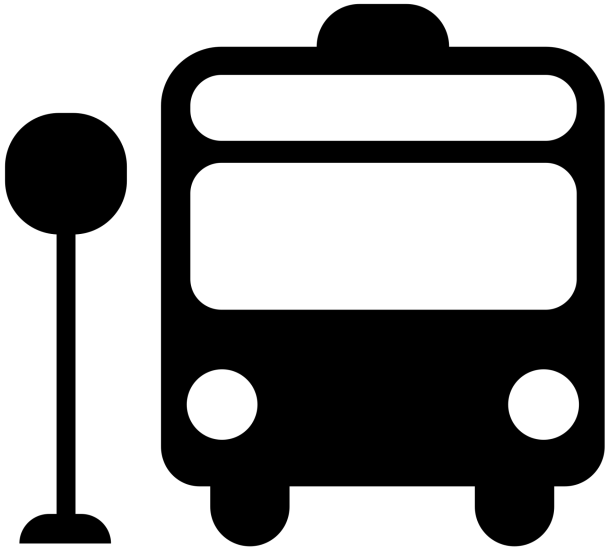


U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Agência de viagens

Grupo 80:

Alexandre Costa (up202005319)

Ana Beatriz Fontão (up202003574)

Ana Rita Oliveira (up202004155)

Desenho de Algoritmos - DA

Descrição do problema

O objetivo deste projeto é a criação de um sistema capaz de apoiar a gestão de pedidos para transporte de grupos de pessoas de um local de origem para um local de destino.

Os cenários propostos dividem-se em dois principais: se os grupos se podem dividir (cenário 1) ou não(cenário 2).

Foram fornecidos dez grafos acíclicos, pesados e dirigidos onde cada nó representa cada paragem e cada arco representa a viagem num autocarro de um nó para outro. Cada aresta é caracterizada por duração (da viagem) e capacidade (do autocarro).

Cenário 1.1 - Formalização

A solução ótima deste problema passa por maximizar a dimensão do grupo que vai passar na rede, sendo que o grupo não se pode separar. Isto significa que se pretende maximizar a capacidade da aresta com capacidade mínima no caminho do nó de origem ao nó de destino.
Consideremos:

- E, o conjunto de todas as arestas
- V, o conjunto de todos os nós
- P, caminho com as arestas desde o nó de origem ao nó de destino
- i, o nó que representa a paragem número i
- c(i, j), a capacidade do autocarro que vai da aresta i à paragem j

Desta forma a função objetivo é:

maximizar $\min(c(i, j)),\ (i,j) \in E, (i,j) \in P$

Cenário 1.1 - Algoritmos relevantes

Este cenário trata-se de um Widest Path Problem. Foi utilizado uma variação do algoritmo de Dijkstra para o resolver. Foi também criado um vetor auxiliar para guardar as capacidades mínimas máximas dos edges e uma min priority queue de pares (capacidade, nó de destino).

Essencialmente, o algoritmo utiliza a estrutura normal de um algoritmo de Dijkstra, mas a capacidade serve de bottle neck, no sentido de a capacidade máxima de um caminho é sempre igual a capacidade da aresta com menos capacidade.

Cenário 1.1 - Análise de Complexidade

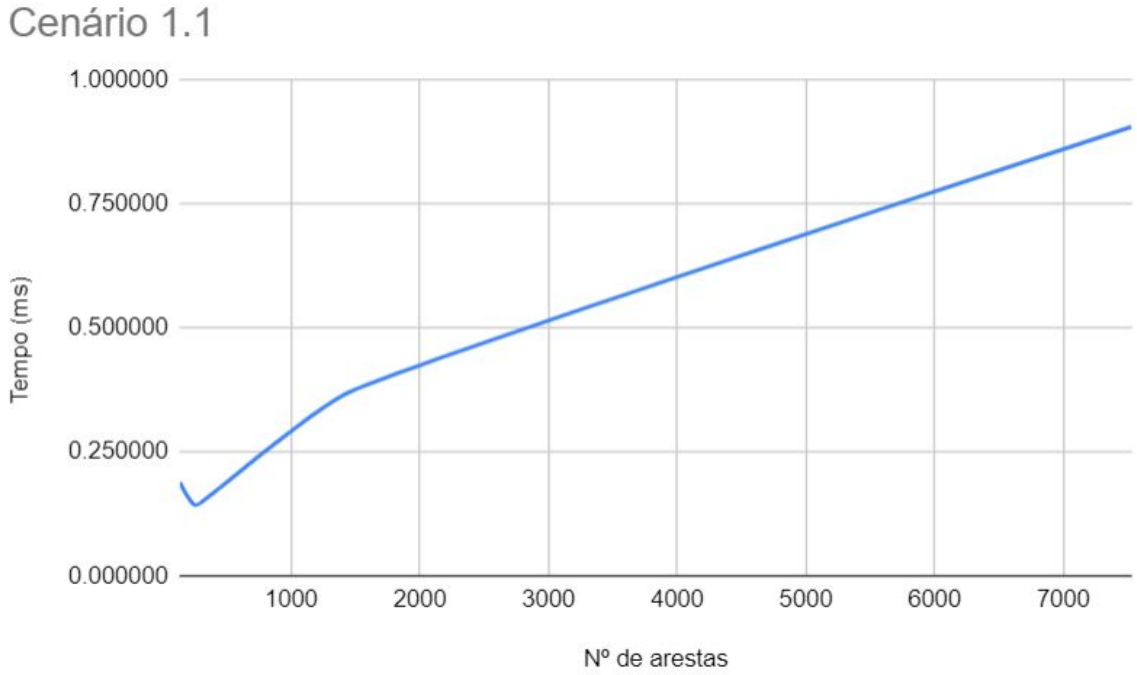
A complexidade espacial das funções desenvolvidas para este cenário é:

- widestPath: $O(V + E)$

A complexidade temporal é:

- widestPath: $O(E * \log(V))$

Cenário 1.1 - Resultados da avaliação empírica



Cenário 1.2 - Formalização

A solução deste problema passa por maximizar a dimensão do grupo que vai passar na rede, sendo que o grupo não se pode separar, e minimizar o número de transbordos, sem privilegiar nenhum dos dois. Isto significa que se pretende maximizar a capacidade da aresta com capacidade mínima no caminho do nó de origem ao nó de destino ou encontrar o caminho com o menor número de nós, podendo por esta razão haver soluções pareto-ótimas.
Consideremos:

- E, o conjunto de todas as arestas
- V, o conjunto de todos os nós
- P, caminho com as arestas desde o nó de origem ao nó de destino
- i, o nó que representa a paragem número i
- c(i, j), a capacidade do autocarro que vai da aresta i à paragem j

Cenário 1.2 - Algoritmos relevantes

Para maximizar a dimensão do grupo, foi utilizado o mesmo algoritmo do cenário 1.1, ou seja, uma variação do algoritmo de Dijkstra.

Para encontrar o caminho mais curto, foi utilizado uma BFS, não tendo em conta o peso/capacidade das arestas. Após obter o caminho mais curto, é calculada a capacidade mínima desse caminho.

Após obter estas duas soluções, elas são comparadas. Caso elas sejam pareto-ótimas, ou seja, uma não é necessariamente melhor que a outra, ambas são apresentadas, senão é apenas mostrada a melhor solução.

Cenário 1.2 - Análise de Complexidade

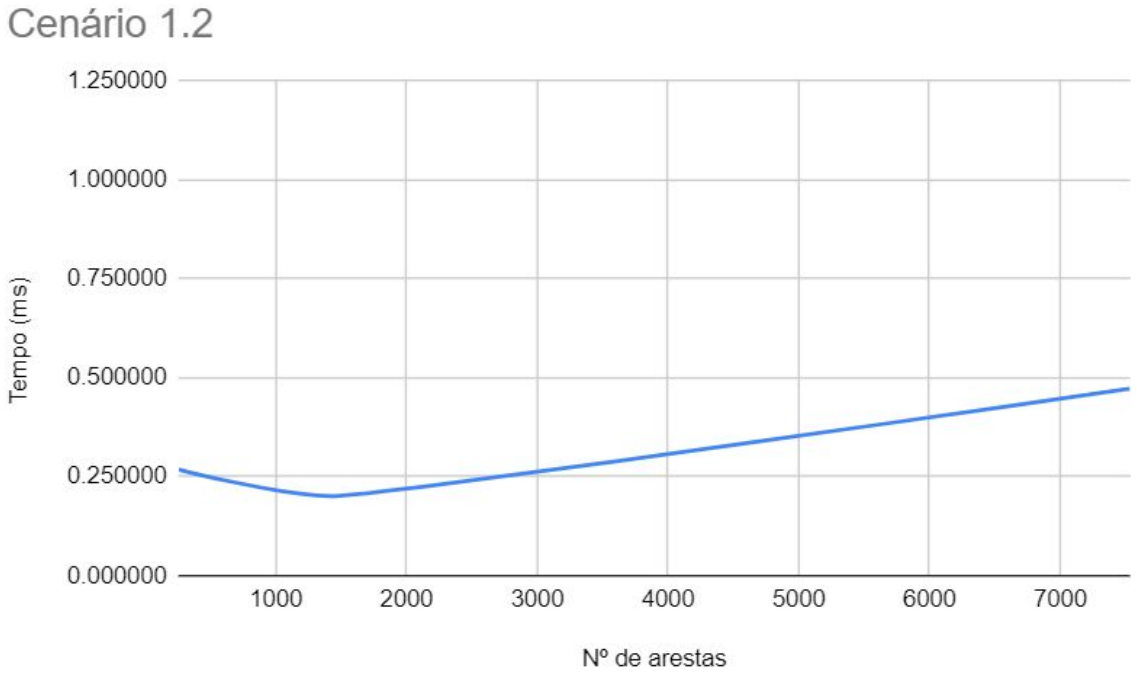
A complexidade espacial das funções desenvolvidas para este cenário é:

- widestPath: O(V + E)
- bfs: O(2V + E)

A complexidade temporal é:

- widestPath: O(E * log(V))
- bfs: O(V+E)

Cenário 1.2 - Resultados da avaliação empírica



Cenário 2.3 - Formalização

A solução ótima deste problema passa por maximizar o fluxo numa rede. No nosso cenário isso significa maximizar a quantidade de pessoas que podem viajar naquela rede.
Consideremos:

- E é o conjunto de todas as arestas
- i, o nó que representa a paragem número
- c(i, j), a capacidade do autocarro que vai da aresta i à paragem j
- f(i, j), o fluxo (i.e. a quantidade de pessoas que vão no autocarro) da aresta i à j
- $0 \leq f(i, j) \leq c(i,j), (i, j) \in E$

Desta forma a função objetivo é:

maximizar $|f| = \sum f(i, j)\ , \forall\ (i, j) \in E$

Cenário 2.3 - Algoritmos relevantes

Para resolver este cenário foi utilizado o algoritmo de Edmonds-Karp, que passa por:

- Aplicar a BFS (Breadth First Search) para encontrar o caminho de nó de origem Src até a um nó de destino Target.
- Atualizar o fluxo de cada aresta, respeitando sempre que $f(i, j) \leq c(i,j), (i, j) \in E$
- Atualizar o fluxo máximo $|f|$, sendo que este começa na 1ª iteração a 0
- Ir iterando os três passos anteriores até não haver mais caminhos de Src até Target, obtendo assim $|f|$

Para este algoritmo foi necessário fazer uma função BFS que tivesse em conta a restrição $f(i, j) \leq c(i,j), (i, j) \in E$, e ignorasse arestas que estivessem saturadas ($f(i, j) = c(i,j)$).

Cenário 2.3 - Análise de Complexidade

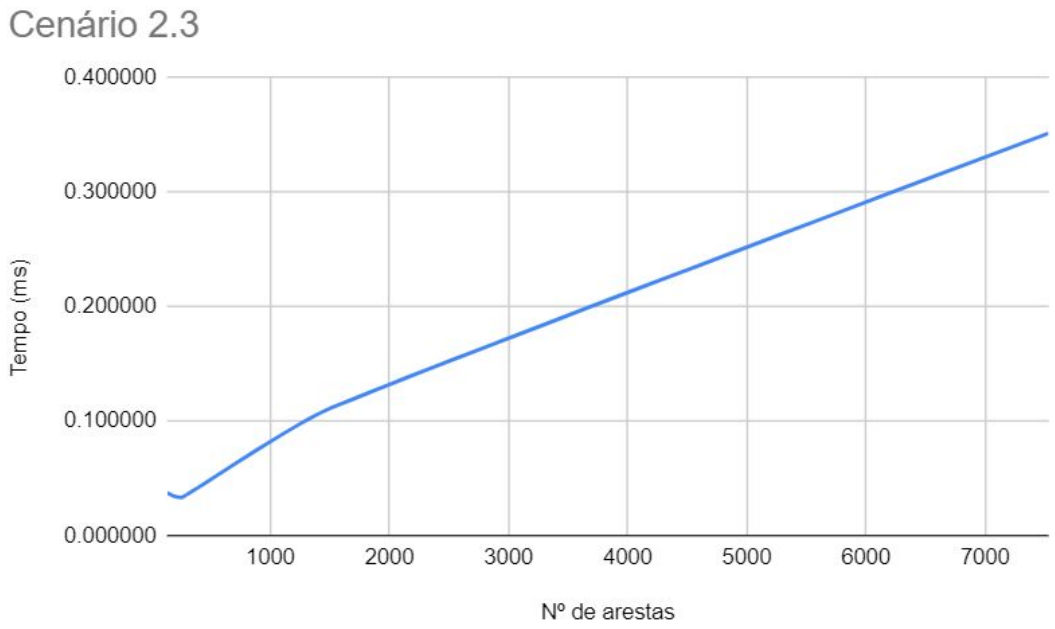
A complexidade espacial das funções desenvolvidas para este cenário é:

- edmondsKarp (com EK): O(V + E)
- updateFlow: O(V * E)
- bfsFF: O(2V + E)

A complexidade temporal é:

- edmondsKarp (com EK): $O(V * E^2)$
- updateFlow: O(V + E)
- bfsFF: O(E + V)

Cenário 2.3 - Resultados da avaliação empírica



Cenário 2.1 e 2.2 - Formalização

Os cenários 2.1 e 2.2, a nível de formalização e de algoritmos são semelhantes ao 2.3. É necessário apenas ter algumas considerações extra:

- O utilizador fornece um nº de pessoas para viajar na rede, o **groupSize**
- $|f| \leq groupSize$
- O utilizador pode aumentar o groupSize em quantas unidades quiser, desde que nunca ultrapasse o fluxo máximo.

Desta forma, a função objetivo é:

$$|f| = groupSize \text{ , sendo que } |f| = \sum f(i, j) \text{ , } \forall \ (i, j) \in E$$

Caso o groupSize fornecido seja superior ao fluxo máximo, o fluxo máximo é apresentado.

Cenário 2.1 e 2.2 - Algoritmos relevantes

Tal como no cenário 2.3, foi utilizado o algoritmo de Edmonds-Karp, mas desta vez com uma pequena alteração: em vez de se iterar até não haver mais caminhos possíveis, itera -se até ao fluxo ser maior ou igual ao groupSize.

Caso o fluxo seja maior que o groupSize, ao atualizar o fluxo das arestas do caminho calculado pela BFS, aumenta-se o fluxo em groupSize - |f|, de forma a que o fluxo máximo seja igual ao groupSize.

À medida que o utilizador decide incrementar o groupSize, o algoritmo é utilizado outra vez até o utilizador decidir não aumentar mais o groupSize ou o groupSize ser maior ou igual que o fluxo máximo.

Cenário 2.1 e 2.2 – Análise de Complexidade

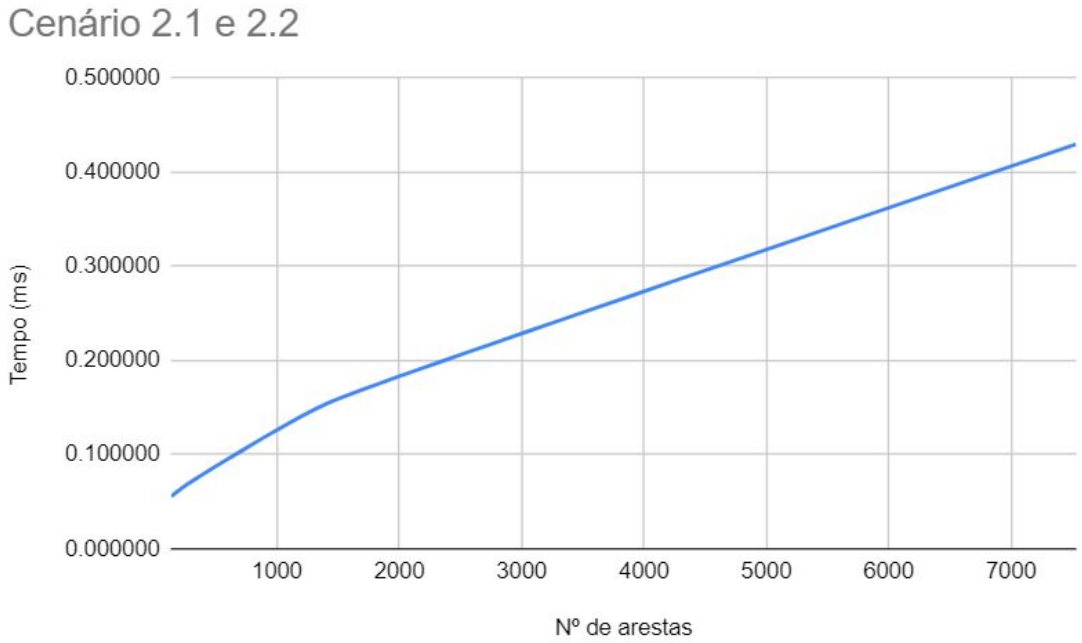
A complexidade espacial das funções desenvolvidas para este cenário é:

- edmondsKarp (com EK): O(2V + E)
- updateFlow: O(V * E)
- bfsFF: O(2V + E)

A complexidade temporal é:

- edmondsKarp (com EK): $O(V * E^2)$
- updateFlow: O(V + E)
- bfsFF: O(E + V)

Cenário 2.1 e 2.2 – Resultados da avaliação empírica



Cenário 2.4 - Formalização

A solução ótima para este problema passa por obter a duração mínima do caminho do primeiro até ao último nó. A duração mínima é igual ao comprimento do caminho máximo. O comprimento de um caminho é dado por $\sum dij$, $(i, j) \in A$, sendo A o conjunto de arcos que formam o caminho.

Sabemos que o Earliest Start time de i é a data de início mais próxima para a tarefa i. Este valor é igual ao valor máximo de Earliest Finish (data de conclusão mais próxima para a tarefa) dos seus precedentes. Estes valores são obtidos através das seguintes expressões:

$ESij = \max\{ EFki \mid (k, i) \in A\}$, A é o conjunto de arcos do grafo

$EFij = ESij + dij$, d representa a duração do arco (i, j)

Neste cenário consideramos ES[i] = ESij e ES1j = 0 (1 é o acontecimento de início de projeto)

Cenário 2.4 - Algoritmos relevantes

Para este problema utilizámos o método do caminho crítico. Foram usadas como estruturas de dados auxiliares: uma fila S, inicialmente vazia, um vetor GrauE, onde é guardado o número de arcos com destino no nó (dado pelo índice), e finalmente um vetor earliestStart, inicializado a 0, onde são guardados os valores de earliest start de cada nó.

Com base no tamanho de S, criámos um ciclo onde obtemos o earliest start de todos os nós somando o earliest start do nó precedente e a duração do arco que os liga. Este valor pode ser alterado caso exista outro arco com destino no mesmo nó cujo earliest start seja maior.

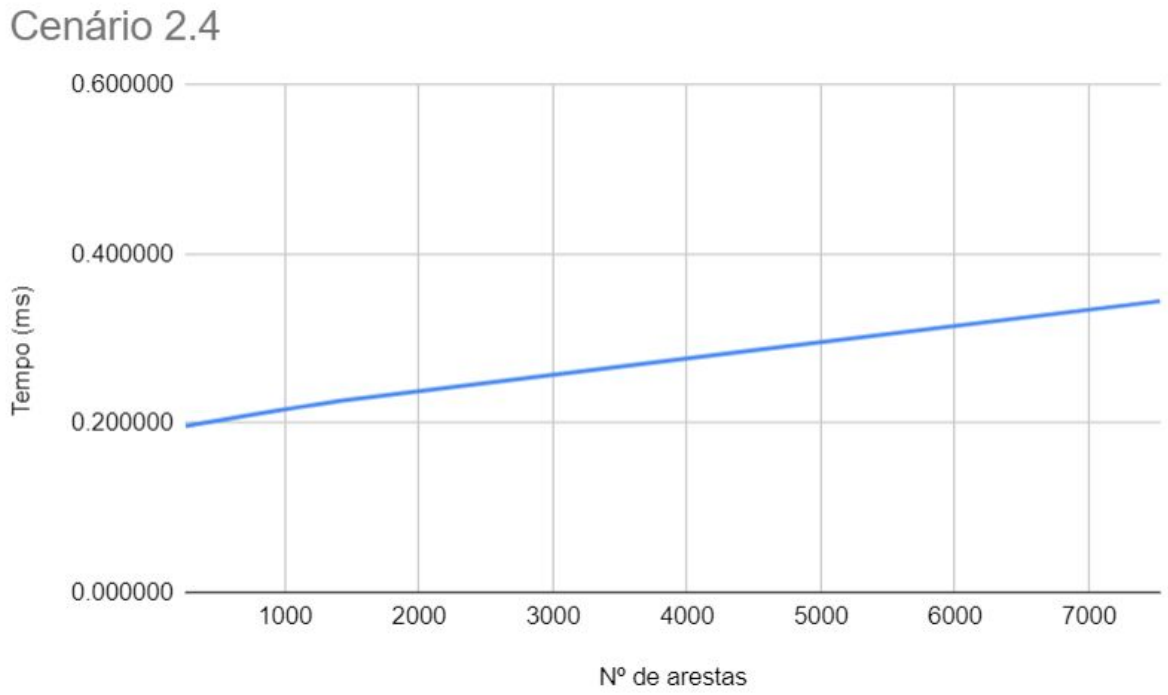
Cenário 2.4 - Análise de Complexidade

A complexidade espacial da função desenvolvida para este cenário é O(3 * V).

A complexidade temporal é O(3 * V + E + V) em que E representa o número de arestas do grafo e V é o número de vértices.

Como é possível observar no próximo slide, foram feitas verificações da complexidade temporal usando os dados fornecidos.

Cenário 2.4 – Resultados da avaliação empírica



Cenário 2.5 - Formalização

Neste problema, nós interpretámos o tempo máximo de espera como a maior folga total(FT) existente e os locais como todos os nós cuja folta total fosse diferente de 0.

Sabemos que o Latest Finish time de i é a data de conclusão mais afastada da tarefa i. Este valor é igual ao valor mínimo de Latest Start (data de início mais afastada para a tarefa) dos seus sucessores. Estes valores são obtidos através das seguintes expressões:

$LFij = \min \{ LSjk \mid (j, k) \in A\}$, A é o conjunto de arcos do grafo

$LSij = LFij - dij$, d representa a duração do arco (i, j)

$FTij = LSij - ESij = LFij - ESij = LFij - ESIJ - dij$

Neste cenário consideramos LF[J] = LFij e LFjn = duração mínima (n é o acontecimento fim de projeto)

Cenário 2.5 - Algoritmos relevantes

Para este problema utilizámos também o método do caminho crítico. Foram criadas três funções para a resolução deste cenário: latestFinish, que retorna um vetor com os valores de Latest Finish de cada nó(dado pelo índice); transposeGraph que transpõe o grafo e maxWaitingTime, onde obtemos as folgas totais.

A primeira função é relativamente semelhante à função elaborada para o cenário anterior sendo que em vez de calcular os earliest starts, calcula os latest finishes.

A terceira função usa as duas funções anteriores e a função do cenário anterior e visa calcular as folgas totais dos nós do grafo. Aqui, a folga máxima e todos os nós com folgas totais maiores que 0 são apresentadas ao utilizador.

Cenário 2.5 - Análise de Complexidade

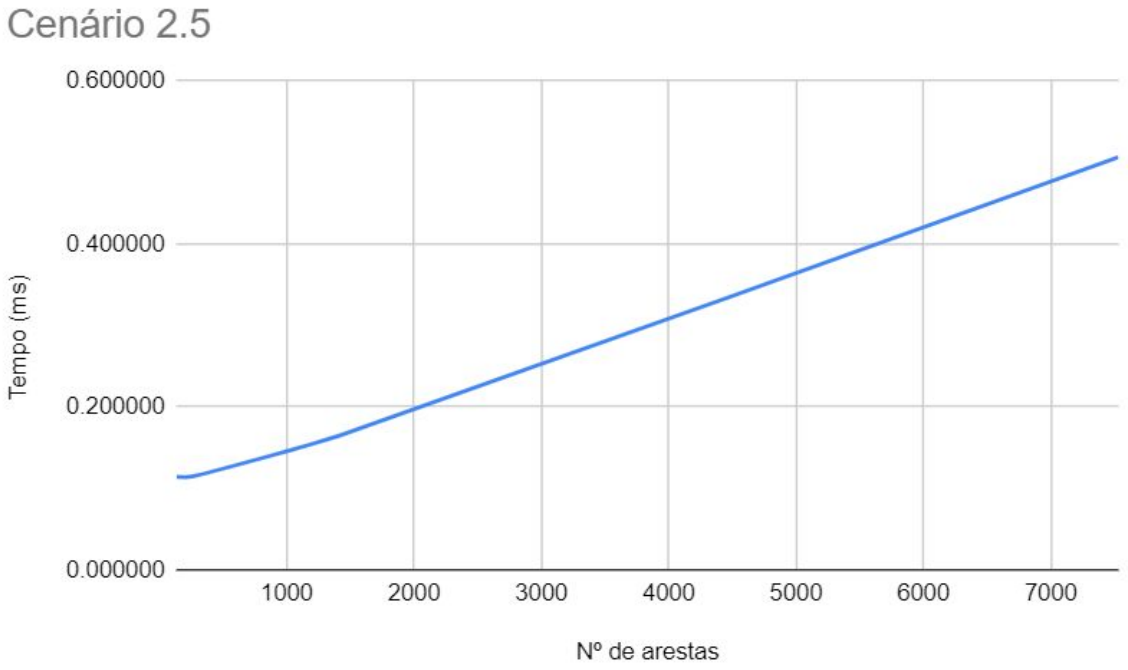
A complexidade espacial das funções desenvolvidas para este cenário são:

- latestFinish : O(3 * V + n)
- transposeGraph : O(n) (n é o tamanho do próprio grafo)
- maxWaitingTime: O(3*V)

A complexidade temporal é:

- latestFinish : O(3 * V + E + V)
- transposeGraph : O(E * V)
- maxWaitingTime: O(E * V)

Cenário 2.5 - Resultados da avaliação empírica



Avaliação e dificuldades do grupo

Dificuldades:

- Melhorar algoritmos, por exemplo, em termos de complexidade temporal
- Adaptar algoritmos conhecidos de modo tornarem-se aplicáveis aos nossos dados e objetivos

Distribuição do trabalho:

- Alexandre Costa - 33.3%
- Ana Beatriz Fontão - 33.3%
- Ana Rita Oliveira - 33.3%