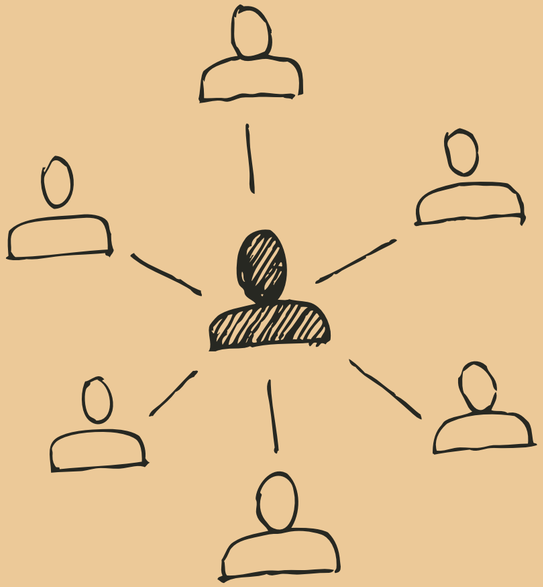
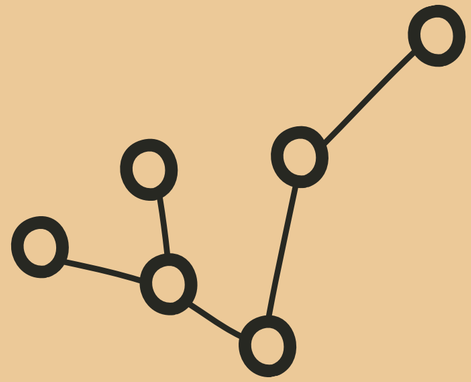
TEORIA DOS GRAFOS

Lukas Freitas de Carvalho - 202376033





Redimensionamento da Matriz

O código redimensiona a matriz toda vez que sua capacidade total é preenchida, realocando toda matriz com o dobro de tamanho.

```cpp
void Grafo_matriz::resize(int novaCapacidade) {
NodeVertex* newVertices = new NodeVertex[novaCapacidade]();
for (int i = 0; i < getOrdem(); i+=1) {
    newVertices[i].setValue(vertices[i].getValue());
    newVertices[i].setGrau(vertices[i].getGrau());
}
delete[] vertices;
vertices = newVertices;

NodeEdge*** novaMatriz = nullptr;
if (eh_direcionado()) {

    novaMatriz = new NodeEdge**[novaCapacidade];
    for (int i = 0; i < novaCapacidade; i+=1) {
        novaMatriz[i] = new NodeEdge*[novaCapacidade]();
    }

    for (int i = 0; i < capacidade; i+=1) {
        for (int j = 0; j < capacidade; j+=1) {
            novaMatriz[i][j] = matriz_adjacencia[i][j];
        }
    }

    for (int i = 0; i < capacidade; i+=1) {
        delete[] matriz_adjacencia[i];
    }
} else {
    int novoTamanho = novaCapacidade * (novaCapacidade - 1) / 2;
    int tamanhoAntigo = capacidade * (capacidade - 1) / 2;
    novaMatriz = new NodeEdge**[1];
    novaMatriz[0] = new NodeEdge*[novoTamanho]();

    for (int i = 0; i < tamanhoAntigo; i+=1) {
        novaMatriz[0][i] = matriz_adjacencia[0][i];
    }

    delete[] matriz_adjacencia[0];
}

delete[] matriz_adjacencia;
matriz_adjacencia = novaMatriz;
capacidade = novaCapacidade;
}
```

Algoritmo de Dijkstra

O método maiorMenorDistancia implementa o algoritmo de Dijkstra para calcular a menor distância entre dois vértices em um grafo ponderado. Ele inicializa as distâncias como infinitas e processa os vértices, sempre escolhendo o mais próximo ainda não visitado. Para cada vizinho, a distância é atualizada se for possível encontrar um caminho mais curto. Ao final, retorna a menor distância ou -1 se os vértices não forem conectados.

```cpp
string Grafo::retornaMaiorMenorDistancia()
{
    if (arestaPonderada()) {
        for (int i = 0; i < getOrdem(); i+=1) {
            for (int j = 0; j < getOrdem(); j+=1) {
                NodeEdge* aresta = getAresta(i, j);
                if (aresta != nullptr)
                {
                    if(aresta->getPeso() < 0){
                        return "\nNão é permitido o uso de pesos negativos nas arestas";
                    }
                }
            }
        }
    }
    float maior = -1;
    int indexX = -1;
    int indexY = -1;
    for(int i = 0; i<getOrdem(); i+=1)
    {
        for(int j = 0; j<getOrdem(); j+=1)
        {
            if(i != j)
            {
                float valor = maiorMenorDistancia(i,j);
                if(valor > maior)
                {
                    indexX = i;
                    indexY = j;
                    maior = valor;
                }
            }
        }
    }
    if(maior == -1)
    {
        return "\nNão existe nenhum caminho de nenhum vértice para nenhum vértice";
    }
    else
    {
        return "(" + to_string(indexX+1) + "-" + to_string(indexY+1) + ") " + to_string(maior);
    }
}
```

```cpp
float Grafo::maiorMenorDistancia(int ponto1, int ponto2) {
    NodeVertex* noPontoUm = getVertice(ponto1);
    NodeVertex* noPontoDois = getVertice(ponto2);

    if (noPontoUm == nullptr || noPontoDois == nullptr) {
        cout << "Erro: Vértices não encontrados." << endl;
        return -1;
    }
    if(noPontoUm->getGrau() == 0 && noPontoDois->getGrau() == 0)
    {
        return -1;
    }

    //Variável grande que eu usei para simular o infinito
    const float INF = 1e9;
    float* distancias = new float[getOrdem()]();
    bool* visitados = new bool[getOrdem()]();


    for (int i = 0; i < getOrdem(); i+=1) {
        distancias[i] = INF;
        visitados[i] = false;
    }

    distancias[ponto1] = 0;

    for (int count = 0; count < getOrdem(); count+=1) {
        int u = -1;
        float min_dist = INF;

        for (int i = 0; i < getOrdem(); i+=1) {
            if (!visitados[i] && distancias[i] < min_dist) {
                min_dist = distancias[i];
                u = i;
            }
        }

        if (u == -1) break;
        visitados[u] = true;

        for (int v = 0; v < getOrdem(); v+=1) {
            if (u == v) continue;
            NodeEdge* aresta = getAresta(u, v);
            if (aresta != nullptr) {
                float peso = aresta->getPeso();
                if (distancias[u] + peso < distancias[v]) {
                    distancias[v] = distancias[u] + peso;
                }
            }
        }
    }

    float valor = distancias[ponto2];
    delete[] distancias;
    delete[] visitados;

    if (valor == INF) {
        return -1;
    } else {
        return valor;
    }

}
```

OBRIGADO!