Algoritmos e Complexidade
LEI/LCC (2º ano)

8ª Ficha Prática

O objectivo desta ficha é o estudo da análise agregada de algoritmos.

1. Relembre a definição de *heap* na seguinte definição de heaps dinâmicas. Os campos `size` e `used` serão usados para guardar o número máximo de elementos que a heap pode armazenar e o número de elementos que a heap tem armazenados.

   ```
   typedef struct {
    int   size;
    int   used;
    int  *heap;
   } Heap;
   ```

   Considere que se encontram definidas as funções `bubbleUp` e `bubbleDown` de manuseamento de heaps que executam em tempo $O(\log(N))$ para uma heap de tamanho N.

   A inserção numa heap vai então ser feita da seguinte forma:

   - Se a heap não estiver cheia (i.e., `used` menor do que `size`) o novo elemento é acrescentado no fim e será feito `bubbleUp` para o colocar no local apropriado.
   - Se a heap estiver cheia, é primeiro alocado espaço para uma heap com o dobro da capacidade, copiado o conteúdo da heap anterior para a nova, libertado o espaço da heap anterior e efectuado o procedimento anterior.

   A remoção de um elemento (o menor) de uma heap será feita da seguinte forma:

   - Começa-se por remover o elemento da posição 0, colocando lá o valor do último, e aplicando a função `bubbleDown`.
   - Se após esta a heap tiver 75% da sua capacidade não ocupada, é alocado espaço para uma heap com metade da capacidade, copiado o conteúdo da heap anterior para a nova e libertado o espaço desta última.

   (a) Apresente uma implementação das operações `insertHeap` e `extractMin`

   ```
   Heap insertHeap(Heap *h, Int x);   // Insere um elemento na heap
   Heap extractMin(Heap *h, Elem *x);  // Retira o mínimo da heap
   ```

   (b) Mostre que para uma sequência de N inserções ou remoções, o custo amortizado de cada uma destas operações permanece igual a $O(\log(N))$

2. Uma implementação possível de uma fila de espera (*Queue*) utiliza duas pilhas A e B, por exemplo:

   ```
   typedef struct queue {
    Stack a;
    Stack b;
   } Queue;
   ```

- A inserção (`enqueue`) de elementos é sempre realizada na pilha A;
- para a remoção (`dequeue`), se a pilha B não estiver vazia, é efectuado um *pop* nessa pilha; caso contrário, para todos os elementos de A excepto o último, faz-se sucessivamente *pop* e *push* na pilha B. Faz-se depois *pop* do último, que é devolvido como resultado.

(a) Efectue a análise do tempo de execução no melhor e no pior caso das funções `enqueue` e `dequeue`, assumindo que todas as operações das pilhas são realizadas em tempo constante.

(b) Mostre que o custo amortizado de cada uma das operações de `enqueue` ou `dequeue` numa sequência de N operações é $O(1)$

3. Uma `quack` é uma estrutura que combina as funcionalidades de uma `queue` com as de uma `stack`. Pode ser vista como uma lista de elementos em que são possíveis três operações:

   - `push` que adiciona um elemento;
   - `pop` que remove o último elemento inserido;
   - `pull` que remove o elemento inserido há mais tempo.

   Apresente uma implementação de *quack*s usando 3 `stacks` garantindo que o custo amortizado de cada uma das três operações é $O(1)$, assumindo que todas as operações das `stacks` são realizadas em tempo constante.

4. Uma *multistack* é uma lista de stacks s_0, s_1, ..., s_k, na qual a stack s_i pode acomodar no máximo 3^i elementos. A inserção e remoção (`mPush` e `mPop`) fazem-se na stack s_0.

   - sempre que se pretende inserir um elemento numa stack que está cheia, começa-se por transferir todos os elementos para a stack seguinte (criando-a se não existir) e inserindo o dito elemento de seguida.
   - sempre que ao remover um elemento de uma stack, ela ficar vazia, ela é cheia com elementos da stack seguinte, ou removida (caso seja a última).

   Todas as operações com as stacks constituintes são feitas com as habituais operações de `pop` e `push` que se assume que executam em tempo constante.

   (a) Qual o custo, no pior caso, de inserir um novo elemento numa stack com N elementos?

   (b) Mostre que o custo amortizado da operação `mPush` é $O(\log(N))$ onde N é o número de elementos após a última inserção.