



**UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL REI**

**Lidimar dos Santos Junior**

**Higor Henrique Alves**

**Algoritmos e Estrutura de Dados III**

**2º trabalho prático**

São João del Rei – MG

2016

# Introdução

O objetivo desse trabalho, é criar um programa que gere configurações para um dado estacionamento, levando em conta a localização dos veículos e qual o menor caminho para o veículo Z sair do estacionamento com sucesso. O estacionamento é representado por uma matriz  $m \times m$ , sendo que os veículos não podem fazer curva. Há dois tipos de veículos, carros, com dimensões 2 por 1 e caminhões com dimensões 3 por 1. O problema é dividido em duas partes. Na primeira parte, o programa deve receber como entrada a configuração do estacionamento e um conjunto de manobras pré-estabelecidas. Na segunda parte, o programa deve receber como entrada a configuração do estacionamento e desenvolver efetivamente as manobras necessárias para a saída do carro Z de duas formas distintas: por tentativa e erro, onde o programa deve visitar sistematicamente todas as possíveis opções de movimento até chegar ao objetivo, e por uma heurística, onde não precisa necessariamente se chegar a uma solução para o problema. Os principais requisitos do programa são:

- A matriz deve ser 6 x 6;
- Não pode haver veículos nas diagonais;
- Deve haver detecção e alerta de configurações físicas impossíveis (por exemplo dois veículos ocupando o mesmo espaço e manobras inviáveis (colisão entre veículos e com os muros do estacionamento));
- Os veículos só podem se mover nos eixos X e Y.

## Especificações do problema

O problema consiste em achar o menor caminho possível para a saída do veículo Z do estacionamento em dois tipos de algoritmos e verificar qual é mais eficiente. Os movimentos possíveis são para cima e para baixo, para esquerda e para a direita, não sendo possível fazer curvas, mudando os veículos de eixo. O algoritmo receberá um arquivo de entrada com a localização dos veículos no estacionamento que será usado durante toda a execução do programa, e primeiramente um arquivo com as manobras que os veículos devem fazer, que será usado apenas na primeira parte do programa. Após esses passos, o programa deverá gerar um novo arquivo de manobras automaticamente, utilizando-se os algoritmos de tentativa e erro e de heurística.

# Primeira Parte

Na primeira parte, foi colocado que o programa deve receber a configuração inicial dos veículos no estacionamento e as manobras a serem realizadas de um arquivo texto e assim retornar ao usuário se foi possível a retirada o veículo de nome **Z** de dentro do estacionamento.

## Configuração Inicial

**Veículos:** Populamos nosso estacionamento com os veículos lidos do arquivo de entrada. Os quais podem ser paralelos ao eixo *X* ou ao *Y*, não sendo permitido veículos na diagonal. Cada linha do arquivo de entrada informa as especificações dos veículos, como exemplifica a tabela abaixo:

Nome. Veiculo	Tipo	Paralelo	Posição
<b>Z</b>	2	X	X2Y4
<b>A</b>	3	Y	X6Y3

**Manobras:** Arquivo ao qual contém o detalhamento dos movimentos que os veículos devem realizar para concluir a saída com sucesso. Cada linha do arquivo indica onde e quantos movimentos que o veículo irá realizar. Em caso de impossibilidade de se retirar o veículo, o usuário será informado.

Nome veiculo	Eixo	Movimentação
<b>A</b>	Y	-2

# Segunda Parte

Nesta parte, o programa precisa gerar planos de manobras de veículos de forma autônoma. Foram utilizadas duas formas para se chegar ao resultado desejado e para se avaliar qual é a melhor:

## Backtracking

Onde todas as possibilidades de movimentação devem ser testadas, ou quase todas, pois sem um limite o programa pode demorar muito para acabar e ser ineficaz para o objetivo do programa.

## Heurística

Uma estratégia criada para resolver o problema, a qual não necessariamente deve obter uma solução.

# Soluções Propostas

## Primeira Parte

Primeiramente é lido as localizações dos veículos no arquivo de entrada, logo depois é alocado dinamicamente memória para a matriz (ixj) que representa o estacionamento. Busca o carro especificado no arquivo de entrada e retorna a posição dentro da matriz. A matriz é toda zerada antes de receber os veículos. O programa então insere o veículo na matriz verificando se há colisões ou posição impossível. Após a inserção de veículos, o programa lê o arquivo de manobras e aloca dinamicamente memória para o vetor de manobras. Realiza as movimentações devidas lidas do arquivo manobras, verificando se ocorreu colisão com outros veículos ou com muros do estacionamento. Após o termino das manobras, o programa retorna ao usuário se o veículo Z saiu do estacionamento com sucesso.

## Segunda Parte

É realizado os mesmos procedimentos da primeira parte, com exceção da leitura e execução das manobras estabelecidas. Para tentativa e erro foi utilizado o algoritmo de backtracking, que consiste em um refinamento da busca por força bruta, em que múltiplas soluções podem ser eliminadas sem serem visitadas. O programa percorre todas as possíveis movimentações para que o veículo Z saia do estacionamento. A cada movimento executado pelo algoritmo é verificado se houve colisão com outros veículos ou com as bordas do estacionamento. A heurística baseou-se numa versão mais otimizada do backtracking, onde os movimentos que não são satisfatórios para a conclusão do problema, são eliminados do vetor caminho e não sendo possível sua visita novamente, evitando um loop infinito e otimizando o algoritmo.

# Analise de Complexidade

A primeira parte do programa consiste em inserir os veículos lidos do arquivo veiculos.txt em uma matriz que representa o estacionamento. Verificando se há colisões entre veículos e/ou posicionamento dos mesmos fora dos limites da matriz. Sequencialmente ao passo anterior, há a leitura do arquivo manobras.txt, e consequentemente a realização dos passos descritos no arquivo manobras.txt, sempre verificando se houve colisão ou extrapolação dos limites da matriz. Caso essas verificações sejam verdadeiras, o programa irá emitir um aviso ao usuário de que a manobra não poderá ser realizada. Com esses passos definidos, sua complexidade será:

$$O(C * O(n) + O(n^2))$$

## Backtracking

Inserir veículos na matriz é repetido em todos os passos do programa. O algoritmo de tentativa e erro consiste em inserir os veículos lidos do arquivo veiculos.txt e após a inserção, gera os passos para saída do veículo Z. Assim sendo, sua complexidade será:

$O(n!)$

## Heurística

É uma variação do tentativa e erro com poda, onde há uma tentativa e se houver algum obstáculo no quadrante próximo, ele sinaliza esse quadrante para que o algoritmo não volte a visitar essa localidade. Sua complexidade será:

$O(n)$

## Funções

```
int Verifica_carro() = O(C)  $\Omega(1)$ :
```

Função que busca um veículo específico no vetor de veículos e retorna a posição que ele se encontra

```
void adiciona_carro() = O(1):
```

Função que adiciona um novo veículo ao vetor de veículos.

```
void insere_carro() = O(1)
```

Função que adiciona o novo veículo à matriz do estacionamento e verifica se não há nenhuma irregularidade na inserção

```
void move_carro_estac() = O(1):
```

Função que faz a movimentação do veículo e o adiciona na matriz do estacionamento

```
int colisao() = O(m)  $fO(2m+2)$ :
```

Verificador se no movimento realizado não se chocou com outro veículo ou parede

```
int verifica() =  $fO(C*O(insere_veiculo)+O(M*O(colisao))+1)$ :
```

Função que recebe dois arquivos de entrada, o primeiro corresponde à posição inicial dos carros no estacionamento e o segundo as movimentações que cada veículo irá realizar. Essa função verifica se o conjunto de manobras, descrito pelo segundo arquivo, são válidas

```
int gera_passo()
```

```
= $fO(C+lim*C(1+2N!)+O(zera_vetor)+N!(4+O(move_varro_estac)))$  ==
```

```
 $fO(C+lim*C+7lim*CN!)$ :
```

Função que tende a gerar todas as possibilidades de movimentação dos veículos inseridos. É criado um vetor interno que possui o tamanho  $limC$  e cada posição pode assumir valores de  $0 \leq mv \leq 4C$

```
void backtrack() =
```

```
 $O(geraPassoIt) fO(O(zera_mat)+N*O(inserir_veiculo)+O(geraPassoIt))$  ==
```

```
 $fO(N+O(C+lim*C+7lim*CN!))$ :
```

Função que recebe um arquivo de entrada, este que corresponde às posições iniciais dos veículos. Essa função tende a buscar um caminho para o carro "Z" até a saída do estacionamento, ou dizer que a posição inicial é impossível de encontrar um caminho. Seu método é testar todas as possibilidades de movimentação do veículo

```
int heuristica() = O(N)
```

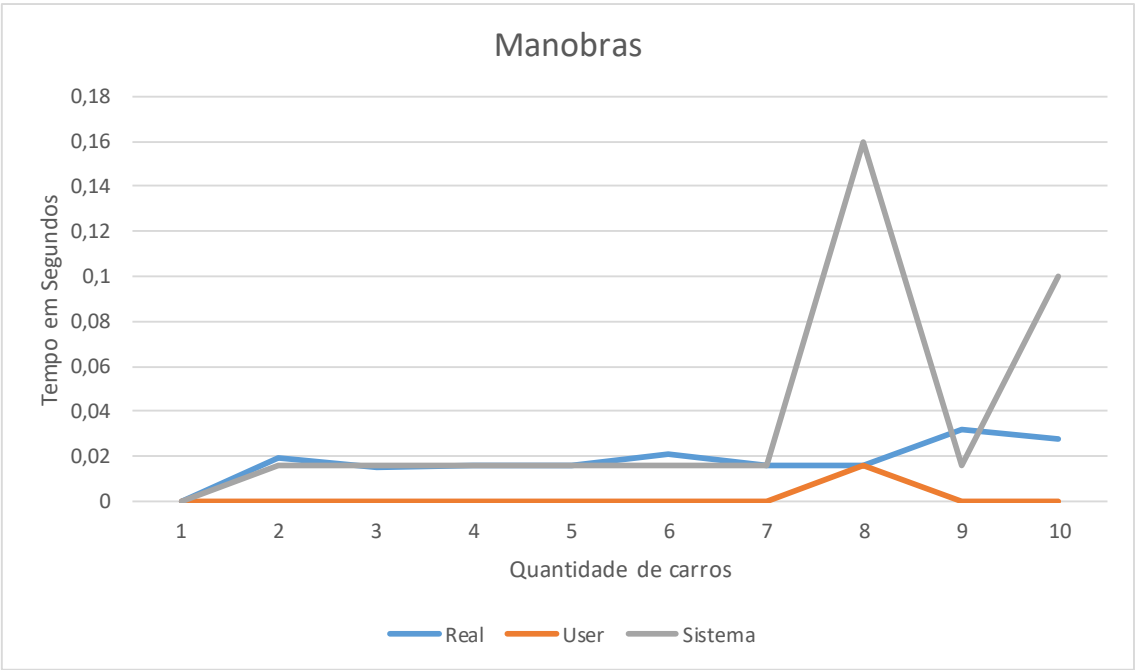
```
 $fO(zera_matriz)+N*O(inserir_veiculo)+100(5+O(move_carro_estac))$  ==
```

```
 $fO(N+500)$ :
```

Função que recebe um arquivo de entrada, este que corresponde às posições iniciais dos veículos. Essa função tende a achar um caminho para o carro "Z" na maioria das vezes até a saída do estacionamento pode não encontrar mesmo se houver, ou dizer que a posição inicial é impossível de encontrar um caminho. Seu método é tentar buscar um caminho entre os carros até encontrar a saída

# Analise dos Resultados

## Verificação de Manobras

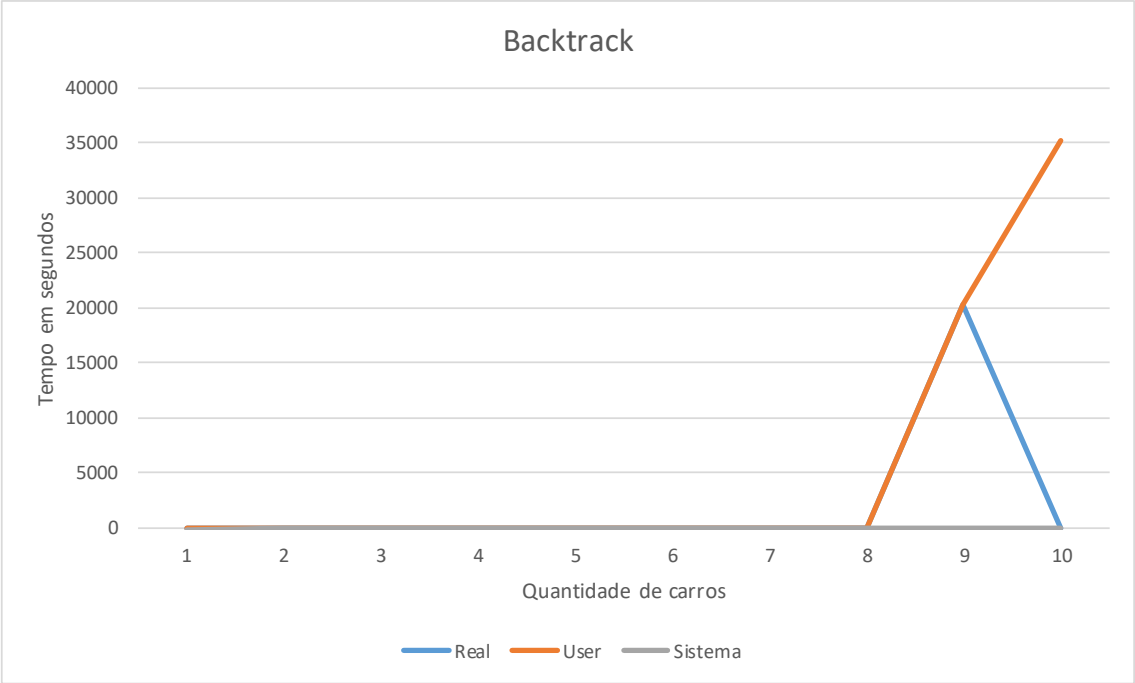


(Grafico 1, tempo em função da quantidade)

Carros	Real	User	Sistema
1	0	0	0
2	0,019	0	0,016
3	0,015	0	0,016
4	0,016	0	0,016
5	0,016	0	0,016
6	0,021	0	0,016
7	0,016	0	0,016
8	0	0,016	0,16
9	0,032	0	0,016
10	0,028	0	0,1

A verificação das manobras baseada nas manobras que o próprio programa gera usando a função backtrack, são muito satisfatórias já que os passos são sempre bem colocados, fazendo que gaste um pequeno tempo para retirar o carro Z.

# Backtrack

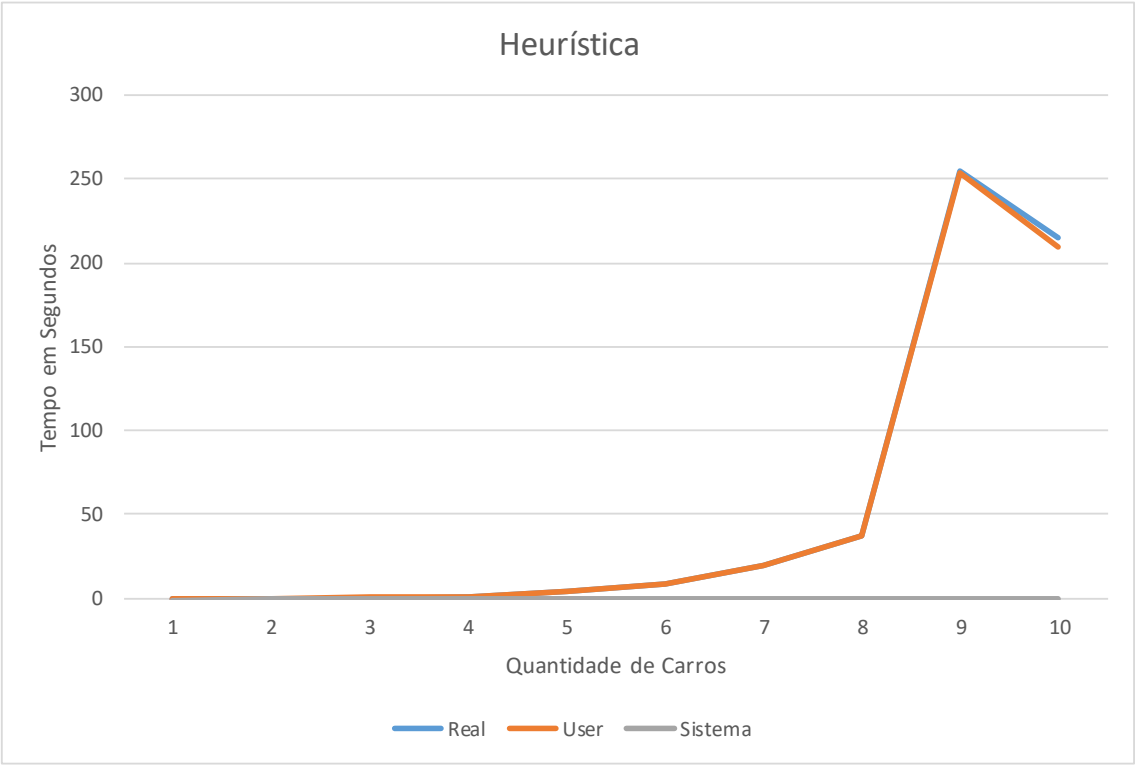


(Gráfico 2)

Carros	Real	User	Sistema
1	0	0	0
2	0,022	0	0,016
3	0,316	0,328	0,016
4	1,232	1,219	0,016
5	3,826	3,781	0,016
6	9,116	9,094	0,016
7	19,34	19,297	0,016
8	37	36,938	0,016
9	20.254,32	20.254	0,032
10	36.215.16	35.200	0,016

A retirada do carro usando o método de backtrack é bastante eficaz já que o mesmo vai podando as possibilidades das quais não são boas para a resolução, assim diminuindo as possibilidades que possam e fazendo um processo mais rápido que uma heurística de tentativa e erro.

# Heurística



(Gráfico 3)

Carros	Real	User	Sistema
1	0	0	0
2	0,022	0	0,016
3	0,316	0,328	0,016
4	1,232	1,219	0,016
5	3,826	3,781	0,016
6	9,116	9,094	0,016
7	19,34	19,297	0,016
8	37	36,938	0,016
9	254,32	254	0,032
10	215,16	210	0,016

A resolução por heurística nos mostrou que na maioria dos casos foi satisfatória, alcançando o resultado desejado e com um baixo custo de tempo. Por ser um backtrack com podas, se mostrou muito mais rápido que sua versão original, pois ele não visita um quadrante mais de 1 vezes, resumindo suas soluções.



Apenas em casos com mais de 8 veículos, que tanto o backtrack original quanto o modificado tiveram seus resultados incongruentes devido ao alto custo de temporário, muitas vezes ultrapassando 2 horas para a conclusão.