INF 112 – Trabalho Prático 0

Alunos:
Rodrigo Eduardo de Oliveira Bauer Chichorro – 92535
Matheus Ferreira Nunes – 92537

a) Criou-se um código, com o nome de tp0a.cpp, que, dado um número de produtos, gera de forma recursiva todas as combinações possíveis de produtos que podem ser escolhidas para o problema da mochila.

Forma de uso:

Ao executar o código, será pedido ao usuário o número de produtos. Logo após, serão imprimidas na tela representações de todas as combinações possíveis, em que cada linha representa uma combinação diferente, de 0 a m-1, em que m é número máximo de combinações possíveis, e cada coluna representa um produto diferente, de 0 a n-1, em que n é o número de produtos que foi digitado pelo usuário.

Por exemplo, ao passar o valor 2 como número de produtos, teremos como saída:

```
Combinacao 0: 1 1
Combinacao 1: 1 0
Combinacao 2: 0 1
Combinacao 3: 0 0
```

Assim, na combinação 0 pegamos os produtos 0 e 1, que possuem o valor 1 (verdadeiro). Na combinação 1 pegamos apenas o produto 0; na combinação 2 pegamos apenas o produto 1; e na combinação 3 não pegamos nenhum produto.

O programa gera a matriz de booleanos que representa as combinações coluna por coluna.

Lógica do gerador de combinações:

Como mostrado anteriormente, representamos uma combinação através de um arranjo de booleanos (que é uma linha da matriz), em que **true** significa que carregaremos o produto e **false** significa que não carregaremos o produto. Assim, podemos imaginar uma combinação como um número binário.

Sabemos que o número total de combinações é 2^n, em que n é o número de produtos.

Logo, para gerar as combinações, temos que a primeira combinação será 2^n-1, que é preencher todas as posições do arranjo com **true**.

A seguir, caso ainda existam combinações para gerar, teremos que o número binário que representa a próxima combinação é a combinação anterior menos 1.

Exemplo: Caso a combinação anterior tenha sido 11010, a próxima será 11010 – 00001 = 11001.

b) O código do item a) foi copiado, adaptado e renomeado para tp0b.cpp. Este novo código recebe como entrada o número de produtos, o peso limite da mochila e o peso e valor individual de cada produto, e dá como resposta a melhor combinação que cabe na mochila.

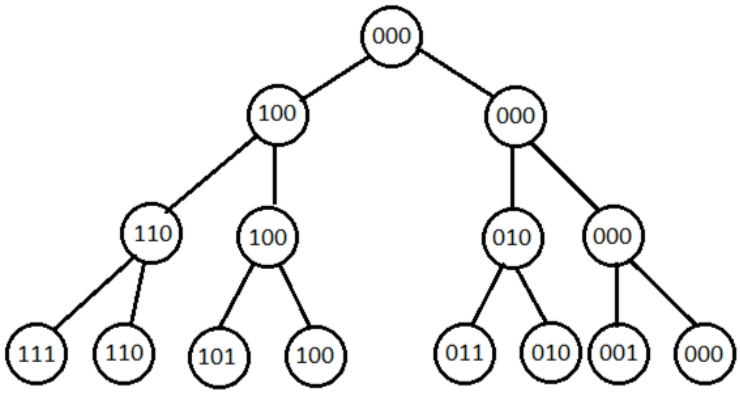
O código gera todas as combinações, da mesma forma que o tp0a.cpp, e depois verifica qual é a melhor solução, através de comparações simples entres as combinações.

A saída é dada no seguinte formato (Exemplo de um entrada qualquer):
Devemos carregar os produtos: 0, 2, levando assim 6Kg e 80 reais.

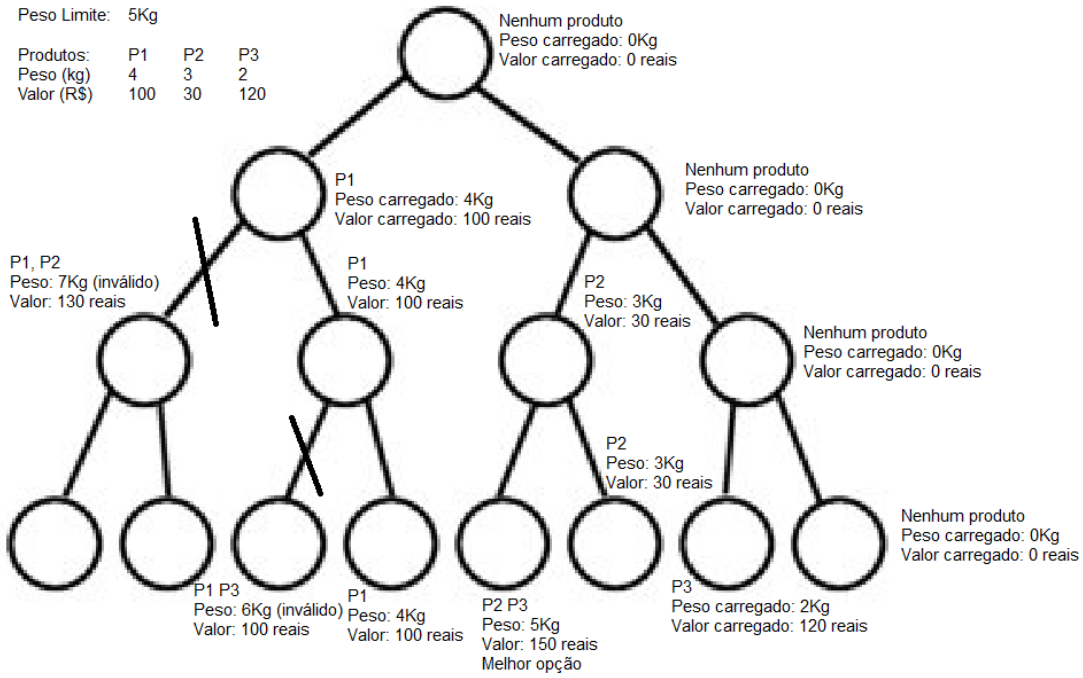
c) O problema que persiste no código é a falta de um caso de parada se o peso dos produtos extrapola a capacidade limite da mochila. Para resolver, basta implementar o backtracking, a fim de diminuir o número de combinações a serem verificadas.

d) Sendo que temos 32 combinações totais no exemplo (2^5), e que dessas 32, 8 delas envolvem selecionar os produtos P1 e P2 [2^(5-2)], temos que o computador faria apenas a primeira dessas 8 combinações, e as 24 restantes, totalizando 25 cálculos, pois 7 cálculos foram “economizados”.

e)



f)



g) Um terceiro código, criado a partir do tp0b.cpp foi criado, chamado de tp0_backtracking.cpp. Contudo, modificamos o gerador de combinações que foi

usado no tp0a.cpp e tp0b.cpp. Agora, ele gera as combinações de forma iterativa (preferimos trabalhar nesta forma), e utiliza uma variável temporária (opcoes_finais) para qualquer uma combinção que será testada. A melhor opção é armazenada em uma outra variável (resposta).

O backtracking é realizado da seguinte forma:

A combinação é criada utilizando uma lógica quase igual às versões anteriores do código. Assim, começamos da combinação 2^n-1, que é preencher todo o arranjo com **true**, ou seja, lebar todos os produtos.

Contudo, testamos se a combinação que foi gerada, coluna por coluna – ou seja, produto por produto –, é válida. Caso ela não seja, pulamos todas as combinações que tenham a configuração que falhou.

Exemplo: Suponha que, ao testar a combinação 10111, descobrimos que a solução falhou em 101. Assim, pularemos todas as combinações no formato 101xx, e iremos para a combinação 10011.

O “pulo” é realizado de uma maneira simples: se queremos pular todas as combinações no formato 101xx, devemos subtrair 00100 da combinação atual, no caso 10111. Assim, teremos 10111 – 00100 = 10011.

Caso a combinação ainda esteja dentro do limite de peso, a próxima combinação a ser testada será a que possui 1 unidade a menos da atual, como era feito nas versões anteriores. Por exemplo, se última combinação testada foi 11010, a próxima será 11001.