Projeto e Análise de Algoritmos

Luigi Domenico Cecchini Soares - 553229
luigi.soares@sga.pucminas.br

28 de maio de 2018

1 Problema

Alfred deseja planejar o que cozinhar nos próximos dias. Ele pode cozinhar vários pratos. Para cada prato, o custo dos ingredientes e o lucro final é conhecido. Se um prato é cozinhado duas vezes seguidas, o valor do lucro na segunda vez é 50 porcento do lucro na primeira vez. Se ele é preparado uma terceira vez ou mais em seguida, o valor do lucro é zero. Por exemplo, se um prato, que gera um lucro v, é cozinhado três vezes em seguida, o lucro final desses três dias é 1.5v. Ajude-o a construir o cardápio que maximiza o lucro sob a restrição de que seu orçamento não seja excedido.

2 Abordagem Gulosa

Um algoritmo guloso sempre realiza a escolha que apresenta a melhor solução para o subproblema atual. Em outras palavras, tal algoritmo considera a solução ótima local, levando em conta que isto levará a uma solução ótima global. Pensando nisso, é possível modelar o problema descrito anteriormente de forma que sempre seja realizada uma escolha que pareça ótima para o momento.

Para se desenvolver um algoritmo desse tipo, alguns passos devem ser seguidos. O primeiro passo é transformar o problema a ser otimizado em um no qual, realizando uma escolha local, é encontrado um novo subproblema para se resolver. Em outras palavras, deve-se encontrar uma subsestrutura ótima, para que o algoritmo possa funcionar em termos dela.

2.1 Algoritmo

Para o problema em questão, a decisão está relacionada ao prato a ser cozinhado em um dia i, de modo que ao final de k dias obtenha-se o maior lucro possível. Sendo assim, a subestrutura ótima diz respeito ao conjunto de pratos Q que oferecem o melhor lucro possível em um dia i. Tomando como base a ideia utilizada na abordagem gulosa do problema da mochila, que leva em conta a relação entre valor e peso de determinado item, é possível definir o critério de seleção como sendo o prato de melhor custo-benefício ($custo/lucro$) para um dia i.

Entretanto, o problema descrito anteriormente introduz uma variável a mais, que diz respeito a repetição de um prato em dias seguidos. Para lidar com essa nova variável, basta construir uma lista de pratos que leve em consideração três estados de um mesmo prato: sendo cozinhado pela primeira, segunda e terceira vez. Cada estado apresenta um custo-benefício de acordo com seu lucro. Assim, é possível construir uma lista ordenada, de forma que o primeiro prato da lista sempre seja o que oferece o melhor custo-benefício.

O terceiro estado é um caso especial, uma vez que seu lucro é zero. Por isso, seu custo-benefício também será definido como zero. Porém, tais pratos devem aparecer no final da lista. Dessa forma, a lista será ordenada parcialmente (em relação aos estados um e dois) e terá os pratos do terceiro estado inseridos ao final.

Levando em conta todos os detalhes, definidos acima, para modelagem desse problema, é possível chegar ao seguinte algoritmo:

Algorithm 1 Algoritmo Guloso

```
1:  function MontarMenu(P, K, M)
2:      menu := {}
3:      for i := 0 ate K − 1 do
4:          tome p como sendo o primeiro prato de P
5:          while P nao esta vazio e custo de p maior que M do
6:              descarte p
7:          if P nao esta vazio then
8:              menu := menu ∪ p
9:
10:     if |menu| < K − 1 then
11:         return {}
12:     else
13:         return menu
```

O algoritmo acima apresenta alguns casos a serem analisados. O primeiro diz respeito a situação em que, em algum dos K dias, todos os pratos serão descartados. Se isso acontecer, o custo do algoritmo, em relação aos testes condicionais realizados (que representam o maior custo dentre as instruções existentes), será de $(K-1)*2+1*(|P|+1)$, uma vez que em um dos dias serão realizados $|P|$ testes, descartando todos os pratos, e mais um teste para verificar se P está vazio. Sendo assim, o custo assintótico será $\mathcal{O}(K+|P|)$.

No segundo caso, nenhum prato será descartado em nenhum dos dias. Dessa forma, em todos os dias serão realizados 2 testes condicionais. Assim, o custo do algoritmo será de $2*K$, levando a um custo assintótico linear $\mathcal{O}(K)$.

Por fim, um último caso a ser considerado é a situação em que alguns pratos serão descartados em todos os dias. Assim, tomando o conjunto $Q_i \subset P$ como sendo o conjunto de pratos a serem descartados no dia i, para que pelo menos um prato seja descartado em todos os dias, temos que:

$$\sum_{i=1}^{K} |Q_i| \leq |P|$$

Na pior das hipóteses, ao fim do último dia todos os pratos terão sidos descartados. Sendo

assim, esse último caso nos leva ao mesmo custo assintótico do primeiro: $\mathcal{O}(K+|P|)$. Conclui-se, portanto, que os custos desse algoritmo serão $\mathcal{O}(K)$ para o melhor caso e $\mathcal{O}(K+|P|)$ no pior caso.

2.2 Solução ótima

A próxima etapa do desenvolvimento de um algoritmo guloso está relacionada a provar que sempre existirá uma solução ótima local, que, por sua vez, levará a uma solução ótima global. Porém, no caso do algoritmo acima, não é possível encontrá-la. Para provar, basta considerar um conjunto de pratos P em que um prato $p \in P$ apresenta um custo c elevado, próximo de um orçamento máximo M, e um lucro também elevado.

Tome P como sendo um produto cartesiano CXL entre os custos e lucros dos pratos. Considere que os elementos de P sejam $p_1 = (2, 5)$, $p_2 = (9, 100)$ e $p_3 = (2, 4)$. Considere ainda que serão utilizados 3 dias e que o orçamento é de R$ 10,00.

A melhor solução para esse caso seria cozinhar os pratos p_1, p_3 e p_1, respectivamente. Esta sequência resultaria em um lucro de R$ 14,00. Porém, o algoritmo guloso descrito acima daria como solução um cojunto vazio. Isso ocorre, porque o algoritmo considera apenas o custo-benefício do prato p_2, que é muito superior ao dos outros pratos, embora seu custo seja bastante elevado. Assim, ao se escolher o prato p_2, não restará orçamento para os próximos dias.

3 Programação Dinâmica

Essa técnica, assim como o método de divisão e conquista, consiste em resolver problemas combinando soluções de subproblemas. A aplicação desse método se dá quando um subproblema compartilha outros subproblemas. Nesse contexto, um algoritmo de divisão e conquista resultaria em um esforço maior que o necessário, resolvendo repetidamente os subproblemas em comum.

Em contrapartida, a abordagem de uma programação dinâmica consiste em resolver esses subprolemas comuns uma única vez e salvá-los em uma tabela. Dessa forma, evita-se processar novamente um subproblema já resolvido anteriormente. Esse método é tipicamente aplicado em problemas de otimização, como o descrito na seção 1.

Assim como na abordagem gulosa, um dos pré-requisitos para se desenvolver um algoritmo através do paradigma de programação dinâmica é existir uma subestrutura ótima. Um problema apresenta subestrutura ótima se uma solução ótima para ele contém uma solução ótima para seus subproblemas. Para o problema em questão, a subestrutura ótima diz respeito a um conjunto de pratos $Q \subseteq P$, onde P é o conjunto total de pratos, que dão o melhor lucro possível em um conjunto de dias X, $|X| <= K$.

3.1 Algoritmo

O algoritmo para resolver o problema descrito na seção 1 consiste em processar subproblemas, comparando-os com subproblemas anteriores, de forma que se guarde o melhor lucro possível para cada situação. Um subproblema consiste em um dia i, associado a um prato p

e a um orçamento m. Assim, deve-se calcular o maior lucro possível para esse subproblema S_i, p, m, somando-se o lucro de p ao maior lucro encontrado no subproblema S_i-1, p_1^n, $m-c$. Com isso, é possível desenvolver o seguinte algoritmo:

Algorithm 2 Programação Dinâmica

```
1: function MontarMenu(P, K, M)
2:     menu := {}
3:     dp := matriz[K][|P|][M]
4:     caminho := matriz[K][|P|][M]
5:     inicializar posicoes de dp e caminho com -1
6:     for i := 0 ate K − 1 do
7:         for j := 0 ate |P| − 1 do
8:             for m := 0 ate M do
9:                 if m − custo de P_j ≥ 0 then
10:                    if i = 0 then
11:                        dp[i][j][m] := lucro de P_0
12:                    else
13:                        lucro_max := −1
14:                        prato_ant := −1
15:                        for p := 0 ate |P| − 1 do
16:                            if orcamento restante for suficiente para o resto dos dias then
17:                                lucro_parcial := dp[i − 1][p][m − custo de j]
18:                                if se P_j estiver sendo repetido pela 1ª vez then
19:                                    reduzir lucro de P_j pela metade
20:                                else
21:                                    if se P_j estiver sendo repetido pela 2ª vez then
22:                                        reduzir lucro de P_j para 0
23:                                atualizar lucro_max considerando lucro_parcial e priorizando 1º prato nao repetido e depois prato com menor custo
24:                                atualizar prato_ant de acordo com atualizacao do lucro_max
25:                        dp[i][j][m] := lucro_max
26:                        caminho[i][j][m] := prato_ant
27:    procurar prato p com maior no lucro no dia K − 1 com orcamento M
28:    preencher menu a partir do caminho do prato p
29:    return menu
```

O algoritmo acima apresenta complexidade igual no pior e no melhor caso, uma vez que depende da quantidade de dias, número de pratos e orçamento máximo. Vale ressaltar que o número de pratos apresenta influencia maior, por aparecer em dois momentos (subproblema atual e busca por melhor lucro de subproblema anterior). Sendo assim, é possível concluir que o custo assintótico para ele será $\Theta(K * |P| * M|P|)$.

Por fim, é importante considerar os algoritmos clássicos utilizados como base para o desenvolvimento da solução do problema citado anteriormente. O algoritmo que deu a maior base para o anterior é o da mochila, que consiste em otimizar o lucro de acordo com itens a

serem carregados. Além deste, foi considerado, para a montagem do caminho dos pratos, o algoritmo de busca da maior sequência comum em uma string.

4 Testes

4.1 Especificações

- **Processador:** Intel Core i5 - 7200U - 2.50Ghz, 2.71 Ghz
- **Memória Ram:** 8GB
- **Sistema Operacional** Windows 10 Home Single Language - 64bits

4.2 Abordagem gulosa x programação dinâmica

A comparação entre as duas técnicas de projeto de algoritmos foi realizada em 3 etapas. A primeira, considerando uma variação na quantidade de dias e fixando o número de pratos e o orçamento máximo. A segunda, nos mesmos moldes da primeira, mas variando o número de pratos e fixando a quantidade de dias. Por fim, a terceira, nos mesmos moldes das duas anteriores, mas variando o orçamento máximo.

Nos 3 testes, é possível perceber que o tempo de execução do algoritmo guloso se apresenta de forma linear, enquanto o tempo de execução do algoritmo referente a programação dinâmica tem um crescimento elevado. Além disso, em relação ao consumo de memória, é possível constatar que o dinâmico também apresenta uma média de consumo maior, uma vez que é necessário utilizar matrizes de 3 dimensões.

- Primeiro teste:
  Nesse teste, a quantidade de dias varia de 1 a 21. O restante da entrada consiste em 50 pratos e orçamento máximo de 100.

Figura 1: Tempo de execução

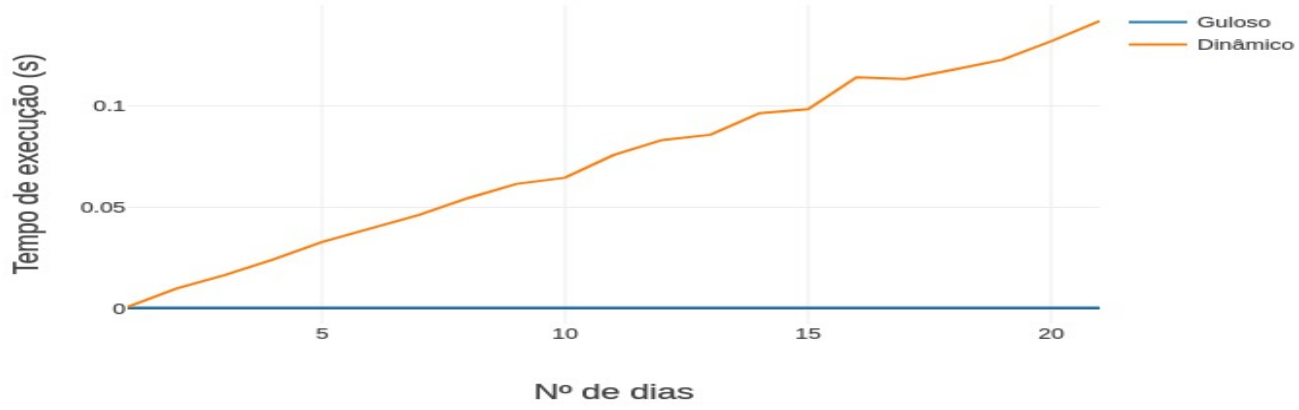
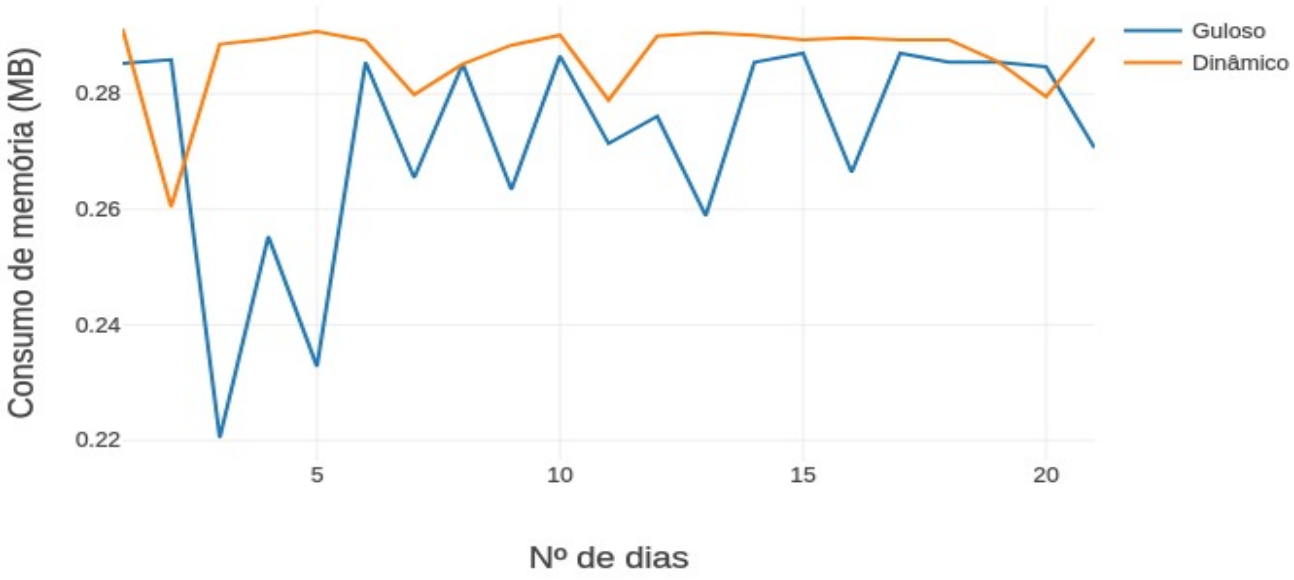


Figura 2: Consumo de memória



- Segundo teste:
  Nesse teste, o número de pratos varia entre 1 e 50. O restante da entrada consiste em 21 dias e orçamento máximo de 100.

Figura 3: Tempo de execução

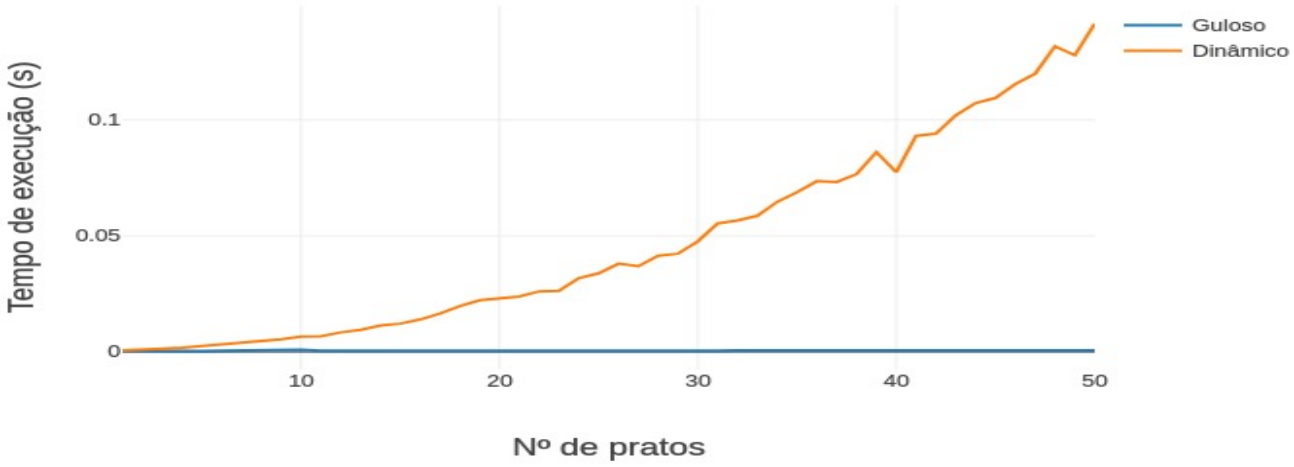
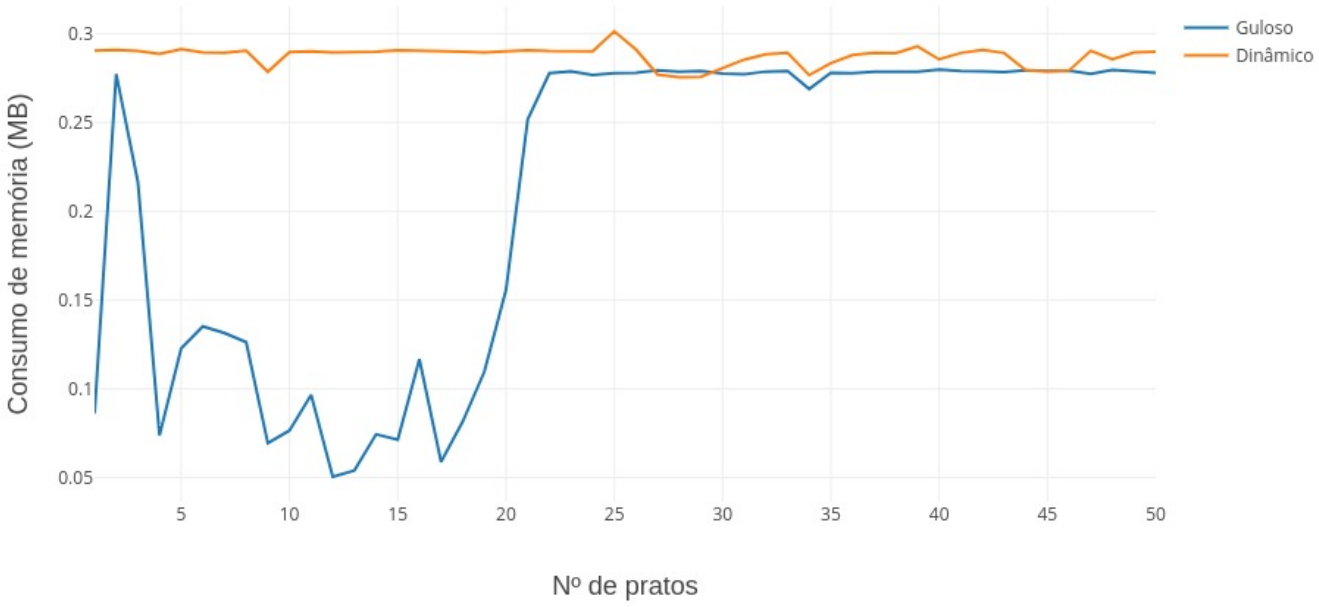


Figura 4: Consumo de memória



- Terceiro teste:
  Nesse teste, o orçamento varia entre 0 e 100. O restante da entrada em 21 dias e 50 pratos.

Figura 5: Tempo de execução

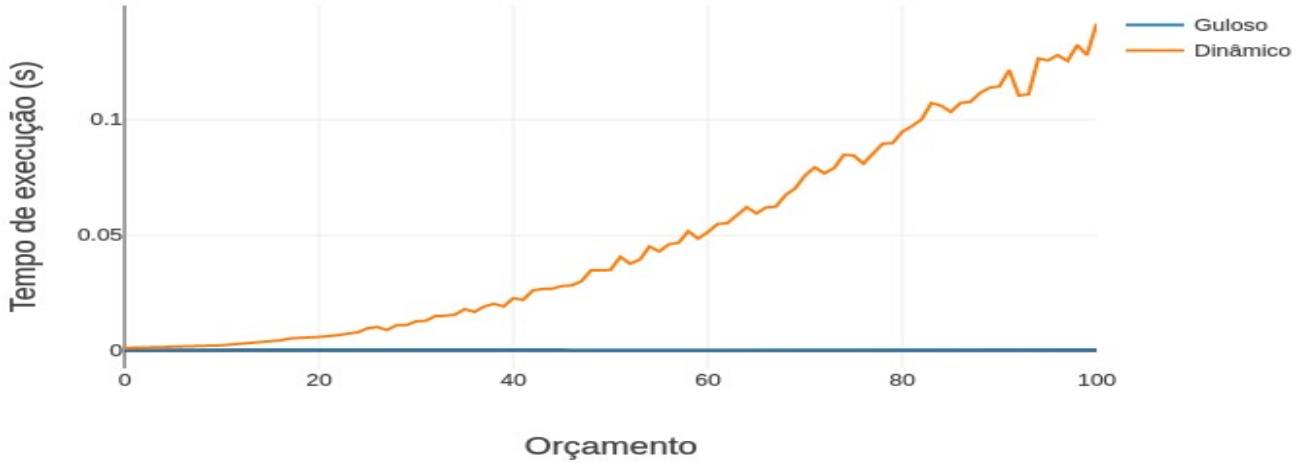
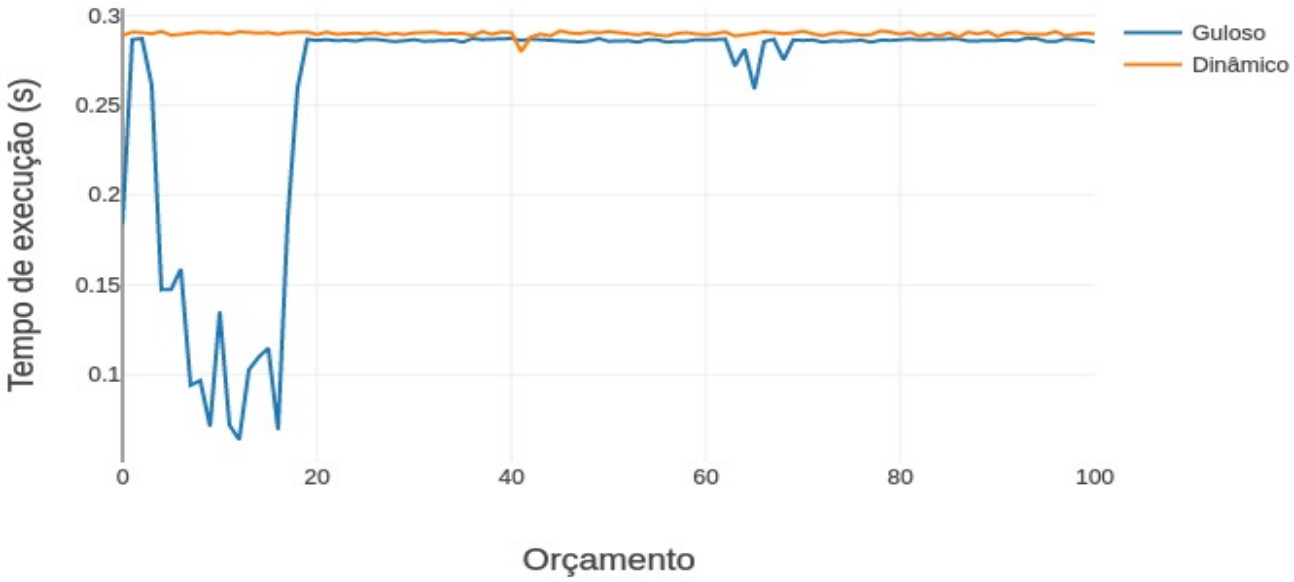


Figura 6: Consumo de memória



5 Bibliografia

CUNHA, Felipe. Projeto e análise de algoritmos. 01 feb. 2018, 14 jun. 2018. Notas de Aula.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.