

Teoria dos Grafos

Aula 20

Aula passada

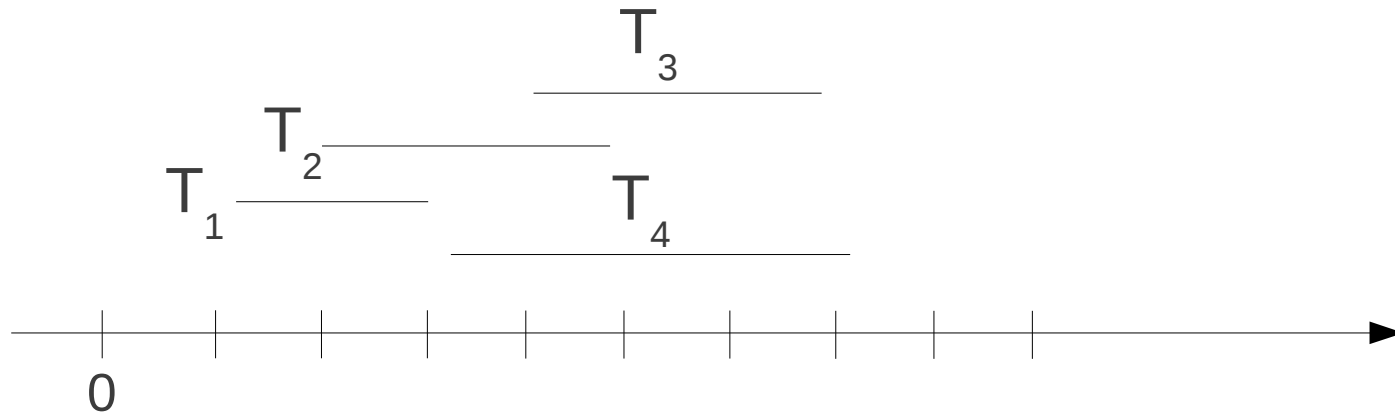
- Escalonando tarefas no tempo (*interval scheduling*) com pesos
- Programação Dinâmica

Aula de hoje

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

Escalonamento de Tarefas

- Tarefas possuem dependência temporal



- Estrutura do problema está mais explícita
 - Tarefas que colidem no tempo não podem ser executadas juntas
- Fácil identificar recursão
- Iremos relaxar esta dependência

Escalonamento de Tarefas

- N tarefas
- Cada tarefa leva tempo t_i para executar
- T: tempo total disponível



- **Objetivo:** Maximizar o uso do orçamento de tempo T (minimizar a sobra)
 - número de tarefas não interessa
- **Problema:** Quais tarefas executar?

Investimentos

- N investimentos possíveis
- Cada investimento tem preço p_i
- W: orçamento disponível
- **Objetivo:** Maximizar os 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 leva a maior soma

Problema da Soma de Subconjunto

■ Exemplo

■ $N = \{1, 2, 3, 4, 5, 6\}$

■ $w_1 = 2, w_2 = 7, w_3 = 11, w_4 = 3, w_5 = 2, w_6 = 4$

■ $W = 10$

■ Subconjunto ótimo: $O = \{2, 4\}$

■ custo de $O = w_2 + w_4 = 10$

■ 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 se conhece algoritmo guloso
que resolva (de maneira ótima)
este problema!**

Algoritmo para o Problema

- Abordagem via programação dinâmica
- Estudo da estrutura da solução ótima
- Considere “O” o conjunto de tarefas ótima
 - subconjunto que maximiza o número de objetos
- O que podemos dizer sobre o último objeto do conjunto de objetos ?
- Duas possibilidades
 - Pertence a O
 - Não pertence a O

Analizando Solução Ótima

- Se n não pertence a O , então 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
 - $OPT(n) = OPT(n-1)$
 - onde $OPT(n)$ é o custo da solução ótima com os primeiros n objetos
 - soma dos pesos dos objetos do maior subconjunto com os primeiros n objetos cuja soma é menor que W

Analizando Solução Ótima

- Se n pertence a O , então...

O que podemos dizer neste caso?

- Remover n não necessariamente exclui nenhum outro objeto
 - difícil escrever OPT apenas removendo um objeto
- O que ocorre?

Limite W diminui (de w_n)

Adicionando uma Variável

- Definir OPT somente em função dos objetos é impossível (dois OPT(n-1))
- **Idéia:** adicionar outra variável para facilitar a 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 custo da solução ótima com os primeiros n objetos e limite w

Definindo Solução Ótima

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

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

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

Soma dos elementos tem que ser menor que w

Exemplo

■ $N = \{1, 2, 3, 4, 5, 6\}$

■ $w_1 = 2, w_2 = 7, w_3 = 10, w_4 = 3, w_5 = 6, w_6 = 4$

■ $W = 10$

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

■ $OPT(1, 1) = ?$

■ $OPT(1, 2) = ?$

■ $OPT(2, 5) = ?$

■ $OPT(2, 8) = ?$

■ $OPT(2, 10) = ?$

■ $OPT(3, 10) = ?$

Custo da Solução Ótima

- Se i não pertence a solução ótima
 - $OPT(i, w) = OPT(i-1, w)$
- Se i pertence a solução ótima
 - $OPT(i, w) = w_i + OPT(i-1, w - w_i)$
- Qual das duas será utilizada?
 - A que der maior valor!
- Ou seja, quando $w > w_i$
(caso contrário, não podemos levar i)
 - $OPT(i, w) = \max(OPT(i-1, w), w_i + OPT(i-1, w - w_i))$

Algoritmo

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

SubsetSum-OPT(n, W)

Array $M[0, \dots, n ; 0, \dots, W]$

Init $M[0, w] = 0 \quad w=0, 1, \dots, W$

For $i=1, 2, \dots, n$

 For $w=0, 1, \dots, W$

 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?

Complexidade

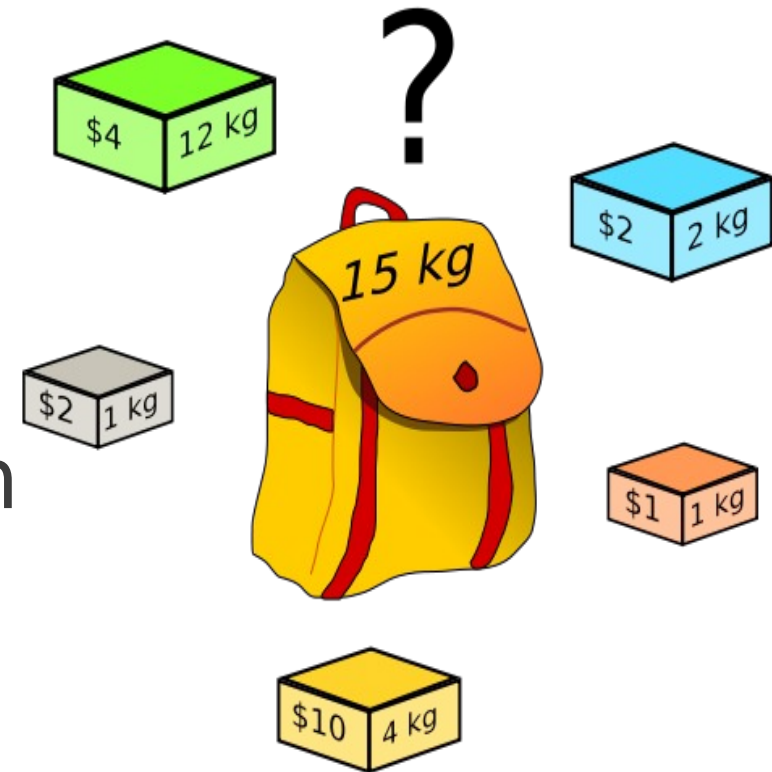
- Tempo de execução do algoritmo?
- Observações
 - calcular cada elemento da matriz M , tempo constante
 - Tempo total: número de elementos da matriz M
 - $O(nW)$
- Tempo de execução é ***pseudo-polinomial***
 - polinomial na magnitude de W , mas não no tamanho da entrada
 - exponencial no tamanho da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

Obtendo o Conjunto Ótimo

- Como obter o conjunto Ótimo?
 - dado M ?
- Mesma idéia que problema anterior
- Se i pertence ao ótimo, então temos que
 - $w_i + \text{OPT}(i-1, w-w_i) > \text{OPT}(i-1, w)$
- Algoritmo iterativo, para cada objeto verifica se ele pertence ou não ao ótimo
 - Utiliza matriz M
 - Complexidade?

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)
 - e não soma dos pesos



Problema da Mochila

- Exemplo (ao lado)

- $N = \{1, 2, 3, 4, 5\}$

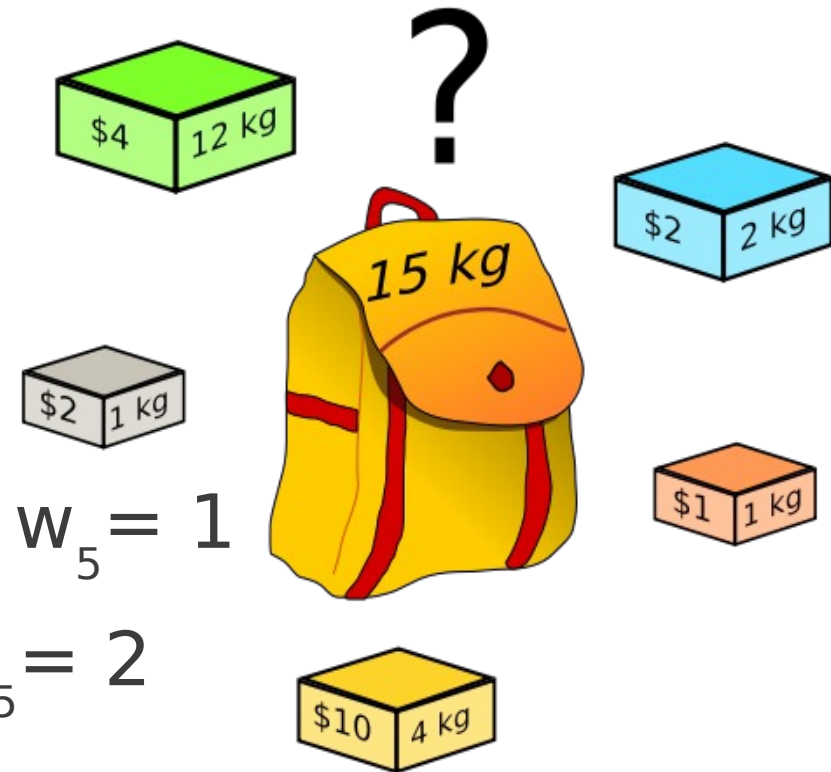
- $w_1 = 12, w_2 = 2, w_3 = 1, w_4 = 4, w_5 = 1$

- $v_1 = 4, v_2 = 2, v_3 = 1, v_4 = 10, v_5 = 2$

- $W = 15$

- **Problema:** maximizar *valor* dos objetos a serem incluídos na mochila

- e não soma dos pesos



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 dos subconjuntos
do conjunto $\{1, 2, \dots, i\}$

Soma dos valores

- 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

- $OPT(i, w) = OPT(i-1, w)$

- Se i pertence a solução ótima

- $OPT(i, w) = v_i + OPT(i-1, w - w_i)$

- Qual das duas será utilizada?

- A que der maior valor!

- Ou seja, quando $w > w_i$

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

Diferença!
Valor e não peso
do objeto

Algoritmo é idêntico