AED - Relatório

Universidade de Aveiro(*UA*)

Anderson Lourenço, Sara Almeida, Hugo Correia

Projeto nº1 - Speed Run

Algoritmos e Estrutura de Dados

DETI - Departamento de Electrónica, Telecomunicações e Informática

Anderson Lourenço, Sara Almeida, Hugo Correia

(108579) aaokilourenco@ua.pt,

(108796) sarafalmeida@ua.pt,

(108215) hf.correia@ua.pt

5/11/2022

Índice

Capítulo 1

Introdução

Speed run

No contexto da disciplina de Algoritmos e Estrutura de Dados foi-nos proposto um primeiro assignment que tem como base um automóvel que tem de atravessar uma estrada divida em diversos segmentos com base em várias regras específicas.

Assim, o objetivo deste trabalho prático foi, através de diversas abordagens, estudar o número mínimo de movimentos que esse automóvel precisa para chegar ao segmento final da estrada, sem ultrapassar os limites de velocidade de cada segmento e respeitando todas as suas regras de movimentação.

Specs:

Computadores e as suas especificações		
Nº PC/Nº Mec	Memória RAM	CPU
PC1 108215	16GB	AMD Ryzen™ 5 5500
PC2 108579	32GB	AMD Ryzen™ 9 5900X 12-Core Processor
PC3 108796	16GB	AMD Ryzen™ 7 4800H with Radeon Graphics

Contextualização do Problema

Inicialmente, dependendo do número mecanográfico passado como argumento ao programa, é gerada uma estrada. Por sua vez, esta estrada é divida em diversos segmentos com diferentes limites de velocidade definidos. A estrada terá, no máximo, n segmentos (definidos pelo programa inicial do professor).

O automóvel, para além de ter obrigatoriamente de respeitar os limites de velocidade de cada segmento (definidos também pelo programa inicial do professor), só pode reduzir a sua velocidade em **1** unidade de velocidade (*break*), manter a sua velocidade (*cruise*) ou aumentar a sua velocidade em **1 u.v.** (*accelerate*), excluindo a posição 0, em que o automóvel tem obrigatoriamente de aumentar a sua velocidade. Cada unidade de velocidade reduzida, mantida ou aumentada corresponde a uma posição recuada, mantida ou avançada da estrada, respetivamente. É também muito importante ter em conta que o automóvel tem velocidade 0 no primeiro segmento e tem de chegar ao último com velocidade 1, onde vai reduzir para 0 e, finalmente, parar.

Para aumentar a sua velocidade, o automóvel depende única e exclusivamente da posição anterior e da velocidade com que saiu da mesma.

Dando um exemplo mais prático:

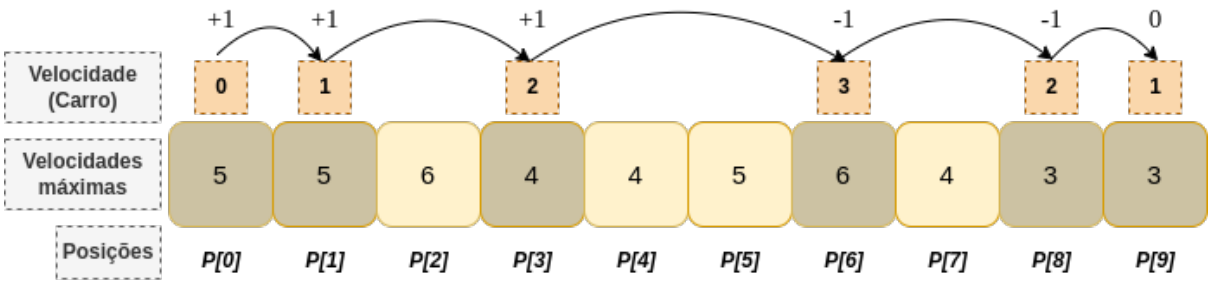

Figura 1.1: Exemplo de movimento de um carro

Tendo por base a ***figura ilustrada acima***, podemos observar que na **P[0]** o carro se encontrava com velocidade **0** mas, vendo-se na obrigação de avançar, aumentou a velocidade em uma unidade. Após aumentar a velocidade, o carro irá percorrer o número de casas igual à nova velocidade. Neste caso, passará para **P[1]**. De seguida, o carro vai verificar se consegue **aumentar/diminuir/manter** a velocidade. Neste caso, como a velocidade **máxima** das duas próximas casas é superior à velocidade do carro caso este aumentasse em **1** unidade a velocidade, o mesmo passará a ter velocidade **2** e percorrerá duas casas acabando em **P[3]**. E assim sucessivamente até ao final da estrada.

Capítulo 2

Métodos

Na procura de soluções eficientes e com baixo nível de complexidade computacional, foram necessários diversos algoritmos.

No nosso caso, acabámos por conseguir quatro algoritmos explorando diferentes técnicas de algoritmia:

1. **Brute Force (Professor)**
2. **Brute Force Melhorada**
3. **While**
4. **While-Dynamic**

Tanto a primeira como a segunda são funções recursivas abordando o problema através de **Ternary Trees**. Apesar de apresentarem métodos simples, acabam por ter uma complexidade computacional elevada, destacando, no entanto, um decréscimo acentuado entre a versão base e a versão melhorada.

Relativamente às soluções ***While***, destaca-se a simplicidade do algoritmo em pról da complexidade computacional baixa em ambos os casos, sendo estas definidas apenas usando loops *while* e *for* e validadas atráves de diversas condições *if*. Entre estas, a versão dinâmica destaca-se pela capacidade de armazenar valores ao longo dos movimentos garantindo uma chegada à solução mais rápida e eficaz.

Brute Force (Professor)

Brute Force descreve um estilo de programação primitivo em que o programador depende unicamente do poder de processamento do computador ao invés de usar métodos eficientes para simplificar o problema. É ignorada a escala do problema sendo aplicados métodos que seriam benéficos em pequenos problemas diretamente em problemas maiores prejudicando, assim, o desempenho dos mesmos. Por outras palavras é o equivalente a um código cansado ou repetido até à exaustão.

A solução disponibilizada pelo professor pode ser abordada e explicada através deste mesmo método. No método Brute Force todos os casos possiveis de movimentos são percorridos através de uma função recursiva.

Este método é dividido em duas grandes partes:

Primeiramente, são percorridas todas as possíveis variações de velocidade (desnecessário e lento), validando sempre se a velocidade é no mínimo **1**, nunca ultrapassa o limite de velocidade nem a posição final. Por cada variação de posição são também verificadas todas as posições no intervalo entre esse movimento, garantido o cumprimento dos limites de velocidade. Caso chegue à nova posição cumprindo todas as anteriores condições, voltará recursivamente a percorrer o mesmo processo. Outro grande problema deste método, substancialmente aprimorado na solução melhorada, é o facto de se focar em diminuir a velocidade sempre que possivel em primeiro lugar, em vez de manter ou até mesmo aumentar.

Por outro lado, cada vez que o método é chamado ocorrerá uma validação essencial que verificará se a posição onde se encontra é a final e se tem todas as condições para finalizar o movimento. Caso essa solução tenha um número de movimentos inferior à ultima solução definida como a melhor, será guardada como a "nova"melhor para futuras comparações.

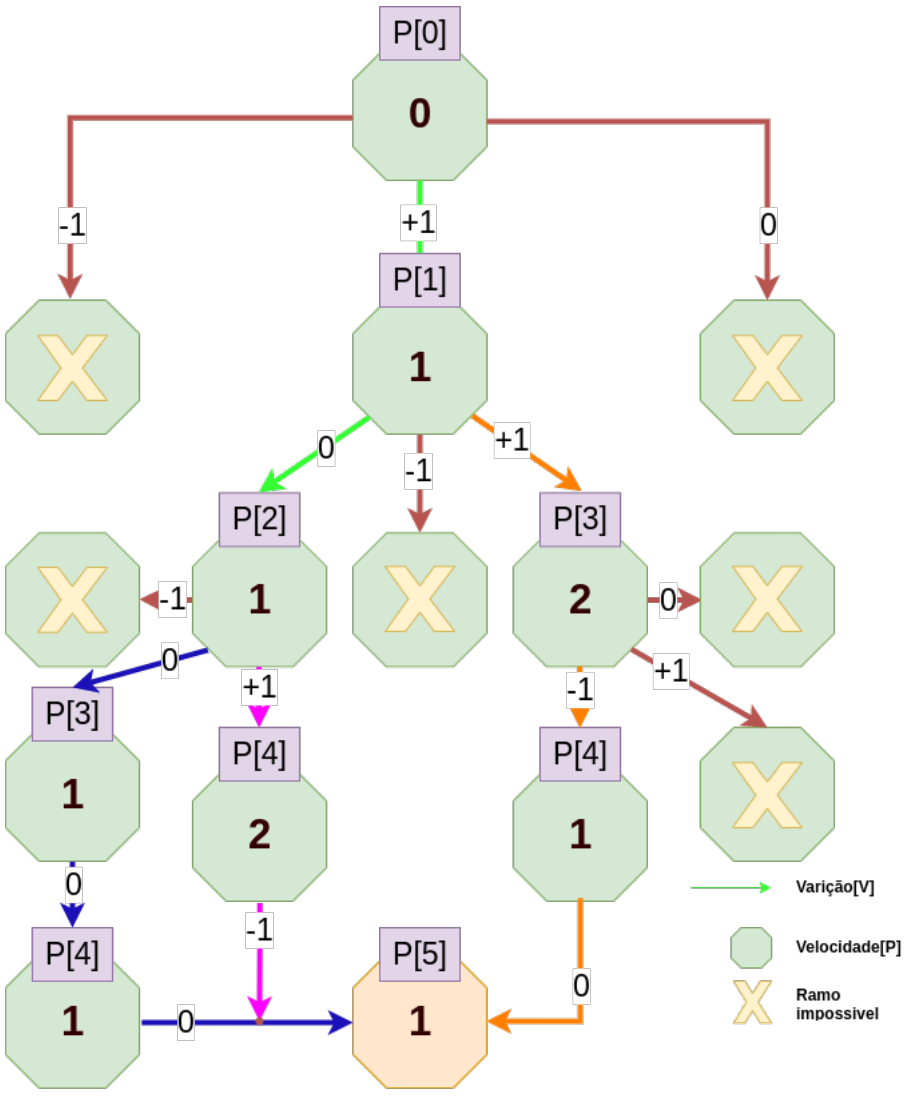

Figura 2.1: Primeiros 5 casos possíveis (**Solução Professor**)

Nota: Assume-se que a condição de limites de velocidade é sempre respeitada neste exemplo.

Aplicando ao exemplo **demonstrado em cima** e considerando que o carro começa em **P[0]** com **velocidade = 0**, terá **3** opções de variação de movimento. Como é verificado apenas uma delas será validada tendo em consideração que não pode ser utilizado o caminho com **V[-1]** nem **V[0]**.

```
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);
}
```

Agora em **P[1]**, apenas duas são as hipotéses válidas (*manter* ou *aumentar*), confirmando-se na mesma a condição acima. Fugindo para o lado esquerdo do gráfico teremos dois caminhos possíveis (Rosa e Azul). No Azul, iremos manter a **velocidade = 1** ao longo do percurso todo. O mesmo é possível, pois não existe nenhuma condição que o impede, o que também é mais uma prova de ineficiência do método (mais a frente veremos a solução melhorada em que com a aplicação de pruning). Iremos obter, portanto, um total de 5 movimentos (pior caso).

Mais à direita, seguindo o caminho Rosa haverá uma variação na velocidade, saindo de **P[2]** com **velocidade = 2**, sendo, por isso, obrigado a reduzir a velocidade em **P[4]** (qualquer outra hipotése quebraria as condições definidas no problema). O mesmo se verifica no caminho Laranja, apenas com ordem inversa e com as mesmas condições a impedirem algumas variações de movimento.

Como antes referido, a principal pejoraçáo deste problema é focar-se em diminuir a velocidade. isso implica que antes de até mesmo chegar à solução Azul (menos eficiente das que chegaram ao fim), o programa já percorreu um número considerável de caminhos pouco eficientes.

Em suma devido à natureza abrangente deste método, verifica-se uma complexidade computacional bastante elevada sendo que no pior caso é definida por $\mathcal{O}(\mathbf{3^n})$.

Brute Force Melhorada

Numa primeira tentativa de encontrar uma solução mais eficiente começámos por analisar o algoritmo dado pelo professor e perceber o que podíamos melhorar no mesmo.

Inicialmente, detetámos a problemática do ciclo *for* que percorria todas as possíveis variações de velocidade e que começava por verificar se era possível diminuir a velocidade, de seguida se podia manter e só depois se podia aumentar. Decidimos alterar o mesmo para verificar a sequência contrária, ou seja, se é possível *aumentar* a velocidade, caso não o seja, se é possível *manter* ou, caso contrário, se é possível *diminuir*, ou seja, **dando prioridade ao aumento da velocidade**.

No entanto, vimos ser necessária uma condição essencial que utiliza como base o método de **Decision Tree Pruning**. Dada, neste caso, uma **Ternary Tree**, este método é uma técnica que reduz o seu tamanho **removendo** os ramos que não são precisos e não contribuem para a resolução do problema por não serem os mais eficientes.

```
if (solution_3_best.positions[move_number] > position)
{
    return;
}
```

No nosso caso, colocámos este método em prática através de uma pequena condição *if*, que **podemos observar acima**, que para uma mesma posição verifica se a solução atual é mais ou menos eficiente que a solução anteriormente guardada como melhor solução. Caso o caminho da solução atual não seja o melhor caminho a fazer, este ramo será **removido** da árvore, caso contrário, esta solução passará a ser guardada como **melhor solução**, e assim sucessivamente para todas os caminhos possíveis.

Em conclusão, nesta solução o programa acaba por verificar todas as variações de velocidade possíveis, o que é igualmente ineficiente em comparação à solução dada pelo professor, mas, para além de ser melhor por se focar no aumento da velocidade, este programa não vai percorrer até ao fim todos os caminhos possíveis, apenas os mais rápidos, ainda que respeitadores de todas as regras, por consequência do método de **Tree Pruning**.

Complexidade Computacional: $\mathcal{O}(n)$ - Remove os ramos ineficientes

While

O método **While**, em alternativa à recursividade, vai conseguir na mesma e eficientemente percorrer o trajeto num espaço de tempo reduzido e com baixa complexidade ($\mathcal{O}(\mathbf{n})$). O que o destaca é a antecipação com que opera, garantindo que estará sempre precavido de precipitadas descidas de velocidade.

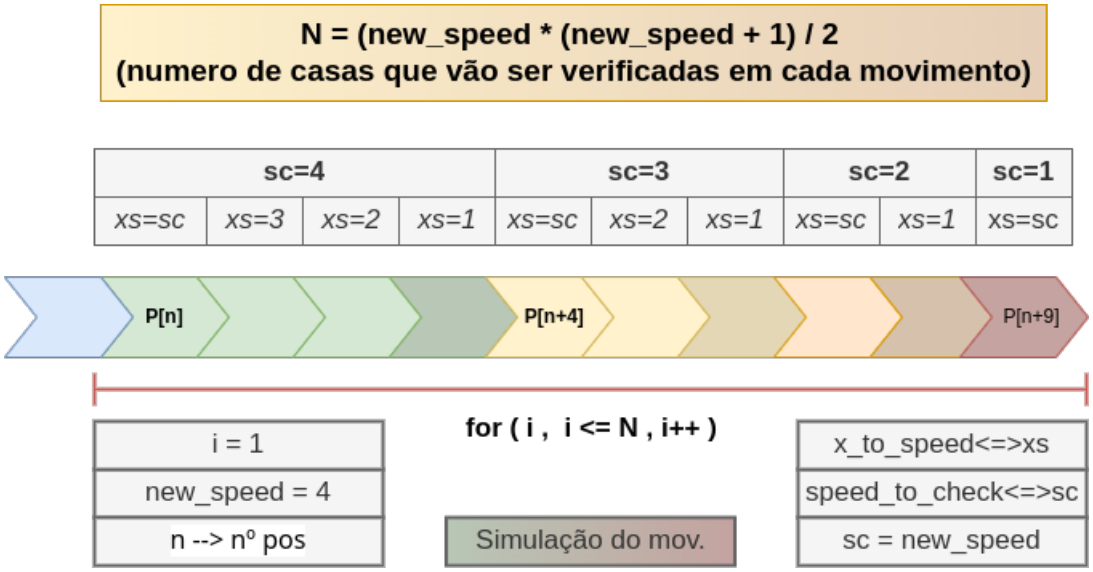

Figura 2.2: Exemplo teórico da função **While**

Na base deste método está um ciclo que percorre as diferentes variações de velocidade e a equação definida na figura acima como **N**, tendo sido a mesma descoberta através de testagem de distâncias de travagem com diferentes velocidades iniciais (**até velocidade = 7**). Ao analisar o padrão concluimos que se enquadrava numa série divergente sendo que os resultados cumpriam o valor de **números triangulares**. A partir desse ponto, bastou garantir que passaria por todas as posições com a velocidade adequada de travagem. Se em cada iteração do carro conseguirmos garantir que esta condição se cumpre estaremos sempre aptos para conseguir uma solução eficaz. Claro que, tal como em ambos os métodos anteriores, existem condições que são essenciais para o cumprimento da trama. Ainda dentro deste ciclo temos que garantir que tanto o movimento futuro do carro, como a simulação dos próximos movimentos cumprem o limite de velocidade em todas as posições desse intervalo e nunca ultrapassam a posição final. Caso alguma das mesmas se verifique o ciclo é quebrado e a solução prossegue para outra variação de velocidade.

Dando um exemplo mais prático e detalhado: Imaginemos que na figura acima, o carro estaria na posição azul com **velocidade=4** e após percorrer todas as variações de velocidade possíveis conclui que manter a velocidade seria a solução mais adequada (excluindo à priori a possibilidade de aumentar).
Irá fazer o caminho, seguindo a fórmula similar, 4 posições à frente da atual.

Complexidade Computacional: $\mathcal{O}(n)$ - Nos piores casos, percorre o array só 1x

While-Dynamic

Para este método, partimos da mesma lógica que a função **While**, porém aplicámos **Dynamic Programing** (estilo **Tabulation**), ou seja, através de dynamic approach, a nossa função é capaz, no momento em que a ($\mathbf{position} + \mathbf{new\_speed} * \frac{\mathbf{new\_speed}+1}{2}$) > **final_position**, são guardadas as respetivas posições associadas a cada **move**, o número de **moves**, a respetiva **position** e o **speed**.

```
for (int j = 1; j <= (new_speed * (new_speed + 1) / 2); j++) {
    if (position + j > final_position) {
        possiblemove = 0;
        save = 1;
        break;
    }

    if (x_to_speed == 0) {
        speed_to_check -= 1;
        x_to_speed = speed_to_check;
    }
    if (max_road_speed[position + j] < speed_to_check) {
        possiblemove = 0;
        break;
    }
    x_to_speed -= 1;
}
```

Foi adotado este método neste respetivo momento, pois é onde o carro começa a **travar** até à **final_position**.

Seguindo para o próximo percurso, no início, os argumentos de entrada da função serão o número de **moves**, o **speed** e a **position**, que foram guardados anteriormente, e a **final_position**.

```
if (save && !saved) {
    solution_2_before = solution_2;
    solution_2_before.n_moves = move_number;
    solution_2_beforespeed = speed;
    saved = 1;
}
if (possiblemove) {
    position += new_speed;
    speed = new_speed;
    move_number++;
    break;
}
```

Complexidade Computacional: $\mathcal{O}(n)$ - Necessita **n** estados para resolver o atual

Capítulo 3

Resultados Obtidos

Brute Force

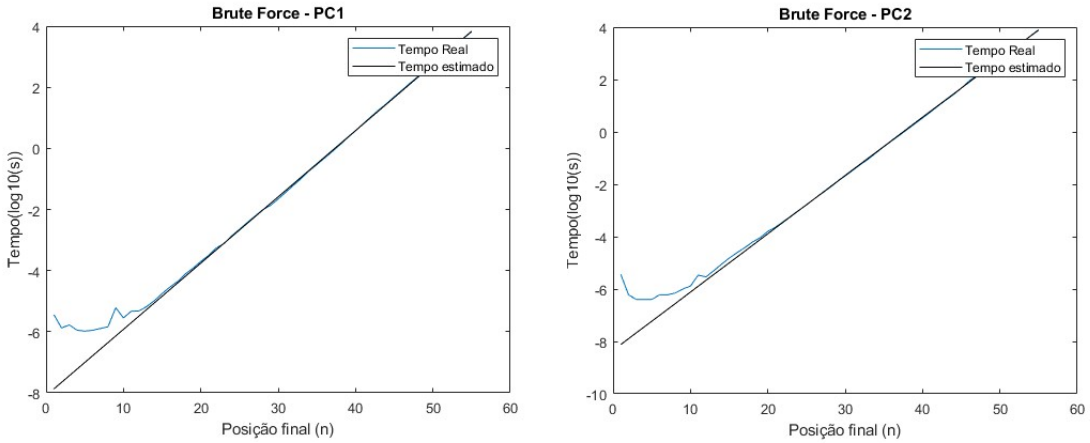

Figura 3.1: Tempo de execução de 51 posições no PC1 e PC2 (respetivamente)

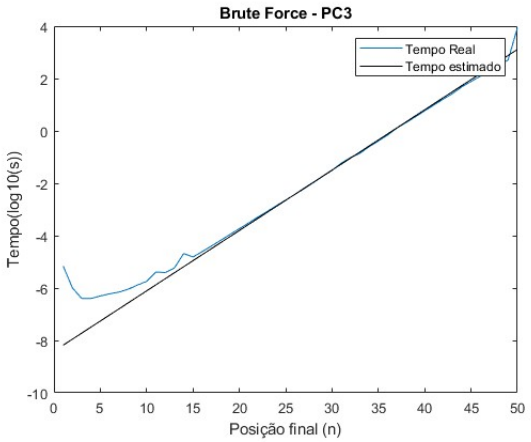

Figura 3.2: Tempo de execução de 50 posições no PC3

Como podemos observar nos gráficos acima representados, onde aplicámos uma regressão linear para facilitar a visualização, a solução Brute Force dada pelo professor corre apenas até, no máximo, às **55 posições** nos 60 minutos definidos no código. Assim, para termos uma melhor noção do tempo de execução do programa até às 800 posições, tivemos de realizar alguns cálculos, através do MatLab, que podemos ver de seguida (exemplo do PC2):

```
A = load("A_108579.txt");
n = A(:,1); %primeira coluna
t = A(:,4); %quarta coluna

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

t800_log = [800 1] * Coefs;
t800 = 10^t800_log;
fprintf("Tempo de execucao estimado para as 800 posicoes:%.3d segundos", t800);
```

Realizando estes cálculos para todos os PCs concluímos que o **tempo de execução estimado para as 800 posições** é $\mathbf{2.304e^{165}s}$ para o PC1, $\mathbf{4.893e^{169}s}$ para o PC2 e $\mathbf{8.791e^{175}s}$ para o PC3.

De seguida, podemos observar os gráficos do tempo de execução de cada uma das outras soluções em todos os PCs:

Brute Force Melhorada

Todos os PCs

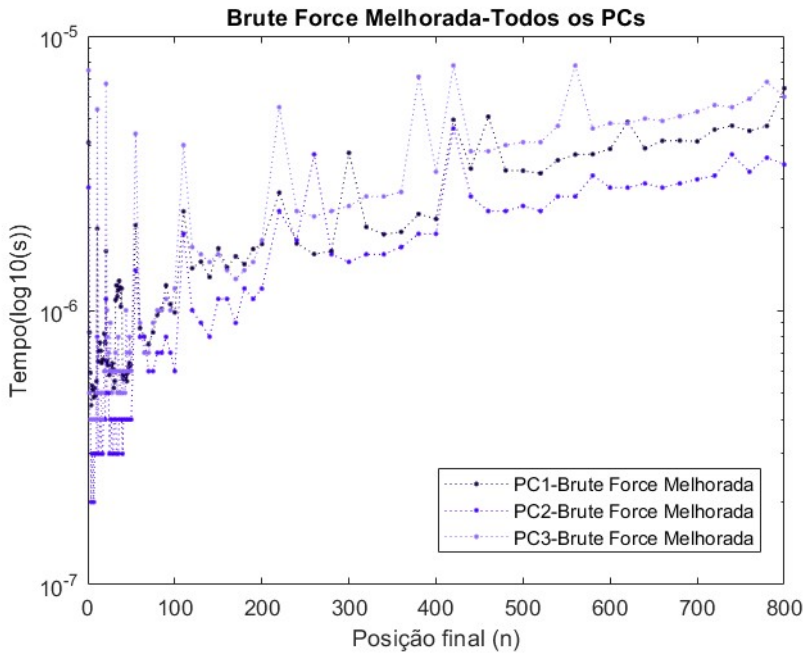

Figura 3.3: Comparação da solução Brute Force Melhorada em todos os PCs

While

Todos os PCs

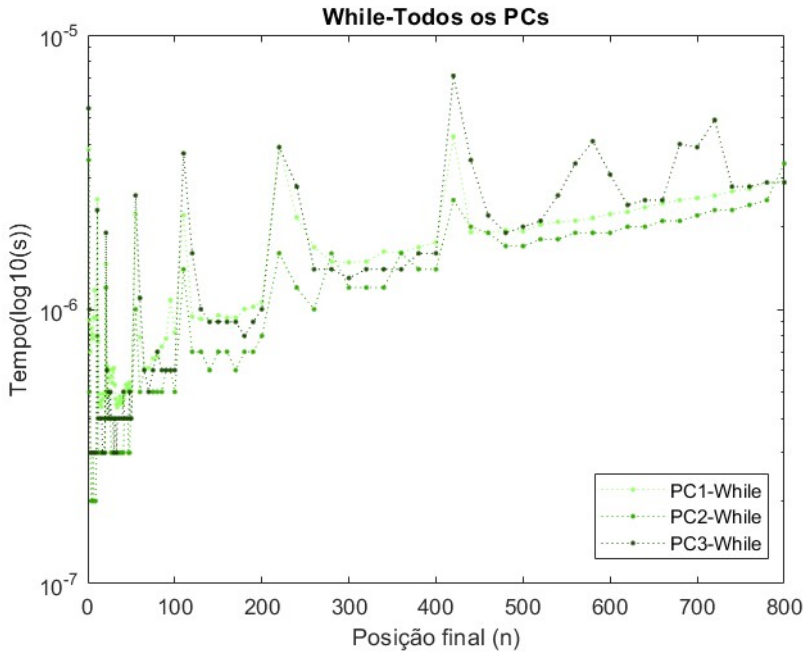

Figura 3.4: Comparação da solução While em todos os PCs

While Dynamic

Todos os PCs

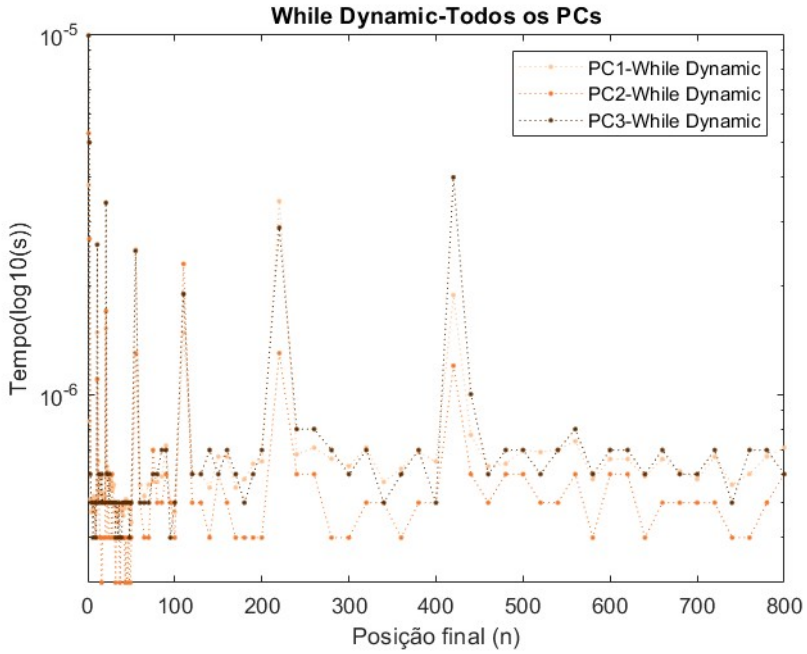

Figura 3.5: Comparação da solução While Dynamic em todos os PCs

Todas as soluções

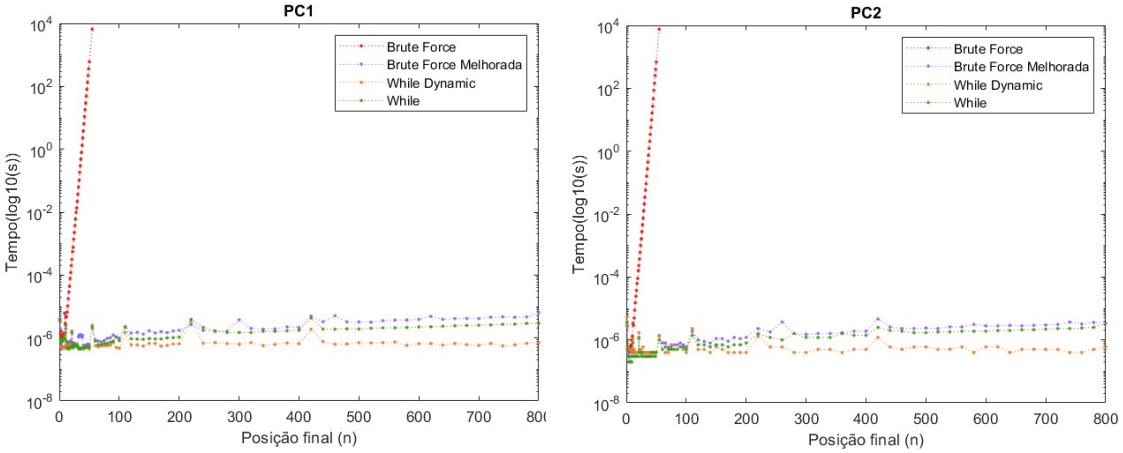

Figura 3.6: Comparação de todas as soluções no PC1 e PC2 (respetivamente)

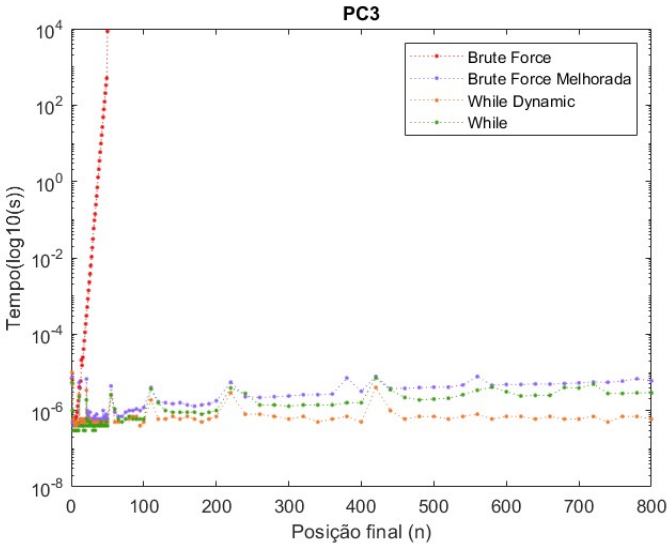

Figura 3.7: Comparação de todas as soluções no PC3

Todas as soluções

Todos os PCs

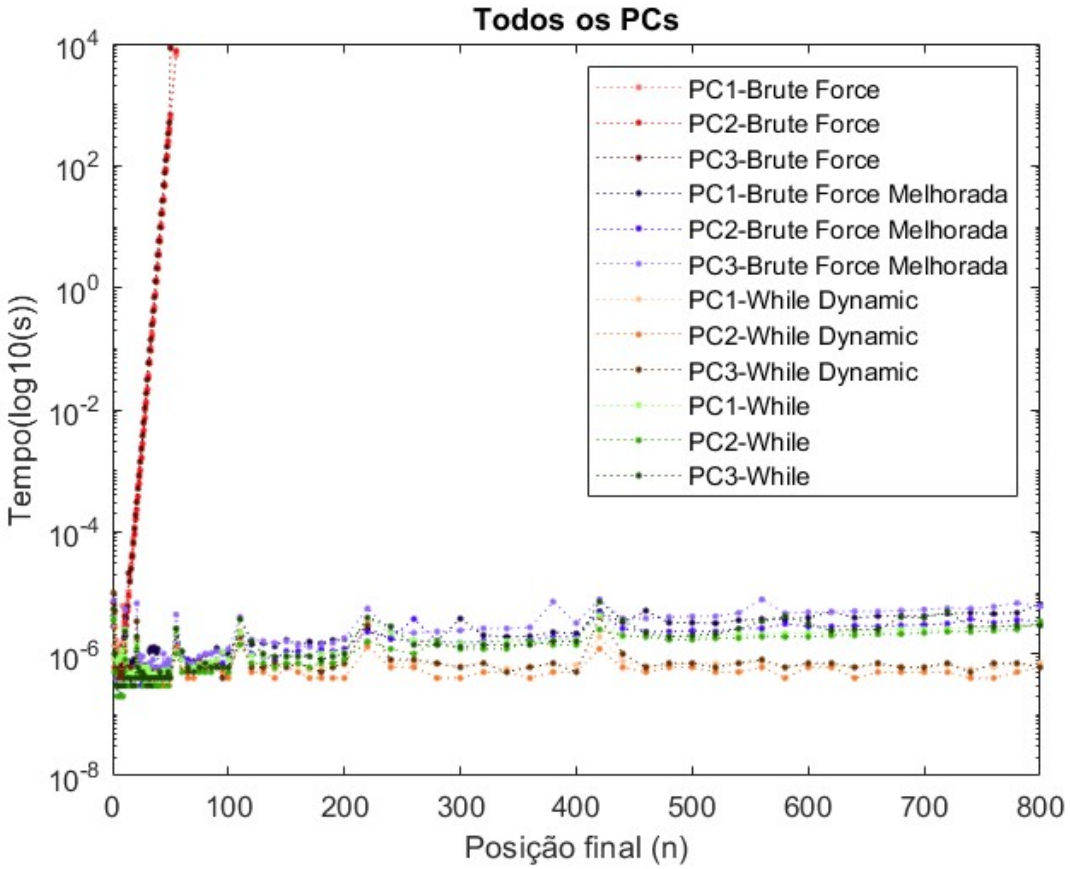

Figura 3.8: Comparação de todas as soluções nos três PCs

Como podemos observar nos gráficos acima representados, as soluções **Brute Force Melhorada**, **While** e **While Dynamic** são significativamente mais eficientes que a solução dada pelo professor, sendo esta apenas possível de correr, no máximo, até à posição **55 nos 60 minutos** estabelecidos no código dado.

Quanto às restantes, é de notar a crescente eficiência das soluções, na ordem **Brute Force Melhorada, While, While Dynamic,** analisando o decrescente tempo que as mesma demoram a chegar à mesma posição final.

Capítulo 4

Conclusão

Em suma, este trabalho para além de aprimorar as nossas capacidades de escrita de linguagem **C**, demonstrou a importância da procura de várias soluções para perceber as vantagens e desvantagens de cada uma delas.

Mais especificamente, com este trabalho conseguimos perceber o quão diferentes complexidades computacionais se refletem em diferentes tempos de execução e como é importante ter isso em conta aquando da procura de uma solução.

Para além disto, e para finalizar, aprendemos imenso sobre a necessidade de uma boa gestão de memória com a finalidade de conseguirmos uma solução mais eficiente.

Contribuição (%)

Anderson Lourenço : 33.(3)%

Sara Almeida : 33.(3)%

Hugo Correia : 33.(3)%

Capítulo 5

Webography

1. https://en.wikipedia.org/wiki/Triangular_number
2. https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
3. https://en.wikipedia.org/wiki/Decision_tree_pruning
4. https://www.geeksforgeeks.org
5. https://www.diagrams.net/
6. https://code.visualstudio.com/
7. https://www.overleaf.com/
8. https://elearning.ua.pt - Slides da Disciplina

Capítulo 6

Código C/Anexo 1

```c
    //
// AED, August 2022 (Tomas 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. 108579  Name: Anderson Lourenco --> PC2
//   N.Mec. 108796  Name: Sara Almeida --> PC3
//   N.Mec. 108215  Name: Hugo Correia --> PC1
//


//
// static configuration
//
#define SOLUTION_SPEED_RUN_A
//#define SOLUTION_SPEED_RUN_B
//#define SOLUTION_SPEED_RUN_C
//#define SOLUTION_SPEED_RUN_D

#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
//

```

```
44 static int max_road_speed[1 + _max_road_size_]; // positions 0.._max_road_size_
45
46 static void init_road_speeds(void)
47 {
48   double speed;
49   int i;
50
51   for(i = 0;i <= _max_road_size_;i++)
52   {
53     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));
54     max_road_speed[i] = (int)floor(0.5 + speed) + (int)((unsigned int)random()
       % 3u) - 1;
55     if(max_road_speed[i] < _min_road_speed_)
56       max_road_speed[i] = _min_road_speed_;
57     if(max_road_speed[i] > _max_road_speed_)
58       max_road_speed[i] = _max_road_speed_;
59   }
60 }
61
62
63 //
64 // description of a solution
65 //
66
67 typedef struct
68 {
69   int n_moves;                          // the number of moves (the number of
       positions is one more than the number of moves)
70   int positions[1 + _max_road_size_];  // the positions (the first one must be
       zero)
71   int spd[1 + _max_road_size_];
72 }
73 solution_t;
```

Brute Force (Professor)

```
1  static solution_t solution_1,solution_1_best;
2  static double solution_1_elapsed_time; // time it took to solve the problem
3  static unsigned long solution_1_count; // effort dispended solving the problem
4
5  static void solution_1_recursion(int move_number,int position,int speed,int
       final_position)
6  {
7    int i,new_speed;
8
9    // record move
10   solution_1_count++;
11   solution_1.positions[move_number] = position;
12   // is it a solution?
13   if(position == final_position && speed == 1)
14   {
15     // is it a better solution?
16     if(move_number < solution_1_best.n_moves)
17     {
18       solution_1_best = solution_1;
19       solution_1_best.n_moves = move_number;
20     }
21     return;
22   }
23   // no, try all legal speeds
24     if(solution_1_best.positions[move_number] > solution_1.positions[
       move_number]){
25     return;
26   }
27
28   for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--)
29     if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed
       <= final_position)
30     {
31       for(i = 0;i <= new_speed && new_speed <= max_road_speed[position + i];i
       ++)
32         ;
33       if(i > new_speed)
34         solution_1_recursion(move_number + 1,position + new_speed,new_speed,
       final_position);
35     }
36 }
37
38 static void solve_1(int final_position)
39 {
40   if(final_position < 1 || final_position > _max_road_size_)
41   {
42     fprintf(stderr,"solve_1: bad final_position\n");
43     exit(1);
44   }
45   solution_1_elapsed_time = cpu_time();
46   solution_1_count = 0ul;
47   solution_1_best.n_moves = final_position + 100;
48   solution_1_recursion(0,0,0,final_position);
49   solution_1_elapsed_time = cpu_time() - solution_1_elapsed_time;
50 }
```

Brute Force Melhorada

```
1  static solution_t solution_3,solution_3_best;
2  static double solution_3_elapsed_time; // time it took to solve the problem
3  static unsigned long solution_3_count; // effort dispended solving the problem
4
5  static void solution_3_recursion(int move_number,int position,int speed,int
       final_position)
6  {
7    int i,new_speed;
8
9    // record move
10   solution_3_count++;
11   solution_3.positions[move_number] = position;
12   // is it a solution?
13   if(position == final_position && speed == 1)
14   {
15     // is it a better solution?
16     if(move_number < solution_3_best.n_moves)
17     {
18       solution_3_best = solution_3;
19       solution_3_best.n_moves = move_number;
20     }
21     return;
22   }
23   // no, try all legal speeds
24   if (solution_3_best.positions[move_number] > position)
25   {
26     return;
27   }
28     for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--)
29     if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed
       <= final_position)
30     {
31       for(i = 0;i <= new_speed && new_speed <= max_road_speed[position + i];i
       ++)
32         ;
33       if(i > new_speed)
34         solution_3_recursion(move_number + 1,position + new_speed,new_speed,
       final_position);
35     }
36 }
37
38 static void solve_3(int final_position)
39 {
40   if(final_position < 1 || final_position > _max_road_size_)
41   {
42     fprintf(stderr,"solve_3: bad final_position\n");
43     exit(1);
44   }
45   solution_3_elapsed_time = cpu_time();
46   solution_3_count = 0ul;
47   solution_3_best.n_moves = final_position + 100;
48   solution_3_recursion(0,0,0,final_position);
49   solution_3_elapsed_time = cpu_time() - solution_3_elapsed_time;
50 }
```

While

```
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)
  int spd[1 + _max_road_size_];
}
solution_t;

static solution_t solution_4,solution_4_best;
static double solution_4_elapsed_time; // time it took to solve the problem
static unsigned long solution_4_count; // effort dispended solving the problem


static void solution_4_while(int move_number,int position,int speed,int final_position)
{
  int diff_speeds[3] = {1,0,-1};
  int possiblemove = 1;

  while (position != final_position) {
    solution_4.positions[move_number] = position;
    for (int i = 0; i < 3; i++) {
      solution_4_count++;
      possiblemove = 1;
      int new_speed = speed + diff_speeds[i];
      int next_position = position + new_speed;

      if (new_speed < 1 || new_speed > _max_road_speed_ || max_road_speed[position] < new_speed) continue;

      int speed_to_check = new_speed;
      int x_to_speed = speed_to_check;
      for (int j = 1; j <= (new_speed * (new_speed + 1) / 2); j++) {
        if (position + j > final_position) {
          possiblemove = 0;
          break;
        }

        if (x_to_speed == 0) {
          speed_to_check -= 1;
          x_to_speed = speed_to_check;
        }

        if (max_road_speed[position + j] < speed_to_check) {

          possiblemove = 0;

          break;
        }
        x_to_speed -= 1;
      }

      if (possiblemove) {
        position += new_speed;
        speed = new_speed;
        move_number++;
        break;
      }
    }
```

```
58
59    }
60    solution_4.positions[move_number] = position;
61    solution_4_best = solution_4;
62    solution_4_best.n_moves = move_number;
63    return;
64  }
65
66  static void solve_4(int final_position)
67  {
68    if(final_position < 1 || final_position > _max_road_size_)
69    {
70      fprintf(stderr,"solve_4: bad final_position\n");
71      exit(1);
72    }
73    solution_4_elapsed_time = cpu_time();
74    solution_4_count = 0ul;
75    solution_4_best.n_moves = final_position + 100;
76    solution_4_while(0,0,0,final_position);
77    //int move_number,int position,int speed,int final_position
78    solution_4_elapsed_time = cpu_time() - solution_4_elapsed_time;
79  }
```

While-Dynamic

```
static solution_t solution_2,solution_2_best, solution_2_before;
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 int solution_2_beforespeed;
static void solution_2_while(int move_number,int position,int speed,int
    final_position)
{
  printf("Starting move %d at position %d with speed %d\n", move_number,
    position, speed);
  int diff_speeds[3] = {1,0,-1};
  // print move number, speed and position to the screen
  int ending = 0;
  int save=0, saved = 0;
  int possiblemove = 1;

  while (position != final_position) {
  solution_2_count++;
  solution_2.positions[move_number] = position;
    for (int i = 0; i < 3; i++) {
      possiblemove = 1;
      int new_speed = speed + diff_speeds[i];
      int next_position = position + new_speed;

      if (new_speed < 1 || new_speed > _max_road_speed_ || max_road_speed[
    position] < new_speed) continue;
      int speed_to_check = new_speed;
      int x_to_speed = speed_to_check;
      for (int j = 1; j <= (new_speed * (new_speed + 1) / 2); j++) {
        if (position + j > final_position) {
          possiblemove = 0;
          save = 1;
          break;
        }

        if (x_to_speed == 0) {
          speed_to_check -= 1;
          x_to_speed = speed_to_check;
        }
        if (max_road_speed[position + j] < speed_to_check) {
          possiblemove = 0;
          break;
        }
        x_to_speed -= 1;
      }
      if (save && !saved) {
        solution_2_before = solution_2;
        solution_2_before.n_moves = move_number;
        solution_2_beforespeed = speed;
        saved = 1;
      }
      if (possiblemove) {
        position += new_speed;
        speed = new_speed;
        move_number++;
        break;
      }
    }

  }
  solution_2.positions[move_number] = position;
  solution_2_best = solution_2;
```

```
59   solution_2_best.n_moves = move_number;
60   return;
61 }
62
63 static void solve_2(int final_position)
64 {
65   if(final_position < 1 || final_position > _max_road_size_)
66   {
67     fprintf(stderr,"solve_2: bad final_position\n");
68     exit(1);
69   }
70   solution_2_elapsed_time = cpu_time();
71   solution_2_count = 0ul;
72   solution_2_best.n_moves = final_position + 100;
73   solution_2 = solution_2_before;
74   printf("move number = %d, starting position = %d, starting speed = %d\n",
       solution_2_before.n_moves, solution_2_before.positions[solution_2_before.
       n_moves], solution_2_beforespeed);
75   solution_2_while(solution_2_before.n_moves,solution_2_before.positions[
       solution_2_before.n_moves],solution_2_beforespeed,final_position);
76   solution_2_elapsed_time = cpu_time() - solution_2_elapsed_time;
77 }
```

```
1  //example of the slides
2
3  static void example(void)
4  {
5    int i,final_position;
6
7    srandom(0xAED2022);
8    init_road_speeds();
9    final_position = 30;
10   solve_1(final_position);
11   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");
12   printf("mad road speeds:");
13   for(i = 0;i <= final_position;i++)
14     printf(" %d",max_road_speed[i]);
15   printf("\n");
16   printf("positions:");
17   for(i = 0;i <= solution_1_best.n_moves;i++)
18     printf(" %d",solution_1_best.positions[i]);
19   printf("\n");
20 }
21
22 static void example2(void)
23 {
24   int i,final_position;
25
26   srandom(0xAED2022);
27   init_road_speeds();
28   final_position = 30;
29   solve_2(final_position);
30   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,"While loop");
31   printf("mad road speeds:");
32   for(i = 0;i <= final_position;i++)
33     printf(" %d",max_road_speed[i]);
34   printf("\n");
35   printf("positions:");
36   for(i = 0;i <= solution_2_best.n_moves;i++)
37     printf(" %d",solution_2_best.positions[i]);
38   printf("\n");
39 }
40
41 static void example3(void)
42 {
43   int i,final_position;
44
45   srandom(0xAED2022);
46   init_road_speeds();
47   final_position = 30;
48   solve_3(final_position);
49   make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],
       solution_3_best.n_moves,&solution_3_best.positions[0],
       solution_3_elapsed_time,solution_3_count,"Plain recursion but better");
50   printf("mad road speeds:");
51   for(i = 0;i <= final_position;i++)
52     printf(" %d",max_road_speed[i]);
53   printf("\n");
54   printf("positions:");
55   for(i = 0;i <= solution_3_best.n_moves;i++)
56     printf(" %d",solution_3_best.positions[i]);
57   printf("\n");
```

```
58 }
59
60 static void example4(void)
61 {
62   int i,final_position;
63
64   srandom(0xAED2022);
65   init_road_speeds();
66   final_position = 30;
67   solve_4(final_position);
68   make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],
       solution_4_best.n_moves,&solution_4_best.positions[0],
       solution_4_elapsed_time,solution_4_count,"While loop but better");
69   printf("mad road speeds:");
70   for(i = 0;i <= final_position;i++)
71     printf(" %d",max_road_speed[i]);
72   printf("\n");
73   printf("positions:");
74   for(i = 0;i <= solution_4_best.n_moves;i++)
75     printf(" %d",solution_4_best.positions[i]);
76   printf("\n");
77 }
78
79
80 //
81 // main program
82 //
83
84 int main(int argc,char *argv[argc + 1])
85 {
86 # define _time_limit_  3600.0
87   int n_mec,final_position,print_this_one;
88   char file_name[64];
89
90   // generate the example data
91   if(argc == 2 && argv[1][0] == '-' && argv[1][1] == 'e' && argv[1][2] == 'x')
92   {
93     example5();
94     return 0;
95   }
96   // initialization
97   n_mec = (argc < 2) ? 0xAED2022 : atoi(argv[1]);
98   srandom((unsigned int)n_mec);
99   init_road_speeds();
100  // run all solution methods for all interesting sizes of the problem
101  final_position = 1;
102  solution_2_elapsed_time = 0.0;
103  printf("    + --- ---------------- --------- +\n");
104  printf("    |                plain recursion |\n");
105  printf("--- + --- ---------------- --------- +\n");
106  printf("  n | sol            count  cpu time |\n");
107  printf("--- + --- ---------------- --------- +\n");
108  while(final_position <= _max_road_size_/* && final_position <= 20*/)
109  {
110    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;
111    printf("%3d |",final_position);
112    // first solution method (very bad)
113    #ifdef SOLUTION_SPEED_RUN_A
114    if(solution_1_elapsed_time < _time_limit_)
115    {
116      solve_1(final_position);
```

```
117       if(print_this_one != 0)
118       {
119         sprintf(file_name,"%03d_1_%s.pdf",final_position,argv[1]);
120         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");
121       }
122       printf(" %3d %16lu %9.3e |",solution_1_best.n_moves,solution_1_count,
      solution_1_elapsed_time);
123     }
124     else
125     {
126       solution_1_best.n_moves = -1;
127       printf("                                  |");
128     }
129     #endif
130     // second solution method (less bad)
131     // ...
132     #ifdef SOLUTION_SPEED_RUN_B
133     if(solution_2_elapsed_time < _time_limit_)
134     {
135       solve_2(final_position);
136       if(print_this_one != 0)
137       {
138         sprintf(file_name,"%03d_2.pdf",final_position);
139         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");
140       }
141       printf(" %3d %16lu %9.3e |",solution_2_best.n_moves,solution_2_count,
      solution_2_elapsed_time);
142     }
143     else
144     {
145       solution_2_best.n_moves = -1;
146       printf("                                  |");
147     }
148     #endif
149     // third solution method (less bad)
150     // ...
151     #ifdef SOLUTION_SPEED_RUN_C
152     if(solution_3_elapsed_time < _time_limit_)
153     {
154       solve_3(final_position);
155       if(print_this_one != 0)
156       {
157         sprintf(file_name,"%03d_3.pdf",final_position);
158         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,"Plain recursion");
159       }
160       printf(" %3d %16lu %9.3e |",solution_3_best.n_moves,solution_3_count,
      solution_3_elapsed_time);
161     }
162     else
163     {
164       solution_3_best.n_moves = -1;
165       printf("                                  |");
166     }
167     #endif
168     // fourth solution method (less bad)
169     // ...
170     #ifdef SOLUTION_SPEED_RUN_D
```

```
171     if(solution_4_elapsed_time < _time_limit_)
172     {
173       solve_4(final_position);
174       if(print_this_one != 0)
175       {
176         sprintf(file_name,"%03d_4.pdf",final_position);
177         make_custom_pdf_file(file_name,final_position,&max_road_speed[0],
      solution_4_best.n_moves,&solution_4_best.positions[0],
      solution_4_elapsed_time,solution_4_count,"Plain recursion");
178       }
179       printf(" %3d %16lu %9.3e |",solution_4_best.n_moves,solution_4_count,
      solution_4_elapsed_time);
180     }
181     else
182     {
183       solution_4_best.n_moves = -1;
184       printf("                                    |");
185     }
186     #endif
187 
188     //done
189     printf("\n");
190     fflush(stdout);
191     // new final_position
192     if(final_position < 50)
193       final_position += 1;
194     else if(final_position < 100)
195       final_position += 5;
196     else if(final_position < 200)
197       final_position += 10;
198     else
199       final_position += 20;
200   }
201   printf("--- + --- ---------------- --------- +\n");
202   return 0;
203 # undef _time_limit_
204 }
```

Capítulo 7

PDFs/Anexo 2

PC1

Brute Force

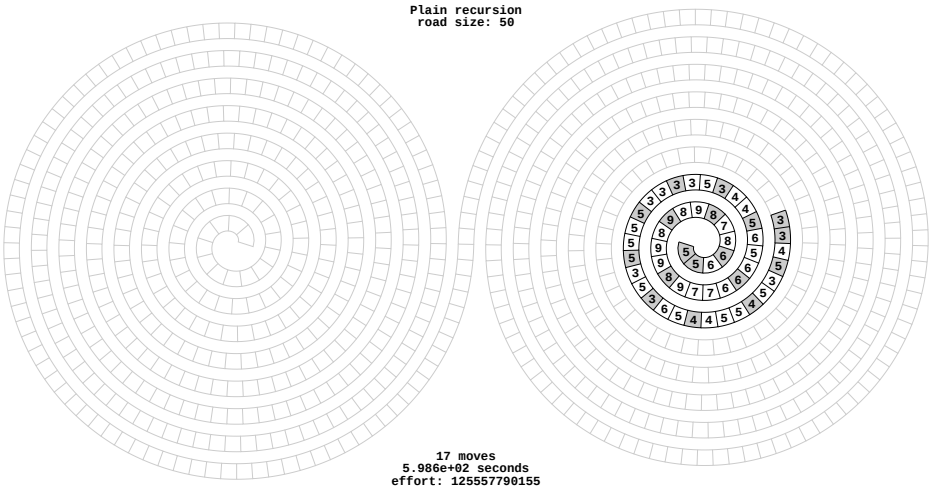

Figura 7.1: Solução Brute Force (Professor)

Brute Force Melhorada

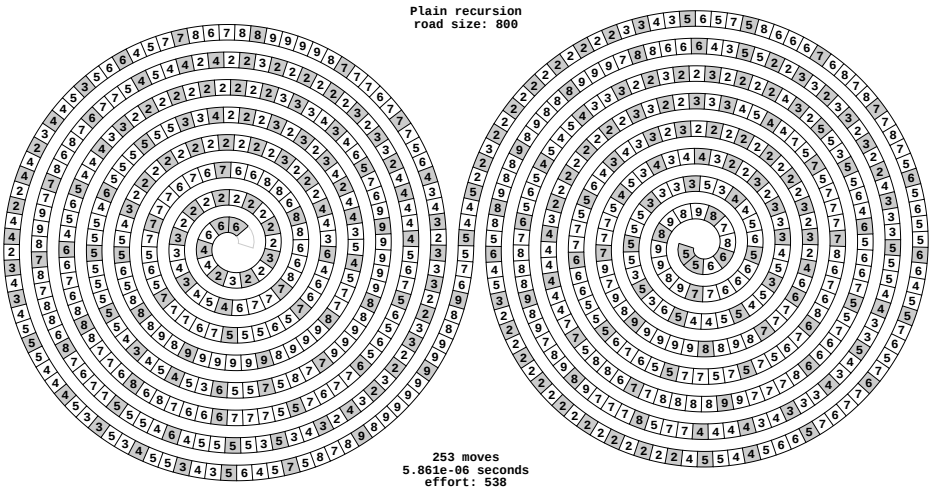

Figura 7.2: Solução Brute Force Melhorada

While

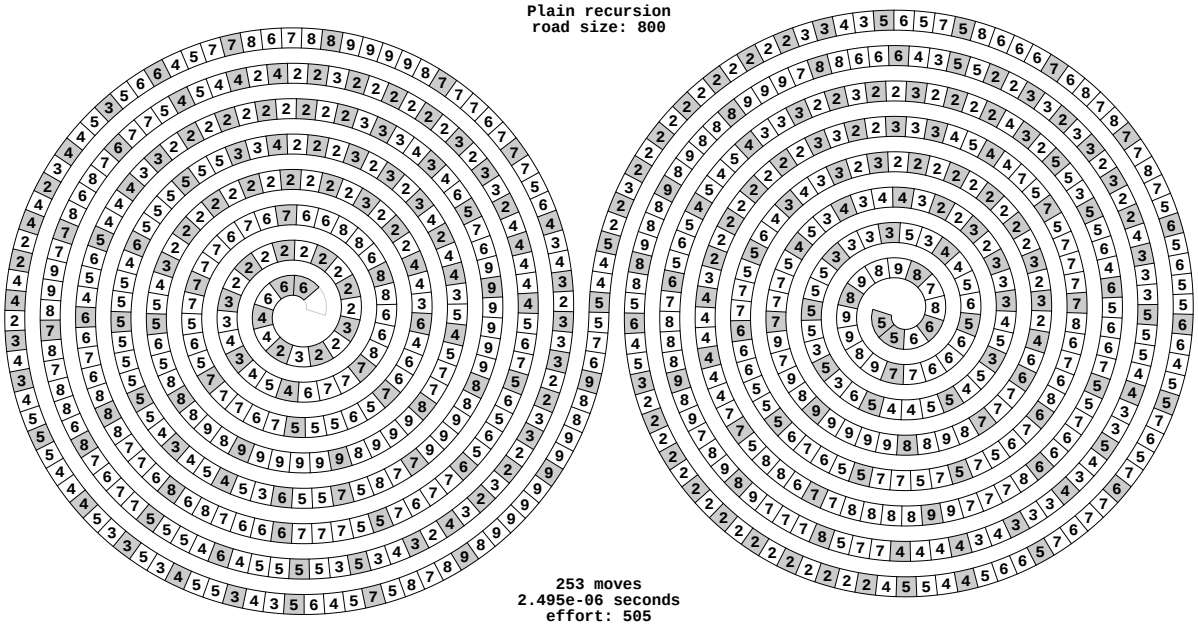

Figura 7.3: Solução While

While Dynamic

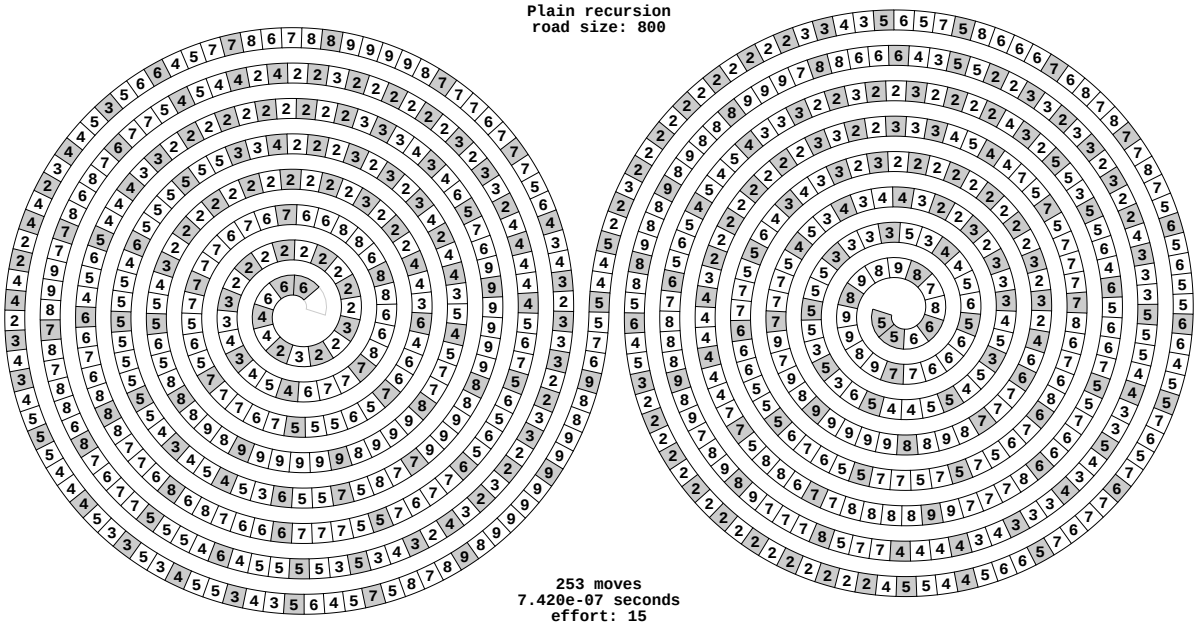

Figura 7.4: Solução While Dynamic

PC2

Brute Force

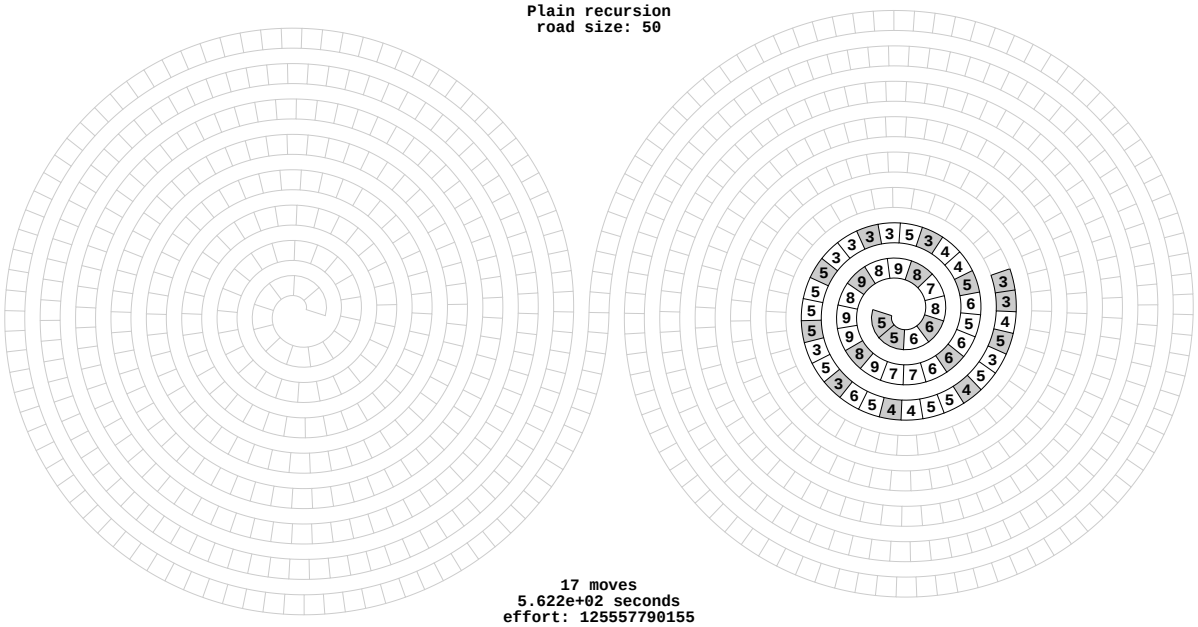

Figura 7.5: Solução Brute Force (Professor)

Brute Force Melhorada

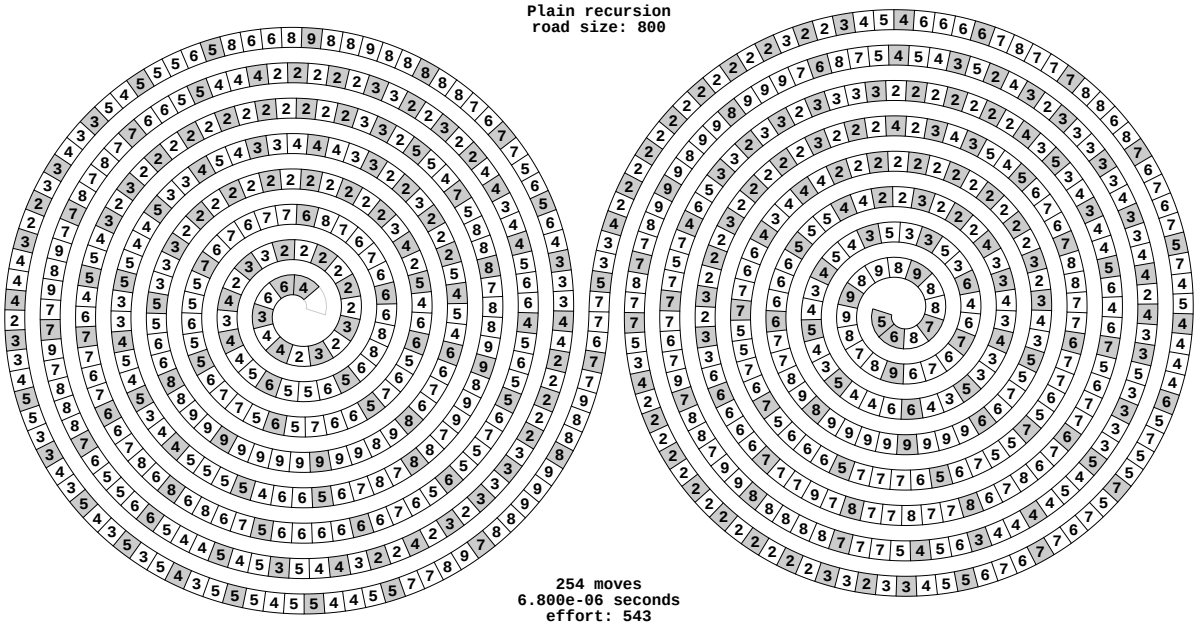

Figura 7.6: Solução Brute Force Melhorada

While

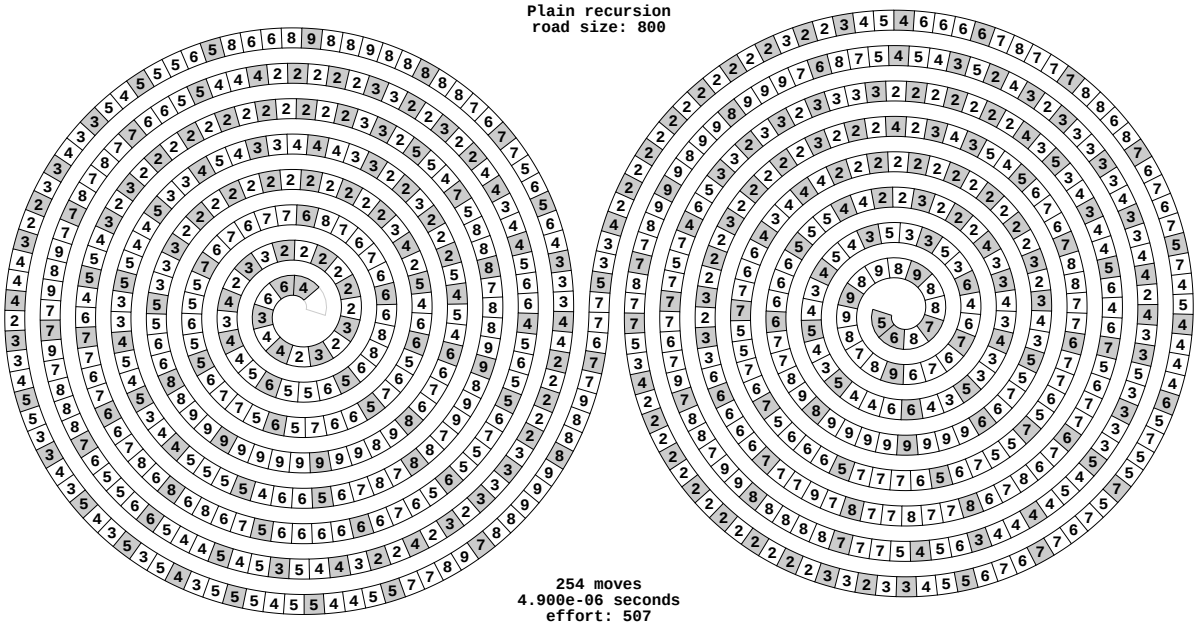

Figura 7.7: Solução While

While Dynamic

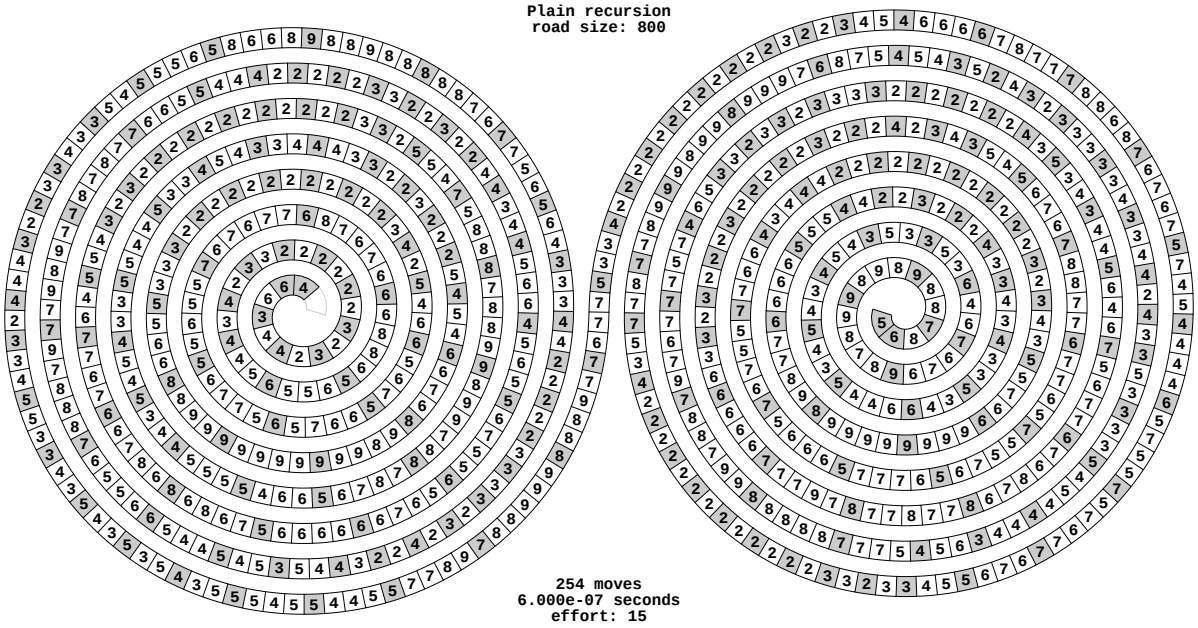

Figura 7.8: Solução While Dynamic

PC3

Brute Force

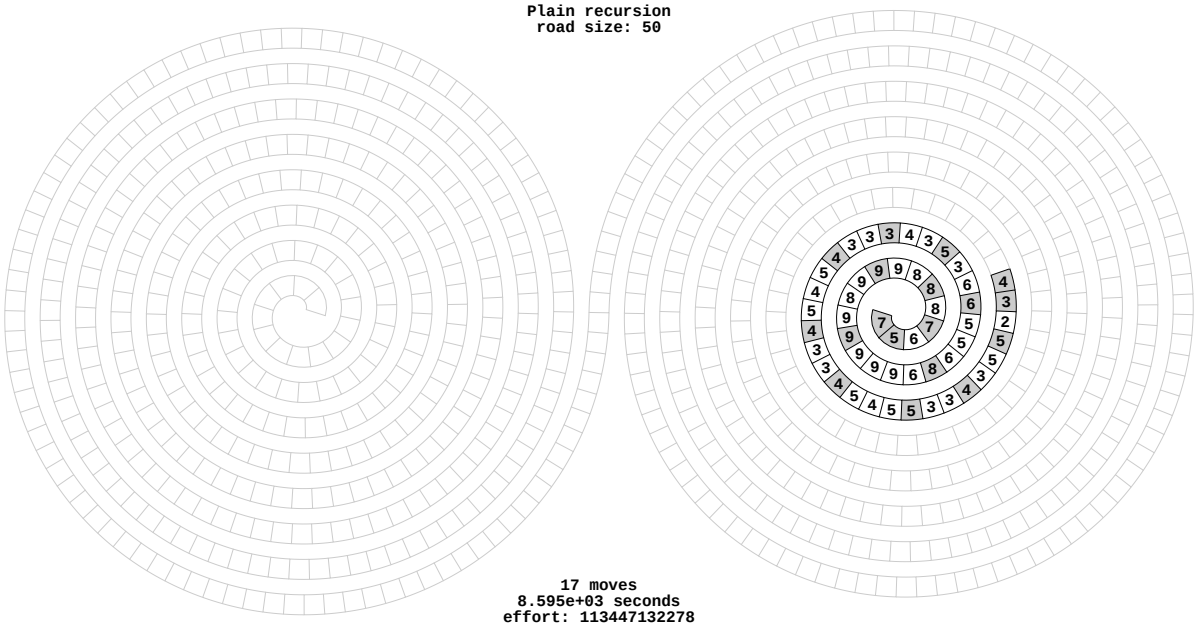

Figura 7.9: Solução Brute Force (Professor)

Brute Force Melhorada

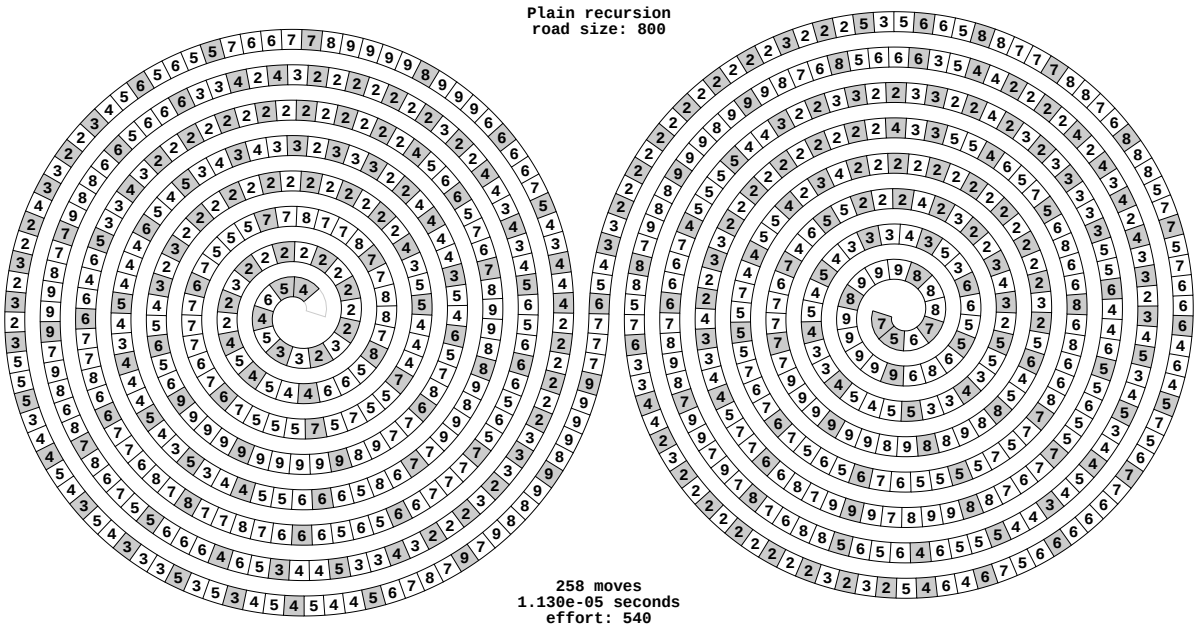

Figura 7.10: Solução Brute Force Melhorada

While

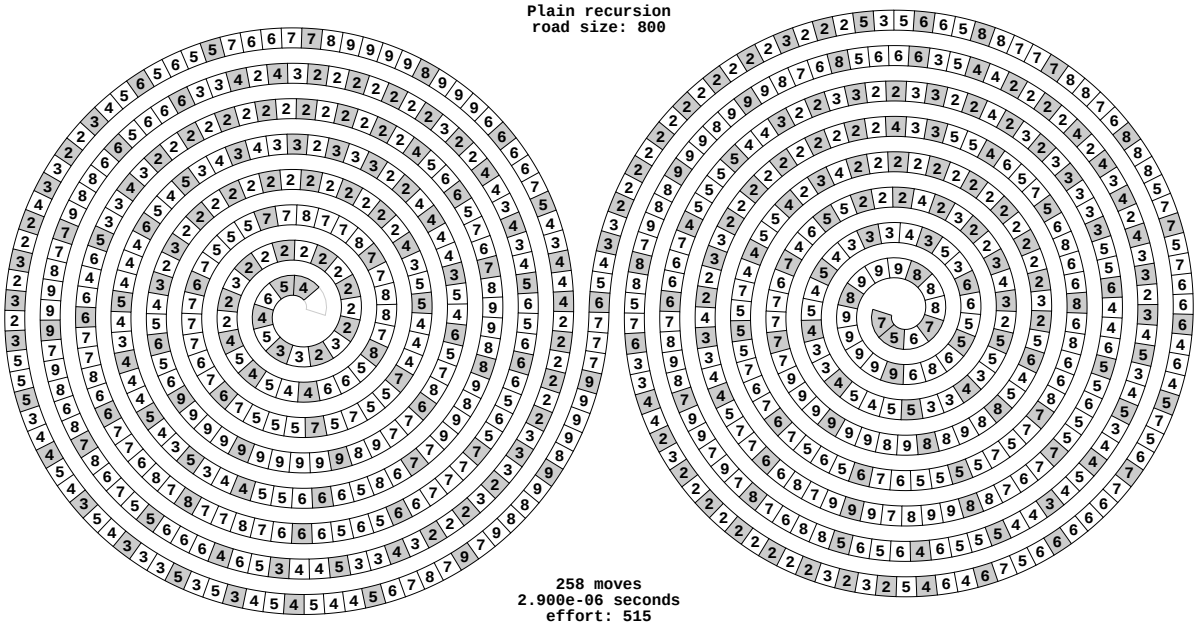

Figura 7.11: Solução While

While Dynamic

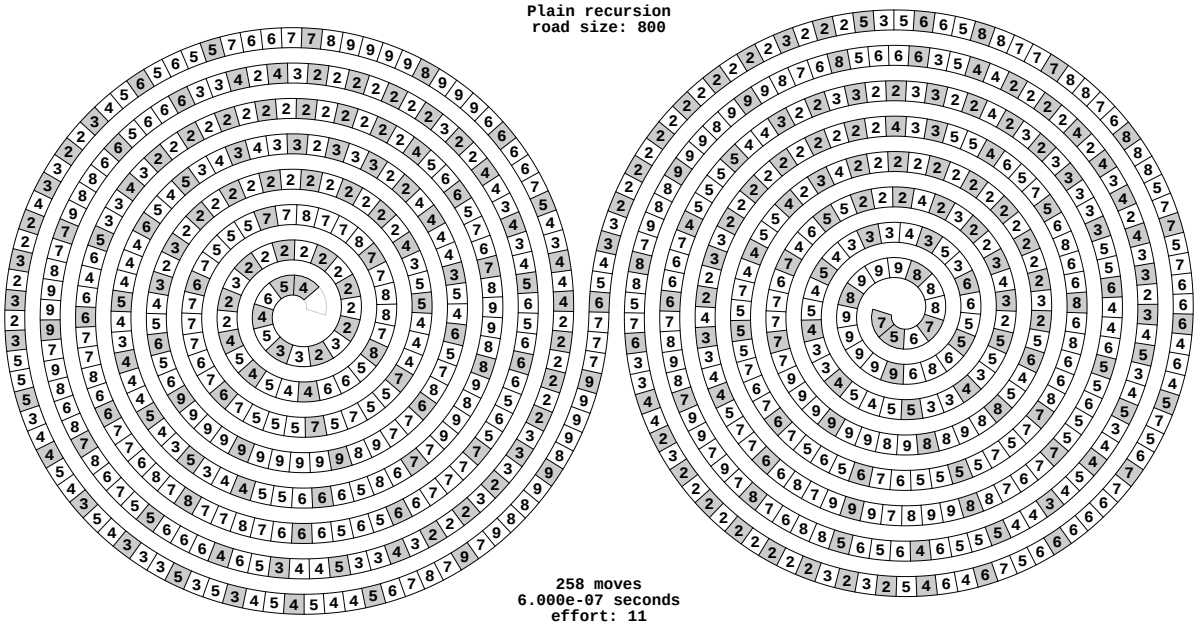

Figura 7.12: Solução While Dynamic

Capítulo 8

Código MatLab/Anexo 3

Comparação do tempo real/estimado da solução Brute Force no PC1

```
A = load("solution_1_108215.txt");

n = A(:,1); %primeira coluna
t = A(:,4); %quarta coluna

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

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, Ntotal*Coefs,'k')

t800_log = [800 1] * Coefs;
t800 = 10^t800_log;
fprintf("Tempo de execucao estimado para as 800 posicoes no PC1:%.3d ", t800);

title("Brute Force - PC1")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("Tempo Real", "Tempo estimado")
```

Comparação do tempo real/estimado da solução Brute Force no PC2

```
A = load("A_108579.txt");

n = A(:,1); %primeira coluna
t = A(:,4); %quarta coluna

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

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, Ntotal*Coefs,'k')

t800_log = [800 1] * Coefs;
t800 = 10^t800_log;
fprintf("Tempo de execucao estimado para as 800 posicoes:%.3d segundos", t800);

title("Brute Force - PC2")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("Tempo Real", "Tempo estimado")
```

Comparação do tempo real/estimado da solução Brute Force no PC3

```
A = load("solution_1_108796.txt");

n = A(:,1); %primeira coluna
t = A(:,4); %quarta coluna

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

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, Ntotal*Coefs,'k')

t800_log = [800 1] * Coefs;
t800 = 10^t800_log;
fprintf("Tempo de execucao estimado para as 800 posicoes no PC3:%.3d ", t800);

title("Brute Force - PC3")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("Tempo Real", "Tempo estimado")
```

Comparação da solução Brute Force Melhorada em todos os PCs

```
valores2 = load("solution_3_108215.txt"); %PC1
valores2_2 = load("C_108579.txt"); %PC2
valores2_3 = load("solution_3_108796.txt"); %PC3

%PC1
n2 = valores2(:,1); %primeira coluna
t2 = valores2(:,4); %quarta coluna

%PC2
n2_2 = valores2_2(:,1); %primeira coluna
t2_2 = valores2_2(:,4); %quarta coluna

%PC3
n2_3 = valores2_3(:,1); %primeira coluna
t2_3 = valores2_3(:,4); %quarta coluna

figure(3)

%PC1
p2 = semilogy(n2,t2);
p2.LineStyle = ":";
p2.Color = "#16004C";
p2.Marker = ".";
hold on;

%PC2
p2_2 = semilogy(n2_2,t2_2);
p2_2.LineStyle = ":";
p2_2.Color = "#4900FF";
p2_2.Marker = ".";
hold on;

%PC3
p2_3 = semilogy(n2_3,t2_3);
p2_3.LineStyle = ":";
p2_3.Color = "#8E63F9";
p2_3.Marker = ".";
hold on;

title("Brute Force Melhorada-Todos os PCs")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("PC1-Brute Force Melhorada","PC2-Brute Force Melhorada","PC3-Brute Force
    Melhorada")
```

Comparação da solução While em todos os PCs

```
%PC1
valores3 = load("solution_4_108215.txt");

%PC2
valores3_2 = load("D_108579.txt");

%PC3
valores3_3 = load("solution_4_108796.txt");

%PC1
n4 = valores3(:,1); %primeira coluna
t4 = valores3(:,4); %quarta coluna

%PC2
n4_2 = valores3_2(:,1); %primeira coluna
t4_2 = valores3_2(:,4); %quarta coluna

%PC3
n4_3 = valores3_3(:,1); %primeira coluna
t4_3 = valores3_3(:,4); %quarta coluna

figure(4)

%PC1
p4 = semilogy(n4,t4);
p4.LineStyle = ":";
p4.Color = "#95FF65";
p4.Marker = ".";
hold on;

%PC2
p4_2 = semilogy(n4_2,t4_2);
p4_2.LineStyle = ":";
p4_2.Color = "#32A700";
p4_2.Marker = ".";
hold on;

%PC3
p4_3 = semilogy(n4_3,t4_3);
p4_3.LineStyle = ":";
p4_3.Color = "#184902";
p4_3.Marker = ".";
hold on;

title("While-Todos os PCs")

xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")

legend("PC1-While","PC2-While","PC3-While")
```

Comparação da solução While Dynamic em todos os PCs

```
%PC1
valores1 = load("solution_2_108215.txt");

%PC2
valores1_2 = load("B_108579.txt");

%PC3
valores1_3 = load("solution_2_108796.txt");

%PC1
n3 = valores1(:,1); %primeira coluna
t3 = valores1(:,4); %quarta coluna

%PC2
n3_2 = valores1_2(:,1); %primeira coluna
t3_2 = valores1_2(:,4); %quarta coluna

%PC3
n3_3 = valores1_3(:,1); %primeira coluna
t3_3 = valores1_3(:,4); %quarta coluna

figure(2)

%PC1
p3 = semilogy(n3,t3);
p3.LineStyle = ":";
p3.Color = "#FFC08E";
p3.Marker = ".";
hold on;

%PC2
p3_2 = semilogy(n3_2,t3_2);
p3_2.LineStyle = ":";
p3_2.Color = "#F47629";
p3_2.Marker = ".";
hold on;

%PC3
p3_3 = semilogy(n3_3,t3_3);
p3_3.LineStyle = ":";
p3_3.Color = "#612B00";
p3_3.Marker = ".";
hold on;

title("While Dynamic-Todos os PCs")

xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")

legend("PC1-While Dynamic","PC2-While Dynamic","PC3-While Dynamic")
```

Comparação de todas as soluções no PC1

```
valores = load("solution_1_108215.txt"); %Brute Force
valores1 = load("solution_2_108215.txt"); %While_Dynamic
valores2 = load("solution_3_108215.txt"); %Brute Force Melhorada
valores3 = load("solution_4_108215.txt"); %While

%% Brute Force
n1 = valores(:,1); %primeira coluna
t1 = valores(:,4); %quarta coluna

figure(1)

p1 = semilogy(n1,t1);
p1.LineStyle = ":";
p1.Color = "#F11010";
p1.Marker = ".";
hold on;

%% Brute Force Melhorada
n2 = valores2(:,1); %primeira coluna
t2 = valores2(:,4); %quarta coluna

p2 = semilogy(n2,t2);
p2.LineStyle = ":";
p2.Color = "#8E63F9";
p2.Marker = ".";
hold on;

%% While Dynamic
n3 = valores1(:,1); %primeira coluna
t3 = valores1(:,4); %quarta coluna

p3 = semilogy(n3,t3);
p3.LineStyle = ":";
p3.Color = "#F47629";
p3.Marker = ".";
hold on;


%% While
n4 = valores3(:,1); %primeira coluna
t4 = valores3(:,4); %quarta coluna

p4 = semilogy(n4,t4);
p4.LineStyle = ":";
p4.Color = "#3C9F11";
p4.Marker = ".";

title("PC1")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("Brute Force","Brute Force Melhorada","While Dynamic","While")
```

Comparação de todas as soluções no PC2

```
valores = load("A_108579.txt"); %Brute Force
valores1 = load("B_108579.txt"); %While_Dynamic
valores2 = load("C_108579.txt"); %Brute Force Melhorada
valores3 = load("D_108579.txt"); %While

%% Brute Force
n1 = valores(:,1); %primeira coluna
t1 = valores(:,4); %quarta coluna

figure(1)
p1 = semilogy(n1,t1);
p1.LineStyle = ":";
p1.Color = "#F11010";
p1.Marker = ".";
hold on;

%% Brute Force Melhorada
n2 = valores2(:,1); %primeira coluna
t2 = valores2(:,4); %quarta coluna

p2 = semilogy(n2,t2);
p2.LineStyle = ":";
p2.Color = "#8E63F9";
p2.Marker = ".";
hold on;

%% While Dynamic
n3 = valores1(:,1); %primeira coluna
t3 = valores1(:,4); %quarta coluna

p3 = semilogy(n3,t3);
p3.LineStyle = ":";
p3.Color = "#F47629";
p3.Marker = ".";
hold on;


%% While
n4 = valores3(:,1); %primeira coluna
t4 = valores3(:,4); %quarta coluna

p4 = semilogy(n4,t4);
p4.LineStyle = ":";
p4.Color = "#3C9F11";
p4.Marker = ".";

title("PC2")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("Brute Force","Brute Force Melhorada","While Dynamic","While")
```

Comparação de todas as soluções no PC3

```
valores = load("solution_1_108796.txt"); %Brute Force
valores1 = load("solution_2_108796.txt"); %While_Dynamic
valores2 = load("solution_3_108796.txt"); %Brute Force Melhorada
valores3 = load("solution_4_108796.txt"); %While

%% Brute Force
n1 = valores(:,1); %primeira coluna
t1 = valores(:,4); %quarta coluna

figure(1)
p1 = semilogy(n1,t1);
p1.LineStyle = ":";
p1.Color = "#F11010";
p1.Marker = ".";
hold on;

%% Brute Force Melhorada
n2 = valores2(:,1); %primeira coluna
t2 = valores2(:,4); %quarta coluna

p2 = semilogy(n2,t2);
p2.LineStyle = ":";
p2.Color = "#8E63F9";
p2.Marker = ".";
hold on;

%% While Dynamic
n3 = valores1(:,1); %primeira coluna
t3 = valores1(:,4); %quarta coluna

p3 = semilogy(n3,t3);
p3.LineStyle = ":";
p3.Color = "#F47629";
p3.Marker = ".";
hold on;


%% While
n4 = valores3(:,1); %primeira coluna
t4 = valores3(:,4); %quarta coluna

p4 = semilogy(n4,t4);
p4.LineStyle = ":";
p4.Color = "#3C9F11";
p4.Marker = ".";

title("PC3")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")
legend("Brute Force","Brute Force Melhorada","While Dynamic","While")
```

Comparação de todas as soluções em todos os PCs

```
%PC1
valores = load("solution_1_108215.txt"); %Brute Force
valores1 = load("solution_2_108215.txt"); %While Dynamic
valores2 = load("solution_3_108215.txt"); %Brute Force Melhorada
valores3 = load("solution_4_108215.txt"); %While

%PC2
valores_2 = load("A_108579.txt"); %Brute Force
valores1_2 = load("B_108579.txt"); %While Dynamic
valores2_2 = load("C_108579.txt"); %Brute Force Melhorada
valores3_2 = load("D_108579.txt"); %While

%PC3
valores_3 = load("solution_1_108796.txt"); %Brute Force
valores1_3 = load("solution_2_108796.txt"); %While Dynamic
valores2_3 = load("solution_3_108796.txt"); %Brute Force Melhorada
valores3_3 = load("solution_4_108796.txt"); %While

%Brute Force
%PC1
n1 = valores(:,1); %primeira coluna
t1 = valores(:,4); %quarta coluna

%PC2
n1_2 = valores_2(:,1); %primeira coluna
t1_2 = valores_2(:,4); %quarta coluna

%PC3
n1_3 = valores_3(:,1); %primeira coluna
t1_3 = valores_3(:,4); %quarta coluna

figure(5)

%PC1
p1 = semilogy(n1,t1);
p1.LineStyle = ":";
p1.Color = "#FF6666";
p1.Marker = ".";
hold on;

%PC2
p1_2 = semilogy(n1_2,t1_2);
p1_2.LineStyle = ":";
p1_2.Color = "#F11010";
p1_2.Marker = ".";
hold on;

%PC3
p1_3 = semilogy(n1_3,t1_3);
p1_3.LineStyle = ":";
p1_3.Color = "#660101";
p1_3.Marker = ".";
hold on;

title("Todos os PCs")
xlabel("Pos. final (n)")
ylabel("Tempo(log10(s))")

%% Brute Force Melhorada
%PC1
n2 = valores2(:,1); %primeira coluna
```

```
t2 = valores2(:,4); %quarta coluna

%PC2
n2_2 = valores2_2(:,1); %primeira coluna
t2_2 = valores2_2(:,4); %quarta coluna

%PC3
n2_3 = valores2_3(:,1); %primeira coluna
t2_3 = valores2_3(:,4); %quarta coluna

%PC1
p2 = semilogy(n2,t2);
p2.LineStyle = ":";
p2.Color = "#16004C";
p2.Marker = ".";
hold on;

%PC2
p2_2 = semilogy(n2_2,t2_2);
p2_2.LineStyle = ":";
p2_2.Color = "#4900FF";
p2_2.Marker = ".";
hold on;

%PC3
p2_3 = semilogy(n2_3,t2_3);
p2_3.LineStyle = ":";
p2_3.Color = "#8E63F9";
p2_3.Marker = ".";
hold on;

%% While Dynamic
%PC1
n3 = valores1(:,1); %primeira coluna
t3 = valores1(:,4); %quarta coluna

%PC2
n3_2 = valores1_2(:,1); %primeira coluna
t3_2 = valores1_2(:,4); %quarta coluna

%PC3
n3_3 = valores1_3(:,1); %primeira coluna
t3_3 = valores1_3(:,4); %quarta coluna

%PC1
p3 = semilogy(n3,t3);
p3.LineStyle = ":";
p3.Color = "#FFC08E";
p3.Marker = ".";
hold on;

%PC2
p3_2 = semilogy(n3_2,t3_2);
p3_2.LineStyle = ":";
p3_2.Color = "#F47629";
p3_2.Marker = ".";
hold on;

%PC3
p3_3 = semilogy(n3_3,t3_3);
p3_3.LineStyle = ":";
p3_3.Color = "#612B00";
p3_3.Marker = ".";
```

```
hold on;


%% While
%PC1
n4 = valores3(:,1); %primeira coluna
t4 = valores3(:,4); %quarta coluna

%PC2
n4_2 = valores3_2(:,1); %primeira coluna
t4_2 = valores3_2(:,4); %quarta coluna

%PC3
n4_3 = valores3_3(:,1); %primeira coluna
t4_3 = valores3_3(:,4); %quarta coluna

%PC1
p4 = semilogy(n4,t4);
p4.LineStyle = ":";
p4.Color = "#95FF65";
p4.Marker = ".";
hold on;

%PC2
p4_2 = semilogy(n4_2,t4_2);
p4_2.LineStyle = ":";
p4_2.Color = "#32A700";
p4_2.Marker = ".";
hold on;

%PC3
p4_3 = semilogy(n4_3,t4_3);
p4_3.LineStyle = ":";
p4_3.Color = "#184902";
p4_3.Marker = ".";
hold on;

legend("PC1-Force Break","PC2-Force Break","PC3-Force Break", ...
    "PC1-Force Break Melhorada","PC2-Force Break Melhorada","PC3-Force Break
    Melhorada", ...
    "PC1-While Dynamic","PC2-While Dynamic","PC3-While Dynamic", ...
    "PC1-While","PC2-While","PC3-While")
```

Capítulo 9

Soluções/Anexo 4

PC1

Brute Force (Professor)			
		plain recursion	
n	sol	count	cpu time
1	1	2	3.637e-06
2	2	3	1.322e-06
3	3	5	1.673e-06
4	3	8	1.122e-06
5	4	13	1.042e-06
6	4	22	1.112e-06
7	5	36	1.272e-06
8	5	60	1.443e-06
9	5	100	6.082e-06
10	6	167	2.815e-06
11	6	279	4.679e-06
12	6	465	4.769e-06
13	7	777	6.642e-06
14	7	1297	1.039e-05
15	7	2165	1.773e-05
16	7	3614	2.886e-05
17	8	6031	4.391e-05
18	8	10065	7.777e-05
19	8	16795	1.200e-04
20	8	28024	1.993e-04
21	9	46758	3.138e-04
22	9	78011	5.491e-04
23	9	130089	7.623e-04
24	9	216968	1.381e-03
25	9	359706	2.245e-03
26	10	597823	3.738e-03
27	10	995046	6.060e-03
28	10	1655498	9.785e-03
29	10	2757259	1.358e-02
30	10	4593012	2.209e-02
31	11	7651017	3.697e-02
32	11	12747967	6.277e-02
33	11	21239691	1.042e-01
34	12	35390165	1.838e-01
35	12	58969547	2.958e-01
36	12	98258424	4.829e-01
37	13	163727428	7.910e-01
38	13	272817267	1.320e+00
39	13	454593881	2.269e+00
40	14	757489987	3.759e+00
41	14	1262204160	6.443e+00
42	14	2114047092	1.090e+01
43	15	3533472456	1.804e+01
44	15	5898663878	2.912e+01
45	15	9850621736	4.871e+01
46	16	16352251531	8.022e+01
47	16	27196767437	1.331e+02
48	16	45288671397	2.165e+02
49	16	75373390362	3.556e+02
50	17	125557790155	5.986e+02
55	19	1421772009146	6.593e+03

Brute Force Melhorada			
		plain recursion	
n	sol	count	cpu time
1	1	2	4.088e-06
2	2	3	8.310e-07
3	3	5	5.910e-07
4	3	5	4.510e-07
5	4	7	5.310e-07
6	4	8	5.110e-07

```
  7    5              10 4.810e-07
  8    5              12 5.210e-07
  9    5              10 4.910e-07
 10    6              13 5.510e-07
 11    6              15 1.983e-06
 12    6              13 6.510e-07
 13    7              17 7.110e-07
 14    7              20 7.620e-07
 15    7              20 7.120e-07
 16    7              16 6.420e-07
 17    8              20 6.610e-07
 18    8              23 6.610e-07
 19    8              24 8.220e-07
 20    8              21 7.620e-07
 21    9              25 1.643e-06
 22    9              29 6.520e-07
 23    9              30 6.310e-07
 24    9              28 7.010e-07
 25    9              21 5.810e-07
 26   10              24 5.910e-07
 27   10              27 6.320e-07
 28   10              26 6.410e-07
 29   10              24 6.110e-07
 30   10              20 5.210e-07
 31   11              22 5.510e-07
 32   11              24 1.092e-06
 33   11              22 1.232e-06
 34   12              24 1.122e-06
 35   12              26 1.182e-06
 36   12              24 1.282e-06
 37   13              26 1.213e-06
 38   13              28 1.032e-06
 39   13              26 1.202e-06
 40   14              28 5.810e-07
 41   14              30 5.610e-07
 42   14              28 5.620e-07
 43   15              31 5.710e-07
 44   15              33 5.810e-07
 45   15              31 5.510e-07
 46   16              34 5.920e-07
 47   16              36 6.220e-07
 48   16              35 6.410e-07
 49   16              32 6.310e-07
 50   17              34 6.010e-07
 55   19              37 2.044e-06
 60   21              42 8.610e-07
 65   23              48 8.020e-07
 70   25              53 7.520e-07
 75   26              59 8.310e-07
 80   27              68 9.620e-07
 85   28              73 1.012e-06
 90   30              66 1.233e-06
 95   32              70 1.052e-06
100   35              76 9.820e-07
110   39              87 2.294e-06
120   42              98 1.423e-06
130   44             108 1.503e-06
140   46             113 1.322e-06
150   48             115 1.683e-06
160   52             110 1.433e-06
170   57             121 1.573e-06
180   60             131 1.473e-06
190   62             146 1.673e-06
200   64             146 1.744e-06
220   70             146 2.685e-06
240   77             168 1.753e-06
260   81             171 1.603e-06
280   88             182 1.643e-06
300   96             205 3.747e-06
320  101             213 2.013e-06
340  111             235 1.893e-06
360  115             244 1.933e-06
380  120             259 2.244e-06
400  128             270 2.154e-06
420  134             291 4.949e-06
440  139             293 3.286e-06
460  147             310 5.079e-06
480  152             324 3.236e-06
500  159             332 3.226e-06
520  166             348 3.156e-06
540  171             379 3.526e-06
560  178             375 3.697e-06
580  185             398 3.707e-06
600  189             415 3.877e-06
620  195             410 4.859e-06
640  203             431 3.897e-06
660  208             440 4.148e-06
680  214             452 4.157e-06
700  222             475 4.128e-06
720  227             484 4.559e-06
740  236             506 4.709e-06
760  241             521 4.508e-06
780  244             526 4.699e-06
800  253             538 6.442e-06

While --- ----------------- --------- +
    |                plain recursion |
--- + --- ----------------- --------- +
  n | sol            count  cpu time |
```

| --- | --- | --- | --- |
1	1	1	3.827e-06
2	2	3	9.120e-07
3	3	5	7.020e-07
4	3	5	8.520e-07
5	4	7	8.120e-07
6	4	7	7.820e-07
7	5	9	9.320e-07
8	5	9	1.172e-06
9	5	9	9.310e-07
10	6	11	9.310e-07
11	6	11	2.515e-06
12	6	11	7.610e-07
13	7	13	4.910e-07
14	7	13	4.500e-07
15	7	13	4.610e-07
16	7	13	4.410e-07
17	8	15	4.710e-07
18	8	15	4.910e-07
19	8	15	4.910e-07
20	8	15	5.000e-07
21	9	17	1.463e-06
22	9	17	6.210e-07
23	9	17	4.810e-07
24	9	17	5.610e-07
25	9	17	6.010e-07
26	10	19	6.010e-07
27	10	19	5.710e-07
28	10	19	5.410e-07
29	10	19	6.010e-07
30	10	19	6.110e-07
31	11	21	5.310e-07
32	11	21	4.710e-07
33	11	21	4.500e-07
34	12	23	4.400e-07
35	12	23	4.510e-07
36	12	23	4.610e-07
37	13	25	4.810e-07
38	13	25	4.710e-07
39	13	25	4.610e-07
40	14	27	4.510e-07
41	14	27	4.710e-07
42	14	27	4.910e-07
43	15	29	5.010e-07
44	15	29	5.310e-07
45	15	29	5.110e-07
46	16	31	5.210e-07
47	16	31	5.310e-07
48	16	31	5.410e-07
49	16	31	5.110e-07
50	17	33	5.010e-07
55	19	37	2.224e-06
60	21	41	7.910e-07
65	23	45	6.110e-07
70	25	49	6.110e-07
75	26	51	6.620e-07
80	27	53	6.720e-07
85	28	55	7.310e-07
90	30	59	7.820e-07
95	32	63	1.082e-06
100	35	69	8.220e-07
110	39	77	2.204e-06
120	42	83	9.410e-07
130	44	87	9.220e-07
140	46	91	9.020e-07
150	48	95	9.520e-07
160	52	103	9.320e-07
170	57	113	9.310e-07
180	60	119	1.002e-06
190	62	123	1.021e-06
200	64	127	1.062e-06
220	70	139	3.888e-06
240	77	153	2.164e-06
260	81	161	1.683e-06
280	88	175	1.493e-06
300	96	191	1.482e-06
320	101	201	1.492e-06
340	111	221	1.623e-06
360	115	229	1.613e-06
380	120	239	1.683e-06
400	128	255	1.753e-06
420	134	267	4.268e-06
440	139	277	1.913e-06
460	147	293	1.903e-06
480	152	303	1.943e-06
500	159	317	1.924e-06
520	166	331	2.034e-06
540	171	341	2.084e-06
560	178	355	2.104e-06
580	185	369	2.154e-06
600	189	377	2.224e-06
620	195	389	2.264e-06
640	203	405	2.354e-06
660	208	415	2.435e-06
680	214	427	2.504e-06
700	222	443	2.545e-06
720	227	453	2.595e-06
740	236	471	2.695e-06
760	241	481	2.746e-06

```
780  244              487 2.896e-06
800  253              505 2.915e-06
```

While-Dynamic

	plain recursion	
n	sol	count	cpu time
1	1	1	3.807e-06
2	2	2	8.420e-07
3	3	2	5.110e-07
4	3	2	4.710e-07
5	4	2	4.710e-07
6	4	2	4.810e-07
7	5	3	5.210e-07
8	5	3	5.110e-07
9	5	3	4.710e-07
10	6	3	5.110e-07
11	6	3	1.483e-06
12	6	3	6.520e-07
13	7	4	5.610e-07
14	7	4	5.310e-07
15	7	4	5.010e-07
16	7	4	5.910e-07
17	8	4	5.210e-07
18	8	4	5.310e-07
19	8	4	5.110e-07
20	8	4	5.210e-07
21	9	5	1.523e-06
22	9	5	6.310e-07
23	9	5	6.110e-07
24	9	5	5.810e-07
25	9	5	5.510e-07
26	10	5	5.820e-07
27	10	5	5.710e-07
28	10	5	5.410e-07
29	10	5	5.510e-07
30	10	5	5.610e-07
31	11	3	4.910e-07
32	11	3	4.710e-07
33	11	3	4.410e-07
34	12	3	4.610e-07
35	12	3	4.400e-07
36	12	3	4.800e-07
37	13	3	4.810e-07
38	13	3	4.810e-07
39	13	3	4.710e-07
40	14	3	4.910e-07
41	14	3	4.600e-07
42	14	3	4.810e-07
43	15	3	4.910e-07
44	15	3	5.110e-07
45	15	3	5.010e-07
46	16	4	5.010e-07
47	16	4	5.010e-07
48	16	4	4.810e-07
49	16	4	4.910e-07
50	17	3	4.410e-07
55	19	5	2.525e-06
60	21	4	5.610e-07
65	23	4	5.210e-07
70	25	5	5.610e-07
75	26	5	5.720e-07
80	27	5	5.710e-07
85	28	6	5.910e-07
90	30	7	7.210e-07
95	32	4	5.010e-07
100	35	4	4.710e-07
110	39	6	1.483e-06
120	42	6	6.010e-07
130	44	6	6.010e-07
140	46	7	5.510e-07
150	48	7	6.710e-07
160	52	10	6.710e-07
170	57	7	5.510e-07
180	60	6	5.810e-07
190	62	6	6.410e-07
200	64	7	6.510e-07
220	70	12	3.437e-06
240	77	8	6.810e-07
260	81	8	7.110e-07
280	88	10	6.620e-07
300	96	10	6.310e-07
320	101	9	7.110e-07
340	111	11	5.710e-07
360	115	7	6.210e-07
380	120	8	6.910e-07
400	128	13	6.510e-07
420	134	8	1.884e-06
440	139	10	7.710e-07
460	147	11	6.310e-07
480	152	8	6.410e-07
500	159	11	7.010e-07
520	166	9	6.910e-07
540	171	8	7.020e-07
560	178	13	7.410e-07
580	185	9	5.810e-07
600	189	8	6.610e-07
620	195	11	6.610e-07

```
640  203              9 5.920e-07
660  208              9 6.610e-07
680  214              9 6.110e-07
700  222             10 5.810e-07
720  227              9 6.710e-07
740  236             11 5.610e-07
760  241              8 6.010e-07
780  244              8 6.720e-07
800  253             15 7.110e-07
```

PC2

Brute Force (Professor)			---
		plain recursion	
n	sol	count	cpu time
1	1	2	3.600e-06
2	2	3	6.000e-07
3	3	5	4.000e-07
4	3	8	4.000e-07
5	4	13	4.000e-07
6	4	22	6.000e-07
7	5	36	6.000e-07
8	5	60	7.000e-07
9	5	100	1.000e-06
10	6	167	1.300e-06
11	6	279	3.400e-06
12	6	465	2.900e-06
13	7	777	5.000e-06
14	7	1297	8.800e-06
15	7	2165	1.510e-05
16	7	3614	2.430e-05
17	8	6031	3.760e-05
18	8	10065	6.240e-05
19	8	16795	9.040e-05
20	8	28024	1.591e-04
21	9	46758	2.312e-04
22	9	78011	3.749e-04
23	9	130089	6.121e-04
24	9	216968	1.017e-03
25	9	359706	1.633e-03
26	10	597823	2.773e-03
27	10	995046	4.533e-03
28	10	1655498	7.407e-03
29	10	2757259	1.276e-02
30	10	4593012	2.098e-02
31	11	7651017	3.440e-02
32	11	12747967	5.751e-02
33	11	21239691	9.227e-02
34	12	35390165	1.592e-01
35	12	58969547	2.738e-01
36	12	98258424	4.497e-01
37	13	163727428	7.427e-01
38	13	272817267	1.236e+00
39	13	454593881	2.051e+00
40	14	757489987	3.425e+00
41	14	1262204160	5.703e+00
42	14	2114047092	9.763e+00
43	15	3533472456	1.628e+01
44	15	5898663878	2.682e+01
45	15	9850621736	4.639e+01
46	16	16352251531	8.668e+01
47	16	27196767437	1.445e+02
48	16	45288671397	2.531e+02
49	16	75373390362	3.981e+02
50	17	125557790155	6.823e+02
55	19	1421772009146	7.556e+03

Brute Force Melhorada			----
		plain recursion	
n	sol	count	cpu time
1	1	2	2.800e-06
2	2	3	5.000e-07
3	3	5	4.000e-07
4	3	5	2.000e-07
5	4	7	3.000e-07
6	4	8	3.000e-07
7	5	10	2.000e-07
8	5	12	3.000e-07
9	5	10	3.000e-07
10	6	13	3.000e-07
11	6	15	8.000e-07
12	6	13	3.000e-07
13	7	17	3.000e-07
14	7	20	4.000e-07
15	7	20	4.000e-07
16	7	16	3.000e-07
17	8	20	3.000e-07
18	8	23	4.000e-07
19	8	24	4.000e-07
20	8	21	4.000e-07
21	9	25	1.100e-06
22	9	29	5.000e-07
23	9	28	5.000e-07
24	9	25	5.000e-07
25	9	19	3.000e-07
26	10	21	4.000e-07
27	10	23	4.000e-07
28	10	21	3.000e-07
29	11	23	3.000e-07
30	11	25	4.000e-07
31	11	23	4.000e-07
32	12	25	4.000e-07

33	12	27	3.000e-07
34	12	25	3.000e-07
35	13	27	3.000e-07
36	13	29	4.000e-07
37	13	27	4.000e-07
38	14	29	4.000e-07
39	14	31	4.000e-07
40	14	29	3.000e-07
41	15	32	4.000e-07
42	15	34	4.000e-07
43	15	32	4.000e-07
44	16	35	4.000e-07
45	16	37	4.000e-07
46	16	36	4.000e-07
47	16	33	4.000e-07
48	17	35	4.000e-07
49	17	37	4.000e-07
50	17	35	4.000e-07
55	19	38	1.400e-06
60	22	44	8.000e-07
65	24	51	8.000e-07
70	26	56	6.000e-07
75	27	62	6.000e-07
80	28	71	7.000e-07
85	29	75	7.000e-07
90	30	67	8.000e-07
95	32	68	7.000e-07
100	35	74	6.000e-07
110	39	83	1.900e-06
120	42	97	1.000e-06
130	44	106	9.000e-07
140	46	110	8.000e-07
150	48	107	1.100e-06
160	53	111	1.100e-06
170	58	121	9.000e-07
180	61	134	1.200e-06
190	63	138	1.100e-06
200	65	152	1.200e-06
220	72	150	2.300e-06
240	80	170	1.800e-06
260	84	176	3.700e-06
280	91	190	1.600e-06
300	98	210	1.500e-06
320	103	219	1.600e-06
340	113	241	1.600e-06
360	117	249	1.700e-06
380	122	271	1.900e-06
400	131	281	1.900e-06
420	136	304	4.600e-06
440	141	301	2.600e-06
460	148	315	2.300e-06
480	154	337	2.300e-06
500	160	340	2.400e-06
520	168	358	2.300e-06
540	173	390	2.600e-06
560	179	381	2.600e-06
580	186	403	3.100e-06
600	190	418	2.800e-06
620	197	421	2.800e-06
640	205	445	2.900e-06
660	209	450	2.800e-06
680	215	460	2.900e-06
700	223	480	3.000e-06
720	227	485	3.100e-06
740	237	507	3.700e-06
760	241	519	3.200e-06
780	245	536	3.600e-06
800	254	543	3.400e-06

While

		plain recursion	
n	sol	count	cpu time
1	1	1	3.500e-06
2	2	3	5.000e-07
3	3	5	3.000e-07
4	3	5	2.000e-07
5	4	7	2.000e-07
6	4	7	3.000e-07
7	5	9	2.000e-07
8	5	9	2.000e-07
9	5	9	2.000e-07
10	6	11	3.000e-07
11	6	11	8.000e-07
12	6	11	3.000e-07
13	7	13	3.000e-07
14	7	13	3.000e-07
15	7	13	3.000e-07
16	7	13	3.000e-07
17	8	15	3.000e-07
18	8	15	3.000e-07
19	8	15	3.000e-07
20	8	15	3.000e-07
21	9	17	1.200e-06
22	9	17	4.000e-07
23	9	17	4.000e-07
24	9	17	4.000e-07
25	9	17	4.000e-07

26	10	19	4.000e-07
27	10	19	3.000e-07
28	10	19	3.000e-07
29	11	21	3.000e-07
30	11	21	3.000e-07
31	11	21	3.000e-07
32	12	23	3.000e-07
33	12	23	3.000e-07
34	12	23	3.000e-07
35	13	25	3.000e-07
36	13	25	3.000e-07
37	13	25	3.000e-07
38	14	27	3.000e-07
39	14	27	3.000e-07
40	14	27	3.000e-07
41	15	29	3.000e-07
42	15	29	4.000e-07
43	15	29	4.000e-07
44	16	31	4.000e-07
45	16	31	4.000e-07
46	16	31	4.000e-07
47	16	31	3.000e-07
48	17	33	3.000e-07
49	17	33	4.000e-07
50	17	33	4.000e-07
55	19	37	1.000e-06
60	22	43	5.000e-07
65	24	47	6.000e-07
70	26	51	5.000e-07
75	27	53	5.000e-07
80	28	55	5.000e-07
85	29	57	5.000e-07
90	30	59	6.000e-07
95	32	63	6.000e-07
100	35	69	5.000e-07
110	39	77	1.400e-06
120	42	83	7.000e-07
130	44	87	7.000e-07
140	46	91	6.000e-07
150	48	95	7.000e-07
160	53	105	7.000e-07
170	58	115	6.000e-07
180	61	121	7.000e-07
190	63	125	7.000e-07
200	65	129	8.000e-07
220	72	143	1.600e-06
240	80	159	1.200e-06
260	84	167	1.000e-06
280	91	181	1.600e-06
300	98	195	1.200e-06
320	103	205	1.200e-06
340	113	225	1.200e-06
360	117	233	1.600e-06
380	122	243	1.400e-06
400	131	261	1.400e-06
420	136	271	2.500e-06
440	141	281	2.000e-06
460	148	295	1.900e-06
480	154	307	1.700e-06
500	160	319	1.700e-06
520	168	335	1.800e-06
540	173	345	1.800e-06
560	179	357	1.900e-06
580	186	371	1.900e-06
600	190	379	1.900e-06
620	197	393	2.000e-06
640	205	409	2.000e-06
660	209	417	2.100e-06
680	215	429	2.100e-06
700	223	445	2.200e-06
720	227	453	2.300e-06
740	237	473	2.300e-06
760	241	481	2.400e-06
780	245	489	2.500e-06
800	254	507	3.400e-06

While-Dynamic

		plain recursion	
n	sol	count	cpu time
1	1	1	5.300e-06
2	2	2	2.700e-06
3	3	2	6.000e-07
4	3	2	5.000e-07
5	4	2	5.000e-07
6	4	2	5.000e-07
7	5	3	5.000e-07
8	5	3	5.000e-07
9	5	3	4.000e-07
10	6	3	4.000e-07
11	6	3	1.100e-06
12	6	3	5.000e-07
13	7	4	5.000e-07
14	7	4	4.000e-07
15	7	4	4.000e-07
16	7	4	3.000e-07
17	8	4	4.000e-07
18	8	4	4.000e-07

19	8	4	4.000e-07
20	8	4	4.000e-07
21	9	5	1.700e-06
22	9	5	4.000e-07
23	9	5	5.000e-07
24	9	5	5.000e-07
25	9	5	4.000e-07
26	10	3	4.000e-07
27	10	3	4.000e-07
28	10	3	6.000e-07
29	11	3	4.000e-07
30	11	3	4.000e-07
31	11	3	4.000e-07
32	12	3	3.000e-07
33	12	3	4.000e-07
34	12	3	4.000e-07
35	13	3	4.000e-07
36	13	3	4.000e-07
37	13	3	3.000e-07
38	14	3	4.000e-07
39	14	3	4.000e-07
40	14	3	4.000e-07
41	15	3	4.000e-07
42	15	3	4.000e-07
43	15	3	3.000e-07
44	16	4	4.000e-07
45	16	4	3.000e-07
46	16	4	3.000e-07
47	16	4	4.000e-07
48	17	3	3.000e-07
49	17	3	3.000e-07
50	17	3	4.000e-07
55	19	4	1.300e-06
60	22	4	5.000e-07
65	24	4	4.000e-07
70	26	5	4.000e-07
75	27	5	7.000e-07
80	28	5	5.000e-07
85	29	6	5.000e-07
90	30	6	6.000e-07
95	32	6	5.000e-07
100	35	4	4.000e-07
110	39	6	2.300e-06
120	42	5	5.000e-07
130	44	6	5.000e-07
140	46	7	4.000e-07
150	48	7	6.000e-07
160	53	10	5.000e-07
170	58	7	4.000e-07
180	61	5	4.000e-07
190	63	6	4.000e-07
200	65	6	4.000e-07
220	72	13	1.300e-06
240	80	10	6.000e-07
260	84	8	6.000e-07
280	91	10	4.000e-07
300	98	10	4.000e-07
320	103	9	5.000e-07
340	113	11	5.000e-07
360	117	7	4.000e-07
380	122	8	5.000e-07
400	131	15	5.000e-07
420	136	8	1.200e-06
440	141	10	6.000e-07
460	148	10	5.000e-07
480	154	8	6.000e-07
500	160	11	6.000e-07
520	168	10	5.000e-07
540	173	8	5.000e-07
560	179	12	6.000e-07
580	186	8	4.000e-07
600	190	8	6.000e-07
620	197	12	6.000e-07
640	205	9	4.000e-07
660	209	8	5.000e-07
680	215	10	5.000e-07
700	223	11	5.000e-07
720	227	8	5.000e-07
740	237	11	4.000e-07
760	241	7	4.000e-07
780	245	8	5.000e-07
800	254	15	6.000e-07

PC3

Brute Force (Professor)			---
		plain recursion	
n	sol	count	cpu time
1	1	2	6.900e-06
2	2	3	1.000e-06
3	3	5	4.000e-07
4	3	8	4.000e-07
5	4	13	5.000e-07
6	4	22	6.000e-07
7	5	36	7.000e-07
8	5	60	9.000e-07
9	5	100	1.300e-06
10	6	167	1.800e-06
11	6	279	4.100e-06
12	6	465	3.900e-06
13	7	777	5.900e-06
14	7	1297	2.090e-05
15	7	2165	1.530e-05
16	7	3614	2.470e-05
17	8	6031	4.050e-05
18	8	10065	6.710e-05
19	8	16795	1.117e-04
20	8	28024	1.854e-04
21	9	46758	3.048e-04
22	9	78011	5.117e-04
23	9	129304	8.390e-04
24	9	214870	1.397e-03
25	9	357608	2.281e-03
26	10	594940	3.765e-03
27	10	990850	6.244e-03
28	10	1650517	1.059e-02
29	11	2749395	1.841e-02
30	11	4580952	3.115e-02
31	11	7632406	5.889e-02
32	12	12717296	9.556e-02
33	12	21190409	1.401e-01
34	12	35490924	2.467e-01
35	13	59016828	4.137e-01
36	13	98217631	6.957e-01
37	13	163720286	1.283e+00
38	14	272564243	2.088e+00
39	14	454112641	3.466e+00
40	14	756688615	5.858e+00
41	15	1267297736	9.787e+00
42	15	2107317118	1.632e+01
43	15	3506801191	2.663e+01
44	16	5845493836	4.812e+01
45	16	9731532417	7.689e+01
46	16	16213152360	1.231e+02
47	16	27016079686	2.052e+02
48	17	43222379171	3.331e+02
49	17	70231605982	5.066e+02
50	17	113447132278	8.595e+03

Brute Force Melhorada			----
		plain recursion	
n	sol	count	cpu time
1	1	2	7.500e-06
2	2	3	5.000e-07
3	3	5	4.000e-07
4	3	5	4.000e-07
5	4	7	4.000e-07
6	4	8	4.000e-07
7	5	10	4.000e-07
8	5	12	4.000e-07
9	5	10	4.000e-07
10	6	13	5.000e-07
11	6	15	5.400e-06
12	6	13	4.000e-07
13	7	17	5.000e-07
14	7	20	5.000e-07
15	7	20	5.000e-07
16	7	16	4.000e-07
17	8	20	4.000e-07
18	8	23	5.000e-07
19	8	24	6.000e-07
20	8	21	6.000e-07
21	9	25	6.700e-06
22	9	29	1.000e-06
23	9	28	8.000e-07
24	9	25	7.000e-07
25	9	19	6.000e-07
26	10	21	9.000e-07
27	10	23	6.000e-07
28	10	21	5.000e-07
29	11	23	6.000e-07
30	11	25	6.000e-07
31	11	23	6.000e-07
32	12	25	7.000e-07
33	12	27	6.000e-07

34	12	25	5.000e-07
35	13	27	8.000e-07
36	13	29	6.000e-07
37	13	27	5.000e-07
38	14	29	6.000e-07
39	14	31	5.000e-07
40	14	29	5.000e-07
41	15	32	6.000e-07
42	15	34	6.000e-07
43	15	32	5.000e-07
44	16	35	1.000e-06
45	16	37	7.000e-07
46	16	36	6.000e-07
47	16	33	6.000e-07
48	17	35	8.000e-07
49	17	36	7.000e-07
50	17	34	6.000e-07
55	20	40	4.400e-06
60	22	44	9.000e-07
65	25	51	7.000e-07
70	26	54	7.000e-07
75	28	61	9.000e-07
80	29	71	1.000e-06
85	30	73	1.000e-06
90	31	65	1.100e-06
95	34	71	1.000e-06
100	36	75	1.200e-06
110	41	88	4.000e-06
120	44	99	1.700e-06
130	46	109	1.600e-06
140	48	114	1.500e-06
150	50	117	1.600e-06
160	54	111	1.400e-06
170	59	121	1.300e-06
180	62	134	1.400e-06
190	64	138	1.500e-06
200	66	150	1.800e-06
220	74	151	5.500e-06
240	81	170	2.300e-06
260	85	176	2.200e-06
280	92	187	2.300e-06
300	100	206	2.400e-06
320	105	216	2.600e-06
340	114	235	2.600e-06
360	119	247	2.700e-06
380	124	259	7.100e-06
400	133	277	3.200e-06
420	139	298	7.800e-06
440	144	299	3.800e-06
460	152	315	3.800e-06
480	158	337	4.000e-06
500	164	339	4.100e-06
520	172	357	4.100e-06
540	177	387	4.700e-06
560	184	382	7.800e-06
580	190	400	4.600e-06
600	194	413	4.800e-06
620	200	415	4.800e-06
640	208	439	5.000e-06
660	212	442	4.900e-06
680	218	451	5.100e-06
700	226	474	5.300e-06
720	231	482	5.600e-06
740	240	502	5.500e-06
760	245	524	5.900e-06
780	248	517	6.800e-06
800	258	540	6.000e-06

While

		plain recursion	
n	sol	count	cpu time
1	1	1	5.400e-06
2	2	3	1.000e-06
3	3	5	3.000e-07
4	3	5	3.000e-07
5	4	7	3.000e-07
6	4	7	3.000e-07
7	5	9	3.000e-07
8	5	9	3.000e-07
9	5	9	3.000e-07
10	6	11	3.000e-07
11	6	11	2.300e-06
12	6	11	4.000e-07
13	7	13	4.000e-07
14	7	13	4.000e-07
15	7	13	4.000e-07
16	7	13	4.000e-07
17	8	15	3.000e-07
18	8	15	3.000e-07
19	8	15	3.000e-07
20	8	15	4.000e-07
21	9	17	1.900e-06
22	9	17	6.000e-07
23	9	17	4.000e-07
24	9	17	5.000e-07
25	9	17	4.000e-07
26	10	19	5.000e-07

```
 27   10               19 4.000e-07
 28   10               19 4.000e-07
 29   11               21 4.000e-07
 30   11               21 3.000e-07
 31   11               21 4.000e-07
 32   12               23 4.000e-07
 33   12               23 3.000e-07
 34   12               23 4.000e-07
 35   13               25 4.000e-07
 36   13               25 4.000e-07
 37   13               25 4.000e-07
 38   14               27 4.000e-07
 39   14               27 4.000e-07
 40   14               27 4.000e-07
 41   15               29 5.000e-07
 42   15               29 4.000e-07
 43   15               29 4.000e-07
 44   16               31 4.000e-07
 45   16               31 4.000e-07
 46   16               31 4.000e-07
 47   16               31 4.000e-07
 48   17               33 5.000e-07
 49   17               33 4.000e-07
 50   17               33 4.000e-07
 55   20               39 2.600e-06
 60   22               43 1.100e-06
 65   25               49 6.000e-07
 70   26               51 5.000e-07
 75   28               55 6.000e-07
 80   29               57 7.000e-07
 85   30               59 6.000e-07
 90   31               61 6.000e-07
 95   34               67 6.000e-07
100   36               71 6.000e-07
110   41               81 3.700e-06
120   44               87 1.600e-06
130   46               91 1.000e-06
140   48               95 9.000e-07
150   50               99 9.000e-07
160   54              107 9.000e-07
170   59              117 9.000e-07
180   62              123 8.000e-07
190   64              127 9.000e-07
200   66              131 1.000e-06
220   74              147 3.900e-06
240   81              161 2.800e-06
260   85              169 1.400e-06
280   92              183 1.400e-06
300  100              199 1.300e-06
320  105              209 1.400e-06
340  114              227 1.400e-06
360  119              237 1.400e-06
380  124              247 1.600e-06
400  133              265 1.600e-06
420  139              277 7.100e-06
440  144              287 3.500e-06
460  152              303 2.200e-06
480  158              315 1.900e-06
500  164              327 2.000e-06
520  172              343 2.100e-06
540  177              353 2.600e-06
560  184              367 3.400e-06
580  190              379 4.100e-06
600  194              387 3.100e-06
620  200              399 2.400e-06
640  208              415 2.500e-06
660  212              423 2.500e-06
680  218              435 4.000e-06
700  226              451 3.900e-06
720  231              461 4.900e-06
740  240              479 2.800e-06
760  245              489 2.800e-06
780  248              495 2.900e-06
800  258              515 2.900e-06

While Dynamic ------------ --------- +
    |                plain recursion |
--- + --- ---------------- --------- +
  n | sol            count  cpu time |
--- + --- ---------------- --------- +
  1   1                1 9.900e-06
  2   2                2 5.000e-06
  3   3                2 6.000e-07
  4   3                2 5.000e-07
  5   4                2 5.000e-07
  6   4                2 4.000e-07
  7   5                3 5.000e-07
  8   5                3 5.000e-07
  9   5                3 4.000e-07
 10   6                3 5.000e-07
 11   6                3 2.600e-06
 12   6                3 5.000e-07
 13   7                4 6.000e-07
 14   7                4 6.000e-07
 15   7                4 5.000e-07
 16   7                4 6.000e-07
 17   8                4 5.000e-07
 18   8                4 5.000e-07
 19   8                4 5.000e-07
```

20	8	4	5.000e-07
21	9	5	3.400e-06
22	9	5	6.000e-07
23	9	5	6.000e-07
24	9	5	5.000e-07
25	9	5	6.000e-07
26	10	3	5.000e-07
27	10	3	5.000e-07
28	10	3	5.000e-07
29	11	3	5.000e-07
30	11	3	5.000e-07
31	11	3	5.000e-07
32	12	3	4.000e-07
33	12	3	5.000e-07
34	12	3	5.000e-07
35	13	3	4.000e-07
36	13	3	5.000e-07
37	13	3	4.000e-07
38	14	3	4.000e-07
39	14	3	4.000e-07
40	14	3	5.000e-07
41	15	3	5.000e-07
42	15	3	5.000e-07
43	15	3	5.000e-07
44	16	4	5.000e-07
45	16	4	5.000e-07
46	16	4	5.000e-07
47	16	4	5.000e-07
48	17	3	4.000e-07
49	17	3	5.000e-07
50	17	3	5.000e-07
55	20	4	2.500e-06
60	22	4	5.000e-07
65	25	4	5.000e-07
70	26	4	5.000e-07
75	28	5	6.000e-07
80	29	5	6.000e-07
85	30	6	7.000e-07
90	31	6	7.000e-07
95	34	4	4.000e-07
100	36	4	5.000e-07
110	41	6	1.900e-06
120	44	6	6.000e-07
130	46	6	6.000e-07
140	48	7	7.000e-07
150	50	7	6.000e-07
160	54	10	7.000e-07
170	59	7	6.000e-07
180	62	5	5.000e-07
190	64	6	6.000e-07
200	66	6	7.000e-07
220	74	14	2.900e-06
240	81	9	8.000e-07
260	85	8	8.000e-07
280	92	11	7.000e-07
300	100	9	6.000e-07
320	105	8	7.000e-07
340	114	11	5.000e-07
360	119	7	6.000e-07
380	124	8	7.000e-07
400	133	12	5.000e-07
420	139	8	4.000e-06
440	144	10	1.000e-06
460	152	11	6.000e-07
480	158	8	7.000e-07
500	164	11	7.000e-07
520	172	9	6.000e-07
540	177	8	7.000e-07
560	184	13	8.000e-07
580	190	8	6.000e-07
600	194	8	7.000e-07
620	200	11	7.000e-07
640	208	9	6.000e-07
660	212	8	7.000e-07
680	218	9	6.000e-07
700	226	9	6.000e-07
720	231	9	7.000e-07
740	240	10	5.000e-07
760	245	8	7.000e-07
780	248	8	7.000e-07
800	258	11	6.000e-07