Relatório 4 - Estruturas de Dados Avançadas - INF1010- 2021.2

Alunos:

Marcos Vinicius Araujo Almeida - 1910869
Breno Azevedo Marot - 1910423

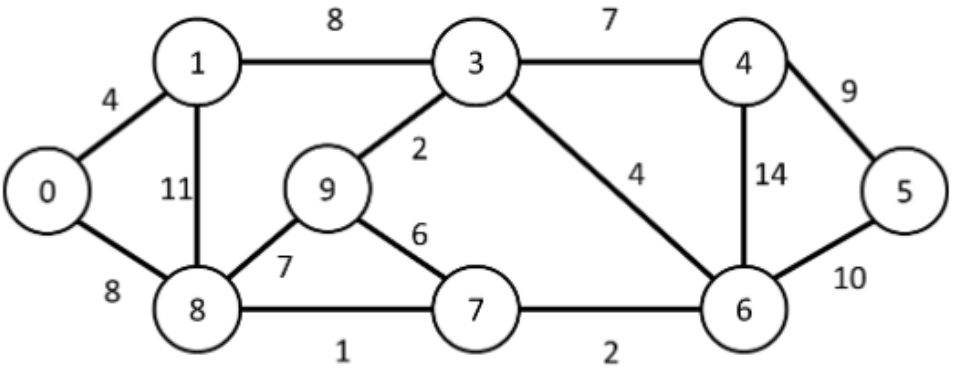
Turma:

3WB

Professor:

Luiz Fernando Seibel

Criando o grafo



Para podermos criar o grafo acima, foram criadas estruturas e algumas funções auxiliares.

```c
typedef struct _viz
{
    int noj;
    int peso;
    struct _viz *prox;
} Viz;

typedef struct _grafo
{
    int nv;     /* numero de nos ou vertices */
    int na;     /* numero de arestas */
    Viz **viz; /* viz[i] aponta para a lista
    de arestas incidindo em i */
} Grafo;
```

- A **struct _viz** representa a vizinhança de cada vértice e é composta pelo peso de cada aresta
- A **struct _viz** é chamada na **struct _grafo** como um vetor de listas encadeadas

```
Grafo *criaGrafoDado(int nv, int na)
{
    Grafo *g = grafoCria(nv, na);
    // grafo->viz[no1] = criaViz(grafo->viz[no1], no2, peso);

    g->viz[0] = criaViz(g->viz[0], 1, 4);
    g->viz[0] = criaViz(g->viz[0], 8, 8);

    g->viz[1] = criaViz(g->viz[1], 8, 11);
    g->viz[1] = criaViz(g->viz[1], 3, 8);
    g->viz[1] = criaViz(g->viz[1], 0, 4);

    g->viz[8] = criaViz(g->viz[8], 0, 8);
    g->viz[8] = criaViz(g->viz[8], 1, 11);
    g->viz[8] = criaViz(g->viz[8], 9, 7);
    g->viz[8] = criaViz(g->viz[8], 7, 1);

    g->viz[9] = criaViz(g->viz[9], 8, 7);
    g->viz[9] = criaViz(g->viz[9], 7, 6);
    g->viz[9] = criaViz(g->viz[9], 3, 2);

    g->viz[3] = criaViz(g->viz[3], 9, 2);
    g->viz[3] = criaViz(g->viz[3], 1, 8);
    g->viz[3] = criaViz(g->viz[3], 6, 4);
    g->viz[3] = criaViz(g->viz[3], 4, 7);

    g->viz[7] = criaViz(g->viz[7], 9, 6);
    g->viz[7] = criaViz(g->viz[7], 8, 1);
    g->viz[7] = criaViz(g->viz[7], 6, 2);

    g->viz[4] = criaViz(g->viz[4], 3, 7);
    g->viz[4] = criaViz(g->viz[4], 6, 14);
    g->viz[4] = criaViz(g->viz[4], 5, 9);

    g->viz[6] = criaViz(g->viz[6], 7, 2);
    g->viz[6] = criaViz(g->viz[6], 3, 4);
    g->viz[6] = criaViz(g->viz[6], 4, 14);
    g->viz[6] = criaViz(g->viz[6], 5, 10);

    g->viz[5] = criaViz(g->viz[5], 4, 9);
    g->viz[5] = criaViz(g->viz[5], 6, 10);

    return g;
}
```

- Os vértices foram inseridos de forma manual seguindo o modelo comentado no início da função. Cada posição do vetor **viz** representa a posição do vértice observado e é preenchido pela sua vizinhança (seguido com peso das arestas).
- A função criaViz está definida dentro do código fonte que foi enviado junto deste relatório

Criação da árvore mínima

Para criarmos a árvore mínima seguindo o método de *Kruskal* , devemos ordenar todas as arestas do grafo em questão em ordem crescente, para isso utilizamos uma lista encadeada, cuja estrutura foi definida como:

```c
typedef struct elemento
{
    int peso;
    int ini;
    int fim;
    struct elemento *prox;
} Elemento;
```

- *peso* indica o peso de cada aresta a ser armazenada
- *ini* indica o nó de origem da aresta
- *fim* indica o destino da aresta

OBS.: É importante ressaltar que destino e origem não indicam que o grafo está direcionado!

Após isso devemos ler as informações do Grafo e ordená-las em ordem crescente

```c
Elemento *preencheListaOrdenada(Grafo *g)
{
    int i;
    Viz *p;
    Elemento *lst = lst_cria();
    Elemento *aux;

    for (i = 0; i < g->nv; i++)
        for (p = g->viz[i]; p != NULL; p = p->prox)
            lst = lista_insere_ordenado(lst, p->peso, i, p->noj);

    for (aux = lst; aux != NULL; aux = aux->prox)
        lst = lista_retira(lst, aux->peso, aux->fim, aux->ini);

    return lst;
}
```

O algoritmo consiste em ler o grafo, inserir os pesos das arestas, nós de início e de destino em uma mesma lista encadeada. Após a inserção devemos remover as duplas ocorrências dos destinos e origens (já que desse modo, inserimos as mesmas arestas 2 vezes).

Por fim, para criar a árvore mínima utilizamos a função abaixo:

```c
void geraArvoreMinima(Grafo *g, Elemento *arestas)
{
    UniaoBusca *ub = cria(g->nv);
    Elemento *p;
    int vertex1, vertex2;
    int cont = 0;

    for (p = arestas; p != NULL; p = p->prox)
    {
        vertex1 = p->ini;
        vertex2 = p->fim;

        if (vertex1 == 9)
            vertex1 = 2;
        if (vertex2 == 9)
            vertex2 = 2;

        // printf("[%d, %d] - %d\n", vertex1, vertex2, ub->v[ub_uniao(ub, vertex2, vertex1)]);
        int b1 = ub_busca(ub, vertex1);

        if (ub->v[ub_uniao(ub, vertex1, vertex2)] != b1)
        {
            // printf("[%d, %d] - %d\n\n", vertex1, vertex2, p->peso);
            vetorAdjacencias[cont] = p->peso;
            cont++;
        }
    }
}
```

Primeiramente criamos o union-find e percorremos a lista encadeada de arestas. É importante ressaltar que o vértice 2 não estava definido no grafo, logo precisamos “setar” a posição 2 do vetor de adjacências para o valor 9. Após isso, aplicamos um uma busca do vértice 1 e unimos vértice 1 com vértice 2. Por fim, podemos preencher o vetor de adjacências com os pesos das arestas para uma melhor visualização. Exibindo o resultado temos: