UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática
Prof. Dr. Ivan Carlos Alcântara de Oliveira Teoría dos Grafos

Relatório do Projeto

Parte 1

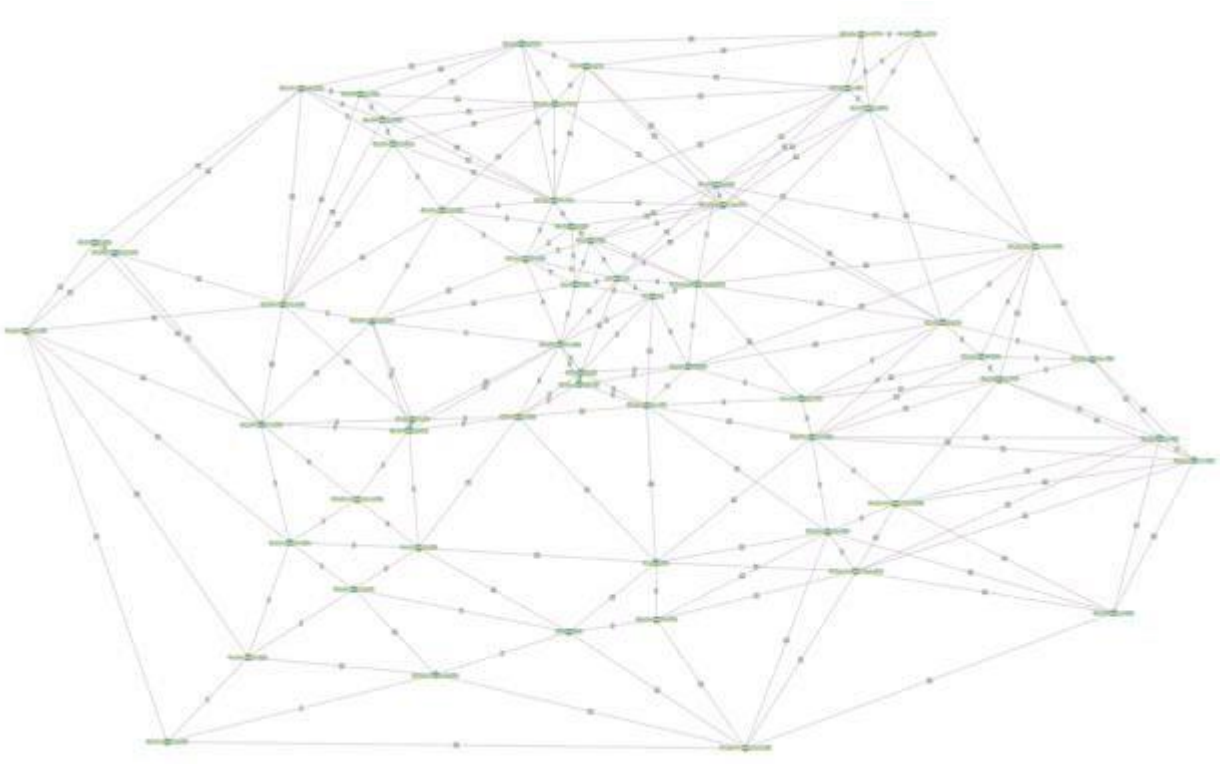
Nome do Integrante	RA
Alix D Avellar Pretto Sanches	10395159
Amanda Gois Smanioto	10395773
Luigi Uematsu	10394168

Relatório

Nos últimos anos, a demanda por veículos elétricos tem crescido bastante devido à sua contribuição para a redução das emissões de carbono e para os impactos ambientais associados ao transporte. No entanto, um dos desafios enfrentados pelos motoristas de veículos elétricos é a localização de postos de recarga acessíveis. O objetivo do projeto é mapear os postos de recarga disponíveis nas proximidades do Mackenzie tendo como ponto de partida o campus Higienópolis e identificar possíveis infraestrutura de recarga para veículos elétricos.

A falta de acessibilidade aos postos de recarga tem sido um dos principais problemas de veículos elétricos. Muitos proprietários de veículos elétricos enfrentam dificuldades para encontrar postos de recarga próximos, o que pode limitar a sua mobilidade. Este projeto tende a enfrentar esse problema fornecendo informações precisas e atualizadas sobre a localização dos postos de recarga.

Grafo: http://graphonline.ru/pt/?graph=VWQnnQAdfyRgtrMm



Legenda (Vértices):

- Mackenzie: 1
- Eletroposto 1: 2
- Ezvolt Charging: 3
- EletroPosto Volvo 2: 4
- EletroPosto Volvo 3: 5
- Ezvolt Charging 2: 6
- EletroPosto Volvo 4: 7
- BMW Charging: 8
- EletroPosto Paulista: 9
- EletroPosto Trianon: 10
- EletroPosto Volvo 5: 11
- BMW Charging 2: 12
- Estação Porshe: 13
- Ezvolt Charging 3: 14
- EletroPosto Mini Autostar: 15
- EletroPosto Spaces: 16
- EletroPosto Volvo 6: 17
- EletroPosto Centro: 18
- Neocharge: 19
- EletroPosto Emporium: 20
- Estação Villa-Lobos: 21
- Estação de Carregamento: 22
- EletroPosto Centro 2: 23
- EletroPosto Pão-de-Açucar: 24
- Estação de Carregamento 2: 25
- Porsche Estação: 26
- EletroPosto Volvo 7: 27
- Porsche Estação 2: 28
- Posto Eletrix: 29
- EletroPosto Ipiranga: 30
- EletroPosto Camargo: 31
- Ezvolt Charging 4: 32
- Ezvolt Charging 5: 33
- EletroPosto Bourbon: 34
- EletroPosto Porsche 3: 35
- EletroPosto St Marche: 36
- Shell Charging: 37
- EletroPosto Autovagas: 38
- BMW Estação 2: 39
- Ezvolt Charging 6: 40
- EletroPosto Novotel: 41
- Ezvolt Charging 7: 42
- EletroPosto Centro 3: 43
- Volve Charging: 44
- Ezvolt Charging 8: 45
- Perenne: 46
- EletroPosto Shopping Light: 47
- Nissan Charging: 48
- Porsche Charging 4: 49
- EletroPosto Volvo 8: 50
- EletroPosto ASSAÍ: 51
- Ezvolt Charging 9: 52
- EletroPosto Centro 4: 53
- EletroPosto Volvo 9: 54
- Ezvolt Charging 10: 55
- Ezvolt Charging 11: 56
- EletroPosto WRK: 57
- INCHARGE: 58
- EletroPosto Volvo 10: 59
- Estação de Carregamento 3: 60

TGrafoND.h

```c
/*
Alix D'Avelar Pretto Sanches     » 10395951
Amanda Gois Smanioto             » 10395773
Luigi Uematsu                    » 10396148
*/

#ifndef TGRAFOND_H
#define TGRAFOND_H
struct TGrafoND {
    int n; // Número de vértices
    int m; // Número de arestas
    int **adj; // Matriz de adjacência
};

void inicializa_TGrafoND(struct TGrafoND *grafo, int n);
void libera_TGrafoND(struct TGrafoND *grafo);
void insereA_TGrafoND(struct TGrafoND *grafo, int v, int w, int peso);
void insereVertice_TGrafoND(struct TGrafoND *grafo);
void insereAresta_TGrafoND(struct TGrafoND *grafo, int v, int w, int peso);
void removeAresta_TGrafoND(struct TGrafoND *grafo, int v, int w);
void removeVertice_TGrafoND(struct TGrafoND *grafo, int v);
void removeA_TGrafoND(struct TGrafoND *grafo, int v, int w);
void mostrarConteudoArquivo(const char *nomeArquivo);
void lerGrafoArquivo(const char *nomeArquivo, struct TGrafoND *grafo);
void gravarDadosArquivo(const char *nomeArquivo, struct TGrafoND
```

```
25  void gravarDadosArquivo(const char *nomeArquivo, struct TGrafoND
    *grafo);
26  int verificaConexidade_TGrafoND(struct TGrafoND grafo);
27  void apresentarConexidadeEGrafoReduzido(struct TGrafoND *grafo);
28  void show_TGrafoND(struct TGrafoND grafo);
29  int podeColorir(struct TGrafoND grafo, int v, int cores[], int cor,
    int num_cores);
30  void colorirGrafo(struct TGrafoND grafo);
31  int verificarGrafoEuleriano(struct TGrafoND grafo);
32  int verificarPercursoEuleriano(struct TGrafoND grafo);
33  int verifica_ciclo_hamiltoniano(int ciclo[], int n, struct TGrafoND
    grafo);
34  int proximo_vertice(int ciclo[], int n, struct TGrafoND grafo);
35  int verificarGrafoHamiltoniano(struct TGrafoND grafo);
36  #endif
```

TGrafoND.c

```c
/*
Alix D'Avelar Pretto Sanches      » 10395951
Amanda Gois Smanioto              » 10395773
Luigi Uematsu                     » 10396148
*/

#include <stdio.h>
#include <stdlib.h>
#include "TGrafoND.h"

void inicializa_TGrafoND(struct TGrafoND *grafo, int n) {
    grafo->n = n;
    grafo->m = 0;
    grafo->adj = (int **)malloc(n * sizeof(int *));

    for (int i = 0; i < n; i++)
        grafo->adj[i] = (int *)calloc(n, sizeof(int)); // Inicializa a matriz com zeros
}
void libera_TGrafoND(struct TGrafoND *grafo) {
    for (int i = 0; i < grafo->n; i++)
        free(grafo->adj[i]);

    free(grafo->adj);
    grafo->n = 0;
    grafo->m = 0;
}
void insereA_TGrafoND(struct TGrafoND *grafo, int v, int w, int
```

```
peso) {
28      if (grafo->adj[v][w] == 0) {
29          grafo->adj[v][w] = peso;
30          grafo->adj[w][v] = peso;
31          grafo->m++;
32      }
33  }
34  void removeA_TGrafoND(struct TGrafoND *grafo, int v, int w) {
35      if (grafo->adj[v][w] == 0) {
36          printf("Aresta não existe\n");
37          return;
38      }
39
40      grafo->adj[v][w] = 0;
41      grafo->adj[w][v] = 0;
42      grafo->m--;
43  }
44  void insereVertice_TGrafoND(struct TGrafoND *grafo) {
45
46      int novoN = grafo->n + 1;
47      grafo->adj = realloc(grafo->adj, novoN * sizeof(int *));
48      grafo->adj[novoN - 1] = (int *)calloc(novoN, sizeof(int));
49
50      for (int i = 0; i < novoN; i++)
51          grafo->adj[i] = realloc(grafo->adj[i], novoN *
sizeof(int));
52      for (int i = 0; i < novoN; i++) {
53          grafo->adj[i][novoN - 1] = 0;
```

```
54          grafo->adj[novoN - 1][i] = 0;
55      }
56
57      grafo->n = novoN;
58  }
59  void removeVertice_TGrafoND(struct TGrafoND *grafo, int v) {
60      if (v < 0 || v >= grafo->n) {
61          printf("Vértice inválido\n");
62          return;
63      }
64
65
66      for (int i = 0; i < grafo->n; i++) {
67          if (grafo->adj[v][i] != 0)
68              removeA_TGrafoND(grafo, v, i);
69      }
70
71
72      free(grafo->adj[v]);
73      for (int i = v; i < grafo->n - 1; i++)
74          grafo->adj[i] = grafo->adj[i + 1];
75      grafo->adj = realloc(grafo->adj, (grafo->n - 1) * sizeof(int
    *));
76      grafo->n--;
77  }
78  void removeAresta_TGrafoND(struct TGrafoND *grafo, int v, int w) {
79      if (v < 0 || v >= grafo->n || w < 0 || w >= grafo->n || grafo-
    >adj[v][w] == 0) {
```

```
80          printf("Aresta não existe\n");
81          return;
82      }
83
84      removeA_TGrafoND(grafo, v, w);
85  }
86  void insereAresta_TGrafoND(struct TGrafoND *grafo, int v, int w,
    int peso) {
87      if (v < 0 || v >= grafo->n || w < 0 || w >= grafo->n) {
88          printf("Vértices inválidos\n");
89          return;
90      }
91
92      insereA_TGrafoND(grafo, v, w, peso);
93  }
94  void show_TGrafoND(struct TGrafoND grafo) {
95      printf("n: %d\n", grafo.n);
96      printf("m: %d\n", grafo.m);
97
98      for (int i = 0; i < grafo.n; i++) {
99          printf("\n");
100
101         for (int w = 0; w < grafo.n; w++)
102             printf("%d ", grafo.adj[i][w]);
103     }
104 }
105 void mostrarConteudoArquivo(const char *nomeArquivo) {
106     FILE *arquivo = fopen(nomeArquivo, "r");
107     if (arquivo == NULL) {
```

```c
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        exit(EXIT_FAILURE);
    }

    int caractere;
    while ((caractere = fgetc(arquivo)) != EOF) {
        printf("%c", caractere);
    }

    fclose(arquivo);
}
void lerGrafoArquivo(const char *nomeArquivo, struct TGrafoND *grafo) {
    FILE *arquivo = fopen(nomeArquivo, "r");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        exit(EXIT_FAILURE);
    }

    // Lê o número de vértices
    fscanf(arquivo, "%d", &(grafo->n));
    inicializa_TGrafoND(grafo, grafo->n);

    for (int i = 0; i < grafo->n; i++) {
        int numero;
        fscanf(arquivo, "%d", &numero);
    }
```

```
    // Lê o número de arestas
    fscanf(arquivo, "%d", &(grafo->m));

    // Lê as arestas e insere no grafo
    for (int i = 0; i < grafo->m; i++) {
        int v, w, peso;
        fscanf(arquivo, "%d %d %d", &v, &w, &peso);
        insereA_TGrafoND(grafo, v - 1, w -1, peso);
    }


}
void gravarDadosArquivo(const char *nomeArquivo, struct TGrafoND
*grafo) {
    FILE *arquivo = fopen(nomeArquivo, "w");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        return;
    }


    fprintf(arquivo, "%d\n", grafo->n);


    for (int i = 0; i < grafo->n; i++) {
        fprintf(arquivo, "%d\n", i + 1);
    }

```

```
163     fprintf(arquivo, "%d\n", grafo->m);
164
165
166     for (int i = 0; i < grafo->n; i++) {
167         for (int j = i + 1; j < grafo->n; j++) {
168             if (grafo->adj[i][j] != 0) {
169                 fprintf(arquivo, "%d %d %d\n", i + 1, j + 1, grafo-
    >adj[i][j]);
170             }
171         }
172     }
173
174     fclose(arquivo);
175 }
176 int verificaConexidade_TGrafoND(struct TGrafoND grafo) {
177
178     int *visitado = (int *)malloc(grafo.n * sizeof(int));
179     for (int i = 0; i < grafo.n; i++)
180         visitado[i] = 0;
181
182     int fila[grafo.n];
183     int frente = 0, tras = 0;
184
185
186     fila[tras++] = 0;
187     visitado[0] = 1;
188
189     while (frente != tras) {
190         int atual = fila[frente++];
```

```
190             int atual = fila[frente++];
191             for (int i = 0; i < grafo.n; i++) {
192                 if (grafo.adj[atual][i] == 1 && !visitado[i]) {
193                     fila[tras++] = i;
194                     visitado[i] = 1;
195                 }
196             }
197         }
198 
199 
200         for (int i = 0; i < grafo.n; i++) {
201             if (!visitado[i]) {
202                 free(visitado);
203                 return 1;
204             }
205         }
206 
207         free(visitado);
208         return 0;
209     }
210     void apresentarConexidadeEGrafoReduzido(struct TGrafoND *grafo) {
211 
212         int conectividade = verificaConexidade_TGrafoND(*grafo);
213 
214         if (conectividade) {
215             printf("O grafo não é conexo.\n");
216             return;
217         } else {
```

```
218         printf("O grafo é conexo.\n");
219     }
220
221
222     struct TGrafoND grafoReduzido;
223     int numVerticesReduzidos = 0;
224
225
226     for (int i = 0; i < grafo->n; i++) {
227         if (!grafoReduzido.adj) {
228             inicializa_TGrafoND(&grafoReduzido, grafo->n);
229         }
230
231         if (grafoReduzido.adj) {
232             for (int j = 0; j < grafo->n; j++) {
233                 if (grafo->adj[i][j] != 0) {
234                     insereA_TGrafoND(&grafoReduzido, i, j, grafo-
>adj[i][j]);
235                 }
236             }
237             numVerticesReduzidos++;
238         }
239     }
240
241     printf("Número de vértices do grafo reduzido: %d\n",
numVerticesReduzidos);
242
243
244     show_TGrafoND(grafoReduzido);
```

```
247     libera_TGrafoND(&grafoReduzido);
248 }
249 int podeColorir(struct TGrafoND grafo, int v, int cores[], int
    cor, int num_cores) {
250     for (int i = 0; i < grafo.n; i++) {
251         if (grafo.adj[v][i] && cores[i] == cor)
252             return 0;
253     }
254     return 1;
255 }
256
257 // Função para colorir o grafo
258 void colorirGrafo(struct TGrafoND grafo) {
259     int num_cores = 0;
260     int cores[grafo.n];
261     for (int i = 0; i < grafo.n; i++)
262         cores[i] = -1;
263     for (int v = 0; v < grafo.n; v++) {
264         int cor_disponivel = 0;
265         while (1) {
266             int cor_usada = 0;
267             for (int i = 0; i < grafo.n; i++) {
268                 if (grafo.adj[v][i] && cores[i] == cor_disponivel)
    {
269                     cor_usada = 1;
270                     break;
271                 }
272             }
273             if (!cor_usada) {
```

```
274                     cores[v] = cor_disponivel;
275                     break;
276                 }
277                 cor_disponivel++;
278             }
279             if (cor_disponivel > num_cores)
280                 num_cores = cor_disponivel;
281         }
282         printf("Quantidade de partições: %d\n", num_cores + 1);
283         printf("Cores atribuídas aos vértices:\n");
284         for (int i = 0; i < grafo.n; i++)
285             printf("Vértice %d: Cor %d\n", i + 1, cores[i]);
286     }
287     int verificarGrafoEuleriano(struct TGrafoND grafo) {
288         for (int v = 0; v < grafo.n; v++) {
289             int grau = 0;
290             for (int i = 0; i < grafo.n; i++) {
291                 if (grafo.adj[v][i] != 0) {
292                     grau++;
293                 }
294             }
295             if (grau % 2 != 0) {
296                 return 0;
297             }
298         }
299         return 1;
300     }
301     int verificarPercursoEuleriano(struct TGrafoND grafo) {
302         int componente_conectado = verificaConexidade_TGrafoND(grafo);
```

```c
    if (!componente_conectado) {
        printf("O grafo não é conectado, portanto não possui um percurso euleriano.\n");
        return 0;
    }
    int num_vertices_impares = 0;
    for (int v = 0; v < grafo.n; v++) {
        int grau = 0;
        for (int i = 0; i < grafo.n; i++) {
            if (grafo.adj[v][i] != 0) {
                grau++;
            }
        }
        if (grau % 2 != 0) {
            num_vertices_impares++;
        }
    }
    if (num_vertices_impares == 0 || num_vertices_impares == 2) {
        printf("O grafo possui um percurso euleriano.\n");
        return 1;
    } else {
        printf("O grafo não possui um percurso euleriano.\n");
        return 0;
    }
}
int proximo_vertice(int ciclo[], int n, struct TGrafoND grafo) {
    for (int i = 0; i < grafo.n; i++) {
        if (!ciclo[i])
            return i;
```

```
331     }
332     return -1;
333 }
334 
335 int verifica_ciclo_hamiltoniano(int ciclo[], int n, struct
    TGrafoND grafo) {
336     if (n == grafo.n) {
337         if (grafo.adj[ciclo[n - 1]][ciclo[0]] != 0) {
338             return 1;
339         } else {
340             return 0;
341         }
342     }
343 
344     int ultimo_vertice = ciclo[n - 1];
345     for (int i = 0; i < grafo.n; i++) {
346         if (!ciclo[i] && grafo.adj[ultimo_vertice][i] != 0) {
347             ciclo[n] = i;
348             if (verifica_ciclo_hamiltoniano(ciclo, n + 1, grafo))
349                 return 1;
350             ciclo[n] = -1;
351         }
352     }
353     return 0;
354 }
355 
356 int verificarGrafoHamiltoniano(struct TGrafoND grafo) {
357     if (grafo.n <= 2) {
```

```
358         return 0;
359     }
360
361     int ciclo[grafo.n];
362     for (int i = 0; i < grafo.n; i++) {
363         ciclo[i] = -1;
364     }
365
366     ciclo[0] = 0;
367     if (verifica_ciclo_hamiltoniano(ciclo, 1, grafo)) {
368         return 1;
369     }
370
371     return 0;
372 }
```

main.c

```c
/*
Alix D'Avelar Pretto Sanches      » 10395951
Amanda Gois Smanioto              » 10395773
Luigi Uematsu                     » 10396148
*/

#include <stdio.h>
#include <stdlib.h>
#include "TGrafoND.h"

int main() {
    struct TGrafoND grafo;
    inicializa_TGrafoND(&grafo, 0);
    int opcao;
    do {
        printf("\nMenu:\n");
        printf("1. Ler dados do arquivo grafo.txt\n");
        printf("2. Gravar dados no arquivo grafo.txt\n");
        printf("3. Inserir vértice\n");
        printf("4. Inserir aresta\n");
        printf("5. Remove vértice\n");
        printf("6. Remove aresta\n");
        printf("7. Mostrar conteúdo do arquivo\n");
        printf("8. Mostrar grafo\n");
        printf("9. Apresentar a conexidade do grafo e o
reduzido;\n");
        printf("10. Colorir o grafo\n");
        printf("11. Verificar se o grafo é euleriano\n");
```

```
28          printf("12. Verificar percurso euleriano\n");
29          printf("13. Verificar se o grafo é Hamiltoniano\n");
30          printf("0. Sair\n");
31          printf("Escolha uma opção: ");
32          scanf("%d", &opcao);
33
34          switch (opcao) {
35              case 1: {
36                  lerGrafoArquivo("grafo.txt", &grafo);
37                  break;
38              }
39              case 2: {
40                  gravarDadosArquivo("grafo.txt", &grafo);
41                  break;
42              }
43              case 3: {
44                  insereVertice_TGrafoND(&grafo);
45                  printf("Vértice inserido com sucesso.\n");
46                  break;
47              }
48              case 4: {
49                  int v, w, peso;
50                  printf("Digite os números dos vértices (v w) e o
    peso da aresta: ");
51                  scanf("%d %d %d", &v, &w, &peso);
52                  insereA_TGrafoND(&grafo, v - 1, w - 1, peso);
53                  printf("Aresta inserida com sucesso.\n");
54                  break;
55              }
```

```
56                case 5: {
57                    int vertice;
58                    printf("Digite o número do vértice a ser removido:
    ");
59                    scanf("%d", &vertice);
60                    removeVertice_TGrafoND(&grafo, vertice - 1);
61                    printf("Vértice removido com sucesso.\n");
62                    break;
63                }
64                case 6: {
65                    int v, w;
66                    printf("Digite os números dos vértices (v w) da
    aresta a ser removida: ");
67                    scanf("%d %d", &v, &w);
68                    removeAresta_TGrafoND(&grafo, v - 1, w - 1);
69                    printf("Aresta removida com sucesso.\n");
70                    break;
71                }
72                case 7: {
73                    printf("Conteúdo do arquivo grafo.txt: \n");
74                    mostrarConteudoArquivo("grafo.txt");
75                    break;
76                }
77                case 8: {
78                    printf("Grafo: \n");
79                    show_TGrafoND(grafo);
80                    break;
81                }
82                case 9: {
```

```
                apresentarConexidadeEGrafoReduzido(&grafo);
                break;
            }
            case 10: {
                colorirGrafo(grafo);
                break;
            }
            case 11: {
                if (verificarGrafoEuleriano(grafo))
                  printf("O grafo é euleriano.\n");
                else
                  printf("O grafo não é euleriano.\n");
                break;
            }
            case 12: {
                verificarPercursoEuleriano(grafo);
                break;
            }
            case 13: {
                if (verificarGrafoHamiltoniano(grafo))
                    printf("O grafo é hamiltoniano.\n");
                else
                    printf("O grafo não é hamiltoniano.\n");
            }
            case 0: {
                printf("FIM\n");
                break;
            }
            default: {
                printf("Opção inválida. Tente novamente.\n");
                break;
            }
        }
    } while (opcao != 0);

    libera_TGrafoND(&grafo);

    return 0;
}
```

Saida

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 1

Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 2
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 3
Vértice inserido com sucesso.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 4
Digite os números dos vértices (v w) e o peso da aresta: 4 12 22
Aresta inserida com sucesso.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 5
Digite o número do vértice a ser removido: 4 12 22
Vértice removido com sucesso.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 6
Digite os números dos vértices (v w) da aresta a ser removida: 4 12 22
Aresta removida com sucesso.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 7
Conteúdo do arquivo grafo.txt:
60
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 8
Grafo:
n: 60
m: 419

0 1 0 0 4 11 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 13 0 0
0 0 0 0 0 0 2 3 0 0 5 0 0 0 0 0 0 0 0 0 0
1 0 1 0 2 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 9 0 0 0
0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 8 0 0 0 0 0 0 0 10 0 0 0
0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0
4 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 6 12 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0
11 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 0 0 0 0 1 14 0
 0 0 0 13 0 0 9 19 0 0 0 0 0 0 0 0 0 0 0 16
0 0 0 0 0 0 0 5 5 0 0 10 0 0 0 0 13 0 15 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 12 0 0 7 0 0 0 0 0 0 0 3 0 0 0
11 0 0 0 0 0 5 0 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 9 0 0 0 1 0 0 0 0 0 0 7 0 0 0
0 0 0 0 0 0 5 1 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 7 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 20 15 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 7 14 0 0 0 0 0 0
0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 10 2 2 0 0 0 5 5 0 0 0 0 17 0 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 5 0 1 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 10 0 0 0 0 0 0
0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 5 1 0 7 7 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 5 4 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 7 7 5 0 6 0 0 0 14 0 0 0 0 0 0 0 0 0 10 0 0 0 13 16 17 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 13 0 0 0 0 0 0 0 4 6 0 4 14 0 17 0 0 0 0 5 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 10 0 0 0 0 0 2 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 0 0
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 9
O grafo não é conexo.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
0. Sair
Escolha uma opção: 0
FIM
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
10. Colorir o grafo
11. Verificar se o grafo é euleriano
12. Verificar percurso euleriano
13. Verificar se o grafo é Hamiltoniano
0. Sair
Escolha uma opção: 10
Quantidade de partições: 1
Cores atribuídas aos vértices:
```

```
Escolha uma opção: 10
Quantidade de partições: 6
Cores atribuídas aos vértices:
Vértice 1: Cor 0
Vértice 2: Cor 1
Vértice 3: Cor 0
Vértice 4: Cor 1
Vértice 5: Cor 2
Vértice 6: Cor 2
Vértice 7: Cor 0
Vértice 8: Cor 1
Vértice 9: Cor 2
Vértice 10: Cor 0
Vértice 11: Cor 0
Vértice 12: Cor 3
Vértice 13: Cor 0
Vértice 14: Cor 2
Vértice 15: Cor 0
Vértice 16: Cor 1
Vértice 17: Cor 2
Vértice 18: Cor 0
Vértice 19: Cor 1
Vértice 20: Cor 1
Vértice 21: Cor 3
Vértice 22: Cor 2
Vértice 23: Cor 3
Vértice 24: Cor 0
Vértice 25: Cor 1
Vértice 26: Cor 4
Vértice 27: Cor 2
Vértice 28: Cor 4
Vértice 29: Cor 3
Vértice 30: Cor 3
Vértice 31: Cor 4
Vértice 32: Cor 0
Vértice 33: Cor 0
Vértice 34: Cor 1
Vértice 35: Cor 2
```

```
Vértice 35: Cor 2
Vértice 36: Cor 0
Vértice 37: Cor 2
Vértice 38: Cor 3
Vértice 39: Cor 4
Vértice 40: Cor 1
Vértice 41: Cor 2
Vértice 42: Cor 3
Vértice 43: Cor 2
Vértice 44: Cor 0
Vértice 45: Cor 0
Vértice 46: Cor 2
Vértice 47: Cor 3
Vértice 48: Cor 1
Vértice 49: Cor 2
Vértice 50: Cor 5
Vértice 51: Cor 3
Vértice 52: Cor 3
Vértice 53: Cor 4
Vértice 54: Cor 4
Vértice 55: Cor 4
Vértice 56: Cor 4
Vértice 57: Cor 4
Vértice 58: Cor 3
Vértice 59: Cor 4
Vértice 60: Cor 5
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
10. Colorir o grafo
11. Verificar se o grafo é euleriano
12. Verificar percurso euleriano
13. Verificar se o grafo é Hamiltoniano
0. Sair
Escolha uma opção: 11
O grafo não é euleriano.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
10. Colorir o grafo
11. Verificar se o grafo é euleriano
12. Verificar percurso euleriano
13. Verificar se o grafo é Hamiltoniano
0. Sair
Escolha uma opção: 12
O grafo não possui um percurso euleriano.
```

```
Menu:
1. Ler dados do arquivo grafo.txt
2. Gravar dados no arquivo grafo.txt
3. Inserir vértice
4. Inserir aresta
5. Remove vértice
6. Remove aresta
7. Mostrar conteúdo do arquivo
8. Mostrar grafo
9. Apresentar a conexidade do grafo e o reduzido;
10. Colorir o grafo
11. Verificar se o grafo é euleriano
12. Verificar percurso euleriano
13. Verificar se o grafo é Hamiltoniano
0. Sair
Escolha uma opção: 13
O grafo não é hamiltoniano.
FIM
```

Link do Video: https://youtu.be/6Ovl9FjV2UU
Link do GitHub: https://github.com/LuidaoKun/Projeto-Grafos-Carros-El-tricos