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: DIVIDIR PARA CONQUISTAR, ALGORITMOS GULOSOS

Prática 11

OBS.: Esta prática usa código da prática 10 e novos arquivos.

Parte 1: Praticando Dividir para Conquistar (D&C) e Algoritmos Gulosos (Greedy)

Problema 1: Implemente a solução usando Dividir p/ Conquistar do problema Subseqmax.

Implemente a função `subseqMaxDC_Rec()` no arquivo **subseqmax.cpp**. 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()` (já implementada), a qual 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` (parâmetros passados por referência). Essa solução tem complexidade $O(N \log N)$.

Desafio (Opcional): Tente resolver o problema Subseqmax usando uma estratégia gulosa.

Seja criativo.

Problema 2: Implemente uma heurística que tenta solucionar o problema Subsetsum usando uma estratégia gulosa.

Implemente a função `subsetSumGreedy(array, k, subset)` no arquivo **subsetsum.cpp** usando uma estratégia que aplique o princípio guloso: escolha o que for imediatamente melhor e não volta atrás. Exemplos:

- Priorizar números pequenos.
- Priorizar números grandes.
- Priorizar números medianos (menor diferença o valor médio do array).

ATENÇÃO: A estratégia gulosa não é garantida de encontrar uma solução (provavelmente vai falhar na maioria das vezes), sendo portanto uma solução aproximada baseada em heurística.

Desafio (Opcional): Tente resolver o problema Subsetsum com uma heurística baseada em D&C.

Exemplo de heurística: se K = 20, veja se é possível achar a solução dividindo o array em duas metades e procurando por K' = 10 em cada uma delas.

Parte 2: Resolvendo problemas "reais" com D&C e Greedy.

Problema 1: Revisitando o problema LC703 do LeetCode usando D&C.

Na Prática 4 resolvemos o problema *Kth Largest Element ...* usando a lista ordenada desenvolvida no início daquela prática. Desta vez usaremos uma estratégia baseada em Dividir para Conquistar. Especificamente, usaremos um algoritmo similar ao Quicksort chamado Quickselect (também proposto por Hoare), o qual é especializado em achar o n-ésimo elemento em ordem de um array de dados. Esse algoritmo realiza uma ordenação parcial do array para achar o elemento enquanto procura pelo elemento.

Implemente a função `quickselect()` no arquivo **kthlargest.cpp. Dica:** se você implementou a função `partition()` da prática 4, pode reusá-la aqui. Do contrário, terá que implementá-la.

Problema 2: Implemente a função `solve()` em **mochila.cpp** usando uma estratégia gulosa.

O problema consiste em, dada uma mochila de capacidade limitada K, colocar os itens dentro da mochila de forma a maximizar o valor total sem exceder a capacidade K. São dados dois arrays de N inteiros cada: um com os pesos do itens, outro com os seus respectivos valores. Use uma estratégia gulosa para computar o valor máximo que pode ser obtido. **Dica:** use ordenação.

Desafio (Opcional): Tente resolver o problema da mochila binária com um algoritmo guloso.

Assuma agora que os itens devem ser pegos por inteiro e use a mesma estratégia (ou parecida) que a anterior. O resultado é o ótimo?