Análise de algoritmos

Algoritmos gulosos

Conteúdo I

Introdução: Algoritmos gulosos I

- Similar a programação dinâmica
- Usados para problemas de otimização
- Para muitos problemas de otimização é um exagero usar programação dinâmica: algoritmos mais simples e mais eficientes darão conta da mesma tarefa
- Um **algoritmo guloso** sempre faz a escolha que parece ser a *melhor no momento*
- Faz uma *escolha ótima local* na esperança de que essa escolha leve a uma *solução ótima global*
- Nem sempre produzem soluções ótimas,
  mas as produzem para muitos problemas

Introdução: Algoritmos gulosos II

- Método guloso funciona bem para uma ampla faixa de problemas, incluindo:
  - Algoritmos de árvore geradora mínima
  - Algoritmo de Dijkstra para para caminhos mais curtos que partem de uma única origem

Introdução: Um problema de seleção de atividades I

- n **atividades** requerem o uso *exclusivo* de um recurso comum
  - Exemplo: O uso de uma sala de aula
- Conjunto de atividades $S = \{a_1, ..., a_n\}$ que desejam usar um recurso
- a_i precisa usar o recurso durante o período $[s_i, f_i)$
  - s_i: tempo de início
  - f_i: tempo de término
  - Onde $0 \leq s_i < f_i < \infty$
  - a_i e a_j são compatíveis se $s_i \geq f_j$ ou $s_j \geq f_i$
- **Objetivo**: Selecionar o maior conjunto possível de atividades não sobrepostas (**mutuamente compatíveis**)
- Supomos que as atividades estão ordenadas pelo tempo de término:

$$f_1 \leq f_2 \leq f_3 \leq ... \leq f_{n-1} \leq f_n$$

- **Nota**: Podem haver outros objetivos
  - Escalonar sala pelo maior período de tempo
  - Maximizar as taxas de renda de aluguel

Introdução: Um problema de seleção de atividades II

- Exemplo:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- (...)

Introdução: Um problema de seleção de atividades III

- $\{a_3, a_9, a_{11}\}$
- $\{a_1, a_4, a_8, a_{11}\}$
- $\{a_2, a_4, a_9, a_{11}\}$

A subestrutura ótima do problema de seleção de atividades I

- S_{ij}: conjunto de atividades que começam após o término da atividade a_i e terminam antes do início da atividade a_j
- Suponha que queremos determinar um conjunto máximo de atividades mutuamente compatíveis em S_{ij} e que tal subconjunto máximo é A_{ij}, que inclui algum atividade a_k
- Incluindo a_k em uma solução ótima, ficamos com dois subproblemas em mãos:
  - S_{ik} (começam após o término de a_i e terminam antes do início de a_k)
  - S_{kj} (começam após o término de a_k e terminam antes do início de a_j)
- Sejam $A_{ik} = A_{ij} \cap S_{ik}$ e $A_{kj} = A_{ij} \cap S_{kj}$
  - A_{ik}: contém atividade em A_{ij} que terminam antes do início de a_k

A subestrutura ótima do problema de seleção de atividades II

- A_{kj}: contém atividade em A_{ij} que começam após o término de a_k

- Assim: $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$
- E portanto: o conj. de tam. máx. A_{ij} de ativ. mutuamente exclusivas em S_{ij} consiste em $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$
- Ótima, pelo argumento de recortar e colar (no livro)
- Isso sugere que poderíamos usar PD
- Denotando tamanho de uma sol. ótima para o conjunto S_{ij} por $c[i, j]$ teremos

$$c[i, j] = c[i, k] + c[k, j] + 1$$

- Teremos de examinar todas as atividades em S_{ij}:

A subestrutura ótima do problema de seleção de atividades III

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset\ , \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset\ . \end{cases}$$

- Poderíamos fazer um algoritmo recursivo e memoizá-lo ou trabalhar de baixo para cima
- Porém, estaríamos ignorando uma outra característica importante do problema que podemos usar e seria muito vantajoso

Fazendo a escolha gulosa I

- E se pudéssemos escolher uma atividade para acrescentar à nossa solução ótima sem ter de resolver primeiro todos os subproblemas?
  - Nos pouparia de ter de considerar todas as escolhas na última recorrência
  - No problema de seleção de atividades precisamos considerar somente a **escolha gulosa**

Fazendo a escolha gulosa II

- Intuição:
  - Deveríamos escolher uma atividade que deixa o recurso disponível para o maior número possível de outras atividades
  - Agora, entre as atividades que acabamos escolhendo, uma deve ser a primeira a terminar
  - Portanto, escolher a atividade em S que tenha o tempo de término mais cedo, já que isso deixaria o recurso disponível para atividades posteriores
    - Atividades ordenadas: a escolha gulosa é a atividade a_1
  - Fazendo a escolha gulosa, restará somente um subproblema: determinar atividades que começam após o término e a_1. Por que não temos de considerar atividades que terminam antes de a_1 começar?
    - $s_1 < f_1$, e f_1 é o tempo mais cedo de término de qualquer atividade
    - Portanto, nenhuma atividade pode ter o tempo de término menor ou igual a s_1
    - Todas as que são compatíveis com a_1 devem começar depois que a_1 terminar

Fazendo a escolha gulosa III

- O problema exibe subestrutura ótima
  - Seja $S_k = \{a_i \in S : s_i \geq f_k\}$ com aquelas que começam após o término de a_k
  - Escolhendo a_1, S_1 permanecerá como um único problema a resolver
  - S.o. diz: Se a_1 estiver na solução ótima, uma sol. ótima para o problema original consistirá de a_1 e todas as atividades em uma solução ótima para o subproblema S_1

Fazendo a escolha gulosa IV

- Nossa intuição está correta? A escolha gulosa – na qual escolhemos a primeira atividade a terminar – é sempre parte de alguma solução ótima?

Fazendo a escolha gulosa V

Teorema 16.1

Considere um subproblema qualquer não vazio S_k, e seja a_m uma atividade em S_k com o tempo de término mais cedo. Então, a_m estará incluída em algum subconjunto de tamanho máximo de atividades mutuamente compatíveis de S_k.

Prova: Seja A_k um subconjunto de tamanho máximo de atividade mutuamente compatíveis em S_k, e seja a_j a atividade em A_k que tem o tempo de término mais cedo. Se $a_j = a_m$, terminamos aqui, visto que já mostramos que a_m está em algum subconjunto de tamanho máximo de atividades mutuamente compatíveis de S_k. Se $a_j \neq a_m$ considere o conjunto $A'_k = A_k - \{a_j\} \cup \{a_m\}$, que é A_k substituindo a_j por a_m. As atividades em A'_k são disjuntas, o que decorre porque as atividades em A_k são disjuntas, a_j é a primeira atividade a terminar em A_k e $f_m \leq f_j$. Visto que $|A'_k| = |A_k|$, concluímos que A'_k é um subconjunto de tamanho máximo de atividades mutuamente compatíveis de S_k e ele inclui a_m.

Fazendo a escolha gulosa VI

- # de subproblemas na solução ótima: antes $= 2$; depois $= 1$
- # de escolhas a considerar: antes $= j - i - 1$; depois $= 1$

- Podemos escolher repetidamente a atividade que termina primeiro
- Manter somente as atividades compatíveis com ela
- Repetir o processo até não restar nenhuma atividade

- Os tempos de término das atividades que escolhermos deve crescer estritamente
- Podemos considerar cada atividade apenas uma vez no total, em ordem monotonicamente crescente de tempos de término

Fazendo a escolha gulosa VII

- Algoritmo para seleção de atividades não precisa funcionar de baixo para cima, como PD baseado em tabela
- Pode ser de cima para baixo:
  - Escolhendo uma atividade para colocar na solução ótima; e
  - Resolvendo o problema de escolher atividade entre as que são compatíveis com as já escolhidas
- Projeto normalmente usado em algoritmo gulosos

Um algoritmo guloso recursivo I

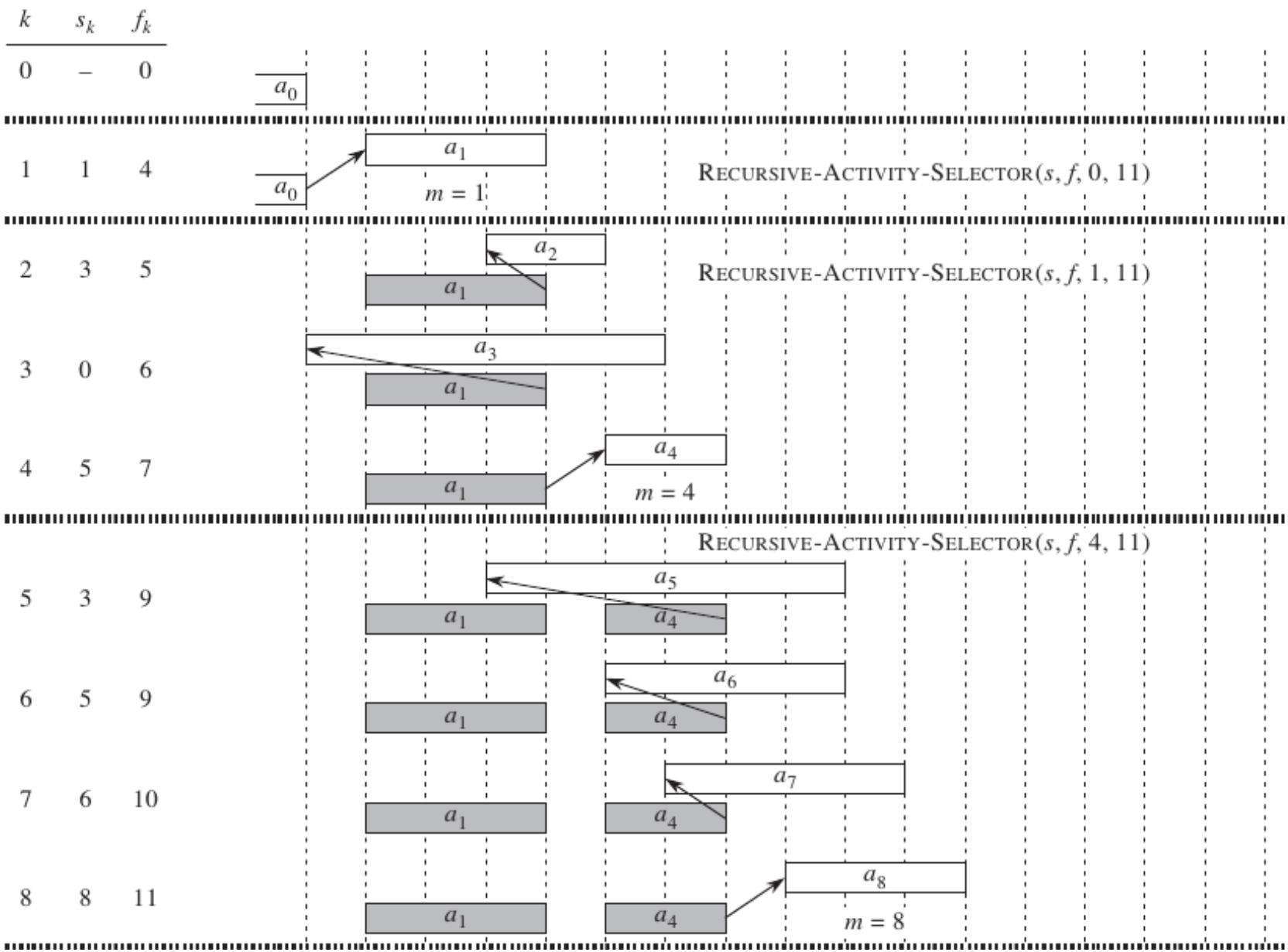
- Procedimento recursivo direto
- Como parâmetros os tempos de início e término por meio dos arranjos s e f,
- ...índice k define o subproblema S_k a ser resolvido e
- ...o tamanho n do problema original
- Retorna um conjunto de tamanho máximo de atividade mutuamente compatíveis em S_k
- Consideramos que as entradas já estão ordenadas por tempo de término
- Se não, podemos fazer em $O(n \lg n)$
- Atividade fictícia a_0, com $f_0 = 0$. Assim, subproblema S_0 é o conjunto inteiro de atividades S
- Chamada inicial:
  `Recursive-Activity-Selector(s, f, 0, n)`

Um algoritmo guloso recursivo II

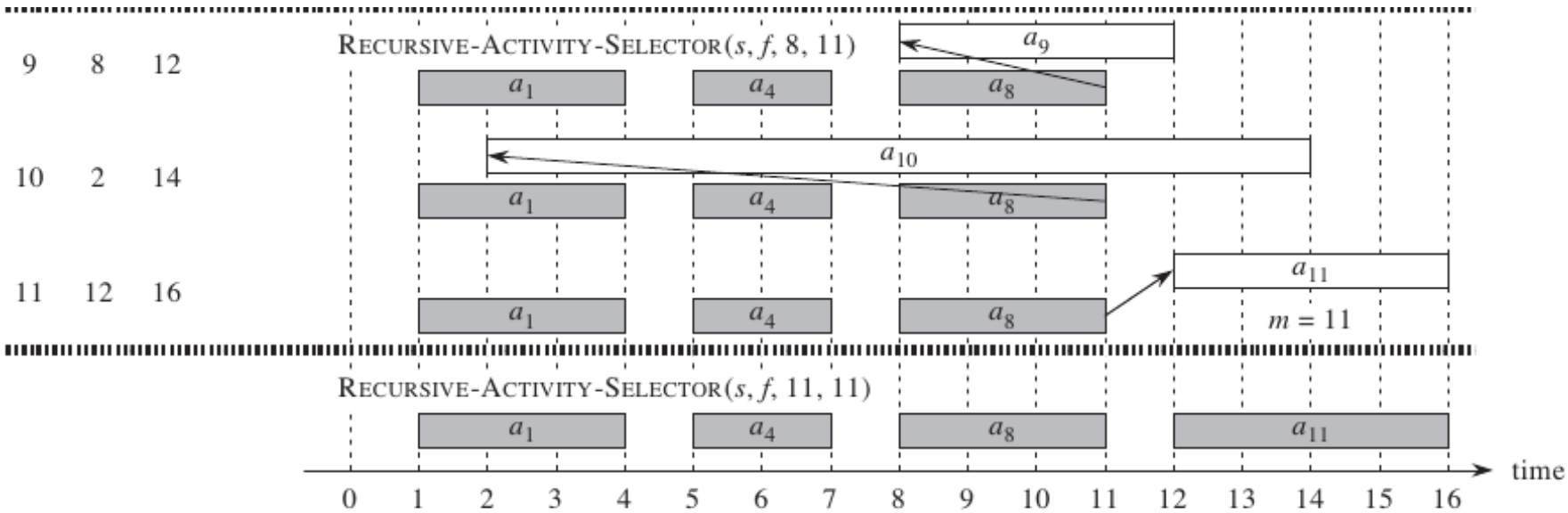
```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

- Considerando entrada ordenada, o tempo é $\Theta(n)$
- Considerando todas as chamadas recursivas, cada atividade é examinada exatamente uma vez no teste do laço `while` da linha 2
- Em particular, a atividade a_i é examinada na última chamada feita na qual $k < i$

Um algoritmo guloso recursivo III



Um algoritmo guloso recursivo I



Um algoritmo guloso iterativo I

- O procedimento recursivo é quase um "recurso de cauda"
- Converter para um iterativo é uma tarefa direta
- Também considera entradas ordenadas
- Retorna atividades selecionadas no conjunto A

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

Um algoritmo guloso iterativo II

- k indexa a adição mais recente a A (corresp. à a_k)
- Em ordem crescente: f_k é sempre o tempo máximo de qualquer atividade em A
- Primeiro seleciona a_1
- Laço `for` encontra a atividade que termina mais cedo em S_k: considera cada atividade a_m por vez, e a acrescenta a A se for compatível com todas as anteriores
- Tempo: $\Theta(n)$

Exercícios I

- 2^a ed.: 16.1-1 a 16.1-4.
- 3^a ed.: 16.1-1 a 16.1-5.
  - 16.1-3 da 3^a ed. é igual ao 16.1-4 da 2^a ed.
  - 16.1-4 da 3^a ed. é igual ao 16.1-3 da 2^a ed.
  - 16.1-5 é novo na 3^a ed.

Introdução: Elementos da estratégia gulosa I

- Algoritmo guloso faz uma sequência de escolhas
- Para cada ponto de decisão, escolhe a opção que parece melhor no momento
- O processo visto, foi um pouco mais complicado que o normal:
  1. Determinar a subestrutura ótima do problema
  2. Desenvolver uma solução recursiva
  3. Provar que, se fizermos a escolha gulosa, restará somente um subproblema
  4. Provar que, é sempre seguro fazer a escolha gulosa
  5. Desenvolver um algoritmo recursivo que implemente a estratégia gulosa
  6. Converter o algoritmo recursivo em um algoritmo iterativo

Introdução: Elementos da estratégia gulosa II

- Vimos os fundamentos de PD de um algoritmo guloso
  - No problema de seleção de atividades: primeiro definimos os subproblemas S_{ij}, no qual i e j variavam
  - Constatamos que, se sempre fizéssemos a escolha gulosa, poderíamos restringir os subproblemas à forma S_k
- Alternativamente, poderíamos ter formado nossa subestrutura ótima tendo em mente:
  - Uma escolha gulosa
  - De modo que deixasse apenas um subproblema
- Poderíamos ter começado:
  - Descartando o segundo índice e definindo subproblemas da forma S_k
  - Provado que uma escolha gulosa (primeira atividade a_m a terminar em S_k), combinada com uma solução ótima para S_m, produz uma solução ótima para S_k

Introdução: Elementos da estratégia gulosa III

- Passos para projeto de algoritmos gulosos:
  1. Expressar o problema de otimização como um problema no qual fazemos uma escolha e ficamos com um único subproblema para resolver
  2. Provar que sempre existe uma solução ótima para o problema original que usa a escolha gulosa, de modo que a escolha gulosa é sempre segura
  3. Demonstrar subestrutura ótima mostrando que, tendo feito a escolha gulosa, o que resta é um subproblema com a seguinte propriedade: se combinarmos uma solução ótima para o subproblema com a escolha gulosa que fizemos, chegamos a uma solução ótima para o problema original
- Nenhum método geral para dizer se um algoritmo guloso é ótimo, mas dois componentes fundamentais são:
  - Propriedade de escolha gulosa
  - Subestrutura ótima

Propriedade de escolha gulosa I

- **Propriedade da escolha gulosa**: podemos montar uma solução globalmente ótima fazendo escolhas (gulosas) locais ótimas
- Programação dinâmica
  - Faz uma escolha em cada passo
  - Escolha depende de sabermos soluções ótimas para subproblemas
    - Resolver *primeiro* os subproblemas
  - Resolve de *baixo para cima*
- Algoritmos gulosos
  - Faz uma escolha a cada passo
  - Faz a escolha *antes* de resolver os subproblemas
  - Resolve de *cima para baixo*

Propriedade de escolha gulosa II

- Tipicamente, temos de mostrar a propriedade da escolha gulosa, como fizemos para a seleção de atividades:
  - Examinar uma solução globalmente ótima para algum subproblema
  - Mostrar como modificar a solução para usar a escolha gulosa no lugar de alguma outra escolha
  - Resultando em um subproblema semelhante, porém menor
- Podemos obter ganho de eficiência com a propriedade da escolha gulosa:
  - Pré-processar entrada para colocá-la na ordem gulosa
  - Ou, caso os dados sejam dinâmicos, usar uma fila de prioridades

Subestrutura ótima I

- Um problema exibe **subestrutura ótima** se uma solução ótima para o problema contiver soluções ótimas para subproblemas
- Essa propriedade é um componente fundamental para avaliar a aplicabilidade de:
  - PD
  - Algoritmos gulosos
- Na subestrutura ótima apresentada, vimos:
  - Se uma solução ótima S_{ij} incluir uma atividade a_k...
  - ...então, ela também deve conter soluções ótimas para S_{ik} e S_{kj}
  - Dada essa subestrutura ótima, demonstramos que, se soubéssemos qual usar como a_k...
  - ...poderemos construir uma sol. ótima para S_{ij},
  - selecionando a_k e as atividades em sols. ótimas para subproblemas S_{ik} e S_{kj}
  - Com base nisto, criamos a recorrência

Subestrutura ótima II

- Normalmente, usamos uma abordagem mais direta em relação à subestrutura ótima quanto aplicamos algoritmos gulosos
- Supor que chegamos a um subproblema por termos feito a escolha gulosa no problema original
- Basta demostrar que uma solução ótima para o subproblema, combinada com a escolha gulosa já feita, produz um solução ótima para o problema original

Estratégia gulosa *versus* programação dinâmica I

- O problema da mochila é um bom exemplo da diferença
- Ler item **Estratégia gulosa versus programação dinâmica** da seção 16.2 **Elementos da estratégia gulosa**

Exercícios I

- 2^a ed.: 16.2-1 a 16.2-7.
- 3^a ed.: 16.2-1 a 16.2-7.
  - O 16.2-4 da 3^a ed. é diferente do 16.2-4 da 2^a ed., mas eles possuem ideia semelhante.

Referências

- Thomas H. Cormen et al. Introdução a Algoritmos. 3ª edição em português. Capítulo 16.
- Thomas H. Cormen et al. Introdução a Algoritmos. 3ª edição em inglês. Capítulo 16.
- Thomas H. Cormen et al. Introdução a Algoritmos. 2ª edição em português. Capítulo 16.