

- **Objectivos:**

- Implementar os algoritmos de Prim e Kruskal para determinar uma árvore de expansão mínima de um grafo não dirigido pesado conexo.
- O programa deve também aceitar grafos não conexos, devendo determinar nestes casos tantas árvores quanto os componentes conexos do grafo.
- O grafo deve ser definido num ficheiro de texto, com uma linha de texto para cada aresta do grafo, com os nomes dos dois vértices ligados pela aresta e a capacidade da aresta.
- O programa deve ter como parâmetros de chamada (passados na linha de comandos) o nome do ficheiro com a especificação do grafo e o nome do algoritmo a usar (Prim ou Kruskal).
- Após a execução do algoritmo o programa deve escrever no *standard output* uma árvore de expansão mínima (ou conjunto) obtida pelo algoritmo escolhido.

- **Descrição:**

Os principais conceitos da aplicação, são representados do seguinte modo:

A classe **AlreadyAddedException** representa as excepções que ocorrem quando o programa tenta adicionar ao conjunto de vértices do grafo inicial um vértice que já existe nesse conjunto. Esta classe pertence à excepção **RunTimeException**.

A classe **AlreadyConnectedException** representa as excepções que ocorrem quando o programa tenta estabelecer uma ligação entre dois vértices que já se encontram ligados. Esta classe pertence à excepção **RunTimeException**.

A classe **EqualVerticesException** representa as excepções que ocorrem quando o programa tenta adicionar ao conjunto de arestas do grafo inicial uma aresta que contém uma ligação entre dois vértices iguais. Esta classe pertence à excepção **RunTimeException**.

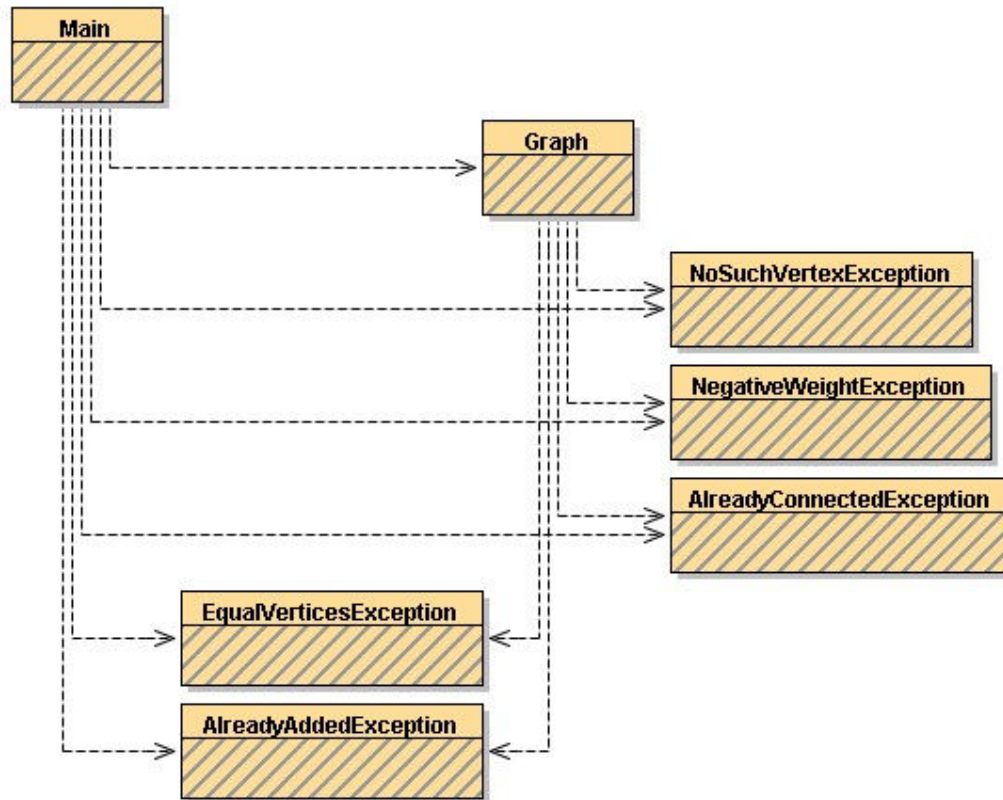
A classe **NegativeWeightException** representa as excepções que ocorrem quando o programa tenta adicionar ao conjunto de arestas do grafo inicial uma aresta que contém peso negativo. Esta classe pertence à excepção **RunTimeException**.

A classe **NoSuchVertexException** representa as excepções que ocorrem quando o programa tenta adicionar ao conjunto de arestas do grafo inicial uma aresta que contém pelo menos um vértice desconhecido ao grafo inicial. Esta classe pertence à excepção **RunTimeException**.

A classe **Graph** representa o grafo em si, com todas as características que este possui e com duas classes internas: **Edge** e **Vertex**. Estas representam, respectivamente, as arestas e os vértices de um grafo. É nesta classe que estão contidos os algoritmos para o cálculo da árvore de expansão mínima (*Prim* e *Kruskal*).

A classe **Main** é classe executável do programa que determina, entre outras coisas, se o ficheiro se encontra no formato correcto, se o grafo escrito no ficheiro contém uma árvore única ou se está dividido em várias árvores e se toda a informação para processar o grafo se encontra disponibilizada e correcta. Esta classe possui três classes internas: **VertexInfo**, **EdgeInfo** e **GraphInfo**, que disponibilizam à classe informação sobre os vértices, as arestas e o próprio grafo, respectivamente, mesmo antes de o grafo ser realmente construído. Assim torna-se mais simples o processamento da informação e a determinação dos erros que possam ocorrer na construção de um grafo.

As classes acima descritas estão organizadas do seguinte modo:



- **Implementação:**

- **AlreadyAddedException:**

*public AlreadyAddedException()* – construtor da exceção sem argumentos.

*public AlreadyAddedException(String s)* – construtor da exceção que recebe como argumento a String s que representa a exceção.

- **AlreadyConnectedException:**

*public AlreadyConnectedException()* – construtor da exceção sem argumentos.

*public AlreadyConnectedException(String s)* – construtor da exceção que recebe como argumento a String s que representa a exceção.

- **EqualVerticesException:**

*public EqualVerticesException()* – construtor da exceção sem argumentos.

*public EqualVerticesException(String s)* – construtor da exceção que recebe como argumento a String s que representa a exceção.

- **NegativeWeightException:**

*public NegativeWeightException()* – construtor da exceção sem argumentos.

*public NegativeWeightException(String s)* – construtor da exceção que recebe como argumento a String s que representa a exceção.

- **NoSuchVertexException:**

*public NoSuchVertexException()* – construtor da exceção sem argumentos.

*public NoSuchVertexException(String s)* – construtor da exceção que recebe como argumento a String s que representa a exceção.

- **Graph:**

*private HashMap vertices* – *HashMap* que contém todos os vértices do grafo.

*private LinkedList edges* – *LinkedList* que contém todas as arestas do grafo.

*private LinkedList edgesKruskal* – *LinkedList* que contém as arestas a serem processadas no algoritmo de *Kruskal*.

*public Graph()* – construtor do grafo.

*public void addVertex(String name)* – método que adiciona um vértice ao grafo com o nome *name*.

*public void addEdge(String v1Name, String v2Name, int weight)* – método que adiciona ao grafo uma aresta com o peso *weight* e que contém os vértices com os nomes *v1Name* e *v2Name*.

*public void prim()* – método que executa o algoritmo de *Prim* no grafo.

*public void kruskal()* – método que executa o algoritmo de *Kruskal* no grafo.

*public String getAlgorithmResult()* – método que devolve a *String* com o resultado da execução de um dos dois algoritmos possíveis.

*public String toString()* – método que devolve a *String* que representa o grafo.

*private void resetAll()* – método que limpa o conteúdo do grafo.

*private Edge selectEdgePrim(LinkedList verticesToProcess)* – método que devolve a aresta a seguir pelo algoritmo de *Prim* de acordo com a *LinkedList* de vértices *verticesToProcess*.

*private Edge selectEdgeKruskal(LinkedList edgesToProcess)* – método que devolve a aresta a seguir pelo algoritmo de *Kruskal* de acordo com a *LinkedList* de arestas *edgesToProcess*.

*private int indexOf(LinkedList trees, Vertex vertex)* – método que devolve o *index* do vértice *vertex* na *LinkedList* de árvores *trees*.

- **Vertex:**

*String name* – nome do vértice.

*LinkedList edges* – arestas ligadas ao vértice.

*public Vertex()* – construtor do vértice.

- **Edge:**

*Vertex v1* – vértice 1 da aresta.

*Vertex v2* – vértice 2 da aresta.

*int weight* – peso da aresta.

*public Edge()* – construtor da aresta.

- **Main:**

*private String filename* – *String* com o nome do ficheiro que contém o grafo a ser processado.

*private String algorithm* – *String* com o algoritmo a executar sobre o grafo.

*private File file* – ficheiro que contém o grafo a ser processado.

*private LinkedList verticesInfo* – *LinkedList* que contém todos os *VertexInfo*.

*private LinkedList edgesInfo* – *LinkedList* que contém todas as *EdgeInfo*.

*private LinkedList graphsInfo* – *LinkedList* que contém todos os *GraphInfo*.

*private LinkedList graphs* – *LinkedList* que contém todos os grafos.

*public Main(String fileName, String algorithm)* – construtor da classe **Main** que recebe o nome do ficheiro a abrir e o algoritmo a usar.

*public void run()* – método que é chamado pelo sistema e que invoca os métodos abaixo descritos.

*private void validate()* – método que valida o nome do ficheiro e o algoritmo a serem usados pelo programa.

*private void loadFile()* – método que abre o ficheiro a usar e verifica se o formato deste é o correcto, “lançando” excepções se estas se verificarem.

*private void buildGraphsInfo()* – método que constrói um **GraphInfo** com toda a informação já encontrada e processada.

*private void buildGraphs()* – método que constrói um **Graph** com a informação obtida no **GraphInfo**.

*private void printAll()* – método que imprime para o *standard output* todos os grafos antes de serem executados sobre eles o algoritmo pretendido.

*private void algorithm()* – método que indica ao utilizador qual o algoritmo que está a ser usado e se este já começou a ser executado.

*private void showResult()* – método que mostra o resultado da execução do algoritmo, ou seja, a árvore de expansão mínima.

*private boolean checkString(String s)* – método que verifica se uma linha do ficheiro contém caracteres alfanuméricos.

*private int indexOf(String vertexName)* – método que devolve o *index* do vértice com o nome *vertexName* na *LinkedList verticesInfo*.

#### - VertexInfo

*String name* – nome do vértice.

*public VertexInfo(String name)* – construtor do **VertexInfo** que recebe o nome do vértice a criar.

#### - EdgeInfo

*String v1Name* – nome do vértice 1 da aresta.

*String v2Name* – nome do vértice 2 da aresta.

*int weight* – peso da aresta.

*public EdgeInfo(String v1Name, String v2Name, int weight)* – construtor da **EdgeInfo** que recebe o nome do primeiro vértice, o nome do segundo vértice e o peso da aresta.

#### - GraphInfo

*LinkedList verticesInfo* – *LinkedList* que contém os vértices do **GraphInfo**.

*LinkedList edgesInfo* – *LinkedList* que contém as arestas do **GraphInfo**.

*public GraphInfo()* – construtor do **GraphInfo**.

- **Problemas e Soluções:**

- **Estruturação da informação de um grafo:**

Neste tópico residuiu, na nossa opinião, a maior dificuldade na realização do trabalho, visto a importância que uma boa representação da informação do grafo teria na realização dos algoritmos e na rapidez com que estes seriam feitos.

Decidimos criar três classes internas na classe **Main**: **VertexInfo**, **EdgeInfo** e **GraphInfo**. Estas classes revelaram-se bastante úteis, já que permitiram-nos guardar e processar a informação lida do ficheiro, criando vértices, arestas e grafos intermédios, que facilitaram a procura de erros e o processamento da informação.

- **Implementação do algoritmo de Prim:**

Ao tentar descobrir a melhor maneira de implementar o algoritmo de Prim surgiram algumas hipóteses bastante fiáveis e aproximadas. Escolher a melhor foi de facto o problema...

Optámos por utilizar uma estrutura do tipo HashMap para guardar os vértices do grafo e uma do tipo LinkedList para guardar as arestas. Usámos também uma LinkedList para guardar os vértices a processar e que ia sendo preenchida conforme os vértices fossem ligados.

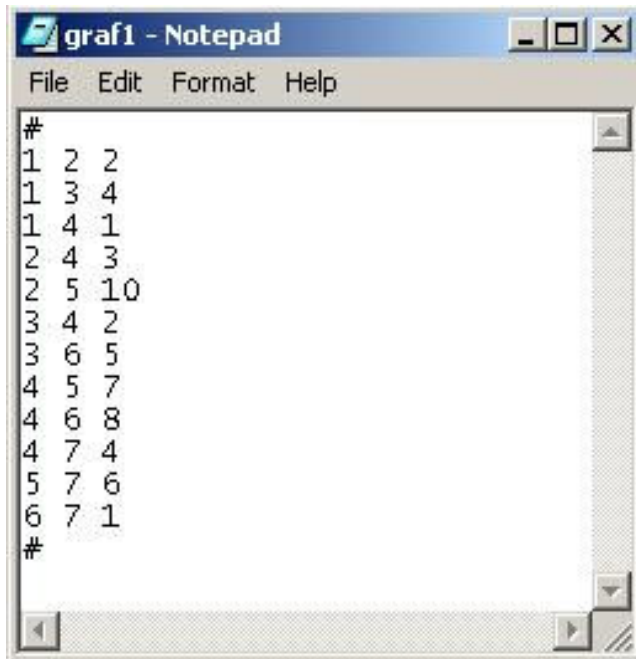
- **Implementação do algoritmo de Kruskal:**

Tal como na implementação do algoritmo de Prim, também na implementação do algoritmo de Kruskal surgiram alguns problemas na adopção da melhor maneira de o fazer.

Relativamente ao algoritmo de Prim, a única diferença é que enquanto este usa uma LinkedList de vértices a serem processados o de Kruskal utiliza uma LinkedList de arestas a serem processadas e vai sendo preenchida conforme se procede a uma ligação.

- **Modo de Utilização:**

- **Formato do ficheiro de entrada:**



No início do ficheiro é necessário colocar um cardinal (#) para indicar ao programa que pode começar a ler arestas do grafo, terminando quando encontrar novamente outro cardinal (#).

Entre os cardinais deve definir as arestas do grafo do seguinte modo:

- O primeiro elemento representa o primeiro vértice.
- O segundo elemento representa o segundo vértice.
- O terceiro elemento representa o peso da aresta.

- **Passagem de argumentos na linha de comandos:**



Para correr o programa basta executar o comando de Java, indicando o *package* (**MinimumSpanningTree**) e a *classe executável* (**Main**) e indicar os argumentos que pretende utilizar:

- O primeiro argumento representa o ficheiro a ser utilizado para a leitura do grafo.
- O segundo argumento representa o algoritmo a ser executado pelo programa (P – *Prim* ou K – *Kruskal*)



- Erros:



```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf2.txt K
    O ficheiro "D:\AEDII\graf2.txt" nao foi encontrado
D:\AEDII>
```

Erro que ocorre quando o programa não consegue abrir o ficheiro especificado pelo utilizador.



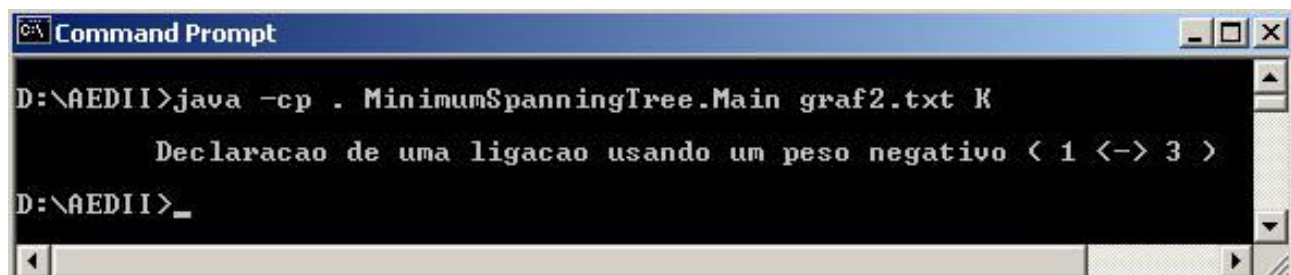
```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf1.txt eK
    Algoritmo invalido
D:\AEDII>_
```

Erro que ocorre quando o programa não reconhece o algoritmo especificado pelo utilizador.



```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf2.txt K
    Declaracao duplicada de uma ligacao < 1 <-> 2 >
D:\AEDII>_
```

Erro que ocorre quando o programa detecta uma duplicação de uma ligação.



```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf2.txt K
    Declaracao de uma ligacao usando um peso negativo < 1 <-> 3 >
D:\AEDII>_
```

Erro que ocorre quando o programa detecta que uma aresta tem peso negativo.



```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf2.txt K
    Declarao de uma ligacao de um vertice para si proprio < 1 <-> 1 >
D:\AEDII>_
```

Erro que ocorre quando o programa detecta uma ligação de um vértice para si próprio.

- Saída de dados:

```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf2.txt P
    Grafo 1
    1 <-> 2 < 2 >
    1 <-> 3 < 4 >
    1 <-> 4 < 1 >
    2 <-> 4 < 3 >
    2 <-> 5 < 10 >
    3 <-> 4 < 2 >
    3 <-> 6 < 5 >
    4 <-> 5 < 7 >
    4 <-> 6 < 8 >
    4 <-> 7 < 4 >
    5 <-> 7 < 6 >
    6 <-> 7 < 1 >

    Grafo 2
    8 <-> 9 < 2 >
    9 <-> 10 < 3 >

    A executar o Algoritmo de Prim no grafo 1...
    A executar o Algoritmo de Prim no grafo 2...

    Grafo 1 - Arvore de expansao minima
    5 <-> 7 < 6 >
    7 <-> 4 < 4 >
    2 <-> 1 < 2 >
    4 <-> 3 < 2 >
    6 <-> 7 < 1 >
    1 <-> 4 < 1 >

    Grafo 2 - Arvore de expansao minima
    9 <-> 10 < 3 >
    8 <-> 9 < 2 >

D:\AEDII>_
```

```
Command Prompt
D:\AEDII>java -cp . MinimumSpanningTree.Main graf2.txt K

Grafo 1
1 <-> 2 < 2 >
1 <-> 3 < 4 >
1 <-> 4 < 1 >
2 <-> 4 < 3 >
2 <-> 5 < 10 >
3 <-> 4 < 2 >
3 <-> 6 < 5 >
4 <-> 5 < 7 >
4 <-> 6 < 8 >
4 <-> 7 < 4 >
5 <-> 7 < 6 >
6 <-> 7 < 1 >

Grafo 2
8 <-> 9 < 2 >
9 <-> 10 < 3 >

A executar o Algoritmo de Kruskal no grafo 1...
A executar o Algoritmo de Kruskal no grafo 2...

Grafo 1 - Arvore de expansao minima
1 <-> 4 < 1 >
6 <-> 7 < 1 >
1 <-> 2 < 2 >
3 <-> 4 < 2 >
4 <-> 7 < 4 >
5 <-> 7 < 6 >

Grafo 2 - Arvore de expansao minima
8 <-> 9 < 2 >
9 <-> 10 < 3 >

D:\AEDII>
```

Na primeira parte da saída de dados o programa mostra os grafos iniciais, indicando depois ao utilizador que o algoritmo está a ser executado sobre todos os grafos existentes e no fim mostra os resultados dessas execuções (árvores de expansão mínima) ao utilizador.

- **Referências:**

- Publicações**

- C, How To Program, Third Edition  
Deitel & Deitel  
Prentice Hall

- Recursos Web**

- [java.sun.com](http://java.sun.com)