universidade de aveiro

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Knight's tour

Pedro Escaleira, N.Mec. 88821

1. Índice

2. Introdução

Sendo este o segundo dos dois problemas apresentados pelo docente da disciplina de Algoritmos e Estruturas de Dados, aceitei o desafio de o completar.

Este é apresentado como encontrar o número de formas diferentes de um cavalo (peça de xadrez), inicialmente numa determinada posição dum tabuleiro de xadrez, realizar um percurso até a um ponto final, tendo em consideração as regras deste jogo. Este problema pode ter várias variantes onde, por exemplo, se alteram os tamanhos do tabuleiro ou se introduzem novas regras, como o cavalo ter obrigatoriamente de passar por todas as casas do tabuleiro. Estes dois exemplos irão ser explorados nos próximos pontos.

Neste relatório irão, como seria expectável, ser apresentados os vários resultados obtidos e as conclusões construídas a partir destes.

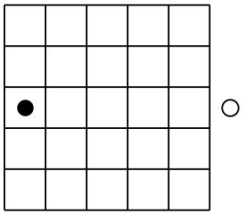
3. Técnicas algorítmicas utilizadas

Como seria esperado num trabalho para uma disciplina de algoritmos, foram usadas técnicas algorítmicas mais apropriadas do que a simples força bruta para resolver o problema. Delas, é de salientar a utilização do método de *backtracking*, lecionado nas aulas desta disciplina, que possibilita que sempre que o programa chegue a “um beco sem saída”, tenha a capacidade de voltar ao passo onde a decisão de seguir esse caminho foi realizada, e seguir por outro caminho. Desta forma, foi possível obter resultados em relativamente pouco tempo em todas as questões solucionadas com o programa feito.

4. Respostas obtidas para as questões propostas

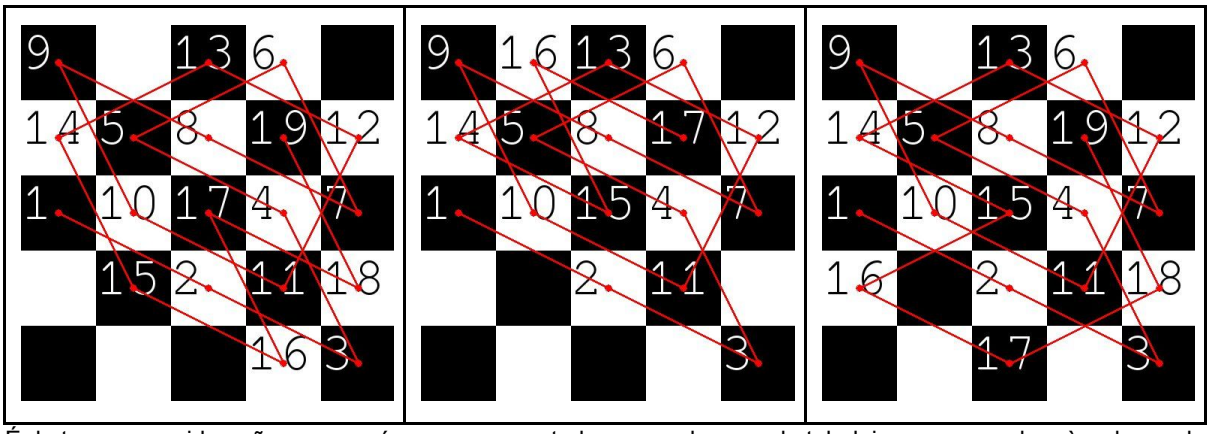
4.1. Número de formas de ir do ponto de partida até ao ponto de chegada num tabuleiro com as configurações apresentadas no enunciado

De acordo com o programa concebido, existem 252094 formas diferentes dum cavalo atravessar um tabuleiro 5x5 desde o ponto inicial até ao final, com a seguinte configuração, onde o ponto inicial é representado pelo ponto preenchido a preto e o ponto final o preenchido a branco:



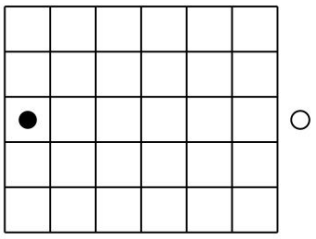
Tabuleiro obtido do enunciado
cedido pelo docente da disciplina

Alguns dos caminhos obtidos foram (só são apresentados os pontos dentro do tabuleiro, sendo que o último “salto de cavalo” corresponde ao caminho entre a casa de maior número para o ponto fora do tabuleiro):



É de ter em consideração que os números apresentados em cada casa do tabuleiro correspondem à ordem pela qual o cavalo se movimenta.

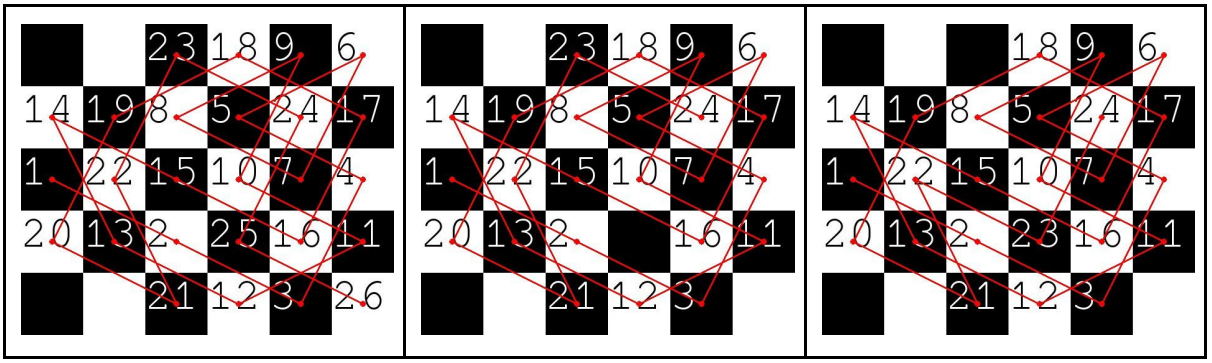
Já para o tabuleiro de configuração:



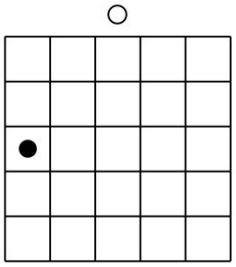
Tabuleiro obtido do enunciado

cedido pelo docente da disciplina

, foi obtido um total de 13193648 formas diferentes do cavalo viajar da posição inicial, a preto, para a posição final, a branco. Mais uma vez, foram também obtidos alguns exemplos gráficos destes caminhos:

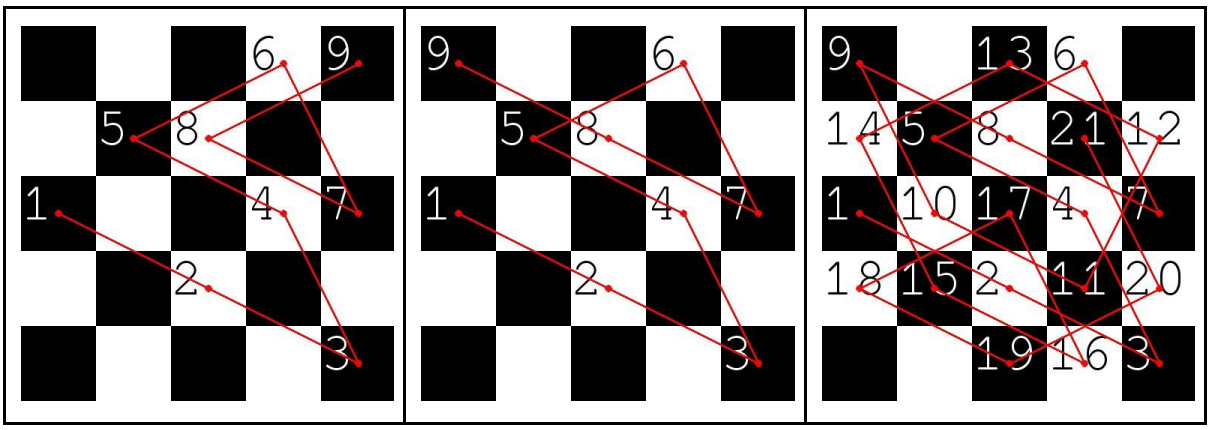


Para o tabuleiro 5x5 da forma apresentada a seguir, foram calculados 257805 trajetos diferentes, sendo este valor muito próximo do obtido para a primeira configuração apresentada:



Tabuleiro obtido do enunciado
cedido pelo docente da disciplina

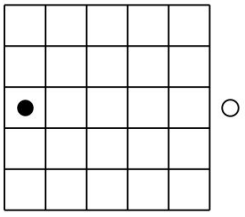
Três destas trajetórias são:



4.2. Número de formas de ir do ponto de partida até ao ponto de chegada num tabuleiro com as configurações apresentadas no enunciado, passando por todas as casas do tabuleiro

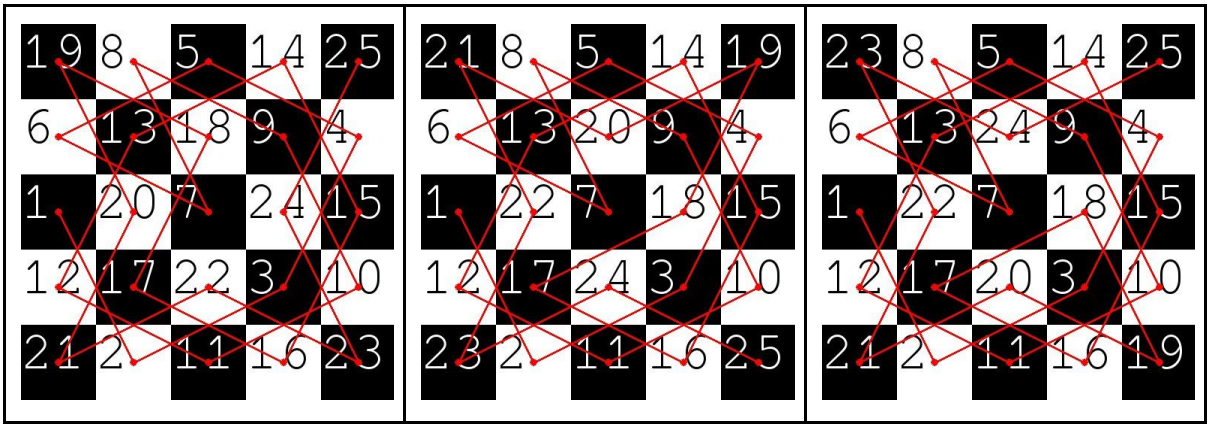
Nesta versão do problema é notável a grande diferença entre o número de caminhos obtidos e o número obtido na versão apresentada no ponto anterior, sendo nesta consideravelmente menor.

Na primeira configuração,

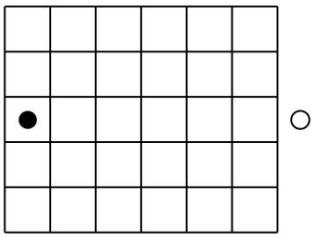


Tabuleiro obtido do enunciado
cedido pelo docente da disciplina

foram obtidas 28 formas diferentes do cavalo visitar todas as casas do tabuleiro, começando no disco "•" e acabando no "◦". Algumas destas rotas são:

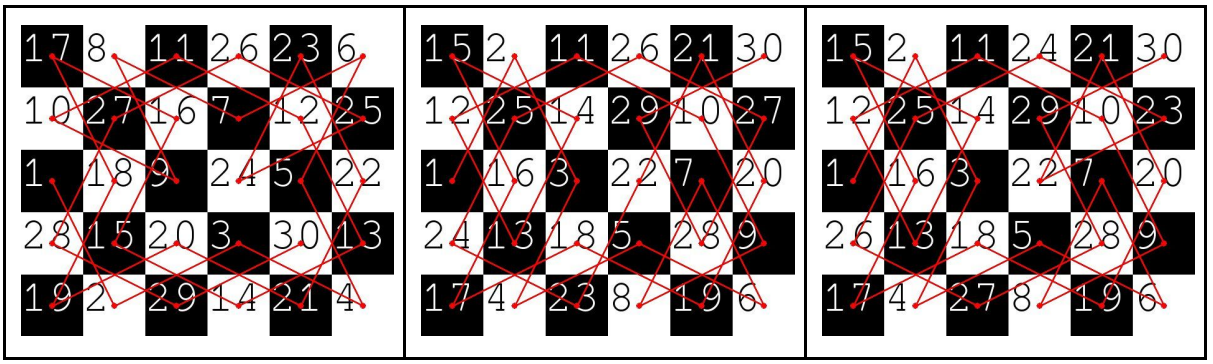


Como era presumível, a quantidade de itinerários que respeitam a regra introduzida neste ponto para a disposição:

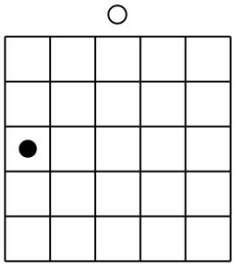


Tabuleiro obtido do enunciado
cedido pelo docente da disciplina

foi ligeiramente maior que para a anterior, já que este tabuleiro é maior. Neste caso o cavalo pode tomar 68 caminhos diferentes passando por todos os pontos do tabuleiro. Destes 68, temos como modelos os seguintes 3:

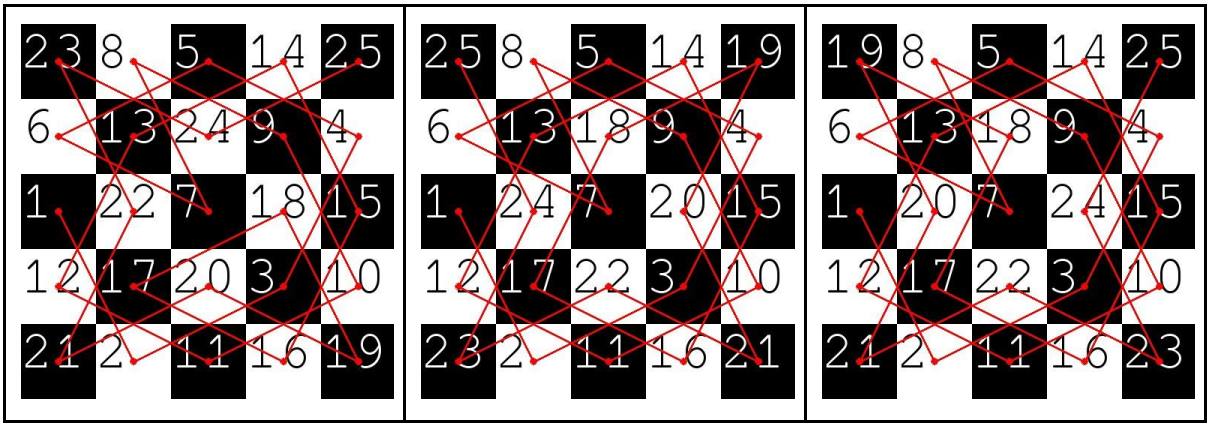


Tal como no ponto anterior, foram também obtidos os rumos diferentes (passando por todas as divisões do tabuleiro) que o cavalo pode tomar no próximo tabuleiro 5x5:



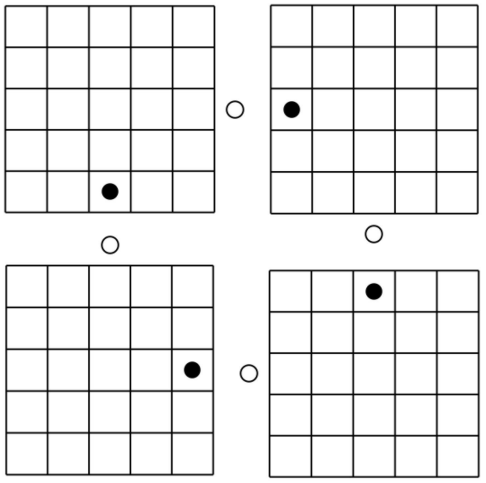
Tabuleiro obtido do enunciado
cedido pelo docente da disciplina

Desta feita, o número desses rumos foi 28, resultado este igual ao tabuleiro também 5x5 de configuração diferente exibido no início da apresentação dos resultados para este enigma, sendo que 3 deles são:



4.3. Descrição de uma maneira de percorrer um tabuleiro de dimensão genérica, passando por todas as casas

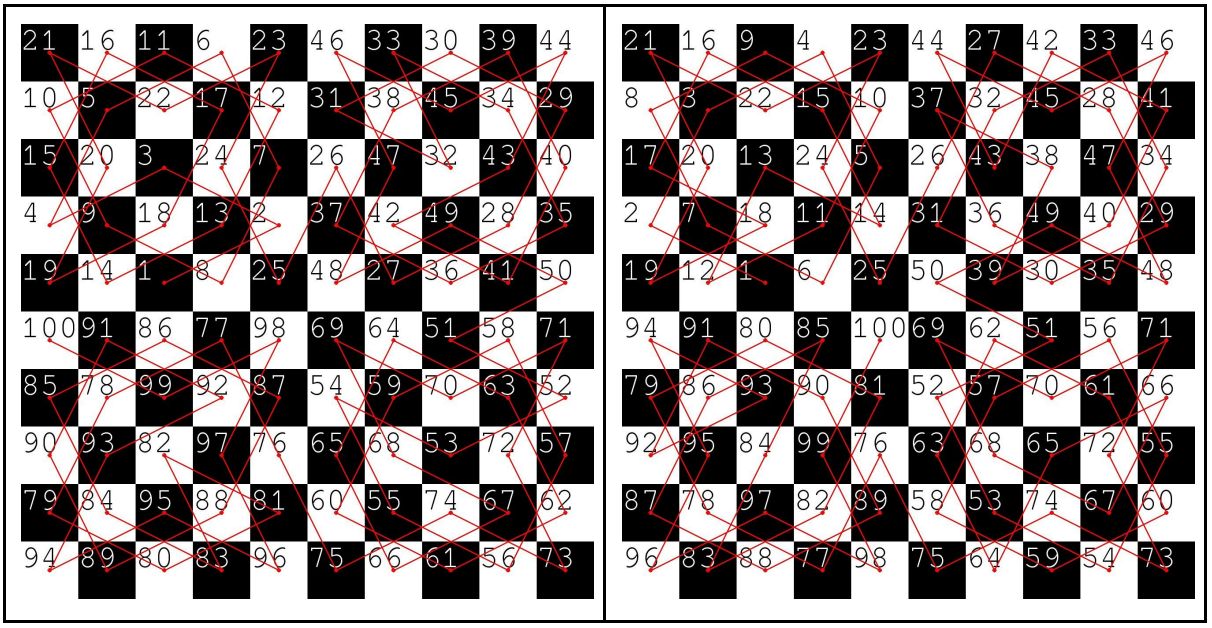
Sendo que não é requisitado que é necessário encontrar o número total de possibilidades diferentes de atravessar o tabuleiro, a técnica que aplicaria neste caso concreto seria a técnica de *divide and conquer*. Para isso, por exemplo num tabuleiro de tamanho 10x10, subdividi-lo-ia em 4 tabuleiros de dimensões 5x5, já que estes, possuindo uma dimensão muito menor, permitem obter rapidamente as trajetórias possíveis entre dois pontos, passando por todos os espaços do tabuleiro. Neste exemplo, poderia ser usada uma das duas configurações 5x5 acima descritas, aplicando rotações, da seguinte forma:



, onde os pontos "◦" corresponderiam ao ponto inicial no tabuleiro seguinte, a "•" .

Como se pode observar na figura acima, neste caso em concreto seria possível saltar da posição final para a inicial com um salto de cavalo. Além do mais, é facilmente perceptível que o número de caminhos obtidos na configuração apresentada é 28*28*28*28=614656, já que, como visto no ponto anterior, o número total de caminhos possíveis para esta configuração 5x5, em que o cavalo passa por todas as divisões do tabuleiro, é 28. Sendo assim, compreende-se que o número total de formas diferentes dum cavalo percorrer um tabuleiro 10x10 resulta num número um tanto "astronómico", impossível de obter em tempo aceitável num computador pessoal atual.

Usando os resultados obtidos para as 4 configurações, foi possível obter exemplos de alguns percursos que podem ser realizados pelo cavalo:



Para um tabuleiro de tamanho ainda maior, bastaria gerar mais configurações de tamanho inferior e, como demonstrado neste protótipo, juntá-las todas num “mega tabuleiro”.

5. Código usado

5.1. problem2.c

Código usado para obter o número total de caminhos possíveis apresentados no relatório.

```c
#include<stdio.h>
#include<stdlib.h>

#define SAVE_DOC 1                                                                  //boolean value: if true (>0), the horse's routes are saved to a file; if false (<=0), nothing is saved

const unsigned long MAX_SAVE = 100;                                                 //max number of grids to save (because, in some cases, there are a lot of routes, wich resuts in a big data file if all of them are saved)
unsigned long num_paths_saved = 0;                                                  //current number of saved paths
FILE *file;

const short MOVEMENT_1 = 2;
const short MOVEMENT_2 = 1;

unsigned int grid_size[2];
unsigned int **grid;

int horse_position[2];
int target_position[2];


/* Recursive function, where are calculated all the horse's possible moviments for a giver chess board (grid)
*
* Arguments:
*    -> short move_x: horse's movement in grid's x axes
*    -> short move_y: horse's movement in grid's y axes
*    -> int step: current horse's step (useful to store in grid's array)
*    -> short pass_throw_every_sqare: boolean value -> if true (>0), the horse is forced to visit every sqare of chess board; if false (<=0), the horse isn't forced to do that
* Return:
*    -> number of all different possible routes
*/
unsigned long long move(short move_x, short move_y, int step, short pass_throw_every_sqare){

  int new_position[] = {horse_position[0] + move_x,                                 //calc of new possible horse's position in the board
                        horse_position[1] + move_y};

  if(new_position[0] == target_position[0] &&                                       //verify if the horse reached the target (if so, the number of possible routes is increased by one and if SAVE_DOC > 0, the grid with that route is stored in the data file)
     new_position[1] == target_position[1]){

     if(!pass_throw_every_sqare ||                                                  //verify if the grid is completely filled (because if pass_throw_every_sqare > 0, the horse must visit all board's squares)
        grid[horse_position[0]][horse_position[1]] == grid_size[0]*grid_size[1]){

        if(SAVE_DOC && (num_paths_saved < MAX_SAVE)){
           num_paths_saved++;
           for(int n1 = 0; n1 < grid_size[0]; n1++){
              for(int n2 = 0; n2 < grid_size[1]; n2++)
                 fprintf(file, "%d\t", grid[n1][n2]);
              fprintf(file, "\n");
           }
           fprintf(file, "\n");
        }

        return 1;
     }

  }

  if(new_position[0] < 0 || new_position[1] < 0 ||                                  //verify if horse don't go outside the grid
     new_position[0] > (grid_size[0] - 1) || new_position[1] > (grid_size[1] - 1) ||  //verify if horse don't go outside the grid
     grid[new_position[0]][new_position[1]] != 0)                                   //verify if the horse already has been in that position
     return 0;


  unsigned long long num_success = 0;                                               //variable where is stored all the horse's routes for all the recursive calls of this function

  int last_horse_position[] = {horse_position[0], horse_position[1]};               //for backtracking

  horse_position[0] = new_position[0];                                              //update horse's x position
  horse_position[1] = new_position[1];                                              //update horse's y position

  grid[horse_position[0]][horse_position[1]] = ++step;                              //save new horse's position in the grid's array (with current step)
  int last_horse_position_marked[] = {horse_position[0], horse_position[1]};        //for backtracking

  /*************** all possible movements ***************/
```

```
  num_success += move(MOVEMENT_1, MOVEMENT_2, step, pass_throw_every_sqare);
  num_success += move(MOVEMENT_1, -MOVEMENT_2, step, pass_throw_every_sqare);
  num_success += move(-MOVEMENT_1, -MOVEMENT_2, step, pass_throw_every_sqare);
  num_success += move(-MOVEMENT_1, MOVEMENT_2, step, pass_throw_every_sqare);

  num_success += move(MOVEMENT_2, MOVEMENT_1, step, pass_throw_every_sqare);
  num_success += move(MOVEMENT_2, -MOVEMENT_1, step, pass_throw_every_sqare);
  num_success += move(-MOVEMENT_2, -MOVEMENT_1, step, pass_throw_every_sqare);
  num_success += move(-MOVEMENT_2, MOVEMENT_1, step, pass_throw_every_sqare);


  /*************** backtracking ***************/
  horse_position[0] = last_horse_position[0];
  horse_position[1] = last_horse_position[1];
  grid[last_horse_position_marked[0]][last_horse_position_marked[1]] = 0;


  return num_success;

}

int main(){
  /*************** initializations ***************/
  short PASS_THROW_EVERY_SQUARE = 0;                                                    //boolean value -> if true (>0), the
horse is forced to visit every sqare of chess board; if false (<=0), the horse isn't forced to do that

  grid_size[0] = 5;                                                                     //width of chess board in number of
squares
  grid_size[1] = 5;                                                                     //height of chess board in number of
squares

  grid = (unsigned int **)malloc(grid_size[0]*sizeof(unsigned int *));                  //chess board initialized with zeros in
all squares
  for(int num = 0; num < grid_size[0]; num++)
     grid[num] = (unsigned int *)calloc(grid_size[1], sizeof(unsigned int));

  horse_position[0] = 2;                                                                //x coordinate of horse (initialized with
its initial position), according to board's size in squares
  horse_position[1] = grid_size[1] - 1;                                                 //y coordinate of horse (initialized with
its initial position), according to board's size in squares

  target_position[0] = grid_size[0];                                                    //x coordinate of target (initialized
with its initial position), according to board's size in squares
  target_position[1] = 2;                                                               //y coordinate of target (initialized
with its initial position), according to board's size in squares


  /*************** problem's resolution ***************/
  if(SAVE_DOC){                                                                         //open the file where is to be stored the
data about the horse's route
     char *file_name = malloc(10);
     sprintf(file_name, "%dx%d.txt", grid_size[0], grid_size[1]);
     file = fopen(file_name, "w");
     free(file_name);
  }

  printf("%llu\n", move(0, 0, 0, PASS_THROW_EVERY_SQUARE));                             //start the horse's course, with no
movement, because the initial position of the horse is not stored in the grid

  if(SAVE_DOC)                                                                          //close the file where is to be stored
the data about the horse's route
     fclose(file);

}
```

5.2. makeBoard.py

Código feito em *python* (usando a biblioteca *pygame*) com o intuito de gerar os tabuleiros de xadrez visualizados neste relatório.

```
import pygame

########################### Constants ###########################
FILE_NAME = "10x10_last.txt"
IMAGE_NAME = "10x10_last_3"
BOARD_NUM = 20

PRINT_NUMBERS = True
SIZE_X = 10
SIZE_Y = 10
SQUARE_SIZE = 100
SQUARE_COLOR = [(0, 0, 0), (255, 255, 255)]
LINE_COLOR = (255, 0, 0)
LINE_WIDTH = 3
CIRCLE_WIDTH = 5

HORSE_STEPS = []
HORSE_POSITIONS_SCREEN = [None]*(SIZE_X*SIZE_Y)


########################### Read data file ###########################
file = open(FILE_NAME, "r")
lines = file.readlines()
for i in range(SIZE_X):
  line = lines[BOARD_NUM*(SIZE_X + 1) + i].split('\t')
  del line[len(line) - 1]
  HORSE_STEPS.append(list(map(int, line)))
file.close()

for n1 in range(SIZE_X):
  for n2 in range(SIZE_Y):
     HORSE_POSITIONS_SCREEN[HORSE_STEPS[n1][n2] - 1] = (n1*SQUARE_SIZE + SQUARE_SIZE//2, n2*SQUARE_SIZE + SQUARE_SIZE//2)


########################### Draw the requested board ###########################
pygame.init()
font = pygame.font.SysFont("couriernew", 72)
clock = pygame.time.Clock()
screen = pygame.display.set_mode((SQUARE_SIZE*SIZE_X, SQUARE_SIZE*SIZE_Y))

running = True
while running:
  for event in pygame.event.get():
       if event.type == pygame.QUIT:
           pygame.quit()
           running = False

  for n1 in range(SIZE_X):
     for n2 in range(SIZE_Y):
        square_color = SQUARE_COLOR[(n1 + n2)%len(SQUARE_COLOR)]
        pygame.draw.rect(screen, square_color, pygame.Rect(n1*SQUARE_SIZE, n2*SQUARE_SIZE, SQUARE_SIZE, SQUARE_SIZE))

        if PRINT_NUMBERS and HORSE_STEPS[n1][n2] != 0:
           font_color = SQUARE_COLOR[(n1 + n2 + 1)%len(SQUARE_COLOR)]
           text = font.render(str(HORSE_STEPS[n1][n2]), True, font_color)
           screen.blit(text, (n1*SQUARE_SIZE, n2*SQUARE_SIZE))

  for i in range(len(HORSE_POSITIONS_SCREEN) - 1):
     if HORSE_POSITIONS_SCREEN[i + 1] == None:
        break;
     pygame.draw.line(screen, LINE_COLOR, HORSE_POSITIONS_SCREEN[i], HORSE_POSITIONS_SCREEN[i + 1], LINE_WIDTH)
     pygame.draw.circle(screen, LINE_COLOR, HORSE_POSITIONS_SCREEN[i], CIRCLE_WIDTH, CIRCLE_WIDTH)
     pygame.draw.circle(screen, LINE_COLOR, HORSE_POSITIONS_SCREEN[i + 1], CIRCLE_WIDTH, CIRCLE_WIDTH)

  pygame.display.flip()

  #save the chess board in an jpeg image and exit the program
  pygame.image.save(screen, IMAGE_NAME + ".jpeg")
  exit()
```

5.3. makeBoardDivideConquer.py

Código feito em *python*, com o intuito de gerar um documento com algumas das rotas realizadas pelo cavalo num tabuleiro 10x10. Para isso, são lidos os dados de 4 documentos, cada um contendo os caminhos realizados pelo cavalo para cada uma das configurações apresentadas no ponto **4.3**.

```
SIZE_X = 5
SIZE_Y = 5
NUMBER_BOARDS = 28

files = [open("5x5_last_top_left.txt", "r"),
        open("5x5_last_bottom_left.txt", "r"),
        open("5x5_last_top_right.txt", "r"),
        open("5x5_last_bottom_right.txt", "r")]
file_write = open("10x10_last.txt", "w")

lines = []
for i in range(len(files)):
  lines.append(files[i].readlines())

all_boards = []
for board_num in range(NUMBER_BOARDS):
  all_boards.append([]);
  for n in range(2):
     for i in range(SIZE_X):
        line = lines[(n%2)*2][board_num*(SIZE_X + 1) + i].split('\t')
        del line[len(line) - 1]

        line += lines[(n%2)*2 + 1][board_num*(SIZE_X + 1) + i].split('\t')
        del line[len(line) - 1]

        all_boards[board_num].append(list(map(int, line)))

for i in range(len(files)):
  files[i].close()

for i in all_boards:
  file_write.write(str(i).replace(", ", "\t")
                         .replace("]\t[", "\t\n")
                          .replace("[[", "")
                          .replace("]]", "\t"))
  file_write.write("\n\n")
file_write.close()
```