Documentação – Trabalho NP2 Estruturas de Dados I

STEFANELLO, Eduardo
NASCIMENTO, Rafael Oliveira
UFFS - 2016

Ordenação Gnome Sort e Quick Sort

1. Gnome Sort

Gnome Sort é um método de ordenação, de complexidade quadrática, que apresenta semelhanças em relação aos métodos Bubble Sort e Insertion Sort. É semelhante ao Insertion Sort pois percorre a estrutura desordenada e a medida que avança vai deixando a parte mais a esquerda ordenada, porém faz uma grande quantidade de trocas, assim como o Bubble Sort.

1.1 Pseudocódigo

```
GnomeSort(Vetor[])
    inteiro proximo = 0, anterior = 0;
    inteiro i = 0
    para(i = 0; i < tamanho - 1; i++)
        se(Vetor[i] > Vetor[i + 1])
            anterior = i
            proximo = i + 1
            Enquanto(Vetor[anterior] > Vetor[proximo])
                Troca(Vetor[anterior],Vetor[proximo])
                se(anterior > 0)
                    anterior--
                se(proximo > 0)
                    proximo--
```

1.2 Implementação:

No presente trabalho, de acordo com as especificações, a ordenação do vetor da estrutura *TpContato* ocorre de maneira semelhante ao exemplo em 1.1, porém fazendo a comparação do campo *nome* com a função da biblioteca *string.h, strcmp.*

No caso da ordenação da lista duplamente encadeada, foi utilizado um índice para percorrer a lista e trabalhar de maneira semelhante à função para vetor. Para isso foram usadas estruturas para contagem de elementos da lista, para voltar à cabeça da lista e para acessar a posição com índice *x.* A comparação entre o campo nome ocorre da mesma forma, com a função *strcmp.*

2. Quicksort

O Quicksort é um método de ordenação, de complexidade logarítmica e recursivo que adota a estratégia de divisão e conquista para ordenar valores em estruturas. A estratégia de dividir para conquistar é feita de forma recursiva e consiste dividir o problema em problemas menores até que estes possam ser resolvidos de forma direta.

O Quicksort consiste em em rearranjar os elementos em duas sublistas, de forma que os valores menores fiquem na lista à esquerda e os maiores na lista à direita, e então subdivide estas listas em listas menores, e assim sucessiva e recursivamente até todos os valores estarem ordenados.

2.1 Passos Básicos

a) Escolher um elemento da lista como pivô.

b) Operação de partição, que arranja a lista de forma que todos os valores menores que o pivô fiquem antes dele e os maiores depois, em duas listas desordenadas.

c) Passo recursivo, que repete os passos anteriores com as sublistas formadas.

2.2 Pseudocódigo

Exemplo de implementação com lista, como usado no trabalho.

```
Divide(Esquerda, Direita)
	TpContato Pivo = Direita->valor
	TpLista I = Esquerda->anterior, J

	para(J = Esquerda; J != Direita; J = J->prox)
		se(strcmp(J->valor.nome, Pivo.nome) <= 0)
			I = (I == NULO) ? Esquerda : I->proximo
			troca(I->valor, J->valor)

	I = (I == NULO)? Esquerda : I->proximo
	troca(I->valor, Direita->valor)

	retorna I

QuickSort(Esquerda, Direita)
	se(Direita != NULO & Esquerda != Direita & Esquerda != Direita->proximo){
		TpLista P = Divide(Esquerda, Direita)
		quickSort(Esquerda, P->anterior)
		quickSort(P->proximo, Direita)
```

2.3 Implementação

Neste trabalho, o Quicksort foi implementado composto de algumas outras funções auxiliares, como a função *Divide*, que é chamada dentro da função QuickSort, além de outras funções como a função de *Troca* e duas funções para identificar a cabeça e a cauda da lista.

Conforme o exemplo em 2.2 e a explicação em 2.1 fica fácil perceber como são utilizados os passos básicos do Quicksort nessa implementação. A escolha do pivô é feita sempre com o elemento mais à esquerda da lista, ou seja, a cabeça.

Para buscar a cabeça e cauda da lista usamos uma função simples que percorre a lista até o último elemento (ou até o primeiro) e retorna seu endereço.

3. Implementações gerais do programa

Este programa foi implementado seguindo as regras presentes na descrição do trabalho, tais como: menu principal, forma de exibição da lista telefônica, cabeçalhos e indicação de tempo.

A Estrutura utilizada para cada contato foi a mesma passada pelo professor e a estrutura de lista implementada usa um campo de valor, dois de endereços (anterior e próximo) e um campo de inteiro que armazena o índice utillizado na função do Gnome Sort.

A função de criação da lista cria cada elemento à direita do que foi criado anteriormente, insere uma valor aleatório e retorna o endereço da cabeça. A função para mostrar a lista recebe o endereço de sua cabeça e percorre a lista exibindo os seus respectivos valores, formatados de acordo com a descrição do trabalho.

Para cada escolha de métodos de ordenação no menu principal, quando necessária a utilização de lista encadeada, uma lista temporária é criada, idêntica a lista criada inicialmente (desordenada), ordena usando o método selecionado e imprime. Após isso a lista temporária é excluída. A lista original é excluída na função de saída do programa.

Em todos os métodos é calculado o tempo de execução de é impresso após a exibição da lista; utiliza a biblioteca *time.h*.

4. Referências:

Algorithm Visualizer <http://algo-visualizer.jasonpark.me/#path=backtracking/n_queens/n_queens> acesso em 09 jun 2016.

Comparison Sorting Visualization <https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html> acesso em 04 jun 2016.

Gnome Sort <https://en.wikipedia.org/wiki/Gnome_sort> acesso em 02 jun 2016.

Quicksort <https://pt.wikipedia.org/wiki/Quicksort> acesso em 02 jun 2016.