

Algoritmos e Estrutura de Dados

Relatório do trabalho sobre o Traveling Salesman Problem

11/11/2018

Bernardo Rodrigues nº Mec: 88835 (50%)

Vinícius Benite Ribeiro nº Mec: 82773 (50%)

**Introdução**

Para a disciplina de Algoritmos e Estrutura de Dados foi-nos proposto a resolução do “Travelling Salesman Problem” um problema da teoria de grafos que envolve o cálculo do melhor e pior caminho que um viajante pode percorrer quando quer viajar por um determinado número de cidades, regressando de volta à cidade inicial.

**Travel Salesman Problem: Brute Force**

Inicialmente para resolver este problema primeiramente, usamos o método de “brute force”, ou seja, calcula-se todas as permutações para n cidades, mantendo constante a cidade inicial, visto que vamos regressar à cidade inicial no final do tour, e em cada permutação calcula a distância necessária para percorrer as cidades.

No final, o programa compara a distância total calculada com a distância mínima e máxima que tem registadas e atualiza conforme necessário, guardando também o percurso para a distância.

O programa vai repetindo os cálculos começando por realizar tours com apenas 3 cidades, até ao máximo de cidades (18), começando sempre os cálculos de novo cada vez que é adicionada uma nova cidade.

Concluindo, o programa dá a indicação do melhor e pior caminho para cada número de cidades percorridas e o respetivo percurso para o atingir, além do tempo gasto para realizar os cálculos.

Como foi preciso calcular todos os possíveis percursos foram necessários o cálculo de n! caminhos diferentes, sendo n o número de cidades por onde o viajante passou.

Isto dá ao problema uma complexidade de O(n!) o que significa que para um número elevado de cidades o programa teria um tempo de execução enorme.

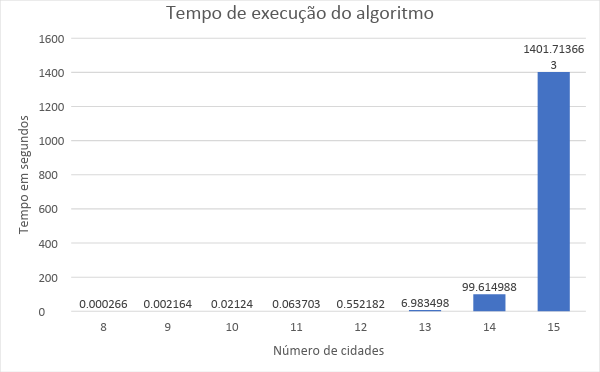
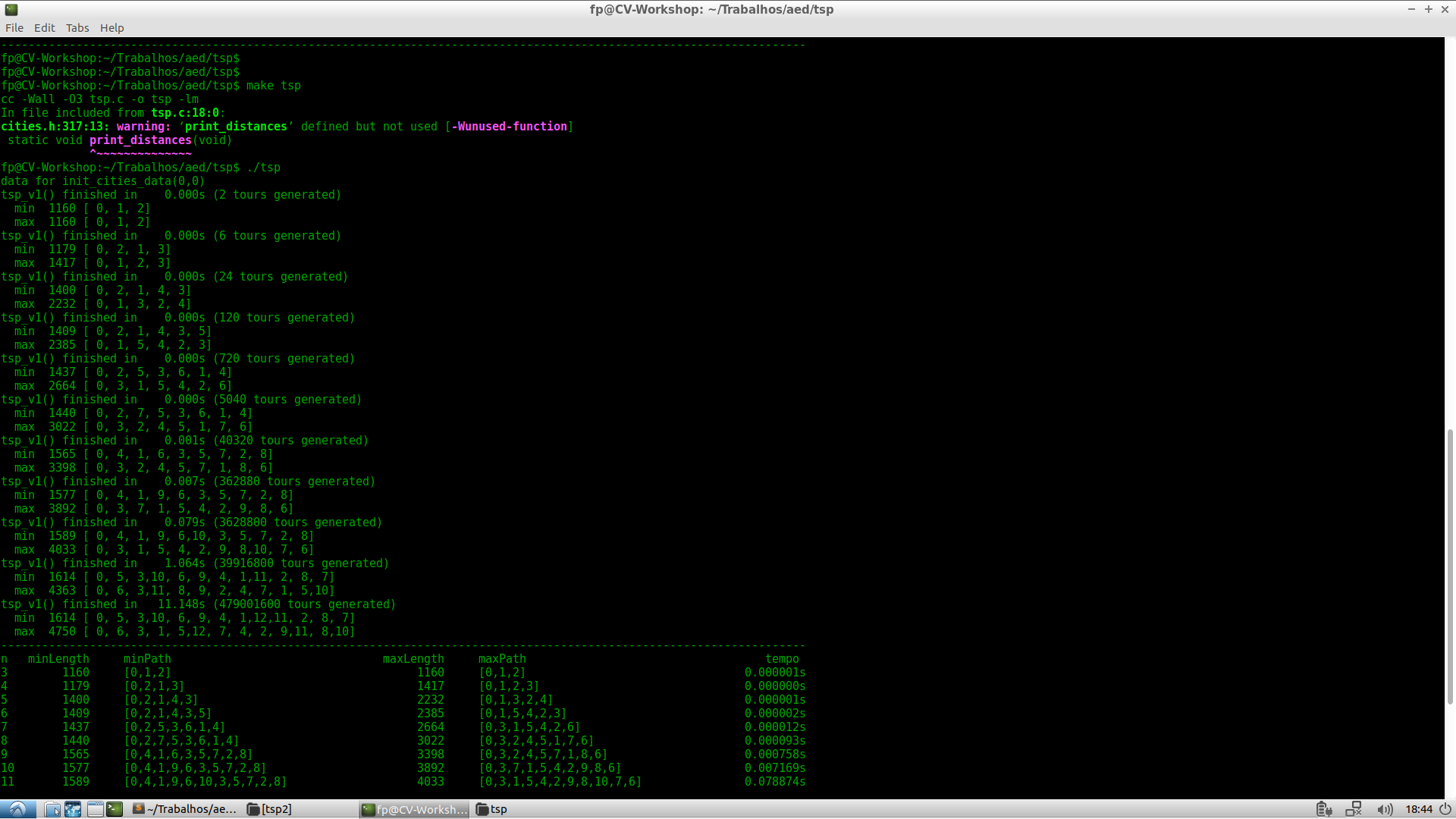
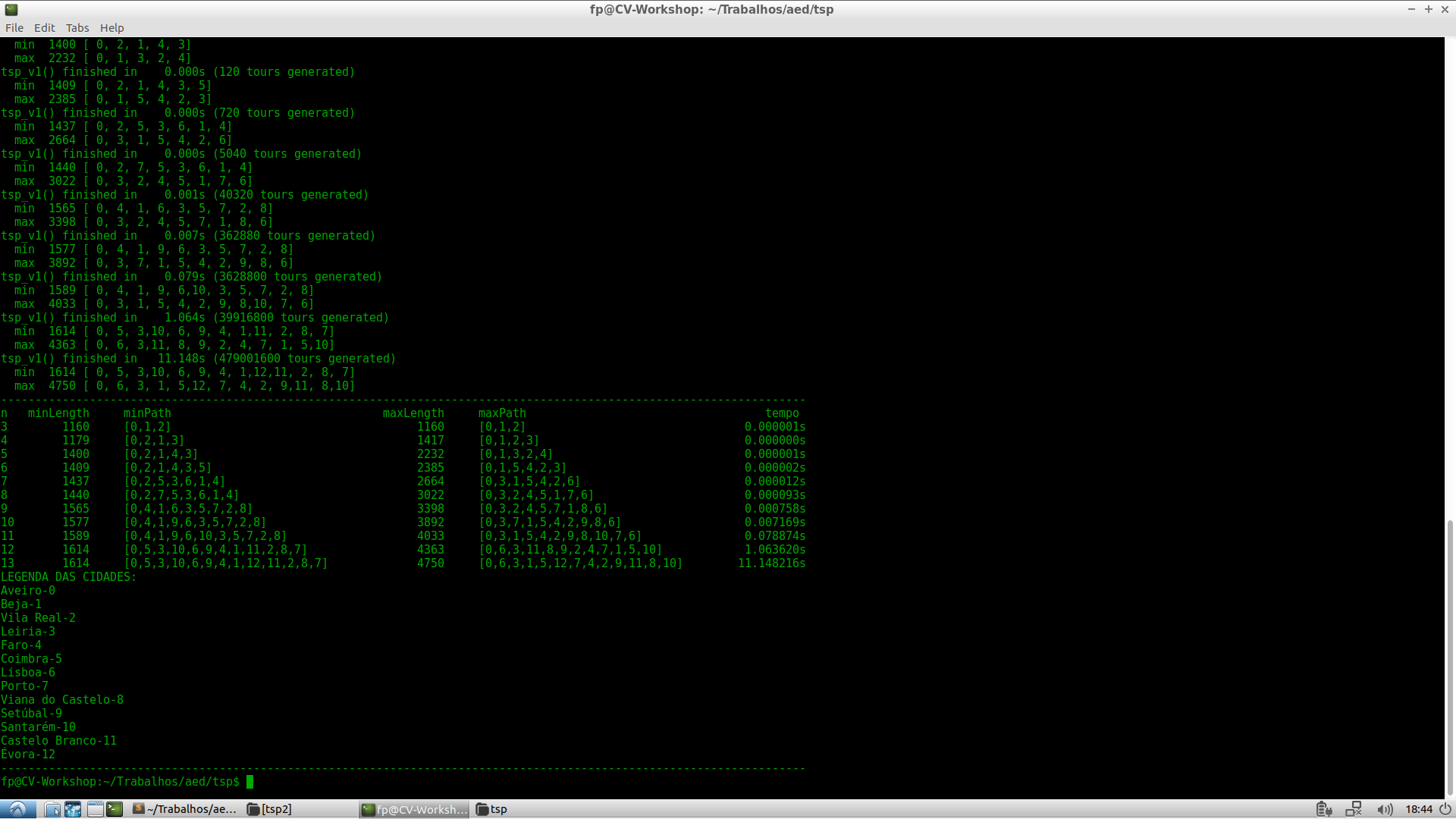
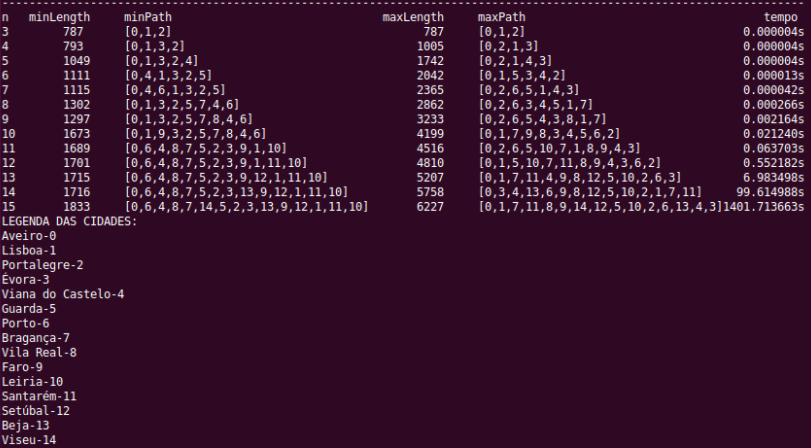


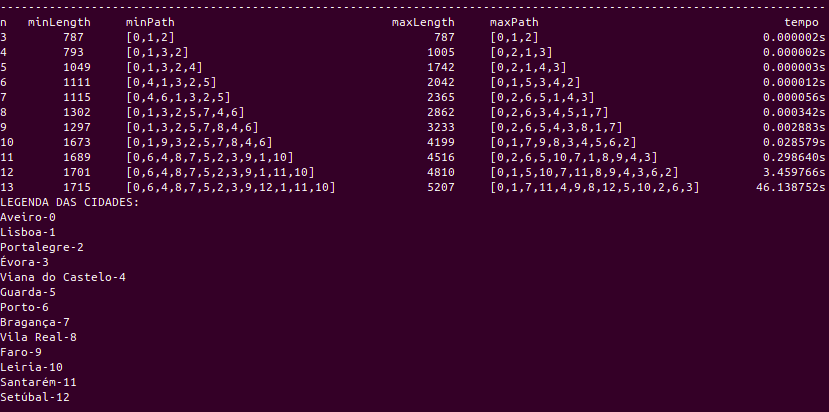
Gráfico do tempo de cálculo por número de cidades. Para n < 8 o tempo é insignificante

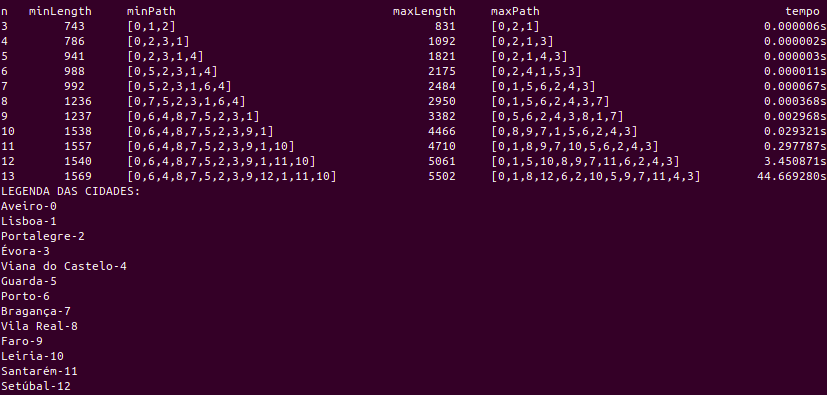
Resultado do programa para nº Mec: 88835



Resultado do programa para nº Mec: 82773

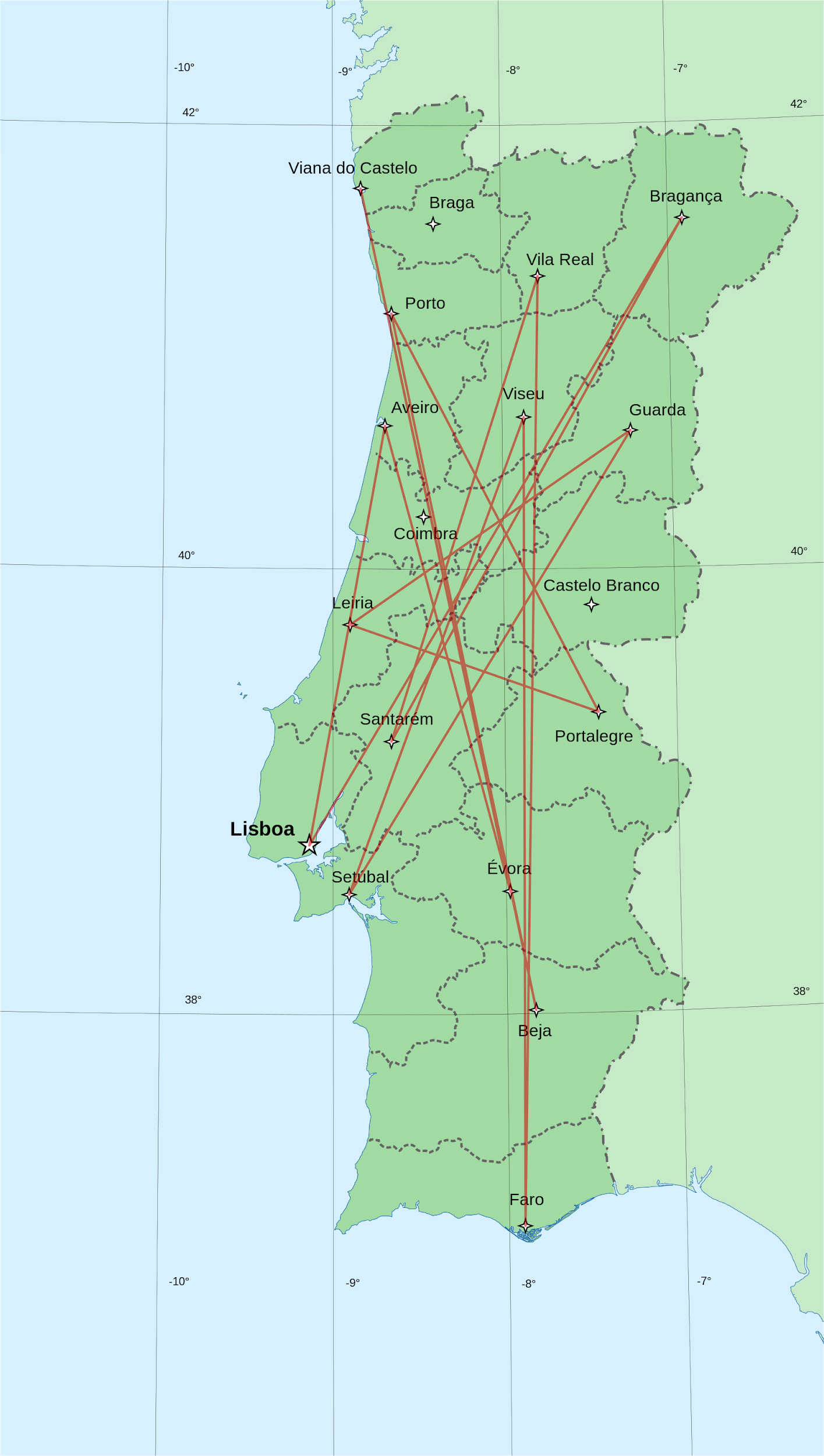
Resultados apresentando a diferença entre special = 0 e special = 1 (para n = 12):

Special = 0

Special = 1



Mapa do melhor tour



Mapa do pior tour

Esta abordagem apesar de simples, não é eficiente pois demora demasiado tempo, nomeadamente porque repete várias vezes cálculos que já foram previamente realizados, por exemplo: quando calcula a viagem pelas cidades 0-1-2-3-4-5, vai repetir desnecessariamente parte dos cálculos para a combinação das cidades (0-1-2-3) (5-4) e é esta repetição de cálculos que aumenta imenso o tempo do programa.

**Travel Salesman Problem: Dynamic Programing**

Para tornar o programa mais eficaz, recorremos ao uso de dynamic programing e de memoization. Deste modo o programa calcula recursivamente as distâncias percorridas, dividindo o problema em problemas menores, ou seja, para calcular a rota para 6 cidades o programa vai primeiro calcular as rotas para 5, que por sua vez vai calcular as rotas para 4 cidades, chegando ao mínimo de 3 cidades.

Adicionalmente a cada posição é atribuída uma máscara, isto é por cada posição visitada é posto um bit de uma variável a 1, por exemplo quando temos 4 cidades e é visitada a cidade 0 (sempre a primeira cidade a ser visitada) temos a máscara 0001, se a seguir visitarmos a cidade 2 o terceiro bit da máscara (a contar da direita) é posto a 1, ficando a máscara 0101, sendo o percurso feito até a momento 0-2-…

O programa inicia o tour na cidade 0 e a máscara a 1, vai chamando a função recursivamente ponde sempre um bit da máscara a 1, até que a máscara tenha o valor 2n-1 que é o valor máximo que pode ter quando todos os bits estão a 1 (exemplificando quando temos 4 cidades a máscara cheia ficaria 11112 ou seja 24-1 = 15 ), e retorna a distância da cidade onde estamos correntemente à cidade inicial.

O programa vai repetindo-se e guarda os valores dos sub-tours num array de array (memoization). O array vai ter capacidade [2n][n] onde será guardada a distância de percursos previamente percorridos, tais percursos serão guardados na posição do array [máscara][cidade atual], deste não estaremos que estar a repetir cálculos pois já sabemos a distância.

Exemplificado: Se tivermos 5 cidades e calcularmos a distância de 3-2-1, guardamos o valor na posição [001111] [3] (nota: o primeiro bit vai estar sempre a 1) assim quando calcularmos o percurso 4-3-2-1 teremos apenas de somar a distância de 4 ao valor guardado na posição [001111] [3], o que vai reduzir imenso o tempo de execução do programa, visto que o número de cálculos a ser realizado vai diminuir. Porém para guardar os valores dos percursos vai ser preciso usar alguma memória do computador.

**Código do programa TSP por força bruta**

//

// Bernardo Rodrigues

// n\_Mec = 88835

// Vinicius Benite Ribeiro

// n\_Mec = 82773

// ...

//

// AED, 2018/2019

//

// solution of the traveling salesman problem

//

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/time.h>

#include <sys/resource.h>

#include "cities.h"

#include "../P01/elapsed\_time.h"

//

// record best solutions

//

/\*Struct criado para guardar dados\*/

/\*guarda a distancia minima,distancia maxima,o melhor caminho e o pior caminho\*/

struct guardar\_dados

{

int min\_distance;

int max\_distance;

int max\_tour\_str[max\_n\_cities + 1];

int min\_tour\_str[max\_n\_cities + 1];

double tour\_time;

};

static struct guardar\_dados dados[max\_n\_cities + 1];

/\*Array de structs é criado para guardar as informações de n cidades

ex: dados[3] guarda informações de um percurso com 3 cidades

/ dados[4] informações de um percurso de 4 cidades

No final dos calculos dos percursos é usado o struct para

ver os dados\*/

int p = 0; /\*Esta variavel é usada para criar 2 vetores com o tamanho do maior percurso\*/

static int min\_length, max\_length;

static int min\_tour[max\_n\_cities + 1], max\_tour[max\_n\_cities + 1];

static long n\_tours;

static int debug = 0; /\*Imprime todos os caminhos e a distancia AUMENTA O TEMPO DO PROGRAMA\*/

//

// first solution (brute force, distance computed at the end, compute best and worst tours)

//

void tsp\_v1(int n, int m, int \*a)

{

int i, t;

int distance = 0;

if (m < n - 1)

for (i = m; i < n; i++)

{

t = a[m];

a[m] = a[i];

a[i] = t;

tsp\_v1(n, m + 1, a);

t = a[m];

a[m] = a[i];

a[i] = t;

}

else

{ // visit permutation

n\_tours++;

// modify the following code to do your stuff

for (i = 0; i < n; i++)

{

if (debug == 1)

{

printf("%s-", cities[a[i]].name); /\*Imprime o nome da cidade\*/

printf("%d%s", a[i], (i == n - 1) ? "\n" : " "); /\*Imprime o numero da cidade\*/

}

if (i + 1 == n)

distance += cities[a[i]].distance[a[0]]; /\*No calculo da distancia quando está na ultima cidade, calcula a distancia da cidade em que está para a cidade em que começou o tour; CITIES[ultima\_cidade\_do\_tour].DISTANCE[cidade\_onde\_começou\_o\_tour] \*/

else

distance += cities[a[i]].distance[a[i + 1]]; /\*Por cada cidade que passa, calcula-se a distancia da proxima; CITIES[em\_que\_se\_encontra].DISTANCE[proxima\_cidade\_a\_ser\_adicionada\_no\_tour]\*/

}

/\*No final de cada tour o programa compara a distancia com a distancia maxima e minima\*/

if (debug == 1)

printf("Distancia: %d\n", distance); /\*Imprime a distancia\*/

/\*calculos para a distancia maxima\*/

if (distance > max\_length)

{

max\_length = distance; /\*Update da distancia maxima\*/

memcpy(max\_tour, a, sizeof(max\_tour)); /\*esta função copia o conteudo de a para o conteudo do max\_tour, ficando assim o array max\_tour com o caminho mais longo\*/

}

/\*Calculos para a distancia minima\*/

if (distance < min\_length)

{

min\_length = distance; /\*Update da distancia minima\*/

memcpy(min\_tour, a, sizeof(min\_tour)); /\*copia o conteudo de a para min\_tour\*/

}

}

}

//

// main program

//

int main(int argc, char \*\*argv)

{

int n\_mec, special, n, i, a[max\_n\_cities];

char file\_name[32];

double dt1;

n\_mec = 82773; // CHANGE THIS!

special = 1;

init\_cities\_data(n\_mec, special);

printf("data for init\_cities\_data(%d,%d)\n", n\_mec, special);

fflush(stdout);

#if 0

print\_distances();

#endif

for (n = 3; n <= (max\_n\_cities - 5); n++) /\*AUMENTAR OU DIMINUIR--MAXIMO 18 CIDADES\*/

{

//

// try tsp\_v1

//

dt1 = -1.0;

if (n <= 16)

{

(void)elapsed\_time();

for (i = 0; i < n; i++)

a[i] = i;

min\_length = 1000000000;

max\_length = 0;

n\_tours = 0l;

tsp\_v1(n, 1, a); // no need to change the starting city, as we are making a tour

dt1 = elapsed\_time();

dados[n].tour\_time = dt1; /\*Regista o tempo de cada tour no struct dados\*/

printf("tsp\_v1() finished in %8.3fs (%ld tours generated)\n", dt1, n\_tours);

printf(" min %5d [", min\_length);

for (i = 0; i < n; i++)

printf("%2d%s", min\_tour[i], (i == n - 1) ? "]\n" : ",");

memcpy(dados[n].min\_tour\_str, min\_tour, sizeof(min\_tour)); /\*Regista o menor tour\*/

dados[n].min\_distance = min\_length; /\*Update da distancia minima para um tour com n cidades\*/

printf(" max %5d [", max\_length);

for (i = 0; i < n; i++)

printf("%2d%s", max\_tour[i], (i == n - 1) ? "]\n" : ",");

memcpy(dados[n].max\_tour\_str, max\_tour, sizeof(max\_tour)); /\*Regista o maior tour\*/

dados[n].max\_distance = max\_length; /\*Update da distancia maxima para um tour com n cidades\*/

fflush(stdout);

if (argc == 2 && strcmp(argv[1], "-f") == 0)

{

min\_tour[n] = -1;

sprintf(file\_name, "min\_%02d.svg", n);

make\_map(file\_name, min\_tour);

max\_tour[n] = -1;

sprintf(file\_name, "max\_%02d.svg", n);

make\_map(file\_name, max\_tour);

}

}

}

/\*No final do programa é impresso uma tabela com as informações dos percursos\*/

printf("----------------------------------------------------------------------------------------------------------------------");

printf("\n%s%12s%12s%40s%12s%40s\n", "n", "minLength", "minPath", "maxLength", "maxPath", "tempo");

int formatar = 40; /\*Esta variavel é usada para formatar os printf para facilitar a leitura das informações da tabela\*/

for (n = 3; n <= max\_n\_cities - 5; n++)

{

if (dados[n].min\_distance == 0)

{

p = n; /\*Esta variavel é usada mais à frente no programa para definir o tamanho de 2 vetores\*/

break; /\*Se dados[n].min\_distance == 0 significa que não feito um percurso para n cidades\*/

}

printf("%-9d", n); /\*Imprime o numero de cidades de um percurso\*/

printf("%-9d[", dados[n].min\_distance); /\*Imprime a distancia minima\*/

for (i = 0; i < n; i++)

{

printf("%d%s", dados[n].min\_tour\_str[i], (i == n - 1) ? "]" : ","); /\*Imprime o melhor caminho\*/

}

printf("%\*d%-5s[", formatar, dados[n].max\_distance, ""); /\*Imprime a distancia maxima\*/

for (i = 0; i < n; i++)

{

printf("%d%s", dados[n].max\_tour\_str[i], (i == n - 1) ? "]" : ","); /\*Imprime o pior caminho\*/

}

printf("%\*f%s\n", formatar, dados[n].tour\_time, "s"); /\*Imprime o tempo\*/

if (n < 10)

formatar = formatar - 2; /\*No proximo printf a distancia do minPath para o maxLength,que depende da variavel formatar, diminui\*/

else

formatar = formatar - 3; /\*Quando n>10 o pior caminho ocupa mais um espaço para escrever\*/

}

/\*LEGENDA-> print das cidades e o respetivo numero

para facilitar a leitura dos minPath e maxPath\*/

printf("%s\n", "LEGENDA DAS CIDADES:");

for (i = 0; i <= max\_n\_cities; i++)

{

if (a[i] == 0 && i != 0)

break;

printf("%s-", cities[a[i]].name);

printf("%d\n", a[i]);

}

printf("----------------------------------------------------------------------------------------------------------------------\n");

/\*Criação dos mapas, para estes poderem ser desenhados, foram criados\*/

/\*2 vetores com o tamanho do nº de cidades do maior tour, visto que o programa demora\*/

/\*demasiado tempo quando o é usado o max\_n\_cities\*/

/\*e porque as posições finais do min\_tour e max\_tour\*/

int min\_tour\_map[p + 1], max\_tour\_map[p + 1];

for (i = 0; i <= (max\_n\_cities - 5); i++)

{

min\_tour\_map[i] = min\_tour[i];

max\_tour\_map[i] = max\_tour[i];

}

min\_tour\_map[p] = -1;

max\_tour\_map[p] = -1;

make\_map("pt\_best\_tour.svg", min\_tour\_map);

make\_map("pt\_worst\_tour.svg", max\_tour\_map);

return 0;

}

**Código do programa TSP por programação dinâmica**

////

// Bernardo Rodrigues

// n\_Mec = 88835

// Vinicius Benite Ribeiro

// n\_Mec = 88835

// ...

//

// AED, 2018/2019

//

// solution of the traveling salesman problem

//

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/time.h>

#include <sys/resource.h>

#include "cities.h"

#include "../P01/elapsed\_time.h"

//

// record best solutions

//

/\*Struct criado para guardar dados\*/

/\*guarda a distancia minima,distancia maxima,o melhor caminho e o pior caminho\*/

struct guardar\_dados

{

int min\_distance;

int max\_distance;

int max\_tour\_str[max\_n\_cities + 1];

int min\_tour\_str[max\_n\_cities + 1];

double tour\_time;

};

static struct guardar\_dados dados[max\_n\_cities + 1];

/\*Array de structs é criado para guardar as informações de n cidades

ex: dados[3] guarda informações de um percurso com 3 cidades

/ dados[4] informações de um percurso de 4 cidades

No final dos calculos dos percursos é usado o struct para

ver os dados\*/

int p = 0; /\*Esta variavel é usada para criar 2 vetores com o tamanho do maior percurso\*/

static int min\_length, max\_length, VISITED\_ALL, mask;

static int min\_tour[max\_n\_cities + 1], max\_tour[max\_n\_cities + 1], dp[1 << max\_n\_cities][max\_n\_cities];

static long n\_tours;

//static int debug = 0; /\*Imprime todos os caminhos e a distancia AUMENTA O TEMPO DO PROGRAMA\*/

static int tour\_dp[1 << max\_n\_cities][max\_n\_cities][max\_n\_cities];

int tsp(int mask, int pos, int n)

{

//printf("%d\n",\_\_builtin\_popcount (mask)); //conta os bits da mascara que estão a 1

n\_tours++;

if (mask == VISITED\_ALL)

{

return cities[pos].distance[0];

}

if (dp[mask][pos] != -1)

{

return dp[mask][pos];

}

int min\_ans = 999999;

for (int city = 0; city < n; city++)

{

if ((mask & (1 << city)) == 0)

{

int newAns = cities[pos].distance[city] + tsp(mask | (1 << city), city, n);

if (min\_ans > newAns)

{

min\_ans = newAns;

}

}

}

dp[mask][pos] = min\_ans;

return min\_ans;

}

int tsp\_max(int mask, int pos, int n)

{

if (mask == VISITED\_ALL)

{

return cities[pos].distance[0];

}

if (dp[mask][pos] != -1)

{

return dp[mask][pos];

}

int max = 0;

for (int city = 0; city < n; city++)

{

if ((mask & (1 << city)) == 0)

{

int newAns = cities[pos].distance[city] + tsp\_max(mask | (1 << city), city, n);

//printf("newAns: %d\n",newAns );

//printf("%d\n", newAns);

if (max < newAns)

{

max = newAns;

}

}

}

dp[mask][pos] = max;

return max;

}

//

// main program

//

int main(int argc, char \*\*argv)

{

int n\_mec, special, n, i, a[max\_n\_cities];

char file\_name[32];

double dt1;

n\_mec = 82773; // CHANGE THIS!

special = 0;

init\_cities\_data(n\_mec, special);

printf("data for init\_cities\_data(%d,%d)\n", n\_mec, special);

fflush(stdout);

#if 0

print\_distances();

#endif

for (n = 3; n <= (max\_n\_cities - 3); n++) /\*AUMENTAR OU DIMINUIR--MAXIMO 18 CIDADES\*/

{

dt1 = -1.0;

if (n <= 16)

{

(void)elapsed\_time();

for (i = 0; i < n; i++)

a[i] = i;

min\_length = 1000000000;

max\_length = 0;

n\_tours = 0l;

//-----------------------------------//

VISITED\_ALL = (1 << n) - 1;

mask = 1;

for (int i = 0; i < (1 << max\_n\_cities); i++)

{

for (int j = 0; j < max\_n\_cities; j++)

{

dp[i][j] = -1;

for (int k = 0; k < max\_n\_cities; ++k)

{

tour\_dp[i][j][k] = 0;

}

}

}

min\_length = tsp(mask, 0, n); // no need to change the starting city, as we are making a tour

for (int i = 0; i < (1 << max\_n\_cities); i++)

{

for (int j = 0; j < max\_n\_cities; j++)

{

dp[i][j] = -1;

}

}

max\_length = tsp\_max(mask, 0, n);

//-----------------------------------//

dt1 = elapsed\_time();

dados[n].tour\_time = dt1; /\*Regista o tempo de cada tour no struct dados\*/

printf("tsp\_v1() finished in %8.3fs (%ld tours generated)\n", dt1, n\_tours);

printf(" min %5d [", min\_length);

for (i = 0; i < n; i++)

printf("%2d%s", min\_tour[i], (i == n - 1) ? "]\n" : ",");

memcpy(dados[n].min\_tour\_str, min\_tour, sizeof(min\_tour)); /\*Regista o menor tour\*/

dados[n].min\_distance = min\_length; /\*Update da distancia minima para um tour com n cidades\*/

printf(" max %5d [", max\_length);

for (i = 0; i < n; i++)

printf("%2d%s", max\_tour[i], (i == n - 1) ? "]\n" : ",");

memcpy(dados[n].max\_tour\_str, max\_tour, sizeof(max\_tour)); /\*Regista o maior tour\*/

dados[n].max\_distance = max\_length; /\*Update da distancia maxima para um tour com n cidades\*/

fflush(stdout);

if (argc == 2 && strcmp(argv[1], "-f") == 0)

{

min\_tour[n] = -1;

sprintf(file\_name, "min\_%02d.svg", n);

make\_map(file\_name, min\_tour);

max\_tour[n] = -1;

sprintf(file\_name, "max\_%02d.svg", n);

make\_map(file\_name, max\_tour);

}

}

}

/\*No final do programa é impresso uma tabela com as informações dos percursos\*/

printf("----------------------------------------------------------------------------------------------------------------------");

printf("\n%s%12s%12s%40s%12s%40s\n", "n", "minLength", "minPath", "maxLength", "maxPath", "tempo");

int formatar = 40; /\*Esta variavel é usada para formatar os printf para facilitar a leitura das informações da tabela\*/

for (n = 3; n <= max\_n\_cities; n++)

{

if (dados[n].min\_distance == 0)

{

p = n; /\*Esta variavel é usada mais à frente no programa para definir o tamanho de 2 vetores\*/

break; /\*Se dados[n].min\_distance == 0 significa que não feito um percurso para n cidades\*/

}

printf("%-9d", n); /\*Imprime o numero de cidades de um percurso\*/

printf("%-9d[", dados[n].min\_distance); /\*Imprime a distancia minima\*/

for (i = 0; i < n; i++)

{

printf("%d%s", dados[n].min\_tour\_str[i], (i == n - 1) ? "]" : ","); /\*Imprime o melhor caminho\*/

}

printf("%\*d%-5s[", formatar, dados[n].max\_distance, ""); /\*Imprime a distancia maxima\*/

for (i = 0; i < n; i++)

{

printf("%d%s", dados[n].max\_tour\_str[i], (i == n - 1) ? "]" : ","); /\*Imprime o pior caminho\*/

}

printf("%\*f%s\n", formatar, dados[n].tour\_time, "s"); /\*Imprime o tempo\*/

if (n < 10)

formatar = formatar - 2; /\*No proximo printf a distancia do minPath para o maxLength,que depende da variavel formatar, diminui\*/

else

formatar = formatar - 3; /\*Quando n>10 o pior caminho ocupa mais um espaço para escrever\*/

}

/\*LEGENDA-> print das cidades e o respetivo numero

para facilitar a leitura dos minPath e maxPath\*/

printf("%s\n", "LEGENDA DAS CIDADES:");

for (i = 0; i <= max\_n\_cities; i++)

{

if (a[i] == 0 && i != 0)

break;

printf("%s-", cities[a[i]].name);

printf("%d\n", a[i]);

}

printf("----------------------------------------------------------------------------------------------------------------------\n");

/\*Criação dos mapas, para estes poderem ser desenhados, foram criados\*/

/\*2 vetores com o tamanho do nº de cidades do maior tour, visto que o programa demora\*/

/\*demasiado tempo quando o é usado o max\_n\_cities\*/

/\*e porque as posições finais do min\_tour e max\_tour\*/

int min\_tour\_map[p + 1], max\_tour\_map[p + 1];

for (i = 0; i <= (max\_n\_cities - 6); i++)

{ //FOR ACABA NO NUMERO DE CIDADES CALCULDAS - 1 (max\_n\_cities - 1)

min\_tour\_map[i] = min\_tour[i];

max\_tour\_map[i] = max\_tour[i];

}

min\_tour\_map[p] = -1;

max\_tour\_map[p] = -1;

make\_map("pt\_best\_tour.svg", min\_tour\_map);

make\_map("pt\_worst\_tour.svg", max\_tour\_map);

return 0;

}