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](#_Toc119611714)

[Solução fornecida 4](#_Toc119611715)

[Segunda Solução 13](#_Toc119611716)

[Terceira Solução 15](#_Toc119611717)

# **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.

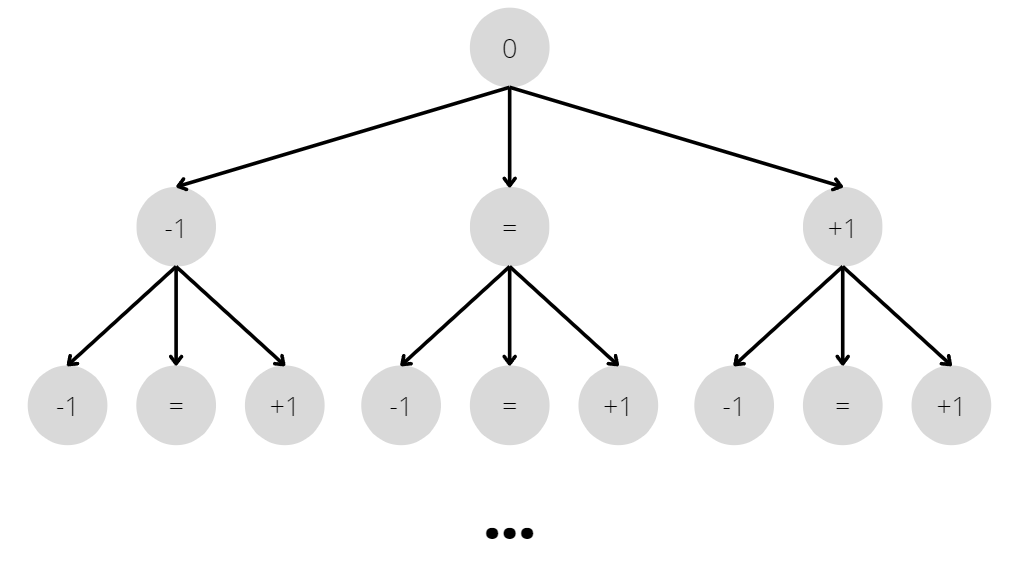
Esta função apesar de chegar a uma solução correta, tem um nível de complexidade alto, não conseguindo ser possível resolver o problema, num tempo consideravelmente reduzido, quando temos uma estrada com bastantes segmentos, ou seja, um número elevado de posições, já que o algoritmo vai testar também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

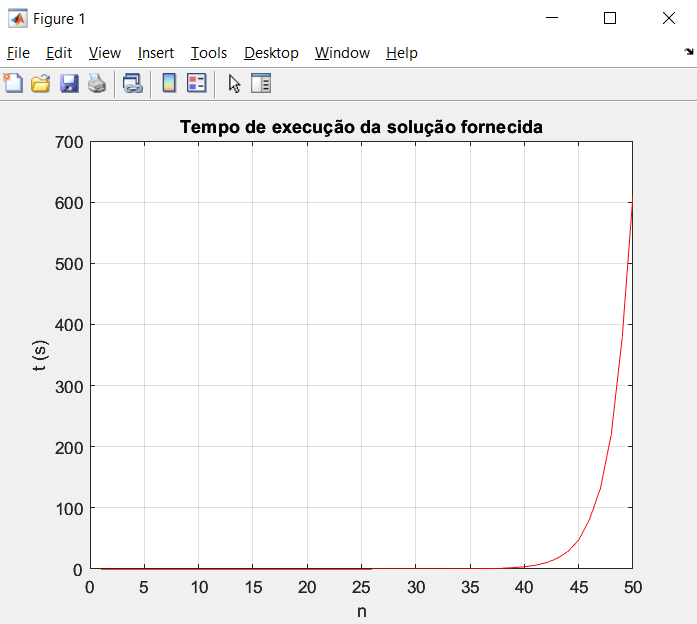
 Fazendo o gráfico do tempo de execução da solução fornecida, obtemos um gráfico exponencial a partir da posição 40.

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.

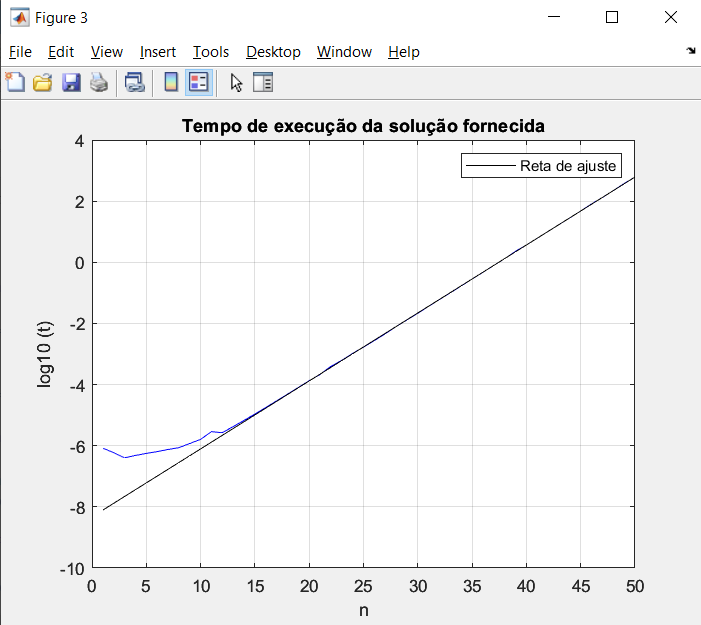
Através desta resta de ajuste, calculamos que a solução fornecida iria demorar 1.114e+162 anos a chegar à posição 800.

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.

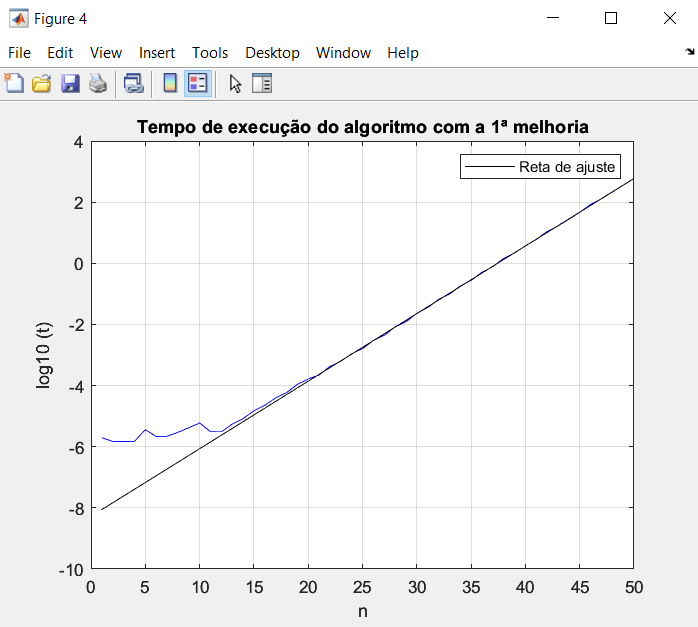
Com o código fornecido e em execução durante uma hora (com o número mecanográfico 107457) era possível chegar à posição 50 em 6.121e+02 segundos, com esta alteração foi possível, durante o mesmo tempo de execução, chegar à posição 50 em 5.935e+02 segundos. É uma melhoria miníma, pois mesmo a tentar acelerar primeiro a função vai na mesma verificar todos os ramos da árvore possíveis.

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

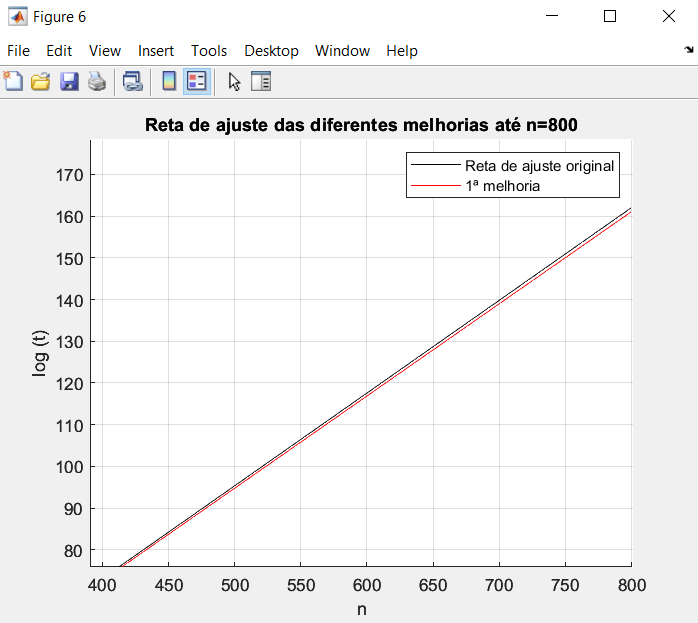
****

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.

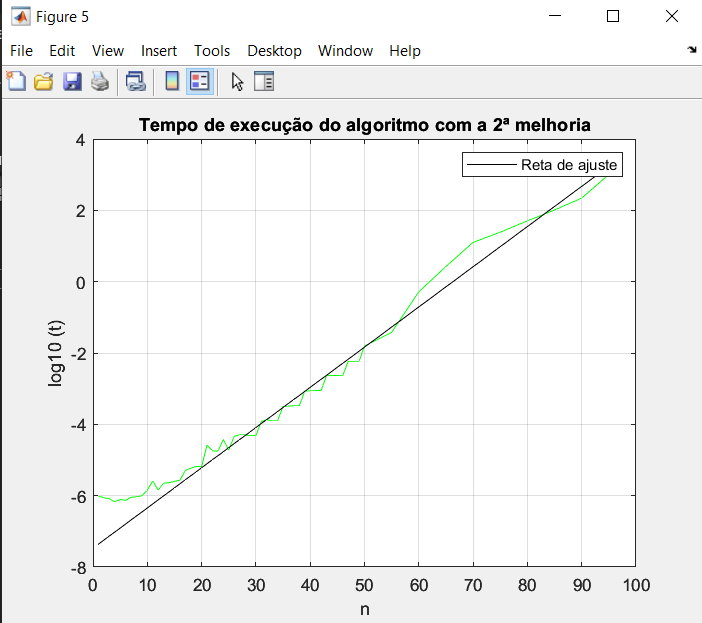
 Apesar de esta melhoria ser bastante simples, pois só acrescentámos duas linhas de código, é uma melhoria que nos permite reduzir o tempo de execução para metade. Com isto, durante uma hora de execução (com o número mecanográfico 107457), já conseguimos chegar à posição 95 em 1.009e+03 segundos.

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

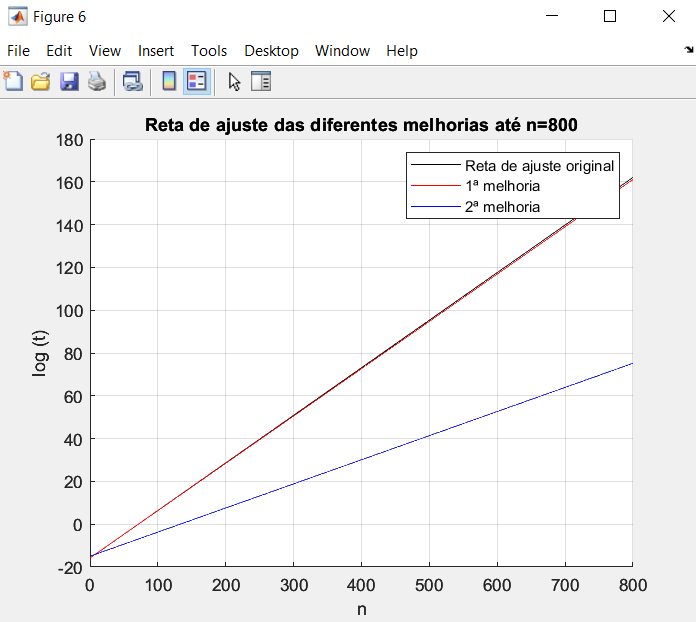


Figura 7- Comparação das diferentes retas de ajuste.

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.

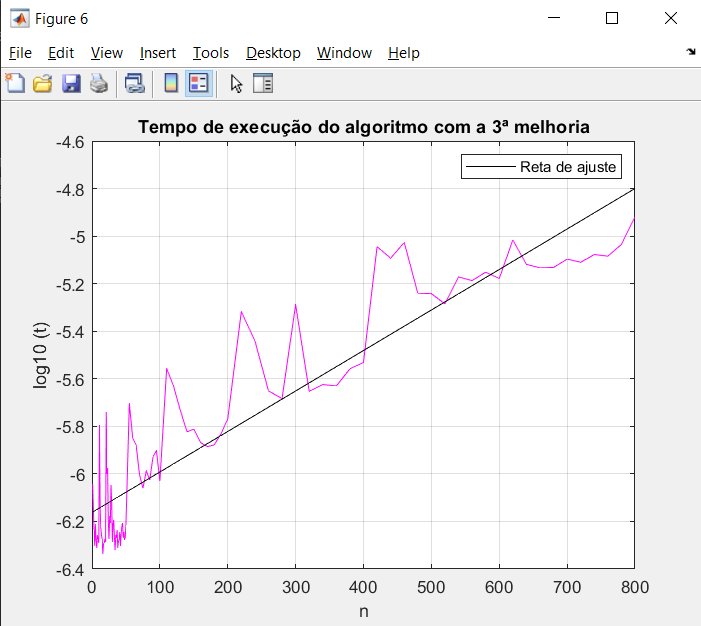
 Esta melhoria, também sendo simples, torna possível, juntamente com as outras melhorias resolver o problema para as 800 posições em menos de um segundo (com o número mecanográfico 107457).

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

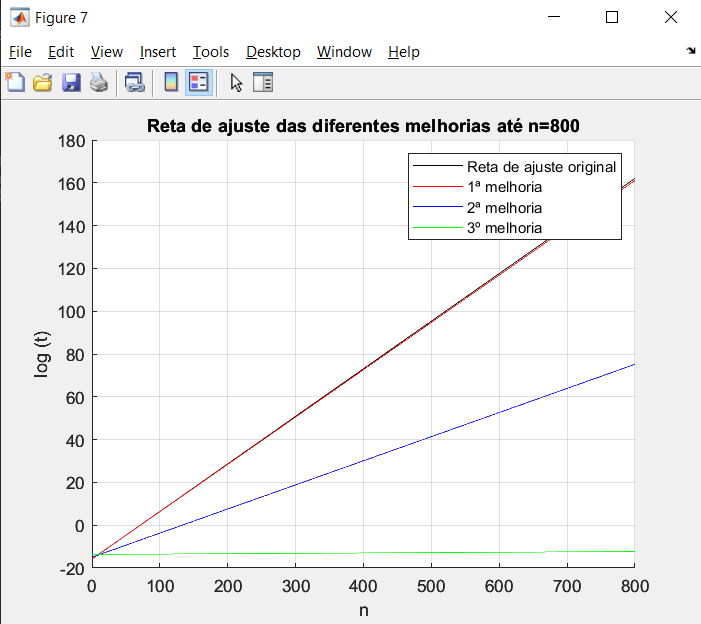


Figura 9- Comparação das diferentes retas de ajuste.

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.

Uma imagem com texto

Descrição gerada automaticamente

Figura 10 - Função da Solução 2

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, criámos uma função (respect\_limits) que tem como parâmetros de entrada a posição onde se encontra, a velocidade a que está a tentar seguir e a posição final da estrada. O objetivo desta é verificar se a velocidade que está a tentar seguir é válida, e, para essa avaliação, o algoritmo 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 posição seguinte.

Se a velocidade cumprir estes dois requisitos a função vai retornar o valor 1, caso Uma imagem com texto

Descrição gerada automaticamentecontrário retorna o valor 0.

Figura 11 - Função usada na solução 2 para verificar a velocidade

Com esta solução temos que o tempo de execução para a resolução do problema é 2.815e-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, de modo que cada subproblema seja resolvido só uma vez.

No contexto do problema em estudo, 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.

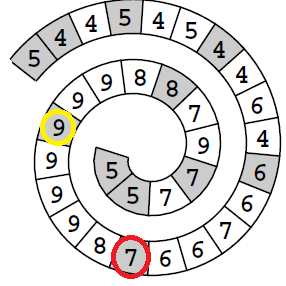
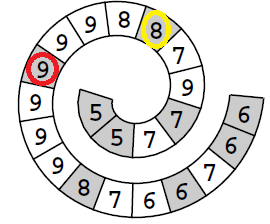
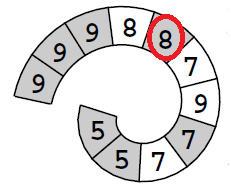


Figura 12 - 10 segmentos

Figura 14 - 20 segmentos

Figura 13 - 30 segmentos

As figuras acima demonstram o descrito anteriormente descrito, a posição marcada a vermelho é a posição onde o carro começa a travar, por já se encontrar perto do fim, e a amarelo está marcado a posição onde o carro parou e que são iguais às já pesquisadas anteriormente.

Com este pensamento, criámos um algoritmo que segue a mesma ideia do algoritmo da solução 2, mas neste é criado um array com as posições onde o carro parou até à posição onde começa a travar. Por se encontrar perto do fim da estrada, é guardado também a velocidade com que ia nessa posição. Assim, na próxima pesquisa é reaproveitado esse array e em vez da pesquisa começar no inico da estrada com velocidade 0, a pesquisa é iniciada na última posição desse array e com a velocidade guardada, uma vez que até aí as posições de paragem vão ser sempre as mesmas, o que torna a procura mais rápida e eficiente, já que não está a fazer duas vezes o Uma imagem com texto

Descrição gerada automaticamentemesmo trabalho.

Para esta solução foi criada uma estrutura nova (solution3\_t) para conseguir guardar o número de saltos, a velocidade e a posição, e poder assim usar esses valores quando necessário.

Figura 15 - Estrutura criada para a solução 3

Foi também reaproveitado o código da solução 2, fazendo só algumas alterações, que são:

Uma imagem com texto

Descrição gerada automaticamente- Dentro do while foi acrescentado um if que só é executado quando o carro chega a uma posição com uma velocidade onde tem de começar a reduzir para respeitar os limites até ao fim da estrada. Este if é executado apenas uma vez.

Figura 16 - Função da Solução 3

Uma imagem com texto

Descrição gerada automaticamente- Na função que verifica os limites de velocidade foram alterados os valores de retorno. Como, para esta solução, é necessário saber exatamente quando é que a verificação falha por se ultrapassar a posição final da estrada é retornado 1 quando isso acontece. De seguida é verificado se não é ultrapassado o limite de velocidade de nenhum segmento de estrada por onde passa, se for retorna 2. Por fim, se a velocidade passar estas duas verificações o valor de retorno é 0.

Figura 17 - Função usada na solução 3 para verificar a velocidade

Uma imagem com texto

Descrição gerada automaticamente Alterámos também a função que executa a função da solução 3, para chamar a solução com o número de saltos, a posição e a velocidade guardadas na estrutura da solução 3.

Figura 18 - Função que chama a função da solução 3

Com esta solução temos que o tempo de execução para a resolução do problema é 6.310e-07 segundos (com o número mecanográfico 107457).

\*gráficos\*

Resultados

Conclusões finais

Web Grafia (se usarmos)

Código C

(apenas o que foi alterado)

Código MATLAB