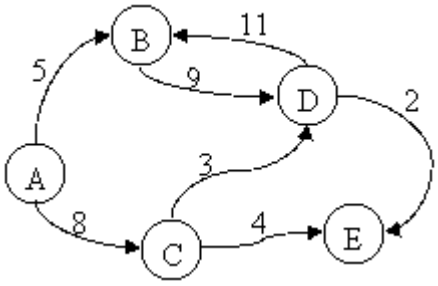
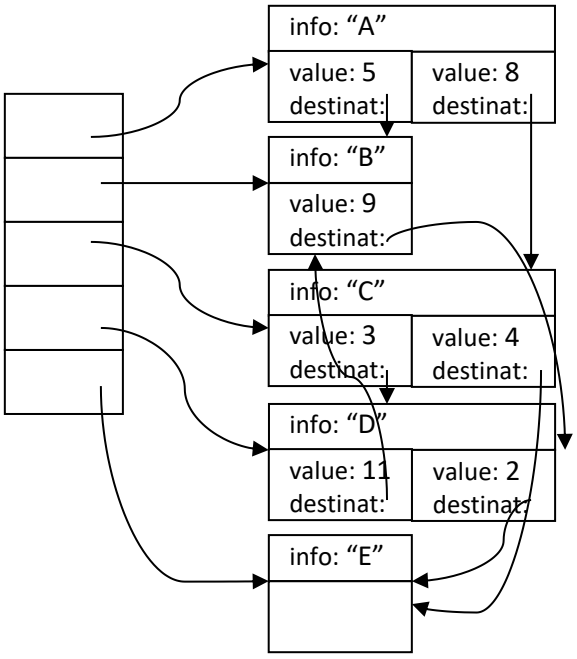
3ªaula prática - Tratamento de Exceções. *Templates* de Classes

- Faça download do ficheiro ***aeda2021_p03.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***graph.h*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)
- Note que os testes unitários deste projecto estão comentados. Retire os comentários à medida que vai implementando os testes.
- Deverá realizar esta ficha respeitando a ordem das alíneas.
- Deve fazer a implementação no ficheiro ***graph.h***.

Enunciado

A classe **Graph** permite representar um grafo orientado, composto por **nós** ligados por **arestas**. A informação contida nos nós e arestas do grafo pode estar associada a tipos de dados diferentes. A classe **Graph** é uma classe genérica com dois argumentos, os **nós** e as **arestas**. Considere que todos os **nós** do grafo são diferentes.
Cada instância da classe **Graph** contém um **vetor de apontadores para nós**. Para cada nó, existe um **vetor de arestas** (ordenadas segundo o nó de destino). A figura seguinte mostra a estrutura de dados para um exemplo.





A declaração da classe **Graph** é a seguinte:

```
template <class N, class E>
class Node{
public:
	N info;
	vector< Edge<N,E> > edges;
	No(N inf) {
		info = inf;
	}
};

template <class N, class E>
class Edge {
public:
	E value;
	Node<N,A> *destination;
	Edge(No<N,E> *dest, E val) {
		value = val;
		destination = dest;
	}
};
```

```
template <class N, class E>
class Graph {
      vector< Node<N,E> *> nodes;
  public:
    Graph();
    ~Graph();
    Graph & addNode(const N &inf);
    Graph & addEdge(const N &begin, const N &end, const E &val);
    Graph & removeEdge(const N &begin, const N &end);
    E & edgeValue(const N &begin, const N &end);
    unsigned numEdges() const;
    unsigned numNodes() const;
    void print(std::ostream &os) const;
};
```

A implementação deve ser efetuada no ficheiro *graph.h*.

a) Implemente:
   - O construtor e o destrutor da classe *Graph*.
   - O método *numNodes()* (que retorna o número de nós do grafo).
   - O método *numEdges()* (que retorna o número de arestas existentes no grafo).

b) Implemente o membro-função *addNode(const N &inf)*, que insere um novo nó no grafo e retorna o grafo alterado (*this*). Esta função deve lançar a exceção *NodeAlreadyExists* caso esse nó já exista (ver teste unitário para esta alínea).

   A exceção *NodeAlreadyExists* já está implementada.

c) Implemente o membro-função *addEdge(const N &begin, const N &end, const E &val)*, que insere uma nova aresta no grafo e retorna o grafo alterado (*this*). Esta função deve lançar a exceção apropriada caso a aresta já exista.

   - Exceção *NodeDoesNotExist*: esta exceção já está implementada.
   - Exceção *EdgeAlreadyExists*:
     - Implemente esta exceção. Implemente o operador <<, que imprime no monitor os valores dos nós extremos da aresta

d) Implemente o membro-função *edgeValue*(*const N &begin, const N &end)*, que retorna uma referência para os dados da aresta especificada. Esta função deve lançar a exceção apropriada caso a aresta não exista no grafo (ver teste unitário para esta alínea).
   - Exceção *EdgeDoesNotExist*:
     - Implemente o operador <<, que imprime no monitor os valores dos nós extremos da aresta

e) Implemente o membro-função *removeEdge(const N &begin, const N &end)*, que elimina uma aresta do grafo e retorna o grafo alterado (*this*). Esta função deve lançar a exceção apropriada caso a aresta não exista no grafo (idêntica à da alínea anterior).

f) Implemente o membro-função *print*(*std::ostream &os)*, que escreve, para um *stream* de saída, a informação do grafo. Para o exemplo indicado anteriormente, a função deve produzir:

(A [B 5] [C 8]) (B [D 9]) (C [D 3] [E 4]) (D [B 11] [E 2]) (E)

g) Utilize a função anterior para implementar o operador de saída <<.

h) Efetue a documentação dos membros-função implementados (use Doxygen).