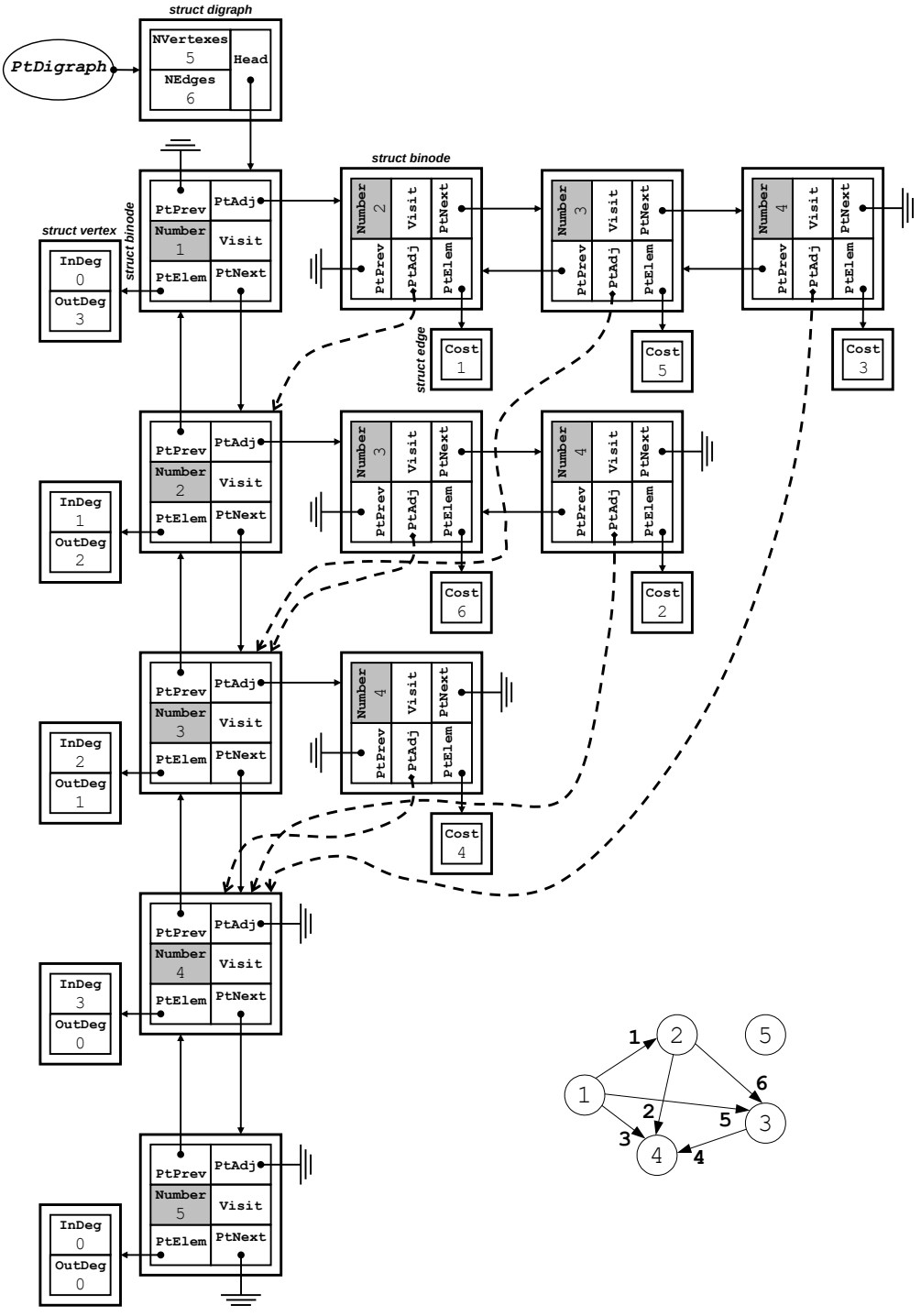
AULAS 12 E 13 - DÍGRAFOS

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 **digrafo5.txt**, que é constituído por 5 vértices e 6 arestas, sendo que o vértice 1 é um vértice fonte, o vértice 4 é um vértice sumidouro e o vértice 5 é um vértice desconexo.

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

- Determinar a classificação (entre 0.0 e 1.0) de um dado vértice de acordo com o seu grau (soma do número de antecessores e de sucessores diretos). A função deve ter o seguinte protótipo:

```
int VertexClassification (PtDigraph pdig, unsigned int pv, double *pclass);
```

A função atribui a `pclass` a classificação do vértice. E devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY, NO_VERTEX (se o vértice `pv` não existir) ou NULL_PTR (se o ponteiro `pclass` 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.

- **Subdividir uma dada aresta substituindo-a** por um vértice e duas arestas adicionais. O identificador do vértice será o próximo índice que se possa utilizar. O custo da aresta dada determina o custo da primeira aresta adicional, que é dado por floor(custo/2.0), e o custo da segunda aresta adicional, que é dado por ceil(custo/2.0) – devendo ser utilizada aritmética inteira para o cálculo destes novos custos. A função deve ter o seguinte protótipo:

```
int DigraphEdgeSplit (PtDigraph pdig, unsigned int pve, unsigned int pvi);
```

Não se esqueça de remover a aresta indicada. A função devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY, NO_MEM ou NO_EDGE (se a aresta pve→pvi não existir).

- **Cindir um dado vértice substituindo-o** por dois vértices ligados por uma aresta de custo zero. Os identificadores dos novos vértices serão os próximos índices que se possa utilizar. Os predecessores do vértice original são agora os predecessores do primeiro vértice acrescentado, e os sucessores do

vértice original são os sucessores do segundo vértice acrescentado. A função deve ter o seguinte protótipo:

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

Não se esqueça de remover o vértice indicado. A função devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY, NO_MEM ou NO_VERTEX (se o vértice `pv` não existir).

- Criar uma fila contendo os vértices que são sucessores diretos de um dado vértice e cuja distância a esse vértice é inferior a `pdst`. A função deve ter o seguinte protótipo:

```
int AllSuccDist (PtDigraph pdig, unsigned int pv, double pdst, PtQueue *pqueue);
```

A função cria uma fila de inteiros contendo a identificação dos vértices e devolve a referência da fila em `pqueue` ou NULL, caso não haja memória para criar e preencher a fila. E devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY, NULL_PTR (se o ponteiro `pqueue` for NULL), NO_VERTEX ou NO_MEM (caso não seja possível criar ou inserir elementos na fila).

- Criar uma fila contendo os vértices que definem o conjunto de arestas incidentes num dado vértice. A função deve ter o seguinte protótipo:

```
int AllInEdgesVertex (PtDigraph pdig, unsigned int pv, PtQueue *pqueue);
```

A função cria uma fila de inteiros com os pares de vértices (vértice emergente seguido do vértice incidente) que definem as arestas incidentes no vértice dado e devolve a referência da fila em `pqueue` ou NULL, caso não haja memória para criar e preencher a fila. E devolve os seguintes valores de retorno: OK, NO_DIGRAPH, DIGRAPH_EMPTY, NO_VERTEX, NULL_PTR 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.

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