# Trabalho 3

***Documentação | Trabalho 3 | INF 610 - Estruturas de Dados e Algoritmos*** > Implementação completa disponível no Github.

## Parte Um - Maximize the value of an expression

> Dada uma matriz A, maximize o valor da expressão $(A[s] - A[r] + A[q] - A[p])$, onde $p$, $q$, $r$ e $s$ são índices da matriz e $s > r > q > p$

### Brute Force

Pela aboradagem de brute force é percorrido todas as soluçoes possiveis. No codigo abaixo temos um algoritimo para o problema se utilizando de brute force, sendo que sua complexidade é $O(n^3)$

```
func BruteForceMaximizeExpression(A []int) int {
max := -1000000000
  for p := 0; p < len(A); p++ {
    for q := p + 1; q < len(A); q++ {
      for r := q + 1; r < len(A); r++ {
        for s := r + 1; s < len(A); s++ {
          if A[s]-A[r]+A[q]-A[p] > max {
            max = A[s] - A[r] + A[q] - A[p]
          }
        }
      }
    }
  }
  return max
}
```

### Dynamic Progaming

Abaixo temos o algoritmo para o problema utilizando a abordagem de programação dinâmica. Para esta abordagem, é criada 4 tabelas para serem armazenados os resultados de cada um dos sub resultados da equação:

- `first[]` armazena o valor de $A[s]$;
- `second[]` armazena o valor de $A[s] - A[r]$;
- `third[]` armazena o valor de $A[s] - A[r] + A[q]$;
- `fourth[]` armazena o valor de $A[s] - A[r] + A[q] - A[p]$.

Assim, o resultado final é armazenado no vetor fourth, podemos justificar que esta é uma abordagem de programação dinâmica, pois estamos criando diferentes tabelas, nas quais estamos armazenando os sub resultados e os usando posteriormente para calcular os próximos sub resultados até chegar ao resultado final. A ordem de complexidade deste algoritmo é $O(n)$.

```
func DynamicProgrammingMaximizeExpression(A []int) int {
    n := len(A)

    if len(A) < 4 {
        panic("Array length must be bigger than 4")
    }

    first := make([]int, n+1)
    second := make([]int, n)
    third := make([]int, n-1)
    fourth := make([]int, n-2)
    for i := 0; i < n-3; i++ {
        first[i] = -1000000000
        second[i] = -1000000000
        third[i] = -1000000000
        fourth[i] = -1000000000
    }
    first[n-2] = -1000000000
```

```
    second[n-2] = first[n-2]
    third[n-2] = second[n-2]

    first[n-1] = -1000000000
    second[n-1] = first[n-1]

    first[n] = second[n-1]

    for i := n - 1; i >= 0; i-- {
        first[i] = int(math.Max(float64(first[i+1]), float64(A[i])))
    }
    for i := n - 2; i >= 0; i-- {
        second[i] = int(math.Max(float64(second[i+1]), float64(first[i+1]-A[i])))
    }
    for i := n - 3; i >= 0; i-- {
        third[i] = int(math.Max(float64(third[i+1]), float64(second[i+1]+A[i])))
    }
    for i := n - 4; i >= 0; i-- {
        fourth[i] = int(math.Max(float64(fourth[i+1]), float64(third[i+1]-A[i])))
    }

    return fourth[0]
}
```

**Comparação dos resultados**

## Parte Dois - *italicized text*