

**CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
DE MINAS GERAIS**

BRUNO NASCIMENTO DAMACENA

TRABALHO PRÁTICO 1

Trabalho apresentado à disciplina
de Laboratório de Algoritmos e
Estrutura de Dados I do curso
de Engenharia de Computação do
Departamento de Computação do
CEFET-MG

Professora Natália Cosse Batista.

**Belo Horizonte
Setembro de 2017**

O problema da mochila é o nome dado ao modelo de uma situação em que é necessário preencher uma mochila com objetos de diferentes pesos e valores. O objetivo do problema é que se preencha a mochila com o maior valor possível, não ultrapassando o peso máximo que a mochila pode suportar.

O objetivo do trabalho era entender o problema da mochila, e gerar algoritmos de solução do mesmo. Foi pedido dois tipos de estratégia para resolver esse problema: por tentativa e erro e de forma gulosa.

O algoritmo por tentativa e erro, como o próprio nome diz, percorre todas as possibilidades de combinações de itens, e guarda a melhor combinação, ou seja, a de maior valor. Esse algoritmo vai sempre retornar a melhor solução, porém tem um custo computacional alto, devido ao teste de todas as possibilidades. Como ele faz combinações de n elementos, seu grau de complexidade é exponencial. A estratégia de tentativa e erro é descrita por esse algoritmo:

```
Algoritmo Tentativa_e_Erro (parcial , geral , tam , num)
    para nao condicao_de_parada
        para i de 0 a num
            se parcial[i].cond = 1
                tlocal = tlocal + parcial[i].item.peso
                vlocal = vlocal + parcial[i].item.valor
            fim se
            se tlocal > tam
                estourou
            fim para
        fim para
        se vlocal > vgeral e nao estourou
            para i de 0 a num
                geral[i] = parcial[i]
            fim para
            vgeral = vlocal
        fim se
        se parcial.cond tiver algum elemento = 0
            condicao_de_parada = 0
        senao
            condicao_de_parada = 1
        fim se
    fim para
fim Tentativa_e_Erro
```

O segundo algoritmo, de estratégia gulosa, tenta sempre pegar o melhor local, afim de ter uma solução global próxima da melhor. Ele tem um custo computacional muito menor, de ordem polinomial, mas ele gera um resultado aproximado. A estratégia gulosa é descrita por esse algoritmo:

```

Algoritmo Guloso (tamanho, numero, item)
    Ordena item de acordo com valor/peso, de forma
    decrescente
    peso=0
    custo=0
    para i de 1 a numero
        se item[i].peso <= tamanho
            adiciona item[i] a solucao
            decrementa o tamanho com o valor
            do peso do item
            incrementa o custo com o valor
            do custo do item
        fim se
    fim para
fim Algoritmo Guloso

```

O custo computacional de cada algoritmo é dado da seguinte forma:

$O(2^n)$ para o algoritmo de Tentativa e Erro;

$O(n^2)$ para o algoritmo Guloso

O algoritmo de força bruta é $O(2^n)$ porque a maior iteração que ele fará é percorrer todas as combinações possíveis de um vetor de n elementos. Já o guloso é $O(n^2)$ porque a maior iteração que ele fará é a ordenação, que tem um custo de $O(n^2)$ no caso médio e no pior caso. De fato, o custo computacional do algoritmo guloso no melhor caso é $O(n)$.

Depois de implementados os pseudo-códigos para cada caso, foi hora de codá-los em uma linguagem de alto nível. A linguagem utilizada foi C. Os dois algoritmos, assim como as instruções de como compilá-los podem ser encontrados no repositório do GitHub <https://github.com/BrunoDamacena/OProblemaDaMochila>

Uma vez implementados, eu criei um banco de dados de entradas, com valores de n indo de 10 a 500, e testando cada algoritmo em cada entrada. Os resultados também se encontram no repositório do GitHub.

Para n menor que 25, o algoritmo de tentativa e erro conseguiu ser executado em poucos segundos (para $n=25$, o tempo de execução foi de aproximadamente 15 segundos. Porém para valores maiores, o algoritmo tentativa e erro se mostrou inviável. Para $n=30$, o tempo de execução foi de quase 10 minutos!).

O algoritmo guloso, em contrapartida, não teve problema nenhum para valores de n grandes, já que calculou problemas com complexidade 1000 em poucos milissegundos.

Para estimar o tempo que o algoritmo de tentativa e erro levaria para gerar um resultado de complexidade $n = 100$, eu tive que calcular o tempo de cada

iteração baseado nos resultados anteriores. Para $n = 30$, o número de iterações foi $2^{30} = 1073741824$. O tempo de execução foi de 577714 milisegundos. Logo, para cada milisegundo, aproximadamente 1858,6 iterações são feitas. Portanto, a fórmula de calcular o tempo de execução do algoritmo de tentativa e erro, aproximadamente, é de $2^n / 1858,6$ milisegundos.

Alguns tempos de execução já calculados:

Para $n = 40$, tempo de execução de 6 dias, 20 horas e 20 minutos;

Para $n = 50$, tempo de execução de 19 anos, 5 meses, 21 dias, 7 horas e 48 minutos;

Para $n = 75$, tempo de execução de mais de 600 mil milênios;

Para $n = 100$, tempo de execução de mais de 20 bilhões de milênios!

Formato de entrada dos dados

Os n itens da loja serão fornecidos em um arquivo de texto que contém na primeira linha a capacidade da mochila, na segunda linha o número de itens da loja e nas linhas restantes, o peso seguido do valor de cada item. Exemplo:

```
10
4
1 5
2 10
4 2
6 6
```

Formato de saída dos dados

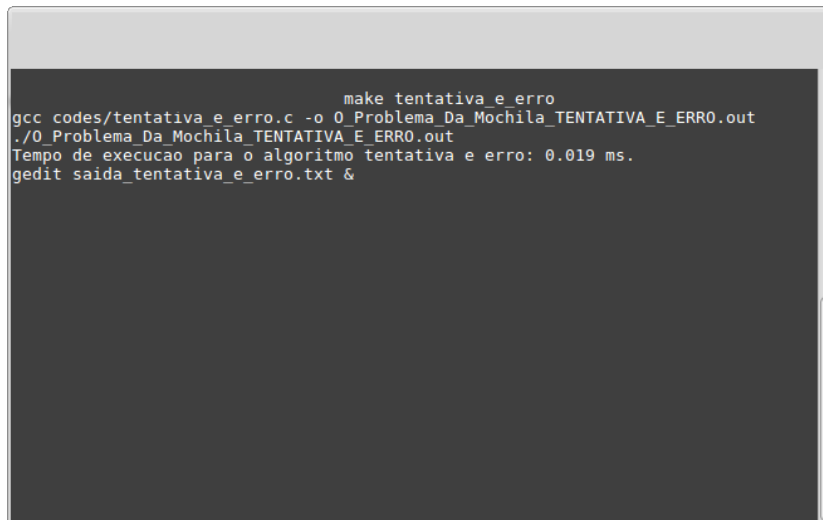
O programa gerará duas saídas: a primeira será um arquivo de texto com todos os itens escolhidos, seus respectivos índices, pesos e valores, o somatório dos pesos e dos valores. Exemplo:

```
Capacidade da mochila: 10
Itens inseridos:

Item 1 de peso 1 e valor 5
Item 2 de peso 2 e valor 10
Item 4 de peso 6 e valor 6

Peso total: 9
Valor total: 21
```

A segunda saída será a impressão em tela do tempo de execução do programa. Exemplo:



```
make tentativa_e_erro
gcc codes/tentativa_e_erro.c -o 0_Problema_Da_Mochila_TENTATIVA_E_ERRO.out
./0_Problema_Da_Mochila_TENTATIVA_E_ERRO.out
Tempo de execução para o algoritmo tentativa e erro: 0.019 ms.
gedit saida_tentativa_e_erro.txt &
```

Instruções de como utilizar o programa

Observação: para melhor funcionamento do programa, este deve ser compilado e executado em Linux, e eu irei ensinar a fazê-lo neste SO. Porém, uma vez que você tenha os algoritmos e uma IDE, você também pode executá-lo no Windows.

Abra o terminal no diretório do código e digite uma das opções:

- "make entrada" para gerar uma entrada válida para o problema, com complexidade a sua escolha;
- "make guloso" para compilar executar o algoritmo guloso e abrir o arquivo solução;
- "make tentativa_e_erro" para compilar e executar o algoritmo tentativa e erro e abrir o arquivo solução;
- "make clean" para apagar os arquivos executáveis e os arquivos de saída.

Essas instruções também podem ser encontradas no repositório do GitHub.