

Algoritmos e Estrutura de Dados - Speed Run

Universidade de Aveiro

*Moves, Speed, Positions, Dynamic
Programming, Execution Time*

David Palricas, Eduardo Alves, Inês Santos



Algoritmos e Estrutura de Dados - Speed Run

Universidade de Aveiro

(108780) David Palricas (104179) Eduardo Alves
davidpalricas@ua.pt eduardoalves@ua.pt

(103477) Inês Santos
ines.santos20@ua.pt

7 de dezembro de 2022

Índice

1 Contribuições de cada elemento	1
2 Introdução	2
3 Descrição dos métodos usados para encontrar a melhor solução	4
4 Descrição da solução encontrada	6
4.1 Gráficos da solução fornecida	7
4.2 Fórmula e Estimativa da posição 800	9
5 Gráficos da solução criada	11
6 Algumas soluções de PDFs gerados	13
7 Matlab	17
8 Apêndice	19
9 Conclusão	26
10 Bibliografia	27

Capítulo 1

Contribuições de cada elemento

Percentagem das contribuições		
David Palricas	Eduardo Alves	Inês Santos
33.33%	33.33%	33.33%

Todos os membros do grupo contribuiram igualmente para todas as componentes desta avaliação.

Capítulo 2

Introdução

Este primeiro trabalho "Speed Run" descreve-nos uma estrada dividida por múltiplos sub-segmentos, cada um destes com um limite de velocidade, ou speed_limit de 1 a 9.

Um carro que começa no primeiro segmento de estrada com uma variável "speed" de 0 terá de alcançar o último segmento de estrada com uma velocidade de 1 no mínimo número de moves possíveis.

Para tal, o carro tem a possibilidade de aumentar, diminuir ou manter a sua velocidade no início de cada move (movimento). O valor da velocidade "speed" corresponde ao número de sub-segmentos que o carro consegue avançar em um único move.

Algumas outras informações necessárias foram:

- position (que corresponde ao sub-segmento em que o carro se encontra)
- final_position (que corresponde ao sub-segmento final da estrada)
- move_number (que corresponde ao número de movimentos)
- new_speed (que corresponde ao valor da speed após o move)
- makefile que nos permite compilar o programa
- elapsed_time.h que nos apresenta o tempo que o programa demora a correr
- make_custom_pdf.c que nos permite transformar os resultado em formato pdf

O array que define a estrada varia, logo tivemos que definir um algoritmo que melhor resolvesse a travessia do carro para todas as possíveis opções de estrada. A seguinte imagem é apenas um exemplo de uma possível estrada e da melhor solução para exemplo dado.

In the following example the road has 31 segments (i.e. final_position=30).

5	5	6	6	8	7	8	9	8	9	8	9	9	8	9	7	7	6	6	6	5	5	4	4	3	5	3	3	3	3	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]

The optimal solution has 10 moves (the 11 car positions are shown in gray):

5	5	6	6	8	7	8	9	8	9	8	9	9	8	9	7	7	6	6	6	5	6	5	4	4	3	5	3	3	3	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]

Neste projeto iremos apresentar:

- As melhorias da solução ineficiente que nos foi atribuída no começo do trabalho.
- A maior final_position para cada um dos nossos números mecanográficos ao fim de uma hora (3600s) a correr o programa.
- As fórmulas que calculam uma boa estimativa do tempo de execução.
- O tempo estimado de execução de uma solução com posição final de 800.
- A nossa sugestão de uma solução melhorada que nos permite chegar à posição de 800 em, aproximadamente, alguns microsegundos.

Capítulo 3

Descrição dos métodos usados para encontrar a melhor solução

A solução já fornecida é uma solução recursiva muito lenta, isto deve-se a esta solução testar todos os valores de velocidade possíveis, primeiro reduzindo a velocidade (Brake), depois mantendo (Cruise) e só depois acelerando (Accelerate) para a conclusão do trajeto. Ou seja, a solução fornecida testa caminhos impráticos tais como o valor da velocidade manter-se sempre a 1 durante o percurso todo ou caminhos que anteriormente já tinham sido superados/calculados (por exemplo, uma tentativa anterior já chegou à posição 30 em 10 moves, e este programa continuará a testar soluções que cheguem à posição 30 com 11 moves ou mais).

Adicionando assim tempos de execução desnecessariamente grandes e uma ocupação de espaço não necessário.

Percebemos então que, para melhorar esta solução, teríamos que pensar de forma diferente. Sendo que a solução ideal não repetiria soluções impráticas. No entanto, sentimos que faria mais sentido avançar passo a passo. Reparámos que na solução fornecida, todas as velocidades eram testadas começando por baixo, como podemos ver na seguinte imagem.

```
// no, try all legal speeds
for(new_speed = speed - 1;new_speed <= speed + 1;new_speed++)
| if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position)
```

Decidimos então, colocar o carro a acelerar até não ser mais possível, alterando este "for" para o da imagem seguinte.

Isto deu-nos uma melhoria instantânea nos resultados, mas a solução ainda se encontrava muito fraca.

```
// no, try all legal speeds
for(new_speed = speed + 1; new_speed >= speed + 1; new_speed--)
| if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position)
```

Na solução final que nós propomos, estas soluções desvantajosas são eliminadas, tornando o caminho muito mais eficiente e rápido, com um tempo de execução até à posição de 800 de, aproximadamente, alguns microsegundos. Isto deve-se ao uso de um array bi-dimensional, também conhecido como matriz, ao qual designámos "bestnmoves". Este array usa os valores das *positions* e da *speed* para armazenar os valores do move_number, permitindo assim ir guardando os valores já obtidos e eliminando as soluções não desejáveis.

```
-- 
if(move_number >= bestnmoves[position][speed])
| |
| return;

bestnmoves[position][speed] = move_number;
```

Por fim, desenvolvemos também uma nova função *solve*, de forma a puder ser possível apresentar os valores desta nova solução.

```
static void solve_2(int final_position)
{
    if(final_position < 1 || final_position > _max_road_size_)
    {
        fprintf(stderr,"solve_2: bad final_position\n");
        exit(1);
    }
    for(int i = 0; i<= final_position; i++){
        for(int j = 0; j<= _max_road_speed_; j++){
            bestmoves[i][j] = final_position + 100;
        }
    }

    solution_2_elapsed_time = cpu_time();
    solution_2_count = 0ul;
    solution_2.best_n_moves = final_position + 100;
    solution_2_recursion(0,0,0,final_position);
    solution_2_elapsed_time = cpu_time() - solution_2_elapsed_time;
}
```

Capítulo 4

Descrição da solução encontrada

Com a solução que nos foi fornecida testámos os resultados que obteríamos ao fim de uma hora, com cada número mecanográfico. Deste modo, pode ser feito um certa comparação com a nossa solução mais otimizada. Em todos os três casos a posição mais alta alcançada foi de 50 nos 3600 segundos de teste, como podemos ver pelas imagens abaixo.

Comparando os gráficos da solution_1_recursion com os gráficos da solução desenvolvida podemos constatar uma enorme diferença, tendo em conta que em apenas microsegundos, a solução desenvolvida atinge a posição 800 (em contraste com os 50 no prazo de uma hora).

4.1 Gráficos da solução fornecida

(a) Valores com o num_mec 104179

```
make: "speed_run" is up to date.
[plainrecursion] ./speed_run 104179
```

n	sol	count	cpu time
1	1	2	0.000e+00
2	1	3	0.000e+00
3	1	5	0.000e+00
4	3	9	0.000e+00
5	4	13	0.000e+00
6	4	23	0.000e+00
7	5	36	0.000e+00
8	6	60	0.000e+00
9	5	100	0.000e+00
10	5	153	0.000e+00
11	6	279	0.000e+00
12	6	465	0.000e+00
13	7	777	0.000e+00
14	7	1297	0.000e+00
15	7	2155	0.000e+00
16	7	3614	0.000e+00
17	8	6413	0.000e+00
18	8	10865	0.000e+00
19	8	16120	0.000e+00
20	8	24624	0.000e+00
21	9	46737	0.000e+00
22	9	77377	0.000e+00
23	9	138831	0.000e+00
24	9	219000	0.000e+00
25	9	359548	0.000e+00
26	9	557700	0.000e+00
27	10	998388	0.000e+00
28	10	1550000	1.500e+00
29	10	273281	0.000e+00
30	10	4592954	3.125e-02
31	11	5500000	0.000e+00
32	11	11747980	7.612e-02
33	11	12500000	0.000e+00
34	12	35687757	2.344e-01
35	12	5996157	0.000e+00
36	12	9986157	6.484e-01
37	13	16463176	1.078e+00
38	13	17500000	0.000e+00
39	13	456303055	2.069e+00
40	13	70000000	0.000e+00
41	14	127001749	2.039e+00
42	14	212000000	0.000e+00
43	14	352120551	2.317e+01
44	15	578752985	3.069e+00
45	15	773770000	0.000e+00
46	15	1628376135	1.409e+02
47	15	2120000000	0.000e+00
48	16	4516381388	2.088e+02
49	16	7737700000	0.000e+00
50	16	15486677694	8.277e+02

(b) Valores com o num_mec 103477

```
make: "speed_run" is up to date.
[plainrecursion] ./speed_run 103477
```

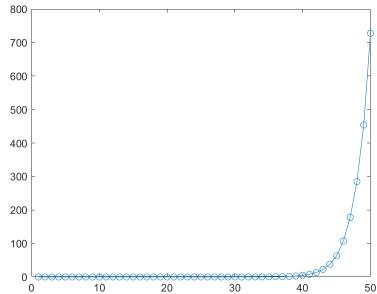
n	sol	count	cpu time
1	1	2	0.000e+00
2	1	5	0.000e+00
3	1	8	0.000e+00
4	2	13	0.000e+00
5	4	22	0.000e+00
6	4	35	0.000e+00
7	5	68	0.000e+00
8	5	109	0.000e+00
9	6	157	0.000e+00
10	6	223	0.000e+00
11	7	373	0.000e+00
12	6	465	0.000e+00
13	7	77377	0.000e+00
14	7	1277	0.000e+00
15	7	2165	0.000e+00
16	7	3135	0.000e+00
17	8	6931	0.000e+00
18	8	10865	0.000e+00
19	8	16120	0.000e+00
20	9	46737	0.000e+00
21	9	77377	0.000e+00
22	9	11747980	0.000e+00
23	9	12500000	0.000e+00
24	9	211597	0.000e+00
25	9	35330000	0.000e+00
26	10	506452	0.000e+00
27	10	753214	0.000e+00
28	10	1552314	0.000e+00
29	10	212000000	0.000e+00
30	11	458839	1.502e-02
31	11	1273153	7.612e-02
32	11	2124668	1.405e-01
33	12	336131	0.000e+00
34	12	59387581	3.708e-01
35	12	998388	0.000e+00
36	12	16475539	1.602e+00
37	13	22700000	0.000e+00
38	13	456246675	2.953e+00
39	14	75946625	4.305e+00
40	14	125000000	0.000e+00
41	14	210977621	1.347e+01
42	14	3152314	0.000e+00
43	15	585657380	3.066e+01
44	15	998388	0.000e+00
45	15	1642497221	0.076e+02
46	16	3152314	0.000e+00
47	16	458839	0.000e+00
48	16	45898197631	2.3074e+02
49	17	75346084138	4.3074e+02
50	17	11721740948	5.3078e+02

(c) Valores com o num_mec 108780

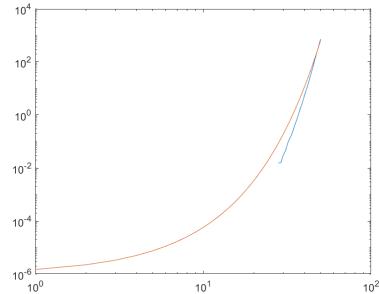
```
make: "speed_run" is up to date.
[plainrecursion] ./speed_run 108780
```

n	sol	count	cpu time
1	1	2	0.000e+00
2	1	5	0.000e+00
3	1	8	0.000e+00
4	2	13	0.000e+00
5	4	22	0.000e+00
6	4	35	0.000e+00
7	5	68	0.000e+00
8	5	109	0.000e+00
9	6	157	0.000e+00
10	6	223	0.000e+00
11	7	373	0.000e+00
12	6	465	0.000e+00
13	7	77377	0.000e+00
14	7	1277	0.000e+00
15	7	2165	0.000e+00
16	7	3135	0.000e+00
17	8	6931	0.000e+00
18	8	10865	0.000e+00
19	8	16120	0.000e+00
20	9	46737	0.000e+00
21	9	77377	0.000e+00
22	9	11747980	0.000e+00
23	9	12500000	0.000e+00
24	9	211597	0.000e+00
25	9	35330000	0.000e+00
26	10	506452	0.000e+00
27	10	753214	0.000e+00
28	10	1552314	0.000e+00
29	10	212000000	0.000e+00
30	11	458839	1.502e-02
31	11	1273153	7.612e-02
32	11	2124668	1.405e-01
33	12	336131	0.000e+00
34	12	59387581	3.708e-01
35	12	998388	0.000e+00
36	12	16475539	1.602e+00
37	13	22700000	0.000e+00
38	13	456246675	2.953e+00
39	14	75946625	4.305e+00
40	14	125000000	0.000e+00
41	14	210977621	1.347e+01
42	14	3152314	0.000e+00
43	15	585657380	3.066e+01
44	15	998388	0.000e+00
45	15	1642497221	0.076e+02
46	16	3152314	0.000e+00
47	16	458839	0.000e+00
48	16	45898197631	2.3074e+02
49	17	75346084138	4.3074e+02
50	17	11721740948	5.3078e+02

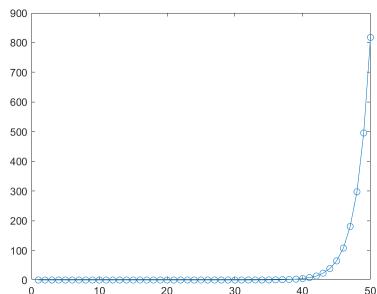
Alguns gráficos mais detalhados de informação sobre a solução fornecida são:



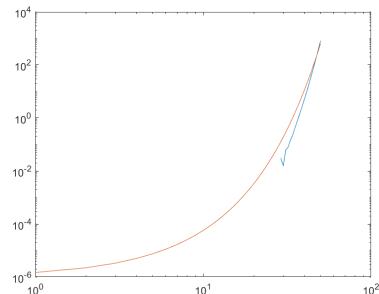
(a) Gráficos - n_{mec} 103477



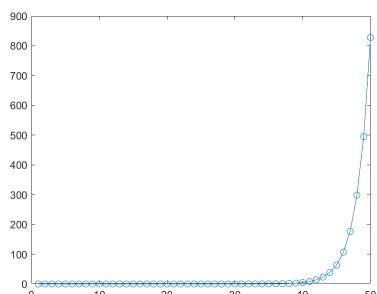
(b) Gráficos - n_{mec} 103477



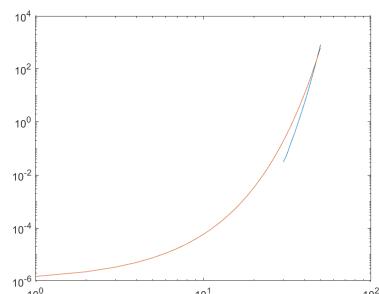
(a) Gráficos - n_{mec} 108780



(b) Gráficos - n_{mec} 108780



(a) Gráficos - n_{mec} 104179



(b) Gráficos - n_{mec} 104179

4.2 Fórmula e Estimativa da posição 800

A partir dos gráficos que vimos previamente, concluímos que a função de cada gráfico é exponencial. Com isto obteremos uma fórmula do tipo ($y = a \cdot b^x$) para cada gráfico, para podermos calcular o tempo em segundos e em anos, que a primeira solução demora a chegar à posição 800. Nas imagens seguintes estarão representados os cálculos necessários para descobrir as constantes a e b , da fórmula anterior, bem como os cálculos para determinar o tempo para alcançar a posição 800 em cada gráfico, e o valor médio deste tempo entre os três gráficos.

$$\begin{aligned}
 & 103\ 477 \\
 & \text{Ponto 1} \rightarrow (50, 727, 6) \\
 & \text{Ponto 2} \rightarrow (30, 0, 03125) \\
 & \left. \begin{array}{l} 727,6 = a \times b^{50} \\ 0,03125 = a \times b^{30} \end{array} \right\} \\
 & \left. \begin{array}{l} a = \frac{727,6}{b^{50}} \\ \hline \end{array} \right. \\
 & \left. \begin{array}{l} \hline \\ 0,03125 = \frac{727,6}{b^{50}} \times b^{30} \end{array} \right. \\
 & \left. \begin{array}{l} \hline \\ 0,03125 = \frac{727,6}{b^{20}} \end{array} \right. \\
 & \left. \begin{array}{l} \hline \\ b^{20} = \frac{727,6}{0,03125} \end{array} \right.
 \end{aligned}$$

(a) Cálculo da fórmula - 103477

$$\begin{aligned}
 & \Leftrightarrow \left. \begin{array}{l} a = 8,796 \times 10^{-9} \\ b = 1,653 \end{array} \right. \\
 & y = 8,796 \times 10^{-9} \times 1,653^x
 \end{aligned}$$

(b) Cálculo da fórmula - 103477 (2)

Ponto 1 $\rightarrow (35; 0,375)$
 Ponto 2 $\rightarrow (50; 817,8)$

$$\begin{cases} 0,375 = a \times b^{35} \\ 817,8 = a \times b^{50} \end{cases}$$

$$\Leftrightarrow \begin{cases} a = \frac{0,375}{b^{35}} \\ 817,8 = \frac{0,375}{b^{35}} \times b^{50} \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} 817,8 = 0,375 b^{15} \\ b^{15} = \frac{817,8}{0,375} \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} b^{15} = 2176 \\ a = 6,080 \times 10^{-9} \end{cases} \Rightarrow y = 6,080 \times 10^{-9} \times 2176$$

$$\Leftrightarrow \begin{cases} b = 1,669 \\ a = 6,080 \times 10^{-9} \end{cases} \Rightarrow y = 6,080 \times 10^{-9} \times 1,669$$

(a) Cálculo da fórmula - 108780

Número Hec $\rightarrow 104179$
 Ponto 1 $\rightarrow (40; 9,84)$
 Ponto 2 $\rightarrow (50; 827,7)$

$$\begin{cases} 9,84 = a \times b^{40} \\ 827,7 = a \times b^{50} \end{cases} \Leftrightarrow \begin{cases} a = \frac{9,84}{b^{40}} \\ 827,7 = \frac{9,84}{b^{40}} \times b^{50} \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} 827,7 = 9,84 \times b^{10} \\ b^{10} = \frac{827,7}{9,84} \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} b^{10} = 84,918 \\ a = 9,84 \times 10^{-9} \end{cases} \Rightarrow y = 9,84 \times 10^{-9} \times 84,918$$

$$\Leftrightarrow \begin{cases} b^{10} = \frac{827,7}{9,84} \\ a = 1,966 \times 10^{-7} \end{cases} \Rightarrow y = 1,966 \times 10^{-7} \times 84,918$$

$$\Leftrightarrow \begin{cases} b = 1,558 \\ a = 1,966 \times 10^{-7} \end{cases} \Rightarrow y = 1,966 \times 10^{-7} \times 1,558$$

(b) Cálculo da fórmula - 104179

Número Hec $\rightarrow 103477$
 $y = 800 \Leftrightarrow$
 $\Leftrightarrow 8,796 \times 10^{-7} \times 1,653^x = 800$
 $\Leftrightarrow x = 3,652 \times 10^{166}$
 $\Leftrightarrow x = 1,157 \times 10^{159}$ anos
 Número Hec $\rightarrow 108780$
 $y = 800 \Leftrightarrow$
 $\Leftrightarrow 6,080 \times 10^{-9} \times 1,669^x = 800 \Leftrightarrow$
 $\Leftrightarrow x = 5,610 \times 10^{167} \Leftrightarrow$
 $\Leftrightarrow x = 1,778 \times 10^{169}$ anos
 Número Hec $\rightarrow 104179$
 $y = 800 \Leftrightarrow$
 $\Leftrightarrow 1,966 \times 10^{-7} \times 1,558^x = 800 \Leftrightarrow$
 $\Leftrightarrow x = 2,226 \times 10^{147}$
 $\Leftrightarrow x = 7,054 \times 10^{135}$ anos

(a) Anos para posição de 800

Média dos valores

$$\bar{x} = \frac{1,157 \times 10^{159} + 1,778 \times 10^{167} + 7,054 \times 10^{135}}{3}$$

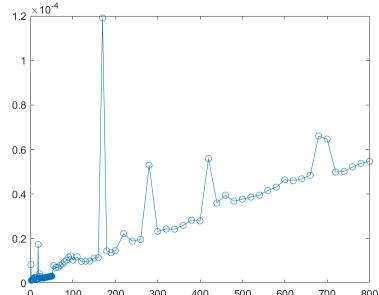
$$\Leftrightarrow \bar{x} = 5,931 \times 10^{161}$$

(b) Anos para posição de 800(2)

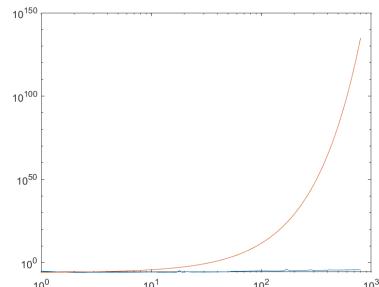
Capítulo 5

Gráficos da solução criada

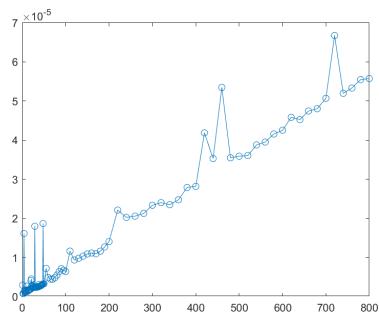
Nas imagens seguintes estão representados os gráficos da solução 2 para cada número mecanográfico. Os gráficos de curvas azuis representam a variação do tempo de execução em função das posições.



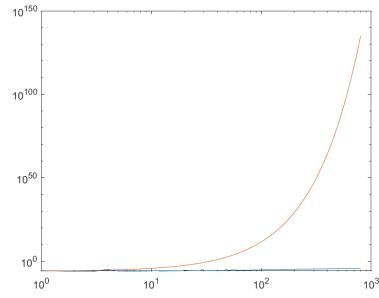
(a) Gráficos - n_mec 104179



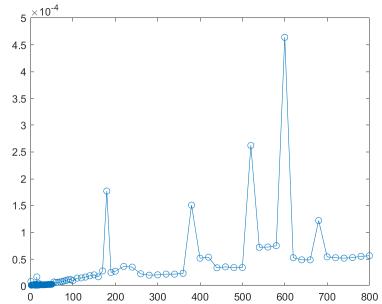
(b) Gráficos - n_mec 104179



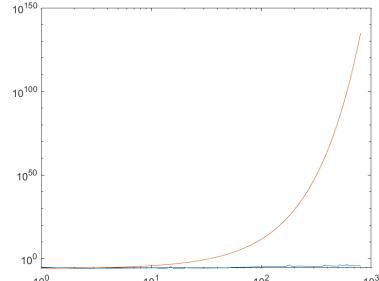
(a) Gráficos - n_mec 103477



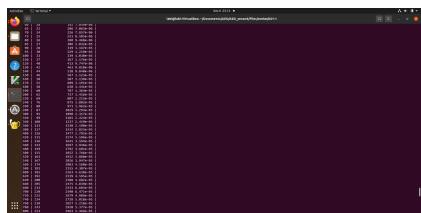
(b) Gráficos - n_mec 103477



(a) Gráficos - n_mec 108780



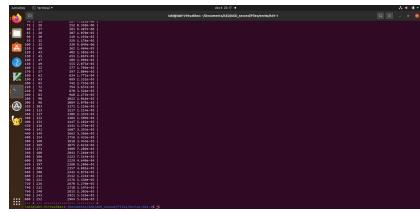
(b) Gráficos - n_mec 108780



(a) Solução melhorada - 104179



(b) Solução melhorada - 103477



(c) Solução melhorada - 108780

As imagens anteriores demonstram os resultados no terminal da solução que criámos.

Podemos concluir que através da observação destes gráficos, a solução 2 é mais eficaz que a solução fornecida, uma vez que alcança mais posições num menor intervalo de tempo.

Capítulo 6

Algumas soluções de PDFs gerados

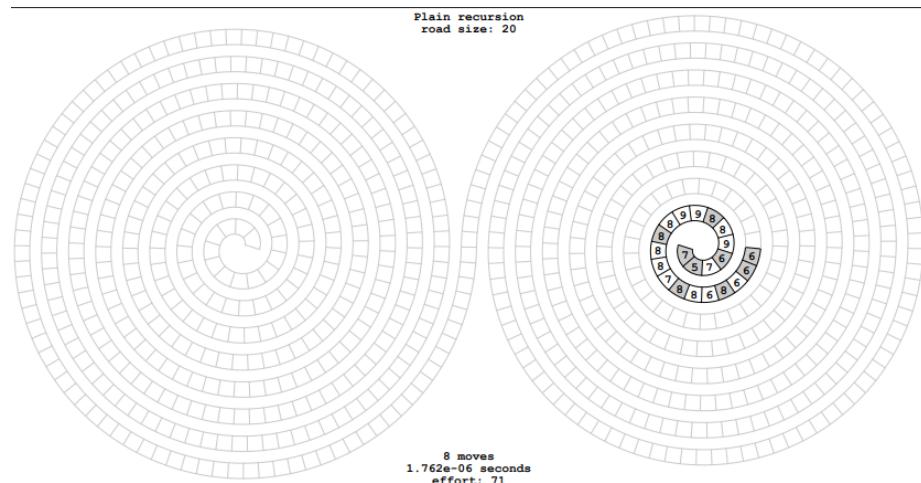


Figura 6.1: 20 posições - seed 108780

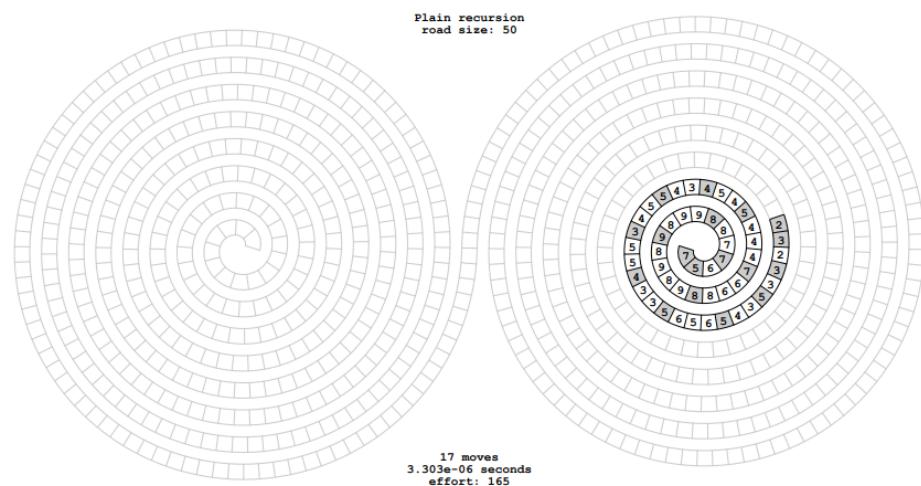


Figura 6.2: 50 posições - seed 103477

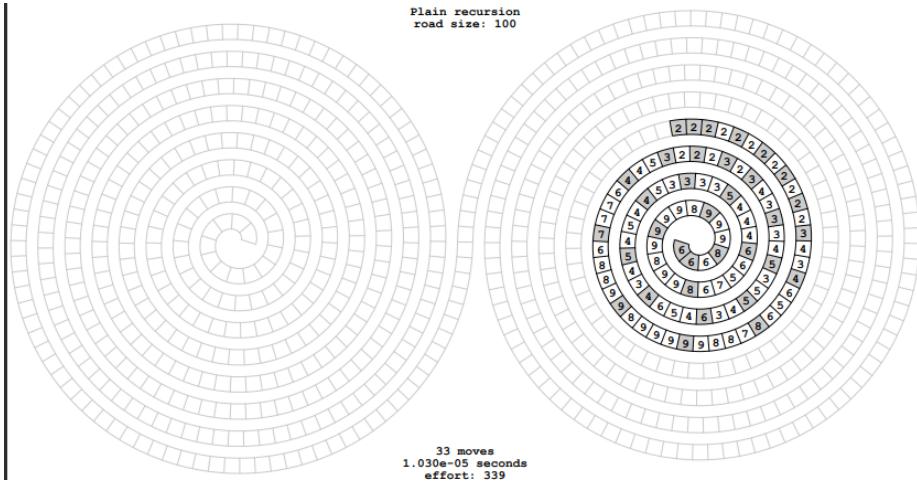


Figura 6.3: 100 posições - seed 104179

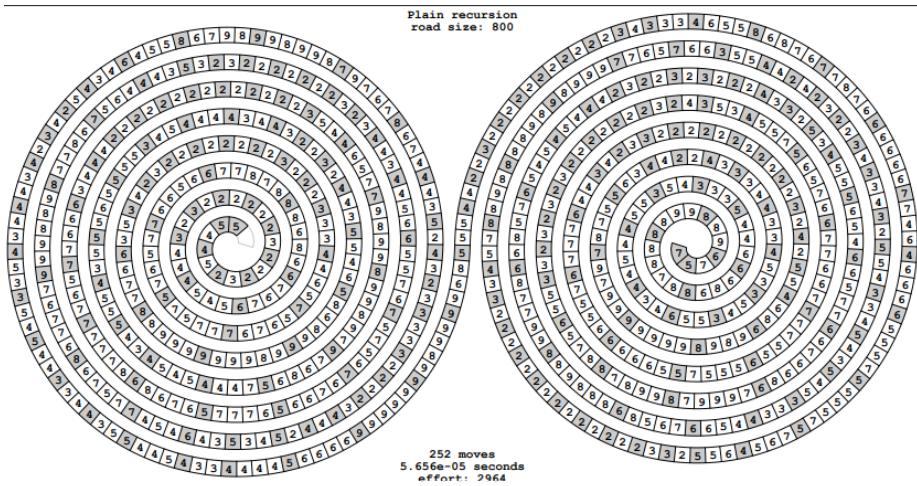


Figura 6.4: 800 posições - seed 108780

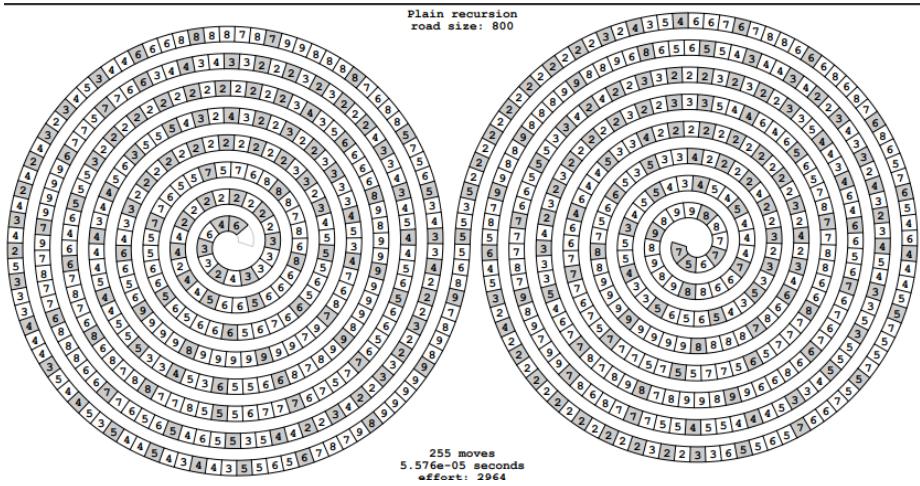


Figura 6.5: 800 posições - seed 103477

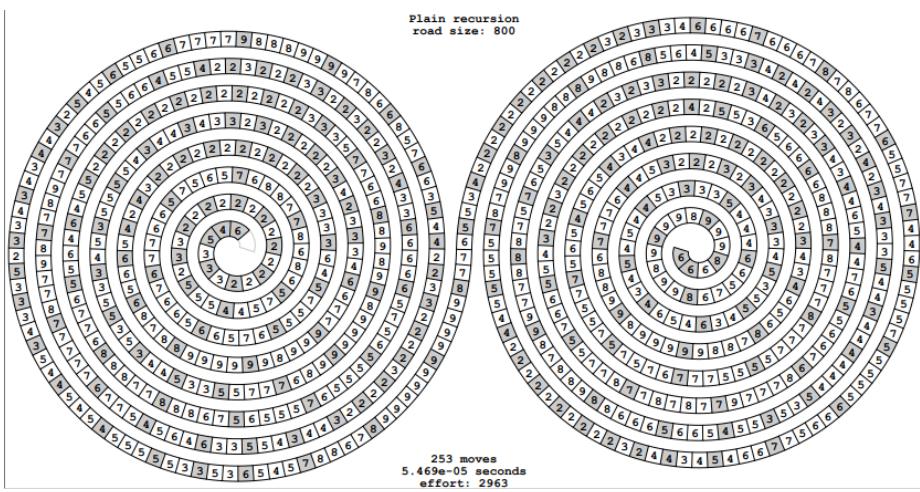


Figura 6.6: 800 posições - seed 104179

Capítulo 7

Matlab

Para desenvolver os gráficos que apresentamos anteriormente, tivemos que recorrer à ferramenta *Matlab*. Alguns excertos do código que usámos para tais gráficos foram:

```
M = load ('Speed_Run_sol2.txt');
n = M(:,1);
nsol=M(:,2);
ncounts=M(:,3);
tempo=M(:,4);

%Gráfico do tempo
plot(n,tempo,'-o')
figure
loglog(n,tempo,n,1.3.^((2*n)/(10.^6)), 'r')
loglog(n,tempo,n,1.5.^n/(10.^6))

% Estimativa inferior para n=800:
t800 = 1.2.^((2*800)/(10.^6))/3600/24/365

t800 = 1.5.^800/(10.^6)/3600/24/365 % em anos

plot(n,t)
loglog(n,t)
semilogy(n,t)
figure
plot(n,log10(t));

t_log=log10(t);

N=[n(20:end) 1+0*n(20:end)];
Coefs=pinv(N)*t_log(20:end);

hold on
Ntotal=[n n*0+1];
plot(n,10.^((Ntotal*Coefs),'r')
plot(n,Ntotal*Coefs,'r')

t800_log=[800 1]*Coefs
t800=10.^t800_log / 3600/24/365

A2 = load("speed_run_weaksolution.txt");
n2=A2(:,1);
t2=A2(:,4);
semilogy(n,t,n2,t2)
A3=load("Speed_Run_sol2.txt");
hold on
semilogy(A3(:,1),A3(:,4),'k')
hold off
```

```
M = load ('Speed_Run_sol2_103477.txt');
n = M(:,1);
nsol=M(:,2);
ncounts=M(:,3);
tempo=M(:,4);

%Gráfico do tempo
plot(n,tempo,'-o')
figure
loglog(n,tempo,n,1.3.^((2*n)/(10.^6)), 'r')
loglog(n,tempo,n,1.5.^n/(10.^6))

% Estimativa inferior para n=800:
t800 = 1.2.^((2*800)/(10.^6))/3600/24/365

t800 = 1.5.^800/(10.^6)/3600/24/365 % em anos

plot(n,t)
loglog(n,t)
semilogy(n,t)
figure
plot(n,log10(t));

t_log=log10(t);

N=[n(20:end) 1+0*n(20:end)];
Coefs=pinv(N)*t_log(20:end);

hold on
Ntotal=[n n*0+1];
plot(n,10.^((Ntotal*Coefs),'r')
plot(n,Ntotal*Coefs,'r')

t800_log=[800 1]*Coefs
t800=10.^t800_log / 3600/24/365

A2 = load("speed_run_weaksolution.txt");
n2=A2(:,1);
t2=A2(:,4);
semilogy(n,t,n2,t2)
A3=load("Speed_Run_sol2_103477.txt");
hold on
semilogy(A3(:,1),A3(:,4),'k')
hold off
```

(a) MatLab - Geral

(b) MatLab - 103477

```

M = load ('Speed_Run_sol2_104179.txt');
n = M(:,1);
nsol=M(:,2);
ncounts=M(:,3);
tempo=M(:,4);

%Gráfico do tempo
plot(n,tempo,'o')
figure
loglog(n,tempo,n,1.3.^(2*n)/(10.^6), 'r')
loglog(n,tempo,n,1.5.^n/(10.^6))

% Estimativa inferior para n=800:
t800 = 1.2.^*(2*800)/(10.^6)/3600/24/365

t800 = 1.5.^800/(10.^6)/3600/24/365 % em anos

plot(n,t)
loglog(n,t)
semilogy(n,t)
figure
plot(n,log10(t));

t_log=log10(t);
N=[n(20:end) 1+0*n(20:end)];
Coefs=pinv(N)*t_log(20:end);

hold on
Ntotal=[n n*0+1];
plot(n,10.^(Ntotal*Coefs),'r')
plot(n,Ntotal*Coefs,'r')

t800_log=[800 1]*Coefs
t800=10^t800_log / 3600/24/365

A2 = load("speed_run_weaksolution.txt");
n2=A2(:,1);
t2=A2(:,4);
semilogy(n,t,n2,t2)
A3=load("Speed_Run_sol2_104179.txt");
hold on
semilogy(A3(:,1),A3(:,4),'k')
hold off

M = load ('Speed_Run_sol2_108780.txt');
n = M(:,1);
nsol=M(:,2);
ncounts=M(:,3);
tempo=M(:,4);

%Gráfico do tempo
plot(n,tempo,'o')
figure
loglog(n,tempo,n,1.3.^(2*n)/(10.^6), 'r')
loglog(n,tempo,n,1.5.^n/(10.^6))

% Estimativa inferior para n=800:
t800 = 1.2.^*(2*800)/(10.^6)/3600/24/365

t800 = 1.5.^800/(10.^6)/3600/24/365 % em anos

plot(n,t)
loglog(n,t)
semilogy(n,t)
figure
plot(n,log10(t));

t_log=log10(t);
N=[n(20:end) 1+0*n(20:end)];
Coefs=pinv(N)*t_log(20:end);

hold on
Ntotal=[n n*0+1];
plot(n,10.^(Ntotal*Coefs),'r')
plot(n,Ntotal*Coefs,'r')

t800_log=[800 1]*Coefs
t800=10^t800_log / 3600/24/365

A2 = load("speed_run_weaksolution.txt");
n2=A2(:,1);
t2=A2(:,4);
semilogy(n,t,n2,t2)
A3=load("Speed_Run_sol2_108780.txt");
hold on
semilogy(A3(:,1),A3(:,4),'k')
hold off

```

(a) MatLab - 104179

(b) MatLab - 108780

Capítulo 8

Apêndice

```

//  

// AED, August 2022 (Tomás Oliveira e Silva)  

//  

// First practical assignment (speed run)  

//  

// Compile using either  

// cc -Wall -O2 -f_use_zlib -D solution_speed_run.c -lm  

// or  

// cc -Wall -O2 -f_use_zlib -I solution_speed_run.c -lm -lz  

//  

// Place your student numbers and names here  

// N.Mec. 103477 Name: Inês Santos  

// N.Mec. 104179 Name: Eduardo Alves  

// N.Mec. 108780 Name: David Palmeiras  

//  

//  

// static configuration  

//  

//  

#define _max_road_size_ 800 // the maximum problem size  

#define min_road_speed_ 2 // must not be smaller than 1, should not 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"  

//  

// 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 + 0.20 * 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() % 30) - 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 solution_t solution_2,solution_2_best;
//static double solution_1_elapsed_time; // time it took to solve the problem
static double solution_2_elapsed_time; // time it took to solve the problem after better solution
//static unsigned long solution_1_count; // effort expended solving the problem after better solution
static unsigned long solution_2_count; // effort expended solving the problem after better solution
static int bestmoves[_max_road_size + 1][_max_road_speed + 1];

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

//     // record move
//     solution_1_count++;
//     solution_1.positions[move_number] = position;
//     // is it a solution?
//     if(position == final_position && speed == 1)
//     {
//         // is it a better solution?
//         if(move_number < solution_1_best.n_moves)
//         {
//             solution_1_best = solution_1;
//             solution_1_best.n_moves = move_number;
//         }
//         return;
//     }
//     // no, try all legal speeds
//     for(new_speed = speed - 1;new_speed <= speed + 1;new_speed++)
//     {
//         if(new_speed >= 1 && new_speed <= _max_road_speed && position + new_speed <= final_position)
//         {
//             // no, try all legal speeds
//             for(new_speed = speed - 1;new_speed <= speed + 1;new_speed++)
//             {
//                 if(new_speed >= 1 && new_speed <= _max_road_speed && position + new_speed <= final_position)
//                 {
//                     for(i = 0;i <= new_speed && new_speed <= _max_road_speed[position + i];i++)
//                     {
//                         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_best.positions[0] = final_position;
    solution_1_elapsed_time = cpu_time() - solution_1_elapsed_time;
}

// static void solution_2_recursion(int move_number,int position,int speed,int final_position)
// {
//     int i, new_speed;
//     // int t_tracegap = ((speed-1)*(speed-2))/2;
//     //solution_2_count++;
//     if(move_number >= bestmoves[position][speed])

```

```

static void solution_2_recursion(int move_number,int position,int speed,int final_position)
{
    int i,new_speed;
    // int t travagem = ((speed-1)*(speed-2))/2;
    //solution_2.count++;
    if(move_number >= bestmoves[position][speed])
        return;
    bestmoves[position][speed] = move_number;
    solution_2.count++;
    solution_2.positions[move_number] = position;

    if(position == final_position && speed == 1){
        // is it a better solution?

        if(move_number < solution_2.best.n_moves)
        {
            solution_2.best = solution_2;
            solution_2.best.n_moves = move_number;
        }
        return;
    }
    // no, try all legal speeds
    for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--){

        if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position)
        {
            for(i = 0;i < new_speed && new_speed <= max_road_speed[position + i];i++){

                // no, try all legal speeds
                for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--){

                    if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position)
                    {
                        for(i = 0;i < new_speed && new_speed <= max_road_speed[position + i];i++){

                            //solution_2.positions[move_number] = position;
                            //solution_2.best.positions[move_number] = solution_2.positions[move_number] + new_speed;

                            if(i > new_speed){ // && move_number + i < solution_2.best.n_moves ){
                                | solution_2_recursion(move_number + i,position + new_speed,new_speed,final_position);
                            }
                        }
                    }
                }
            }
        }
    }
}

static void solve_2(int final_position)
{
    if(final_position < 1 || final_position > _max_road_size_)
    {
        fprintf(stderr,"solve_2: bad final_position\n");
        exit(1);
    }
    for(int i = 0; i< final_position; i++){
        for(int j = 0; j< max_road_speed ; j++){
            | bestmoves[i][j] = final_position + 100;
        }
    }
}

```

```

static void solve_2(int final_position)
{
    if(final_position < 1 || final_position > _max_road_size_)
    {
        fprintf(stderr,"solve_2: bad final_position\n");
        exit(1);
    }
    for(int i = 0; i< final_position; i++){
        for(int j = 0; j< _max_road_speed; j++){
            bestmoves[i][j] = final_position + 100;
        }
    }
    solution_2_elapsed_time = cpu_time();
    solution_2_count = 0;
    solution_2_best.n_moves = final_position + 100;
    solution_2_recursion(0,0,0,final_position);
    solution_2_elapsed_time = cpu_time() - solution_2_elapsed_time;
}
// example of the slides
//
static void example(void)
{
    int l,final_position;
    srand(0xAED2022);
    init_road_speeds();
    // final position = 30;
    // for(l = 0;l < final_position;
    // make_custom_pdf_file("example.pdf",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("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");

    final_position = 30;
    solve_2(final_position);
    make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],solution_2_best.n_moves,&solution_2_best.positions[0],solution_2_elapsed_time,
    solution_2_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_2_best.n_moves;i++)
        printf(" %d",solution_2_best.positions[i]);
    printf("\n");
}

```

```

final_position = 30;
solve_2(final_position);
make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],solution_2_best.n_moves,&solution_2_best.positions[0],solution_2_elapsed_time,
solution_2_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_2_best.n_moves;i++)
    printf("%d",solution_2_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]);
srand((unsigned int)n_mec);
init_road_speeds();
int main(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]);
srand((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;
solution_2_elapsed_time = 0.0;
printf("-----\n");
printf("----- plain recursion\n");
printf("-----\n");
printf(" n | sol count (pu 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("%d |",final_position);
    // first solution method (very bad)
    // if(solution_1_elapsed_time < _time_limit_)
    //{
    //    solve_1(final_position);
    //    if(print_this_one != 0)

```

```

while(final_position <= _max_road_size / 2 && final_position <= 207)
{
    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("%d",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(" %d %dlu %.3e |",solution_1_best.n_moves,solution_1_count,solution_1_elapsed_time);
    // }
    // else
    // {
    //     solution_1_best.n_moves = -1;
    //     printf(" -1 |");
    // }

    // second solution method (less bad)
    // ...

    if(solution_2_elapsed_time < _time_limit_)
    {
        solve_2(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_2_best.n_moves,&solution_2_best.positions[0],solution_2_elapsed_time,solution_2_count,"Plain recursion");
            if(solution_2_elapsed_time < _time_limit_)
            {
                solve_2(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_2_best.n_moves,&solution_2_best.positions[0],solution_2_elapsed_time,solution_2_count,"Plain recursion");
                }
                printf(" %d %dlu %.3e |",solution_2_best.n_moves,solution_2_count,solution_2_elapsed_time);
            }
            else
            {
                solution_2_best.n_moves = -1;
                printf(" -1 |");
            }
        }

        // 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_
}

```

Capítulo 9

Conclusão

Este projeto "Speed_Run" ajudou o desenvolvimento das nossas capacidades de solução de problemas e pensamento crítico na programação e a procura por várias perspetivas e diferentes maneiras de resolver o mesmo problema.

Aprendemos a procurar e fortalecer as fraquezas no nosso código de forma a encontrar soluções mais otimizadas e eficientes. O nosso principal objetivo era melhorar a solução fornecida, e sentimos que conseguimos alcançar o nosso objetivo da melhor forma usando um array bi-dimensional. Alcançando assim resultados muito mais eficientes em relação à recursão já fornecida.

Aprendemos também a recorrer à ferramenta MatLab, que foi útil na criação dos gráficos que apresentámos ao longo deste relatório.

Para finalizar, ambas as soluções funcionam, no entanto, a melhoria de rapidez observada na solution_2_recursion demonstra um aumento significativo na eficácia desta solução. Logo, a partir deste trabalho conseguimos realmente explorar a eficácia de diferentes soluções e respetivos tempos de execução.

Capítulo 10

Bibliografia

https://matlabacademy.mathworks.com/?s_tid=getstart_mlacad

<https://beginnersbook.com/2014/01/2d-arrays-in-c-example/>

<https://www.enjoyalgorithms.com/blog/minimum-number-of-jumps-to-reach-end>