Tecnologia em Análise e Desenvolvimento de Sistemas

Disciplina de Estrutura de Dados - 3 º Semestre

Prof Marques Moreira de Sousa

MANUAL DE USO DOS ALGORITMOS DE BUBBLE SORT

Crislaine C. Sotello de Souza

Campos do Jordão,

2023

1. Introdução

Bubble Sort é um algoritmo de classificação comumente usado em ciência da computação. O Bubble Sort baseia-se na ideia de comparar repetidamente pares de elementos adjacentes e, em seguida, trocar as suas posições se estas estiverem na ordem errada.

O algoritmo recebe esse nome porque os itens borbulham gradualmente até o topo da lista.

2. Conceitos Basicos

O Bubble Sort é um algoritmo de ordenação baseado em comparações. Ele percorre repetidamente a lista, compara elementos adjacentes e os troca se estiverem na ordem errada. Esse processo é repetido até que nenhuma troca seja necessária, indicando que a lista está ordenada.

O bubble sort com frequência é considerado o método de ordenação mais ineficiente, já que ele precisa realizar a troca de itens sem saber qual será sua posição final. Essas trocas “desnecessárias” são muito custosas. Contudo, justamente por realizar passagens pela porção desordenada da lista, o bubble sort consegue fazer o que a maioria dos outros algortimos de ordenação não consegue. Em particular, se durante uma passagem não houver trocas, então sabemos que a lista está ordenada.

Este manual tem por objetivo fornecer instruções claras e o passo a passo para que os usuários possam utilizar o código fonte que implementa o algoritmo de ordenação Bubble Sort. O manual abrange desde os pré-requisitos necessários até a execução de casos de teste específicos, permitindo que os usuários compreendam e experimentem o funcionamento do algoritmo em diferentes cenários.

3. Objetivos:

Os principais objetivos deste manual são:

- **Facilitar o acesso ao código**: Proporcionar aos usuários um guia claro sobre como obter o código-fonte, seja por download ou copiando-o manualmente.
- **Instruções para Compilação e Execução**: Orientar os usuários sobre os passos necessários para compilar o código-fonte usando um compilador C e como executar o programa

resultante.

- **Realização de Testes**: Apresentar casos de teste específicos, como melhor caso, pior caso e caso médio, para permitir que os usuários observem o comportamento do algoritmo em diferentes situações.
- **Entendimento do Bubble Sort**: Proporcionar uma experência prática para que os usuários compreendam como o algoritmo Bubble Sort funciona e como lida com diferentes configurações de entrada.
- **Promover a Experimentação**: Encorajar os usuários a explorar o código e realizar seus próprios testes, promovendo uma compreensão mais profunda do agoritmo e da ordenação de arrays.

O manual visa capacitar os usuários a interagirem eficientemente com o código do Bubble Sort, promovendo aprendizado prático e facilitando a experimentação com diferentes casos de teste.

4. Pré-requisito

Certifique-se de ter um compilador C instalado em seu sistema. Exemplos incluem GCC, Clang ou MSVC. E um ambiente de desenvolvimento C, como Code::Blocks, Visual Studio Code ou qualquer editor de texto de sua preferência.

A ferramenta utilizada para a implementação do sistema foi o Code::Blocks.

5. Acessar o código

Se você nunca utilizou o Code::Blocks e deseja acessar o código-fonte do Bubble Sort, siga os passos abaixo:

Antes de tudo, é necessário instalar o Code::Blocks no seu sistema. Você pode baixar o instalador no site oficial: Code::Blocks Download e siga as instruções de instalação fornecidas pelo instalador.

1. Abrir o Code::Blocks:

   Após a instalação, abra o Code::Blocks no seu sistema.

2. Criar um novo Arquivo:

No menu principal, clique em “File” (Arquivo) e selecione “New” (Novo). Escolha “Empty File” (Arquivo Vazio) e clique em “Go”.

3. Escrever e Salvar o código:

No novo arquivo em branco, escreva o código C do Buble Sort.

Clique em “File” (Arquivo) e escolha “Save As” (Salvar como).

Escolha um diretório e dê um nome ao arquivo, por exemplo, “bubblesort.c”.

Certifique-se de adicionar a extensão “.c” ao nome do arquivo.

4. Compilar e executar o código:

No Code::Blocks, vá até o menu “Build” (compilar) e clique em cima dele.

4.1. Verificação por erros de compilação:

Crtifique-se de verificar a janela “Build Log” (Registro de Compilação) para quaisquer erros durante a compilação. Caso ocorrem erros, eles serão destacados nesta janela.

4.2. Executar:

Após a compilação bem-sucedida, vá até o menu “Build” (Compilar) e escolha “Run” (Executar).

Se tudo estiver correto, a saída do programa será exibida na área de console.

Ao seguir este passo a passo, você terá compilado e executado o código do Bubble Sort. Este método oferece a liberdade de construir o código do zero e experimentar com diferentes partes do algoritmo.

4. Funcionamento do Bubble Sort:

O funcionamento do Bubble Sort pode ser descrito em três etapas principais:

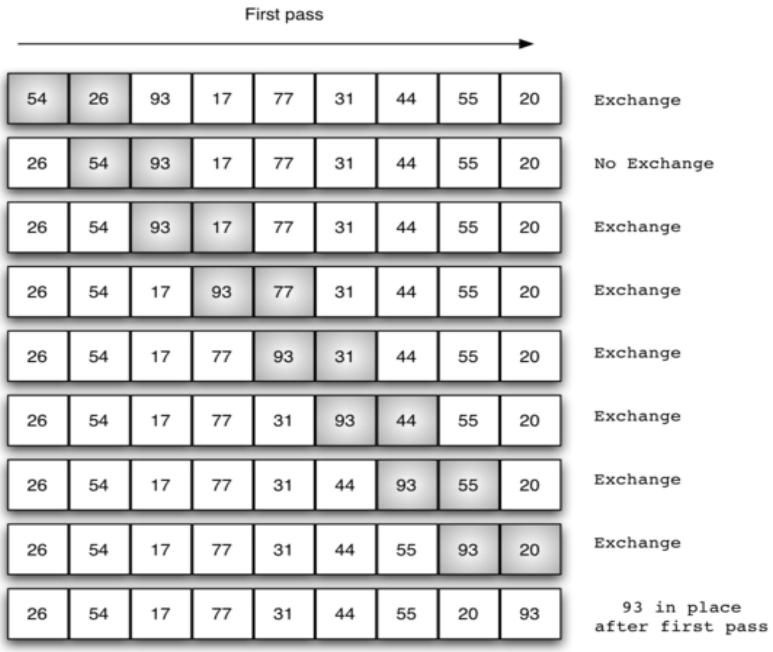
4.1. Comparação: Para cada elemento na lista, compara-se com o próximo. Se estiverem fora de ordem, uma troca é realizada.

4.2. Troca: A troca de elementos é feita quando um elemento é maior ou menor, dependendo da ordem desejada (decrescente ou crescente) que o próximo na lista.

4.3. Iteração: As etapas de comparação e troca são repetidas até que a lista esteja ordenada. Uma iteração completa percorre toda a lista.

Abaixo mostrarei uma figura e um exemplo didático para facilitar o entendimento do funcionamento do Bubble Sort.

A Figura mostra a primeira passagem de um bubble sort. Os itens sombreados são aqueles que estão sendo comparados para verificar se estão fora de ordem. Se existem *n* itens na lista, então existem n −1 pares de itens que precisam ser comparados na primeira passagem. É importante observar que o maior valor na lista esteja em alguma comparação, ele será continuamente empurrado até o fim da passagem.



5. Exemplo de uso didático

Organizando Notas de Alunos em uma Sala de Aula com Bubble Sort

1. **Cenário Inicial:**
   - Você tem uma lista de notas de alunos em uma sala de aula: **[78, 92, 65, 84, 70]**.

2. **Aplicação do Bubble Sort:**
   - Você decide usar o Bubble Sort para organizar as notas dos alunos em ordem crescente.

3. **Passos do Bubble Sort:**

   - Passo 1: Comparação e troca, se necessário.
     - Comparação: 78 < 92 (sem troca)
     - Nova ordem: **[78, 92, 65, 84, 70]**

   - Passo 2: Comparação e troca, se necessário.
     - Comparação: 92 > 65 (troca)
     - Comparação: 92 > 84 (troca)

- Comparação: 92 > 70 (troca)
- Nova ordem: **[78, 65, 70, 84, 92]**

- Passo 3: Comparação e troca, se necessário.
  - Comparação: 78 < 65 (troca)
  - Nova ordem: **[65, 78, 70, 84, 92]**

- Passo 4: Comparação e troca, se necessário.
  - Comparação: 78 < 70 (troca)
  - Nova ordem: **[65, 70, 78, 84, 92]**

- Passo 5: Comparação e troca, se necessário.
  - Comparação: 78 < 84 (sem troca)
  - Nova ordem: **[65, 70, 78, 84, 92]**

4. **Resultado Final:**
   - As notas dos alunos na sala de aula estão agora organizadas em ordem crescente: **[65, 70, 78, 84, 92]**.

Neste exemplo, o Bubble Sort é aplicado para organizar as notas dos alunos em uma sala de aula. Isso ilustra como o algoritmo pode ser utilizado para classificar informações em um contexto educacional. Vale ressaltar que, em cenários práticos, para grandes conjuntos de dados, algoritmos mais eficientes seriam preferíveis.

6. Implementação em C:

A implementação em C do Bubble Sort

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Troca os elementos se estiverem fora de ordem
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
```

7. Cenários de Uso:

O Bubble Sort é mais adequado para pequenas listas ou quando a lista está quase ordenada. Em cenários de grande volume de dados, outros algoritmos de ordenação, como o Merge Sort ou Quick Sort, são geralmente preferíveis.

8. Pseudocódigos

Abaixo está dois exemplos de pseudocódigo do algoritmo de ordenação Bubble Sort:

```
procedimento bubble_sort(array: array[0..n-1])
  para i de 0 até n - 1 faça
    para j de 0 até n - i - 1 faça
      se array[j] > array[j + 1] então
        trocar(array[j], array[j + 1])
      fim se
    fim para
  fim para
fim procedimento
```

Este algoritmo funciona da seguinte forma:

1. O algoritmo começa com uma iteração externa que começa no índice 0 do array e termina no índice n - 1.
2. Dentro da iteração externa, há uma iteração interna que começa no índice 0 e termina no índice n - i - 1.
3. Para cada elemento da iteração interna, o algoritmo compara o elemento atual com o elemento seguinte.
4. Se o elemento atual for maior que o elemento seguinte, o algoritmo troca os dois elementos.
5. O algoritmo repete as iterações externas e internas até que a iteração interna não faça nenhuma troca.

Aqui está outro exemplo de pseudocódigo do algoritmo de ordenação Bubble Sort:

```
procedimento bubble_sort(array: array[0..n-1])
  flag := verdadeiro
  enquanto flag faça
    flag := falso
    para i de 0 até n - 1 faça
      se array[i] > array[i + 1] então
        trocar(array[i], array[i + 1])
        flag := verdadeiro
      fim se
    fim para
  fim enquanto
fim procedimento
```

Este algoritmo funciona da mesma forma que o primeiro exemplo, com a diferença de que ele usa uma variável flag para verificar se houve alguma troca na iteração interna. Se não houver nenhuma troca, o algoritmo termina.

Estes são apenas dois exemplos de pseudocódigo do algoritmo de ordenação Bubble Sort. Existem muitos outros exemplos possíveis.

9. Complexidade do algoritmo

O algoritmo de ordenação bolha, embora seja popular, apresenta **desempenho ruim** se comparado a outros algoritmos de ordenação.

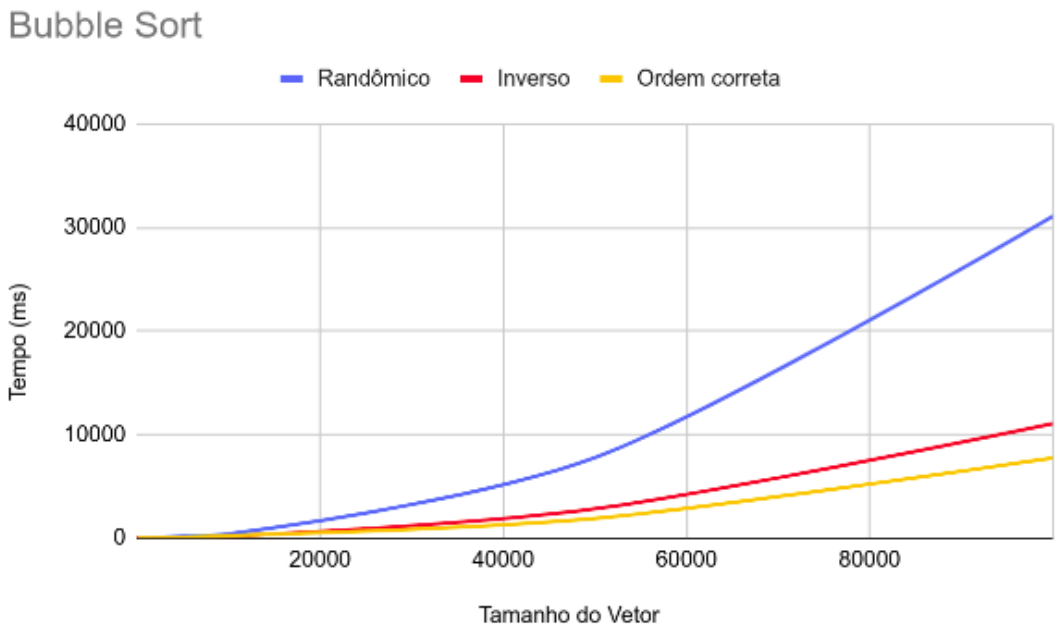
Isso se deve à sua complexidade, que é quadrática, ***O(n²)***, ou seja, o esforço computacional despendido pelo algoritmo varia de **ordem quadrática** de acordo com o tamanho do problema. Isso acontece porque o método bolha utiliza **dois laços de repetição aninhados**.

Uma complexidade quadrática, $O(n^2)$, indica basicamente que conforme o tamanho da lista aumenta, **o tempo** para executar aumenta **quadraticamente**.

Como assim? Suponhamos que para uma lista de 10 elementos, o algoritmo consome 100 segundos de tempo. Então, se aumentarmos a lista para 30 elementos, o tempo consumido passa a ser 900 segundos. Sendo assim, observe que **o tempo** gasto **neste exemplo** é o número de elementos ao quadrado, ou seja, complexidade quadrática.

A complexidade do *Bubble sort* é O(n^2) no pior caso e no caso médio, onde n é o número de elementos na lista.

No entanto, no melhor caso, quando a lista já está ordenada, a complexidade é O(n).



Observação: randômico é um array criado com valores aleatórios, inverso é aquele que a ordem é contrária à esperada e ordem correta é aquele que a ordem está ordenada.

10. Conclusão

Devido ao problema de desempenho, o algoritmo da bolha só é indicado para problemas pequenos, que requeiram uma pequena quantidade de dados. Assim, para problemas maiores e mais complexos, existem outros algoritmos que são mais eficientes, como *quick sort e merge sort*.

11. Referências

https://www.shiksha.com/online-courses/articles/bubble-sort-algorithm-with-code/

https://medium.com/turing-talks/efici%C3%AAncia-de-algoritmos-ordenando-com-bubble-sort-selection-sort-e-random-sort-382e04b2f523

https://www.dio.me/articles/algoritmo-bubble-sort-comentado