RESOLUÇÃO DE PROBLEMAS DE NATUREZA DISCRETA

Sistema de Estacionamento

Curso: Ciência da Computação

SUMÁRIO

Introdução

FUNÇÃO TETO

A função "teto" (do inglês "ceiling") é uma operação matemática que transforma qualquer número decimal no menor inteiro que seja maior ou igual a X.

Introdução

APLICAÇÕES DA FUNÇÃO TETO

- situações onde é necessário garantir que qualquer fração de uma unidade seja considerada como uma unidade inteira.

- ex: cobrança de serviços que são tarifados por hora, onde mesmo que você utilize apenas uma parte de uma hora, a cobrança é feita pela hora inteira.

SISTEMA

- sistema automatizado para o controle de estacionamento de um shopping center;
- gerencia a entrada e saída de veículos;
- calcula automaticamente o valor a ser cobrado pelo estacionamento;
- prevê a origem das placas de veículos, utilizando o padrão Mercosul.

Objetivos

AUTOMATIZAÇÃO DO CONTROLE

- Registrar o tempo do veículo.
- Manter um registro atualizado da posição de cada veículo em uma matriz mxn.

CÁLCULO AUTOMÁTICO DE TARIFAS

- Implementar a cobrança com uma tolerância de 15 minutos.
- Tarifar R$ 5,00 até três horas e R$ 2,50 para cada hora subsequente ou fração de hora, utilizando a função teto para o cálculo.

ORIGEM DAS PLACAS

- Identificar de qual estado brasileiro é a placa do veículo utilizando uma base de dados de combinações de letras específicas para cada estado.
- Informar caso a placa não corresponda a nenhum dos estados atribuídos ao projeto.

METODOLOGIA

- JavaScript
- Implementação com HTML e CSS
- Não foram usados bibliotecas

RESULTADOS

ANÁLISE COMBINATÓRIA E EXPLICAÇÃO DO CÓDIGO

Geração de Placas

- Neste projeto, objetivo é desenvolver um código para identificar o estado brasileiro de um veículo a partir de sua placa no padrão Mercosul, em uso desde 2018. Essas placas possuem uma nova combinação de 4 letras e 3 números (LLLNLNN). A identificação do estado de origem é feita pelas três letras iniciais da placa, e para conseguirmos realizar a geração dessas placas usaremos a análise combinatória.

Geração de Placas

Análise Combinatória: A analise combinatória está associada com o processo de contagem, ou seja, o estudo dessa área da matemática possibilita-nos desenvolver ferramentas que nos auxiliam na realização de contagens de maneira mais eficiente, na analise e contagem de todas as combinações possíveis.

Geração de Placas

- **São Paulo** – Total: 14.196.000 + 70.304.000 + 6.760.000 + 234.000 + 338.000 = 91.832.000

BFA a BZZ

1	21	26	26	10^3	=	14.196.000

CAA a FZZ

4	26	26	26	10^3	=	70.304.000

Geração de Placas

- **São Paulo** – Total: 14.196.000 + 70.304.000 + 6.760.000 + 234.000 + 338.000 = 91.832.000

GAA a GJZ

1	10	26	10^3	=	6.760.000

GKA a GKI

1	1	9	10^3	=	234.000

Geração de Placas

- **São Paulo** – Total: 14.196.000 + 70.304.000 + 6.760.000 + 234.000 + 338.000 = 91.832.000

QSN a QSZ

1	1	13	26	10^3	=	338.000

Geração de Placas

- **Rio de Janeiro** = Total: 546.000 + 8.788.000 + 14.196.000 + 130.000 + 26.000 + 286.000 + 676.000 + 572.000 = 25.220.000
- KMF a LVE

KMF a KMZ

1	1	21	26	10^3	=	546.000

KNA a KZZ

1	13	26	26	10^3	=	8.788.000

Geração de Placas

- **Rio de Janeiro** = Total: 546.000 + 8.788.000 + 14.196.000 + 130.000 + 26.000 + 286.000 + 676.000 + 572.000 = 25.220.000
- KMF a LVE

LAA a LUZ

1	21	26	26	10^3	=	14.196.000

KNA a KZZ

1	1	5	26	10^3	=	130.000

Geração de Placas

- **Rio de Janeiro** = Total: 546.000 + 8.788.000 + 14.196.000 + 130.000 + 26.000 + 286.000 + 676.000 + 572.000 = 25.220.000
- KMF a LVE

LVA a LVE

1	1	5	26	10^3	=	130.000

RIO

1	1	1	26	10^3	=	26.000

Geração de Placas

- **Rio de Janeiro** = Total: 546.000 + 8.788.000 + 14.196.000 + 130.000 + 26.000 + 286.000 + 676.000 + 572.000 = 25.220.000
- RIP a RKV

RIP a RIZ

1	1	11	26	10^3	=	286.000

RJA a RJZ

1	1	26	26	10^3	=	676.000

Geração de Placas

- **Rio de Janeiro** = Total: 546.000 + 8.788.000 + 14.196.000 + 130.000 + 26.000 + 286.000 + 676.000 + 572.000 = 25.220.000
- RIP a RKV

RKA a RKV

1	1	22	26	10^3	=	572.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 52.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000
- MOX a MTZ

MOX a MOZ

1	1	3	26	10^3	=	78.000

RJA a RJZ

1	4	26	26	10^3	=	2.704.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 52.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000
- MOX a MTZ

MOX a MOZ

1	1	3	26	10^3	=	78.000

MPA a MSZ

1	4	26	26	10^3	=	2.704.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 52.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000
- MOX a MTZ

MTA a MTZ

1	1	26	26	10^3	=	676.000

OCV a ODT

1	1	5	26	10^3	=	130.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 520.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000
- MOX a MTZ

ODA a ODT

1	1	20	26	10^3	=	520.000

OCV a ODT

1	1	5	26	10^3	=	130.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 520.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000

OVE a OVF

1	1	2	26	10^3	=	52.000

OVH a OVL

1	1	5	26	10^3	=	130.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 520.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000

OYD a OYK

1	1	8	26	10^3	=	208.000

PPA a PPZ

1	1	26	26	10^3	=	676.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 520.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000

QRB a QRM

1	1	12	26	10^3	=	312.000

RBA a RBJ

1	1	10	26	10^3	=	260.000

Geração de Placas

- **Espírito Santo** – Total: 78.000 + 2.704.000 + 676.000 + 650.000 + 520.000 + 130.000 + 208.000 + 676.000 + 312.000 + 260.000 + 260.000 = 6.006.000

RQM a RQV

1	1	10	26	10^3	=	260.000

RBA a RBJ

1	1	10	26	10^3	=	260.000

Geração de Placas

Total de placas possiveis para o sudeste:

91.832.000 + 25.220.000 + 6.006.000 = 177.112.000

```
//Variáveis globais
const matriz = [];

// BFA a GKI São Paulo
// KMF a LVE Rio de Janeiro
// MOX a MTZ Espírito Santo
```

- variável array vazio
- armazena os veículos do estacionamento

```
// Mapeamento dos estados das placas com intervalos
const estadoPlacas = {
  "São Paulo": [["BFA", "BZZ"], ["GKI", "GKI"], ["QSN", "QSZ"]],
  "Rio de Janeiro": [["KMF", "LVE"], ["RIO", "RIP"], ["RKV", "RKV"]],
  "Espirito Santo": [["MOX", "MTZ"], ["OCV", "ODT"], ["OVH", "OVL"], ["OYD", "OYK"], ["PPA", "PPZ"], ["QRB", "QRM"], ["RBA", "RBJ"], ["RQM"
};
```

- mapeia as sequências de letras das placas de veículos do Sudeste

```
// Função para adicionar um veículo à matriz
function adicionarVeiculo() {
  const placaInput = document.getElementById('placa-input');
  const placa = placaInput.value.toUpperCase();
```

- const placaInput - recebe o elemento do input HTML, onde o usuário digita a placa. Esse elemento deve ter um id igual a 'placa-input'

- const placa - pega o valor do input e converte para letras maiúsculas.

```
// Verificar se a placa é válida
if (!validarPlaca(placa)) {
  alert('Placa inválida. Insira uma placa no padrão correto.');
  return;
}
```

- função chamada validarPlaca e passando a placa como argumento
- se a função validarPlaca retornar false, é mostrado um alerta para o usuário informando que a placa é inválida.
- return - sai da função adicionarVeiculo, evitando que outra lógica seja executada.

```
// Verificar se a placa já está na matriz
if (placaNaMatriz(placa)) {
  alert('Essa placa já está na matriz.');
  return;
}

// Obter o estado da placa
const estado = obterEstadoPlaca(placa);
```

- função placaNaMatriz; argumento placa. verifica se a placa já está presente na matriz.
- se placaNaMatriz retornar true, é mostrado um alerta para o usuário informando que a placa já foi adicionada anteriormente.
- return - sai da função.
- const estado - é chamado a função obterEstadoPlaca com a placa como argumento. determina o estado de registro da placa.

- criação do objeto veiculo com suas propriedades (placa do veículo, estado, tempo estacionaado e valor monetário).
- matriz.push(veiculo) - adicionamos o objeto na matriz.
- atualizarTabela - atualiza a interface do usuário, mostrando o novo estado da matriz (representada por uma tabela no HTML).
- placaInput.value - limpa o campo onde o usuário digitou a placa, preparando para a entrada de uma nova placa.

```
    // Adicionar veículo à matriz
    const veiculo = {
        placa: placa,
        estado: estado,
        tempo: 0,
        valor: 0
    };
    matriz.push(veiculo);

    // Atualizar a tabela
    atualizarTabela();

    // Limpar o campo de entrada
    placaInput.value = '';
}
```

- loop for...in para iterar sobre as propriedades de um objeto chamado estadoPlacas. Este objeto faz o mapeamento de estados para intervalos de placas.
- acessamos os intervalos de placas associados ao estado atual e armazenamos esses intervalos na constante intervalos.
- placaEntreIntervalo(placa, intervalos[i][0], intervalos[i][1]) - verifica se a placa está dentro do intervalo definido por intervalos[i][0] (início do intervalo) e intervalos[i][1] (fim do intervalo). Se a placa estiver dentro do intervalo, a função retorna true.

```
// Função para obter o estado da placa
function obterEstadoPlaca(placa) {
  for (const estado in estadoPlacas) {
    const intervalos = estadoPlacas[estado];
    for (let i = 0; i < intervalos.length; i++) {
      if (placaEntreIntervalo(placa, intervalos[i][0], intervalos[i][1])) {
        return estado;
      }
    }
  }
  return 'Desconhecido';
}
```

- Se a condição for verdadeira, a função obterEstadoPlaca retorna o estado atual, indicando que a placa pertence a esse estado.
- Se a placa não corresponder a nenhum intervalo de placas em qualquer estado, a função retorna 'Desconhecido', indicando que o estado da placa não pôde ser determinado.

```
// Função para verificar se a placa está dentro do intervalo
function placaEntreIntervalo(placa, inicio, fim) {
  return placa >= inicio && placa <= fim;
}
```

- placaEntreIntervalo recebe três argumentos: placa (placa do veículo que queremos verificar), inicio (início do intervalo de placas) e fim (fim do intervalo de placas).
- placa >= inicio - verifica se a placa é maior ou igual ao inicio do intervalo.
- placa <= fim - verifica se a placa é menor ou igual ao fim do intervalo.
- && - retorna true somente se ambas as condições forem verdadeiras, ou seja, se a placa estiver dentro do intervalo.

```
// Função para remover um veículo da matriz
function removerVeiculo() {
  const placaInput = document.getElementById('placa-saida-input');
  const placa = placaInput.value.toUpperCase();

  // Verificar se a placa é válida
  if (!validarPlaca(placa)) {
    alert('Placa inválida. Insira uma placa no padrão correto.');
    return;
  }
```

- document.getElementById('placa-saida-input') - seleciona o elemento HTML (input) onde o usuário digita a placa do veículo que deseja remover. O elemento deve ter um id igual a 'placa-saida-input'.
- placaInput.value - obtém o valor digitado pelo usuário no input.
- toUpperCase() - converte o valor para letras maiúsculas.

- if{...} - esta função verifica se a placa está no formato correto.
- alert - se validarPlaca retornar false, mostra um alerta para o usuário informando que a placa é inválida.
- return - função removerVeiculo é encerrada.

- const index - armazena o índice do veículo na matriz. além disso, chama uma função encontrarVeiculo, esta função deve procurar na matriz a placa especificada e retornar o índice correspondente.
- if (index === -1): Verifica se encontrarVeiculo retornou -1, ou seja, o veículo com a placa especificada não foi encontrado na matriz.
- alert - se o índice for -1, exibe um alerta ao usuário que o veículo não foi encontrado na matriz.
- return - encerra a execução da função removerVeiculo.

```
// Verificar se o veículo está na matriz
const index = encontrarVeiculo(placa);
if (index === -1) {
  alert('Veículo não encontrado na matriz.');
  return;
}

// Remover veículo da matriz
matriz.splice(index, 1);

// Atualizar a tabela
atualizarTabela();
```

- matriz.splice - remove um elemento do array matriz a partir do índice index.
- atualizarTabela() - atualiza a interface do usuário para para mostrar a lista sem o veículo removido.

```
    // Exibir mensagem de remoção
    const registroSaida = document.getElementById('registro-saida');
    registroSaida.textContent = 'Carro ' + placa + ' foi removido do estacionamento.';

    // Limpar o campo de saída
    placaInput.value = '';
}
```

- document.getElementById - seleciona o elemento HTML com o id 'registro-saida'. é um elemento onde queremos exibir uma mensagem ao usuário.
- registroSaida.textContent - a propriedade textContent do elemento registroSaida é definida para o texto que queremos exibir. 'Carro ' + placa + ' foi removido do estacionamento.' está é a mensagem de remoção.
- placaInput.value - limpa o valor do input de texto onde o usuário digitou a placa, preparando-o para a próxima entrada.

- if (index === -1) - verifica se encontrarVeiculo retornou -1, ou seja, o veículo com a placa especificada não foi encontrado na matriz.
- index === -1 - retorna -1 se nenhum elemento satisfaz a condição.

```
// Função para adicionar tempo de estacionamento a um veículo na matriz
function adicionarTempoEstacionamento() {
  const placaInput = document.getElementById('placa-saida-input');
  const placa = placaInput.value.toUpperCase();

  // Verificar se a placa é válida
  if (!validarPlaca(placa)) {
    alert('Placa inválida. Insira uma placa no padrão correto.');
    return;
  }

  // Verificar se o veículo está na matriz
  const index = encontrarVeiculo(placa);
  if (index === -1) {
    alert('Veículo não encontrado na matriz.');
    return;
  }
```

  - alert - se o índice for -1, exibe um alerta informando ao usuário que o veículo não foi encontrado na matriz.
  - return - encerra a execução da função adicionarTempoEstacionamento.

- document.getElementById('tempo-input') - seleciona o elemento HTML com o id 'tempo-input'. este é um input onde o usuário digita o tempo adicional de estacionamento.

```
const tempoInput = document.getElementById('tempo-input');
const tempo = parseInt(tempoInput.value);

// Verificar se o tempo é válido
if (isNaN(tempo) || tempo < 0) {
  alert('Tempo inválido. Insira um valor numérico positivo.');
  return;
}
```

- const tempoInput - declara uma constante tempoInput que armazena a referência ao elemento de input.
- tempoInput.value - obtém o valor atual digitado pelo usuário no campo de input.
- parseInt(tempoInput.value) - converte o valor do input de string para um número inteiro.
- isNaN(tempo) - verifica se tempo é NaN (Not a Number). isso acontece se a conversão com parseInt falhar e não resultar em um número válido.
- tempo < 0 - verifica se tempo é menor que zero. um tempo negativo seria inválido.

- matriz[index].tempo - Acessa a propriedade tempo do objeto veículo localizado no índice index dentro da matriz.
- =tempo - Atualiza essa propriedade com o novo valor de tempo obtido do input.

```
    // Atualizar o tempo do veículo na matriz
    matriz[index].tempo = tempo;

    // Calcular o valor a pagar
    matriz[index].valor = calcularValorEstacionamento(tempo);

    // Atualizar a tabela
    atualizarTabela();

    // Limpar os campos
    placaInput.value = '';
    tempoInput.value = '';
}
```

- matriz[index].valor - acessa a propriedade valor do objeto veículo localizado no índice index dentro da matriz.
- calcularValorEstacionamento(tempo) - chama a função calcularValorEstacionamento, para calcular o custo do estacionamento com base no tempo.

```
// Função para atualizar a tabela com a matriz de veículos
for (let i = 0; i < matriz.length; i++) {
  const veiculo = matriz[i];
  const linha = document.createElement('tr');
  linha.innerHTML = '<td>' + (i + 1) + '</td><td>' + veiculo.placa + '</td><td>' + obterEstadoPlaca(veiculo.placa) + '</td><td>' + veiculo.
  tabela.appendChild(linha);
}
}
```

- const veiculo = matriz[i] - declara uma constante veiculo que armazena o elemento atual da matriz na iteração. cada elemento é um objeto representando um veículo.
- linha.innerHTML - criação da tabela e sua composição.

```
// Função para validar o formato da placa
function validarPlaca(placa) {
  const regexPlaca = /^[A-Z]{3}\d[A-Z]{1}\d{2}$/;
  return regexPlaca.test(placa);
}
```

- const regexPlaca - armazena a expressão regular.
- /^[A-Z]{3}\d[A-Z]{1}\d{2}$/ - define a expressão regular que será usada para validar o formato da placa.
- return regexPlaca.test(placa) - utiliza o método test da expressão regular para verificar se a string placa corresponde ao padrão definido por regexPlaca.
- regexPlaca.test(placa) - retorna true se placa corresponder ao padrão da expressão regular, e false caso contrário.

```
// Função para verificar se a placa já está na matriz
function placaNaMatriz(placa) {
  return matriz.some(veiculo => veiculo.placa === placa);
}
```

- matriz.some(...) - some é um método embutido de arrays em JavaScript que testa se pelo menos um dos elementos no array passa no teste implementado pela função fornecida. ele retorna true se a função de teste retornar true para algum dos elementos do array; caso contrário, retorna false.
- veiculo => veiculo.placa === placa - Esta é uma função de seta (arrow function) que atua como a função de teste para o método some. para cada veiculo na matriz, a função verifica se a propriedade placa do objeto veiculo é igual à placa fornecida como argumento para a função placaNaMatriz.
- veiculo.placa === placa - verifica se a placa do veículo atual (veiculo.placa) é igual à placa que estamos procurando (placa). se for igual, a função de seta retorna true; caso contrário, retorna false.

```
// Função para encontrar o índice do veículo na matriz
function encontrarVeiculo(placa) {
  for (let i = 0; i < matriz.length; i++) {
    if (matriz[i].placa === placa) {
      return i;
    }
  }
  return -1;
}
```

- verificamos se a placa do veículo no índice i da matriz é igual à placa fornecida como argumento. se encontrarmos um veículo cuja placa corresponda à placa fornecida, a função retorna o índice i imediatamente e a execução da função é interrompida.
- se o loop terminar sem encontrar nenhum veículo com a placa correspondente, a função retorna -1. este valor é usado como um indicador de que a placa não foi encontrada na matriz.

- se tempo < tolerancia(15 min), retorna 0.
- se tempo <= 180, retorna o valorHora a ser pago.
- se tempo de permanência > 180 minutos (3 horas):

```
// Função para calcular o valor do estacionamento
function calcularValorEstacionamento(tempo) {
  const tolerancia = 15;
  const valorHora = 5.0;
  const valorFracaoHora = 2.5;

  if (tempo <= tolerancia) {
    return 0;
  } else if (tempo <= 180) {
    return valorHora;
  } else {
    const horas = Math.ceil(tempo / 60);
    return valorHora + (horas - 3) * valorFracaoHora;
  }
}
```

- calcula o número de horas arredondando para cima usando Math.ceil(tempo / 60). isso garante que qualquer fração de hora é contada como uma hora inteira.
- calcula o valor total adicionando ao valor fixo das primeiras 3 horas (valorHora) o custo das horas adicionais. As horas adicionais são calculadas subtraindo 3 do número total de horas e multiplicando pelo valor da fração de hora (valorFracaoHora).

```
// Função para remover um veículo da matriz por índice
function removerVeiculoPorIndice(index) {
    const veiculo = matriz[index];
    matriz.splice(index, 1);
    atualizarTabela();
    const registroSaida = document.getElementById('registro-saida');
    registroSaida.textContent = 'Carro ' + veiculo.placa + ' foi removido do estacionamento.';
}
```

- método splice - remover um elemento da matriz. o método splice altera o conteúdo de um array removendo elementos existentes e/ou adicionando novos elementos.
- document.getElementById('registro-saida') - obtém o elemento HTML com o ID registro-saida. local onde exibiremos a mensagem de saída do veículo.
- registroSaida.textContent = 'Carro ' + veiculo.placa + ' foi removido do estacionamento.' - define o conteúdo de texto do elemento registroSaida para uma mensagem que inclui a placa do veículo removido.

CONCLUSÃO

- sistema automatizado para gerenciar o estacionamento de um shopping center.
- conhecimentos sobre o cálculo automatizado de tarifas baseadas no tempo de estacionamento;
- o uso de estruturas de dados complexas, como matrizes para alocação de vagas;
- integração de uma base de dados para identificação de placas de veículos.
- implementação de tecnologias pode melhorar a qualidade do serviço oferecido aos clientes, aumentando a satisfação e a eficiência operacional do shopping center.

OBRIGADO(A)!

INTEGRANTES:

- ALANIS BITENCOURT
- GUADALUPE PRADO
- LETÍCIA CUNHA
- VICTOR RIOS