Projeto e analíse de algoritmos

Amanda Goulart

Fevereiro 2022

1 Analise o custo computacional do algoritmo a seguir, que calcula o valor de um polinômio de grau n, da forma $P(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, onde os coeficientes são números de ponto flutuante armazenados no vetor a[0...n], e o valor de n é maior que zero. Todos os coeficientes podem assumir qualquer valor, exceto o coeficiente a que é diferente de zero.

```
1  Algoritmo 1:
2  soma = a[0]
3  Repita para i = 1 at n
4      Se a[i] =\ 0.0 ento
5          potncia  = x
6          Repita para j = 2 at i
7              potncia  = potncia  * x
8          Fim repita
9          soma = soma + a[i] * potencia
10     Fim se
11 Fim repita
12 Imprima(soma)
```

1.1 Faça a análise do algoritmo acima determinando a equação que descreve a quantidade de multiplicações realizadas no pior caso e melhor caso.

1.1.1 Melhor caso

Primeiramente, para o melhor caso temos que quando todos os coeficientes, exceto a_n, são iguais a zero. Diante disso ele executará o laço interno da linha 6 somente uma vez.

Como as operações de declarar, soma tem valores tem custos irrisórios, irei me atentar ao laço de repetição, para determinar a equação teremos que ter uma somatoria a principio devido ao laço de repetição, para o primeiro laço, na linha 3 neste caso teremos custo:

$$n$$

Já o segundo laço, da linha 6, teremos o custo de:

$$\sum_{j=2}^{n} 1$$

Sendo assim para o melhor caso temos que :

$$T(n) = n + \sum_{j=2}^{n} 1$$

1.1.2 Pior caso

No pior caso temos que todos os coeficientes são diferentes de 0. Então além de analizar o laço de repetição interna, que já temos, iremos que analisar o laço externo, então para ele temos o custo de:

$$\sum_{i=1}^{n} 1$$

Agora para T(n):

$$T(n) = \sum_{i=1}^{n}(1 + \sum_{j=2}^{n} 1)$$

1.2 Mostre a qual a classe de complexidade a equação do pior caso do item anterior pertence utilizando a definição da notação Θ.

Agora sabendo a equação do pior caso, vamos expandir T(n):

$$T(n) = \sum_{i=1}^{n}(1 + \sum_{j=2}^{n} 1)$$

$$T(n) = \sum_{i=1}^{n}(1 + i - 1)$$

$$\sum_{i=1}^{n} i$$

Sabendo que a Notação O(g(n)) = f(n) : existe constantes positivas C1 e N_0, tais que $0 \leq f(n) \leq c * g(n), \forall n \geq n_0$

Dado f(n) = (n*n+n)/2 e g(n) = n*n

Adotando n_0 = 1 e c = 2:

$$0 \leq (n * n + n)/2 \leq 2 * n * n$$

$$0 \leq n * n + n \leq 4 * n * n$$

$$0 \leq n \leq 3 * n * n$$

$$n \geq n_0 = 1$$

Provando assim que $T(n) = O(n^2)$

2 Considere o problema de retornar n centavos de troco com o número mínimo de moedas. As moedas são de valores que pertencem a um dado conjunto de inteiros D que inclui necessariamente o valor de 1 centavo. A quantidade de moeda de cada valor a ser utilizada no troco é ilimitada.

2.1 Demonstre a propriedade de subestrutura ótima do problema.

Vamos tentar adotando a tatica de pegar a moeda de maior valor, desde que ela não seja maior que o troco.

Suponhamos que temos as seguinte moedas 25, 10, 5 e 1 centavos e queremos fazer um troco de 42 centavos, então podemos fazer:

1 - $42 - 25 = 17$
2 - $17 - 10 = 7$
3 - $7 - 5 = 2$
4 - $2 - 1 = 1$
5 - $1 - 1 = 0$
O total de 5 moedas é a solução ótima!

Logo qualquer quantia D , subtraímos de maneira gulosa o maior valor de denominação que não seja maior que D .

2.2 Esse problema pode ser resolvido corretamente por meio de um algoritmo guloso? Caso negativo, mostre um contra-exemplo. Caso positivo, demonstre a sua propriedade gulosa.

Suponhamos o conjunto 21, 20, 1, o jeito ótimo de se pagar 40 centavos é com 2 moedas de 20. O algoritmo guloso escolheria 1 de 21, e seria forçado a adicionar mais 19 de 1 depois. Sendo menos eficiente, sendo o ideal programação dinâmica.

```c
int troco(int vetor[], int tam, int valor) {
    int T =0, t[MAX], i;
    for(i=0; i<tam; i++) {
        t[i] = valor/vetor[i];
        valor = valor - t[i]* vetor[i];
        T = T + t[i];
    }
    return T;
}
```

2.3 Projete um algoritmo por programação dinâmica que resolve o problema de forma ótima.

Com a subestrutura ótima consequimos projetar o seguinte algoritmo:

```python
def troco(N, Coins):

    best = [0] * (N + 1);
    best[0] = 1;
```

```
    for i in range(len(Coins)):
        for j in range(len(best)):
            if (Coins[i] <= j):
                best[j] += best[(int)(j - Coins[i])];


    return best[N];
```

3 Responda, para cada afirmação abaixo, se a afirmação é: i) verdadeira, ii) falsa, iii) verdadeira se P $\neq$ NP, ou iv) falsa se P $\neq$ NP. Justifique as suas respostas.

3.1 Existem problemas em P que estão em NP-completo.

3.1.1 Falsa se $P \neq NP$

$P \in NP$, ou seja, todo problema polinomial de decisão é polinomialmente verificável. Tanto que ainda não encontraram um problema NP que não seja P

3.2 Se a redução de um problema A em um problema B é em tempo polinomial e o problema A é de complexidade exponencial, então B também tem complexidade exponencial.

3.2.1 Verdadeira

Isso se deve polo conceito de intratabilidade, onde A para B indica que B é no mínimo tão dificil que A

3.3 Se o problema A pode ser polinomialmente reduzido a B e A pertence a P, então B pertence a P.

3.3.1 Falsa

Devido ao contra-exemplo que B pode pertencer a NP.

3.4 Seja A um problema NP, então A pode ser polinomialmente reduzido a um problema NP-Completo B.

3.4.1 Verdadeira se $P \neq NP$

Por definição Se existe algum problema NP-Completo que pode ser decidido em tempo polinomial então para todo A $\in NP, noentantono NP - Completo B.$