ESTRUTURAS DE DADOS II

MSC. DANIELE CARVALHO OLIVEIRA

DOUTORANDA EM CIÊNCIA DA COMPUTAÇÃO - USP

MESTRE EM CIÊNCIA DA COMPUTAÇÃO – UFU

BACHAREL EM CIÊNCIA DA COMPUTAÇÃO - UFJF

ALGORITMOS GULOSOS

3 ALGORITMOS GULOSOS

- Para resolver um problema, um algoritmo guloso escolhe, em cada iteração, o objeto mais apetitoso que vê pela frente.
- O objeto escolhido passa a fazer parte da solução que o algoritmo constrói.
- Um algoritmo guloso é míope: ele toma decisões com base nas informações disponíveis na iteração corrente, sem olhar as consequências que essas decisões terão no futuro.
- Um algoritmo guloso jamais se arrepende ou volta atrás: as escolhas que faz em cada iteração são definitivas.

4 ALGORITMOS GULOSOS

- Embora algoritmos gulosos pareçam obviamente corretos, a prova de sua correção é, em geral, muito sutil.
- Para compensar, algoritmos gulosos são muito rápidos e eficientes.
  - (É preciso dizer, entretanto, que os problemas que admitem soluções gulosas são um tanto raros.)

5 EXEMPLOS DE ALGORITMOS GULOSOS

- Problema da árvore de Huffman
- Problema da arborescência maximal de peso mínimo num grafo (algoritmos de Prim e Kruskal)
- Problema do escalonamento de intervalos
- Problema da mochila fracionária
- Problema do troco

6 PROBLEMA DO ESCALONAMENTO DE TAREFAS

- A figura abaixo especifica uma coleção de intervalos e uma subcoleção disjunta máxima (sdm) da coleção. A sdm é indicada pelos 1 do seu vetor característico x:

s	4	6	13	4	2	6	7	9	1	3	9
f	8	7	14	5	4	9	10	11	6	13	12
x	0	1	1	0	1	0	0	1	0	0	0

- É fácil verificar que a coleção de 4 intervalos definida por x é disjunta. Mas não é óbvio que ela seja máxima. Você tem certeza de que não existem 5 intervalos disjuntos dois a dois?

PROBLEMA DO ESCALONAMENTO DE TAREFAS

- O problema pode ser resolvido por um algoritmo guloso que é mais rápido e mais simples que a programação dinâmica.
- O algoritmo supõe $f1 \leq \ldots \leq fn$ e $n \geq 0$ e devolve uma sdm da coleção de intervalos definida por (s,f,n):

```
SDM-GULOSO (s, f, n)
f0 ← −∞
X ← { }
i ← 0
para k ← 1 até n faça
    se sk > fi
        então X ← X ∪ {k}
            i ← k
devolva X
```

8 PROBLEMA DO ESCALONAMENTO DE TAREFAS

- O algoritmo é guloso: ele abocanha o primeiro intervalo viável k que encontra, sem se preocupar com o que vem depois.
- Uma vez tomada a decisão de colocar k na sdm (linhas 6 e 7), essa decisão nunca será revista.
- O algoritmo guloso consome O(n) unidades de tempo no pior caso. Isso não inclui o tempo O(n log n) necessário para fazer a ordenação prévia dos intervalos.

O PROBLEMA DA MOCHILA FRACIONÁRIA

- O problema da mochila fracionária também é conhecida como problema da mochila contínua. Em inglês, o problema é conhecido como fractional knapsack problem. Ele é bem mais fácil que o problema da mochila booleana e pode ser resolvida por um algoritmo guloso muito eficiente.

- Dados vetores $(x1,x2,\ldots,xn)$ e $(p1,p2,\ldots,pn)$, denotaremos por $x \cdot p$ o produto escalar de x por p. Portanto, $x \cdot p = x1p1 + x2p2 + \ldots + xnpn$.

PROBLEMA DA MOCHILA FRACIONÁRIA: Dados vetores naturais $(p_1, p_2, \ldots, p_n)$, $(v_1, v_2, \ldots, v_n)$ e um número natural c, encontrar um vetor racional $(x_1, x_2, \ldots, x_n)$ que maximize $x \cdot v$ sob as restrições $x \cdot p \leq c$ e $0 \leq x_i \leq 1$ para todo i.

- Diremos que (p1,…,pn) é o vetor de pesos, que (v1,…,vn) é o vetor de valores e que c é a capacidade do problema.

- Imagine que tenho n objetos que eu gostaria de colocar em uma mochila de capacidade c. Cada objeto i tem peso pi e valor vi. Posso escolher uma fração (entre 0% e 100%) de cada objeto para colocar na mochila. Quero fazer isso de modo a respeitar a capacidade da mochila e maximizar o seu valor.

O PROBLEMA DA MOCHILA FRACIONÁRIA

- O seguinte algoritmo guloso resolve o problema da mochila fracionária. O algoritmo exige que os dados estejam em ordem crescente de valor específico (ou seja, valor por unidade de peso)
- $v1/p1 \leq v2/p2 \leq \ldots \leq vn/pn$

```
MOCHILA-FRACIONÁRIA (p, v, n, c)
para j ← n decrescendo até 1 faça
    se p_j ≤ c
        então x_j ← 1
              c ← c − p_j
        senão x_j ← c/p_j
              c ← 0
devolva x
```

O PROBLEMA DA MOCHILA FRACIONÁRIA

- O algoritmo é guloso porque, em cada iteração, abocanha o objeto de maior valor específico dentre os disponíveis, sem se preocupar com o que vai acontecer depois. O algoritmo jamais se arrepende do valor atribuído a um componente de x.

- É evidente que o consumo de tempo do algoritmo é $O(n)$. (Isso não inclui o tempo $\Theta(n \log n)$ necessário para colocar os objetos em ordem crescente de valor específico.) Em outras palavras, o consumo de tempo é da mesma ordem de grandeza que o tempo gasto com a leitura dos dados. Portanto, o algoritmo é muito rápido.

PROBLEMA DO TROCO

- Imagine que você trabalha no caixa de um supermercado. Sempre que um cliente chega e paga sua compra, você deve entregar a ele o troco em moedas.
- Você particularmente gosta dessas moedas, e quer entregar o menor número de moedas possível ao cliente.
- Para um valor de troco N, e com um estoque infinito de cada uma das M moedas de diferentes valores m1, m2, …, mM, quais e quantas moedas você deve entregar ao cliente de modo que o total de moedas seja o mínimo possível?
- Resolva com um algoritmo guloso!!!

Extra

- Beecrowd:
  - 1034; 1054; 1055; 1084; 1086; 1222; 1524; 1590; 1594; 1637; 1643; 1661; 1966

FIM DA AULA 18

Próxima aula:

Programação Dinâmica