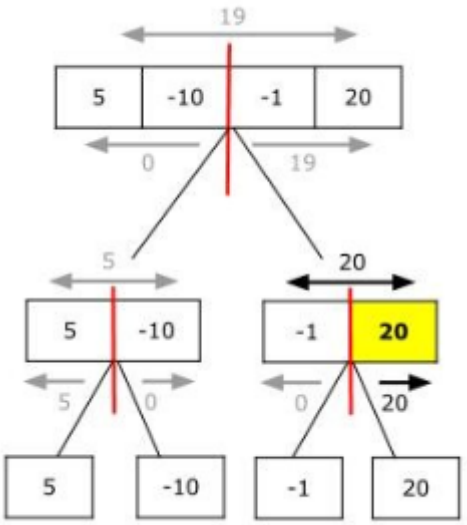
Trabalho 3: Algoritmo para o Cálculo do Maior Subarray Contíguo, com Divisão e Conquista

Nomes: Henrique Fugie de Macedo e Luiz Otávio Silva Cunha
Matrículas: 0056151 e 0056153

Algoritmo para o Cálculo do Maior Subarray Contíguo

O problema do Maior Subarray Contíguo, também conhecido como "Maximum Subarray Sum" ou "Máxima Soma de Subsequência Contígua", consiste em encontrar a soma máxima de um subarray (sequência de elementos consecutivos) em um array dado. Este é um problema clássico de programação dinâmica e pode ser resolvido de forma eficiente utilizando a técnica de Kadane. O algoritmo de Kadane mantém a soma máxima atual e atualiza-a conforme percorre o array, ignorando subarrays cuja soma se torna negativa.



O algoritmo de Kadane aproveita o fato de que, se a soma de um subarray se torna negativa, não é benéfico incluir esse subarray em um subarray maior, pois isso diminuiria a soma total. Portanto, o algoritmo "abandona" subarrays negativos, começando a calcular a soma a partir do próximo elemento do array.

O tempo de execução do algoritmo de Kadane é linear, O(n), tornando-o uma escolha eficiente para resolver o problema do Maior Subarray Contíguo. É uma abordagem simples e elegante que não requer armazenamento adicional de subarrays intermediários, economizando espaço e melhorando a eficiência.

Divisão e Conquista

A abordagem de divisão e conquista é uma técnica algorítmica fundamental que consiste em resolver um problema dividindo-o em subproblemas menores, resolvendo esses subproblemas de forma recursiva e, em seguida, combinando suas soluções para obter a solução do problema original. Vamos discutir como essa abordagem funciona, seus pontos positivos e alguns exemplos de problemas em que é comumente aplicada.

O funcionamento dela ocorre da seguinte forma: ele começa dividindo o problema original em subproblemas menores e mais simples, em seguida cada subproblema é resolvido de forma recursiva e no final os resultados são combinados para formar a solução do problema original.

Explicação do código

O código é divido em 3 funções, função cruzada, função maiorSubArray e função verifica, cada uma delas tem um papel diferente.

- Função maiorSubArray: Recebe como entrada o array, a posição de início da verificação, e a posição de fim para a verificação, depois percorre do início até o fim para verificar o maior sub vetor.

```python
def maiorSubArray(a, inicio, fim):
    max_atual = max_global = -999999
    aux = []
    i1 = i2 = f1 = f2 = inicio
    ig1 = ig2 = fg1 = fg2 = inicio

    # Encontra a maior soma do subvetor
    for i in range(inicio, fim):

        if a[i] > max_atual + a[i]:
            max_atual =  a[i]
            i1 = i
            f1 = i
        else:
            max_atual = max_atual + a[i]
            f1 = i

        if max_atual > max_global:
            max_global = max_atual
            ig1 = i1
            fg1 = f1

    aux = a[ig1:fg1+1]  # Captura o subvetor com a maior soma
    return aux, ig1, fg1
```

- Função Cruzada: Muito semelhante a função de cima, porém ela verifica do final até o fim do array sendo assim, verificando ele todo.

```python
def cruzada(vetor, inicio, meio, fim):
    max_esq = -999999  # Inicializa a maior soma do subvetor à esquerda com um valor muito baixo
    max_atual = 0     # Inicializa a soma atual
    inicio_cruzamento = meio  # Inicializa o índice de início do subvetor cruzado
    fim_cruzamento = meio    # Inicializa o índice de fim do subvetor cruzado

    # Encontra a maior soma no subvetor à esquerda
    for i in range(meio, inicio - 1, -1):
        max_atual = max_atual + vetor[i]
        if max_atual > max_esq:
            max_esq = max_atual
            inicio_cruzamento = i

    max_dir = -999999  # Inicializa a maior soma do subvetor à direita com um valor muito baixo
    max_atual = 0     # Reinicializa a soma atual

    # Encontra a mai (variable) max_atual: Any | Literal[0]
    for i in range(m
        max_atual = max_atual + vetor[i]
        if max_atual > max_dir:
            max_dir = max_atual
            fim_cruzamento = i

    aux = vetor[inicio_cruzamento:fim_cruzamento+1]  # Captura o subvetor com a maior soma cruzada
    return aux
```

- Função verifica: Verifica qual é sub array é maior, da esquerda, da direita, ou do cruzamento de ambas.

```python
def verifica(esq, meio, direita):
    s1 = 0
    s2 = 0
    s3 = 0

    # Calcula a soma dos subvetores esquerdo, cruzado e direito
    for i in range(len(esq)):
        s1 = s1 + esq[i]
    for i in range(len(direita)):
        s2 = s2 + direita[i]
    for i in range(len(meio)):
        s3 = s3 + meio[i]

    # Compara as somas e retorna a maior soma e o subvetor correspondente
    if (s1 > s2) and (s1 > s3):
        return s1, esq
    if (s2 > s1) and (s2 > s3):
        return s2, direita
    if (s3 > s1) and (s3 > s2):
        return s3, meio
    aux = []
    aux.append(s1)
    aux.append(s2)
    aux.append(s3)
    maior = aux.index(max(aux))
    if maior == 0:
        return s1, esq
    elif maior == 1:
        return s2, direita
```