Respostas à Lista 2 da disciplina de Introdução à Análise de Algoritmos

por Guilherme de Abreu Barreto[1]

Exercício 1

O seguinte algoritmo, proposto por George Dantzig[2], ordena uma lista de itens em ordem decrescente de razão valor por peso, para então inserir tais itens em sequência na mochila até que isto não seja mais possível.

```
greedyKnapsack (v,w,n,W)
    quickSort(v,w,n)
    j ← 1
    for i ← 1 to n, do
        if (w[i] > W),
            then continue
        W ← W - w[i]
        I[j] ← i
        j ← j + 1
    end
    return I
end
```

Um exemplo de saída de um programa escrito de acordo com tal algoritmo[3] é:

```
> time ./greedy_knapsack.out < test.txt
Capacidade da mochila: 1350
Conteúdos:
Item: 7     Valor:   830 Peso:    56 Razão: 14.821428
Item: 27    Valor:   654 Peso:   112 Razão: 5.839286
Item: 37    Valor:   836 Peso:   154 Razão: 5.428571
Item: 15    Valor:   393 Peso:    94 Razão: 4.180851
Item: 36    Valor:   437 Peso:   122 Razão: 3.581967
Item: 50    Valor:   894 Peso:   264 Razão: 3.386364
Item: 43    Valor:   690 Peso:   208 Razão: 3.317308
Item: 31    Valor:   647 Peso:   206 Razão: 3.140777
Item: 17    Valor:    66 Peso:    74 Razão: 0.891892
Capacidade restante da mochila: 60
Valor total armazenado: 5447
```

```
_____________________________________________________________
Executed in    2,02 millis    fish           external
   usr time    1,90 millis  453,00 micros    1,45 millis
   sys time    0,18 millis  180,00 micros    0,00 millis

>
```

O algoritmo apresentado trata-se de um *algoritmo de aproximação*. Isto é, este não necessariamente provê a solução ao problema, mas provê uma solução que se aproxima desta de maneira computacionalmente eficiente por uma margem demonstrável. Por exemplo, considere os seguintes parâmetros:

- Mochila com capacidade 4;
- Lista com 3 itens:
  - Item 1:
    - Peso: 3;
    - Valor: 5.
  - Itens 2 e 3:
    - Peso: 2;
    - Valor: 3.

Selecionar os itens 2 e 3 garantem o maior valor total para a capacidade da mochila: 6. Não obstante o algoritmo guloso selecionará armazenar o item 1 pois este possui a melhor razão valor por peso dos três: $(1{,}\overline{6} > 1{,}5)$ e assim esgotará a capacidade da mochila com um valor total de 5.

A margem em questão para este algoritmo guloso é a garantia que este preenche *pelo menos* metade da capacidade da mochila com itens de maior valor possível. Mas este não garante que a mochila seja preenchida em sua totalidade com itens de maior valor possível, o que é a proposta do problema. ■

Exercício 2

O seguinte algoritmo calcula todas as combinações possíveis de n elementos e W capacidades para a mochila, fazendo uso de uma matriz $M_{(n+1)\times(W+1)}$ para armazenar resultados parciais (incluindo portanto as situações de uma mochila de dimensão 0 e uma combinação de 0 elementos). Tais resultados informam as combinações de maior valor ao acrecer capacidade e elementos, eventualmente alcançando a resposta final em que temos uma mochila de capacidade W e uma lista de n elementos disponíveis.

```
dynamicKnapsack (v,w,n,W)
    for j ← 1 to W + 1, do
        M[1,j] ← 0
    for i ← 2 to n + 1, do
        for j ← 1 to W + 1, do
            if (w[i - 1] > j - 1), then
                M[i,j] ← M[i - 1,j]
            else
                M[i,j] ← max(M[i - 1,j], v[i - 1] + M[i - 1,j - w[i - 1]])
    k ← 0
    for i ← n + 1 to 2, do
        if (M[i][j] = M[i - 1][j]), then
            continue
        j ← j - w[i - 1]
        k ← k + 1
        I[k] ← i - 1
    end
    return I
end
```

Um exemplo de saída de um programa escrito de acordo com tal algoritmo[4] para a mesma entrada[5] utilizada para o programa anterior é:

```
> time ./dynamic_knapsack.out < test.txt
Capacidade da mochila: 1350
Conteúdos:
Item: 7     Valor:   830 Peso:    56 Razão: 14.821428
Item: 24    Valor:   649 Peso:   219 Razão: 2.963470
Item: 27    Valor:   654 Peso:   112 Razão: 5.839286
Item: 31    Valor:   647 Peso:   206 Razão: 3.140777
Item: 36    Valor:   437 Peso:   122 Razão: 3.581967
Item: 37    Valor:   836 Peso:   154 Razão: 5.428571
Item: 43    Valor:   690 Peso:   208 Razão: 3.317308
Item: 50    Valor:   894 Peso:   264 Razão: 3.386364
Capacidade restante da mochila: 9
Valor total armazenado: 5637


_____________________________________________________
Executed in    3,29 millis    fish           external
   usr time    3,26 millis  526,00 micros    2,74 millis
   sys time    0,20 millis  204,00 micros    0,00 millis

>
```

Este algoritmo contempla todas as combinações de elementos sem por isso possuir tempo de execução exponencial $\theta(2^n)$ o qual observaríamos em soluções recursivas de força bruta. Ao armazenar resultados parciais, a solução apresenta um tempo de execução pseudo-polinomial proporcional às dimensões da matriz, $\theta((n+1)(W+1))$; assim como ocupa espaço de memória nesta mesma proporção. ■

Exercício 3

Thomas Cormen, et al.[6], descrevem uma variedade de métodos de análise amortizada passíveis de avaliar o tempo de execução do algoritmo proposto. Dentre os quais, estes exemplificam o uso do denominado *método contábil*. Este último consiste em

> [Atribuirmos] cobranças diferentes a operações diferentes, sendo que algumas operações são cobradas a mais ou a menos do que realmente custam. Denominamos **custo amortizado** o valor que cobramos por uma operação. Quando o custo amortizado de uma operação excede seu custo real, atribuímos a diferença a objetos específicos na estrutura de dados como **crédito**. Mais adiante, o crédito pode ajudar a pagar operações posteriores cujo custo amortizado é menor que seu custo real. Assim, podemos considerar o custo amortizado de uma operação como repartido entre seu custo real e o crédito que é depositado ou consumido.

Os custos amortizados das operações são escolhidos cuidadosamente, de forma que, no pior caso, "o custo amortizado total de uma sequência de operações dá um limite superior para o custo real total da sequência." Tal que

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

Sendo c_i o custo real da i-ésima operação, $\hat{c}_i$ o custo amortizado da mesma, e portanto a diferença entre estes valores o crédito armazenado (Ibid.).

Ao analisar a operação de incrementar um contador binário para um arranjo A de k bits inicialmente todos iguais a zero, é possível notar que toda operação de atribuir o valor 0 a um bit (linha 3), ou "desligá-lo" é contingente a uma operação anterior de atribuí-lo o valor 1 (linha 6), ou "ligá-lo". Por isso, Cormen, et al., designam um custo amortizado de 2 unidades para a operação de atribuir o valor 1 a um bit. Uma unidade é imediatamente consumida pela operação, enquanto a outra é armazenada enquanto crédito para seu eventual desligamento.

Assim, o custo de se desligar bits no interior do laço *while* é pago cada qual com o crédito acumulado por estes ao serem ligados. O procedimento *INCREMENTA* liga no máximo um bit por execução e, portanto, o custo amortizado de uma operação *INCREMENTA* é de no máximo 2 unidades. A quantidade de 1s no contador nunca se torna negativa e, portanto, a quantia de crédito permanece não negativa o tempo todo. Assim, para n operações *INCREMENTA*, o custo amortizado total é assintoticamente equivalente a $O(n)$, o que, pela fórmula anteriormente descrita, trata-se do pior caso do custo real total. ■

---

1. Número USP: 12543033; Turma: 04. ↩
2. DANTZIG, G. B. Discrete-Variable Extremum Problems. **Operations Research**, v. 5, n. 2, p. 266–288, 1 abr. 1957. ↩

3. BARRETO, G. **greedy_knapsack.c**. Disponível em: https://git.disroot.org/SI/semestre_2/src/branch/master/Introdu%c3%a7%c3%a3o%20%c3%a0%20An%c3%a1lise%20de%20Algoritmos/Lista%202/greedy_knapsack.c. Acesso em: 14 dec. 2021. ↩
4. BARRETO, G. **dynamic_knapsack.c**. Disponível em: https://git.disroot.org/SI/semestre_2/src/branch/master/Introdu%c3%a7%c3%a3o%20%c3%a0%20An%c3%a1lise%20de%20Algoritmos/Lista%202/dynamic_kanpsack.c. Acesso em: 14 dec. 2021. ↩
5. BARRETO, G. **test.txt**. Disponível em: https://git.disroot.org/SI/semestre_2/src/branch/master/Introdu%c3%a7%c3%a3o%20%c3%a0%20An%c3%a1lise%20de%20Algoritmos/Lista%202/test.txt. Acesso em: 14 dec. 2021. ↩
6. CORMEN, T. et al. **Algoritmos: teoria e prática**. Tradução: Arlete Marques. 3. ed. Rio de Janeiro: Elsevier, 2012. ↩