Algoritmos e Estruturas de Dados - Relatório 1
Comparação de Métodos de Resolução do Subset-Sum Problem

Filipe Pires	85122
João Alegria	85048

Introdução

Este relatório abarca não só a descrição da tarefa a nós proposta, da solução por nós implementada e dos resultados por nós obtidos, como também uma breve listagem dos maiores problemas de implementação que encontramos e uma conclusão sobre o trabalho no geral.

Tarefa / Objetivo

Foi-nos apresentado um problema conhecido há algum tempo de nome *Subset-Sum Problem* que consiste na tarefa de encontrar, de entre um conjunto de elementos, quais deles somados dão outro número já delineado. O nosso objetivo na realização do projeto era, portanto, já com uma base de código fornecido pelo professor, solucionar este problema através de três estratégias de implementação, todas elas com complexidades computacionais e tempos de execução diferente.

Depois de implementados esses três métodos, restava-nos testar as implementações e retirar os tempos de execução de modo a pudermos fazer os cálculos necessários para comparar as três maneiras, tal como médias, variâncias e também gráficos para visualizar a evolução do programa ao longo do código e do tempo.

Solução / Implementação

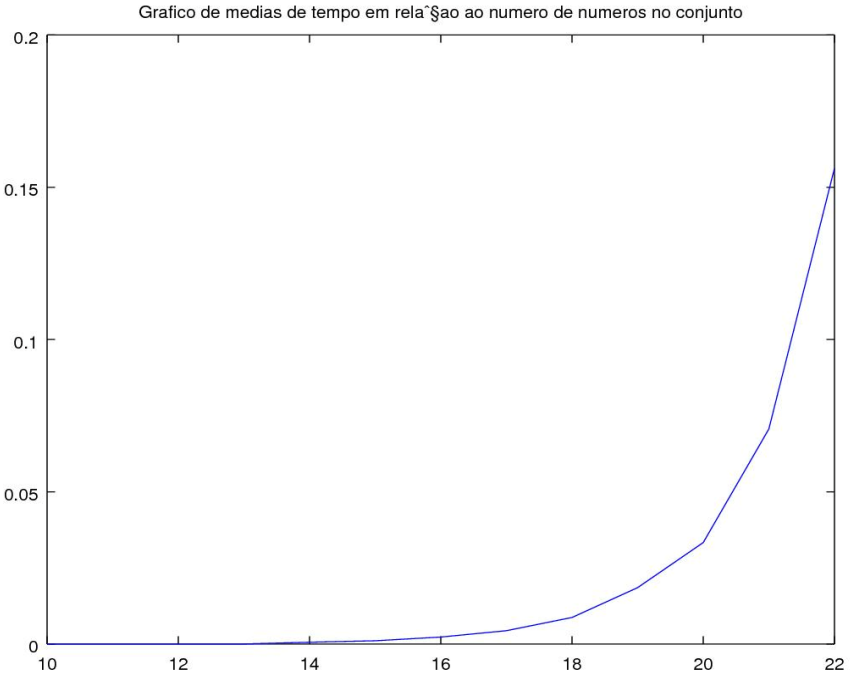
Para cada problema temos três componentes: o conjunto de números que serão somados; o número para o qual procuramos as somas dentro do conjunto; a máscara binária que nos dá a solução do exercício.

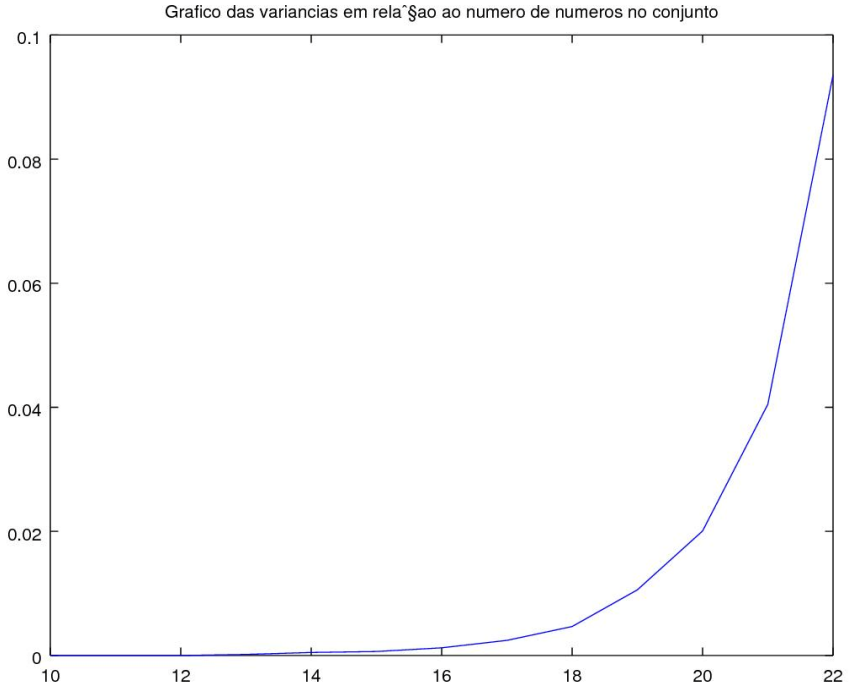
O objetivo final das três implementações é o mesmo: encontrar a máscara do conjunto que somado resulta no valor procurado. Ou seja, o resultado para cada problema é apresentado através de uma máscara de 1's e 0's, na qual cada número representa um elemento do conjunto. Caso o elemento esteja representado por um 0 na máscara, significa que este não pertence à soma final, caso esteja representado por um 1, significa que é um dos números cuja soma dá o valor que procuramos.

O conjunto de máscaras abaixo representadas são um exemplo de resultados para conjuntos de 10 elementos. Para o *Subset-Sum Problem* com conjuntos deste tamanho, temos:

[1010110011], [0000110010], [0100101000], [1001011010], [0110111001], [1110110110], [1110100111], [0000111111], [1111010010], [1100010100], [0001100111], [1101110110], [1100001010], [1111001000], [1001011000], [1001101111], [1001101100], [1000010000], [1101011001], [1011011000], [0001100001], [0111111000], [1100010111], [1110000011],[1001001011], [1001110100], [0000110110], [0100001101], [0111111100], [1011100101],[0111001101], [1001110000], [1111010101], [0101101010], [1010011011], [0000110110],[1010100000], [0001011101], [0100110000], [0111100001], [0100100101], [0011111100],[1001111000], [0101110111], [0011101011], [1101001101], [1010010110], [1100101101], [1100011110], [0011110110], [0110001101], [1011111110], [1010101111], [0100110000],[1010100110], [1100101101], [0001110001], [0101000100], [0100000110], [1111100000], [0111001101], [0001110110], [1100011001], [1001110000], [1000000000], [1110000011],[1111000000], [1101000110], [1111101101], [1111010001], [0101010111], [1110011011],[1110100001], [0011011000], [1111001010], [1101100111], [1100111100], [1010001101], [0011100000], [0001010111], [1101011001], [0110001100], [1001000100], [1010011100],[0101111000], [0000101000], [1000000101], [0000010110], [0111110001], [1111001001], [0101000101], [0010100101], [0010001101], [0000101010], [1101110101], [0011001100], [1100101011], [1001011000], [1001010110], [0111100010].

Como já foi mencionado, este relatório analisa as implementações de três estratégias de abordagem ao *Subset-Sum Problem*. A primeira a ser mencionada é do estilo “força bruta”, ou seja, nesta abordagem são geradas todas as combinações possíveis de elementos do conjunto e realizadas as respetivas somas; caso a soma coincidisse com o número pretendido, assumíamos ter encontrado o subconjunto de números cuja soma resultava no valor pretendido. Facilmente se concluiu que esta maneira seria a menos eficiente, visto que (no seu pior caso) se tem que percorrer todos os subconjuntos possíveis, o que resulta num código de complexidade $O(2^n)$. Os gráficos abaixo apresentados e conseguidos a partir de testes a esta estratégia comprovam essa conclusão:



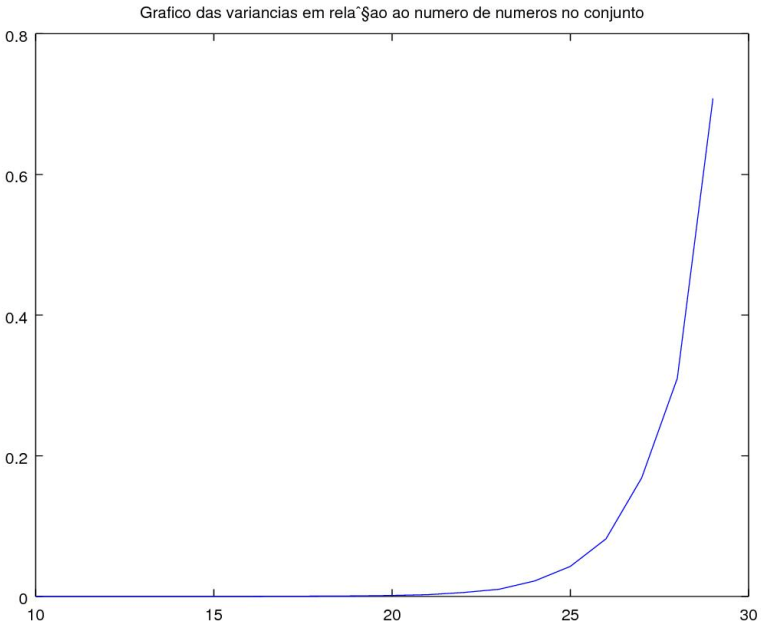


A segunda estratégia é chamada de *Branch and Bound* e resume-se na ideia de que, para cada elemento do conjunto, se gera todos os subconjuntos do conjunto principal que contenham esse elemento. De seguida, caso se encontre um subconjunto de entre os gerados em que a soma de todos os seus elemento seja o número que procurávamos, então está encontrado o subconjunto final.

Nesta abordagem o processo de procura tem uma característica reduz o tempo de execução e a complexidade computacional comparativamente com a implementação anterior: a implementação desta estratégia é feita de tal forma que é garantido que os elementos já processados não entram nos subconjuntos dos elementos ainda por processar (isto porque como não foi encontrado o subconjunto pretendido podemos concluir que o elemento que baseou a geração dos subconjuntos não pode entrar no subconjunto final).

Rapidamente se conclui que esta implementação é melhor que a anterior pois, existindo a ideia de limite de elementos que podem entrar no subconjunto, muitas das possibilidades de conjuntos são descartadas antes sequer de se “perder” tempo com elas. Nesta implementação consegue-se processar alguns dos conjuntos mais compridos fornecidos pelo professor mas continua a ser uma solução limitada.

Os gráficos abaixo apresentados representam o estudo feito ao *Branch and Bound*.

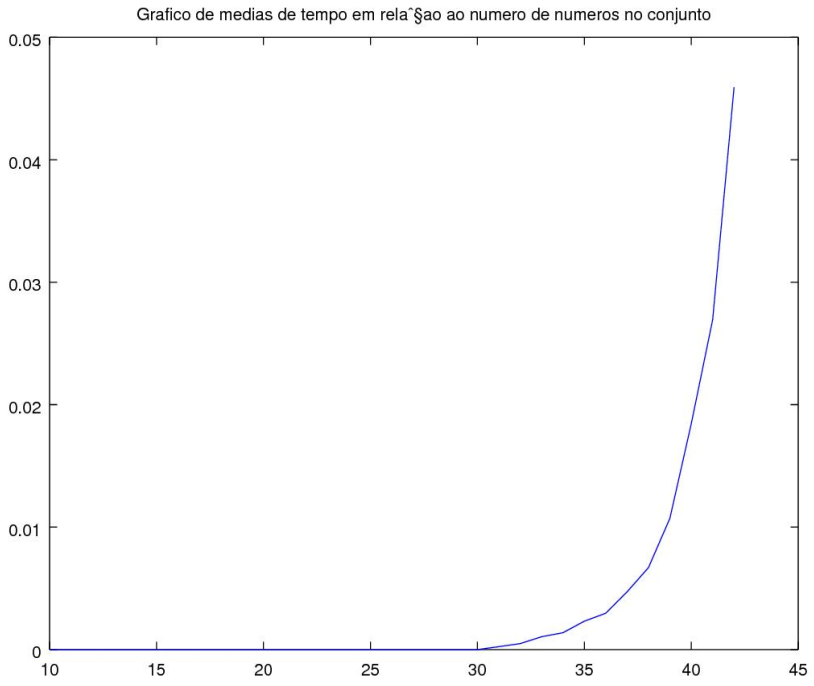


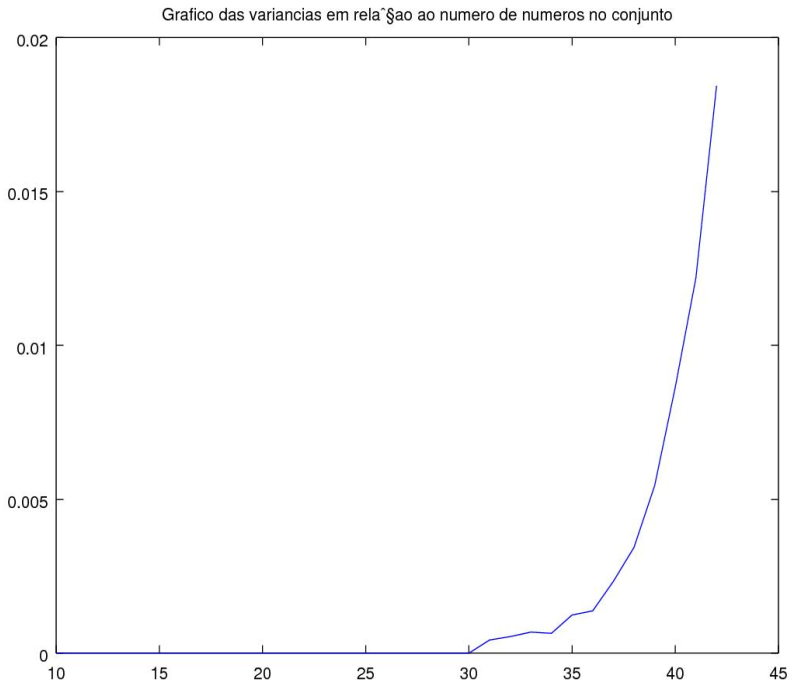
Por fim analisamos a última estratégia, de nome *Meet in the Middle* e considerada como a mais eficaz das três. Esta estratégia caracteriza-se pela divisão do problema em dois: separa-se o conjunto inicial em dois, metade para cada lado, e gera-se todos os subconjuntos possíveis desses dois conjuntos, guardando as suas somas juntamente com as suas máscaras. Fica-se então com dois conjuntos de somas e máscaras das respetivas somas, a serem ordenados de forma a reduzir o tempo de processamento do algoritmo, isto porque de seguida são declaradas duas variavéis cujos valores são o índice do conjunto com as somas menores (primeira metade do conjunto inicial e o índice do conjunto com as somas maiores (segunda metade do conjunto inicial).

Resumindo, após organizarmos os conjuntos das somas, itera-se sobre eles (no de somas menores no sentido crescente, enquanto que no das somas maiores é num sentido decrescente): alteram-se os índices segundo algumas condições até que as somas nas posições apontadas resultem no número que estávamos à procura, resultando na descoberta da máscara da solução (junção das máscaras que estão ligadas às sub-somas).

Esta é a abordagem mais eficiente pois diminui a complexidade computacional para $O(2^{n/2})$ (que resulta do facto de se processar metade do conjunto principal de cada vez), o que é um ganho considerável neste tipo de problemas de longos tempos de processamento. Com esta implementação quase que conseguimos processar todos os problemas fornecidos pelo professor, ficando apenas limitados pelas capacidades de processamento dos nossos computadores.

Abaixo estão apresentados os gráficos que representam o estudo feito ao *Meet in the Middle*.





Problemas / Dificuldades

Ainda que o desenvolvimento do código tenha encontrado alguns percalços no seu desenvolvimento, nós rapidamente solucionamos a questão, pois muitas vezes eram erros de sintaxe, e caso o erro fosse de alguma maneira mais subtil e que nos passasse despercebido, pudemos pedir ajuda ao professor e as dicas obtidas facilitou-nos bastante a questão dos bugs e erros encontrados.

A implementação mais difícil a completar foi a *Meet in the Middle* pois é, das três, a mais complexa em termos de código, mas acaba por compensar visto ser a menos complexa a nível computacional. Os únicos erros encontrados nesta fase foram solucionados com perseverança e calma e podemos garantir que o programa está completo.

Conclusão

Concluímos com este trabalho que a estratégia *Meet in the Middle* é a melhor para solucionar o *Subset-Sum Problem*.

Podemos retirar relativamente a este relatório também que o funcionamento dos elementos do grupo como par de trabalho é muito produtivo e fluiu muito bem no desenvolvimento de uma solução para o problema.

Quanto ao trabalho dos dois elementos, concluímos que mantivemos uma comunicação constante de forma a estarmos sempre a par do ponto de situação do trabalho e que funcionamos bem neste tipo de tarefas.

Quanto à solução por nós implementada, podemos concluir que apresenta as três implementações e que todas elas estão completamente operacionais, e pensamos que a sua implementação está feita de maneira a serem o mais eficazes possível.

Fontes de Pesquisa:

https://stackoverflow.com/
https://www.tutorialspoint.com/cprogramming/index.htm
https://www.tutorialspoint.com/c_standard_library/index.htm