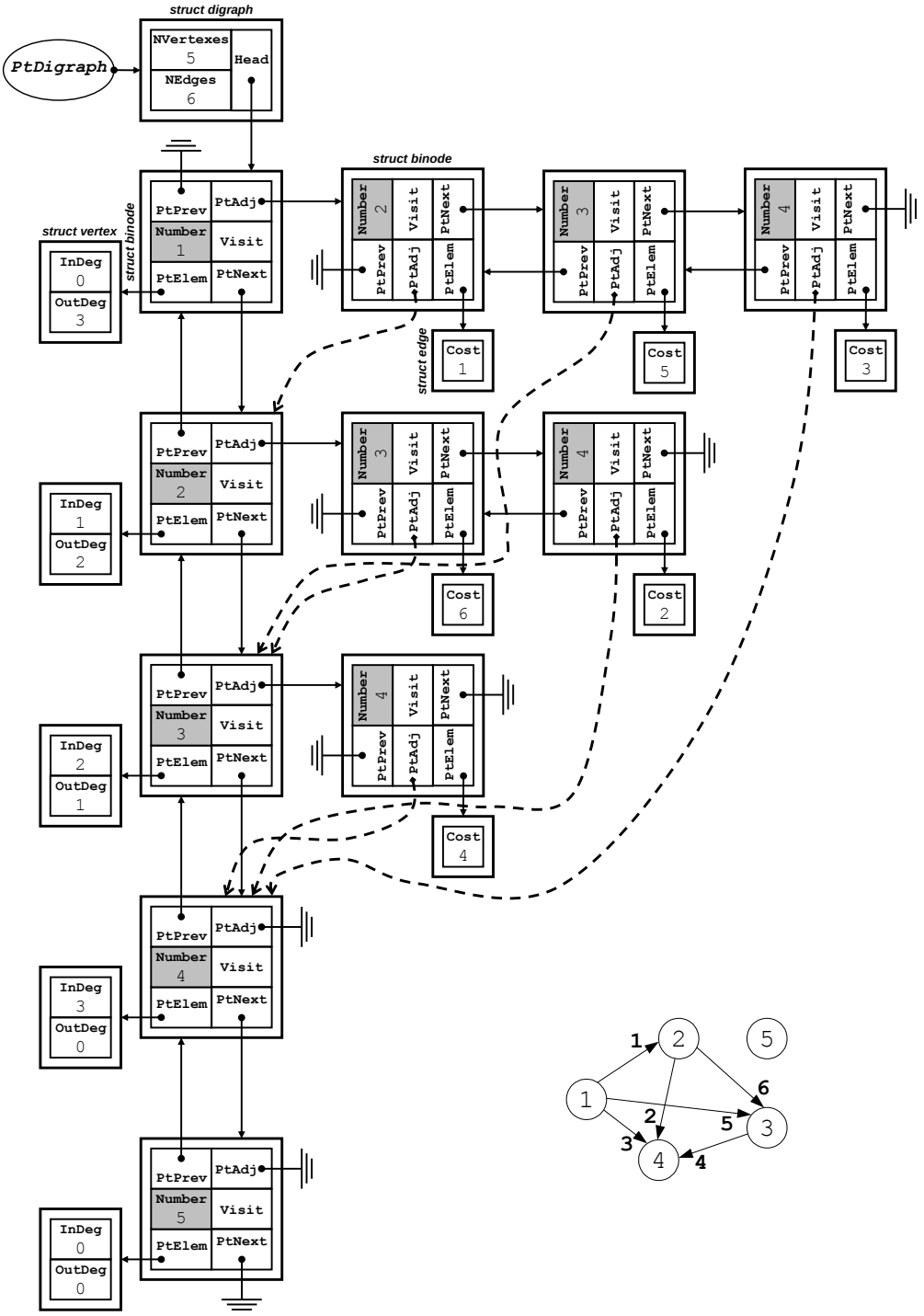
AULA 13 - DÍGRAFOS

Comece por ler o Capítulo 10 – Grafos (ver 10.3 − Implementação do Grafo, 10.4 − Caracterização do Grafo e 10.5 − Dígrafo/Grafo dinâmico, páginas 456-470). Estude o funcionamento da implementação do dígrafo/grafo dinâmico e analise a sua implementação, para se familiarizar com os algoritmos.

O tipo de dados abstrato DIGRAPH_DYNAMIC é constituído pelo ficheiro de interface **digraph.h** e pelo ficheiro de implementação **digraph.c** e implementa a manipulação de dígrafos/grafos dinâmicos, usando listas biligadas genéricas para manter as listas dos vértices e das arestas ordenadas por ordem crescente, tal como se mostra na figura seguinte.



NOME: **Nº MEC:**

Para testar o tipo de dados – as operações básicas sobre dígrafos e as operações propostas para a aula e projeto final – é fornecido o programa de simulação gráfica **simdigraph.c** e a *makefile* **mkdigraph**. Comece por testar convenientemente toda a sua funcionalidade básica.

Depois, acrescente-lhe a seguinte funcionalidade:

- Verificar de que tipo é um vértice. A função deve ter o seguinte protótipo:

```
int VertexType (PtDigraph pdig, unsigned int pv);
```

A função deve devolver: NO_DIGRAPH (se o dígrafo não existir); DIGRAPH_EMPTY (se o dígrafo estiver vazio); NO_VERTEX (se o vértice não existir); SINK se ele for um vértice sumidouro; SOURCE se ele for um vértice fonte; DISC se ele for um vértice desconexo; ou OK se for um vértice comum.

Teste a função para o **digrafo6.txt**, que é constituído por seis vértices e sete arestas, sendo que o vértice 5 é um vértice fonte, o vértice 3 é um vértice sumidouro e o vértice 6 é um vértice desconexo.

O tipo de dados providencia a função interna DijkstraPQueue que implementa o algoritmo de Dijkstra usando uma fila com prioridade baseada num amontoado binário. Esta função está simplificada, porque assume que as sequências foram previamente validadas pelas funções invocadoras. Acrescente ao tipo de dados as seguintes funções, que devem usar o algoritmo de Dijkstra:

- Determinar os caminhos mais curtos a partir de um dado vértice. Para esse efeito deve implementar uma função para executar de forma segura o algoritmo de Dijkstra. A função serve como *front-end* da função interna DijkstraPQueue e tem de validar as condições de execução da mesma. A função deve ter o seguinte protótipo:

```
int Dijkstra (PtDigraph pdig, unsigned int pv, unsigned int pvpred[],
                                                int pvcost[]);
```

A função deve devolver: NO_DIGRAPH (se o dígrafo não existir); DIGRAPH_EMPTY (se o dígrafo estiver vazio); NULL_PTR (se algum dos ponteiros para as sequências for NULL); ou NO_VERTEX (se o vértice não existir). A função devolve a lista de vértices predecessores na sequência pvpred e os custos dos respetivos caminhos na sequência pvcost. A dimensão destas sequências é igual ao número de vértices do dígrafo.

- Determinar os vértices alcançáveis a partir de um dado vértice. Esta função deve usar a função interna DijkstraPQueue e depois devolver apenas a lista dos vértices alcançáveis. A função deve ter o seguinte protótipo:

```
int Reach (PtDigraph pdig, unsigned int pv, unsigned int pvlist[]);
```

A função deve devolver: NO_DIGRAPH (se o dígrafo não existir); DIGRAPH_EMPTY (se o dígrafo estiver vazio); NULL_PTR (se o ponteiro para a sequência for NULL); NO_VERTEX (se o vértice não existir); ou NO_MEM (se não existir memória para criar as sequências necessárias para invocar o algoritmo de Dijkstra). A função devolve os vértices alcançáveis na sequência pvlist, sendo que a posição 0 da sequência indica o número de vértices alcançáveis. A dimensão desta sequência é igual ao número de vértices do dígrafo.

Comece por determinar manualmente os vértices alcançáveis e os caminhos mais curtos para todos os vértices do **digrafo6.txt**. Depois, teste estas duas funções e compare os resultados.

TRABALHO FINAL SOBRE DÍGRAFOS

Adicionar ao tipo de dados abstrato DIGRAPH_DYNAMIC funções que permitam efetuar as seguintes operações:

- Verificar se um dado **dígrafo** G é **completo**. A função deve ter o seguinte protótipo:

```
int DigraphComplete (PtDigraph pdig, unsigned int *pcomp);
```

A função atribui a `pcomp` o valor 1, se o dígrafo for completo, e o valor 0, caso contrário. E devolve os seguintes valores de retorno: OK, NO_DIGRAPH (se o dígrafo não existir), DIGRAPH_EMPTY (se o dígrafo estiver vazio) ou NULL_PTR (se o ponteiro `pcomp` for NULL).

- Construir o **dígrafo transposto** G^T de um dado dígrafo G. O dígrafo transposto G^T é um dígrafo com os mesmos vértices e com as mesmas arestas do dígrafo G, estando as arestas invertidas. A função deve ter o seguinte protótipo:

```
PtDigraph DigraphTranspose (PtDigraph pdig);
```

A função começa por criar um dígrafo nulo e de seguida insere os vértices. Depois, insere as arestas simétricas e, finalmente, devolve a referência do dígrafo criado ou NULL, no caso de inexistência de memória.

- Verificar se um dado dígrafo G é **fortemente conexo**, isto é, se existe um caminho entre qualquer par de vértices do dígrafo G. A função deve ter o seguinte protótipo:

```
int DigraphStronglyConnected (PtDigraph pdig, unsigned int *pstrong);
```

A função atribui a `pstrong` o valor 1, se o dígrafo for fortemente conexo, e o valor 0, caso contrário. E devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY, NO_MEM (se não existir memória para criar as sequências necessárias para invocar o algoritmo de Dijkstra) ou NULL_PTR (se o ponteiro `pstrong` for NULL).

- Implementar o **fecho transitivo** de um dado dígrafo G. Uma aresta (vi, vj) é inserida no dígrafo, se e só se, vj é alcançável a partir de vi e essa aresta ainda não existe no dígrafo. Considere que as novas arestas a inserir no dígrafo têm custo unitário. A função deve ter o seguinte protótipo:

```
int DigraphTransitiveClosure (PtDigraph pdig);
```

A função devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY ou NO_MEM.

Atenção:

Apesar de habitualmente considerarmos que os vértices se encontram sequencialmente numerados, com início em 1, deve implementar os algoritmos de maneira o mais versátil possível. Ou seja, deve sempre varrer e processar a lista de vértices do dígrafo.

Para construir o fecho transitivo e verificar se um dígrafo é fortemente conexo deverá obrigatoriamente utilizar a função que determina os vértices alcançáveis (função **Reach**) que foi proposta no guião das aulas práticas.

Também deve respeitar os protótipos das funções propostos para poder simular toda a funcionalidade com o programa **simdigraph.c**.

NOME: **Nº MEC:**