ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Folha de Exercícios 2

Algoritmos e Estruturas de Dados

Licenciatura em Engenharia de Sistemas Informáticos

1. As stacks são estruturas de dados dinâmicas que seguem uma filosofia LIFO (Last in, First Out), em que os dados são organizados como uma pilha. Existem três métodos, um designado *push*, para colocar um elemento no topo da pilha, um método *top*, para obter o valor no topo da pilha, e um método *pop* que retira o elemento do topo da pilha.

   (a) Implemente uma estrutura de dados para guardar uma pilha de inteiros;

   (b) Implemente as funções indicadas. Garanta que todas têm complexidade $\mathcal{O}(1)$.

   (c) Implemente uma função para cada uma das quatro operações algébricas básicas (soma, subtração, divisão e multiplicação), que obtêm dois elementos da stack, calculam a operação em causa, e colocam o resultado novamente na stack.

   Exemplo: para calcular $4 + 3 - 2$ será executado:

   ```
   stack = push(stack, 2);
   stack = push(stack, 3);
   stack = push(stack, 4);
   stack = add(stack);
   stack = sub(stack);
   printf("Result: %d\n", top(stack));
   ```

2. Outra estrutura de dados bem conhecida é a queue. Esta estrutura, também designada por fila de espera, segue a filosofia FIFO (first in, first out), e simula uma fila: o primeiro a chegar, é o primeiro a sair! Neste contexto existem três operações típicas, *enqueue*, que coloca um elemento no final da fila, *dequeue*, que remove o elemento do início da fila, e o *peek*, que obtém o valor do elemento no início da fila.

   Considerando a relevância desta estrutura de dados, seria importante a sua implementação de forma genérica, permitindo que fosse usada para qualquer tipo de dados (sejam apenas strings, ou registos de funcionários, ou imagens).

   Para isso, considere que cada posição da lista é representada por:

   ```
   typedef struct _queue {
       void *data;
       struct _queue *next;
   } Queue;
   ```

   Nesta estrutura, o apontador `void*` aponta para uma zona de memória onde os dados estão.

   (a) Implemente a função para a adição de um item ao fim da fila de espera:

   ```
   Queue* enqueue(Queue* queue, void* data)
   ```

   Considerando que o tipo de dados é desconhecido, a função não deve copiar os dados, mas apenas colocar a célula da queue a apontar para a memória explicitada.

   (b) Implemente a função para obter o primeiro elemento da fila de espera:

   ```
   void* peek(Queue* queue)
   ```

   (c) Implemente a função que remove o primeiro elemento da fila de espera:

```
Queue* dequeue(Queue* queue)
```

Segue-se um exemplo de uso:

```
char *nome1 = strdup("Alberto Simoes");
char *nome2 = strdup("Mario Joao Vale");
Queue *queue = NULL;
queue = enqueue(queue, nome1);
queue = enqueue(queue, nome2);
printf("Primeiro da lista: %s\n", (char*) peek(queue));
queue = dequeue(queue);
printf("Proximo da lista: %s\n", (char*) peek(queue));
```

Tome em atenção a necessidade de castings explícitos.

3. A principal vantagem de uma lista duplamente ligada é o facto de poder ser percorrida em ambas as direções. Implemente uma lista duplamente ligada para registar o número de infetados por COVID-19 por distrito.

   (a) Cada célula deve conter o distrito (nome) e o número de infetados;

   (b) Deverá ser possível listar os distritos ordenando pelo número de infetados (quer por ordem crescente, quer por ordem decrescente).

   (c) A função de adição deverá registar um novo distrito, caso ainda não exista, ou permitir a atualização do número de infetados.

   (d) Adicione a possibilidade de, para cada distrito, existir uma lista de municípios e o número de infetados. Ao adicionar um município, será indicado o seu distrito, e o número de infetados. Deverá atualziar os dados na lista de municípios, e atualizar o valor na lista de distritos.