INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: BACKTRACKING E DIVIDIR P/ CONQUISTAR

Prática 13

Parte 0: Preparação

Passo 1: Essa prática uso os mesmo arquivos usados na Prática 12.

Parte 1: Problema Subconjunto com Soma K com Backtracking

Passo 1: Em **subsetsum.cpp**, implemente a função `subsetSumBTRec()`**:**

A solução deve usar ***Backtracking***, implementando a formulação abaixo:

$$SSK(n,k) = \begin{cases} falso, & n < 0 \; ou \; k < 0 \\ verdadeiro, & k = 0 \\ SSK(n-1,k) \; ou \; SSK(n-1,k-a[n-1]) \end{cases}$$

$SSK(n,k)$ responde à pergunta: "É possível somar k usando os primeiros n elementos do array?", sendo que o elemento na posição $n-1$ pode ou não ser usado. Essa solução realiza o *pruning* da busca ao retorna falso no momento que $k < 0$. Apesar de ser mais inteligente que a solução em Força Bruta (Prática 12) pelo uso de *pruning*, essa solução ainda tem complexidade $O(2^N)$.

Nos parâmetros de `subsetSumBTRec(array, n, k, subset)`, `array` é o conjunto total de valores, `subset` é um vetor booleano que diz se um determinado valor de `array` pertence (`true`) ou não (`false`) à solução; n e k seguem a formulação acima. A função `subsetSumBT()` apenas chama `subsetSumBTRec()` com o parâmetros iniciais corretos (isto é, $SSK(N-1,k)$, onde N é tamanho do *array*).

Passo 2: Compile e rode a aplicação.

Faça testes modificando o tamanho do *array* para ver o impacto no tempo.

Passo 3: Adicione um teste para o pior caso em `subsetSumBT()`.

Se a soma desejada for maior que todos os elementos do *array* somados, o código de `subsetSumBTRec()` vai testar todas as possibilidades desnecessariamente antes de falhar. Adicione código que evitar que isso aconteça. Faça novos testes.

Parte 2: Problema da Subsequência com Soma Máxima com Dividir p/ Conquistar

Passo 1: Em **subseqmax.cpp**, implemente `subseqMaxDC_Rec()`:

A função `subseqMaxDC()` é o ponto de entrada para a solução usando Dividir p/ Conquistar. Essa função apenas chama `subseqMaxDC_Rec()` com o parâmetros iniciais adequados. `subseqMaxDC_Rec()` é a função que realmente calcula a sequência de soma máxima de forma recursiva. Essa função usa `subseqMaxMiddle()`, que acha a sequência de soma máxima que passa pelo meio do array (`middle`), onde meio é um ponto entre início (`start`) e final (`finish`). O valor da maior soma é retornado pelas funções, e o intervalo da sequência é salva em `ini` e `end`. Essa solução tem complexidade $O(N \log N)$, como explicado em sala.

Passo 2: Compile e rode a aplicação.

Faça testes modificando o tamanho do *array* para ver o impacto no tempo.