Trabalho Speed Run

1º Relatório

Diogo Ferreira – 99984 (33%)

Miguel Miragaia – 108317 (33%)

Gonçalo Lopes – 107572 (33%)

DETI

Universidade Aveiro

03/12/2022

Índice

Lista de Figuras 4
1 - Introdução 4
2 Métodos 5
2.1 - Recursivo Base 5
2.2 - Recursivo Melhorado 5
2.3 - Recursivo Final 6
3 - Resultados 7
3.1 - Recursivo Base 7
3.2 - Recursivo Melhorado 9
3.3 - Recursivo Final 12
4 - Discussão de Resultados e Conclusão 19

Índice de figuras

Figura 1 – Valores usando o método recursivo fornecido. 9

Figura 2 – Gráfico correspondente aos valores obtidos com o uso do método recursivo base. 9

Figura 3 – Gráfico corresponde aos valores obtidos devidos á melhoria do código usado na Figura 1, usando o método recursivo melhorado. 10

Figura 4 – Juntando ambas as Figuras 1 e 2 obtém-se uma melhor capacidade de se observar e comparar ambos os métodos usados. 12

Figura 5 – Valores obtidos usando o método recursivo final. 17

Figura 6 – Comparação do tempo de execução de todos os métodos utilizados ao longo do relatório. 18

Nota: Todos os valores utilizados nunca são totalmente iguais a zero, tendo os valores elevadas casas decimais iguais a zero, preferiu-se assim reduzir essas para conveniência de leitura e formatação

Lista de Figuras

1 Para analisarmos o quão eficiente é o código recursivo base do speed run, traçamos um gráfico com o tempo, **s**, que o programa levou a calcular as soluções do problema, para cada ponto **n**, sendo assim o tempo (**s**) uma função de **n**.

Neste gráfico podemos notar claramente que quanto maior for o **n**, maior é o tempo que o programa precisa para calcular o resultado, obtendo um crescimento exponencial. O desvio começa relativamente pequeno, mas logo torna-se algo muito negativo.

2 Igualmente á Figura 1, traçamos o gráfico corresponde ao método melhorado da função recursiva base, tendo novamente o tempo (**s**) uma função de **n**.

3 Resulta da unificação de ambos os gráficos das Figuras 1 e 2, com o propósito de se obter uma melhor perspetiva entre as diferenças das resoluções.

4 Semelhantemente á Figura 3, unimos as Figuras 1 e 2 com um novo gráfico, o do método recursivo final, para termos uma melhor perspetiva sobre todas as resoluções.

1 - Introdução

Neste relatório iremos explicar e descrever o método de resolução que usamos para computar soluções para o trabalho A01, Speed Run.

Este problema consiste em obter o número mínimo de movimentos para chegarmos á posição final (*n* = 800) cumprindo certos limites de velocidade, tendo assim de acabar com velocidade igual a 1, sendo o objetivo principal conseguir essa posição final com um tempo de execução de microssegundos.

2 Métodos

2.1 - Recursivo Base

O código dado usa funções recursivas para computar o número de passos precisos para se chegar à penúltima posição com velocidade 1. O problema com esta implementação é que o programa para cada passo precisa de verificar cada posição individualmente fazendo com que o tempo preciso para achar uma solução vá aumentando exponencialmente á medida que a posição também aumenta.

2.2 - Recursivo Melhorado

Para o nosso primeiro método de resolução nós aproveitamos algum do código dado pelo professor e tentamos melhorar a sua implementação recursiva. Seguindo assim uma sugestão do professor das aulas práticas, inicializamos um array bidimensional, *bestnmoves*, com o *max_road_size* e o *max_road_speed* onde iremos posteriormente guardar os valores das posições e os seus respetivos speeds permitidos para essa mesma posição. Este processo é executado através de um *for* duplo que primeiramente percorre todas as posições possíveis e para cada uma delas percorre também as velocidades permitidas nessa posição e introduz no array bidimensional *bestnmoves* um valor impossível

(*final_position* + 100) para que deste modo, inicialmente não seja possível começar numa posição melhor que a que temos.

Utilizando parte do código dado pelo professor usamos uma condição *if* onde iremos verificar se o *bestnmoves* já terá guardado nele melhores valores de posição e velocidade que correspondem a um número de movimentos, onde no caso negativo, atualiza os seus valores para os melhores (número de movimentos atual) e no caso positivo, ignora esses mesmos valores. Esta implementação permite ao programa ignorar os piores casos possíveis fazendo a sua velocidade melhorar consideravelmente ao método inicial, mas provando-se insuficiente no final.

2.3 - Recursivo Final

Sendo ambos o código fornecido bem como o ***recursivo melhorado*** métodos de resolução insuficientes para obtermos a solução pretendida tivemos de mais uma vez desenvolver o código à procura de mais uma forma de podermos resolver o problema.

Esta solução encontrou-se devido ao desenvolvimento da *solution_1_recursion* pertencente ao código base, onde tal como para o método anterior esta implementação será mencionada por ***recursivo final***.

Nesta nova iteração, adicionamos uma nova condição quando a *solution_1_recursion* faz o teste para todas as velocidades legais em cada posição onde através da condição de verificação *if*, verificamos se a melhor posição atingida para o respetivo movimento (*solution_1_best.positions [move_number]*) é superior à posição atual para o mesmo movimento. Confirmandose é efetuado o *return*, ou seja, nao tentamos as velocidades legais para a posição atual, visto que estariamos a calcular algo que não seria utilizado e desta forma otimizamos o codigo em termos de velocidade. Porém se a condição *if* não for

verificada somos obrigados a testar as velocidades legais pois temos uma nova melhor posição para o movimento. Este teste é feito com um ciclo *for* que testa para a velocidade a manter-se, aumentar ou diminuir em uma unidade de forma decrescente, ou seja, começando no *speed* com maior valor até ao de menor.

Desta maneira conseguimos resolver o problema e obter a posição final em apenas **253** iterações.

Todo o código usado neste trabalho estará nos diferentes Anexos.

3 - Resultados

3.1 - Recursivo Base

Na Tabela 1 bem como na Figura 1 podemos ver os resultados correspondentes ao método usado no código fornecido, tendo apenas corrido o programa para n = 40.

N	***Solution***	***Count***	***Time***
1	1	2	0.000e+00
2	2	3	0.000e+00
3	3	5	0.000e+00
4	3	8	0.000e+00
5	4	13	0.000e+00
6	4	22	0.000e+00
7	5	36	0.000e+00
8	5	30	0.000e+00

9	5	100	0.000e+00
10	6	167	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	2165	0.000e+00
16	7	3614	0.000e+00
17	8	6031	0.000e+00
18	8	10065	0.000e+00
19	8	16795	0.000e+00
20	8	28024	0.000e+00
21	9	46758	0.000e+00
22	9	78011	0.000e+00
23	9	130089	0.000e+00
24	9	216968	0.000e+00
25	9	359706	0.000e+00
26	10	597823	0.000e+00
27	10	995046	1.562e-02
28	10	1655498	0.000e+00
29	10	2757259	3.125e-02
30	10	4593012	1.562e-02
31	11	7651017	4.688e-02
32	11	12747967	7.812e-02
33	11	21239691	1.406e-01
34	12	35390165	2.031e-01
35	12	58969547	3.750e-01
36	12	98258424	5.937e-01

37	13	163727428	1.016e+00
38	13	272817267	1.719e+00
39	13	454593881	2.891e+00
40	14	757489987	4.781e+00

Figura 1 – Valores usando o método recursivo fornecido.

Focando apenas nos valores de ***N*** e ***Time***, podemos traçar assim o seu gráfico:

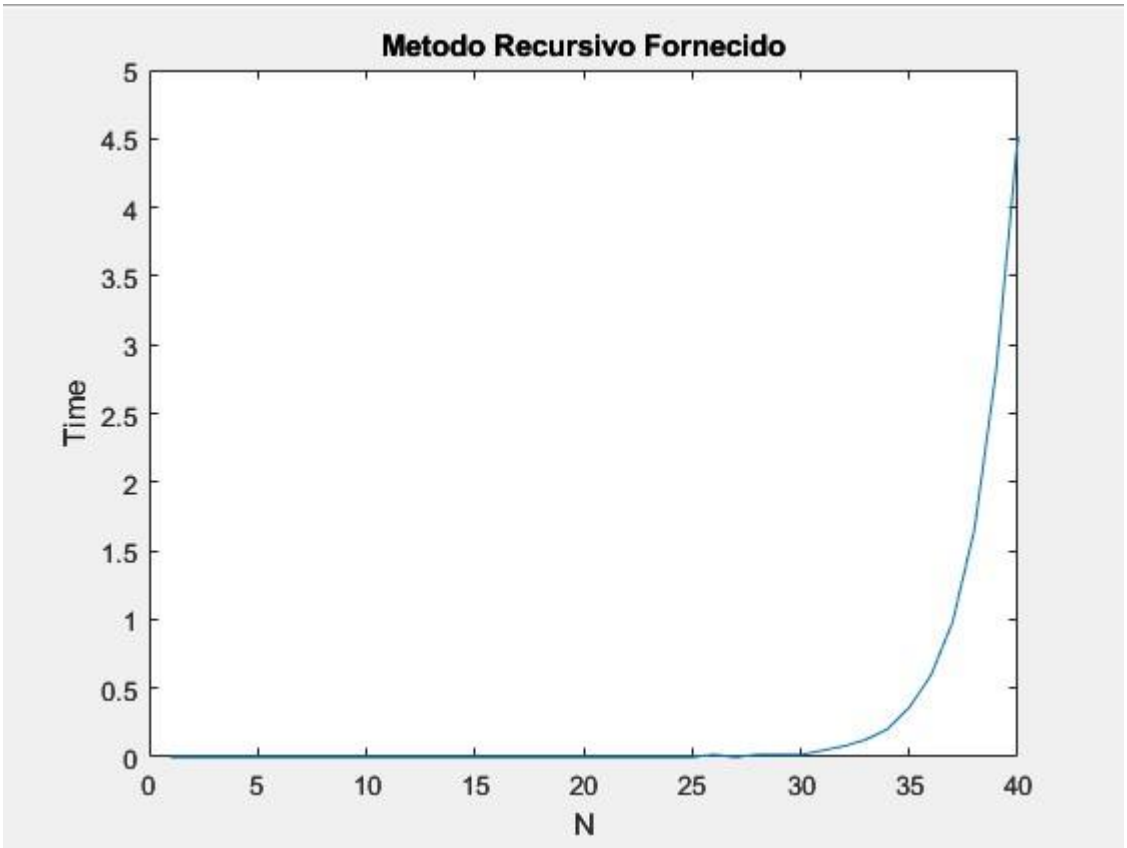


Figura 2 – Gráfico correspondente aos valores obtidos com o uso do método recursivo base.

3.2 - Recursivo Melhorado

Usando agora o método ***recursivo melhorado***, o programa consegue alcançar uma posição n = 300, sendo que passando deste ponto encontramos o mesmo problema do código base onde o tempo de execução começa a aumentar

exponencialmente ao ponto de se tornar impossível resolver este trabalho.

Com o objetivo de termos uma perspetiva da melhoria de um código para o outro, traçámos da mesma forma anterior o gráfico da Figura 2.

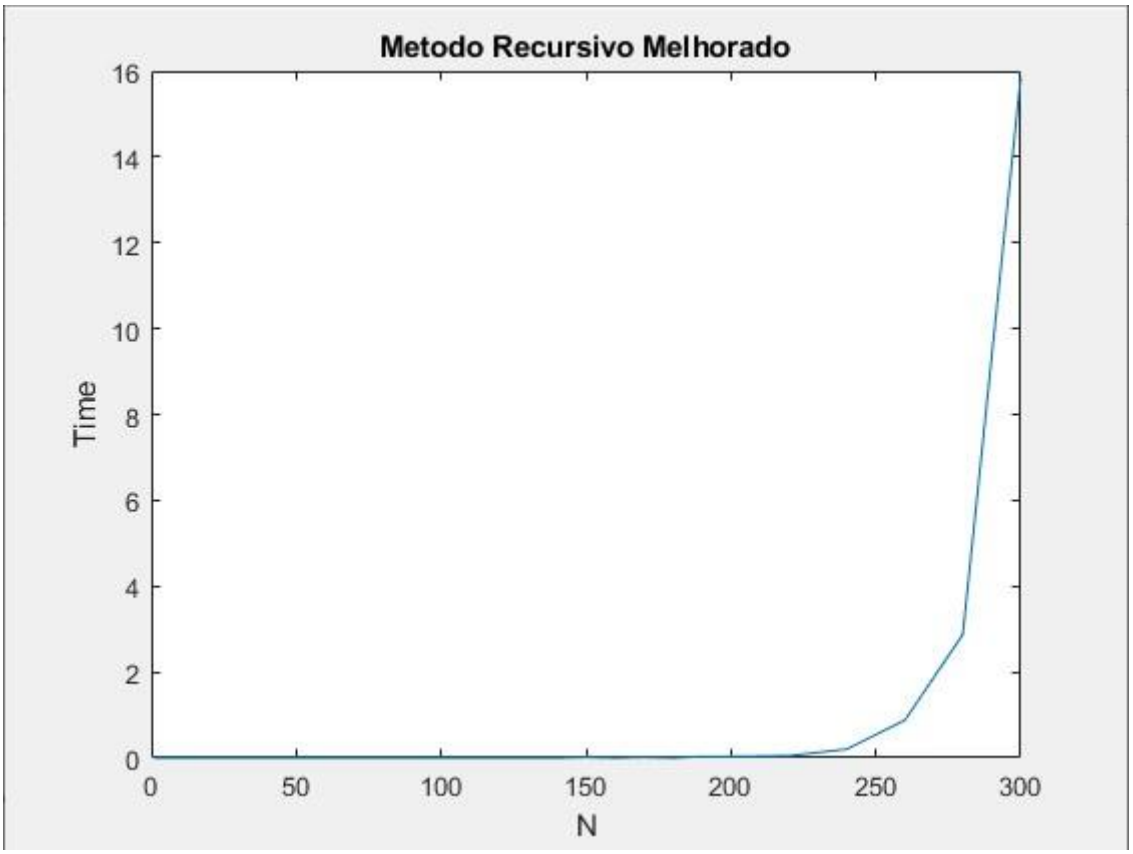


Figura 3 – Gráfico corresponde aos valores obtidos devidos á melhoria do código usado na Figura 1, usando o método recursivo melhorado.

Com a Figura 2, é notável a diferença que a implementação do array *bestnmoves* teve, ao ignorar os casos por onde o programa já tinha estado focando apenas em arranjar os melhores casos possíveis.

Ainda assim a implementação fica longe do resultado esperado visto que começamos a notar um ligeiro aumento de tempo no ponto n = 180, outro aumento mais significativo em n = 260 atingindo assim o seu máximo em n = 300, ficando aquém da almejada posição final n = 800.

Não será fornecido a Tabela dos valores usados na Figura 2 devido a sua quantidade de valores, inconsequentes, visto que se trata de uma abordagem falha em relação aos objetivos propostos.

Uma melhor exemplificação entre as diferenças de cada método é fornecida pela Figura 3.

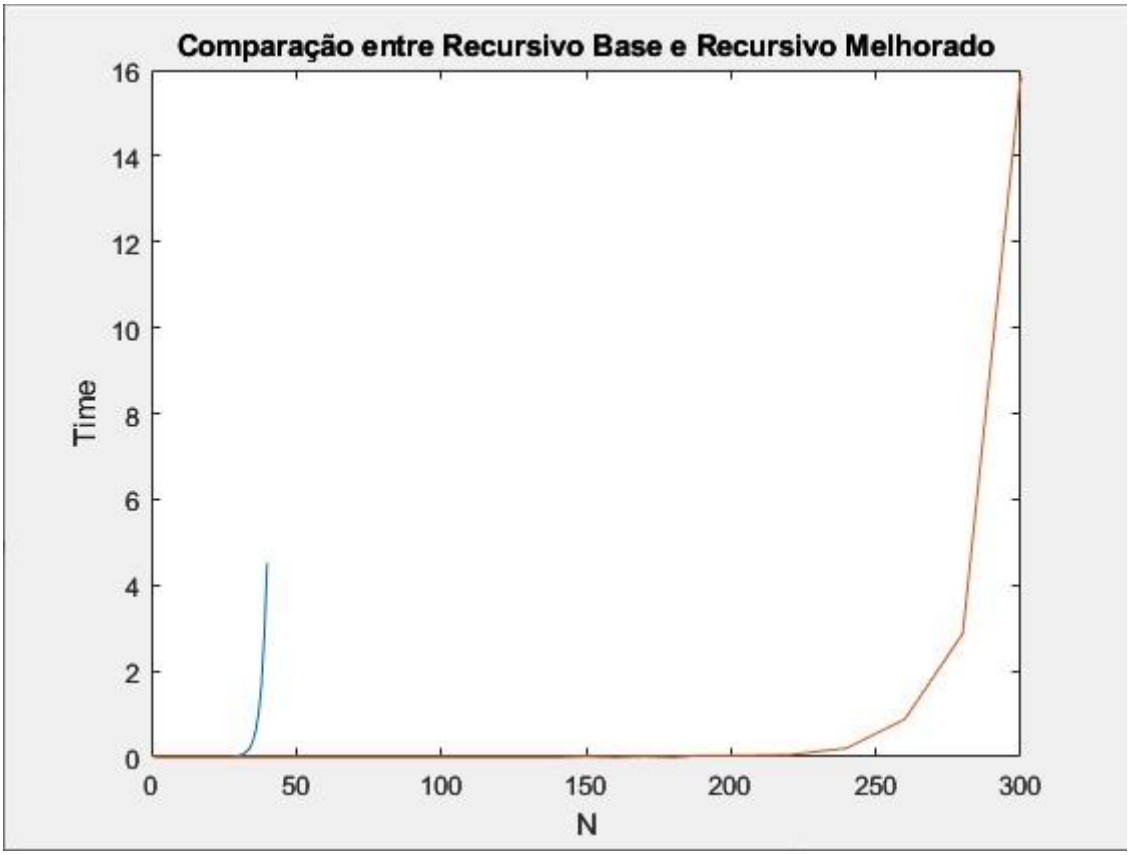


Figura 4 – Juntando ambas as Figuras 1 e 2 obtém-se uma melhor capacidade de se observar e comparar ambos os métodos usados.

3.3 - Recursivo Final

Segue na Tabela 2, todos os valores obtidos graças a este método, visto que foi o único desenvolvido durante o trabalho capaz de cumprir o objetivo traçado inicialmente.

Subsequentemente irá ser fornecido a Figura 4, onde identicamente á Figura 3, uniram-se os gráficos dos tempos de execução de todos os métodos utilizados ao longo do relatório.

N	***Solution***	***Count***	***Time***
1	1	2	0.000000e+00

2	2	3	0.000000e+00
3	3	5	0.000000e+00
4	3	5	0.000000e+00
5	4	7	0.000000e+00
6	4	8	0.000000e+00
7	5	10	0.000000e+00
8	5	12	0.000000e+00
9	5	10	0.000000e+00
10	6	13	0.000000e+00
11	6	15	0.000000e+00
12	6	13	0.000000e+00
13	7	17	0.000000e+00
14	7	20	0.000000e+00
15	7	20	0.000000e+00
16	7	16	0.000000e+00
17	8	20	0.000000e+00
18	8	23	0.000000e+00
19	8	24	0.000000e+00
20	8	21	0.000000e+00

21	9	25	0.000000e+00
22	9	29	0.000000e+00
23	9	30	0.000000e+00
24	9	28	0.000000e+00
25	9	21	0.000000e+00
26	10	24	0.000000e+00
27	10	27	0.000000e+00
28	10	26	0.000000e+00
29	10	24	0.000000e+00
30	10	20	0.000000e+00
31	11	22	0.000000e+00
32	11	24	0.000000e+00
33	11	22	0.000000e+00
34	12	24	0.000000e+00
35	12	26	0.000000e+00
36	12	24	0.000000e+00
37	13	26	0.000000e+00
38	13	28	0.000000e+00
39	13	26	0.000000e+00

40	14	28	0.000000e+00
41	14	30	0.000000e+00

42	14	28	0.000000e+00
43	15	31	0.000000e+00
44	15	33	0.000000e+00
45	15	31	0.000000e+00
46	16	34	0.000000e+00
47	16	36	0.000000e+00
48	16	35	0.000000e+00
49	16	32	0.000000e+00
50	17	34	0.000000e+00
55	19	37	0.000000e+00
60	21	42	0.000000e+00
65	23	48	0.000000e+00
70	25	53	0.000000e+00
75	26	59	0.000000e+00
80	27	68	0.000000e+00
85	28	73	0.000000e+00
90	30	66	0.000000e+00

95	32	70	0.000000e+00
100	35	76	0.000000e+00
110	39	87	0.000000e+00
120	42	98	0.000000e+00

130	44	108	0.000000e+00
140	46	113	0.000000e+00
150	48	115	0.000000e+00
160	52	110	0.000000e+00
170	57	121	0.000000e+00
180	60	131	0.000000e+00
190	62	146	0.000000e+00
200	64	146	0.000000e+00
220	70	146	0.000000e+00
240	77	168	0.000000e+00
260	81	171	0.000000e+00
280	88	182	0.000000e+00
300	96	205	0.000000e+00
320	101	213	0.000000e+00
340	111	235	0.000000e+00

360	115	244	0.000000e+00
380	120	259	0.000000e+00
400	128	270	0.000000e+00
420	134	291	0.000000e+00
440	139	293	0.000000e+00
460	147	310	0.000000e+00
480	152	324	0.000000e+00
500	159	332	0.000000e+00
520	166	348	0.000000e+00
540	171	379	0.000000e+00
560	178	375	0.000000e+00
580	185	398	0.000000e+00
600	189	415	0.000000e+00
620	195	410	0.000000e+00
640	203	431	0.000000e+00
660	208	440	0.000000e+00
680	214	452	0.000000e+00
700	222	475	0.000000e+00
720	227	484	0.000000e+00
740	236	506	0.000000e+00

760	241	521	0.000000e+00
780	244	526	0.000000e+00
800	253	538	0.000000e+00

Figura 5 – Valores obtidos usando o método recursivo final.

Com a Tabela 2 podemos observar que ao longo de todas as iterações o Tempo é sempre 0, sendo assim a solução final resolvida instantaneamente, sendo uma melhoria imensa em comparação com os outros métodos.

Com o propósito de se analisar além, aumentamos a escala de amostra das casa decimais para ordens de 50, sendo sempre mostrado um tempo de execução de 0 segundos. Assim preferimos usar apenas 6 casas decimais, na forma de microssegundos.

Finalmente, para se obter mais uma vez uma melhor perspetiva da diferença entre as resoluções até agora, teremos a Figura 4, obtida da mesma maneira que a Figura 1, onde serão ilustrados todos os gráficos dos métodos usados ao longo deste relatório.

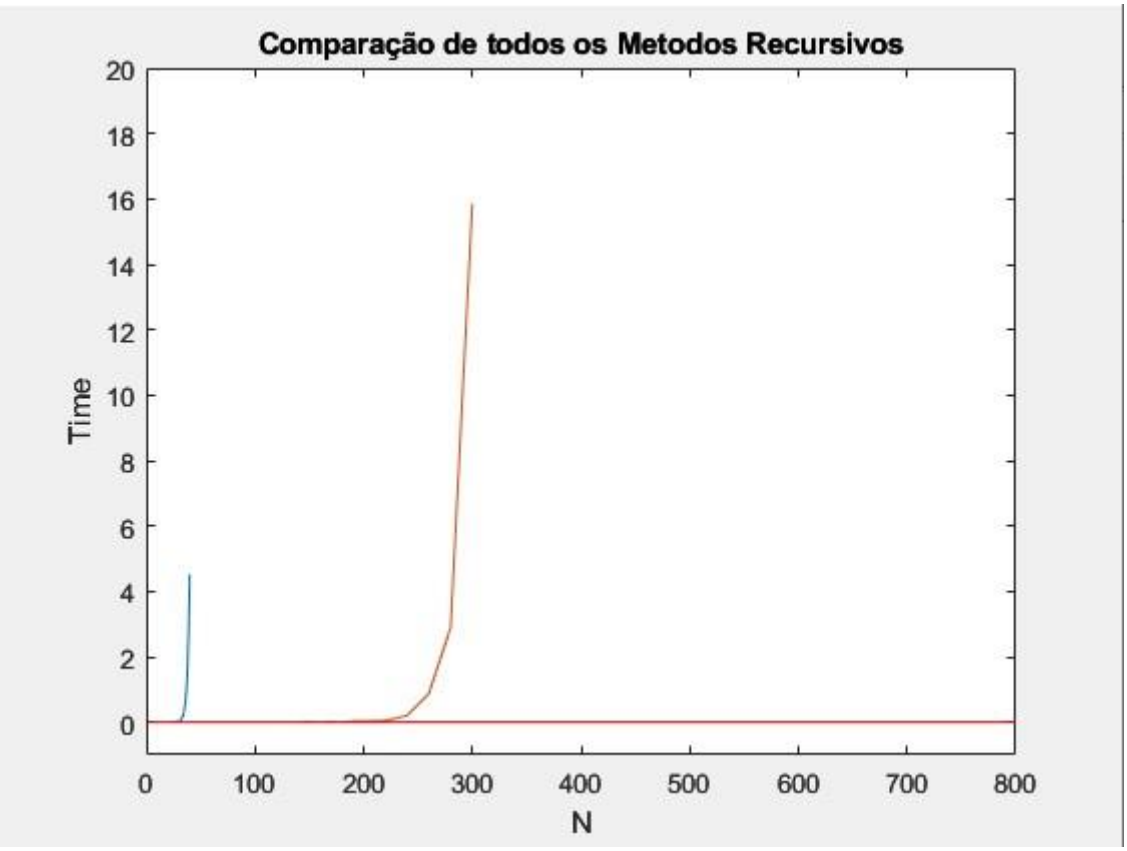


Figura 6 – Comparação do tempo de execução de todos os métodos utilizados ao longo do relatório.

Graças á Figura 4 consegue-se observar que mesmo muito além do ponto *n* = 300, a função foi resolvida praticamente instantaneamente, como mencionado ainda em instantes, sendo assim de facto possível resolver este problema em questão de microssegundos e ainda melhor como no nosso caso.

4 - Discussão de Resultados e Conclusão

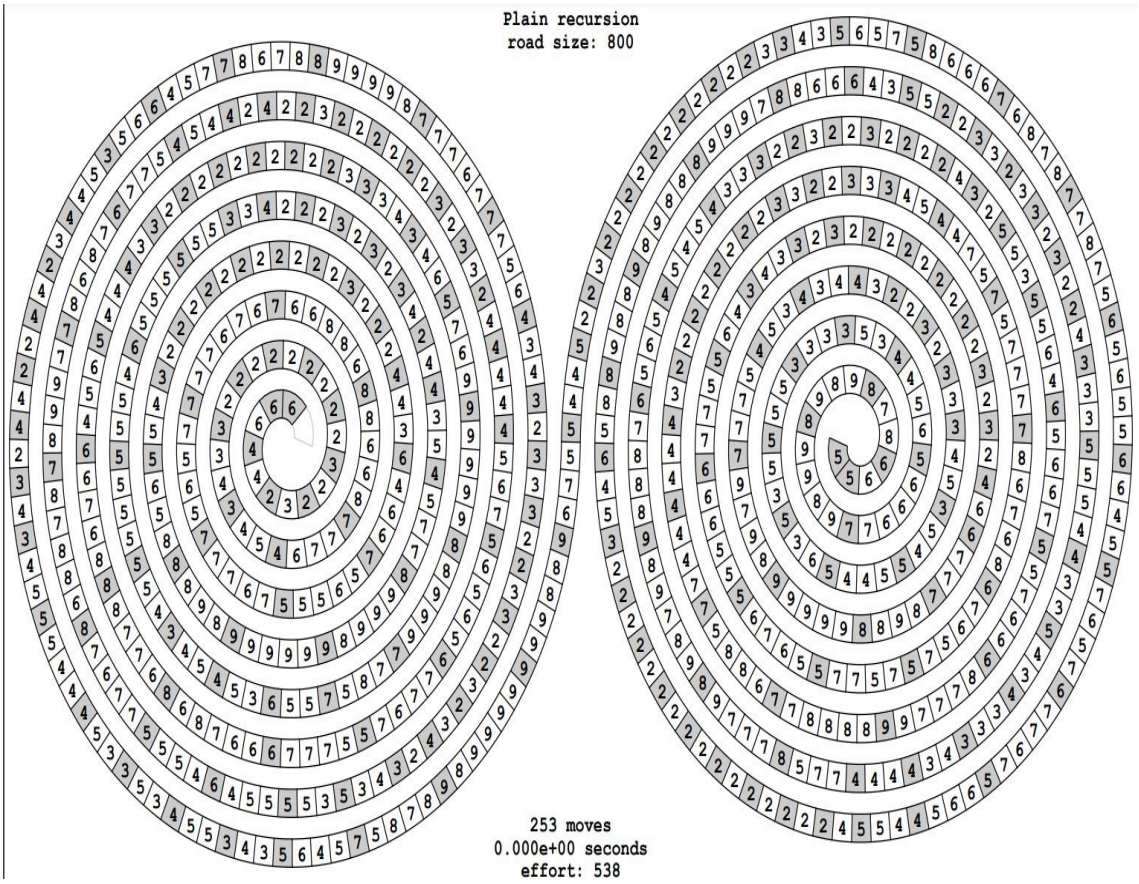
Ao longo do relatório apresentámos dentro da forma recursiva, duas diferentes formas que usámos para resolver o problema proposto do trabalho *speed run*.

Observámos que com o método ***recursivo melhorado*** em relação ao código fornecido para o trabalho, obtemos uma melhoria significativa, donde passamos de um n = 40 para um n = 300. Mesmo com este aumento, não cumpriu a ambição de resolver o problema em questão de microssegundos, pois mesmo com um n = 300, notamos um aumento repentino do tempo de processamento em dois pontos em específico: n = 180 e n = 260.

Deste modo tivemos de repensar a nossa abordagem de modo a conseguirmos solucionar uma resolução que cumpra o objetivo de resolver em microssegundos.

Finalmente, chegamos à conclusão de que com o método ***recursivo melhorado*** se adicionássemos condições *if* sobre o movimento onde, se verificado, não testa as velocidades poupando tempo de processamento, criando-se assim o método ***recursivo final***. Com estas exclusões somos capazes de chegar à posição final n = 800, tendo sido efetuadas no total **253** iterações, com um tempo de execução praticamente instantâneo.

Com este resultado infinitamente melhor em relação aos restante é de facto possível cumprir o objetivo inicial.


Plain recursion
road size: 800
253 moves
0.000e+00 seconds
effort: 538

Anexo 1

Código fornecido

```
//
// 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
//   N.Mec. XXXXXX  Name: XXXXXXX
//



//
// 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"



//
// 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
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;
}



//
// 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,solutio
n_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) ? 0x108317 : 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_b
est.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_
}
```

Anexo 2

Recursivo Melhorado

```
//
// 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
//   N.Mec. XXXXXX  Name: XXXXXXX
//


//
// 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"


//
// 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_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
int bestnmoves[_max_road_size_][_max_road_speed_];
//static double solution_for_elapsed_time; // time it took to solve 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;
```

```
  //if bestnmoves[position][speed] better than the ones of move number
then continue
  if (bestnmoves[position][speed] < move_number){
    return;
  }
  bestnmoves[position][speed] = move_number;

  // is it a solution?
  if(position == final_position && speed == 1)
  {
    // is it a better solution?
    //printf("Soluçao com  %d moves \n",move_number);
    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--)   //edit:
alterei de forma a percorrer os speeds na forma inversa e ganhar tempo
    if(new_speed >= 1 && new_speed <= _max_road_speed_ && position +
new_speed <= final_position)
    {
      for(i = 1;i <= new_speed && new_speed <= max_road_speed[position +
i];i++)  //alterei o primeiro parametro para 1 em vez de 0
        ;
      if(i > new_speed)
        solution_2_recursion(move_number + 1,position +
new_speed,new_speed,final_position);
    }
}

//new solve
static void solve_2(int final_position)
{
  //for duplo, inicializar o bestnmoves para a posicao final_position e
final_speed ( em baixo)
  for (int p=0;p<=final_position;p++){
    for(int s=1;s<max_road_speed[p];s++){
      bestnmoves[p][s]=final_position + 100;
    }
  }
  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;
}
//
// example of the slides
//

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

  srandom(0x108317);
  init_road_speeds();
  final_position = 30;
  solve_2(final_position);
  make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],sol
ution_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) ? 108536 : 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_2_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_)
  {
    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);
    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],s
olution_2_best.n_moves,&solution_2_best.positions[0],solution_2_elapsed_ti
me,solution_2_count,"Plain recursion");
      }
      printf(" %3d %16lu %9.3e
|",solution_2_best.n_moves,solution_2_count,solution_2_elapsed_time);
    }
    else
    {
      //bestnmoves = -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_
}
```

Anexo 3

Recursivo Final

```
//
// AED, August 2022 (TomÃ¡s 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
//   N.Mec. XXXXXX  Name: XXXXXXX
//


//
// 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"
```

```c
#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;


//
// 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

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
  if (solution_1_best.positions[move_number] >
solution_1.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_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;
}


//
// example of the slides
//

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

  srandom(0x108317);
  init_road_speeds();
  final_position = 30;
  solve_1(final_position);
  make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],sol
ution_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();
  // 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_)
  {
    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],s
olution_1_best.n_moves,&solution_1_best.positions[0],solution_1_elapsed_ti
me,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_
}
```