Funções da interface gráfica (implementada em MainFrame.java):

- “Carregar novo grafo”: recebe a String do caminho do arquivo de origem do novo grafo e o enum do tipo de grafo que foi pedido e envia para a função carregarGrafo dos algoritmos, o resultado é salvo num objeto grafo genérico da Main;

- “Visualizar a representação do grafo”: chama a função toString do grafo salvo e exibe o resultado na tela de resultados;

- “Ver AGM de Kruskall”: executa as funções de algoritmo agmUsandoKruskall e custoDaArvoreGeradora (passando o conjunto de arestas da AGM feita), e exibe as informações das arestas e do custo na tela de resultados;

- “Ver as arestas do grafo e seus tipos”: executa as 4 funções relacionadas aos tipos de arestas (arestasDeArvore, arestasDeRetorno, arestasDeAvanco e arestasDeCruzamento) e exibe as informações de cada conjunto na tela de resultados. Exibe uma tela de erro caso pelo menos um dos vértices não exista naquele grafo;

- “Ver o caminho mínimo”: executa uma função de cálculo de caminho mínimo (menorCaminho, caso a opção de usar os pesos esteja desabilitada, e caminhoMaisCurto, caso ela esteja habilitada) e a função custoDoCaminho para calcular o custo daquele caminho, e exibe as informações das arestas e do custo do caminho na tela de resultados;

- “Adicionar aresta no grafo”: executa uma função de grafo adicionarAresta e exibe o conjunto de arestas salvas, apontando qual é a aresta nova, na tela de resultados. Exibe uma tela de erro caso pelo menos um dos vértices não exista naquele grafo;

- “Ver grau e adjacentes do vértice”: executa as funções de grafo adjacentesDe e grauDoVertice para o vértice passado e exibe as informações obtidas das funções na tela de resultados. Exibe uma tela de erro caso o vértice não exista naquele grafo;

- “Setar o peso da aresta”: executa a função de grafo setarPeso e exibe o peso da aresta selecionada antes e depois da alteração na tela de resultados. Pode exibir uma tela de erro caso pelo menos um dos vértices não exista naquele grafo, ou eles não tenham arestas entre si;

- “Fazer busca em largura”: executa a função de algoritmo buscaEmLargura e exibe na tela de resultados as informações dos vértices relacionadas àquela busca (distância da origem e pai do vértice) e mostra as arestas que foram montadas na busca;

- “Fazer busca em largura”: executa a função de algoritmo buscaEmProfundidade e exibe na tela de resultados as informações dos vértices relacionadas àquela busca (tempo de descoberta e tempo de finalização) e mostra as arestas que foram montadas na busca;

Funções de algoritmos em grafos (implementadas em Algoritmos.java):

- carregarGrafo: cria e retorna uma estrutura de Grafo, ela consiste em abrir um arquivo de texto, cujo caminho foi especificado nos parâmetro, escrever suas linhas em um array de strings e por meio de um bloco switch-case decide qual modelo de representação de grafo será usado e retorna a nova instancia;

- buscaEmLargura: realiza uma busca em largura em um grafo a partir do 1º vértice salvo e retorna as arestas resultantes das descobertas de vértices, as outras informações relacionadas a busca (distância da origem e pai do vértice) são salvas dentro do próprio vértice;

- buscaEmProfundidade: realiza uma busca em profundidade em um grafo a partir do 1º vértice salvo e retorna as arestas resultantes das descobertas de vértices, as outras informações relacionadas a busca (tempo de descoberta e tempo de finalização) são salvas dentro do próprio vértice. Possui como auxiliar a função BEP_Visit, que faz a busca recursiva pelos pontos e retorna as arestas nova da busca;

- menorCaminho: executa o algoritmo de Dijkstra para buscar o caminho mais curto entre 2 pontos de um grafo, porém considera que todas as arestas possuem peso igual a 1.0, o desconsidera o seu peso real e só analisa o número de passos que foi dado. Pode retornar uma exceção caso os pontos enviados não pertençam ao grafo em questão. Possui como auxiliares as funções menorPasso_SP e relaxa (versão sem o parâmetro w), responsáveis por procurar qual o menor passo a ser dado a partir de um conjunto de vértices (sem considerar seus pesos) e relaxar uma determinada aresta considerando seu peso como 1.0, respectivamente;

- existeCiclo: realiza uma busca em profundidade e retorna verdadeiro caso ocorra um caso em que um vértice cinza encontre outro vértice cinza, caso isso nunca ocorra a função retorna o valor falso;

- agmUsandoKruskall: cria uma Árvore Geradora Mínima de Kruskall e retorna as arestas resultantes das árvores da floresta gerada pelo algoritmo. Possui como auxiliar a função arestasOrdenadas, que devolve um conjunto com as arestas reais do grafo em ordem crescente de acordo com seus pesos;

- custoDaArvoreGeradora: recebe um conjunto de arestas, gera a AGM de Kruskall para aquele grafo, verifica se as arestas enviadas são iguais as da árvore gerada e, se for correspondente, soma o peso das arestas da árvore;

- ehArvoreGeradora: recebe um conjunto de arestas, gera a AGM de Kruskall para aquele grafo e verifica se as arestas enviadas são iguais as da árvore gerada, se forem retorna verdadeiro, senão retorna falso;

- caminhoMaisCurto: executa o algoritmo de Dijkstra “real” para buscar o caminho mais curto entre 2 pontos de um grafo, considerando todas as arestas reais no grafo e seus pesos. Pode retornar uma exceção caso os pontos enviados não pertençam ao grafo em questão. Possui como auxiliares as funções menorPasso e relaxa (versão com o parâmetro w), responsáveis por procurar qual o menor passo a ser dado a partir de um conjunto de vértices e relaxar uma determinada aresta, respectivamente;

- custoDoCaminho: recebe um conjunto de arestas e 2 vértices (uma origem e um destino), gera o caminho mais curto (utilizando a função “caminhoMainCurto()”) para aqueles pontos naquele grafo, verifica se as arestas enviadas são iguais ao caminho gerado e, se for correspondente, soma o peso das arestas do caminho;

- ehCaminho: recebe um conjunto de arestas e 2 vértices (uma origem e um destino), gera o caminho mais curto (utilizando a função “caminhoMainCurto()”) para aqueles pontos naquele grafo, verifica se as arestas enviadas são iguais ao caminho gerado, se forem retorna verdadeiro, senão retorna falso;

- arestasDeArvore: realiza uma busca em profundidade, porém ao invés de criar arestas novas para a árvore, coleta todas as arestas existentes entre 2 vértices a partir de uma nova descoberta de vértice em um conjunto, e retorna o conjunto com as arestas correspondentes

da árvore que seria montada. Possui como auxiliar a função BEP_Visit_AAr, responsável pela busca recursiva e pelo retorno das arestas analisadas;

- arestasDeRetorno: realiza uma busca em profundidade, porém ao invés de criar arestas novas para a árvore, coleta todas as arestas existentes onde um vértice é adjacente a um outro que já foi observado, e retorna o conjunto com as arestas de retorno da árvore que seria montada Possui como auxiliar a função BEP_Visit_ARe, responsável pela busca recursiva e pelo retorno das arestas analisadas;

- arestasDeAvanco: realiza uma busca em profundidade comum, compara quais arestas reais caem no caso onde seus vértices pertencem a uma mesma árvore e a origem é mais distante do centro que o destino, e retorna o conjunto com as arestas em que esse caso é ocorrido. Usa como auxiliar a função arestasOrdenadas, que já foi descrita anteriormente;

- arestasDeCruzamento: realiza uma busca em profundidade, porém vai acompanhando o crescimento da floresta resultante das buscas, e depois verifica para todas as arestas reais, quais são aquelas em que seus vértices pertencem a árvores diferentes, e envia o conjunto de arestas em que esse caso ocorre. Usa como auxiliar a função arestasOrdenadas, que já foi descrita anteriormente;

Funções de grafos (implementadas em MatrizAdj.java, MatrizInc.java e ListaAdj.java):

*funções variam de acordo com a classe, mas o funcionamento básico segue o descrito abaixo

- toString: Override da função padrão para objetos, monta uma String com a representação gráfica do grafo e os seus números totais de vértices e arestas;

- adicionarAresta: adapta a representação gráfica para a nova aresta adicionada e coloca a aresta em si no conjunto de arestas salvas do grafo. Caso seja a função com o parâmetro peso, gera a aresta com aquele valor específico, caso não seja, usa o valor de peso como 1.0;

- existeAresta: usa a representação do grafo para verificar se um vértice especifico recebe pelo menos uma aresta de uma origem especificada;

- grauDoVertice: usa a representação gráfica para contar quantos vértices são adjacentes a um determinado vértice. Retorna uma exceção caso o vértice passado não exista no grafo;

- numeroDeVertices: retorna o tamanho do conjunto de vértices daquele grafo;

- numeroDeArestas: retorna o tamanho do conjunto de arestas daquele grafo;

- adjacentesDe: usa a representação do grafo para identificar quais vértices recebem arestas do vértice passado, e retorna uma lista com eles;

- setarPeso: procura a 1ª aresta entre os vértices passados, altera seu peso, e altera a representação do grafo para acompanhar. Retorna uma exceção caso os vértices passados não pertençam ao grafo, ou eles não possuam arestas entre si;

- vertices: retorna um conjunto com os vértices salvos no grafo;

- arestasEntre: recebe 2 vértices e verifica quais das arestas salvas tem aquela origem especifica e aquele destino especifico e retorna as arestas analisadas. Retorna uma exceção caso os vértices passados não pertençam ao grafo, ou eles não possuam arestas entre si.