universidade de aveiro

deti departamento de eletrónica, telecomunicações e informática

Trabalho SpeedRun

Algoritmos e Estruturas de Dados

Universidade de Aveiro

Licenciatura em Engenharia de Computadores e Informática

André Silva 98651

50%

Isaac Moura 105065

50%

8 dezembro 2022

Índice

Introdução ao Problema

Uma estrada é dividida em segmentos, cada um com o mesmo comprimento e com um limite de velocidade. A velocidade é determinada pelo número de segmentos que um carro é capaz de avançar com um único movimento.

Em cada movimento o carro pode:

1) Manter a velocidade
2) Aumentar a sua velocidade (+1)
3) Diminuir a sua velocidade (-1)

O carro é colocado no primeiro segmento de estrada com uma velocidade igual a 0. É pretendido que o carro chegue ao último segmento de estrada com **velocidade igual a 1** para que consiga travar e acabar com velocidade igual a 0.

Objetivo: Determinar o número mínimo de movimentos necessários para alcançar a posição final.

Variáveis:

1) Posição do carro: integer position
2) Posição Final: integer final_position
3) Velocidade: integer speed

Movimento do carro:

1. Escolher nova velocidade (new_speed): Pode ser speed – 1, speed ou speed +1.
2. Avançar para a nova posição: new_position = position + new_speed.
3. Verificar condição (Não pode exceder limite de velocidade):
   New_speed <= max_road_speed[position + i]

Estas são as condições que formam o movimento do carro e que moldam a resolução do problema.

Solução Original:

Para a resolução deste problema uma solução é encontrada se chegarmos ao fim da estrada, ou seja, à última posição, com uma velocidade de 1, de modo a pararmos. Esta condição é representada pelo excerto de código seguinte:

```
// record move
  solution_1_count++;
  solution_1.positions[move_number] = position;
  // is it a solution?
  if(position == final_position && speed == 1)
  {
```

Nesta resolução vão sendo geradas soluções que, se forem melhores que a anterior, passam a ser a solução temporária final. A solução final terá o número mínimo de saltos que o carro percorre na estrada até chegar ao fim. Código de comparação das soluções:

```
// 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;
  }
```

Se a solução encontrada não é a melhor temos de continuar a testar. Temos de garantir que a velocidade do carro se mantém positiva e sempre menor que o limite de velocidade da secção em que se encontra. Temos de garantir também que não estamos na posição final pois é uma das condições de paragem. Código:

```
// 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)
    {
```

Garantidas as condições de partida vamos testando todas as velocidades legais, o que aumenta imenso a complexidade do programa assim como o tempo de execução até encontrar a melhor solução.

A função é recursiva então sempre que o ciclo é terminado é novamente chamada, incrementando o número de saltos assim como atualizando a nova posição.

Assim, a cada nova posição é necessário calcular todas as possibilidades de descolamento do carro.

```
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);
    }
}
```

Performance da solução:

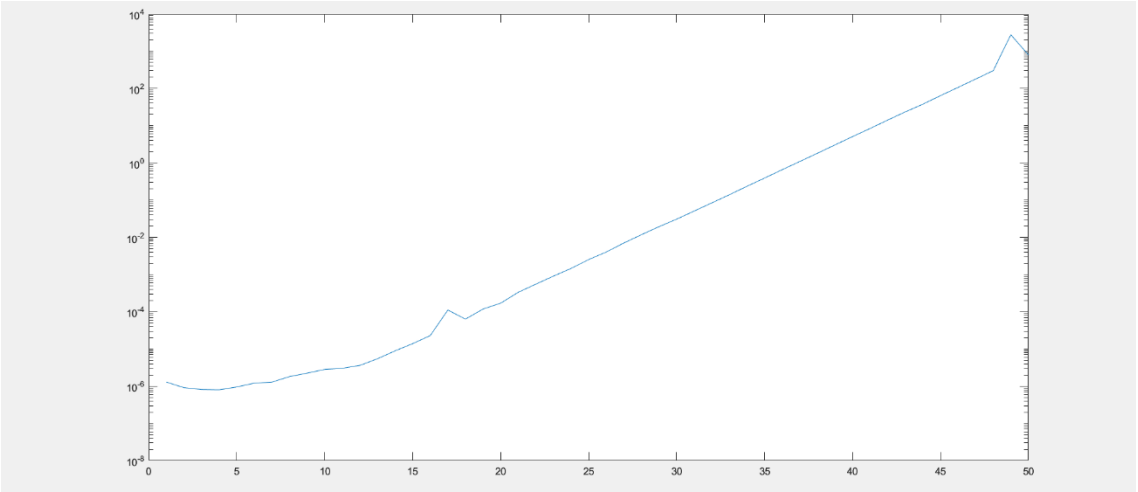
Esta solução é muito ineficiente sendo que nunca será possível chegar perto do objetivo de 800 posições. Para um tempo limite de 1 hora, a posição alcançada pelo programa foi de 50 sendo que neste último passo o programa precisou de 795 segundos para encontrar a solução.

Teste na linha de comandos:

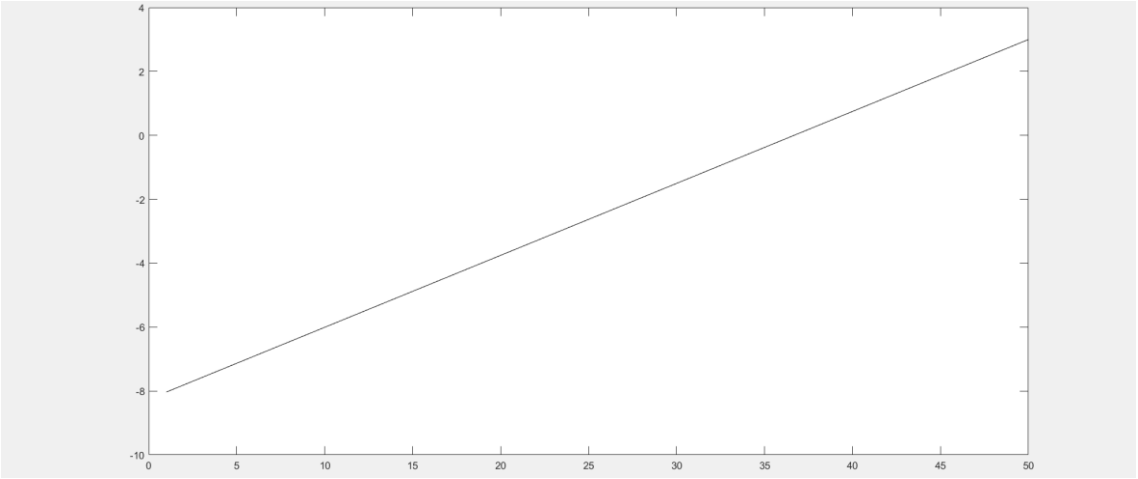
```
36 |  13        98146882 6.507e-01 |
37 |  13       164383779 1.094e+00 |
38 |  13       274754730 1.820e+00 |
39 |  14       458665608 3.002e+00 |
40 |  14       765958555 5.009e+00 |
41 |  14      1278845073 8.579e+00 |
42 |  15      2134309310 1.409e+01 |
43 |  15      3561456082 2.339e+01 |
44 |  15      5942883738 3.845e+01 |
45 |  15      9857304601 6.580e+01 |
46 |  16     16386653246 1.092e+02 |
47 |  16     27279154773 1.828e+02 |
48 |  16     45398760074 3.021e+02 |
49 |  16     72586704974 4.901e+02 |
50 |  17    117894255175 7.955e+02 |
```

Análise:

Com os dados da execução anterior pretendemos observar como o programa se comporta através de uma função entre os tempos de execução (eixo Y) e a posição alcançada (eixo X).



Podemos, em matlab aproximar os resultados com uma regressão linear:



Deste modo queremos calcular o tempo que o programa levaria a alcançar a posição 800 assim como obter uma fórmula que fornece a estimativa dos tempos de execução.

Usando esta solução o programa levaria **3.12 e140** anos para uma estrada de 800 segmentos.

Solução Final

A estrutura inicial da solução é idêntica à anterior uma vez que os critérios de movimento do carro se mantém assim como a condição para chegar ao fim da estrada. Código:

```
static solution_t solution_fast,solution_fast_best;
static double solution_fast_elapsed_time; // time it took to solve
the problem
static unsigned long solution_fast_count; // effort dispended
solving the problem
static int solution_fast_recursion(int move_number,int position,int
speed,int final_position){
  int i,new_speed = 0;

  // record move
  solution_fast_count++;
  solution_fast.positions[move_number] = position;
  // is it a solution?
  if(position == final_position && speed == 1)
  {
  // is it a better solution?
  if(move_number < solution_fast_best.n_moves)
  {
    solution_fast_best = solution_fast;
    solution_fast_best.n_moves = move_number;
  }
  return 1;
  }
```

Foi utilizado um sistema de controlo binário para identificar quando se encontra uma solução e essa solução é melhor. Assim temos return 1 ou return 0.

Nesta solução começamos a testar as velocidades maiores ao contrário da solução original:

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

Mantemos a parte que garante que o limite de velocidade do segmento não é excedido com a nova velocidade:

```
if(new_speed >= 1 && new_speed <= _max_road_speed_ && position +
new_speed <= final_position)
    {
```

Tal como na solução original vamos verificando se não ultrapassamos o limite de velocidade ao longo dos segmentos seguintes:

```
for(i = 0;i <= new_speed && new_speed <= max_road_speed[position +
i];i++);
```

A posição é atualizada a cada iteração:

```
int new_pos = position + new_speed;
```

Quando o ciclo termina (como na primeira solução) é testado se chegamos ao fim da estrada com uma velocidade de 1.

```
if(i > new_speed){
        if(new_speed != 1 && new_pos == final_position){
          continue;
        }
```

Se isso não se verificar então não temos uma solução pelo que temos de chamar a função novamente(recursividade):

```
int sol_fast = solution_fast_recursion(move_number + 1,position +
new_speed,new_speed,final_position);
```

Quando encontramos a solução, ou seja, obtemos um return de 1, a solução é guardada:

```
if(sol_fast == 1)
          return 1;
```

Performance da Solução:

Esta solução é extremamente mais eficiente do que a original uma vez que completa o objetivo do programa para uma estrada de 800 segmentos em microssegundos.

A otimização resulta do teste com velocidades maiores em vez de começar pelas velocidades menores. Vamos usando a posição em vez de estar sempre a recalcular com recursividade.

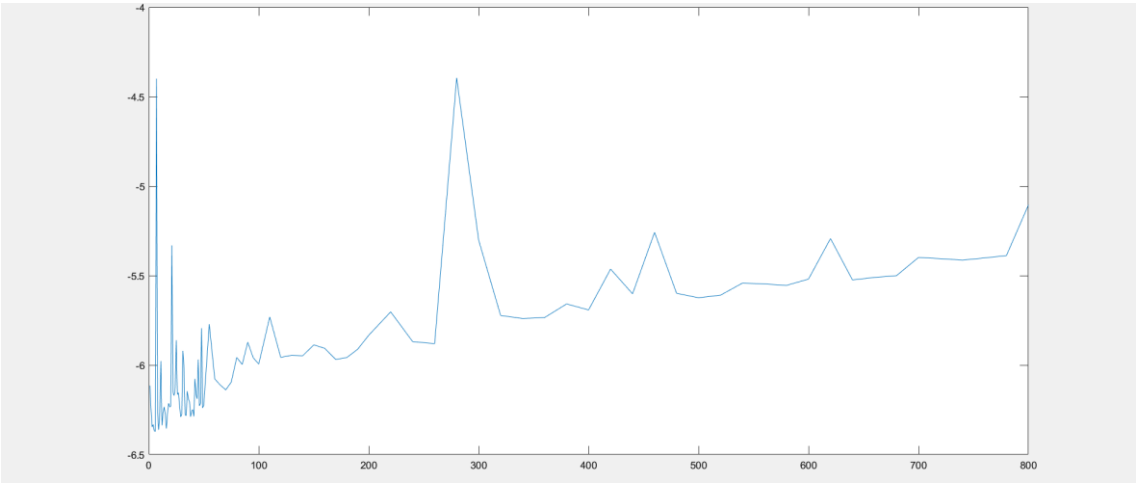
Apesar de a estrutura das soluções ser muito semelhante, pequenas alterações levaram a um aumento de performance de milhões de vezes permitindo chegar ao fim do problema em muito pouco tempo.

Teste na linha de comandos:

Análise:

Com os dados da execução anterior pretendemos observar como o programa se comporta através de uma função entre os tempos de execução (eixo Y) e a posição alcançada (eixo X).

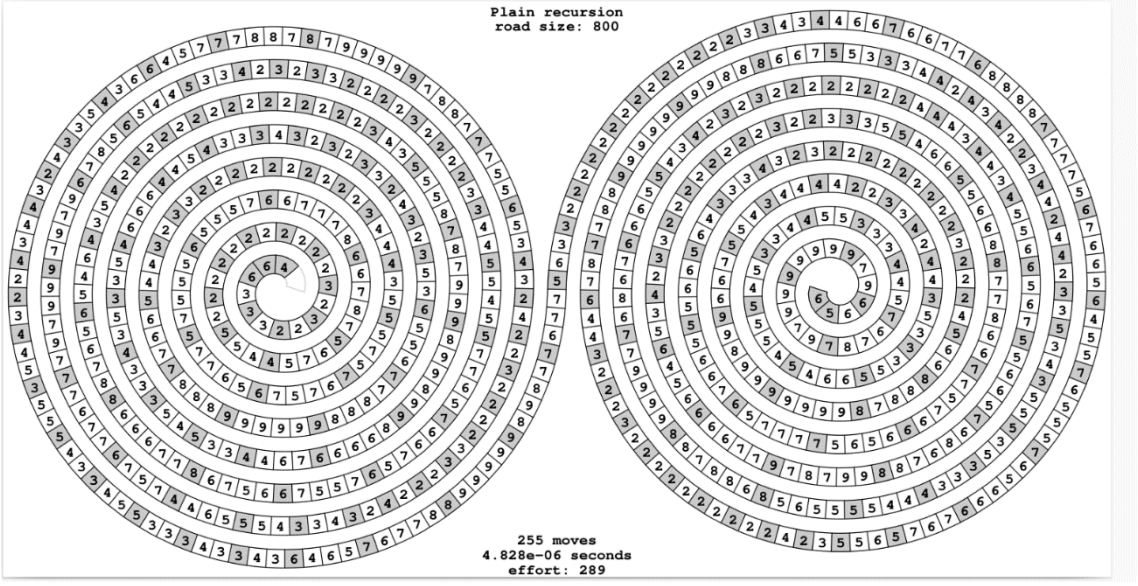
Foi aplicada uma escala logarítmica para maior perceção dos resultados.



Podemos observar algum “ruído” no início do gráfico que se deve ao facto de os valores temporais serem muito pequenos. A curva vai crescendo muito lentamente conforme o número de segmentos da estrada vai aumentado.

Assim podemos concluir que a solução encontrada mantém a sua rapidez e eficiência apesar do elevado número de segmentos. O tempo de execução do programa para encontrar a solução 800 foi de **4.75e-06 segundos**.

PDF gerado pelo programa:



Apêndice

Contém todo o código utilizado para a resolução do problema assim como implementação em MatLab usado no tratamento dos dados.

Código em MatLab:

```
A = load("tempos_iniciais.txt");

n = A(:,1);
t = A(:,2);

figure
plot(n,t);

semilogy(n,t);


t_log = log10(t);
plot(n,t_log)

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


Ntotal = [n n*0+1];
plot(n,Ntotal*Coefs,'k');
```

Tempos_iniciais.txt -> Dados extraídos do programa através da linha de comandos:

./speed_run 105065 > tempos_iniciais.txt

Código Completo do Programa:

```
//
// 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. 98651 Name: Andre Silva
//   N.mec. 105065  Name: Isaac Moura
//


//
// 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 solution_t solution_fast,solution_fast_best;
static double solution_fast_elapsed_time; // time it took to solve
the problem
static unsigned long solution_fast_count; // effort dispended
solving the problem
static int solution_fast_recursion(int move_number,int position,int
speed,int final_position){
  int i,new_speed = 0;

  // record move
  solution_fast_count++;
  solution_fast.positions[move_number] = position;
  // is it a solution?
  if(position == final_position && speed == 1)
  {
  // is it a better solution?
  if(move_number < solution_fast_best.n_moves)
```

```
  {
    solution_fast_best = solution_fast;
    solution_fast_best.n_moves = move_number;
  }
  return 1;
  }

  // no, try all legal speeds
  for(new_speed = speed +1 ;new_speed >= speed - 1;new_speed--){ //
Começa pelas velocidades maiores em vez de menores

    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++); // Verifica se new_speed é
valido para todos os segmentos

      int new_pos = position + new_speed;     // atualiza posição

      if(i > new_speed){
        if(new_speed != 1 && new_pos == final_position){  //
Termina trajeto (fim da estrada), nao acaba com speed 1
          continue;
        }



        int sol_fast = solution_fast_recursion(move_number +
1,position + new_speed,new_speed,final_position);
        if(sol_fast == 1)
          return 1;
      }
    }

  }
  return 0;
}
static void solve_fast(int final_position)
{
  if(final_position < 1 || final_position > _max_road_size_)
  {
    fprintf(stderr,"solve_1: bad final_position\n");
    exit(1);
  }
  solution_fast_elapsed_time = cpu_time();
```

```
  solution_fast_count = 0ul;
  solution_fast_best.n_moves = final_position + 100;
  solution_fast_recursion(0,0,0,final_position);
  solution_fast_elapsed_time = cpu_time() -
solution_fast_elapsed_time;
}


//
// example of the slides
//

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

  srandom(0xAED2022);
  init_road_speeds();
  final_position = 30;
  solve_fast(final_position);
  make_custom_pdf_file("example.pdf",final_position,&max_road_speed
[0],solution_fast_best.n_moves,&solution_fast_best.positions[0],sol
ution_fast_elapsed_time,solution_fast_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_fast_best.n_moves;i++)
    printf(" %d",solution_fast_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_fast_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_fast_elapsed_time < _time_limit_)
    {
      solve_fast(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_spe
ed[0],solution_fast_best.n_moves,&solution_fast_best.positions[0],s
olution_fast_elapsed_time,solution_fast_count,"Plain recursion");
      }
      printf(" %3d %16lu %9.3e
|",solution_fast_best.n_moves,solution_fast_count,solution_fast_ela
psed_time);
    }
    else
    {
      solution_fast_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_
}
```

PDF gerados extra: