

# Teoria dos Grafos

## Aula 12

### Aula de hoje

- Problema da soma do subconjunto (*subset sum*)
- Programação dinâmica
- Problema da mochila

### Roteiro

- Paradigma guloso
- Coloração
- Número cromático
- Algoritmo guloso
- Teorema das 4 cores

# Escalonamento de Tarefas

- N tarefas
- Cada tarefa leva tempo  $t_i$  para executar
- T: tempo total disponível
- **Objetivo:** Maximizar uso do tempo  $T$  disponível (minimizar a sobra)
  - número de tarefas não interessa
- **Problema:** Quais tarefas executar?



# Investimentos

- N investimentos possíveis (integrais, não divisíveis)
- Cada investimento tem preço  $p_i$
- W: orçamento disponível
- **Objetivo:** Maximizar compra dos investimentos dentro do orçamento (minimizar sobra)
- **Problema:** Quais investimentos fazer?



# Problema da Soma de Subconjunto

- Abstração destes problemas e muitos outros
  - *Subset Sum Problem*
- Dado um conjunto de  $N$  objetos, cada um com peso inteiro  $w_i$ , e um limite  $W$
- **Solução:** subconjunto de objetos tal que soma dos pesos seja menor (ou igual) a  $W$
- **Problema:** Determinar subconjunto que possui a maior soma (minimiza a sobra)

# Problema da Soma de Subconjunto

## ■ Exemplo

- $N = \{1, 2, 3, 4, 5, 6\}$
- $w_1 = 4, w_2 = 5, w_3 = 11, w_4 = 3, w_5 = 2, w_6 = 3$
- $W = 10$
- Subconjunto ótimo:  $O = \{2, 5, 6\}$ 
  - valor de  $O = w_2 + w_5 + w_6 = 10$
- Solução é sempre única?

**Algoritmo para o problema?**

# Algoritmo Guloso

- Idéias para um guloso?
- Menor peso primeiro
  - Funciona? Contra-exemplo?
- Maior peso primeiro
  - Funciona? Contra-exemplo?

**Não conhecemos algoritmo guloso  
que seja ótimo (sempre)!**

# Programação Dinâmica

- Técnica para construção de algoritmos
- Baseada na análise e decomposição da estrutura do problema
  - construir uma recursão para solução ótima
- Memorização (armazenamento) de resultados temporários

**Técnica fundamental e poderosa!**

- Difícil ser empregada em casos gerais

# Algoritmo para o Problema

- Abordagem via programação dinâmica
- Considere “ $O$ ” o conjunto ótimo de objetos
  - um subconjunto que minimiza a sobra
- Considere o peso total desse subconjunto
  - dado por  $OPT(n)$
- O que podemos dizer sobre o objeto  $n$  do conjunto de objetos ?
- Duas possibilidades
  - pertence a  $O$
  - não pertence a  $O$

# Analizando Solução Ótima

- O que podemos afirmar se  $n$  não pertence a  $O$ ?
  - $O$  é a solução ótima para o problema com os outros  $n-1$  objetos
  - pois caso contrário,  $O$  pode ser melhorado para o problema com  $n-1$  objetos
- Se  $n$  não pertence a  $O$ , então
  - $\text{OPT}(n) = \text{OPT}(n-1)$
  - onde  $\text{OPT}(n)$  é o valor da solução ótima considerando os primeiros  $n$  objetos
  - soma dos pesos dos objetos do subconjunto ótimo considerando os  $n$  primeiros objetos cuja soma é menor que  $W$

# Analizando Solução Ótima

- Se  $n$  pertence a  $O$ , então...

**O que podemos dizer  
neste caso?**

- Incluir objeto  $n$  não necessariamente exclui nenhum outro objeto
  - difícil escrever OPT apenas removendo um objeto
- O que ocorre quando  $n$  pertence a  $O$ ?

**Limite  $W$  diminui (de  $w_n$ )**

# Adicionando uma Variável

- Definir OPT somente em função do número de objetos é impossível
- **Idéia:** adicionar outra variável para facilitar definir o subproblema

**Que variável?**

- Valor do orçamento,  $W$
- Se  $n$  pertence a  $O$ , então
  - $OPT(n, W) = w_n + OPT(n-1, W - w_n)$
  - onde  $OPT(n, w)$  é o valor da solução ótima com os primeiros  $n$  objetos e orçamento  $w$

# Definindo Solução Ótima

- Duas variáveis facilita a recursão
- Mas aumenta o número de subproblemas que precisamos resolver
  - ótimo em função do limite W
- Definição de OPT:

$$OPT(i, w) = \max_S \sum_{j \in S} w_j$$

Um subconjunto  
do conjunto  $\{1, 2, \dots, i\}$

Soma dos elementos do  
conjunto S tem que ser  
menor que W (restrição)

# Exemplo

- $N = \{1, 2, 3, 4, 5, 6\}$
- $w_1 = 2, w_2 = 5, w_3 = 10, w_4 = 3, w_5 = 2, w_6 = 3$
- $W = 10$
- $\text{OPT}(1, 1) = ?$
- $\text{OPT}(1, 2) = ?$
- $\text{OPT}(2, 5) = ?$
- $\text{OPT}(2, 8) = ?$
- $\text{OPT}(2, 10) = ?$
- $\text{OPT}(3, 10) = ?$

$$\text{OPT}(i, w) = \max_S \sum_{j \in S} w_j$$

# Custo da Solução Ótima

- Se  $i$  não pertence a solução ótima
  - $\text{OPT}(i, w) = \text{OPT}(i-1, w)$
- Se  $i$  pertence a solução ótima
  - $\text{OPT}(i, w) = w_i + \text{OPT}(i-1, w - w_i)$
- Qual das duas será utilizada?
  - a de maior valor, claro!
- Ou seja, quando  $w > w_i$   
(caso contrário, não podemos adicionar  $i$ )
  - $\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i))$

# Algoritmo

- Algoritmo para calcular  $\text{OPT}(n, W)$ ?
  - iterativo, não recursivo (mas utilizando recursão)

SubsetSum-OPT( $n, W$ )

```
Array M[0, . . . , n ; 0, . . . , W]  
Init M[0, w]=0 w=0, 1, . . . , W
```

```
For i=1, 2, . . . , n
```

```
  For w=0, 1, . . . , W
```

```
    Se w < w[i] entao
```

```
      M[i, w] = M[i-1, w]
```

```
    Senao
```

```
      M[i, w] = max(M[i-1, w],  
                      w[i] + M[i-1, w-w[i]])
```

```
Retorna M[n, W]
```

# Exemplo

- $N = \{1, 2, 3\}$
- $w_1 = 2, w_2 = 3, w_3 = 4$
- $W = 8$
- Tabela M que será construída?

		w (orçamento)								
		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0
i (objeto)		1	0	0	2	2	2	2	2	2
		2	0	0	2	3	3	5	5	5
		3	0	0	2	3	4	5	6	7

**Inicialização**

**Solução ótima**

# Complexidade

- Tempo de execução do algoritmo?
- Observações
  - calcular cada elemento da matriz M, tempo constante
  - Tempo total: número de elementos da matriz M
  - $O(nW)$
- Tempo de execução é ***pseudo-polynomial***
  - polinomial no valor de W, mas não no tamanho da entrada
  - exponencial no tamanho da entrada
    - este problema é difícil, não é polinomial

# Obtendo o Conjunto Ótimo

- Como obter o conjunto Ótimo?
  - dado  $M$ ?
- Se  $i$  pertence ao ótimo, então temos que
  - $w_i + OPT(i-1, w-w_i) > OPT(i-1, w)$
- Algoritmo iterativo, para cada objeto verifica se ele pertence ou não ao ótimo
  - Utiliza matriz  $M$
  - Começa com último objeto,  $M[n, w]$
  - Complexidade?

# Obtendo o Conjunto Ótimo

- Se  $i$  pertence ao ótimo, então temos que
  - $w_i + \text{OPT}(i-1, w-w_i) > \text{OPT}(i-1, w)$
- Exemplo

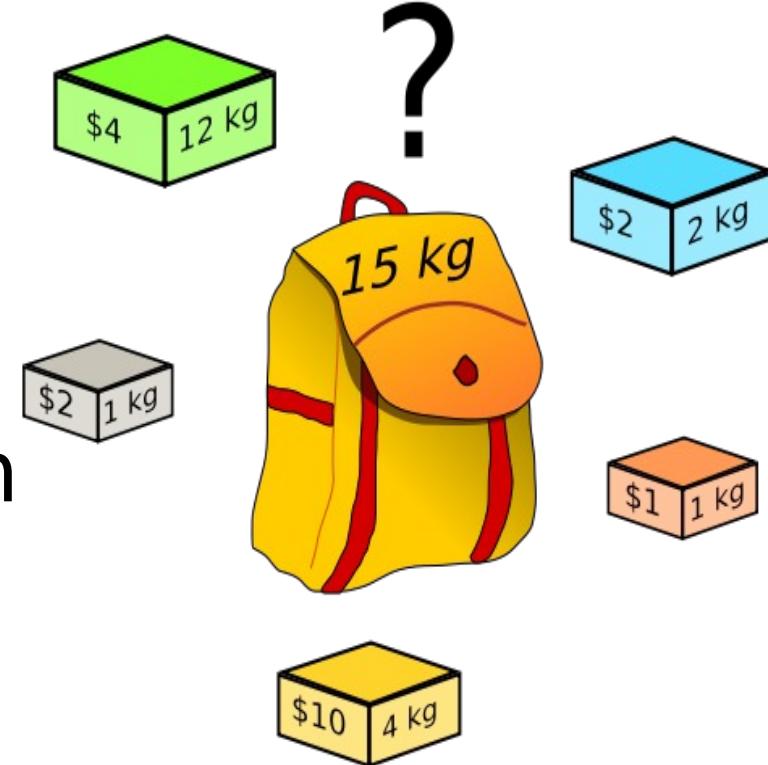
	w (orçamento)									
	0	1	2	3	4	5	6	7	8	
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	2	2	2	2	2	2	
2	0	0	2	3	3	5	5	5	5	
3	0	0	2	3	4	5	6	7	7	

Legend:

- = pertence
- = não pertence

# Problema da Mochila

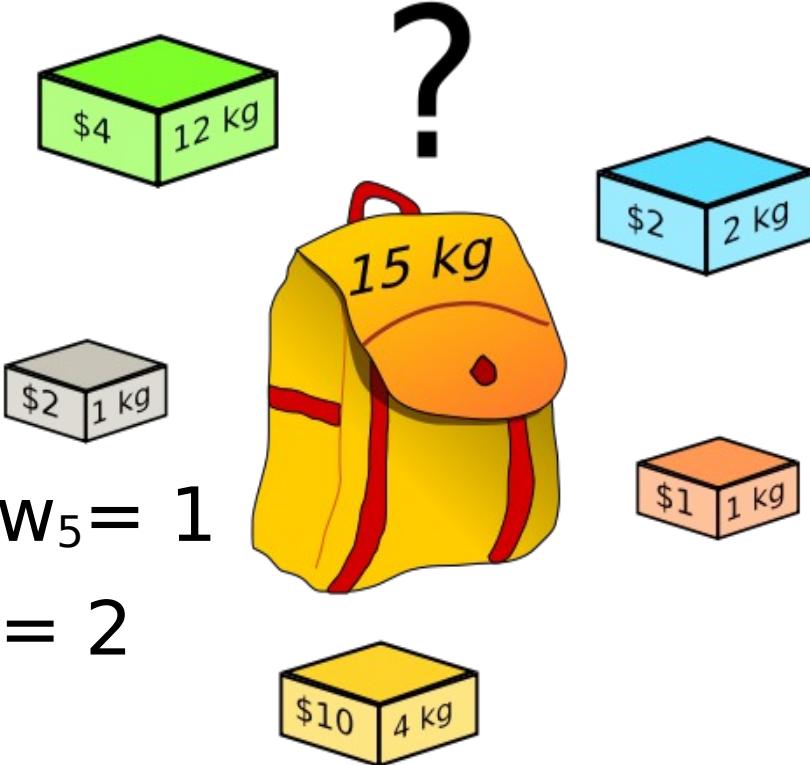
- Generalização do problema anterior
  - *knapsack problem*
- Objetos possuem *valor* além do peso
- Pesos continuam limitando os objetos
- **Problema:** maximizar *valor* do subconjunto (soma dos valores dos objetos)
  - sobre restrição de peso total



# Problema da Mochila

- Exemplo (ao lado)

- $N = \{1, 2, 3, 4, 5\}$
- $w_1 = 12, w_2 = 2, w_3 = 1, w_4 = 4, w_5 = 1$
- $v_1 = 4, v_2 = 2, v_3 = 1, v_4 = 10, v_5 = 2$
- $W = 15$



- **Problema:** maximizar *valor* dos objetos a serem incluídos na mochila
  - sobre restrição de peso total

# Análise da Solução Ótima

- $OPT(i, w)$  = valor da solução ótima com os  $i$  primeiros objetos e com limite de peso  $w$

$$OPT(i, w) = \max_S \sum_{j \in S} v_j$$

Um subconjunto do conjunto  $\{1, 2, \dots, i\}$

Soma dos valores do subconjunto  $S$

- Sujeito ao limite de peso  $w$ :  $\sum_{j \in S} w_j \leq w$

# Análise da Solução Ótima

- Se  $i$  não pertence a solução ótima

  - $\text{OPT}(i, w) = \text{OPT}(i-1, w)$

- Se  $i$  pertence a solução ótima

  - $\text{OPT}(i, w) = v_i + \text{OPT}(i-1, w - w_i)$

- Qual das duas será utilizada?

  - A de maior valor, claro!

- Ou seja, quando  $w > w_i$

  - $\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), v_i + \text{OPT}(i-1, w - w_i))$

Única diferença!  
Valor e não peso  
do objeto

Algoritmo é idêntico

# Complexidade e Variações

- $O(nW)$  : pseudo-polynomial (cresce com magnitude do parâmetro e não com quantidade de parâmetros)
- Problema difícil: não se conhece algoritmo polinomial para resolver (vale 1 milhão)
- Algoritmos aproximativos: chegam próximo do ótimo em tempo polinomial
  - usam programação dinâmica
- Variações
  - múltiplas cópias de um mesmo item
  - quantidade fracionária de um item (fácil de resolver)