Algoritmos e Lógica de Programação: Algoritmos de ordenação e busca

Resumo

Referências Bibliográficas

- ARAÚJO, Sandro de. **Lógica de Programação e Algoritmos [recurso eletrônico]**. Contentus: Curitiba, 2020.
- FORBELLONE, André Luiz Villar e EBERSPACHER, Henri Frederico. **Lógica de Programação: a construção de algoritmos e estruturas de dados.** 3ª Edição. Prentice Hall: São Paulo, 2005.

Vamos ver aqui algoritmos de ordenação e busca. Existem vários tipos diferentes de algoritmos de ordenação como o bubblesort, quicksort, mergesort, selectionsort, radixsort, dentre vários outros. Cada algoritmo de ordenação varia desde a sua simplicidade na escrita do código com menos recursos computacionais até sua complexidade com mais recursos computacionais.

Há também diversos tipos de algoritmos de busca como a busca sequencial, a busca binária, a busca com árvores binárias, a busca em árvores balanceadas, dentre vários outros. Cada algoritmo de busca varia desde a sua simplicidade até sua complexidade no desenvolvimento do algoritmo com mais ou menos recursos computacionais e de lógica.

Ordenação em Algoritmos

Um dos algoritmos de ordenação e um dos mais simples de se entender e de desenvolver o algoritmo é o Bubble Sort. O Bubble Sort é um algoritmo que precisa de duas estruturas de repetição encadeada para o seu desenvolvimento e usa o conceito de que o número maior vai flutuando até o final de uma lista ou o número menor até o início da lista.

Vamos exemplificar o funcionamento do bubble sort com uma lista com cinco números inteiros.

Bubble Sort

Perceba que aqui, a cada para de números consecutivos, o maior deles sempre termina a comparação do lado direito e ao final, garantimos que o número 84 está na posição correta, ou seja, sendo o quinto elemento da lista:

15	*36*	*27*	*10*	*84*	
15	*36*	*27*	*10*	*84*	*compara 15 e 36 – não efetua a troca*
15	*36*	*27*	*10*	*84*	*compara 36 e 27 – realiza a troca*
15	*27*	*36*	*10*	*84*	*compara 36 e 10 – realiza a troca*
15	*27*	*10*	*36*	*84*	*compara 36 e 84 – não realiza a troca*
15	*27*	*10*	*36*	*84*	*final do primeiro processo*

Aqui, a comparação leva o número 36 até a sua quarta posição:

15	*27*	*10*	*36*	*84*	*compara 15 e 27 – não efetua a troca*
15	*27*	*10*	*36*	*84*	*compara 27 e 10 – efetua a troca*
15	*10*	*27*	*36*	*84*	*compara 27 e 36 – não efetua a troca*
16	*10*	*27*	*36*	*84*	*final do segundo processo*

Aqui, a comparação leva o número 27 até a sua terceira posição:

15	*10*	*27*	*36*	*84*	*compara 15 e 10 – efetua a troca*
10	*15*	*27*	*36*	*84*	*compara 15 e 27 – não efetua a troca*
10	*15*	*27*	*36*	*84*	*final do terceiro processo*

Aqui, a comparação leva o número 15 até a sua segunda posição e termina, já que o número 10 já termina em sua primeira posição:

10	*15*	*27*	*36*	*84*	*compara 10 e 15 – não efetua a troca*
10	*15*	*27*	*36*	*84*	*final do último processo*

O módulo do algoritmo BubbleSort recebe uma lista com os números a serem ordenados e segue o processo acima para a ordenação dos números da lista:

```
BubbleSort (num[ ] numérico_inteiro)
  início_módulo
   Declarar
    constante n <- num.tamanho numérico_inteiro;
    temp, i, j numérico_inteiro;

     para i de 0 até n-2 passo +1 faça
       para j de 0 até n-2-i passo +1 faça
          se (num[j] > num[j+1])
            então
              temp <- num[j];
              num[j] <- num[j+1];
              num[j+1] <- temp;
          fimse;
       fimpara;
     fimpara;

fim_módulo;
```

Estrutura de Ordenação em Algoritmos

Para praticar a estrutura de ordenação, vamos desenvolver um algoritmo que ordena 10 números inteiros, utilizando a técnica do bubblesort.

Perceba que, neste caso, o algoritmo precisa receber dez números inteiros do usuário, armazená-los em um vetor de 10 posições inteiras, aplicar o módulo do BubbleSort e apresentar as informações ordenadas.

Primeiro, escrevemos o módulo BubbleSort que ordena os números:

```
BubbleSort (num[ ] numérico_inteiro)
  início_módulo
   Declarar
    constante n <- num.tamanho numérico_inteiro;
    temp, i, j numérico_inteiro;

    para i de 0 até n-2 passo +1 faça
      para j de 0 até n-2-i passo +1 faça
        se (num[j] > num[j+1])
          então
            temp <- num[j];
            num[j] <- num[j+1];
            num[j+1] <- temp;
        fimse;
      fimpara;
    fimpara;

fim_módulo;
```

Agora, desenvolvemos o algoritmo que declara um vetor de inteiros com 10 posições, recebe 10 inteiros e armazena esses elementos no vetor, utilizando uma estrutura de repetição. Depois, faz a chamada do

BubbleSort, enviando o vetor com os números recebidos do usuário e, ao final, apresenta os números ordenados:

```
Algoritmo OrdenacaoBolha
  início_algoritmo
   Declarar
    i, num [10] numérico_inteiro;
    res <- "" alfanumérico;

     para i de 0 até 9 passo +1 faça
       escrever("Digite número inteiro");
       ler(num[i]);
     fimpara;

    BubbleSort(num);

    para i de 0 até num.tamanho - 1 passo +1 faça
       res ← res + num[i] + " ";
     fimpara;
    escrever(res);
fimalgoritmo.
```

Busca em Algoritmos

Um dos algoritmos de busca e um dos mais simples de se entender e de desenvolver o algoritmo é a busca sequencial. Na busca sequencial, o algoritmo vai passando por cada posição da lista, iniciando no início da lista, comparando se a informação da lista é a informação buscada. Caso encontre, a busca é parada e a informação é apresentada ou utilizada. Pode acontecer de percorrer a lista toda e o elemento não seja encontrado na lista.

Vamos exemplificar o funcionamento da busca sequencial com uma lista com 5 números inteiros. Vamos buscar o número 10 que existe na lista.

Busca Sequencial do 10

Perceba que aqui, o número 10 a ser buscado é comparado com o primeiro elemento, depois com o segundo e depois com o terceiro. Para cada um destes elementos, a busca é um insucesso. Porém, ao comparar o quarto elemento, a busca é bem-sucedida e as informações são apresentadas:

15 36 27 10 84

compara 15 – não encontrou

compara 36 – não encontrou
compara 27 – não encontrou
compara 10 – encontrou, apresenta

Perceba que aqui, o número 20 a ser buscado é comparado com cada um dos cinco elementos da lista, até finalizar a lista e o emento 20 não é encontrado. Para cada um destes elementos, a busca é um insucesso e a lista termina, apresentando a informação de que o elemento não foi encontrado.

Busca Sequencial do 20
15 36 27 10 84
compara 15 – não encontrou
compara 36 – não encontrou
compara 27 – não encontrou
compara 10 – não encontrou
Compara 84 – não encontrou

O módulo do algoritmo BuscaSequencial recebe um número a ser buscado e uma lista com números com os elementos a serem comparados e segue o processo acima para a busca de um elemento de uma lista:

```
BuscaSequencial (n numérico_inteiro, num[ ] numérico_inteiro)
  início_módulo
   Declarar
    Achou <- falso lógico;
    i <- 0 numérico_inteiro;
   enquanto ((i < num.tamanho) e (não Achou)) faça
      se (n = num[i])
        então
          Achou <- verdadeiro;
       fimse;
       i <- i + 1;
   fimenquanto;

   se (Achou)
     então
        escrever (num[i-1]);
      senão
        escrever (“não encontrado”);
   fimse;

fim_módulo;
```

Estrutura de Busca em Algoritmos

Para praticar a estrutura de busca, vamos desenvolva um algoritmo que recebe um número e busca essa informação num vetor de inteiros, utilizando a técnica da Busca Sequencial.

Perceba que, neste caso, o algoritmo precisa receber uma lista e um número fornecidos pelo usuário, aplicar o módulo de BuscaSequencial e apresentar a informação se o elemento está ou não na lista.

Primeiro, escrevemos o módulo BuscaSequencial que verifica se um elemento está numa lista:

```
BuscaSequencial (n numérico_inteiro, num[ ] numérico_inteiro)
  início_módulo
   Declarar
    Achou <- falso lógico;
    i <- 0 numérico_inteiro;
   enquanto ((i < num.tamanho) e (não Achou)) faça
      se (n = num[i])
        então
          Achou <- verdadeiro;
       fimse;
       i <- i + 1;
   fimenquanto;

   se (Achou)
     então
        escrever (num[i-1]);
      senão
        escrever (“não encontrado”);
   fimse;

fim_módulo;
```

Agora, desenvolvemos o algoritmo que declara um vetor de inteiros com 10 posições, recebe 10 inteiros e armazena esses elementos no vetor, utilizando uma estrutura de repetição. Recebe o número que será verificado se está na lista. Depois faz a chamada do BuscaSequencial enviando o número a ser verificado e

o vetor com os números recebidos pelo usuário e, ao final, apresenta se o número foi ou não encontrado na lista de números do vetor:

```
Algoritmo BuscaSeq
  início_algoritmo
   Declarar
    x, vet [10], num numérico_inteiro;

    para x de 0 até 9 passo +1 faça
      escrever("Digite número inteiro");
      ler(vet[x]);
    fimpara;

     escrever (“digita um número para ser buscado”);
    ler (num);
    BuscaSequencial(num, vet);

fimalgoritmo.
```

Exercícios

1. Assinale a alternativa CORRETA sobre o algoritmo de ordenação que utiliza a ideia de flutuar o maior valor até o final de sua lista, seguindo a mesma ideia a cada sublista:
   a) BubbleSort.
   b) QuickSort.
   c) MergeSort.
   d) SelectionSort.
   e) RadixSort.

2. Sobre o algoritmo de ordenação BubbleSort é CORRETO afirmar que:
   a) é o algoritmo que usa o conceito de que a ordenação acontece das pontas para o centro da lista.
   b) é o algoritmo que usa o conceito de que a ordenação precisa acontecer de uma lista já ordenada.
   c) é o algoritmo que usa o conceito de que é melhor ordenar do meio para as pontas das listas.
   d) é o algoritmo que usa o conceito de que o maior número permanece sempre onde está.
   e) é o algoritmo que usa o conceito de que o número maior vai flutuando até o final de uma lista.

3. Sobre o algoritmo de ordenação BubbleSort é INCORRETO afirmar que:
   a) é um algoritmo de ordenação.
   b) é um dos algoritmos de ordenação mais simples de se entender.
   c) é um dos algoritmos de ordenação mais simples de se desenvolver.
   d) utiliza apenas uma estrutura de repetição para o seu desenvolvimento.
   e) usa o conceito de que o maior número vai flutuando até o final de uma lista.

4. Assinale a alternativa CORRETA sobre o algoritmo de busca que utiliza a ideia de passar por cada posição da lista, comparando com a informação a ser buscada:
   a) Busca sequencial.
   b) Busca binária.
   c) Busca com árvores binárias.
   d) Busca com árvores balanceadas.
   e) Busca com árvores não balanceadas.

5. Sobre o algoritmo de Busca Sequencial é CORRETO afirmar que:
   a) é um dos algoritmos de busca mais difíceis de se desenvolver.
   b) é um dos algoritmos de busca mais difíceis de se entender.
   c) é um dos algoritmos de busca mais complexos de se desenvolver.
   d) é um dos algoritmos de busca mais complexos de se entender.
   e) é um dos algoritmos de busca mais simples de se entender e de desenvolver.

6. Sobre o algoritmo de busca sequencial é INCORRETO afirmar que:
   a) é um dos algoritmos de busca mais simples de se entender.
   b) é um dos algoritmos de busca mais simples de se desenvolver.

c) o algoritmo vai passado por cada posição da lista, iniciando no início da lista, comparando se a informação da lista é a informação buscada.
d) caso a informação seja encontrada, a busca continua até o final da lista.
e) pode acontecer de percorrer a lista toda e o elemento não seja encontrado na lista.

Gabaritos

1. **A**

Um dos algoritmos de ordenação e um dos mais simples de se entender e de desenvolver o algoritmo é o Bubble Sort. O Bubble Sort é um algoritmo que precisa de duas estruturas de repetição encadeada para o seu desenvolvimento e usa o conceito de que o número maior vai flutuando até o final de uma lista ou o número menor até o início da lista.

2. **E**
   O Bubble Sort é um algoritmo que precisa de duas estruturas de repetição encadeada para o seu desenvolvimento e usa o conceito de que o número maior vai flutuando até o final de uma lista ou o número menor até o início da lista.

3. **D**
   Um dos algoritmos de ordenação e um dos mais simples de se entender e de desenvolver o algoritmo é o Bubble Sort. O Bubble Sort é um algoritmo que precisa de duas estruturas de repetição encadeada para o seu desenvolvimento e usa o conceito de que o número maior vai flutuando até o final de uma lista ou o número menor até o início da lista.

4. **A**
   Na busca sequencial, o algoritmo vai passando por cada posição da lista, iniciando no início da lista, comparando se a informação da lista é a informação buscada.

5. **E**
   Um dos algoritmos de busca e um dos mais simples de se entender e de desenvolver o algoritmo é a busca sequencial. Na busca sequencial, o algoritmo vai passando por cada posição da lista, iniciando no início da lista, comparando se a informação da lista é a informação buscada.

6. **D**
   Um dos algoritmos de busca e um dos mais simples de se entender e de desenvolver o algoritmo é a busca sequencial. Na busca sequencial, o algoritmo vai passando por cada posição da lista, iniciando no início da lista, comparando se a informação da lista é a informação buscada. Caso encontre, a busca é parada e a informação é apresentada ou utilizada. Pode acontecer de percorrer a lista toda e o elemento não seja encontrado na lista.