Universidade Federal do Piauí - UFPI

Ciência da Computação – CCN

Professor: Raimundo Santos Moura

Atividade 04 – Estrutura de Dados

Aluno:

Davi do Nascimento Santos

Bubble Sort

O bubble sort é um dos algoritmos mais simples para se ordenar um conjunto de elementos, seu funcionamento consiste em ir percorrendo o vetor comparando os elementos adjacentes (dois a dois), se o elemento estiver na posição errada, é feito a troca de posições, garantindo que os maiores elementos sempre fiquem no final do vetor. esse procedimento ocorre **n** vezes, onde n é a quantidade de elementos do vetor. Para instrumento de estudo vamos usar o bubble sort para ordenar um conjunto de caracteres, os caracteres serão: C O M P U T A C A O U F P I H.

T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
2	C	M	O	P	T	A	C	A	O	U	F	P	I	H	U
3	C	M	O	P	A	C	A	O	T	F	P	I	H	U	U
4	C	M	O	A	C	A	O	P	F	P	I	H	T	U	U
5	C	A	C	A	M	O	F	O	I	H	P	P	T	U	U
6	A	C	A	C	M	F	O	I	H	O	P	P	T	U	U
7	A	A	C	C	F	M	I	H	O	O	P	P	T	U	U
8	A	A	C	C	F	I	H	M	O	O	P	P	T	U	U
9	A	A	C	C	F	H	I	M	O	O	P	P	T	U	U

Como podemos observar, o algoritmo vai colocando os maiores elementos no final do vetor à medida em que eles estão fora de posição. Apesar de terem sido apenas 9 passagens para a correta ordenação do vetor, o algoritmo não é performático com uma grande quantidade de elementos, pois no pior caso ele tem uma complexidade de $O(n^2)$ no pior caso.

Selection Sort

O selection sort é um outro algoritmo bem simples para ordenar um conjunto de elementos, seu funcionamento consiste em achar o menor elemento do vetor e colocá-lo na primeira posição do vetor, fazendo uma troca com elemento que está

naquela posição, na segunda iteração, o segundo menor elemento é colocado na segunda posição e assim por diante, o mesmo procedimento é executado outras **n** vezes para ordenar o vetor por completo.

Para instrumento de estudo vamos usar o selection sort para ordenar um conjunto de caracteres, os caracteres serão: C O M P U T A C A O U F P I H.

T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
2	A	O	M	P	U	T	C	C	A	O	U	F	P	I	H
3	A	A	M	P	U	T	C	C	O	O	U	F	P	I	H
4	A	A	C	P	U	T	M	P	O	O	U	F	P	I	H
5	A	A	C	C	U	T	M	P	O	O	U	F	P	I	H
6	A	A	C	C	F	T	M	P	O	O	U	U	P	I	H
7	A	A	C	C	F	H	M	P	O	O	U	U	P	I	T
8	A	A	C	C	F	H	I	P	O	O	U	U	P	M	T
9	A	A	C	C	F	H	I	M	O	O	U	U	P	P	T
10	A	A	C	C	F	H	I	M	O	O	U	U	P	P	T
11	A	A	C	C	F	H	I	M	O	O	U	U	P	P	T
12	A	A	C	C	F	H	I	M	O	O	P	U	U	P	T
13	A	A	C	C	F	H	I	M	O	O	P	P	U	U	T
14	A	A	C	C	F	H	I	M	O	O	P	P	T	U	U
15	A	A	C	C	F	H	I	M	O	O	P	P	T	U	U

A algoritmo vai fazendo trocas entre dois elementos a cada iteração, apesar de ser um algoritmo simples de ordenação, não é muito performático com uma quantidade massiva de dados, sua complexidade é de $O(n^2)$ no pior caso, sendo n a quantidade de elementos do vetor.

Insertion Sort

O insertion sort é um outro algoritmo bem simples para ordenar um conjunto de elementos, seu funcionamento consiste em ir criar uma coleção com apenas um elemento no início e a medida que vai se adicionando mais elementos, é feita uma verificação para colocar os novos elementos na posição correta, as trocas são feitas de um em um fazendo com que esse elemento seja movido a direita, garantindo com que ele fique na posição correta.

Para instrumento de estudo vamos usar o insertion sort para ordenar um conjunto de caracteres, os caracteres serão: C O M P U T A C A O U F P I H.

T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
2	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
3	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
4	C	M	O	P	U	T	A	C	A	O	U	F	P	I	H
5	C	M	O	P	U	T	A	C	A	O	U	F	P	I	H
6	C	M	O	P	U	T	A	C	A	O	U	F	P	I	H
7	C	M	O	P	T	U	A	C	A	O	U	F	P	I	H
8	A	C	M	O	P	T	U	C	A	O	U	F	P	I	H
9	A	C	C	M	O	P	T	U	A	O	U	F	P	I	H
10	A	A	C	C	M	O	P	T	U	O	U	F	P	I	H
11	A	A	C	C	M	O	O	P	T	U	U	F	P	I	H
12	A	A	C	C	M	O	O	P	T	U	U	F	P	I	H
13	A	A	C	C	F	M	O	O	P	T	U	U	P	I	H
14	A	A	C	C	F	M	O	O	P	P	T	U	U	I	H
15	A	A	C	C	F	I	M	O	O	P	P	T	U	U	H
16	A	A	C	C	F	H	I	M	O	O	P	P	T	U	U

O algoritmo à medida que vai avançando vai colocando o próximo elemento na posição correta, a complexidade do algoritmo é de $O(n^2)$ no pior caso e no caso médio.

ShellSort

O algoritmo de ordenação **shellsort** ordena um conjunto de dados ordenando valores distantes entre si por um **gap**, assim a lista de elementos a serem ordenados é particionada em listas menores proporcionando um desempenho médio melhor que o insetion sort. Abaixo está o passo a passo da ordenação da string da atividade utilizando-se o shellsort com gaps 4 e 1. Em negrito estão destacados os elementos que trocaram de posição, toda vez a troca ocorre comparamos o elemento trocado com os elementos que estão à sua esquerda a fim de posicioná-lo o mais próximo possível da sua posição ideal.

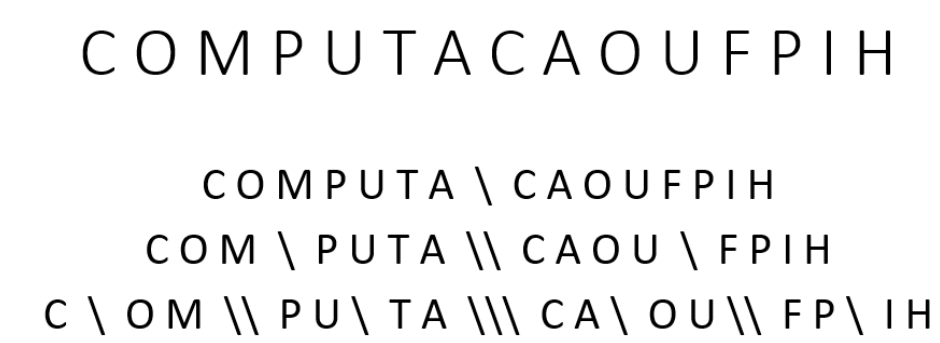
T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
2	C	O	**A**	P	U	T	**M**	C	A	O	U	F	P	I	H
3	A	C	O	**C**	U	T	M	**P**	A	O	U	F	P	I	H
4	A	C	C	O	**A**	T	M	P	**U**	O	U	F	P	I	H
5	A	A	C	C	O	**O**	M	P	U	**T**	U	F	P	I	H
6	A	A	C	C	O	O	M	**F**	U	T	U	**P**	P	I	H
7	A	A	C	C	F	O	O	M	**P**	T	U	P	**U**	I	H
8	A	A	C	C	F	O	O	M	P	**I**	U	P	U	**T**	H
9	A	A	C	C	F	I	O	O	M	P	**H**	P	U	T	**U**
10	A	A	C	C	F	H	I	O	**M**	**O**	P	P	U	T	U
11	A	A	C	C	F	H	I	M	O	O	P	P	**T**	**U**	U

MergeSort

O algoritmo de ordenação MergeSort, é baseado na operação “Merging” a qual combina dois arrays menores ordenados, formando um array só bem ordenado. Esse

algoritmo divide o array em duas partes, até que sobrem apenas arrays de um ou dois elementos, onde a partir disso pode-se fazer a comparação de ordenação. A seguir, iremos implementar este algoritmo com o exemplo proposto pelo professor “C O M P U T A C A O U F P I H”.

O primeiro passo para o começo do algoritmo é separar o array de Strings pela metade, o exemplo oferecido para testes contém 15 posições, sendo assim, o array será dividido em um com 7 posições e outro com 8 posições, após isso, será feita sucessíveis divisões pela metade até que haja somente Strings de uma ou duas posições e comparar entre elas a posição de cada uma, como exemplificação na imagem abaixo:



Em seguida, o algoritmo irá fazer a comparação entre os arrays menores e irá mudar a posição caso seja necessário, com isso, o programa passará a fazer o processo contrário: irá juntar os arrays antes divididos em uma maior só que colocando cada String em sua devida posição (o processo de juntar os arrays em um maior é dado o nome de “merge”), assim como a imagem a seguir:

C O M P U T A C A O U F P I H

C O M P U T A \ C A O U F P I H
C O M \ P U T A \\ C A O U \ F P I H
C \ M O \\ P U \ A T \\\ A C \ O U \\ F P \ I H

C O M P U T A C A O U F P I H

C O M P U T A \ C A O U F P I H
C M O \ A P T U \\ A C O U \ F P I H

Com todos os “merges” devidamente feitos, é devolvido para o usuário o array de Strings completo e com a devida ordenação:

A A C C F I H M O O P P T U U

A C M O P T U \ A C F I H O P U

Quick Sort

No algoritmo de ordenação Quick Sort, escolhe-se um elemento do conjunto de dados a ser ordenado e passamos todos os elementos que são maiores do que ele para a sua direita, esse elemento é chamado de pivô e esse procedimento é repetido

recursivamente para os elementos à esquerda e à direita do pivô até que todo o conjunto de dados esteja ordenado.

T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	C	O	M	P	U	T	A	C	A	O	U	F	P	I	H
2	A	A	C	C	U	T	P	M	O	O	U	F	P	I	H
3	A	A	C	C	U	T	P	M	O	O	U	F	P	I	H
4	A	A	C	C	U	T	P	M	O	O	U	F	P	I	H
5	A	A	C	C	U	T	P	M	O	O	U	F	P	I	H
6	A	A	C	C	H	T	P	M	O	O	U	F	P	I	U
7	A	A	C	C	F	H	P	M	O	O	U	T	P	I	U
8	A	A	C	C	F	H	P	M	O	O	U	T	P	I	U
9	A	A	C	C	F	H	P	M	O	O	I	P	T	U	U
10	A	A	C	C	F	H	I	M	O	O	P	P	T	U	U