1 Tendo por base as bibliotecas de estruturas de dados apresentadas em Programação 2, implemente as funcionalidades pedidas nas duas alíneas seguintes no ficheiro **prob1.c**. Sempre que conveniente utilize as funções disponíveis nas estruturas árvore AVL, heap, fila e vetor.

1.1 Implemente a função `avl_conta_letras` para uma **árvore AVL** (definida pelo nó raiz) que calcula o número total de letras guardados em todos os nós da árvore.

```
int avl_conta_letras(no_avl *no)
```

O parâmetro da função é o apontador para o nó raiz da árvore. Considere que as *strings* contêm apenas letras minúsculas (a-z).

Depois de implementada a função, o programa deverá apresentar:

```
Numero de letras: 2032
```

1.2 Implemente a função `fila_reordena` que reordena uma **fila** de acordo com as prioridades associadas, seguindo uma ordem decrescente. A função deverá utilizar uma fila de prioridade baseada em **heap** para reordenar a fila original.

```
int fila_reordena(fila *original, vetor *prioridades)
```

O primeiro parâmetro da função é o apontador para fila original e o segundo um apontador para um vetor contendo as prioridades associadas. Na posição *i* do vetor está guardada a prioridade do *i*-ésimo elemento da fila.

Os parâmetros de entrada devem ser verificados. A função deve retornar 1 se for bem sucedida e 0 em caso contrário. Indique ainda num comentário no início do código da função qual a complexidade do algoritmo que implementou e uma breve justificação (máximo 20 palavras).

Depois de implementada a função, o programa deverá apresentar:

```
AD_Leonis
Gliese_674
Lacaille_8760
...
Alpha_Centauri_B
Proxima_Centauri
```

2 Tendo por base as bibliotecas de estruturas de dados apresentadas em Programação 2, implemente as funcionalidades pedidas nas duas alíneas seguintes no ficheiro **prob2.c**. Sempre que conveniente utilize as funções disponíveis nas estruturas grafo, tabela de dispersão e vetor.

2.1 Implemente a função td_pesquisa_inversa que procura numa tabela **de dispersão** todas as ocorrências de um determinado valor e devolve um novo vetor contendo todas as chaves que têm esse valor.

```
vetor* td_pesquisa_inversa(tabela_dispersao *td, const char *valor)
```

Por exemplo, se numa tabela de dispersão existirem os pares chave/valor Lisboa/Portugal e Porto/Portugal, a pesquisa inversa pelo valor "Portugal" deve retornar um vetor contendo "Lisboa" e "Porto". Os parâmetros da função são a tabela de dispersão e o valor em relação ao qual se pretende pesquisar as chaves. A função deve retornar o vetor contendo as chaves se for bem sucedida (ainda que o mesmo possa ser vazio) ou NULL em caso contrário, incluindo erro nos parâmetros.

Depois de implementada a função, o programa deverá apresentar:

```
Netherlands airports:
1: Eindhoven
2: Maastricht
```

2.2 Implemente a função grafo_uniao que, dados dois grafos dirigidos com o mesmo número de vértices, cria e devolve um novo grafo correspondente à união dos dois. Considere que a união de dois grafos dirigidos ***g1*** e ***g2*** com o mesmo número de vértices é um grafo dirigido que tem também o mesmo número de vértices, mas cujo conjunto de arestas é formado pela união das arestas de ***g1*** e ***g2***. (ou seja, todas as arestas que existem em ***g1*** ou em ***g2*** ou em ambos).

```
grafo* grafo_uniao(grafo *g1, grafo *g2)
```

Os parâmetros da função são os grafos a partir dos quais se pretende obter a união. A função deve retornar o novo grafo se for bem sucedida ou NULL em caso contrário, incluindo erro nos parâmetros. Indique ainda num comentário no início do código da função qual a complexidade do algoritmo que implementou e uma breve justificação (máximo 20 palavras).

Depois de implementada a função, o programa deverá apresentar:

```
0: 4
1: 4->0->5
2: 3->1
3: 2
4: 3
5: 4
6: 2->3
7: 3->2->6
```