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: FORÇA BRUTA, BACKTRACKING, BRANCH & BOUND

Prática 10

OBS.: Ver código que acompanha a prática.

Parte 1: Praticando Força Bruta, Backtracking e Branch & Bound

Problema 1: Implemente a solução em força bruta do problema Subseqmax/SSM.

Implemente a função `subseqMaxBF(array, ini, end)` no arquivo **subseqmax.cpp**. Nessa função, `array` são os valores em questão, `ini` e `end` são parâmetros de retorno que devem conter ao final as posições iniciais e finais da maior subsequência. A função deve retornar o valor da maior subsequência encontrada. A solução em Força Bruta consiste em testar todas as combinações de início e de final e ver qual oferece o melhor resultado (maior soma). Rode a função `main()` desse arquivo para testar.

Problema 2: Implemente a solução usando *Backtracking* do problema Subsetsum/SSK.

Implemente a função `subsetSumBT(array, k, subset)` no arquivo **subsetsum.cpp**. Nessa função, `array` é o conjunto total de valores, `k` é a soma desejada, e `subset` é um vetor booleano que deve conter a solução (se existir); `subset[i] = true` indica que o i-ésimo elemento do array faz parte da solução. Você pode precisar criar uma função extra para fazer a recursão em si. A função deve retornar `true` caso haja solução, e `false` caso contrário. Rode a função `main()` desse arquivo para testar.

A solução usando *Backtracking* consiste em descer recursivamente testando todas as combinações de elementos até encontrar uma cuja soma seja igual ao valor K. A formulação recursiva (sem *pruning*) é a seguinte:

$$SSK(A,K,i) = \begin{cases} V, se\ K = 0 & \textit{Caso base, achou solução} \\ V, se\ SSK(A,K,i-1) & \textit{Desconsidera o i-ésimo valor} \\ V, se\ SSK(A,K-A[i],i-1) & \textit{Considera o i-ésimo na solução} \\ F, caso\ K\ != 0\ e\ n < 0. & \textit{Esgotou o array e não achou} \end{cases}$$

Onde A é o array, K é a soma buscada, e i é o índice do elemento sendo considerado; inicialmente $i = N - 1$, onde N é o tamanho do array A.

Obs.: Não se deve gerar e guardar todas as combinações na memória, pois isso vai acarretar um uso grande e desnecessário de RAM.

ATENÇÃO: nesta versão do problema SSK vamos considerar apenas valores positivos. Portando, ao montar a árvore de soluções, qualquer ramo que já apresentar uma soma maior que o valor K desejado já pode ser abandonado (*pruning*), pois as soluções dali em diante são garantidas a falhar.

Desafio (Opcional): Implemente a solução usando *Branch & Bound* para o problema Subsetsum/SSK.

Use uma estratégia baseada em BFS e que estima o limite superior dos ramos ainda não explorados, eliminando da busca aqueles que cuja soma não deve atingir o valor desejado. **Obs.:** não há função no código atual para essa solução; adicione uma nova.

Parte 2: Resolvendo problemas "reais" com Força Bruta e Backtracking.

Problema 1: Escreva uma solução que maximize o lucro de uma viagem de negócios.

São dados dois *arrays*: um representando os custos com diárias para um conjunto de dias, e outro com estimativas de vendas a serem realizadas nesses mesmos dias. Assumindo que um vendedor deve escolher uma sequência contígua desses dias para sua viagem de negócios, escreva um algoritmo que aponte qual o lucro máximo (total de vendas – custo total) que esse vendedor vai conseguir.

Implemente a função `max_profit(costs, sales)` em **sales_trip.cpp**.

Desafio 1 (Opcional): Resolva o problema *Best Time to Buy and Sell Stock* no LeetCode (LC121).

Nesse problema é dado um *array* com preços de ações em um sequência de dias. É pedido o lucro máximo que um investidor pode obter, considerando que ele deve escolher apenas um dia para comprar as ações e outro para vendê-las.

Problema 2: Implemente a função `solve()` em **labirinto.cpp** usando *Backtracking*.

A função deve encontrar uma caminho no labirinto entre a origem ('o') e o destino ('d'), exibindo esse caminho ao final. Se baseie nas ideias do problema Chuva e Contaminação (Práticas 1 e 2).

Desafio 2 (Opcional): Resolva o problema *Restore IP Addresses* no LeetCode (LC093).

Nesse problema é dada uma string contendo um endereço IP cujos pontos que separam os octetos foram perdidos. Seu algoritmo deve gerar strings contendo os endereços válidos a partir da string dada.

Desafio 3 (Opcional): Resolva o problema *Partition on K Equal Sum Subsets* no LeetCode (LC698).

Nesse problema é dado um *array* de valores inteiros e seu programa deve responder se é possível participá-lo em K subconjuntos (não subarrays) não vazios cuja soma seja a mesma. Esse problema é uma versão mais sofisticada do problema SSK.