FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

2º Projecto Prático

Algoritmos e Estruturas de Dados

2016/2017

João Afonso Póvoa Marques – 2015227041
José Miguel Saraiva Monteiro – 2015235572
Leonardo Machado Alves Vieira – 2015236155

INDICE

1 - Introdução

Neste projecto foi nos proposto o desenvolvimento e aperfeiçoamento de algoritmos para percorrer um grafo, no caso específico seria um mapa de um rally Dakar, respeitando as condições impostas de percorrer todas as cidades sem repetição, com excepção da cidade inicial que será também a final.

Este problema é uma adaptação do *Traveling Salesman Problem,* cada vértice do grafo corresponde a uma cidade e os pesos das arestas a distância entre as duas cidades, consoante a tarefa a distância pode ser, ou não, a mesma para os dois sentidos. Importante de notar que o grafo (mapa) é completo, isto é, todos os vértices têm uma ligação a todos ou outros restantes.

Sendo assim todas as soluções apresentadas no trabalho são apenas aproximações à solução óptima visto que para este problema ainda não existe solução óptima definida.

Implementámos 4 algoritmos diferentes cada um melhorado em relação ao anterior. Em concreto os algoritmos implementados foram um algoritmo de *brute force,* um algoritmo *brute force* mas mais optimizado, um algoritmo baseado em recursividade e, por fim, um algoritmo recursivo com *greedy*. O *benchmark* foi feito com o auxílio da livraria *time* e com o método *process_time* que permite medir apenas o tempo de processamento do processo, permitindo resultados mais coerentes.

Para correr o programa, basta correr o *gui.py*, num interpretador de python ou num terminal linux.

2- Descrição das estruturas

A estrutura de dado baseada em grafos implementada, foi a fornecidas no material da cadeira. É um estrutura que tem como base dicionários de python com duas classes, uma para o grafo em si e outra para cada vértice do grafo. Apenas foram aplicadas alterações mínimas de modo a facilitar a integração da estrutura no conceito do problema, e para facilitar algumas funções implementadas para os algoritmos.

Não foram feitas nenhumas otimizações às estruturas ao longo do projecto.

3- Descrição do gerador

Foi criado um gerador de mapas que utiliza a estrutura a cima referida. Este gerador permite criar mapas para qualquer um dos dois cenários possíveis, sendo esta informação passada por parâmetro à função. Este gerador de mapas cria todos os caminhos entre cidades com distâncias necessariamente diferente. Para isto é usado a livraria *random* e o seu método *sample*. Este método recebe dois parâmetros, uma lista de n valores e o número de valores que queremos tirar aleatoriamente e sem repetição dessa lista.

Para tornar a geração de mapas mais dinâmica, foi definido que o leque de distâncias disponíveis seria de 10 ao dobro do número de arestas existentes

Apenas é possível criar mapas de forma aleatória.

4- Descrição dos algoritmos implementados

Para resolver este problema foi pensado, após alguma pesquisa, em 4 algoritmos, sendo que 2 deles são na prática otimização dos outros 2 previamente implementados.

Começámos por implementar um algoritmo simples de *Brute Force* e de seguida realizámos algumas otimizações que serão em seguida descritas.

Depois de aprimorar estes algoritmos, foram implementados outros dois recursivos.

Todas estas estrutura poderiam ainda ser otimizadas a vários níveis. Um deles seria organizar melhor as estruturas auxiliares usadas de modo a serem mais eficientes e a ocuparem menos espaço em memória.

Brute Force

Foi o primeiro algoritmo a ser implementado e é o mais primitivo deles todos.

Usa um função auxiliar que recursivamente retorna todas as permutações de uma lista. Seria no entanto vantajoso, em vez de retornar todas a permutações em simultâneo (n! Sendo n o número de vértices), retornar apenas uma permutação ainda não testada! Sendo então essa lista todos os vértices (cidades) do grafo (mapa) menos a cidade de origem, esta função auxiliar retorna todos os caminhos possíveis para

partir e chegar à mesma cidade. Importante notar que este método de permutações apenas funciona porque o grafo é completo (todos os vértices ligam a todos os vértices).

O algoritmo corre todos estes caminhos possíveis e guarda todos as distâncias e o respectivo caminho num dicionário que depois é retornado.

Assim sendo, este algoritmo funciona para qualquer dos dois cenários propostos.

Brute Force Optimizado

Foram depois feitas algumas otimizações ao algoritmo original.

A primeira das quais foi bastante óbvia. O algoritmo passa a calcular a menor distância dos caminhos que já foram testados e durante o cálculo da distância total dos caminhos em falta, se a distância atual for maior do que o mínimo até lá encontrado, o ciclo salta para a próxima iteração, isto é, passa a testar o próximo caminho.

A outra otimização implementada só se aplica a grafos gerados para o primeiro cenário, isto porque esta optimização se baseia em cortar caminhos que sejam inversos. Depois de ter calculado a lista de todos os caminhos possíveis, é criada uma lista auxiliar, que vai actualizando com todos os caminhos já testados. Assim, antes de testar um novo caminho é testado se o caminho inverso se encontra previamente na lista auxiliar.

Para ser possível utilizar a mesma função para os dois cenários, é necessário enviar como argumento qual o cenário a ser testado para “ativar” ou “desativar” esta condição.

O possível optimização das permutações referida no algoritmo anterior, poderia também ser implementada neste caso.

Recursive

Foi pensado mais tarde na possibilidade de implementar um algoritmo recursivo para resolver o problema. Foi então implementado um algoritmo recursivo que é chamado para cada vértice. Este algoritmo conta com uma lista auxiliar de cidades já visitadas. Assim, o algoritmo vai avançando no caminho sabendo quais as cidades em que já passou, evitando assim quebrar as regras do problema. Começa por medir a

distância do vértice atual até a um dos vizinhos ainda não visitados. De seguida compara a distância total que irá percorrer e verifica se é maior do que a menor já encontrada. Caso esta condição se verifique, irá ser imediatamente testado o próximo vizinho, evitando assim chamadas recursivas desnecessárias. Quando chega a um vértice do qual todos os vizinhos foram testados, a condição de saída é ativada e é inserido num dicionário final, a distância total do caminho (chave) e o caminho (valor).

Com este algoritmo também é possível calcular a solução para qualquer cenário do problema.

Recursive + Greedy

Foi depois feita uma alteração no algoritmo recursivo, na expectativa de optimização. Esta alteração consistiu em, ao invés de escolher o vizinho a passar para a próxima chamada recursiva aleatoriamente, escolher do vizinho mais perto para o vizinho mais distante. O resto do algoritmo manteve-se intacto após a alteração.

Porém, como irá ser mostrado na análise de resultados, o resultado final não foi o pretendido, principalmente devido à forma como esta alteração foi implementada.

5 - Ficheiros

Como foi pedido, foi implementada a função de gravação dos grafos já testados. Numa fase inicial todos os grafos testados eram colocados no mesmo ficheiro com muito pouca informação. Esta informação foi sendo aprimorada até à fase final. Numa fase mais avançada a escrita em ficheiros passou a ser efectuada como é pedido no enunciado. Isto é, no seguinte formato: *tarefa_x_y_z,* onde *x* é o cenário utilizado, *y* o número de cidades e *z* o algoritmo utilizado.

Não é possível *reutilizar* o mesmo grafo, logo toda a informação relativa aos testes esteja no ficheiro associado ao grafo em causa.

Todos os ficheiro de teste encontram-se em anexo.

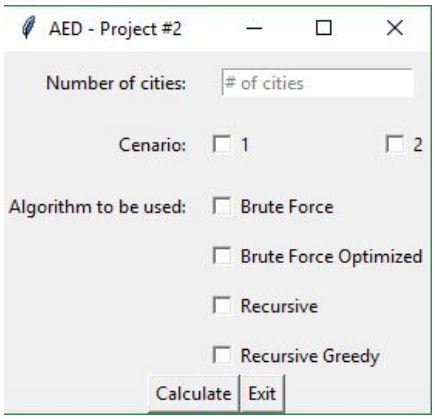
6 - GUI

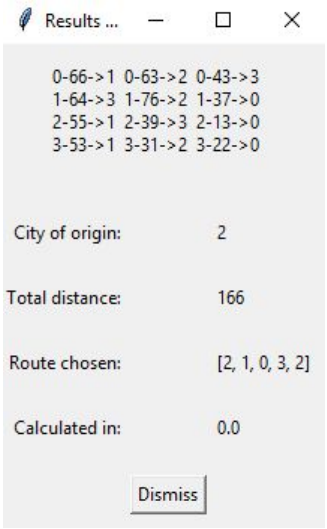
Para este trabalho, foi desenvolvida também uma interface gráfica, que tem como objetivo facilitar a utilização do gerador de mapas, e do programa que os soluciona, tal como nos é pedido na Tarefa 3.

O GUI desenvolvido é muito simples, e consiste num pequeno menu, onde é possível inserir o número de cidades pretendido para a simulação e escolher qual dos cenários e qual (ou quais) algoritmo(s) o utilizador pretende utilizar para a simulação.

Note-se que é apenas possível selecionar um dos cenários de cada vez, ao contrário dos algoritmos, onde podemos selecionar até quatro (número máximo).

Após os valores serem devidamente inseridos, e as opções pretendidas selecionadas, carregando no botão *Calcular*, o programa procede à geração do mapa e ao cálculo do melhor caminho (mais curto). Assim que todos os cálculos estiverem feitos, são automaticamente abertas novas janelas (uma para cada algoritmo selecionado), que nos mostram o grafo (mapa), o ponto de origem, a distância total, o caminho escolhido (mais curto), e o tempo que a solução demorou a ser encontrada com o algoritmo selecionado (em segundos).





7 - Analise de resultados

Para a análise e comparação dos resultados testámos os mesmos mapas para os algoritmos, dentro de valores razoáveis no contexto do problema, isto é, menos de 30 minutos (1800 segundos) aproximadamente .

Neste sentido alguns dos algoritmos não foram testados para números de cidades elevados, devido a impossibilidades temporais.

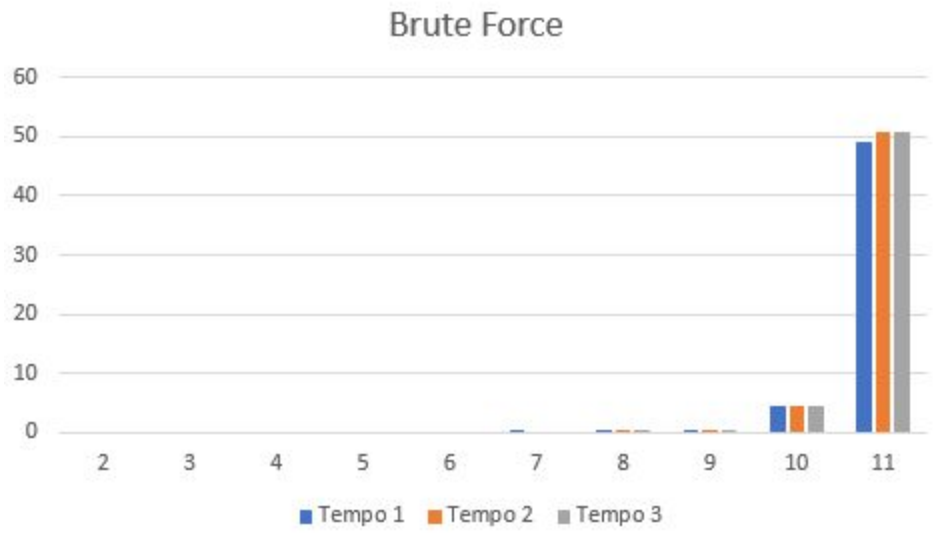
Para cada número de vértices testamos 3 mapas diferentes, com o objectivo de maximizar o número de valores e melhor entender o comportamento dos algoritmos em questão.

Seguem-se os resultados obtidos dos vários testes executados:

Cenário 1:

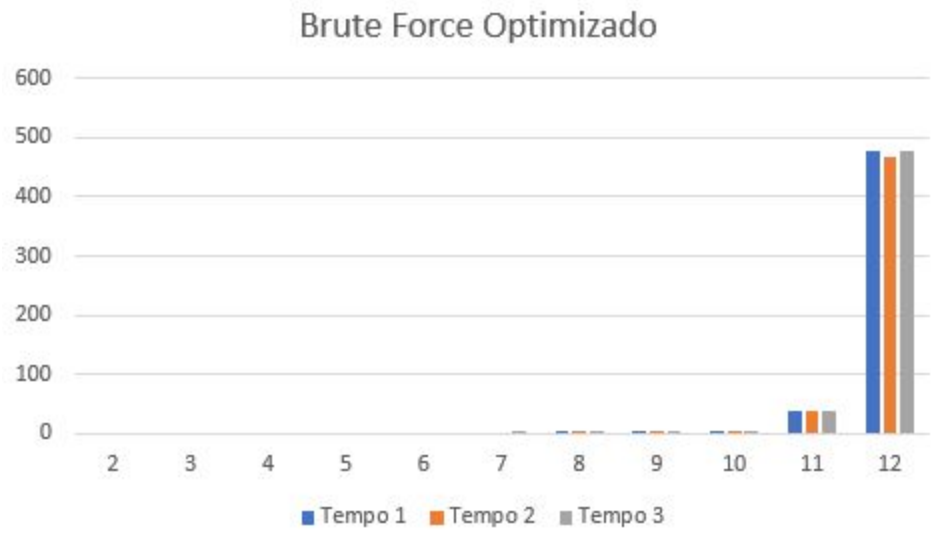
Brute Force

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0,01563	0	0
8	0,04688	0,04688	0,04688
9	0,40625	0,42188	0,42188
10	4,26563	4,5	4,5
11	49,125	51	50,85938


Brute Force
60
50
40
30
20
10
0
2
3
4
5
6
7
8
9
10
11
Tempo 1
Tempo 2
Tempo 3

Brute Force Optimizado

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0,01563
8	0,04688	0,03125	0,03125
9	0,3125	0,32813	0,32813
10	3,35938	3,35928	3,23438
11	36,73438	38,32813	37,29688
12	478,35938	469,07813	477,25



Recursivo

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0,01563	0	0
8	0,01563	0,01563	0
9	0,04688	0,14063	0,0625
10	0,34375	0,3125	0,14063
11	0,25	1,21875	0,45313
12	5,8125	7,875	1,98438
13	1,65625	39,28125	10,84375
14	38,89063	271,54688	5,73438
15	107,09375	63,4375	37,625
16	372,96875	1183,85938	158,0625
17	4038,89063	NA	NA

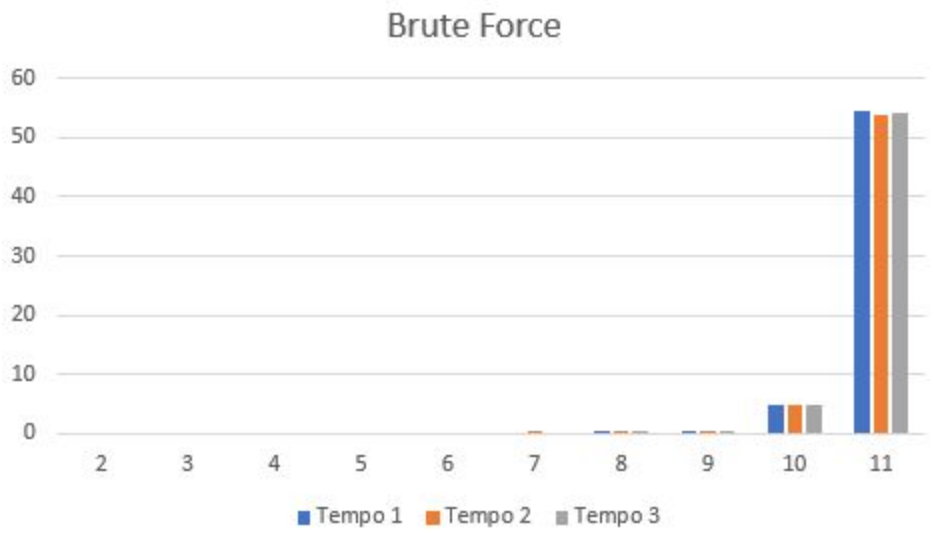
Recursivo com Greedy

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0,01563	0,01563
8	0,04688	0,03125	0,03125
9	0,0625	0,23438	0,125
10	0,59375	0,59375	0,3125
11	0,70313	1,375	0,64063
12	8,21875	6,6875	4,875
13	4,64063	25,57813	6,10938
14	28,26563	241,21875	8,8125
15	83,59375	94,85938	34,28125
16	470,03125	735,34378	72,73438
17	3840,51563	NA	NA

Cenário 2:

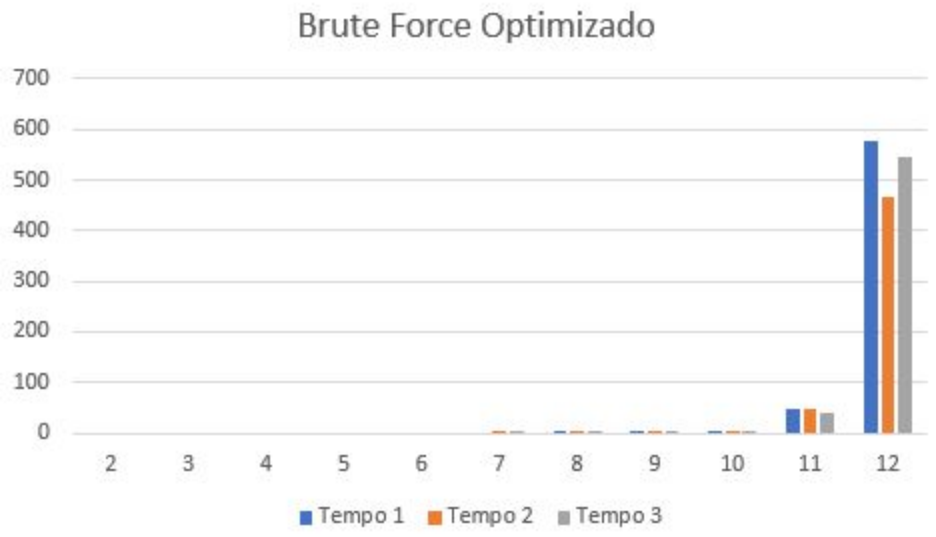
Brute Force

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0,01563	0
8	0,04688	0,04688	0,04688
9	0,4375	0,45313	0,45313
10	4,64063	4,76563	4,73438
11	54,59375	53,75	54,20313



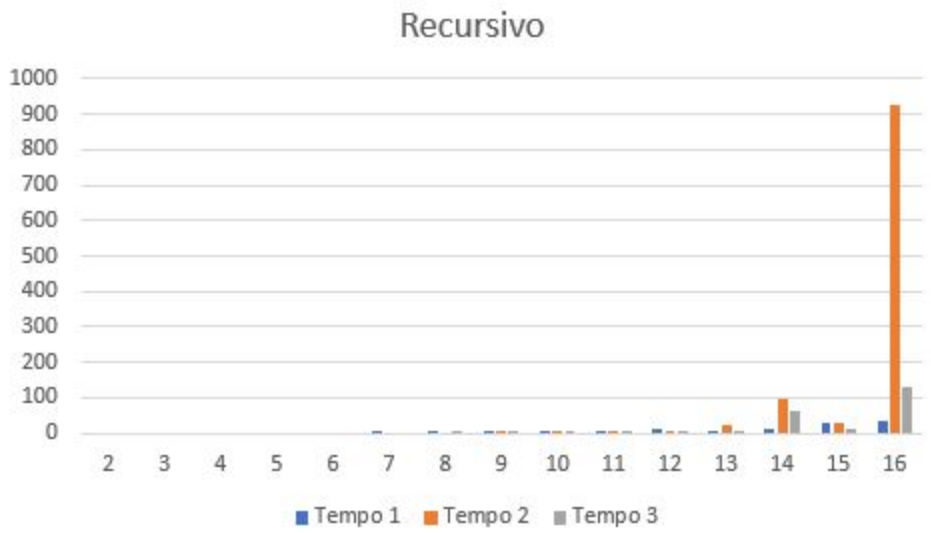
Brute Force Optimizado

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0,01563	0,1563
8	0,04688	0,04688	0,03125
9	0,39063	0,375	0,45313
10	3,73438	3,6875	3,79688
11	46,76563	47,25	40,46875
12	576,35938	467,20313	544,26563



Recursivo

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0,01563	0	0
8	0,01563	0	0,01563
9	0,0625	0,01563	0,09375
10	0,0625	0,0625	0,14063
11	0,54688	1,25	0,14063
12	11,23438	0,23438	2,6875
13	5,39063	23,95313	3,26563
14	9,65625	97,70313	59,78125
15	29,39063	30,60938	11,59375
16	34,10938	926,35938	128,46875



Recursivo com Greedy

Nº Cidades	Tempo 1	Tempo 2	Tempo 3
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0,01563	0	0,01563
8	0,03125	0	0
9	0,07813	0,01563	0,15625
10	0,14063	0,0625	0,17188
11	0,35938	1,78125	0,07513
12	6,82813	0,01875	2,29688
13	2,85938	13,14063	9,59375
14	9,84375	38,45313	106,04688
15	10,71875	4,5625	8,48438
16	35,29688	758,34375	276,9375

Analise dos dados

Como foi referido em cima não foi possível realizar os mesmos testes para todos os algoritmos devido a questões temporais. Em concreto o algoritmos de *Brute Force* ficou pelos 11 vértices sendo que para o 12º o tempo seria, no mínimo, 12 vezes superior ao 11º teste, o mesmo acontece para o *Brute Force Optimizado* que apenas foi possível testar até ao 12º vértice.

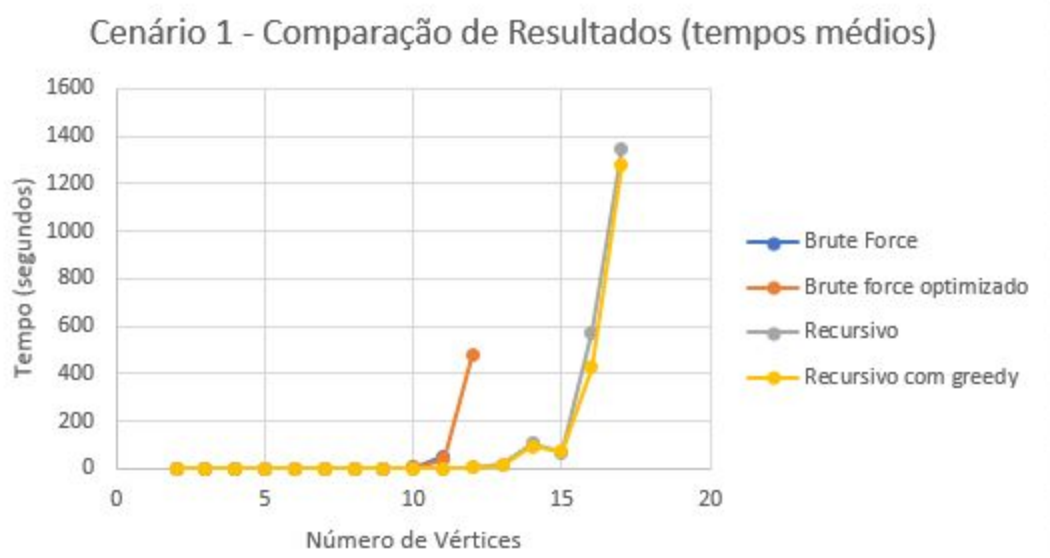
Os algoritmos de melhor performance foram testados uma vez para 17 vértices no cenário 1 mas atendendo aos valores obtidos não foi possível continuar os testes pois o elevado tempo de computação, ainda que este flutuasse bastante, iriam demorar demasiado tempo, decidimos assim que não iriamos avançar com mais testes sobre este número de vértices e números superiores.

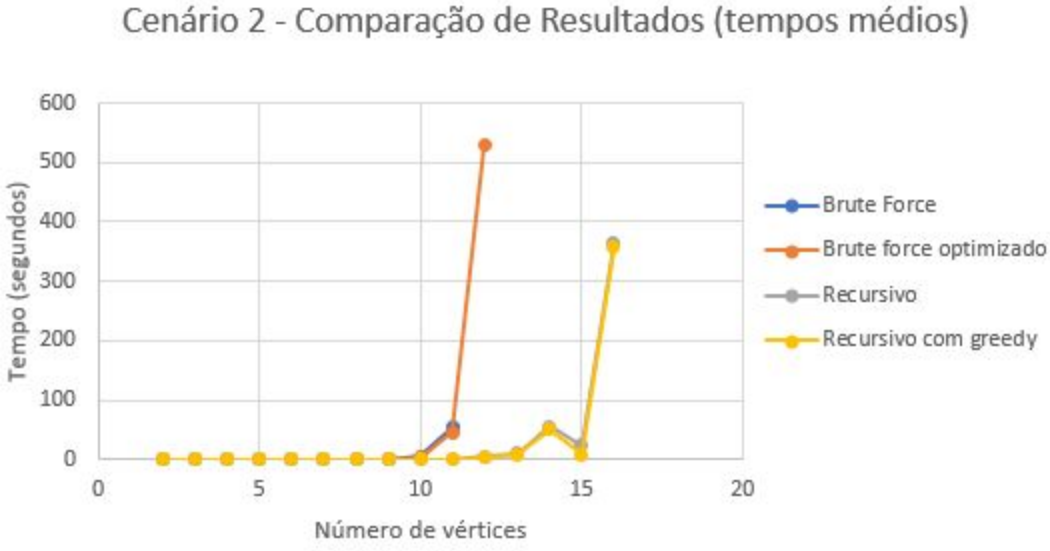
Os algoritmos melhor otimizados têm um problema de valores inconsistentes devido à aleatoriedade dos resultados obtidos. Da maneira que os algoritmos operam estes podem encontrar a solução óptima em muito pouco tempo se este for dos primeiros caminhos a ser testados ou podem levar demasiado tempo a encontrar o caminho óptimo pois este será dos últimos a ser testados.

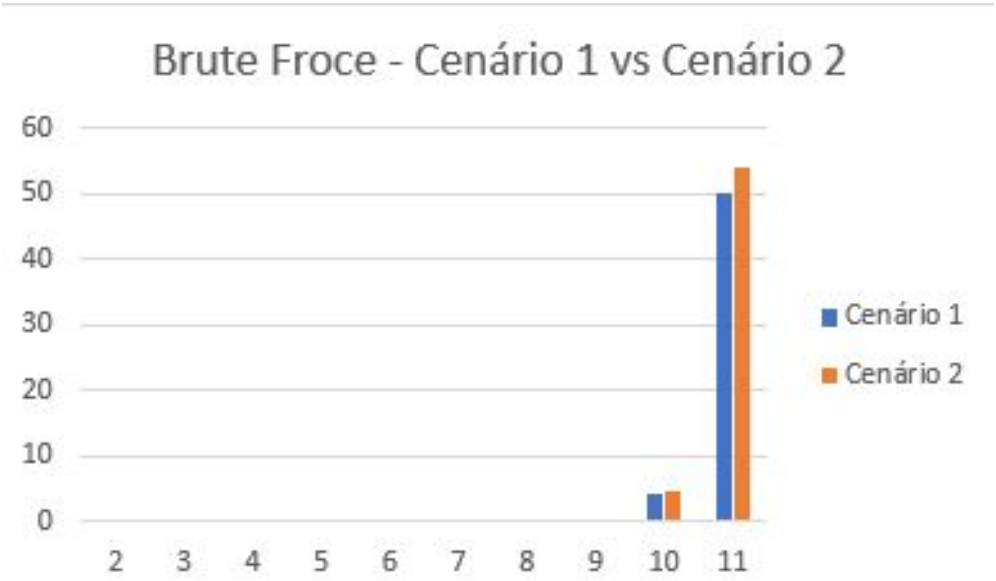
Esta foi a razão principal que nos obrigou a terminar os testes, ainda que alguns dos resultados para 16 vértices sejam consideravelmente bons nada garantia que estes valores se mantivessem. Um exemplo concreto é o primeiro teste do cenário 2 para 16 vértices rondou os 30 segundos para ambos os algoritmos mas o segundo teste ultrapassou os 12 minutos no melhor dos algoritmos.

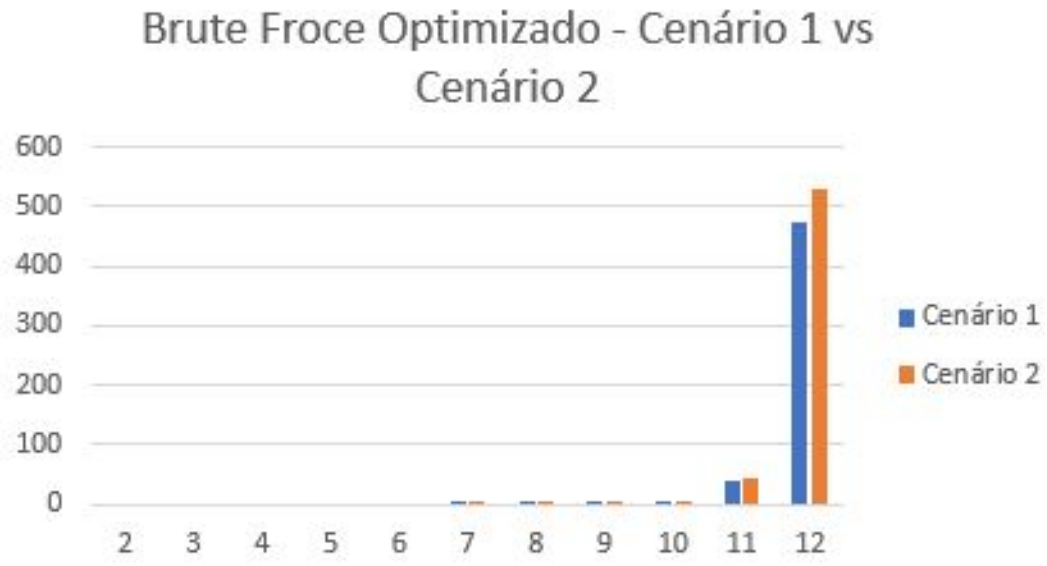
A partir da informação obtida também é possível perceber que o programa encontra uma solução válida em menos de 30 minutos até um máximo de 16 vértices, quando utilizando qualquer um dos algoritmos recursivos.

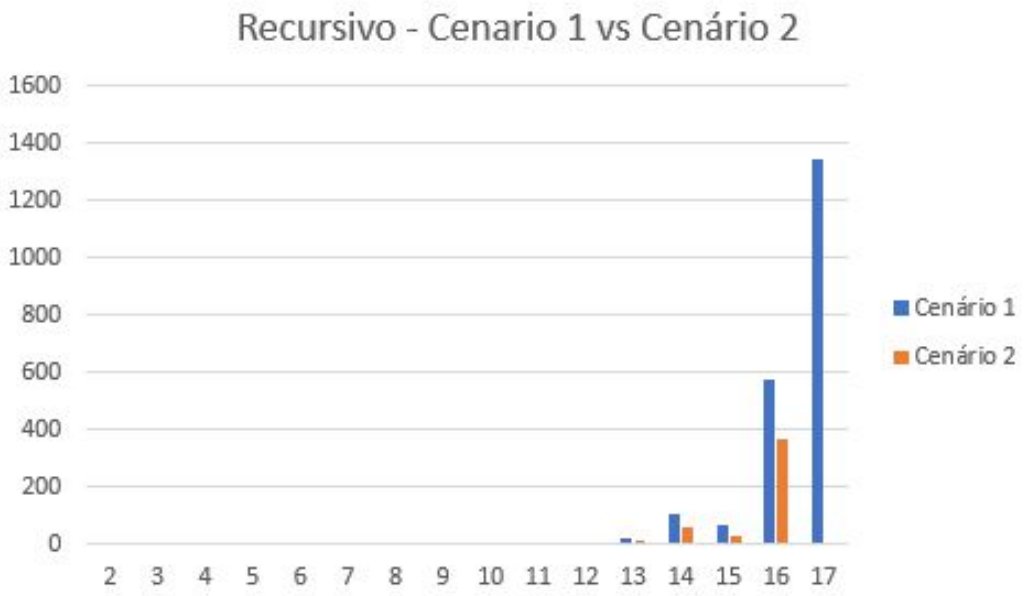
Comparação de Algoritmos

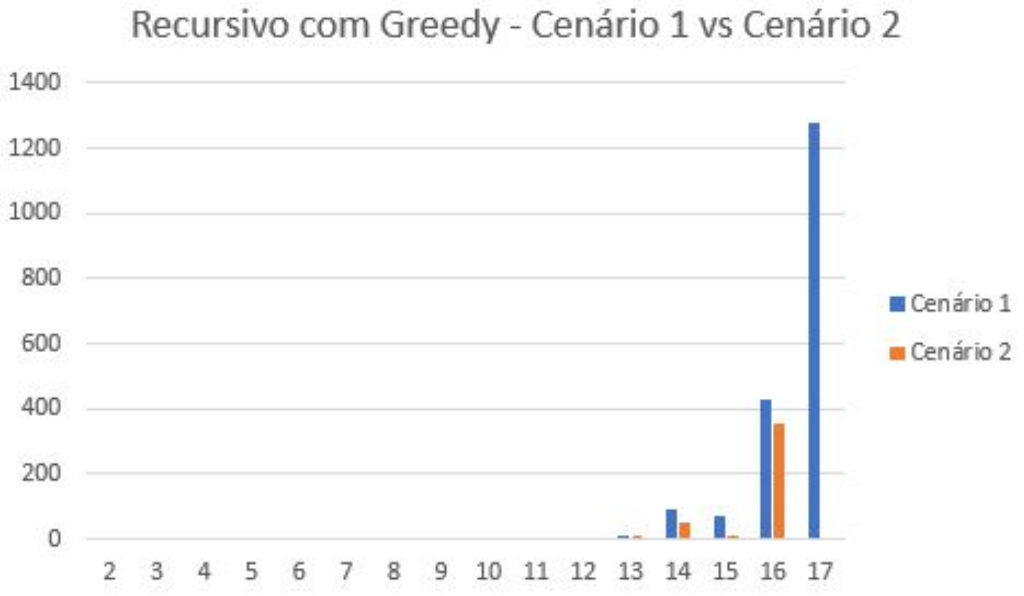





Brute Froce - Cenário 1 vs Cenário 2
60
50
40
30
20
10
0
2
3
4
5
6
7
8
9
10
11
Cenário 1
Cenário 2


Brute Froce Optimizado - Cenário 1 vs Cenário 2
600
500
400
300
200
100
0
2
3
4
5
6
7
8
9
10
11
12
Cenário 1
Cenário 2


Recursivo - Cenario 1 vs Cenário 2
1600
1400
1200
1000
800
600
400
200
0
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Cenário 1
Cenário 2


Recursivo com Greedy - Cenário 1 vs Cenário 2
1400
1200
1000
800
600
400
200
0
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Cenário 1
Cenário 2

É também facilmente visto nos gráficos que em testes para o cenário 1 e 2 se obtém valores tempos.

Para os algoritmos não recursivos o cenários 2 é mais lento. Para os algoritmos recursivos o cenário 2 é mais rápido. Isto acontece porque nos algoritmos nao recursos, principalmente no otimizado, ele vai testar todos os caminhos, e não vai poder cortar os caminhos inversos como faria no cenário 1.

Para os algoritmos recursivos, a diferença é justificada, porque este algoritmos, apesar terem no pior dos casos testar todos os caminhos, não o irão fazer caso já seja menor do o mínimo atual (como foi referido na *Implementação dos Algoritmos*). Assim sendo, se a reversibilidade de caminhos não existir, existe a possibilidade de "cortar" o caminho a ser testado mais rapidamente.

Tal como tínhamos previsto os algoritmos tiveram os resultados esperados e os mais otimizados foram os que retornaram, em média os melhores valores, embora haja de facto situações em que nem sempre o "melhor" algoritmo retorne os melhores resultados em geral este foi o que se revelou melhor tendo em conta todos os testes.

Outro dos *bottlenecks* foi sem dúvida a quantidade de RAM do computador. Revelando-se um impedimento de continuar os testes no Brute Force e uma limitação se tivéssemos continuado com os algoritmos recursivos

8 - Divisão de Tarefas

A divisão de trabalho foi equilibrada. Todos os membros do grupo contribuíram em todos os aspectos do trabalho embora com mais influência em alguns deles.

O Leonardo Vieira teve uma maior incidência na implementação dos algoritmos, embora, obviamente, com a ajuda dos outros membros. A *GUI* foi feita pelo José Monteiro. A recolha de dados e análise dades foi feita pelo João Afonso. Todo os membros tiveram uma participação activa na realização do relatório.

Mais uma vez, foi um projecto equilibrado no que toca à distribuição de trabalho e tarefas.

9 - Conclusão

Após a análise destes valores e as vantagens e desvantagens de cada algoritmo, é possível concluir que os algoritmos recursivos, são sem dúvida mais eficientes em relação aos de brute force. Como seria de esperar, entre os dois algoritmos de brute force, o algoritmo “Brute Force Optimizado” obteve uma melhor performance, demorando 25% menos tempo em média a calcular os resultados.

Em relação à recursividade, nota-se que os resultados são um bocado mais aleatórios, variando entre segundos e vários minutos, para o mesmo número de cidades. Entre os algoritmos recursivos que possuímos, o algoritmo com greedy não é, ao contrário do esperado, o que consegue obter melhores resultados.

Olhando para os dois cenários, é também possível concluir que o cenários 2 em média foi o que consumiu menos tempo.

Outro ponto a realçar, é o facto de que o tempo de cálculo do melhor caminho aumentar consideravelmente, quanto maior for o número de cidades, devido à fatorialidade do cálculo. Isto acontece porque o número de caminhos possíveis é igual ao fatorial do número de cidades menos um (caminhos possíveis = [#cidades-1]!), e em especial, no algoritmo de brute force é preciso comparar todos os caminhos para efetivamente saber qual o melhor, este aumento do número de caminhos possíveis traduz-se no incremento do tempo de cálculo total.