deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

ALGORITMOS E ESTRUTURAS DE DADOS

Speed Run

Licenciatura em Engenharia de Computadores e Informática

Gonçalo Cunha - 33.33%

Mec.: 108352

Guilherme Santos - 33.33%

Mec.: 107961

João Gaspar - 33.33%

Mec.: 107708

Índice

O que é o Speed Run?

Este trabalho consiste numa estrada que se encontra subdividida em diversos segmentos com aproximadamente o mesmo comprimento, tendo cada um limite de velocidade medida pelo número de segmentos que o carro avança num movimento. Em cada movimento o carro pode: reduzir a sua velocidade, mantê-la ou aumentá-la em um valor. Inicialmente o carro é colocado no primeiro segmento de estrada com velocidade zero e tem de alcançar o último segmento com uma velocidade igual a um. O objetivo principal deste trabalho é determinar o número mínimo de movimentos que o carro efetua até chegar à posição requerida.

No exemplo abaixo a estrada tem 10 segmentos (sendo a *final_position* = 9).

5	5	6	6	8	7	8	9	8	9
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Uma solução possível poderá ser (contém 5 movimentos, as 6 posições encontram-se a cinza):

5	5	6	6	8	7	8	9	8	9
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Objetivo

O nosso objetivo com este trabalho é tentar construir e otimizar um algoritmo que permita ao programa resolver este problema para uma posição final de 800 com o menor esforço (*effor*t) e tempo possível.

Soluções

Solução Fornecida (Solução 1)

A primeira solução já nos foi fornecida pelos professores, onde para cada n (*final_position*) o programa executa uma pesquisa exaustiva, ou seja, percorre todas as opções possíveis em busca da mais eficaz para a resolução do problema em causa.

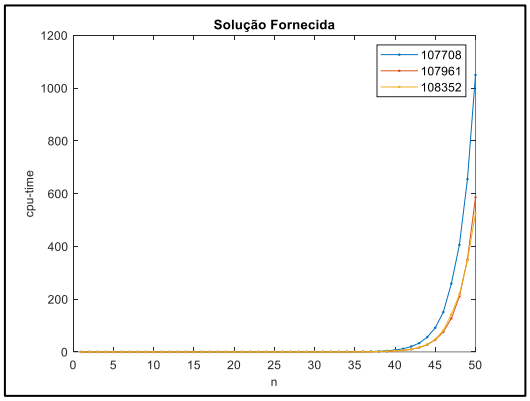
Análise de Dados

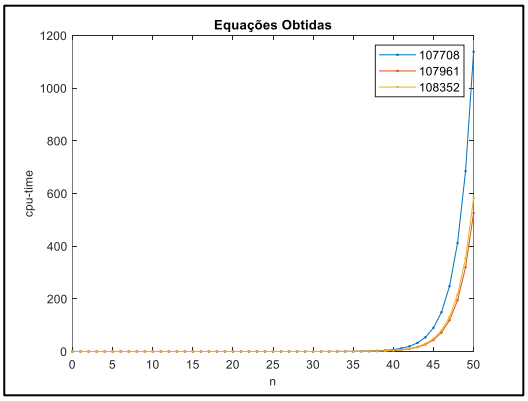
Apesar deste tipo de algoritmo encontrar todas as soluções possíveis garantindo assim a sua precisão, é um algoritmo ineficiente e muito lento pois resolve não considera repetições nem soluções já encontradas.

Para estimar o tempo usado para n = 800, fizemos através de MATLAB uma regressão linear usando o valor de *n* e o logaritmo natural dos valores do *cpu-time* colocando a equação obtida em ordem a *cpu-time*, e substituindo o valor de n por 800, obtemos:

3.04157E+168 segundos	->	N. Mec.: 107708
3.25592E+164 segundos	->	N. Mec.: 107961
5.28713E+164 segundos	->	N. Mec.: 108352

Comparação dos dados obtidos diretamente e pelo método anteriormente referido:



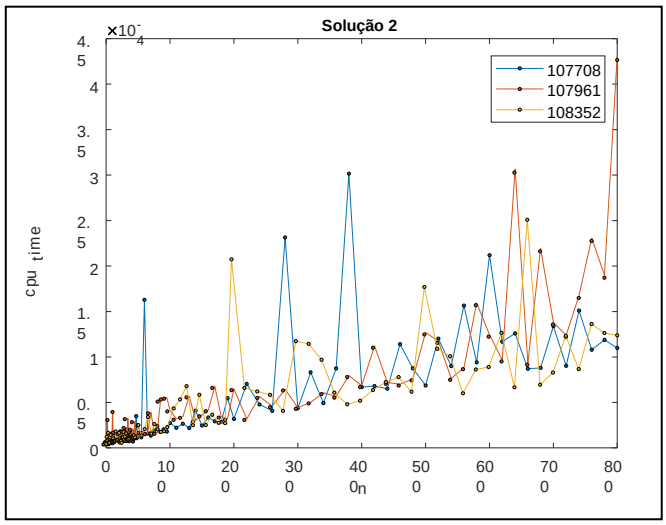


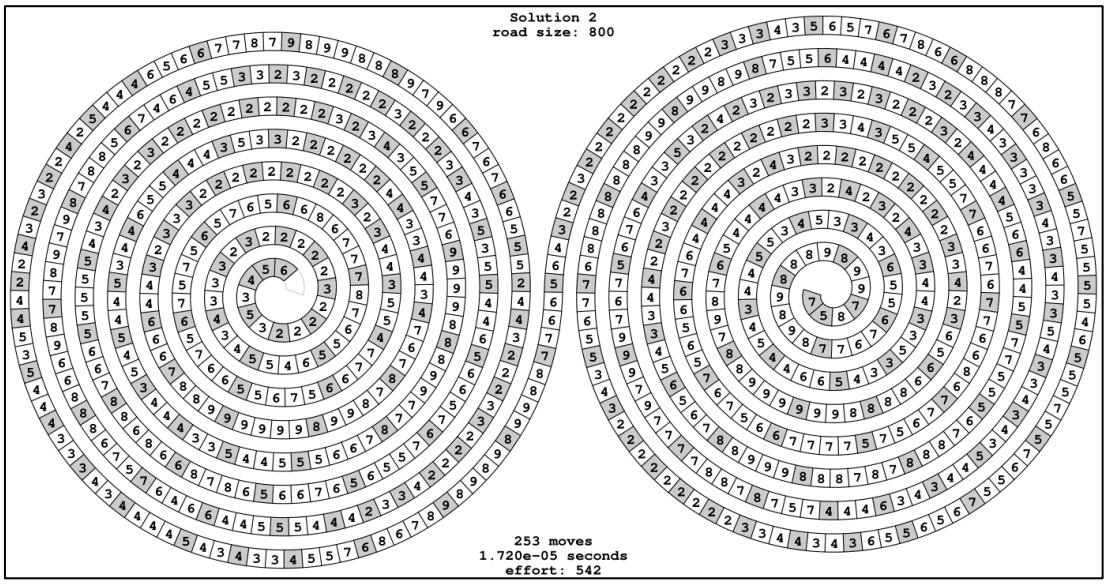
Solução 2

Nesta solução usámos como base a solução 1, com o objetivo de otimizá-la, corrigindo os seus pontos fracos. Para isto começámos por introduzir um "*if*" que para a pesquisa de uma solução se esta tiver um maior número de movimentos que a melhor solução atual.

Para complementar esta alteração, alteramos o ciclo "*for*" que verifica todas as velocidades possíveis para começar pelas velocidades mais altas, uma vez que estas têm maior probabilidade de serem mais eficientes, o que junto com o "*if*" anteriormente referido, melhora drasticamente a solução, reduzindo o seu "*effort*" entre 50 600, dependendo do número mecanográfico usado.

Análise de Dados

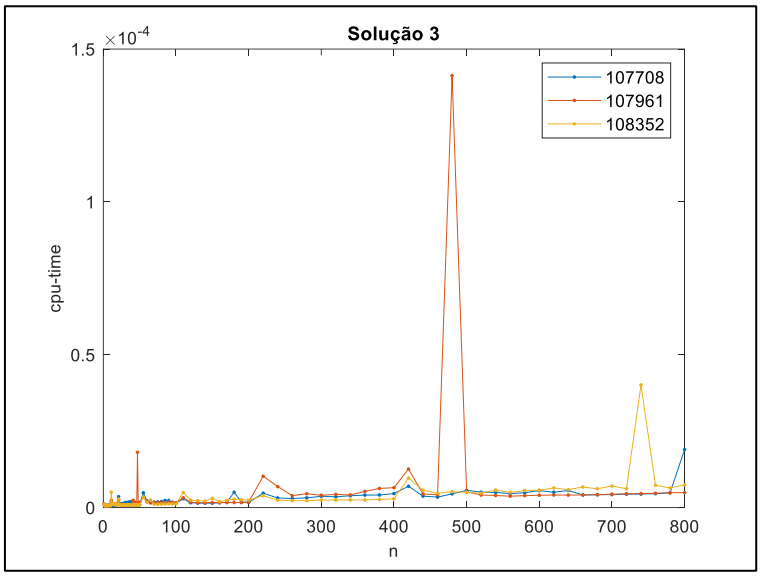


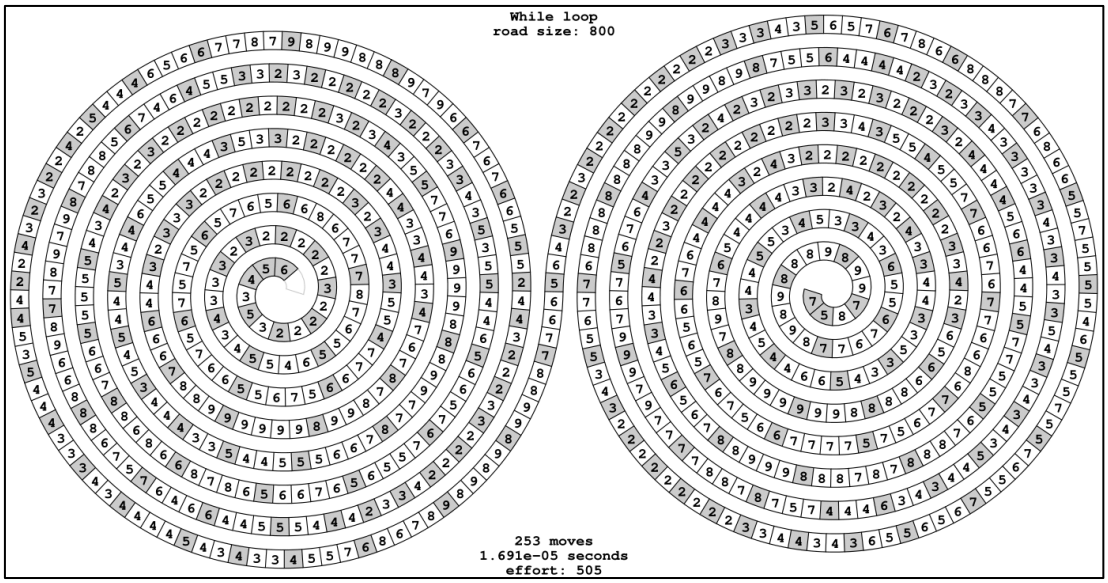


Solução 3

Na solução 3 baseamo-nos num *while-loop* que corre enquanto a posição final não é alcançada, sendo que só altera a posição seguinte quando verifica que é viável após determinar a melhor velocidade para a posição atual.

Análise de Dados





Código em C

```c
// AED, August 2022 (Tomás Oliveira e Silva)
//
// First practical assignment (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
//   N.Mec. 108352  Name: Gonçalo Cunha
//   N.Mec. 107961  Name: Guilherme Santos
//   N.Mec. 107708  Name: João Gaspar
//

//
// 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)
// #define C_MAX 50

//
// 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 + 0.30 * 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;

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

// solution to the problem using recursion

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)
        {
            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_recursion(0, 0, 0, final_position);
    solution_1_elapsed_time = cpu_time() - solution_1_elapsed_time;
}

//
// Student solution(s):
//

// recursive solution to the problem optimised from solution 1

static solution_t solution_2, solution_2_best;
static double solution_2_elapsed_time; // time it took to solve the
problem
static unsigned long solution_2_count; // effort dispended solving the
problem

static void solution_2_recursion(int move_number, int position, int
speed, int final_position)
{
```

```
    int i, new_speed;

    // record move
    solution_2_count++;
    solution_2.positions[move_number] = position;
    // is it a solution?
    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
    if (solution_2_best.positions[move_number] >
solution_2.positions[move_number])
    {
        return;
    }

    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_2_recursion(move_number + 1, 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);
    }
    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;
}
```

```
// solution to the problem using while loop

static solution_t solution_3, solution_3_best;
static double solution_3_elapsed_time; // time it took to solve the
problem
static unsigned long solution_3_count; // effort dispended solving the
problem

static void solution_3_while(int move_number, int position, int speed,
int final_position)
{
    // if 1 adds move to solution
    int viable_move = 1;

    // runs until final position is reached
    while (position < final_position)
    {
        solution_3.positions[move_number] = position;
        // try all speeds
        for (int speed_change = 1; speed_change >= -1; speed_change--)
        {
            solution_3_count++;
            viable_move = 1;
            int new_speed = speed + speed_change;
            int next_position = position + new_speed;

            // check if speed is legal
            if (new_speed < 1 || new_speed > _max_road_speed_ ||
max_road_speed[position] < new_speed)
                continue;

            int spaces_to_check = new_speed;
            int unchecked_spaces = spaces_to_check;

            // check if speed is viable
            for (int n = 1; n <= (new_speed * (new_speed + 1) / 2); n++)
            {
                if (position + n > final_position ||
max_road_speed[position + n] < spaces_to_check)
                {
                    viable_move = 0;
                    break;
                }

                unchecked_spaces -= 1;

                if (unchecked_spaces == 0)
                {
```

```
                    spaces_to_check -= 1;
                    unchecked_spaces = spaces_to_check;
                }
            }
            // if viable, add move to solution
            if (viable_move)
            {
                position = next_position;
                speed = new_speed;
                move_number++;
                break;
            }
        }
    }
    solution_3.positions[move_number] = position;
    solution_3_best = solution_3;
    solution_3_best.n_moves = move_number;
    return;
}

static void solve_3(int final_position)
{
    if (final_position < 1 || final_position > _max_road_size_)
    {
        fprintf(stderr, "solve_3: bad final_position\n");
        exit(1);
    }
    solution_3_elapsed_time = cpu_time();
    solution_3_count = 0ul;
    solution_3_best.n_moves = final_position + 100;
    solution_3_while(0, 0, 0, final_position);
    // int move_number,int position,int speed,int final_position
    solution_3_elapsed_time = cpu_time() - solution_3_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], 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");
}

//
// 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();
    //choose solution to use
    int solution_n;
    printf("Which solution do you want to use (1,2,3)\n");
    scanf("%d", &solution_n);
    final_position = 1; // C_MAX;
    solution_1_elapsed_time = 0.0;
    solution_2_elapsed_time = 0.0;
    solution_3_elapsed_time = 0.0;
    printf("    + --- ---------------- --------- +\n");
    // print solution name
    if (solution_n == 1)
    {
        printf("    |                plain recursion |\n");
    }
    else if (solution_n == 2)
    {
```

```
        printf("    |             improved recursion |\n");
    }
    else if (solution_n == 3)
    {
        printf("    |                while      loop |\n");
    }
    printf("--- + --- ---------------- --------- +\n");
    printf("  n | sol            count  cpu time |\n");
    printf("--- + --- ---------------- --------- +\n");
    while (final_position <= _max_road_size_ /*&& final_position <=
C_MAX*/)
    {
        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_n == 1)
        {

            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 (optimized solution 1)
        else if (solution_n == 2)
        {
            if (solution_2_elapsed_time < _time_limit_)
            {
                solve_2(final_position);
                if (print_this_one != 0)
```

```
                {
                    sprintf(file_name, "%03d_2.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,
"Improved recursion");
                }
                printf(" %3d %16lu %9.3e |", solution_2_best.n_moves,
solution_2_count, solution_2_elapsed_time);
            }
            else
            {
                solution_2_best.n_moves = -1;
                printf("                                  |");
            }
        }
        // third solution method (using while loop)

        else if (solution_n == 3)
        {
            if (solution_3_elapsed_time < _time_limit_)
            {
                solve_3(final_position);
                if (print_this_one != 0)
                {
                    sprintf(file_name, "%03d_3.pdf", final_position);
                    make_custom_pdf_file(file_name, final_position,
&max_road_speed[0], solution_3_best.n_moves,
&solution_3_best.positions[0], solution_3_elapsed_time, solution_3_count,
"While loop");
                }
                printf(" %3d %16lu %9.3e |", solution_3_best.n_moves,
solution_3_count, solution_3_elapsed_time);
            }
            else
            {
                solution_3_best.n_moves = -1;
                printf("                                  |");
            }
        }

        // 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 em MATLAB

```
%% Solução Fornecida %%%%%%%%%%%%%%%%%%

%% Gráfico %%
% 107708 %
f = fopen('speedrun_original_107708.txt','r'); % abrir ficheiro para leitura
mydata1 = textscan(f,'%f%f'); % separar as duas colunas por tab
md2 = mydata1{1,2}; % coluna cpu-time
fclose(f);
% 107961 %
f = fopen('speedrun_original_107961.txt','r'); % abrir ficheiro para leitura
mydata2 = textscan(f,'%f%f'); % separar as duas colunas por tab
md3 = mydata2{1,2}; % coluna cpu-time
fclose(f);
% 108352 %
f = fopen('speedrun_original_108352.txt','r'); % abrir ficheiro para leitura
mydata3 = textscan(f,'%f%f'); % separar as duas colunas por tab
md4 = mydata3{1,2}; % coluna cpu-time
fclose(f);
% fazer grafico %
figure(1);
n = 1:1:50;
plot(n,md2,n,md3,n,md4,'Marker','.');
legend('107708','107961','108352','FontSize',10);
labelo('n','FontSize',10); % legendar eixo dos x
ylabel('cpu-time','FontSize',10); % legendar eixo dos y
title('Solução Fornecida'); % dar título
xlim([0 50]);
ylim([0 1200]);

%% Gráfico Equações %%
p08 = polyfit(n,log(md2),1);
p61 = polyfit(n,log(md3),1);
p52 = polyfit(n,log(md4),1);
n = 0:1:800;
y08 = exp(0.507888709677419).^n * exp(-18.357330645161290);
y61 = exp(0.496723387096774).^n * exp(-18.571576612903222);
y52 = exp(0.497231209677419).^n * exp(-18.494779435483870);
figure(2);
plot(n,y08,n,y61,n,y52,'Marker','.');
legend('107708','107961','108352','FontSize',10);
xlabel('n','FontSize',10); % legendar eixo dos x
ylabel('cpu-time','FontSize',10); % legendar eixo dos y
title('Equações Obtidas'); % dar título
xlim([0 50]);
ylim([0 1200]);

%% Solução 1 %%%%%%%%%%%%%%%%%%

%% Gráfico %%
% 107708 %
f = fopen('speedrun_sol1_107708.txt','r'); % abrir ficheiro para leitura
mydata1 = textscan(f,'%f%f'); % separar as duas colunas por tab
md1 = mydata1{1,1}; % coluna n
md2 = mydata1{1,2}; % coluna cpu-time
fclose(f);
% 107961 %
```

```
f = fopen('speedrun_sol1_107961.txt','r'); % abrir ficheiro para leitura
mydata2 = textscan(f,'%f%f'); % separar as duas colunas por tab
md3 = mydata2{1,2}; % coluna cpu-time
fclose(f);
% 108352 %
f = fopen('speedrun_sol1_108352.txt','r'); % abrir ficheiro para leitura
mydata3 = textscan(f,'%f%f'); % separar as duas colunas por tab
md4 = mydata3{1,2}; % coluna cpu-time
fclose(f);
% fazer grafico %
figure(3);
plot(md1,md2,md1,md3,md1,md4,'Marker','.');
legend('107708','107961','108352','FontSize',10);
xlabel('n','FontSize',10); % legendar eixo dos x
ylabel('cpu-time','FontSize',10); % legendar eixo dos y
title('Solução 1'); % dar título

%% Solução 2 %%%%%%%%%%%%%%%%%%

%% Gráfico %%
% 107708 %
f = fopen('speedrun_sol2_107708.txt','r'); % abrir ficheiro para leitura
mydata1 = textscan(f,'%f%f'); % separar as duas colunas por tab
md1 = mydata1{1,1}; % coluna n
md2 = mydata1{1,2}; % coluna cpu-time
fclose(f);
% 107961 %
f = fopen('speedrun_sol2_107961.txt','r'); % abrir ficheiro para leitura
mydata2 = textscan(f,'%f%f'); % separar as duas colunas por tab
md3 = mydata2{1,2}; % coluna cpu-time
fclose(f);
% 108352 %
f = fopen('speedrun_sol2_108352.txt','r'); % abrir ficheiro para leitura
mydata3 = textscan(f,'%f%f'); % separar as duas colunas por tab
md4 = mydata3{1,2}; % coluna cpu-time
fclose(f);
% fazer grafico %
figure(4);
plot(md1,md2,md1,md3,md1,md4,'Marker','.');
legend('107708','107961','108352','FontSize',10);
xlabel('n','FontSize',10); % legendar eixo dos x
ylabel('cpu-time','FontSize',10); % legendar eixo dos y
title('Solução 2'); % dar título

%% Solução 3 %%%%%%%%%%%%%%%%%%

%% Gráfico %%
% 107708 %
f = fopen('speedrun_sol3_107708.txt','r'); % abrir ficheiro para leitura
mydata1 = textscan(f,'%f%f'); % separar as duas colunas por tab
md1 = mydata1{1,1}; % coluna n
md2 = mydata1{1,2}; % coluna cpu-time
fclose(f);
% 107961 %
f = fopen('speedrun_sol3_107961.txt','r'); % abrir ficheiro para leitura
mydata2 = textscan(f,'%f%f'); % separar as duas colunas por tab
md3 = mydata2{1,2}; % coluna cpu-time
fclose(f);
% 108352 %
```

```
f = fopen('speedrun_sol3_108352.txt','r'); % abrir ficheiro para leitura
mydata3 = textscan(f,'%f%f'); % separar as duas colunas por tab
md4 = mydata3{1,2}; % coluna cpu-time
fclose(f);
% fazer grafico %
figure(5);
plot(md1,md2,md1,md3,md1,md4,'Marker','.');
legend('107708','107961','108352','FontSize',10);
xlabel('n','FontSize',10); % legendar eixo dos x
ylabel('cpu-time','FontSize',10); % legendar eixo dos y
title('Solução 3'); % dar título
```