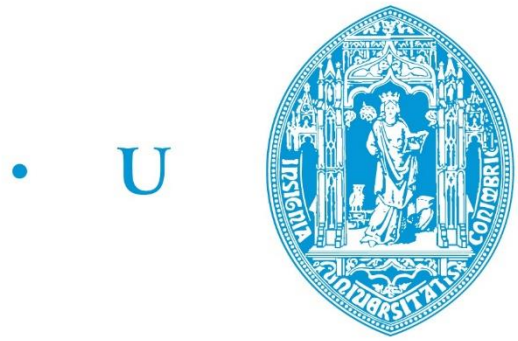


FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Algoritmos e Estruturas de Dados

2016/2017 – 2º Semestre

Trabalho Prático Nº2: Rally Dakar

Trabalho realizado por:
Hugo Teixeira Nº 2002126180 hrafael@student.dei.uc.pt
Rui Rocha Nº 2010135453 rmrocha@student.dei.uc.pt

Índice

Introdução

O trabalho consiste na análise de um dos problemas NP-completo clássico, o problema do caixeiro viajante, aplicado a uma prova de rally. Neste ambiente, o piloto tem que passar por todas as cidades, começando na cidade partida e finalizando na mesma cidade, percorrendo a menor distância no trajeto da prova. Foi-nos dado dois cenários possíveis, no primeiro as distâncias entre duas cidades, sejam elas designadas por A e B, em que AB é igual a BA, no segundo cenário as distâncias são diferentes, isto é, AB é diferente de BA.

Dado esta análise do problema, foi-nos proposto a implementação de geradores de mapas para ambos os cenários, e fazer a correspondente analise do comportamento e complexidade do algoritmo implementado.

Objetivo

Neste trabalho, o objetivo é contruir uma estrutura de dados baseada em grafos, e implementar um algoritmo de pesquisa, para encontrar a melhor solução.

No nosso trabalho usamos uma abordagem em *brutt force*. Esta abordagem soluciona o problema, mas implica uma complexidade elevada para ambos os cenários. No primeiro cenário a complexidade do problema é O((n-2)!), e no segundo cenário a complexidade é O((n-1)!), no pior dos casos.

Ambiente

O computador, portátil, utilizado tem as seguintes características:

- Sony Vaio VPCF1
- Processador: Intel(R) Core(TM) i5 CPU M480 @ 2.67GHz, 64 bits
- 6 GB de memória
- Disco SSD de 240 GB
- SO, Windows 10 64 bits
- Programa: Wing IDE 101 6.0
- Python 3.4.4

Implementação

Nesta secção, é referenciada a forma como são gerados os mapas, o algoritmo de pesquisa da solução para cada uma das tarefas e as estruturas implementadas.

Estruturas

É usado um dicionário para as cidades, que é criado através da classe "Node", esta classe representa um nó do grafo.

O programa tem uma classe, "Graph", onde é gerado o grafo lido do mapa, no qual se quer pesquisar o caminho mais curto, iniciando numa origem atribuída aleatoriamente, passando por todas as outras cidades acabando nessa mesma origem. Esta classe, "Graph", tem uma lista de arestas, caminho entre as cidades, com a origem o destino e a distância entre cada par. E é criada uma lista de sucessores, isto é, uma lista de cidades vizinhas que contem a cidade e a respetiva distância.

No algoritmo de pesquisa, é usada uma lista para guardar o caminho já percorrido e a respetiva distância percorrida. E uma lista de cidades visitadas, porque só se pode passar por uma cidade uma única vez.

Mapas Tarefa 1

Para a "Tarefa 1", é gerada uma lista com distâncias para atribuir a cada caminho entre duas cidades, desordenando a ordem da mesma. Depois para cada par de cidades é atribuído uma distância. Neste mapa só é atribuída uma distância entre cada par de cidades, isto é, a distância entre a cidade A e a cidade B é igual ao seu inverso. Logo no mapa, só apresentamos uma dessas distâncias, porque a outra é igual.

O número de cidades é inserido previamente pelo utilizador.

Mapas Tarefa 2

Para a "Tarefa 2", é gerada uma lista com distâncias, tal como na "Tarefa 2", para atribuir a cada caminho entre duas cidades, desordenando a ordem da mesma. Depois para cada par de cidades é atribuído uma distância. Neste mapa é atribuída uma distância entre cada par de cidades, mas a distância entre a cidade A e a cidade B é diferente da distância da cidade B para A. Logo no mapa são apresentadas todas as distâncias.

Algoritmo de Pesquisa

O algoritmo de pesquisa implementado, percorre todos os caminhos possíveis, isto no pior caso, encontrando quase todas as soluções ou mesmo todas as soluções. Na “Tarefa 1”, não são percorridos todos os caminhos, devido às restrições impostas, mas pode-o percorrer na “Tarefa 2”, se a solução esteja no pior caso, isto é, se a solução se encontra no último caminho que o algoritmo percorre.

Restrições:

- Não percorre o último ramo da árvore gerada pelo grafo, porque este é a combinação de todos os outros ramos anteriores, esta restrição só é a plicada à “Tarefa 1”.
- Depois de encontrada uma solução, o valor encontrado é comparado sempre com os passos que o algoritmo realiza nos outros ramos da árvore gerada através do grafo. Isto é, se a distância entre o caminho de uma cidade D para uma cidade F (ou outra qualquer) é maior que a solução encontrada, o algoritmo para nessa cidade “rejeitando” o caminho. Restrição comum às duas tarefas.

Análise de Dados

Na análise de dados geramos vários mapas para cada um dos cenários do nosso trabalho. Um dos critérios na procura do caminho é a pesquisa de uma solução com o menor custo, passando por todas as cidades. De forma a encontrar soluções que incluam todos vértices do grafo, gerado através do mapa da prova de rally, com a condição de não exceder mais de trinta minutos (mil e oitocentos segundos) de execução do algoritmo de pesquisa.

Como a geração dos mapas são aleatórios não dá para comparar os dois cenários um com o outro.

Nas tabelas abaixo encontram-se os dados retirados das várias experiências analisadas.

Tabela de dados do primeiro cenário

NUMERO DE CIDADES	TEMPO DE PESQUISA DE CAMINHO(MINUTOS)	CUSTO	CAMINHO
5	0,000465034	24	2 -> 1 -> 3 -> 0 -> 4 -> 2
6	0,0016384	27	1 -> 3 -> 0 -> 2 -> 5 -> 4 -> 1
7	0,008089983	51	3 -> 0 -> 4 -> 2 -> 5 -> 6 -> 1 -> 3
8	0,028599619	66	3 -> 2 -> 5 -> 0 -> 7 -> 1 -> 6 -> 4 -> 3
9	0,085692222	71	5 -> 1 -> 8 -> 4 -> 2 -> 6 -> 0 -> 7 -> 3 -> 5
10	0,141868615	82	6 -> 1 -> 2 -> 3 -> 5 -> 7 -> 8 -> 9 -> 0 -> 4 -> 6
11	0,585776721	110	1 -> 0 -> 10 -> 4 -> 6 -> 9 -> 7 -> 2 -> 5 -> 8 -> 3 -> 1
12	4,453686919	145	4 -> 3 -> 6 -> 1 -> 10 -> 5 -> 9 -> 7 -> 0 -> 2 -> 11 -> 8 -> 4
13	10,30956358	154	1 -> 3 -> 8 -> 9 -> 0 -> 6 -> 4 -> 10 -> 2 -> 12 -> 11 -> 5 -> 7 -> 1
14	151,4632886	229	9 -> 11 -> 8 -> 4 -> 2 -> 3 -> 1 -> 0 -> 10 -> 13 -> 6 -> 5 -> 7 -> 12 -> 9
15	210,0354716	233	9 -> 2 -> 5 -> 1 -> 8 -> 4 -> 14 -> 11 -> 7 -> 10 -> 12 -> 0 -> 6 -> 3 -> 13 -> 9
16	1635,383607	287	0 -> 8 -> 4 -> 6 -> 3 -> 7 -> 12 -> 9 -> 1 -> 11 -> 10 -> 5 -> 15 -> 2 -> 13 -> 14 -> 0
17	1613,347745	294	1 -> 3 -> 15 -> 2 -> 12 -> 16 -> 5 -> 9 -> 8 -> 4 -> 0 -> 10 -> 11 -> 7 -> 6 -> 13 -> 14 -> 1

Na execução do mapa com 18 cidades, não foi possível obter resultados em tempo útil, excedendo largamente o limite permitido no enunciado do problema.

O mapa 17 tem um menor valor de execução que o mapa 16, porque depende das distâncias atribuídas a cada par de cidades.

Tabela de dados do segundo cenário

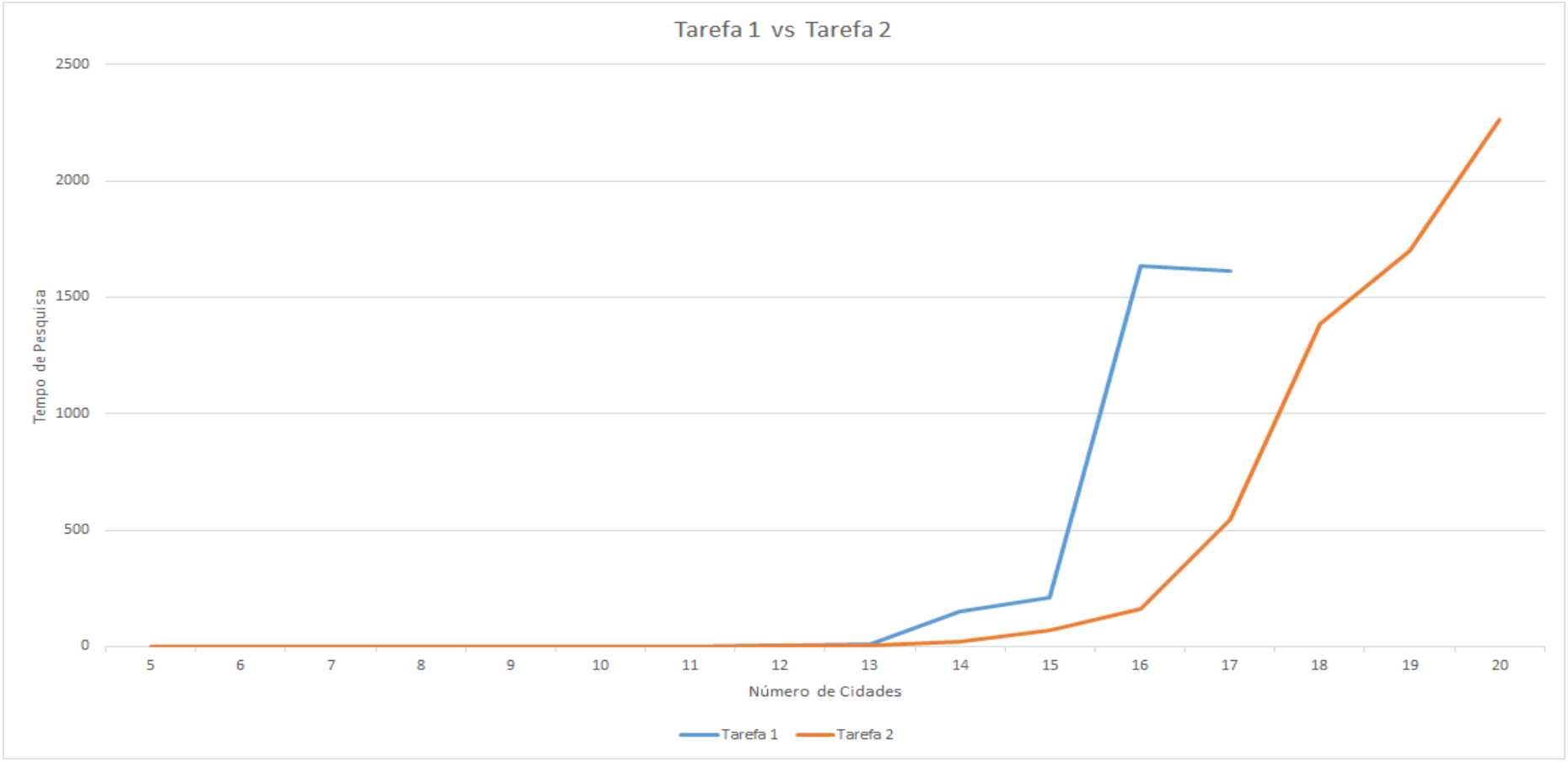
NÚMERO DE CIDADES	TEMPO DE PESQUISA CAMINHO (SEGUNDOS)	CUSTO	CAMINHO
5	0,000407675	26	3 -> 2 -> 4 -> 1 -> 0 -> 3
6	0,001910568	64	3 -> 5 -> 4 -> 1 -> 0 -> 2 -> 3
7	0,003254471	65	6 -> 4 -> 1 -> 0 -> 2 -> 3 -> 5 -> 6
8	0,008819101	69	3 -> 1 -> 4 -> 6 -> 5 -> 7 -> 0 -> 2 -> 3
9	0,029740262	123	3 -> 4 -> 2 -> 1 -> 7 -> 0 -> 8 -> 5 -> 6 -> 3
10	0,102857691	152	0 -> 6 -> 3 -> 4 -> 2 -> 5 -> 9 -> 7 -> 8 -> 1 -> 0
11	0,234263415	178	9 -> 7 -> 10 -> 3 -> 0 -> 1 -> 2 -> 5 -> 6 -> 8 -> 4 -> 9
12	1,927226343	202	10 -> 4 -> 3 -> 9 -> 0 -> 7 -> 2 -> 11 -> 6 -> 1 -> 5 -> 8 -> 10
13	2,385293747	301	12 -> 0 -> 10 -> 2 -> 6 -> 1 -> 9 -> 11 -> 5 -> 8 -> 3 -> 7 -> 4 -> 12
14	18,42801455	343	8 -> 6 -> 0 -> 9 -> 12 -> 5 -> 3 -> 13 -> 10 -> 4 -> 11 -> 7 -> 2 -> 1 -> 8
15	68,40680313	431	7 -> 5 -> 1 -> 8 -> 11 -> 13 -> 0 -> 9 -> 3 -> 6 -> 4 -> 14 -> 10 -> 12 -> 2 -> 7
16	161,7906648	479	2 -> 5 -> 12 -> 13 -> 0 -> 11 -> 4 -> 1 -> 14 -> 6 -> 9 -> 3 -> 7 -> 8 -> 15 -> 10 -> 2
17	547,2138731	468	8 -> 9 -> 13 -> 14 -> 6 -> 0 -> 16 -> 11 -> 4 -> 2 -> 10 -> 15 -> 1 -> 12 -> 7 -> 5 -> 3 -> 8
18	1386,350281	534	0 -> 15 -> 6 -> 10 -> 8 -> 16 -> 3 -> 12 -> 1 -> 14 -> 2 -> 5 -> 17 -> 13 -> 4 -> 7 -> 11 -> 9 -> 0
19	1700,539663	586	3 -> 16 -> 17 -> 10 -> 0 -> 5 -> 4 -> 6 -> 8 -> 14 -> 1 -> 12 -> 2 -> 7 -> 13 -> 11 -> 18 -> 15 -> 9 -> 3
20	2265,306238	587	9 -> 10 -> 1 -> 7 -> 12 -> 13 -> 15 -> 11 -> 14 -> 17 -> 3 -> 4 -> 16 -> 6 -> 18 -> 2 -> 0 -> 8 -> 5 -> 19 -> 9

O mapa número 20 encontra-se a vermelho, porque excede o tempo limite do problema, mas foi possível encontrar em tempo útil uma solução.

Gráfico da Tarefa 1

Gráfico da Tarefa 2

Gráfico Tarefa 1 vs Tarefa 2



Conclusão

Analisando o *benchmark* deste trabalho, verificamos o que a priori se esperava. No cenário com distâncias diferentes nos dois sentidos, era de esperar que o comportamento fosse o esperado. O programa vai escolhendo sempre o percurso mais curto, e na escolha do caminho com menor custo entre a penúltima cidade e a origem, como tendo distancias diferentes em ambos os sentidos facilita neste processo de procura. Já no cenário de distancias iguais em ambos os sentidos, como tendo uma complexidade elevada, neste caso torna-se muito pesado computacionalmente encontrar soluções para mapas com mais de dezassete cidades.

Verificamos um dado curioso, na análise de dados. O tempo de procura nos mapas com número elevado de cidades depende muito dos mapas. Vejamos o caso na tarefa 1 o mapa de 16 cidades demorou mais tempo que o mapa de 17 cidades. Isso demostra que o valor das distâncias entre cidades influência muito a procura de soluções para o problema.

Referências

- Material disponibilizado pelo professor
- https://social.msdn.microsoft.com/Forums/office/pt-BR/4e0f7847-70ba-4538-8d3e-fffb8a821945/algoritmo-gentico-problema-do-caxeiro-viajante?forum=vscsharppt
- http://wiki.icmc.usp.br/images/5/5e/Grafos_Caminhos_Gra%C3%A7a_Rosane_2012.pdf
- http://www.mat.uc.pt/~mcag/FEA2003/Teoria_de_Grafos
- https://pt.wikipedia.org/wiki/Lista_de_algoritmos
- https://en.wikipedia.org/wiki/Travelling_salesman_problem
- https://pt.wikipedia.org/wiki/Problema_do_caixeiro-viajante
- http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/EulerHam/euler_ham.html
- http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/CaminhoMin/caminho_min.html
- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
- http://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html
- https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
- http://www.programming-algorithms.net/article/47389/Bellman-Ford-algorithm
- https://www.cs.cmu.edu/afs/cs/academic/class/15210-s13/www/lectures/lecture13.pdf
- https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm
- http://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/
- http://www.programming-algorithms.net/article/45708/Floyd-Warshall-algorithm
- https://en.wikipedia.org/wiki/A*_search_algorithm
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- http://web.mit.edu/eranki/www/tutorials/search/
- http://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdfA*
- https://pt.wikipedia.org/wiki/Busca_em_largura
- https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/the-breadth-first-search-algorithm
- https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms
- http://www.sciencedirect.com/science/article/pii/S0377221703002480
- https://www.ics.uci.edu/~welling/teaching/271fall09/antcolonyopt.pdf
- http://mathworld.wolfram.com/AntColonyAlgorithm.html