Modelo de Documentação para o Trabalho Prático de AEDS2

Prof. Luiz Chaimowicz

1. Introdução

(Colocar as informações gerais sobre o problema a ser tratado, o que vai ser feito no trabalho, os objetivos, etc)

A manipulação de vetores é um dos aspectos fundamentais em Ciência da Computação. Para facilitar essa manipulação em linguagens estruturadas, pode ser necessário a criação de Tipos Abstratos de Dados que encapsulem os detalhes de implementação.

O objetivo desse trabalho é implementar um Tipo Abstrato de Dados (TAD) **Vetor** para a manipulação de vetores. Deverão ser implementadas funções para leitura do vetor, pesquisa, ordenação e impressão. Esse TAD será testado em um programa que ...

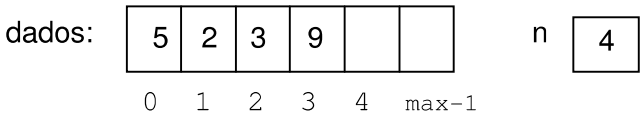
Espera-se com isso praticar os conceitos básicos de programação ...

2. Implementação

(Colocar os detalhes de implementação. A estrutura de dados utilizada, funções e procedimentos criados, funcionamento do programa principal além das decisões de implementação e de informações técnicas como o compilador / ambiente utilizado, como rodar o seu trabalho, etc)

Estrutura de Dados

Para a implementação do trabalho foi criado um Tipo Abstrato de Dado Vetor com a seguinte estrutura:



O campo *dados* é basicamente um int[MAX] que armazena os elementos do vetor enquanto o campo *n* guarda o número de elementos do vetor. Como estamos utilizando alocação estática de memória, foi criada uma constante MAX com o tamanho máximo do vetor.

Funções e Procedimentos

O TAD criado possui as seguintes funções:

void LeVetor(Vetor &v): recebe um vetor por referência e lê os elementos do vetor. Essa função chama a função *scanf* repetidamente para primeiramente ler o tamanho do vetor e em seguida os seus dados.

int Pesquisa(int x, Vetor v): essa função recebe como parâmetro o elemento a ser pesquisado, o vetor e retorna a posição na qual o elemento foi encontrado ou -1 caso ele

não tenha sido encontrado. O algoritmo utilizado para isso é a *Pesquisa Sequencial*: a partir da primeira posição os elementos são comparados um a um com o elemento passado como parâmetro. Em caso de sucesso, o loop é interrompido e a posição do elemento retornado. Se o loop chegar ao final sem encontrar o elemento, o valor -1 é retornado.

void Ordena(Vetor &v): faz a ordenação do vetor em ordem crescente utilizando o algoritmo da Seleção. O algoritmo funciona da seguinte forma: a cada iteração *i* do loop externo, o loop interno seleciona o menor elemento do subvetor [*i+1..n*]. Após selecionado, esse elemento é trocado com o elemento que está na posição *i*. Ao final de n-1 iterações o vetor estará ordenado. A última iteração (*i=n-1*) não é necessária pois o último elemento vai estar necessariamente na sua posição correta.

... (Fazer isso para todas as funções dos TADs e para funções auxiliares que forem eventualmente criadas)

Programa Principal

O programa principal cria uma variável do tipo vetor e depois chama as várias funções do tipo abstrato de dados. Nesse caso o programa implementado é bem simples e serve apenas para testar a implementação do TAD.

O primeiro passo é ler do teclado o tamanho do vetor (máximo 10) e os dados. O programa imprime esse vetor e depois solicita que se entre com um elemento para ser pesquisado. Após a pesquisa, o vetor é ordenado e impresso novamente. Toda a entrada de dados é feita a partir do teclado e toda a saída é direcionada para a tela.

Note que em nenhuma hora o programa acessa a estrutura interna do TAD e utiliza apenas as funções oferecidas em sua interface.

Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código está dividido em três arquivos princpais: *tad.cpp* e *tad.h* implementam o Tipo Abstrato de Dados enquanto o arquivo TP.cpp implementa o programa principal.

O valor especificado para a constante MAX foi 10, de forma a permitir diferentes testes com o programa. Para simplificar a execução, eu resolvi implementar a função... (colocar todas as decisões de implementação que não estejam especificadas no enunciado e que sejam relevantes para o entendimento do seu trabalho)

O compilador utilizado foi o Dev-C++ v. 4.9.9.2 no sistema operacional Windows XP. Para executá-lo, basta digitar TP a partir da linha de comando ou utilizar o ambiente do Dev-C++.

3. Análise de Complexidade

(Nessa seção deve ser feita a análise de complexidade de toda as funções do programa e também do programa principal. Essa análise pode ser feita de forma mais detalhada linha por linha, somando-se as complexidades ou de forma mais geral, explicando a complexidade da função como um todo. De qualquer forma, tem que ficar claro como você chegou nos resultados)

A análise de complexidade será feita em função da variável *n* que representa o número de elementos do vetor.

Função LeVetor: a função LeVetor executa alguns comandos O(1) e depois entra em um loop que é executado n vezes. Dentro desse loop, são feitos apenas comandos O(1) (leitura e atribuição). Portanto temos: O(1) + n.O(1) = **O(n).**

Função Pesquisa: a função possui basicamente um loop que faz a pesquisa do elemento no vetor. No melhor caso, o elemento está na primeira posição do vetor e apenas um uma comparação é feita. Portanto O(1). No pior caso, o elemento está na última posição ou não está presente no vetor. Nesse caso, o loop faz *n* iterações até terminar resultando em uma complexidade linear **O(n).**

Função Ordena: como mencionado, essa função utiliza o algoritmo da Seleção. Esse algoritmo possui 2 loops aninhados: o loop externo faz n-2 iterações (i=0, i<n-1, i++). Dentro desse loop há outro loop que procura o menor elemento da posição i+1 até o final do vetor além de alguns comandos de ordem constante. Portanto a ordem de complexidade vai ser determinada pelo número de iterações do loop interno que vai ser: $n-1 + n-2 + ... + 1 = \mathbf{O(n^2)}$. Mais detalhes sobre a análise de complexidade do algoritmo da seleção pode ser encontrado em [1].

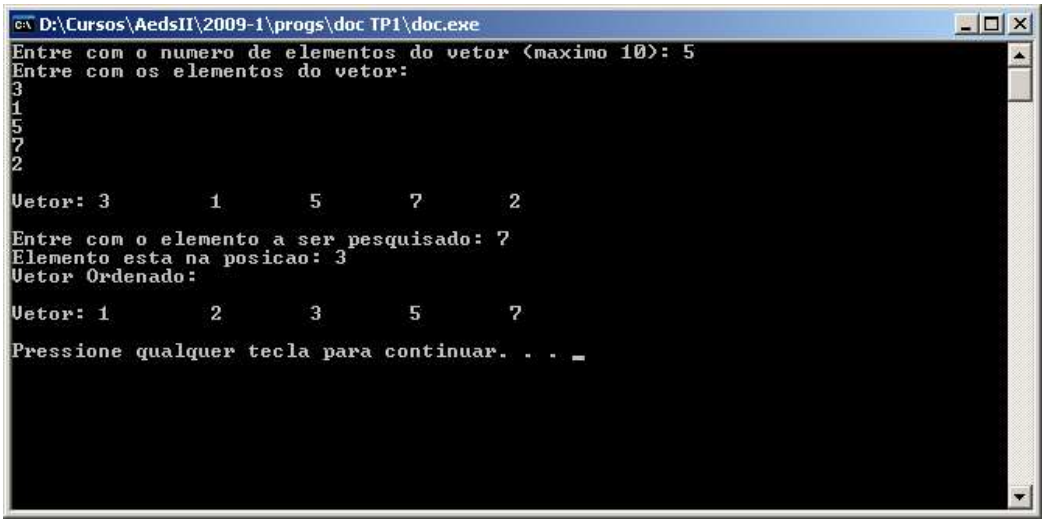
Função Imprime: da mesma forma que a função LeVetor, essa função possui apenas um loop que executa n vezes. Em cada iteração são executados apenas comandos de impressão, que tem ordem de complexidade constante. Logo **O(n)**.

Programa Principal: o programa principal apenas chama uma vez cada uma das funções descritas acima e faz alguns comandos constantes. Dessa forma, a sua complexidade é regida pela função de maior custo computacional: $O(n) + O(n) + O(n^2) + O(n) = \mathbf{O(n^2)}$

4. Testes

(Descrever os testes realizados, mostrando a saída do programa além de eventausi análises que sejam solicitadas no enunciado)

Vários testes foram realizados com o programa de forma a verificar o seu funcionamento. Os testes foram realizados em um Pentium 4, com 1 Gb de memória. A figura abaixo mostra a saída de uma execução típica.



5. Conclusão

(comentários gerais sobre o trabalho e resultados encontrados e as principais dificuldades encontradas em sua implementação).

A implementação do trabalho transcorreu sem maiores problemas e os resultados ficaram dentro do esperado...

A principal dificuldade encontrada foi ...

Referências

[1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2ª Edição, Editora Thomson, 2004.

Anexos

Listagem dos programas:

- tad.h
- tad.cpp
- TP.cpp