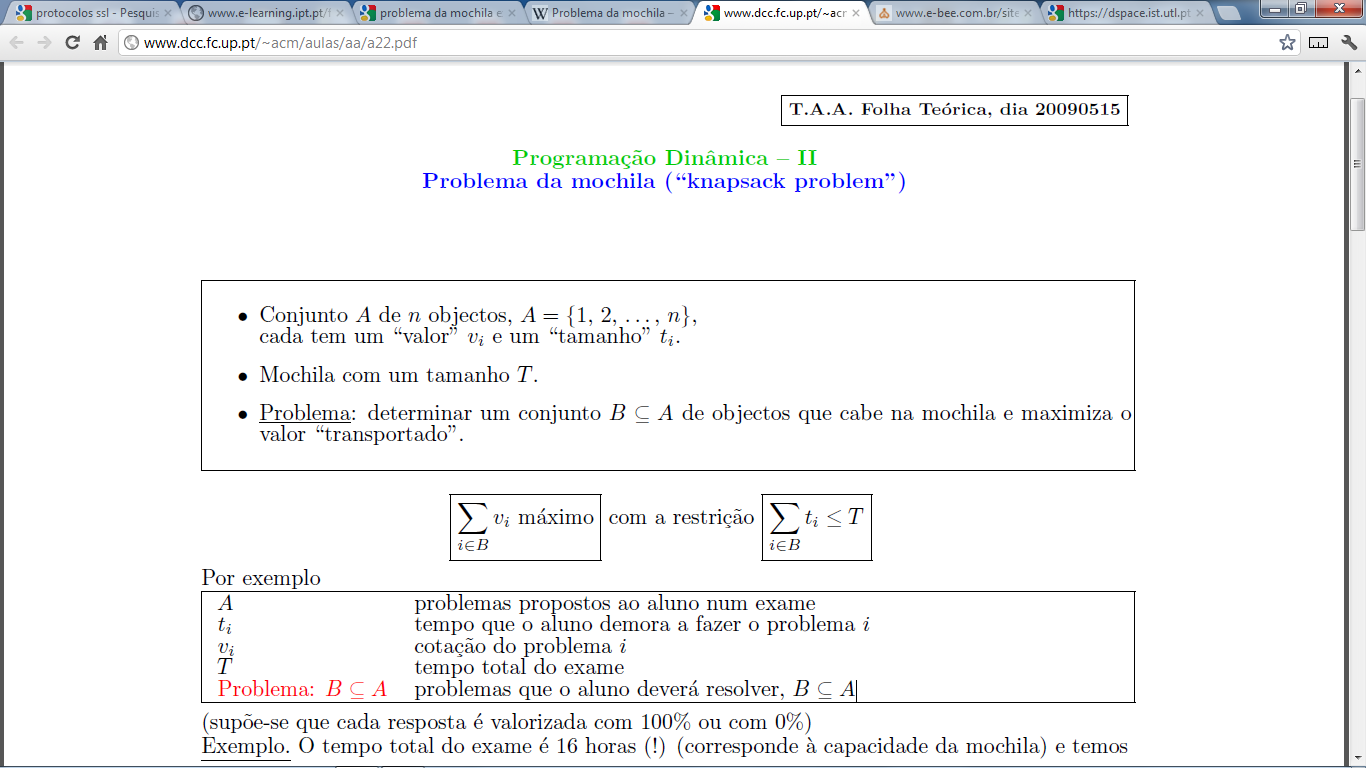
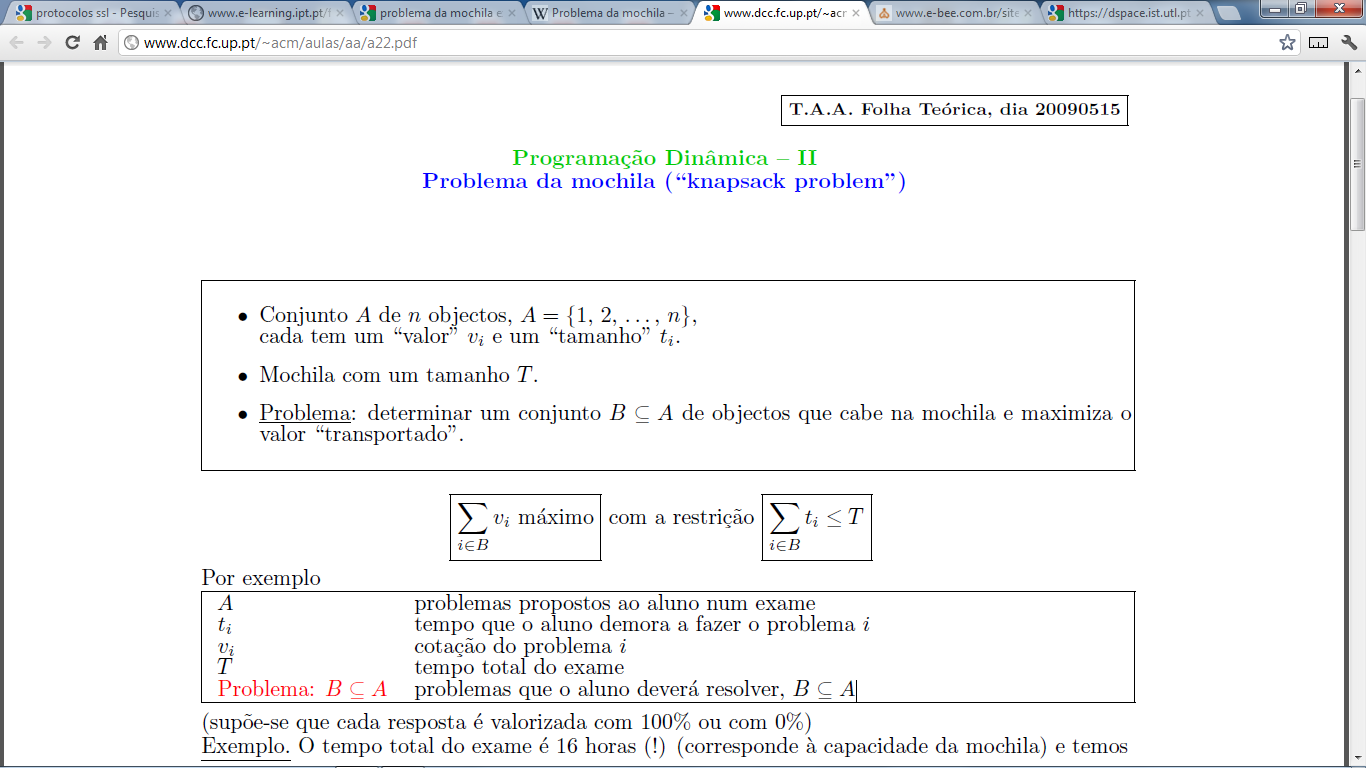
* \_ Conjunto A de n objectos, A = f1; 2; : : : ; ng,
* cada tem um \valor" vi e um \tamanho" ti.
* \_ Mochila com um tamanho T.
* \_ Problema: determinar um conjunto B \_ A de objectos que cabe na mochila e maximiza o
* valor \transportado".





(sup~oe-se que cada resposta \_e valorizada com 100% ou com 0%)

Exemplo. O tempo total do exame \_e 16 horas (!) (corresponde \_a capacidade da mochila) e temos

Problema: 1 2 3 4 5 6

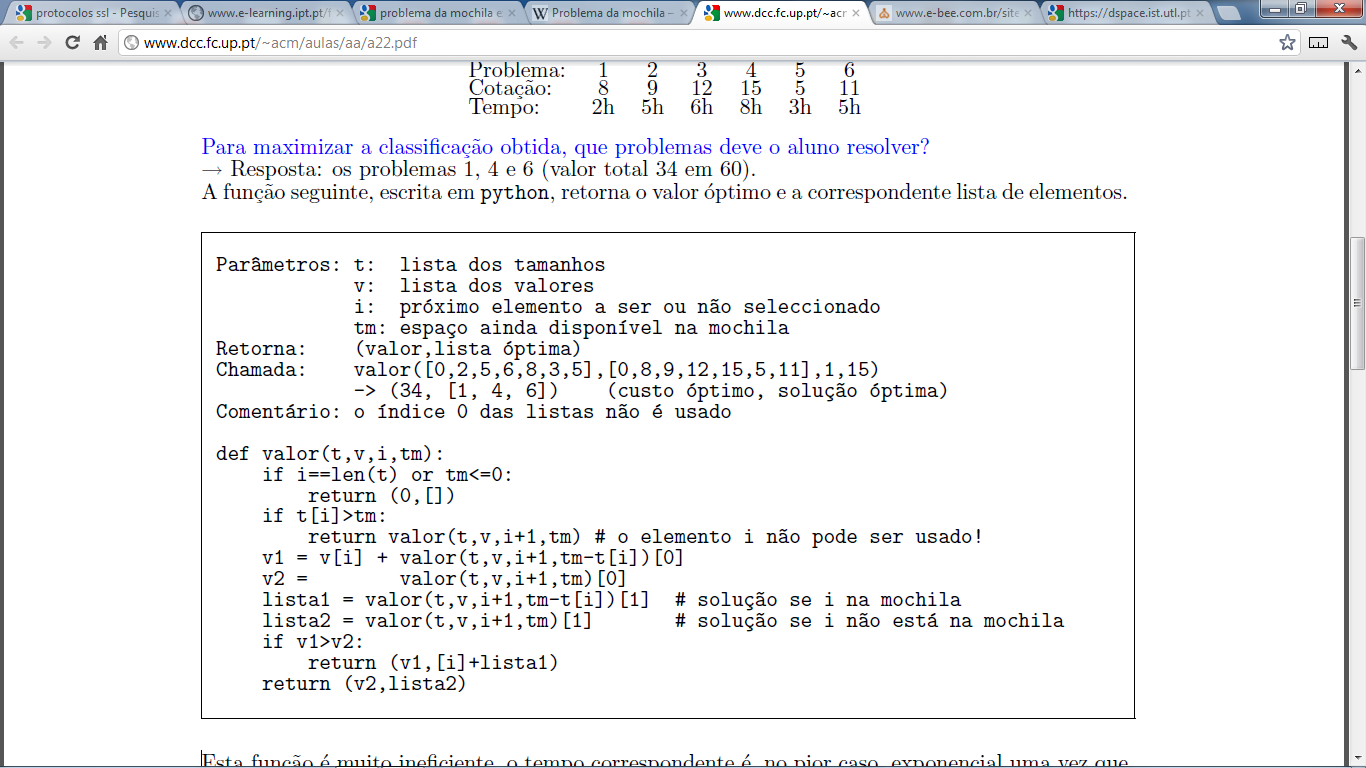
Cotação: 8 9 12 15 5 11

Tempo: 2h 5h 6h 8h 3h 5h

Para maximizar a classificação obtida, que problemas deve o aluno resolver?

! Resposta: os problemas 1, 4 e 6 (valor total 34 em 60).

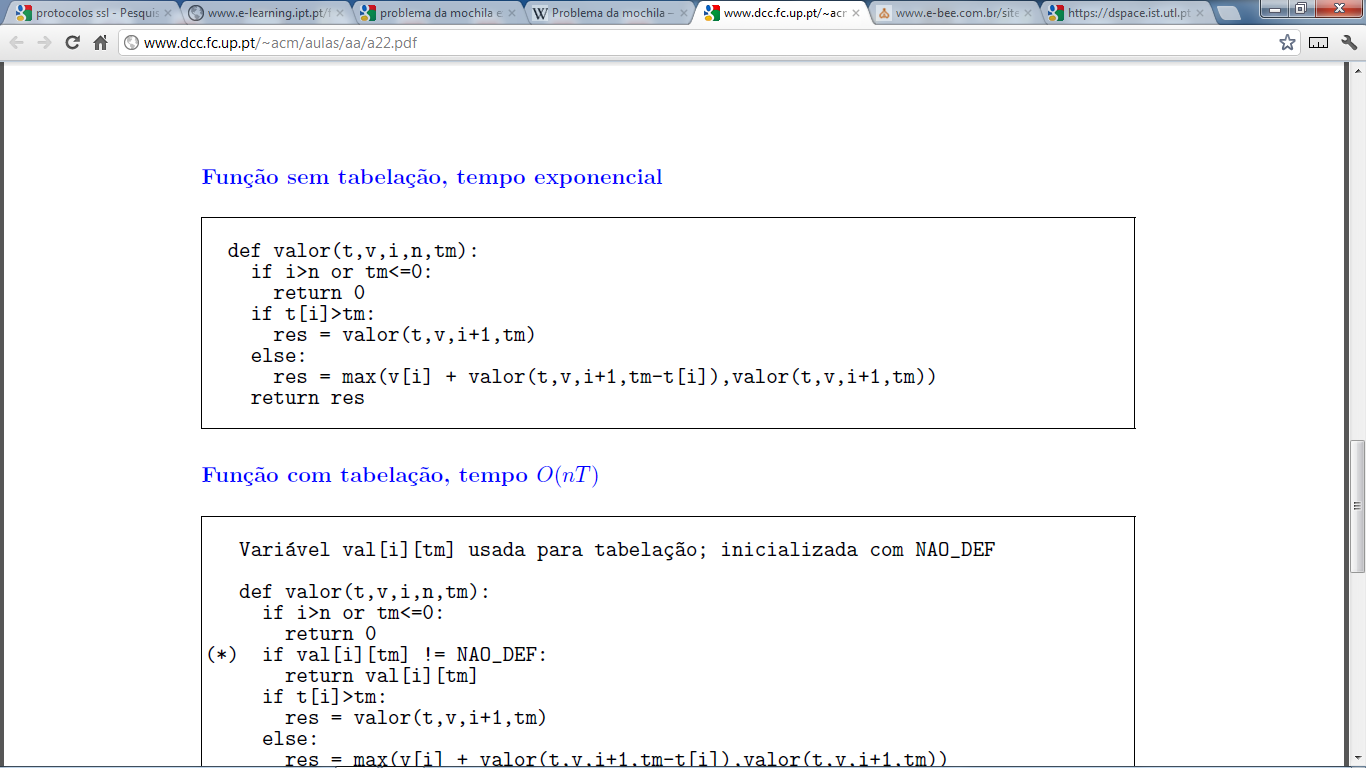
A função seguinte, escrita em python, retorna o valor ótimo e a correspondente lista de elementos.

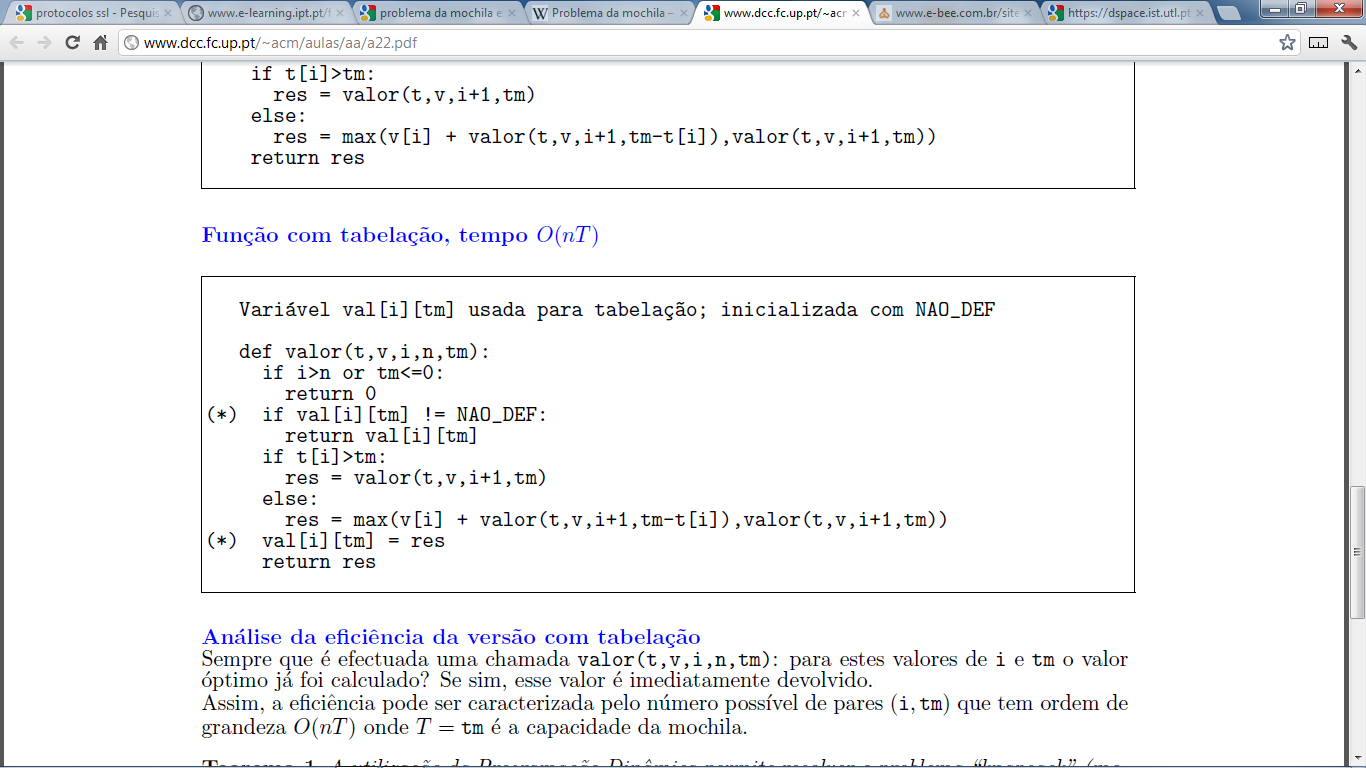


Esta função \_e muito ineficiente, o tempo correspondente \_e, no pior caso, exponencial uma vez que

cada um dos n elementos \_e selecionado ou não (2 alternativas). O numero de subconjuntos de A

com |A| = n \_e 2n e o tempo de execução da função \_e O(2n).





Análise da eficiência da versão com tabulação

Sempre que \_e efetuada uma chamada valor(t,v,i,n,tm): para estes valores de i e tm o valor

óptimo já foi calculado? Se sim, esse valor \_e imediatamente devolvido.

Assim, a eficiência pode ser caracterizada pelo número possível de pares (i; tm) que tem ordem de grandeza O(nT) onde T = tm \_e a capacidade da mochila.

Teorema 1 A utilização da Programação Dinâmica permite resolver o problema \knapsack" (mochila) em tempo O(nT) onde n \_e o número total de elementos e T \_e a capacidade da mochila. Qualquer método sem tabulação baseado na consideração de todos os subconjuntos possíveis demora um tempo exponencial em n.

