IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

ATAD 2019/20

Algoritmos e Tipos Abstratos de Dados

Enunciado de Laboratório

AVALIADO

4

Pesquisa binária (abordagem iterativa e recursiva)

Objetivos

- Implementação de algoritmos de pesquisa Binária, na abordagem iterativa e recursiva.
- Análise e interpretação de algoritmo expresso em pseudocódigo.
- Documentação *Doxygen*.

Pesquisa Binária

Numa sequência ordenada é possível otimizar a pesquisa utilizando uma estratégia de partição sucessiva da sequência em duas metades, para diminuir o número de elementos a pesquisar.

Algoritmo:

- Selecionar o elemento central da sequência e comparar com o valor procurado;
- Se o elemento selecionado for menor que o procurado, podemos excluir a primeira metade; caso contrário excluímos a segunda metade.

Na *figura 1*, podemos verificar um caso de sucesso de uma pesquisa binária pelo valor 19:

1. Na primeira iteração, calcula-se o elemento médio da sequência através da divisão inteira da soma das posições mínima e máxima (start e end), que neste caso é o elemento de índice 5.
2. Como o valor procurado é 19, é maior do que o valor armazenado no elemento médio, significa que se encontra na segunda metade da sequência, passando a posição mínima (start) para o elemento à direita da posição média, neste caso, 11.
3. A análise repete-se, desta vez, o valor procurado é menor do que o valor do elemento médio, pelo que este se vai encontrar na primeira metade, passando por isso, o valor máximo (end) para o elemento à esquerda do valor médio.

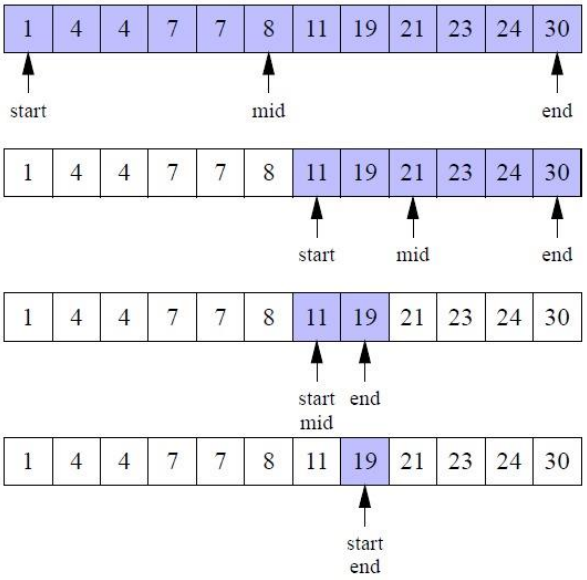

Figura 1 - Pesquisa Binária (exemplo valor encontrado)

4. Quando a posição média coincide com o valor procurado, a pesquisa termina com sucesso, devolvendo o índice desse elemento (neste caso, índice 7).

De seguida, iremos analisar o pseudocódigo referente ao algoritmo descrito:

```
Algorithm binarySearch
    input: val - integer
           arr - ordered (ascending) array of integers
           start - start index for arr, natural number
           end - end index for arr, natural number
    output: arr index of 'val'; -1 if not found - integer
BEGIN
    mid <- (start + end) / 2 //divisão inteira

    IF start > end THEN //caso base 1
        RETURN -1
    ELSE IF arr[mid] = val THEN //caso base 2
        RETURN mid
    ELSE IF arr[mid] > val THEN
        RETURN binarySearch(val, arr, start, mid - 1)
    ELSE
        RETURN binarySearch(val, arr, mid + 1, end)
    END IF
END
```

Figura 2 - pseudocódigo referente à implementação do algoritmo de pesquisa binária (recursiva)

Notas importantes:

Deverá existir um `makefile` que permita compilar os níveis individuais no executável `prog` e com a *flag* -g (símbolos de *debug*), e.g.:

```
n1:
    gcc -o prog -g main1.c
n2:
    gcc -o prog -g main2.c
...
```

Nível 1

1. Crie um novo projeto VS Code (uma pasta no sistema de ficheiros) com os ficheiros `main1.c`, onde deverá constar todo o código desenvolvido no respetivo nível identificado.
2. No ficheiro `main1.c`, codifique a função `main()` por forma a testar todo o código desenvolvido.
3. Declare um vetor **`intArray`** com um valor constante de 10 elementos inteiros.
4. Implemente a função **`randomArray(int *intArray, int size)`** que deve inicializar cada posição do vetor recebido com um valor aleatório [0,99] (sugestão: utilize a função srand() de forma a gerar valores pseudo-aleatórios).
5. Implemente a função **`intArrayPrint(int *intArray, int size)`** que recebe um vetor de inteiros e imprime os seus valores.
6. Implemente também a função **`arraySort(int *intArray, array size)`** que ordena os valores do array recebido, de forma crescente, com recurso a um algoritmo de ordenação conhecido (Selection Sort/Bubble Sort).
7. Teste o código desenvolvido (ver output esperado).

Exemplos de resultados:

```
Vetor gerado aleatoriamente:
6 3 22 59 97 12 90 92 44 18

Vetor ordenado de forma crescente:
3 6 12 18 22 44 59 90 92 97
```

Nível 2

1. Crie o ficheiro main2.c com o código resultante de main1.c.
2. Traduza o algoritmo de pesquisa binária (recursiva) para a linguagem C.
3. Peça ao utilizador que introduza, através do terminal, o valor (inteiro) que pretende pesquisar no vetor gerado de forma aleatória.
4. Utilize este valor para efetuar a pesquisa com recurso ao algoritmo implementado, no vetor gerado.
5. Trate o retorno do algoritmo de pesquisa de forma a informar o utilizador do sucesso/insucesso da procura (ver output esperado).

Exemplos de resultados:

```
Vetor ordenado de forma crescente:
3 6 12 18 22 44 59 90 92 97

Insira um valor a procurar no vetor: 22

Valor 22 encontrado na posição 4 do vetor
```

Nível 3

1. Crie o ficheiro `main3.c` e implemente a função **`main()`** de forma a testar todo o código desenvolvido.
2. Declare e inicialize um vetor **`alphabet`** que contém, de forma ordenada e crescente, as 26 letras existentes no alfabeto latino.
3. Transponha para este ficheiro o algoritmo de pesquisa implementado no nível anterior (nível 2). Analise e proceda às respetivas alterações necessárias para a utilização deste algoritmo com o vetor **`alphabet`**.
4. Com recurso ao mesmo, determine em que posição se encontra, no alfabeto, a letra "**r**" (ver output esperado).

Exemplos de resultados:

```
Letra 'r' encontrada na posição 18 do alfabeto
```

Nível 4

1. Crie um módulo **`stock`** onde defina o tipo **`Product`** para armazenar os atributos referentes a um produto. Este é composto por designação, referência (valor numérico de 9 dígitos) e preço.
2. Adicione ao módulo o tipo **`Stock`**, composto por um produto (**`Product`**) e quantidade. Adicione a função **`Stock stockCreate(char *designation, int reference, float price, int quantity)`** que devolve uma instância do tipo **`Stock`**.
3. Adicione ao módulo stock a função **`void stockPrint(Stock s`**) que recebe um stock e o imprime no formato:

```
Product[designation] - ref: [reference] - price: [price]€ ], qtt: [quantity]
```

4. Adicione ainda a função **`stockArrayPrint(Stock *stockArr, int size)`** que recebe um vetor com todo o stock e o lista.
5. Crie o ficheiro **`main4.c`**, implemente a função **`main()`** de forma a testar todo o código desenvolvido.
6. Declare e inicialize um vetor **`stock`** com um tamanho constante de 10 elementos do tipo **`Stock`**.
7. Valide o código criado, com a impressão do vetor (ver output esperado).

Exemplos de resultados:

```
All products in stock:
        Product[ Cerveja         - ref: 112348654 - price: 0.90€ ], qtt: 30
        Product[ Cafe            - ref: 126789345 - price: 0.40€ ], qtt: 25
        Product[ Agua            - ref: 306982361 - price: 0.50€ ], qtt: 52
        Product[ Limonada        - ref: 401872229 - price: 1.50€ ], qtt:  3
        Product[ Cha             - ref: 500034544 - price: 0.90€ ], qtt:  1
        Product[ Vinho           - ref: 521222345 - price: 1.30€ ], qtt: 16
        Product[ Sumo laranja    - ref: 603481993 - price: 2.10€ ], qtt: 14
        Product[ Leite           - ref: 706787878 - price: 0.60€ ], qtt:  8
        Product[ Galao           - ref: 800045663 - price: 0.85€ ], qtt: 15
        Product[ Agua com gas    - ref: 910226773 - price: 1.65€ ], qtt: 22
```

Nível 5

```
Algorithm orderedSearch
    input: val - integer
           arr - ordered (ascending) array of integers
           start - start index for arr, natural number
           end - end index for arr, natural number
    output: arr index of 'val'; -1 if not found - integer
BEGIN
    WHILE start <= end DO
      mid <- (start + end) / 2 //divisão inteira

      IF arr[mid] = val THEN //caso base 2
          RETURN mid
      ELSE IF arr[mid] > val THEN
          end <- mid - 1
      ELSE
          start <- mid + 1
      END IF
    END WHILE
    RETURN -1
END
```

Figura 3 - pseudocódigo referente à implementação do algoritmo de pesquisa binária (iterativa)

1. Crie o ficheiro main5.c com o conteúdo de main4.c.
2. Traduza o algoritmo de pesquisa binária (iterativo) para a linguagem C.
3. Adapte o algoritmo de forma a procurar informação relativa a um produto no vetor stock, através da sua referência.
4. Apresente a informação relativamente à procura do produto, bem como a quantidade existente do mesmo (ver output esperado).
5. Forneça a documentação **doxygen** para as funções desenvolvidas em todos os níveis e gere a documentação.

Exemplos de resultados:

```
Produto com referência 603481993 encontrado em stock
        - Existem disponíveis 14 unidades
```

(fim de enunciado)