João Lucas Ribeiro – 2019005856 Unifei, Itajubá

Resolução da lista 1 de exercícios de COM-112:

Questão 1

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    /*Primeiro, é preciso criar um vetor*/

    //definindo o seu tamanho
    int n = 10;

    //criando o vetor cheio de tamanho 10
    int v[10] = {32, 45, 12, 37, 1, 0, 4, 7, 10, 9};

    /*aqui, o vetor já está preenchido, basta pedir ao usuário para
    digitar um valor chave para ser procurado no vetor            */
    int chave;
    printf("Digite um numero de 1 a 45 para ser procurado no vetor: ");
    scanf("%d", &chave);

    //se o numero nao estiver nas exigencias, o usuario sera avisado
    if(chave > 45 || chave < 0){
        printf("\nNumero invalido... poxa... ");
    }
    /*procurando o valor chave no vetor. Para isso, usar um laço que
    o percorre até encontrar um valor igual (ou não) ao valor chave*/
    int cont = 1;
    printf("Aviso! se nao aparecer nenhuma informacao abaixo, infelizment
e seu numero nao foi encontrado...\n\n");
    for(int i=0; i<n; i++){

        if(chave == v[i]){ //se o valor chave for igual a um valor do vet
or
            printf("Muita sorte! Valor encontrado!\nEle foi comparado %d
vezes no vetor.", cont);
        }
        //se não o contador soma mais 1 e o número é comparado com o próx
imo
        cont ++;
    }

    return 0;
}
```

Questão 2

Analise o algoritmo desenvolvido e identifique:

(a) Melhor caso. Qual o tempo de execução do algoritmo?

Como o tempo em algoritmos depende da velocidade do computador, do código e da complexidade do mesmo, sua contagem de tempo em segundos não pode ser especificada, pois varia de caso para caso. Então realiza-se a contagem de instruções. Para tal, considera-se primeiro que o valor n=10 que expressa o tamanho do vetor é o tamanho que considerei mais razoável para aumentar a dinâmica com o usuário (nem muito fácil de se encontrar um valor correspondente no vetor, e nem muito difícil). Porém, como é para avaliar o melhor caso, vamos considerar o menor vetor possível, com n=1. Pela contagem, ao analisar o meu algoritmo, o melhor caso seria quando o usuário digitasse o número chave exatamente igual ao primeiro valor presente no vetor, usando assim somente uma comparação. Nesse cenário, a função do número de instruções a serem realizadas seria T(n) = 12 instruções. Esse seria o valor mínimo de instruções previsto, demandando assim o menor tempo na execução do algoritmo.

(b) Pior caso. Qual o tempo de execução do algoritmo?

Seguindo a lógica de resolução do item anterior, o pior caso seria se o valor digitado pelo usuário fosse um número que não estivesse nas exigências do enunciado (No caso, se não fosse um número entre 0 e 45) ou não tivesse nenhum correspondente dentro do vetor. Dessa forma, o valor chave seria comparado n vezes até chegar ao fim do laço “for”, e ainda não seria encontrado nenhum valor correspondente dentro do vetor. Considerando isto, também deve-se evidenciar o fato de o valor n do tamanho do vetor ser decidido pelo programador (no caso, eu escolhi o tamanho n=10, mas poderia ter escolhido um maior, e no pior dos casos esse número poderia ser grande). Sob estas condições, realizou-se a contagem das instruções para este cenário, encontrando-se a função T(n) = 10 + 4n instruções. Se n fosse aumentado exponencialmente, teríamos um total de T(n) = n instruções com n tendendo ao infinito, ou O(n), que demandaria o maior tempo na execução do algoritmo.

Questão 3

Faça a análise contando as instruções fundamentais do algoritmo e descreva como deve ser o vetor de entrada para cada caso.

No melhor caso, contando as instruções fundamentais do algoritmo, o número de instruções seria dado por T(n) = 12 instruções, considerando o menor vetor de entrada possível (com n = 1). No pior caso, contando as instruções fundamentais do algoritmo, o número de instruções seria dado por T(n) = 10 + 4n instruções, considerando o maior vetor de entrada possível (com n assumindo o maior valor possível tal qual fosse suportado pela memória do computador).

Questão 4

Faça um resumo da Seção 2.2 - Análise de Algoritmos do livro Algoritmos: Teoria e Prática - 2a ed. (CORMEN, 2002), referenciado nos slides.

Análise de algoritmos

Analisar um algoritmo significa prever os recursos de que um algoritmo necessitará. Frequentemente deseja-se medir o tempo de computação do algoritmo, encontrando-se o mais eficiente.

Visto que o comportamento de um algoritmo pode ser diferente para cada entrada possível, precisa-se de um meio para resumir esse comportamento em fórmulas simples, de fácil compreensão. Um objetivo imediato é se expressar de uma forma simples de se escrever e manipular, mostrando o que é importante num algoritmo e descartando detalhes desnecessários.

Análise da ordenação por inserção

O tempo de inserção e ordenação de números depende da entrada, quanto maior a entrada, maior o tempo de execução do algoritmo.

Tamanho de entrada: Depende do problema a ser estudado. Em problemas como ordenação ou cálculo de transformações discretas, diz-se 'número de itens da entrada'. Em outros como multiplicação de dois inteiros, se diz o 'número total de bits da entrada', em notação binária comum.

Tempo de execução: é o número de operações ou "etapas" que precisam ser executadas. Estas etapas são dividas entre as linhas do pseudocódigo, demorando tempos diferentes para serem executadas. O tempo de execução do algoritmo é medido pela soma dos tempos de execução de cada instrução. Para calcular T(n), somamos a quantidade de instruções de cada linha, considerando que laços e loops como "for" e "while" são conjuntos e/ou repetições de instruções simples. Esse tempo pode ser expresso em funções lineares como $an + b$, funções quadráticas como $an^2 + bn + c$, tudo dependendo da entrada e da complexidade do algoritmo.

Análise do pior caso e do caso médio

O pior caso é aquele em que o código do algoritmo demora mais tempo para ser executado para uma entrada de tamanho n. Há 3 razões para esta análise:

1. Conhecer o limite superior do pior caso nos permite saber o tempo mais longo possível e ter a esperança de que ele não pode ser muito pior.
2. Para alguns algoritmos, como, por exemplo, em uma busca em um banco de dados, o pior caso pode ocorrer com frequência.
3. Muitas vezes, o caso médio é quase tão ruim quanto o pior caso.

Ordem de crescimento

A **taxa de crescimento** ou a **ordem de crescimento** do tempo de execução é o que realmente importa. Assim, considera-se apenas o termo inicial de uma fórmula T(n), como, por exemplo, n^2 ao invés de $n^2 + 2n + 5$. Isto porque os termos de mais baixa ordem se tornam insignificantes a medida em que n cresce a valores absurdamente grandes, assim como as constantes, e ambos são descartados. Portanto, escrevemos que a ordenação por inserção, por exemplo, é de $O(n^2)$ (theta de n ao quadrado).

Em geral, um algoritmo mais eficiente que outro é aquele que possui uma ordem de crescimento mais baixa no seu pior caso em relação ao tempo de execução. Um algoritmo $O(n^2)$, por exemplo, é mais rápido do que outro $O(n^3)$, sendo também mais eficiente.

Questão 5

Escolha dois exercícios da Seção 2.2 (exercício anterior), escreva o enunciado e resolva-os.

1. Expresse a função $n^3/1000 - 100n^2 - 100n + 3$ em termos da notação 0.

Será n^3, pois podemos ignorar os outros fatores da expressão, já que com n crescendo exponencialmente, os mesmos terão um valor insignificante perto de n^3, e ignoraremos também (1/1000), pelos mesmos princípios apresentados.

2. Como podemos modificar praticamente qualquer algoritmo para ter um bom tempo de execução no melhor caso?

A melhor forma de otimizar um algoritmo, ou seja, melhorar seu tempo de execução no melhor caso, é procurar usar o menor número possível de variáveis, funções e quaisquer outras ações no código que possam demandar um maior processamento, uso da memória e afins. Resumindo, a solução seria “enxutar” o código o máximo possível.