

# TP 2 de Algoritmos e estruturas de dados

---

## Introdução

- [O algoritmo de permutação](#)
- [Introdução](#)
- [TP 2 de Algoritmos e estruturas de dados](#)
- [Módulos](#)
  - [UTILS](#)
    - [Funções](#)
  - [Permuta](#)
    - [Funções](#)
- [Comando para compilar o projeto](#)
- [Compilar a documentação](#)

# Módulos auxiliares

## UTILS

### Funções

Função	Tipo	Argumentos	Descrição
<code>currentTime()</code>	<code>struct tm*</code>	-	Retorna a hora atual do sistema
<code>menu()</code>	<code>void</code>	-	Mostra o menu principal
<code>debug(int)</code>	<code>void</code>	<code>int debugindex</code>	Mostra uma mensagem de debug padrão na cor vermelha na ordem do índice passado

### Permuta#### Funções

Função	Tipo	Argumentos	Descrição
<code>nextPerm(int* , int , int)</code>	<code>int</code>	<code>int* perm, int n, int k</code>	Retorna o próximo elemento da permutação

# A matriz distância

## Definição

A matriz distância é uma matriz quadrada de números inteiros, onde a linha `i` e a coluna `j` representam o custo de ir de `i` para `j`.

## Exemplo:

```
Numero de cidades: 5
Capacidade do veiculo: 20
Demanda de cada cidade: 0 5 10 5 20
Distancia entre cada cidade:

0 2 3 4 1
2 0 1 2 -1
3 1 0 10 -1
4 2 10 0 -1
1 -1 -1 -1 0
```

A matriz é alocada dinamicamente a partir da entrada pelo arquivo que contém o numero de cidades e as distancias entre elas.

```
nmroCidades = atoi(buffer);
matrizDistancias = (int **)malloc(sizeof(int *) * nmroCidades);
if (matrizDistancias == NULL)
{
    perror("malloc");
    return -1;
}
```

A matriz é criada usando numeros `-1` para indicar que não existe uma distancia entre as cidades. E usando `0` para indicar a diagonal principal.

```
for (int i = 0; i < nmroCidades; i++)
{
    matrizDistancias[i] = (int *)malloc(sizeof(int) * nmroCidades);
}
for (int i = 0; i < nmroCidades; i++)
{
    for (int j = 0; j < nmroCidades; j++)
    {
        matrizDistancias[i][j] = -1;
        if(i == j)
            matrizDistancias[i][j] = 0;
    }
}
```

## O algoritmo de permutação

---

```
/* Função que seleciona as cidades que serão visitadas. */
int *selecPerm(int *cidades, int *visitadas, int n)
{ // V = cidades nao visitadas
  // contador para quantas cidades nao foram visitadas
  int count = 0;
  // conta o numero de cidades que nao foram visitadas
  for (int i = 0; i < n; i++)
  {
    if (visitadas[i] == 0)
      count++;
  }
  // cria e preenche o vetor que sera permutado, apenas com as cidades que nao foram
  visitadas
  int *perm = (int *)malloc(sizeof(int) * count); // cria o vetor de tamanho V
  int k = 0;                                     // variavel para continuar a
  contagem do vetor com todas as cidades de onde parou
  for (int i = 0; i < count; i++)                 // preenche o vetor de tamanho V
  com as cidades que nao foram visitadas
  {
    for (int j = k; j < n; j++)
    {
      if (visitadas[j] == 0)
      {
        perm[i] = cidades[j];
        k = j + 1;
        break;
      }
      k = j + 1;
    }
  }
  return perm;
}
```

---

## Algoritmo que conta as cidades não visitadas

---

```
/* Função que calcula o numero de cidades que não foram visitadas. */
int calcNaoVisitadas(int *visitadas, int numeroDeCidades)
{
    // v = cidades nao visitadas
    // contador para quantas cidades nao foram visitadas
    int count = 0;
    // conta o numero de cidades que nao foram visitadas
    for (int i = 0; i < numeroDeCidades; i++)
    {
        if (visitadas[i] == 0)
            count++;
    }
    return count;
}
```

---

## Verificação das repetições do vetor

```
/* Função que verifica se o número é repetido. */
char eh_sem_repeticao(int *num, int r)
{
    int i, j;

    for (i = 0; i < r; i++)
    {
        for (j = 0; j < r && i != j; j++)
        {
            if (num[i] == num[j])
            {
                return 0;
            }
        }
    }

    return 1;
}
```

## O arquivo de testes

*O arquivo de teste é um arquivo de texto que contém um numero inteiro positivo N, que define a quantidade de cidades do teste.*

*Seguido de um numero Q, que define a capacidade de carga do caminhão.*

*Seguido de N inteiros que definem a demanda de cada cidade .*

*e por fim*

```
4
5
2 3 4 5
0 1 1
0 2 3
0 3 1
1 2 3
1 3 2
2 3 4
```

# O funcionamento do programa

---

## ***Inicia o programa***

```
Opcoes:
0 - Sair
1 - Ler arquivo de testes
```

## ***ENTRADA:***

```
1
```

## ***SAIDA:***

```
Opcoes:
0 - Sair
1 - Ler arquivo de testes
Digite o nome do arquivo de testes:
>>>
```

*Digite o nome do arquivo a ser lido pelo programa com os valores para teste*

```
Opcoes:
0 - Sair
1 - Ler arquivo de testes
Digite o nome do arquivo de testes:
>>> teste.txt
```

## ***SAIDA:***

```
Opcoes:
0 - Sair
1 - Ler arquivo de testes
Digite o nome do arquivo de testes:
>> 001.txt
Arquivo lido com sucesso          (pressione enter para continuar)
```

---

# Comando para compilar o projeto

Usando Makefile:

```
make
```

---

## Compilar a documentação

```
make doc
```

- Gera arquivo PDF com o mesmo nome do arquivo markdown (não mudar nome do arquivo .md, crie uma cópia com outro nome e compile a cópia, ou mude o nome do arquivo resultado)

```
md-to-pdf readme.md
```

- *Para gerar o arquivo PDF, é necessário ter instalado o programa `md-to-pdf`, caso não tenha a ferramenta de build de documentação instalada*

```
pip install md-to-pdf
--- ou ---
npm install md-to-pdf
```