Instituto Federal de Educação, Ciência e Tecnologia do maranhão
Algoritmos e Estruturas de Dados ll
Franciele Alves da Silva (20231SI0012)

Relatório

Para além do código fornecido, foram feitas mais duas funções para a realização da atividade, funções essas que serão explicadas a seguir:

Função para Ler e Construir o Grafo

```c
void le_e_constroi_grafo(const char* arquivo, Grafo* gr) {
    FILE* fp = fopen(arquivo, "r");
    if (fp == NULL) {
        perror("Erro ao abrir o arquivo");
        return;
    }

    int nro_vert, nro_arestas;
    fscanf(fp, "%d %d", &nro_vert, &nro_arestas);

    int* coords_x = (int*)malloc(nro_vert * sizeof(int));
    int* coords_y = (int*)malloc(nro_vert * sizeof(int));

    // Leitura das coordenadas dos vértices
    for (int i = 0; i < nro_vert; i++) {
        fscanf(fp, "%*s %d %d", &coords_x[i], &coords_y[i]);
        printf("Coordenadas do vértice %d: (%d, %d)\n", i + 1, coords_x[i], coords_y[i]);
    }

    // Construção do grafo
    for (int i = 0; i < nro_vert; i++) {
        for (int j = i + 1; j < nro_vert; j++) {
            int xd = coords_x[i] - coords_x[j];
            int yd = coords_y[i] - coords_y[j];
            float dist = round(sqrt(xd * xd + yd * yd));
            if (dist > 0) {
                insereAresta(gr, i, j, 0, dist);
                insereAresta(gr, j, i, 0, dist); // Como não é um digrafo
            }
            printf("Distância entre %d e %d: %f\n", i + 1, j + 1, dist);
        }
    }

    free(coords_x);
    free(coords_y);
    fclose(fp);
}
```

- A função fopen abre o arquivo em modo leitura ("r"). Se não conseguir abrir o arquivo, perror imprime um erro e a função retorna.
- `fscanf` lê o número de vértices e arestas do arquivo.
- São alocados dois arrays (`coords_x` e `coords_y`) para armazenar as coordenadas dos vértices.
- Um laço `for` lê as coordenadas de cada vértice e armazena nos arrays. As coordenadas são exibidas para depuração.
- Outro laço `for` calcula a distância entre todos os pares de vértices e insere as arestas no grafo usando a função `insereAresta`. As distâncias também são exibidas para depuração.
- A memória alocada para as coordenadas é liberada e o arquivo é fechado.

Função para Imprimir os Primeiros 10 Vértices e seus Pesos

```c
void imprime(const Grafo* gr) {
    printf("void imprime(const Grafo* gr) tices e seus pesos:\n");
    for (int i = 0; i < 10 && i < gr->nro_vert; i++) {
        printf("Vértice %d:\n", i + 1);
        for (int j = 0; j < gr->grau[i]; j++) {
            int adj = gr->arestas[i][j];
            float peso = gr->eh_ponderado ? gr->pesos[i][j] : 0.0;
            printf("  -> Vértice %d, Peso: %.2f\n", adj + 1, peso);
        }
    }
}
```

- Um laço `for` percorre os primeiros 10 vértices (ou até o número total de vértices se for menor que 10).

- Outro laço `for` percorre as arestas do vértice atual e imprime cada aresta com seu peso. Se o grafo não for ponderado, o peso será 0.0.

Main:

```c
int main() {setlocale(LC_ALL, "");
    // Criação do grafo
    Grafo* gr = cria_Grafo(450, 450, 1); // Número de vértices e arestas
    if (gr Grafo* main::gr
        perror("Erro ao criar o grafo");
        return EXIT_FAILURE;
    }

    // Lê o arquivo e constrói o grafo
    le_e_constroi_grafo("Benchmark-grafo.txt", gr);

    // Imprime os primeiros 10 vértices e seus pesos
    imprime(gr);

    libera_Grafo(gr);

    return EXIT_SUCCESS;
}
```

- A função `cria_Grafo` cria um grafo com 450 vértices e 450 arestas(número máximo por vértice). O terceiro parâmetro `1` indica que o grafo é ponderado.
- A função `le_e_constroi_grafo` é chamada para ler o arquivo e construir o grafo.
- A função `imprime` imprime informações sobre os primeiros 10 vértices e suas arestas com pesos.
- Finalmente, a função `libera_Grafo` é chamada para liberar a memória alocada para o grafo.

Primeiros 10 vértices e suas arestas com pesos: