Aula prática 9

Esta aula tem como objetivo estudar filas de prioridade e heaps e analisar o seu desempenho. O algoritmo de ordenação Heapsort, que recorre a um heap, é também abordado.

Pretende-se implementar uma fila de prioridade com a ajuda de dois tipos diferentes de estruturas de dados: o vetor ordenado e heap. Para tal deverá implementar as funções referidas neste enunciado no ficheiro **priorityqueue.c**.

1. Comece por implementar a fila de prioridade com a ajuda da biblioteca vetor, em que o vetor deverá estar ordenado. Deverá criar as funções de inserção de um novo elemento na fila de prioridade e de remoção do próximo elemento da fila de prioridade:

int *fp_insere*(**vetor *** *fp*, const **char *** *valor*, **int** *prioridade*)

char * *fp_remove*(**vetor *** *fp*)

A função *fp_insere* deve verificar se o tamanho do vetor não é excedido e inserir o novo elemento ordenando por prioridade (caso haja espaço). Sempre que é necessário comparar os elementos, deverá usar a função *maior_que*.

A função *fp_remove* deve verificar se o vetor não está vazio, retornando e removendo o elemento com maior prioridade. A posição onde estava o elemento removido deverá apontar para NULL. Em caso de erro deve retornar NULL.

2. Pretende-se agora implementar uma fila de prioridade usando um heap implementada em vetor. Implemente a função de inserção de um novo elemento e a função de remoção do próximo elemento do *heap:*

int *heap_insere*(**vetor *** *h,* **compara** *cmp*, const **char *** *texto*, **int** *prioridade*)

char * *heap_remove*(**vetor *** *h,* **compara** *cmp*)

O argumento ***cmp*** é um apontador para a função de comparação de elementos que deve ser sempre usada para comparar dois elementos, por exemplo: *cmp(elemento1, elemento2)*

A função *heap_insere* deve verificar se o tamanho do vetor não é excedido e inserir o novo elemento no *heap* (caso haja espaço). A raiz do *heap* deve estar na posição 1 do vetor.

A função *heap_remove* deve verificar se o vetor não está vazio, retornando e removendo o elemento com maior prioridade. A posição onde estava o elemento removido deverá apontar para NULL. Em caso de erro deve retornar NULL.

2.1. Analise os valores obtidos para as comparações necessárias para inserir um novo elemento. Qual esperaria que fosse o valor médio de comparações em ambas as implementações? E no pior caso?

3. ***Heapsort*** é um algoritmo de ordenação que utiliza um **heap** como estrutura auxiliar do processo de ordenação. Sem alterar as funções utilizadas no exercício 2 (heap_carrega, heap_insere, heap_remove e maior_que) é possível utilizar o ***heap*** para ordenar um vetor de *strings* por ordem alfabética. No ficheiro vetor**.c** complete a função *maior_que_HeapSort()* de modo a utilizar o ***heap*** para ordenar um vetor de *strings* por ordem alfabética crescente.

4. Crie um programa que faça uso da fila de prioridade implementada com heap (exercício 1.2) e que utilize o ficheiro **movies.csv** para implementar uma fila de prioridade sobre os filmes melhores classificados de acordo com o IMDB. Tenha em consideração o seguinte:
   - Deve alterar a função de leitura para que seja capaz de processar o ficheiro fornecido.
   - Tenha em atenção que neste caso, ao contrário do problema anterior, uma maior prioridade significa que o valor numérico do ranking é menor. Deverá implementar esta funcionalidade sem alterar as funções heap_insere e heap_remove.
   - O programa deverá imprimir o *ranking* de filmes em ordem crescente. Atente que a impressão efetuada pela função vetor_imprime não fornece o ranking desejado.