Uma imagem com texto

Descrição gerada automaticamente

**Speed Run**

Algoritmos e Estruturas de Dados 2022

Professor Tomás Oliveira e Silva

João Nuno da Silva Luís (107403)

Diana Raquel Rodrigues Miranda (107457)

**Índice**

[Introdução 3](#_Toc119593853)

[Solução fornecida 4](#_Toc119593854)

[Segunda Solução 13](#_Toc119593855)

# **Introdução**

# **Solução fornecida**

A solução fornecida segue o conceito de depth first search, que é um algoritmo utilizado para realizar uma procura numa árvore, estrutura de árvore ou grafo. Intuitivamente, o algoritmo começa num nó raiz e explora tanto quanto possível cada um dos seus ramos, antes de retroceder.

Posto isto, temos no nosso caso de estudo (Speed run) o nó raiz como a primeira posição de onde o carro irá arrancar e desse nó irão sair três novos nós, um com a opção de aumentar a velocidade, um com a opção de a manter e outro com a opção de a diminuir, e, só depois de todas as possibilidades terem sido percorridas, é que o algoritmo vai retroceder e escolher o melhor caminho.

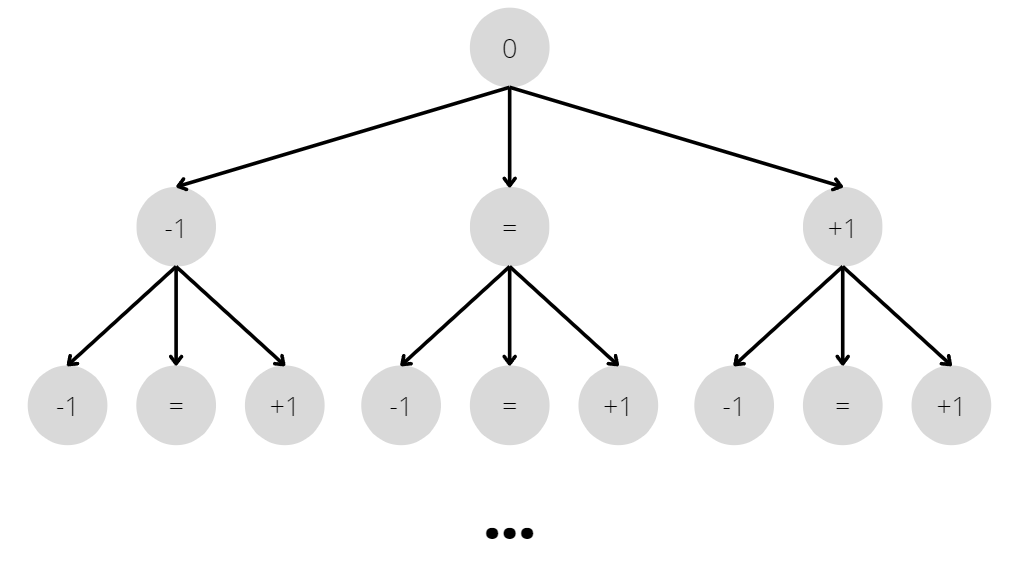
Esta função apesar de chegar a uma solução correta, tem um nível de complexidade alto, não conseguindo ser possível resolver o problema, num tempo consideravelmente reduzido, quando temos uma estrada com bastantes segmentos, ou seja, um número elevado de posições, já que o algoritmo vai testar também um número elevado de nós.

Figura 1 – Exemplo gráfico da árvore percorrida pela função fornecida.

O nosso método para tornar esta solução mais eficiente foi pensar numa maneira de otimizar a pesquisa em árvore e diminuir o número de ramos visitados.

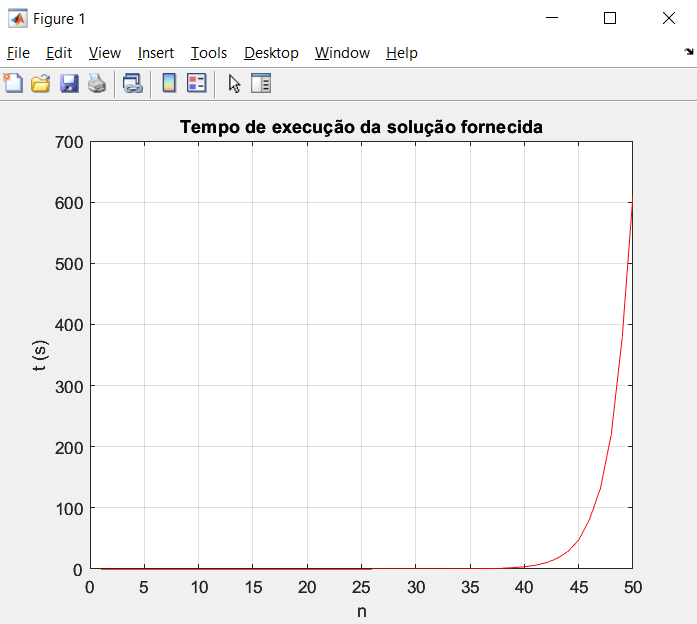
 Fazendo o gráfico do tempo de execução da solução fornecida, obtemos um gráfico exponencial a partir da posição 40.

Figura 2- Tempo de execução da solução fornecida

Este tipo de gráfico não é o melhor para visualizarmos os nossos dados, nem calcular a reta de ajuste, pelo que podemos aplicar um logaritmo na base dez ao eixo dos yy (eixo dos tempos), e ao mesmo tempo calcular a reta de ajuste aos dados obtidos, que nos permite fazer uma previsão do tempo de execução deste algoritmo até à posição final.

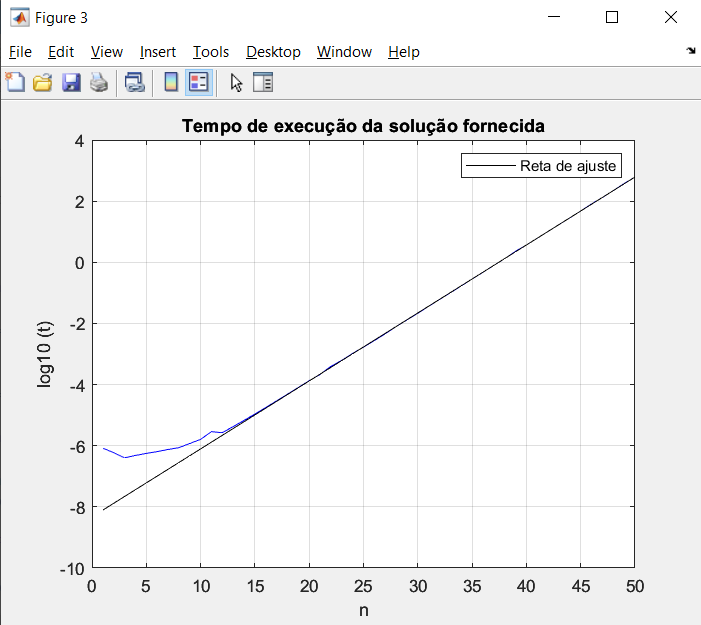
Através desta resta de ajuste, calculamos que a solução fornecida iria demorar 1.114e+162 anos a chegar à posição 800.

Figura 3- Reta de ajuste aos dados da solução fornecida

**1ª melhoria** – Tentar acelerar primeiro.

Como o nosso objetivo é chegar à posição final o mais rápido possível, isto é, com o menor número de saltos, vamos tentar, se for permitido, acelerar sempre.

Uma imagem com texto

Descrição gerada automaticamentePara isso, alterámos o seguinte pedaço de código:

Uma imagem com texto

Descrição gerada automaticamentePara:

Assim, em vez de começar por tentar desacelerar, vai começar por verificar se pode acelerar.

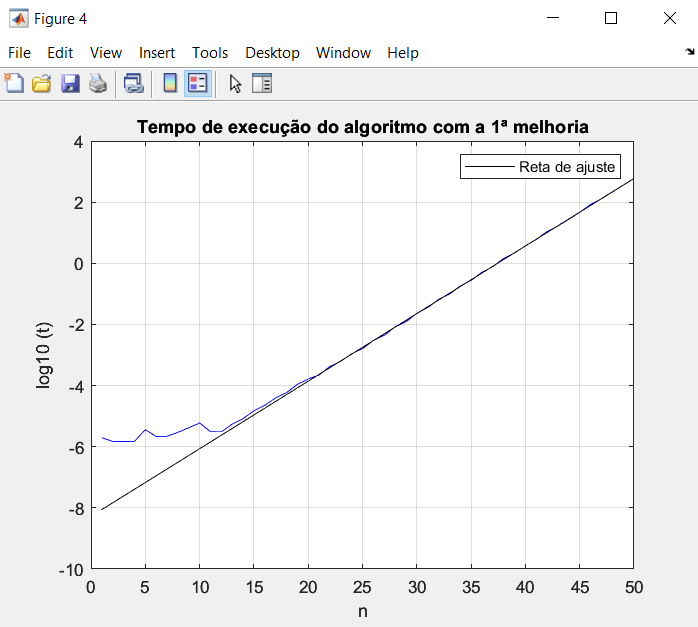
Com o código fornecido e em execução durante uma hora (com o número mecanográfico 107457) era possível chegar à posição 50 em 6.121e+02 segundos, com esta alteração foi possível, durante o mesmo tempo de execução, chegar à posição 50 em 5.935e+02 segundos. Apesar de ser uma melhoria mínima, já é algo a considerar.

Figura 4- Reta de ajuste aos dados da 1ª melhoria

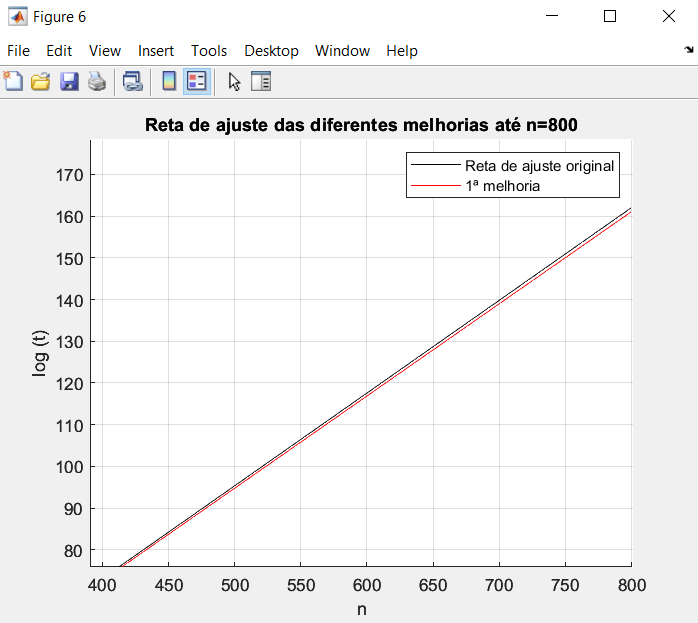
****

Figura 5- Comparação da reta de ajuste até à posição n=800

**2ª melhoria** – Acrescentar um if no código da função fornecida.

Uma imagem com texto

Descrição gerada automaticamente Este if permite que se vá cortando vários ramos que já não têm interesse em ser vistos, já que o que ele faz é verificar se o número de movimentos da solução que está a correr naquele momento é maior do que o número de movimentos já antes guardado como melhor solução, e, se for maior, então podemos parar de ver essa solução, pois já não nos interessa uma vez que já temos uma solução melhor encontrada anteriormente.

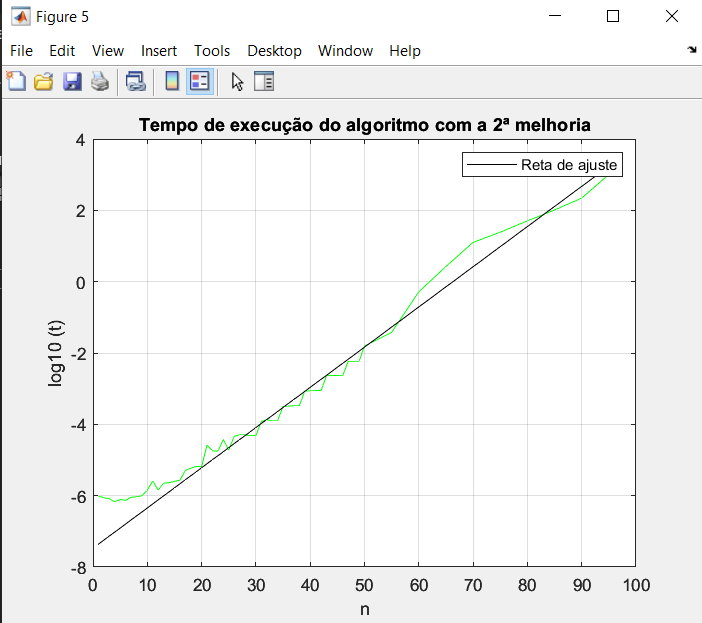
 Apesar de esta melhoria ser bastante simples, pois só acrescentámos duas linhas de código, é uma melhoria que nos permite reduzir o tempo de execução para metade. Com isto, durante uma hora de execução (com o número mecanográfico 107457), já conseguimos chegar à posição 95 em 1.009e+03 segundos.

Figura 6- Reta de ajuste aos dados da 2ª melhoria

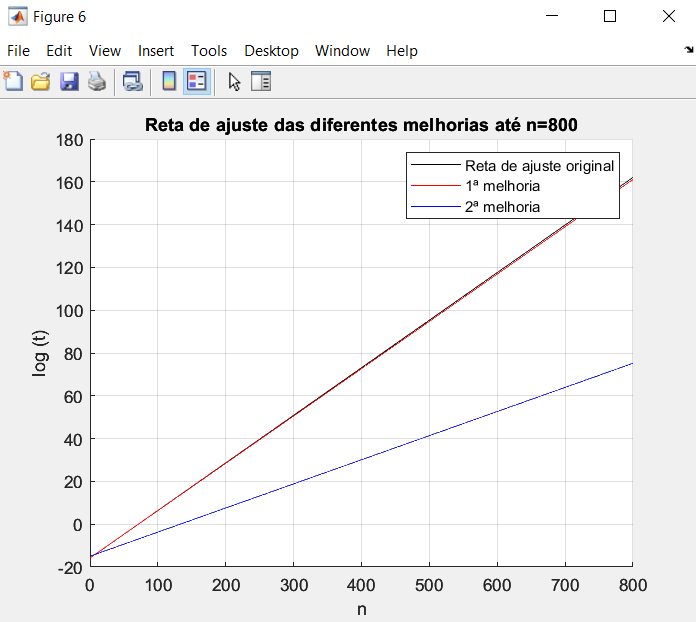


Figura 7- Comparação das diferentes retas de ajuste.

A 2º melhoria permite reduzir o tempo de execução para metade.

**3ª melhoria** – Acrescentar um outro if no código da função fornecida.

Uma imagem com texto

Descrição gerada automaticamente

Neste segundo if, vai ser verificado se numa determinada posição foi possível, na solução já guardada, passar com uma velocidade maior do a que está a ser vista naquele momento. Se tiver sido possível então podemos abandonar a pesquisa desse ramo, pois interessa-nos andar sempre com a velocidade máxima possível.

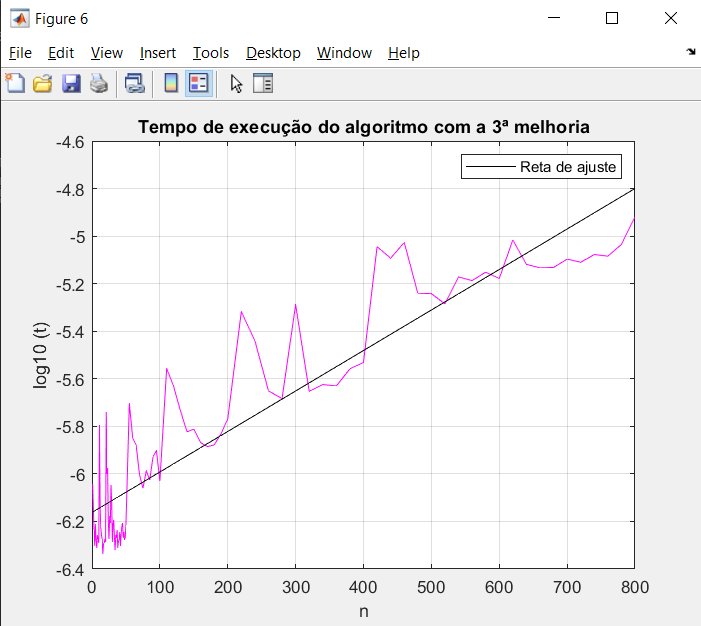
 Esta melhoria, também sendo simples, torna possível, juntamente com as outras melhorias resolver o problema para as 800 posições em menos de um segundo (com o número mecanográfico 107457).

Figura 8-Reta de ajuste aos dados da 3ª melhoria

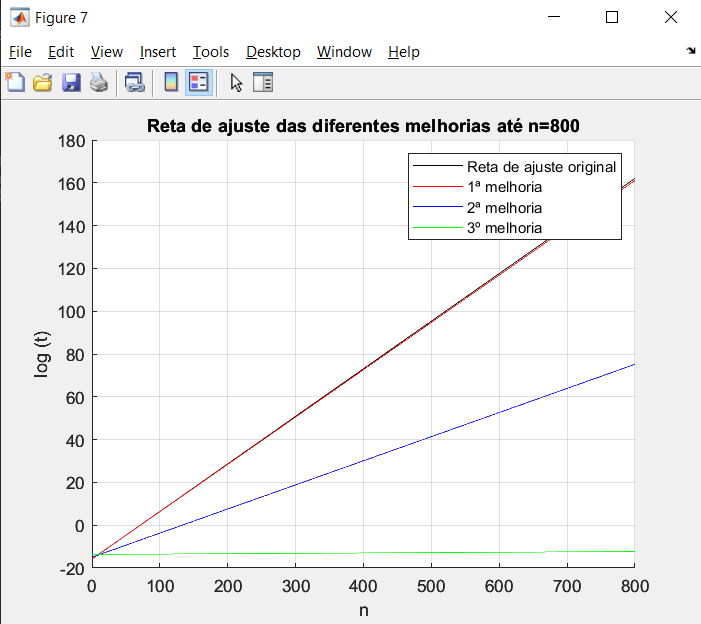


Figura 9- Comparação das diferentes retas de ajuste.

A 2ª e a 3ª melhoria foram baseadas no algoritmo de Branch and Bound, uma vez que a sua função é descartar logo um ramo se essa solução já for pior que a solução já guardada.

# **Segunda** **Solução**

Esta solução foi criada pensando numa maneira de descobrir a melhor solução para o problema percorrendo a estrada toda uma única vez.

Posto isto, começamos por criar um ciclo while que permitisse percorrer toda a estrada, dentro desse ciclo temos um if que vai verificar se é possível aumentar, diminuir ou manter a velocidade. Para conseguir fazer uma verificação que garantisse que em nenhum momento se iria desrespeitar as regras da estrada, ou seja, que em nenhum momento se ia exceder a velocidade, criamos uma função (respect\_limits) que tem como parâmetros de entrada a posição onde se encontra, a velocidade a que quer seguir e a posição final da estrada. O objetivo desta é verificar se a velocidade a que está a tentar seguir é válida, para essa avaliação ele verifica se com essa velocidade teria tempo de travar se já se encontrasse perto do fim da estrada e verifica, também se essa velocidade respeita a velocidade de todos os segmentos por onde vai passar até chegar à seguinte posição, se a velocidade cumprir estes dois requisitos a função vai retornar o valor 1, caso contrário retorna o valor 0.

Com esta solução temos que o tempo de execução para a resolução do problema é 2.795e-06 segundos (com o número mecanográfico 107457).

\*gráfico\*

# **Terceira Solução**

Para esta solução, utilizámos a programação dinâmica, que consiste em dividir um problema de otimização em subproblemas mais simples e guardar a solução para cada subproblema de modo que a cada subproblema seja resolvido só uma vez.

No contexto do problema estudado, o que começámos por pensar foi, por exemplo, numa estrada com 10 segmentos os saltos que o carro vai dar até começar a travar, por já estar perto do fim da estrada, vão ser os mesmo que numa estrada com 20 segmentos até essa posição, e assim em diante.

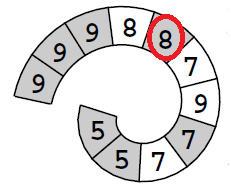
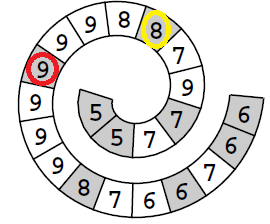
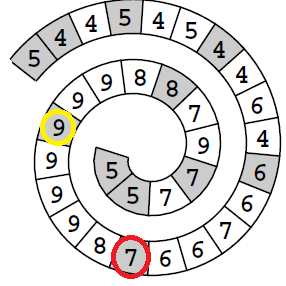


Figura 10 - 30 segmentos

Figura 11 - 20 segmentos

As figuras acima demonstram a ideologia desta solução, as posições com o círculo vermelho são as posições onde o carro começa a travar porque já se encontra perto do fim da estrada, vai então ser guardado num array as posições

Figura 10 - 10 segmentos

Com este pensamento, criámos um algoritmo que segue a mesma ideia do algoritmo da solução 2, mas neste é gu

Resultados

Conclusões finais

Web Grafia (se usarmos)

Código C

(apenas o que foi alterado)

Código MATLAB