Análise e Técnicas de Algoritmos

Jorge Figueiredo

Algoritmos Gulosos (Greedy)

Agenda

- Problemas de otimização
- Conceitos Básicos
- O Problema da Mochila Fracionária
- *Template* Genérico
- Exemplos: Código de Huffman

Problemas de Otimização

- Muitos problemas consideram o conceito de maximizar ou minimizar um determinado valor:
  - Como uma empresa de mudança deve alocar os móveis em um caminhão baú?
  - Como uma companhia telefônica deve rotear chamadas de modo a fazer um melhor uso de suas linhas e conexões?
  - Como alocar as disciplinas para melhor utilizar as salas do REENGE?
- Características:
  - Problemas que podem apresentar diversas soluções.
  - Solução formada por uma seqüência de decisões.
  - Um valor ou custo é associado a cada solução.
  - Acha solução com custo ótimo.

Exemplo 1: Cálculo do Trôco

Descrição: Seja $E= \{e_1, e_2, ..., e_n\}$, $e_1 > e_2 > ... > e_n$, um conjunto de n denominações de moedas (ou cédula), e M um valor positivo que representa o trôco.

Problema: Fornecer o montante M com o menor número de moedas.

Seqüência de decisões: Escolher r_1, depois r_2, ...

Exemplo 2: Problema da Mochila

Descrição: Temos n objetos com pesos $p_1, p_2, ..., p_n$ e lucros $l_1, l_2, ..., l_n$. Temos ainda uma mochila de capacidade M. Se uma fração x_i ($0 \le x_i \le 1$) do objeto i for colocada na mochila, resulta em um lucro $x_i \times l_i$.

Problema: Maximizar o lucro que pode ser levado na mochila.

Seqüência de decisões: Escolher primeiro objeto, escolher segundo objeto, ...

Exemplo 3: Escalonamento de Tarefas

Descrição: Seja E um conjunto de n tarefas. Associamos a cada tarefa um tempo de execução. Fazer o escalonamento das tarefas.

Problema: Minimizar o tempo médio de finalização das tarefas.

Seqüência de decisões: Escolher primeira tarefa, escolher segunda tarefa, ...

Exemplo 4: Caixeiro Viajante

Descrição: Seja *G=(V, E)* um grafo dirigido ponderado. Seja *n* o número de vértices e v_0 o vértice de origem.

Problema: Achar uma turnê de custo mínimo, começando em v_0.

Seqüência de decisões: A partir de v_0 qual é o primeiro vértice do ciclo, o segundo vértice, ...

Método Guloso

- Decisões tomadas de forma isolada em cada passo.
- Estratégia de pegar o melhor no momento:
  - Solução ótima local.
- Quando o algoritmo termina, espera-se que a solução ótima tenha sido encontrada:
  - Alguns problemas são resolvidos de forma ótima.
  - Outros apresentam soluções bem *pobres*.

Exemplo 1: Cálculo do Trôco

Descrição: Seja *E= {100, 50, 10, 5, 1}* , e *M* um valor positivo que representa o trôco.

Estratégia Greedy: No passo *i*, escolher $r_i = j$, tal que $e_j \leq M$ e $e_{j-1} > M$ e subtrair e_j de *M* para o próximo passo.

- É possível provar que a estratégia gulosa funciona neste caso.
- A mesma estratégia funciona para *E= {300, 250, 100, 1}* ?

Exemplo 2: Alocação de tarefas

Descrição: Seja *T= {(T1, 15), (T2, 8), (T3, 3), (T4, 10)}* . Considerar um único processador e alocação não preemptiva. Qual a melhor forma de alocar essas tarefas para minimizar o tempo médio de execução.

Estratégia Greedy 1: ordem de chegada.

Estratégia Greedy 2: ordem crescente do tempo de execução.

- É possível provar que a estratégia 2 sempre apresenta a solução ótima?

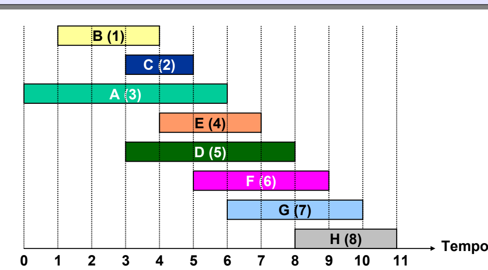
Exemplo 3: Seleção de atividades

- O problema consiste em escolher entre atividades que competem por uso exclusivo de um recurso em comum.
  - Por exemplo, o uso de uma sala de aula.
- Conjunto de atividades $S=\{a_1, \ldots, a_n\}$.
  - a_i necessita do recurso durante o período $[s_i, f_i)$, em que s_i = *tempo inicial* e f_i = *tempo final.*

Objetivo: selecionar o maior número possível de atividades compatíveis (***sem overlap de períodos***).

Exemplo 3: Seleção de atividades

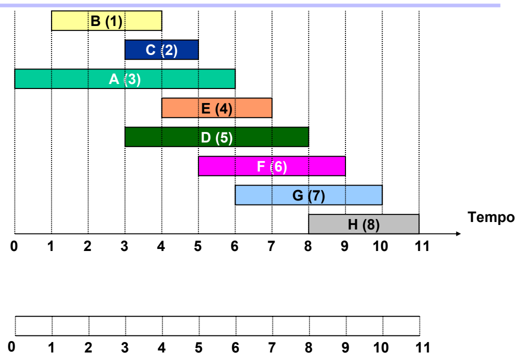
- Assumir que estão ordenadas de forma crescente do tempo final.



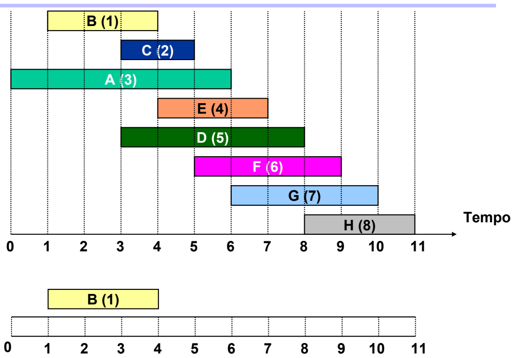
Exemplo 3: Seleção de atividades

SeleçãoAtividades(s, f)
n ← length(s)
A ← {1}
j ← 1
for i ← 2 **to** n **do**
  if $s_i \geq f_j$ **then**
    A ← A ∪ {i}
    j ← i
return A

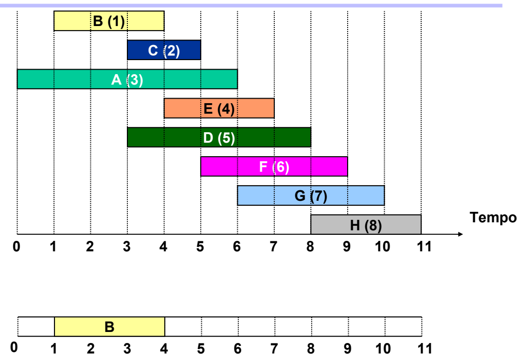
Exemplo 3: Seleção de atividades



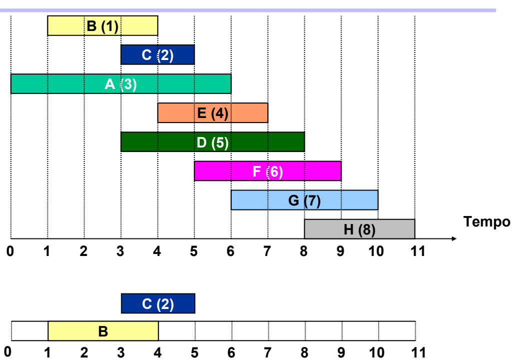
Exemplo 3: Seleção de atividades



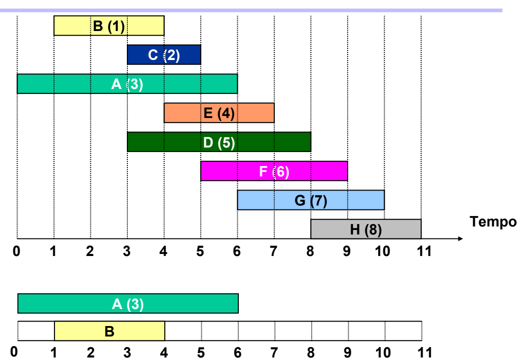
Exemplo 3: Seleção de atividades



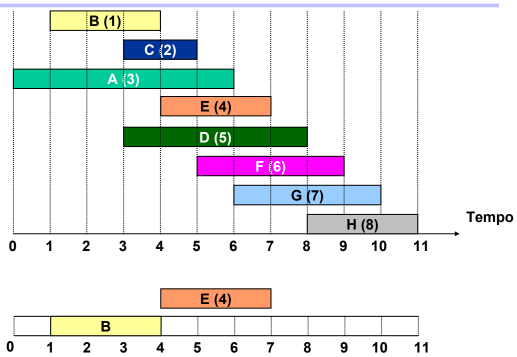
Exemplo 3: Seleção de atividades



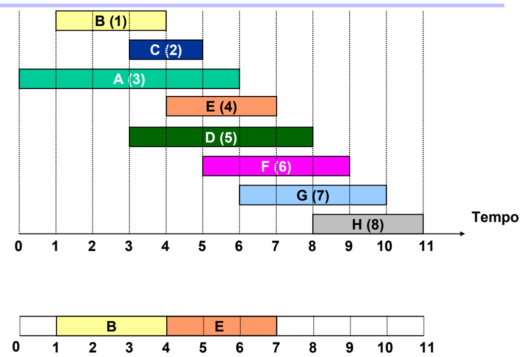
Exemplo 3: Seleção de atividades



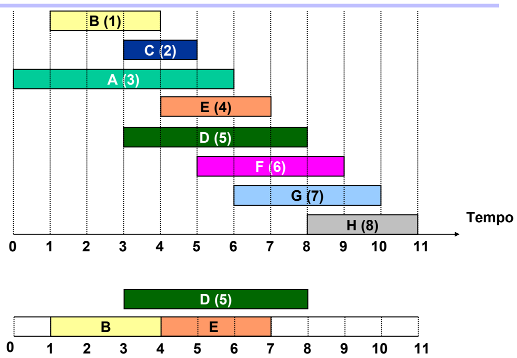
Exemplo 3: Seleção de atividades



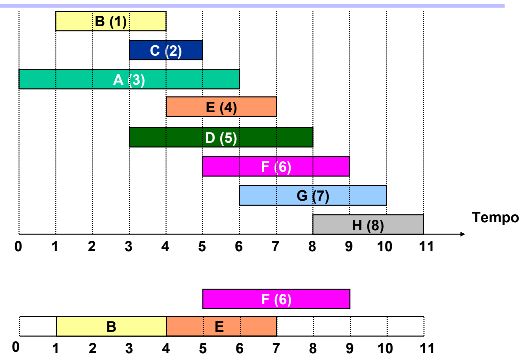
Exemplo 3: Seleção de atividades



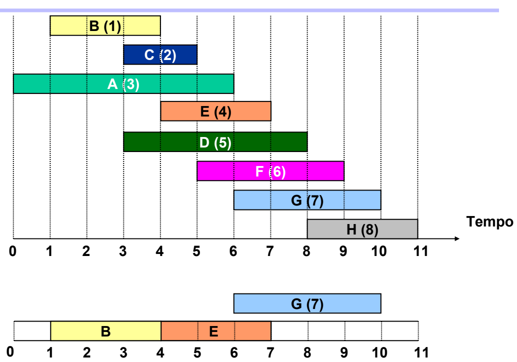
Exemplo 3: Seleção de atividades



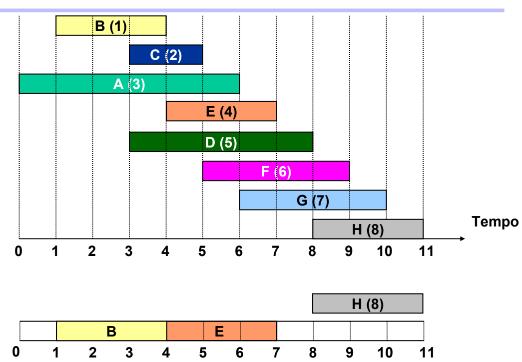
Exemplo 3: Seleção de atividades



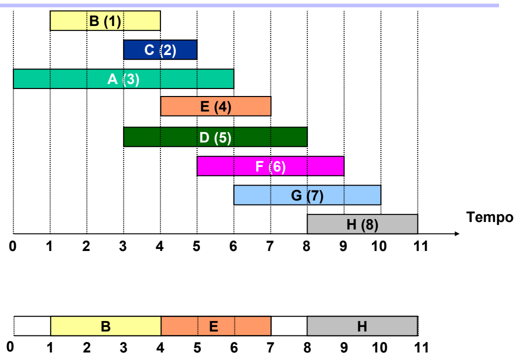
Exemplo 3: Seleção de atividades



Exemplo 3: Seleção de atividades



Exemplo 3: Seleção de atividades



Exemplo 3: Seleção de atividades

- Como provar que a solução sempre produz uma solução ótima?
  - Provar que existe uma solução ótima que começa com a atividade 1.
  - Mostrar que se existir uma solução ótima B que não utiliza a nossa estratégia gulosa, a nossa solução é tão boa quanto B.

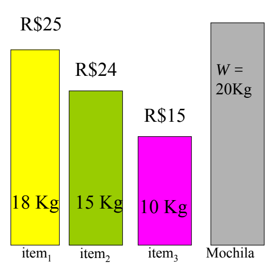
Elementos da Estratégia Gulosa

Existem dois elementos que indicam que a estratégia gulosa pode ser utilizada com sucesso:

1. **Propriedade de Escolha Gulosa**: Uma solução ótima global pode ser obtida a partir de escolhas locais ótimas.
2. **Sub-estrutura ótima**: se uma solução ótima contém dentro dela soluções ótimas para os sub-problemas.

O Problema da Mochila Fracionária

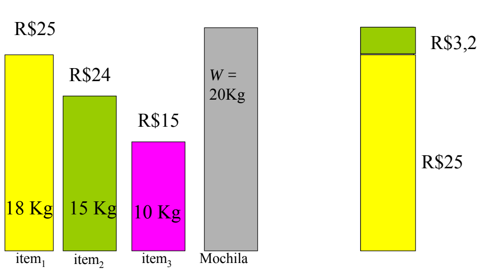
S = {($item_1$, 18, 25), ($item_2$, 15, 24), ($item_3$, 10, 15)} e W= 20



O Problema da Mochila Fracionária

Greedy 1: maximizar o lucro a cada passo

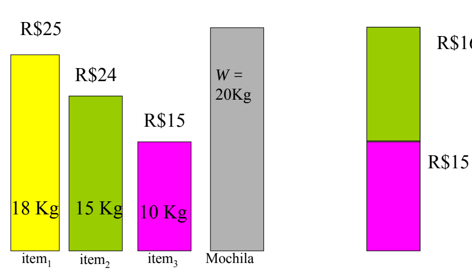
Greedy 1: (1, 2/15, 0) :: Lucro de R$ 28,20



O Problema da Mochila Fracionária

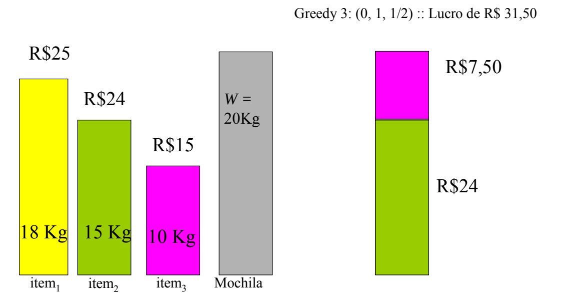
Greedy 2: conservar a capacidade

Greedy 2: (0, 2/3, 1) :: Lucro de R$ 31



O Problema da Mochila Fracionária

Greedy 3: maximiza lucro por unidade de peso



Algoritmo Guloso para Mochila Fracionária

MochilaFracionaria(L, P, W)
► objetos em ordem decrescente de l/p
cap ← W
i ← 1
while $p_i \leq$ cap **do**
    $x_i \leftarrow 1$
    cap ← cap - p_i
    i ← i +1
$x_i \leftarrow$ cap/p_i
for j ← i + 1 **to** n **do**
    $x_j \leftarrow 0$

Prova do Greedy 3

- A solução gulosa tem a seguinte cara:
  - <1, 1, …, 1, c, 0, 0, …, 0> em que a seqüência de 1's tem tamanho k-1, e c < 1.
- Observando dois itens consecutivos, se $x_i < 1 \Rightarrow x_i+1 = 0$.
- A idéia da prova é partir de uma solução ótima e transformá-la na nossa solução gulosa:
  - Seja $<y_1, y_2, \ldots, y_n>$ uma solução ótima.
  - Se essa não é a solução gulosa, existe um i em que $y_i < 1$ e $y_{i+1} > 0$.
  - Encontrar $<y'_1, y'_2, \ldots, y'_n>$ mais parecido com x.
  - Repetir o processo até alcançar a solução gulosa.

Revendo a Estratégia Gulosa

- Os algoritmos que utilizam a estratégia gulosa são simples e de fácil implementação.
- Abordagem top-down.
- Aspectos importantes:
  - Custo de uma solução.
  - Objetivo do problema.
  - Escolha gulosa.

Algoritmo Guloso – Cálculo do Trôco

CalculaTroco(M)
C ← {100, 25, 10, 5, 1}
S ← ∅
soma ← 0
while soma ≠ M **do**
    X ← maior valor de C | soma + X ≤ M
    if item não existe **then**
        return sem solução
    S ← S ∪ {uma moeda de valor X}
    soma ← soma + X
return S

Algoritmo Guloso – Cálculo do Trôco

Lista de Candidatos

CalculaTroco(M)
C ← {100, 25, 10, 5, 1}
S ← ∅
soma ← 0
while soma ≠ M **do**
    X ← maior valor de C | soma + X ≤ M
    if item não existe **then**
        return sem solução
    S ← S ∪ {uma moeda de valor X}
    soma ← soma + X
return S

Algoritmo Guloso – Cálculo do Trôco

Lista de elementos escolhidos

```
CalculaTroco(M)
C ← {100, 25, 10, 5, 1}
S ← ∅
soma ← 0
while soma ≠ M do
      X ← maior valor de C | soma + X ≤ M
      if item não existe then
            return sem solução
      S ← S ∪ {uma moeda de valor X}
      soma ← soma + X
return S
```


Algoritmo Guloso – Cálculo do Trôco

Função VIABILIDADE

```
CalculaTroco(M)
C ← {100, 25, 10, 5, 1}
S ← ∅
soma ← 0
while soma ≠ M do
      X ← maior valor de C | soma + X ≤ M
      if item não existe then
            return sem solução
      S ← S ∪ {uma moeda de valor X}
      soma ← soma + X
return S
```


Algoritmo Guloso – Cálculo do Trôco

Função SOLUÇÃO

```
CalculaTroco(M)
C ← {100, 25, 10, 5, 1}
S ← ∅
soma ← 0
while soma ≠ M do
      X ← maior valor de C | soma + X ≤ M
      if item não existe then
            return sem solução
      S ← S ∪ {uma moeda de valor X}
      soma ← soma + X
return S
```


Algoritmo Guloso – Cálculo do Trôco

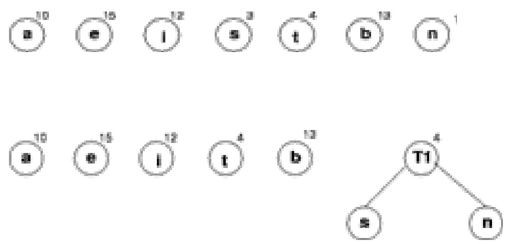
```
AlgoritmoGuloso(C)
C ← conjunto de candidatos
S ← ∅
while C ≠ ∅ e !SOLUÇÃO(S) do
      X ← SELEÇÃO(C)
      C ← C \ {X}
      if VIABILIDADE(S ∪ {X}) then
            S ← S ∪ {X}
if SOLUÇÃO(S) then
      return S
else
      return não tem solução
```

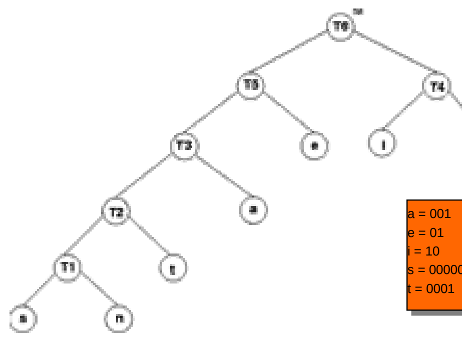

Código de Huffman

- Exemplo:
  - Arquivo de 100.000 caracteres
  - (a, 15000), (e, 25000), (i, 22000), (s, 7000),(t, 5000), (b, 23000), (n, 3000)
  - Usando ASCII extendido (8 bits): 800.000 bits
  - Usando código fixo de 3 bits: 300.000 bits
  - Usar código de tamanho variável?
- Uma possível solução para o problema de Compressão de Arquivos.
- Idéia é usar código de tamanho variável.
- Uso de código prefixo.
- Uso de árvore binária

O Algoritmo de Huffman

- Vamos assumir que o número de caracteres é C.
- Manter uma floresta de árvores.
- O peso de uma árvore é a soma das freqüências de suas folhas.
- C – 1 vezes, selecionar as duas árvores de menor peso e formar uma nova árvore.
- No início do algoritmo, existem C árvores de apenas um nó.
- No final temos apenas uma única árvore. A árvore com o código de Huffman.





Exercício

Problema dos Postos de gasolina:

Input:

D = $[d_1,d_2,...,d_n]$: distâncias entre postos de gasolina

k : autonomia (em km) do tanque de gasolina do carro

Output:

Postos selecionados para abastecer o carro (o número de paradas seja o menor possível)

Assumir:

- $\forall\ 1 \leq i \leq n \quad d_i \leq k$
- d_i é a distância entre postos i-1 e i
- Inicia com o tanque cheio

```
EncontraPostos(P)
S ← Ø
last ← 0
for i = 1 to n do
    if (d_i + last > k)
        S ← S ∪ {p_{i-1}}
        last ← d_i
    last ← last + d_i
return S
```

Escolha da Propriedade Gulosa

- Seja ***S = {s₁, s₂, ... sₖ}*** uma solução ótima.
- Suponha que ***g*** seja a primeira parada determinada pelo nosso algoritmo.
- Temos que mostrar que existe uma solução ótima com a primeira parada sendo ***g***.
  - Se $s_1 = g$, então ***S*** é essa solução.
  - Se $s_1 \neq g$, como o nosso algoritmo escolha o posto mais distante possível, s_1 está antes de ***g***. Podemos dizer que $S' = \{ g, s_2, ..., s_k\}$ é uma solução ótima:
    - Observe que **|*S'*| = |*S*|**.
    - ***S'*** é válido (i.e. não vamos ficar sem gasolina).
    - Por definição da escolha gulosa, podemos alcançar ***g***.
    - Como a distância entre ***g*** e s_2 não é maior do que a distância entre s_1 e s_2, temos combustível suficiente para sair de ***g*** para s_2.
    - O resto de ***S'*** é igual a ***S***. Logo é uma resposta válida.

Subestrutura Ótima

- Seja ***P*** o problema original com uma solução ótima ***S***.
- Após parar no posto ***g***, a uma distância d_i, o subproblema ***P'*** que resta de d_{i+1} para d_n é o mesmo problema, com a diferença da cidade de origem.
- Seja ***S'*** uma solução ótima para ***P'***. É fácil perceber que ***Custo(S) = Custo(S') + 1***.
- Logo, uma solução ótima para ***P*** inclui uma solução ótima para ***P'***.