Miniteste 2 – Exemplo 2 (adaptado de miniteste 2013/14)

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, tabela de dispersão e lista.

1.1 Implemente a função `avl_imprime_ord` que apresenta o conteúdo de uma árvore AVL de forma ordenada:

```
void avl_imprime_ord(arvore_avl* avl)
```

O primeiro parâmetro da função é o apontador para a árvore e o segundo parâmetro é a ordem pela qual deve ser ordenada (crescente se for `>= 0` e decrescente se for `< 0`).

Depois de implementada a função, o programa deverá apresentar:

```
Árvore original: GER ENG BIH BEL CRO FRA ESP NED ITA GRE RUS POR SUI
Ordem alfabética crescente: BEL BIH CRO ENG ESP FRA GER GRE ITA NED
POR RUS SUI
```

1.2 Pretende-se implementar uma função de *login* que utiliza uma tabela de dispersão para armazenar os pares *login* / *password* (contidos num ficheiro).

```
void valida_passwds(FILE *f, lista *login, lista *passwd)
```

Os parâmetros da função são o apontador para o ficheiro que contém os pares login / password e duas listas contendo o *login* e *passwords* a validar. A tabela deve ter tamanho 10 e utilizar a função de dispersão hash_djbm. Considere que no máximo o *login* e *password* têm 25 caracteres.

A função deve imprimir a validação dos *logins*, isto é, se cada par *login / password* está de acordo com os registos contidos do ficheiro.

Depois de implementada a função, o programa deverá apresentar:

```
Login: maria
Password: contrary
Authentication succeeded
Login: tiago
Password: contrary
Authentication failed
Login: ricardo
Password: sheeplost
Authentication failed
```

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, heap e lista.

2.1 Implemente a função `descobre_caminho` que retorna, caso exista, o caminho mínimo entre os vértices origem e destino, passando pelo vértice obrigatório. O caminho deve ser retornado numa lista.

```
lista* descobre_caminho (grafo *g, int origem, int destino, int
                         obrigatorio)
```

Os parâmetros da função são o apontador para o grafo e os vértices a considerar. Depois de implementada a função, o programa deverá apresentar:

```
Descobre caminho – Início 1, Fim 7, Passando por 5
Caminho: 13257
```

2.2 Pretende-se implementar uma função de simulação que utiliza uma fila de prioridade baseada em heap para armazenar os pares acontecimento/tempo. Os pares ação / tempo (em segundos) estão guardadas num ficheiro.

```
int simula_acoes(lista *acoes, lista *tempos, int n)
```

Os dois primeiros parâmetros da função são apontadores para as listas (não ordenadas) de ações e tempos. Um par ação/tempo corresponde à ação na posição *x* da lista de ações e o tempo na mesma posição *x* da lista de tempos. O terceiro parâmetro é o número de ações a imprimir.

A função deve retornar 1 se for bem sucedida e 0 em caso contrário. Considere que no máximo existem 20 ações e que o nome do acontecimento tem 20 caracteres.

Depois de implementada a função, o programa deverá apresentar:

```
1: direita
2: esquerda
3: esquerda
4: baixo
```