speed[position+q+new_speed+new_speed]; q++){
              ;
            }

            if (j>= new_speed&& w >= new_speed && q>= new_speed){
              flag=1;
              speed=new_speed;
              move_number=move_number+1;
              position=position+new_speed;
              solution_1_count++;
              solution_1.positions[move_number] = position;
            }
          }
        }
    if(position == final_position && speed == 1){
      if(move_number < solution_1_best.n_moves){
        solution_1_best = solution_1;
        solution_1_best.n_moves = move_number;
    }
    return;
  }
    }
  }
}
   *//*
    do
    {
      int old_speed;

      move_number-=1;
      position=solution_1.positions[move_number];
      //testa se speed pode dimuir
      speed=solution_1.positions[move_number+1]-solution_1.positions[move_number];
      old_speed=solution_1.positions[move_number]-solution_1.positions[move_number-1]; //com speed  isto nao pode ser mais pequeno
que  o speed-1
      //se for legal diminui speed e avança
      //senao for da mais um salto atras
      if (speed-1>=old_speed-1){
        flag=1;
        position=position+speed;
        move_number=move_number+1;
        solution_1_count++;
        solution_1.positions[move_number] = position;
      }
    } while (flag!=1);
```

```
*//*

static void solve_2(int final_position)
{
  if(final_position < 1 || final_position > _max_road_size_)
  {
    fprintf(stderr,"solve_1: bad final_position\n");
    exit(1);
  }
  solution_1_elapsed_time = cpu_time();
  solution_1_count = 0ul;
  solution_1_best.n_moves = final_position + 100;
  solution_1_recursion(final_position);
  solution_1_elapsed_time = cpu_time() - solution_1_elapsed_time;
}

////////////////////////////////////////////////////////
*/
static void solution_1_recursion(int move_number,int position,int speed,int final_position)
{
  int i, new_speed;

  solution_1_count++;
  solution_1.positions[move_number] = position;

  if(position == final_position && speed == 1)
  {
    if(move_number < solution_1_best.n_moves)
    {
      solution_1_best = solution_1;
      solution_1_best.n_moves = move_number;
    }
    return;
  }

  if (solution_1_best.positions[move_number] >solution_1.positions[move_number]) return;
    for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--){// "for" serve para verificar para >,< ou = velocidade
      if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position ){ //hipotese: para igual position
deve ter sempre menos move_number
        for(i = 0;i <= new_speed && new_speed <= max_road_speed[position + i];i++){ // "for" serve para verificar se cumpre o limite para
cada posição seguinte
          ;
        }
        if(i > new_speed)
          solution_1_recursion(move_number + 1,position + new_speed,new_speed,final_position);
        }
      }
}

static void solve_1(int final_position)
{
  if(final_position < 1 || final_position > _max_road_size_)
  {
    fprintf(stderr,"solve_1: bad final_position\n");
    exit(1);
  }
  solution_1_elapsed_time = cpu_time();
  solution_1_count = 0ul;
  solution_1_best.n_moves = final_position + 100;
  solution_1_recursion(1,1,1,final_position); //aumenta sempre de velocidade no 1º logo
  solution_1_elapsed_time = cpu_time() - solution_1_elapsed_time;
}

//
// example of the slides
//

static void example(void)
{
  int i,final_position;

  srandom(0xAED2022);
  init_road_speeds();
  final_position = 30;
  solve_1(final_position);
```

```
make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],solution_1_best.n_moves,&solution_1_best.positions[0],soluti
on_1_elapsed_time,solution_1_count,"Plain recursion");
  printf("mad road speeds:");
  for(i = 0;i <= final_position;i++)
    printf(" %d",max_road_speed[i]);
  printf("\n");
  printf("positions:");
  for(i = 0;i <= solution_1_best.n_moves;i++)
    printf(" %d",solution_1_best.positions[i]);
  printf("\n");
}
//
// main program
//

int main(int argc,char *argv[argc + 1])
{
# define _time_limit_  3600.0
  int n_mec,final_position,print_this_one;
  char file_name[64];

  // generate the example data
  if(argc == 2 && argv[1][0] == '-' && argv[1][1] == 'e' && argv[1][2] == 'x')
  {
    example();
    return 0;
  }
  // initialization
  n_mec = (argc < 2) ? 0xAED2022    : atoi(argv[1]);
  srandom((unsigned int)n_mec);
  init_road_speeds();
  // run all solution methods for all interesting sizes of the problem
  final_position = 1;
  solution_1_elapsed_time = 0.0;
  printf("    + --- ---------- --------- +\n");
  printf("    |               plain recursion |\n");
  printf("--- + --- ---------- --------- +\n");
  printf("  n | sol           count  cpu time |\n");
  printf("--- + --- ---------- --------- +\n");
  while(final_position <= _max_road_size_/* && final_position <= 20*/)
  {
    print_this_one = (final_position == 10 || final_position == 20 || final_position == 50 || final_position == 100 || final_position == 200 ||
final_position == 400 || final_position == 800) ? 1 : 0;
    printf("%3d |",final_position);
    // first solution method (very bad)
    if(solution_1_elapsed_time < _time_limit_)
    {
      solve_1(final_position);
      if(print_this_one != 0)
      {
        sprintf(file_name,"%03d_1.pdf",final_position);

make_custom_pdf_file(file_name,final_position,&max_road_speed[0],solution_1_best.n_moves,&solution_1_best.positions[0],solution_
1_elapsed_time,solution_1_count,"Plain recursion");
      }
      printf(" %3d %16lu %9.3e |",solution_1_best.n_moves,solution_1_count,solution_1_elapsed_time);
    }
    else
    {
      solution_1_best.n_moves = -1;
      printf("                          |");
    }
    // second solution method (less bad)
    // ...

    // done
    printf("\n");
    fflush(stdout);
    // new final_position
    if(final_position < 50)
      final_position += 1;
    else if(final_position < 100)
      final_position += 5;
    else if(final_position < 200)
      final_position += 10;
```

```
    else
      final_position += 20;
  }
  printf("--- + --- ---------- --------- +\n");
  return 0;
# undef _time_limit_
}
```

Código MatLab usado.

Código usado para comparar os tempo de execução por número mecanográfico:

```
n =1 ;
cpu_time = 4;
107463 =[];   #Neste array colocamos os valores do terminal para o numero mecanográfico correspondente.
108536 =[];   #Neste array colocamos os valores do terminal para o numero mecanográfico correspondente.
108615 =[];  #Neste array colocamos os valores do terminal para o numero mecanográfico correspondente.
figure(1);
plot(107463(:,n), 107463 (:,cpu_time),'r', 108536 (:,n), 108536 (:,cpu_time),'b' ,  108615:,n), 108615
(:,cpu_time),'green')
ylabel("cpu_time (s)")
xlabel("Final position")
legend('107463', '108536', '108615') %Depois substituir por CPUs talvez
title("Gráfico do tempo em função da posição final realizado no computador do aluno 108536.")
```

Código usado para comparar os tempo de execução por processador:

```
n =1 ;
cpu_time = 4;
i7-1185G7 =[];   #Neste array colocamos os valores do terminal para o numero mecanográfico a comparar no
processador referente .
I7-1165G7 =[];   #Neste array colocamos os valores do terminal para o numero mecanográfico a comparar no
processador referente .
I7-1165G7 =[];  #Neste array colocamos os valores do terminal para o numero mecanográfico a comparar no
processador referente .
figure(1);
plot(i7-1185G7(:,n), i7-1185G7 (:,cpu_time),'r', I7-1165G7 (:,n), I7-1165G7 (:,cpu_time),'b' , I7-1165G7:,n), I7-1165G7
(:,cpu_time),'green')
ylabel("cpu_time (s)")
xlabel("Final position")
legend('107463', '108536', '108615')
title("Gráfico do tempo em função dos 3 processadores para xxxxxx.") #substituir para o numero mecanográfico
correspondente.
```

6.Bibliografia

Para a realização do trabalho foi principalmente consultado o PowerPoint relativo à programação em C disponível na página do e-learning da unidade curricular. Foi também utilizado conteúdo obtido em sites para tirar dúvidas pontuais.

Sites vistos:

- https://stackoverflow.com/
- https://www.javatpoint.com/