UNIVERSIDADE FEDERAL DE VIÇOSA - CAMPUS FLORESTAL
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CIÊNCIA DA COMPUTAÇÃO
ALGORITMOS E ESTRUTURAS DE DADOS I

AVALIAÇÃO DE IMPACTO DO PROBLEMA DA MOCHILA CALCULANDO TODAS AS COMBINAÇÕES

OTÁVIO SANTOS 3890

PEDRO CARDOSO 3877

FLORESTAL

2019

SUMÁRIO

1. INTRODUÇÃO 3
2. OBJETIVO 4
3. CÓDIGO 4
4. TEMPO DE EXECUÇÃO 6
5. ESPECIFICAÇÕES 7
6. QUESTIONAMENTO 7
7. CONSIDERAÇÕES FINAIS 8
8. REFERÊNCIAS 9

1. INTRODUÇÃO

Este trabalho fez a implementação de um algoritmo para a resolução do problema da mochila por meio da geração de todas as combinações possíveis, buscando dessa forma encontrar a solução correta para o problema. Além disso, foram executados diversos testes para conseguir entender a função O() do programa e, dessa forma julgar se ele executa em um tempo adequado para tamanhos diferentes de entrada. Por fim, este trabalho buscou também responder à seguinte pergunta: *“Seria razoável executar o seu algoritmo para valores de N maiores do que 200?”.*

2. OBJETIVO

O objetivo desse trabalho prático é permitir a avaliação do impacto causado pelo desempenho dos algoritmos em sua execução real.

3. CÓDIGO

Para a implementação do programa, o algoritmo de geração de todas as combinações possíveis foi retirado do site GeeksForGeeks[1]. A partir desse código, apenas foi preciso fazer algumas alterações para salvar a melhor combinação possível.

Assim, para melhor organização do trabalho, dividimos o programa em três arquivos: *Main.c, Combinacao.h* e *Combinacao.c*.

Combinação é composto pelas seguintes funções:

```
void bestCombination(item itens[], int n, item result[]);

void allCombinations(item itens[], int n, int r, int index, item data[], int i, item res[]);
```

A função allCombinations calcula todas as combinações e retorna a melhor daquele tamanho de entrada. Para melhor explicar, veja a fórmula de combinações simples:

$$C_{n,p} = \frac{n!}{p!(n-p)!}$$

A função allCombinations calcula todas as combinações de tamanho n = tam e p = i, sendo tam a quantidade de itens de entrada e i o tamanho da combinação atual que está sendo calculada, e retorna a melhor combinação de tamanho i.

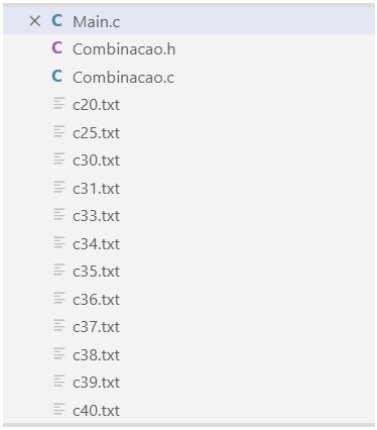
Por sua vez, a função bestCombination recebe o retorno da função allCombinations e salva a que possui o melhor valor para todos os valores de i, ou seja, para as combinações de todos os tamanhos possíveis. Assim, a função bestCombination retorna a melhor combinação para o Main.c, que imprime esse retorno.

Portanto, recapitulando, o programa ficou definido da seguinte maneira: O main possui uma interface com um menu. A partir da escolha da leitura de um arquivo e definido qual arquivo é, é feita a leitura desse arquivo e tudo é salvo em um vetor de itens de tamanho tam. Um item possui os campos peso e valor. Logo em seguida é chamada a

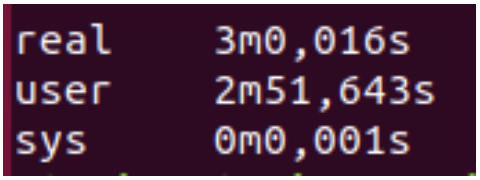
[1] Ver referências.

função bestCombination, passando o vetor itens como parâmetro e seu tamanho. Essa função chama a função allCombinations tam vezes, que por sua vez calcula cada uma das melhores combinações até a de tamanho tam. Por fim, bestCombination retorna a melhor combinação para o main e então esse retorno é impresso na tela.

Para teste do programa, foram utilizados os seguintes arquivos, contendo cada um a respectiva quantidade de itens. Por exemplo, c20.txt contém 20 itens.



Após executar o programa passando como entrada esses valores conseguimos visualizar o tempo de execução gasto pelo programa. Para isso, foi utilizado o comando time do Unix. Esse comando gera o seguinte retorno:



De acordo com o site Hostinger[2], “real – se refere a time decorrido entre a execução e a conclusão do comando. user – é o tempo gasto pelo usuário no processador. sys – é o tempo usado pelo sistema (kernel) para executar o comando.” Dessa forma, para entendimento do tempo de execução do algoritmo foi utilizado o tempo mostrado por user.

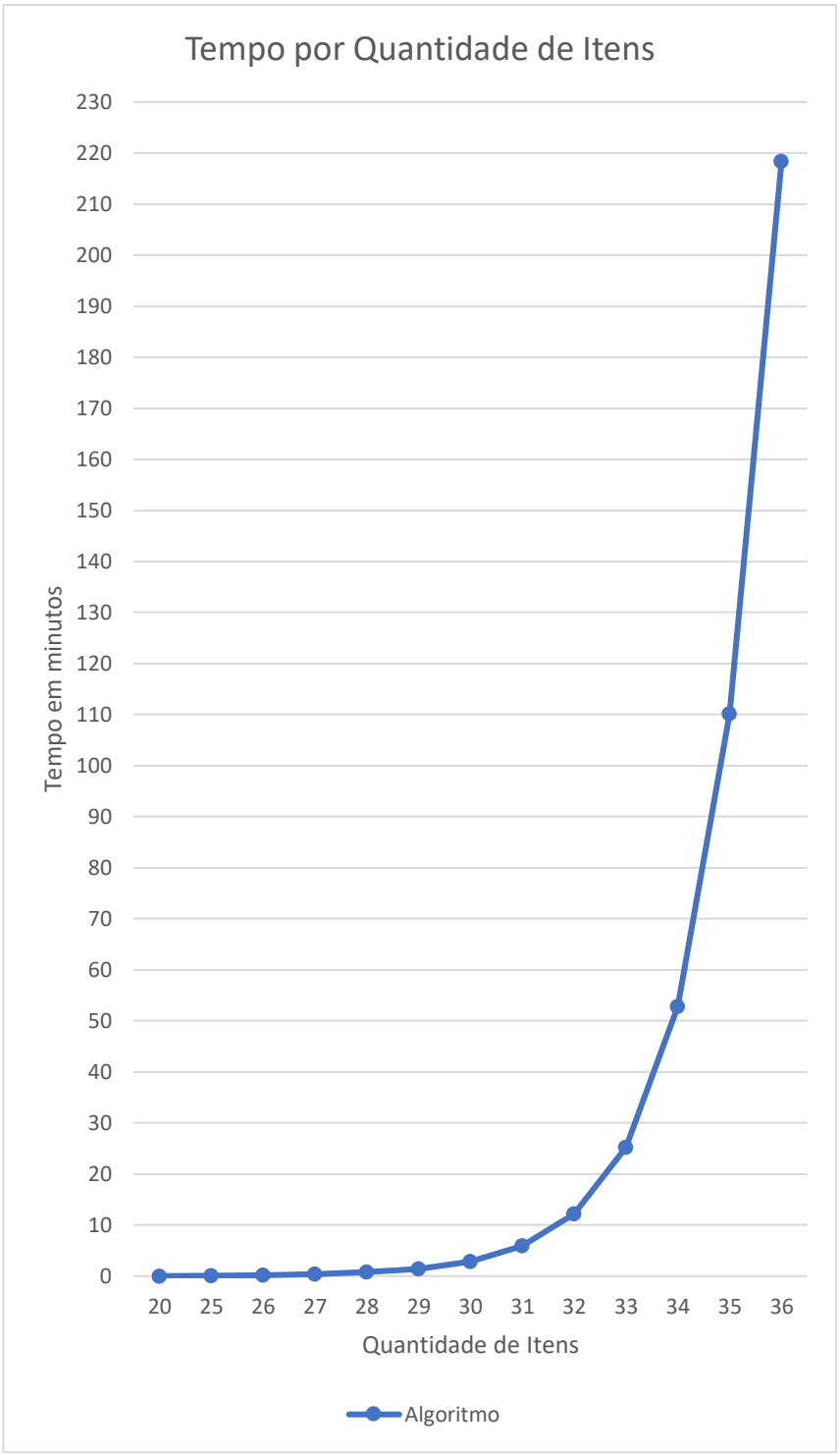
[2] Ver referências.

4. TEMPO DE EXECUÇÃO

Veja a tabela e o gráfico abaixo:

Quantidade de Itens	Tempo gasto (min)
20	0,002
25	0,076
26	0,209
27	0,394
28	0,766
29	1,418
30	2,860
31	5,906
32	12,211
33	25,205
34	52,767
35	110,189
36	218,388
37	440*
38	880*
39	1780*
40	3600*

*estimativas

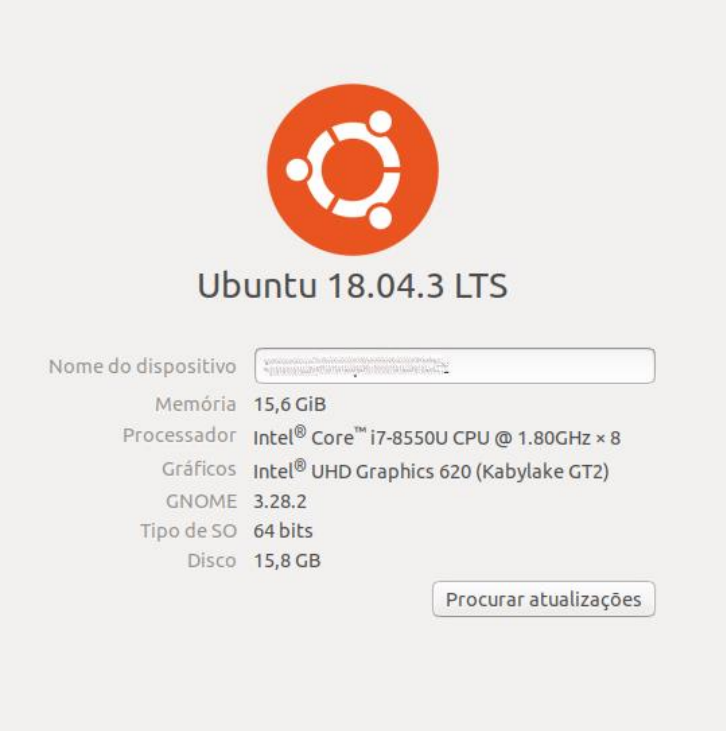


Por meio da tabela, é possível perceber que o tempo gasto para execução do programa pouco mais que dobra a cada aumento de um item de entrada. Além disso, analizando o gráfico, percebe-se um aumento exponencial de tempo gasto para execução do problema. Dessa forma, o algoritmo torna-se exponencialmente mais demorado e,

consequentemente, com um número pequeno de entradas já se torna inviável sua execução para a solução desse problema.

5. ESPECIFICAÇÕES

Especificações da máquina na qual o programa foi testado:



6. QUESTIONAMENTO

Quanto à pergunta: *"Seria razoável executar o seu algoritmo para valores de N maiores do que 200?"*, certamente a resposta é não. Vamos às justificativas. Primeiramente, o número de combinações 11 a 11 existentes entre 200 itens é 387.790.074.428.411.260, mais de 387 quatrilhões de combinações. Somando todas as combinações até 200 a 200, supera em muito a capacidade do computador mais pontente hoje, que para efeito de compração, consegue executar 200 quatrilhões de operações por segundo (IBM Summit). Em segundo lugar, é possível perceber que o tempo um pouco mais que dobra a cada adição de item (Gráfico 1). Dessa forma, o algoritmo levaria um tempo muito elevado para ser executado. Portanto, está claro que não é viável executar o problema para valores maiores que 200. Na verdade, em um computador convencional, como o utilizado para o teste, valores próximos de 40 já se tornam inviáveis, uma vez que já demorariam mais de 60 horas para serem executados.

7. CONSIDERAÇÕES FINAIS

Finalmente, podemos concluir que essa aplicação é ineficiente para resolver o problema da mochila, uma vez que é necessário executar todas as combinações possíveis para encontrar a melhor solução. Além disso, mesmo nesse código, é possível implementar pequenas alterações que podem melhorar o desempenho do algoritmo. Porém, esse não foi o foco desse trabalho.

Assim, chegamos à conclusão que o aumento exponencial gerado pelo algoritmo faz com que o programa se torne inviável para valores maiores que 35, que já leva 1h50min para ser executado. Além disso, nem sequer foi possível testar o tempo de execução do algoritmo para os valores 50, 80 e 100, uma vez que levariam um tempo gigantesco para finalizarem.

8. REFERÊNCIAS

L. Andrei. Como Usar o Comando Time do Linux. Hostinger, 2019. Disponível em: <https://www.hostinger.com.br/tutoriais/comando-time-linux> Acesso em: 31 de outubro de 2019.

PRINT all possible combinations of r elements in a given array of size n. GeeksforGeeks. Disponível em: <https://www.geeksforgeeks.org/print-all-possible-combinations-of-r-elements-in-a-given-array-of-size-n/> Acesso em: 28 de outubro de 2019.