Estruturas de Dados II

Eficiência: Explorando e Analisando Soluções

Prof. Bruno Azevedo

Instituto Federal de São Paulo

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Um Problema Simples

- Considere que temos o seguinte problema: queremos contar o número de pares em um vetor que somam a um valor específico.
- Modelando computacionalmente, nossa entrada é um vetor de n elementos e nossa saída é um número inteiro representando o número de pares que somam a um valor v.
- Por exemplo, suponha que o vetor seja $[1, 4, 3, 2]$ e o valor para a soma seja 5.
- Vamos encontrar todos os pares de elementos que somam 5:
    - O par $(1, 4)$ soma 5.
    - O par $(3, 2)$ soma 5.
- Então, neste exemplo, há 2 pares de elementos que somam 5.
- Vamos criar um algoritmo que resolve esse problema.

Uma Primeira Solução

```
#include<iostream>
int conta_pares_sol_um(int *vetor, int n, int soma) {
    int pares = 0;
    for(int i = 0; i < n - 1; i++) {
        for(int j = i + 1; j < n; j++) {
            if(vetor[i] + vetor[j] == soma)
                pares++;
        }
    }
    return pares;
}
int main() {
    int vetor[] = {1, 5, 7, 4, 2, 5};
    int n = sizeof(vetor) / sizeof(vetor[0]);
    int soma = 6;
    int pares = conta_pares_sol_um(vetor, n, soma);
    std::cout << "Número de pares:" << pares;
    return 0;
}
```

Uma Primeira Solução

```
#include<iostream>
int conta_pares_sol_um(int *vetor, int n, int soma) {
    int pares = 0;
    for(int i = 0; i < n - 1; i++) {
        for(int j = i + 1; j < n; j++) {
            if(vetor[i] + vetor[j] == soma)
                pares++;
        }
    }
    return pares;
}
int main() {
    int vetor[] = {1, 5, 7, 4, 2, 5};
    int n = sizeof(vetor) / sizeof(vetor[0]);
    int soma = 6;
    int pares = conta_pares_sol_um(vetor, n, soma);
    std::cout << "Número de pares:" << pares;
    return 0;
}
```

- Determine a sua complexidade de tempo no pior caso utilizando a notação-O.

Uma Primeira Solução

- Deve estar claro que a parte interessante é o segundo laço.
- O primeiro executará $n - 1$ vezes, ou seja, basicamente o tamanho da entrada. Notem como é de zero até $n - 1$, não incluso.

```
for(int i = 0; i < n - 1; i++) {
    for(int j = i + 1; j < n; j++) {
        if(vetor[i] + vetor[j] == soma)
            pares++;
    }
}
```

- Mas e o segundo laço?

Uma Primeira Solução

- O segundo laço é mais interessante.

```
for(int i = 0; i < n - 1; i++) {
    for(int j = i + 1; j < n; j++) {
        if(vetor[i] + vetor[j] == soma)
            pares++;
    }
}
```

- Na primeira iteração do primeiro laço, o segundo laço executa $n-1$ vezes; de 1 até $n-1$.
- Na segunda iteração do primeiro laço, o segundo laço executa $n-2$ vezes.
- E isso se repete até executar apenas uma vez, quando somamos o penúltimo com o último elemento.

Uma Primeira Solução

- Para quem ainda não entendeu, vamos pensar em um exemplo simples com $n = 5$.
  - O algoritmo soma a posição 0 com as posições 1, 2, 3 e 4.
  - O algoritmo soma a posição 1 com as posições 2, 3 e 4.
  - O algoritmo soma a posição 2 com as posições 3 e 4.
  - O algoritmo soma a posição 3 com a posição 4.
- Temos $n - 1$ execuções do primeiro laço, mas para cada iteração o tamanho do segundo laço diminui.

Uma Primeira Solução

- Portanto, o número de comparações executado por esses dois laços é $(n-1)+(n-2)+\cdots+(n-(n-2))+(n-(n-1))$.
- Podemos simplificar para $(n-1)+(n-2)+\cdots+2+1$.
- Isso é claramente uma P.A. A fórmula para a soma de uma P.A. é dada por:

$$S_n = \frac{n \times (a_1 + a_n)}{2}$$

.

- Onde n é o número de termos e a_1 e a_n são o primeiro e último termos, respectivamente.

Uma Primeira Solução

- Portanto, temos

$$S_n = \frac{(n-1) \times (1 + n - 1)}{2} = \frac{n^2 - n}{2}$$

comparações ocorrendo em nosso algoritmo, no pior caso.

- Para aplicarmos a notação-O, iremos:
  - Ignorar constantes multiplicativas e aditivas.
  - Focar no termo de ordem superior.

Uma Primeira Solução

- Destacando as constantes multiplicativas:

$$\frac{n^2 - n}{2} = \frac{1}{2} \times n^2 - \frac{1}{2} \times n$$

.

- Vamos ignorar o termo de menor ordem: $-\frac{1}{2} \times n$.
- Vamos ignorar a constante multiplicativa: $\frac{1}{2}$.
- Portanto, a complexidade computacional de nosso algoritmo é $O(n^2)$.

Uma Segunda Solução

```
#include<iostream>
int conta_pares_sol_dois(int *vetor, int n, int soma) {
    int ocorrencias[1000] = {0}; //Assumindo que os valores estão entre 0 e 999
    int pares = 0;
    for(int i = 0; i < n; i++) {
        int complemento = soma - vetor[i];
        if(complemento >= 0 && ocorrencias[complemento])
            pares += ocorrencias[complemento];
        ocorrencias[vetor[i]]++;
    }
    return pares;
}
```

- Como ele funciona?
- Determine a sua complexidade de tempo no pior caso utilizando a notação-O.

Uma Segunda Solução

- Uma solução mais flexível, sem a limitação de valores.

```
#include <iostream>
#include <unordered_map>
#include <list>
int conta_pares_sol_dois(int *vetor, int n, int soma) {
    std::unordered_map<int, int> ocorrencias;
    int pares = 0;
    for(int i = 0; i < n; i++) {
        int complemento = soma - vetor[i];
        if(ocorrencias.find(complemento) != ocorrencias.end())
            pares += ocorrencias[complemento];
        ocorrencias[vetor[i]]++;
    }
    return pares;
}
```

Outro Problema: Soma de Subvetores de Tamanho Fixo

- Encontrar a soma de subvetores de tamanho fixo em um vetor.
- Consiste em calcular a soma de todos os subvetores de um determinado tamanho fixo k em um vetor dado.
- Um subvetor é uma sequência contígua de elementos dentro de um vetor.
- Por exemplo, vamos considerar o vetor $V = [1, 2, 3, 4, 5]$ e $k = 3$.

```
Subvetor 1: [1, 2, 3]
Soma = 1 + 2 + 3 = 6

Subvetor 2: [2, 3, 4]
Soma = 2 + 3 + 4 = 9

Subvetor 3: [3, 4, 5]
Soma = 3 + 4 + 5 = 12
```

- Portanto, para o vetor V, com $k = 3$, o resultado é 6, 9 e 12..

Outro Problema: Soma de Subvetores de Tamanho Fixo

- Criem um algoritmo que resolva este problema.
- Determine a sua complexidade de tempo no pior caso utilizando a notação-O.
- Caso crie um algoritmo de complexidade quadrática ou similar, ponderem: é possível criar um algoritmo mais eficiente? Caso não seja possível, argumente a razão dessa impossibilidade. Caso seja possível, escreva o algoritmo e determine a sua complexidade utilizando a notação-O.