Paradigmas de projeto de algoritmos

SCC0601 – Introdução à Ciência da Computação II

Prof. Lucas Antiqueira

Paradigmas

1. Divisão e conquista
2. Algoritmos gulosos
3. Programação dinâmica
4. Algoritmos aproximados

Paradigmas

1. Divisão e conquista
2. Algoritmos gulosos
3. Programação dinâmica
4. Algoritmos aproximados

Obs.: Lista não exaustiva!

Introdução

- O projeto de algoritmos requer abordagens adequadas
- A forma como um algoritmo aborda o problema pode levar a um desempenho ineficiente
- Em certo casos, o algoritmo pode não conseguir resolver o problema em tempo viável

Introdução

- Um problema pode ser resolvido por algoritmos de diferentes complexidades adotando-se diferentes paradigmas.
- Um paradigma pode levar a um algoritmo $O(2^n)$ e outro paradigma, que resolve o mesmo problema, a um algoritmo $O(n^3)$.

DIVISÃO E CONQUISTA

Divisão e conquista

- Passos básicos
  1. Dividir o problema a ser resolvido em subproblemas menores e independentes
  2. Encontrar soluções para as partes
  3. Combinar as soluções obtidas em uma solução global

- Os algoritmos podem utilizar recursão para dividir e combinar

Divisão e conquista

- Dada uma entrada, se ela é suficientemente simples, obtemos diretamente uma saída correspondente.

- Caso contrário, ela é decomposta em entradas mais simples, para as quais aplicamos o mesmo processo, obtendo saídas correspondentes que são posteriormente combinadas.

Divisão e conquista

- Exemplos de algoritmos
  - Busca binária
  - Ordenação por intercalação (MergeSort)

Casos já estudados neste curso

ALGORITMOS GULOSOS

Algoritmos gulosos

- Algoritmos gulosos (*greedy*) são tipicamente usados para resolver problemas de otimização

- Por exemplo, o algoritmo para encontrar o caminho mais curto entre duas cidades
  - Um algoritmo guloso escolhe a estrada que parece mais promissora no instante atual e nunca muda essa decisão, independentemente do que possa acontecer depois

Algoritmos gulosos

- A cada iteração
  - Seleciona um elemento conforme uma função gulosa
  - Examina o elemento selecionado quanto a sua viabilidade
  - Decide a sua participação ou não na solução

Algoritmos gulosos

- Ex.: Algoritmo do troco
  - Dado um conjunto de moedas de todos os valores possíveis (1 real, 50 centavos, 25 centavos, 10 centavos, 5 centavos e 1 centavo), obter o menor número de moedas necessárias para que um montante N seja obtido (o troco).

Algoritmos gulosos

- Solução:
  - Comece com um conjunto vazio de moedas selecionadas.
  - A cada passo, adicione uma moeda de valor máximo possível de modo que a soma não ultrapasse N.

Algoritmos gulosos

```
TROCO (N)
C ← {100, 50, 25, 10, 5, 1}
Sol ← {}
Sum ← 0
ENQUANTO sum ≠ N
    x = máximo de C tal que (sum + x ≤ N)
    Sol ← Sol + {x}
    sum ← sum + x
RETORNE Sol
```

Algoritmos gulosos

Exemplo: Você vendeu um produto a R$ 7,65, e o comprador lhe pagou com uma nota de R$ 10,00. Forneça o troco utilizando o menor número de moedas possível.

Algoritmos gulosos

Exemplo: Você vendeu um produto a R$ 7,65, e o comprador lhe pagou com uma nota de R$ 10,00. Forneça o troco utilizando o menor número de moedas possível.

N = 10,00 – 7,65 = 2,35

Algoritmos gulosos

Exemplo: Você vendeu um produto a R$ 7,65, e o comprador lhe pagou com uma nota de R$ 10,00. Forneça o troco utilizando o menor número de moedas possível.

N = 10,00 – 7,65 = 2,35

Sol = {100 100 25 10}

Portanto, o troco é formado por duas moedas de 1 real, uma de 25 centavos e uma de 10 centavos.

Algoritmos gulosos

- Outros exemplos:
  - Compressão de arquivos (árvore de Huffman)
  - Árvore geradora (spanning tree) mínima
  - etc

PROGRAMAÇÃO DINÂMICA

Programação dinâmica

- É um paradigma de programação que tem como objetivo reduzir o tempo de execução de um programa utilizando soluções ótimas a partir de subproblemas previamente calculados

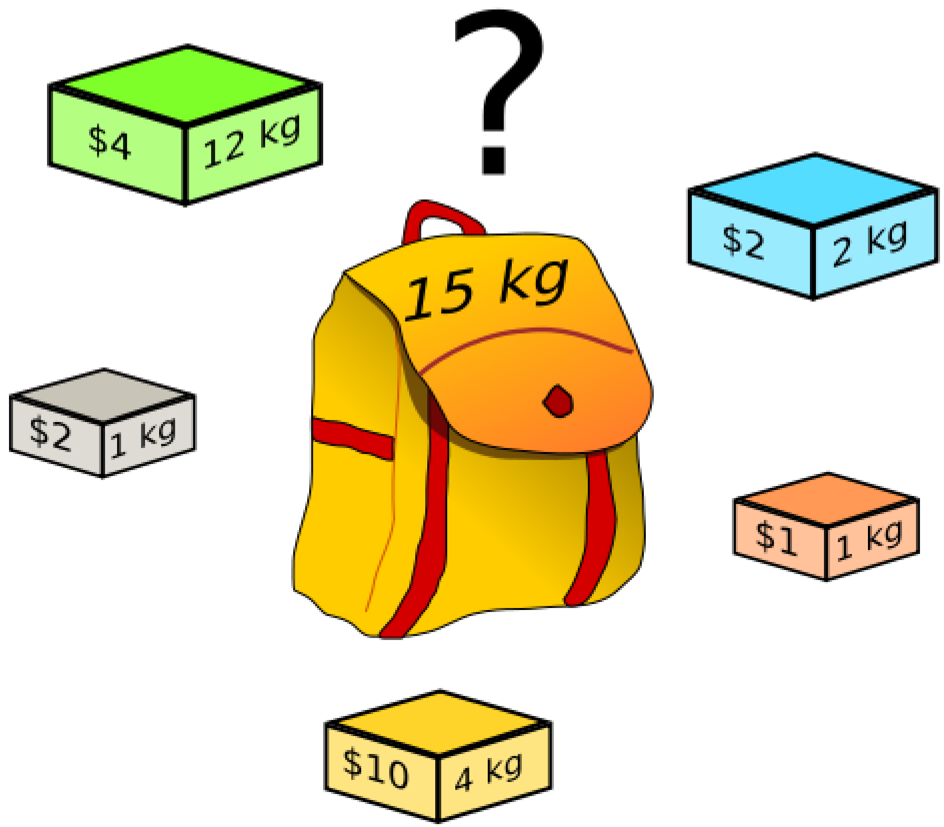
Programação dinâmica

- Numa sequência ótima de escolhas (ou decisões) cada subsequência deve também ser ótima
  - Por exemplo: O menor caminho de São Carlos a São Paulo passando por Jundiaí é dado pelo menor caminho de São Carlos a Jundiaí mais o menor caminho de Jundiaí a São Paulo.

Programação dinâmica

- Passos:
  - Dividir o problema em sub-problemas
  - Computar os valores de uma solução de forma *bottom-up* e armazená-los (memorização)
  - Construir a solução ótima para cada sub-problema utilizando os valores já computados.

Problema da mochila



http://pt.wikipedia.org/wiki/Ficheiro:Knapsack.svg

Como maximizar a soma dos valores dos objetos colocados dentro de uma mochila, dado que não podemos ultrapassar o peso máximo permitido?

Problema da mochila

Dados números $p_1, p_2, \ldots, p_n$, $v_1, v_2, \ldots, v_n$ e um subconjunto X de $\{1,2,\ldots,n\}$, denotaremos por $p(X)$ e $v(X)$ as somas $\sum_{i \in X} p_i$ e $\sum_{i \in X} v_i$ respectivamente.

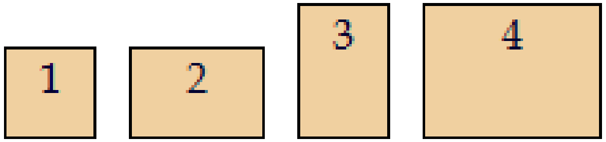
> PROBLEMA DA MOCHILA BOOLEANA: Dados números naturais $p_1, p_2, \ldots, p_n$, $v_1, v_2, \ldots, v_n$ e c, encontrar um subconjunto X de $\{1,2,\ldots,n\}$ que maximize $v(X)$ sob a restrição $p(X) \leq c$.

Diremos $1,2,\ldots,n$ são os *objetos* do problema, que p_i é o *peso* e que v_i é o *valor* do objeto i. (A ordem em que os pesos e os valores são dados é, obviamente, irrelevante.) Diremos que c é a *capacidade da mochila*. Diremos ainda que uma *mochila viável* é qualquer subconjunto X de $\{1,2,\ldots,n\}$ tal que $p(X) \leq c$. O *valor* de uma mochila X é o número $v(X)$. Nosso problema é encontrar uma mochila viável de valor máximo.

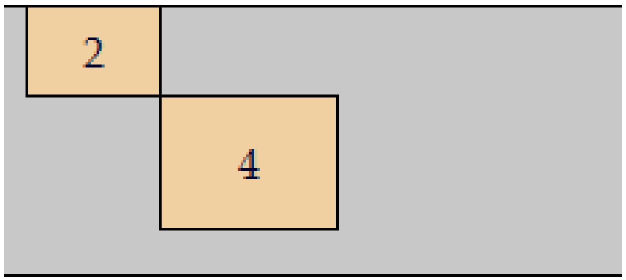
http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Problema da mochila

EXEMPLO Tome $n = 4$, $p = (20,20,30,30)$ e $v = (20,30,20,40)$. Represente cada objeto i por um retângulo de altura p_i e largura v_i.



Qualquer conjunto de objetos pode ser disposto "em escada", com o canto inferior direito de um objeto tocando o canto superior esquerdo do objeto seguinte. A figura abaixo representa o conjunto de objetos {2,4}. Esse conjunto é uma solução da instância do problema que tem capacidade $c = 60$ (altura da faixa cinza).



http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Problema da mochila

A estrutura recursiva do problema

Suponha que $(p_1,\ldots,p_n, v_1,\ldots,v_n, c)$ é uma instância do problema. Seja X uma solução da instância. Temos duas possibilidades: X contém n ou não contém n. No segundo caso, X é solução da subinstância

X não contém n → $(p_1,\ldots,p_{n-1}, v_1,\ldots,v_{n-1}, c)$

No primeiro caso, $p_n \le c$ e $X-\{n\}$ é solução da subinstância

X contém n → $(p_1,\ldots,p_{n-1}, v_1,\ldots,v_{n-1}, c-p_n)$.

Essa estrutura recursiva sugere imediatamente um algoritmo para o problema.

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Problema da mochila

Algoritmo de programação dinâmica

guardar em uma tabela, digamos t, as soluções das (sub)instâncias do problema. A tabela é definida assim: para $i = 0,1,\ldots,n$ e $b = 0,1,\ldots,c$,

$t[i,b]$ é o valor de uma solução
da instância $(p_1,\ldots,p_i, v_1,\ldots,v_i, b)$ do problema.

A tabela t satisfaz a seguinte recorrência: para todo $i \geq 1$ e todo b,

$$t[i,b] = \begin{cases} t[i-1,b] & \text{se } p_i > b \quad \text{e} \\ \max\left(t[i-1,b]\,,\ t[i-1,b-p_i] + v_i\right) & \text{se } p_i \leq b\,. \end{cases}$$

(Observe que todos os números da forma $b-p_i$ são não negativos e portanto a posição $(i-1,b-p_i)$ da tabela está bem definida.)

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Programação dinâmica

- Exemplo:
  - Mochila de capacidade c=5
  - n=4 objetos

	1	**2**	**3**	**4**
p	4	2	1	3
v	500	400	300	450

t	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Programação dinâmica

- Exemplo:
  - Mochila de capacidade c=5
  - n=4 objetos

	1	**2**	**3**	**4**
p	4	2	1	3
v	500	400	300	450

t	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	700	700	800
4	0	300	400	700	750	850

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Exercício

1. Solucione o problema da mochila utilizando um algoritmo guloso.

2. Diga se esse algoritmo sempre fornece a solução ótima.

Programação dinâmica

- Outros exemplos
  - Algoritmo de Dijkstra (caminhos mínimos em grafos)
  - Subsequência máxima
  - etc

ALGORITMOS APROXIMADOS

Algoritmos aproximados

- Gera soluções aproximadas, que podem não ser ótimas, mas são próximas delas
- São vantajosas quando a solução ótima consome muito tempo para ser obtida
- Faz-se necessária uma medida de qualidade

Algoritmos aproximados

- *Exemplo*: O algoritmo da mochila desenvolvido via programação dinâmica tem complexidade $\Theta(nc)$
  - Ou seja, é a complexidade de se preencher (percorrer) a matriz t

Algoritmos aproximados

- *Exemplo*: O algoritmo da mochila desenvolvido via programação dinâmica tem complexidade: $\Theta(nc)$
  - Ou seja, é a complexidade de se preencher (percorrer) a matriz t

- Dependendo da instância a ser resolvida, esse algoritmo pode ser muito ineficiente

Algoritmos aproximados

- *Exemplo*: O algoritmo da mochila desenvolvido via programação dinâmica tem complexidade: $\Theta(nc)$
  - Ou seja, é a complexidade de se preencher (percorrer) a matriz t

- Dependendo da instância a ser resolvida, esse algoritmo pode ser muito ineficiente

- *Alternativa*: fornecer um resultado aproximado, porém mais eficiente

Algoritmos aproximados

- Problema da mochila com solução X aproximada:
  - Garantiremos que $v(X) \geq \frac{1}{2} v(S)$
    - Onde S é a solução ótima

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html

Algoritmos aproximados

- Problema da mochila com solução X aproximada:

  - Garantiremos que $v(X) \geq ½\, v(S)$
    - Onde S é a solução ótima

- Para tanto, tome o *valor específico* de um objeto i como sendo o número v_i/p_i

- Suponha que os objetos estão ordenados de maneira que: $v_1/p_1 \geq v_2/p_2 \geq \ldots \geq v_n/p_n$

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html

Algoritmos aproximados

<u>MOCHILA-QUASE-ÓTIMA</u> (p, v, n, c)

$s \leftarrow 0$
$k \leftarrow 1$
enquanto $k \leq n$ e $s+p_k \leq c$ faça
    $s \leftarrow s + p_k$
    $k \leftarrow k + 1$
$X \leftarrow \{1, 2, \ldots, k-1\}$
se $k > n$ ou $v(X) \geq v_k$
    então devolva X
    senão devolva $\{k\}$

Como esse algoritmo funciona?

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html

Algoritmos aproximados

<u>MOCHILA-QUASE-ÓTIMA</u> (p, v, n, c)

1. $s \leftarrow 0$
2. $k \leftarrow 1$
3. enquanto $k \leq n$ e $s+p_k \leq c$ faça
4. $\quad s \leftarrow s + p_k$
5. $\quad k \leftarrow k + 1$
6. $X \leftarrow \{1, 2, \ldots, k-1\}$
7. se $k > n$ ou $v(X) \geq v_k$
8. $\quad$ então devolva X
9. $\quad$ senão devolva $\{k\}$

Como esse algoritmo funciona?
Abordagem **gulosa** para aproximar a solução ótima:
É dada preferência aos objetos de maiores valores especificos

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html

Algoritmos aproximados

MOCHILA-QUASE-ÓTIMA (p, v, n, c)

$s \leftarrow 0$

$k \leftarrow 1$

enquanto $k \leq n$ e $s + p_k \leq c$ faça

  $s \leftarrow s + p_k$

  $k \leftarrow k + 1$

$X \leftarrow \{1, 2, \ldots, k-1\}$

se $k > n$ ou $v(X) \geq v_k$

  então devolva X

  senão devolva $\{k\}$

Complexidade dessa solução?

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html

Algoritmos aproximados

<u>MOCHILA-QUASE-ÓTIMA</u> (p, v, n, c)

$s \leftarrow 0$

$k \leftarrow 1$

enquanto $k \leq n$ e $s + p_k \leq c$ faça

 $s \leftarrow s + p_k$

 $k \leftarrow k + 1$

$X \leftarrow \{1, 2, \ldots, k-1\}$

se $k > n$ ou $v(X) \geq v_k$

 então devolva X

 senão devolva $\{k\}$

Complexidade dessa solução?
O(n) + O(ordenação inicial)

http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html

Algoritmos aproximados

- *Outros exemplos de aplicação*:
  - Caixeiro viajante
  - Coloração de grafos
  - etc

Créditos

Foi utilizado material do prof. Paulo Feofiloff

http://www.ime.usp.br/~pf/