PAA - Aula 11
Algoritmos Gulosos e Problema do Escalonamento

ou gananciosos (greedy)

Algoritmos gulosos realizam, iterativamente, decisões míopes,

- i.e., aquelas que parecem imediatamente vantajosas.

Geralmente é fácil projetar algoritmos gulosos para um problema.

- Também costuma ser fácil analisar o tempo de execução desses algoritmos.

Mas não é fácil projetar um algoritmo guloso correto,

  - i.e., que sempre devolve a melhor solução.
- Tampouco é fácil provar tal corretude.

As provas de corretude geralmente são:

- Por indução no número de iterações,
  - mostrando que a escolha gulosa está correta a cada passo.
- Usando um argumento de troca entre soluções,
  - que pode ser por contradição,
    - para mostrar que algo diferente da solução gulosa é pior,
  - ou que iterativamente vai transformando uma solução qualquer
    - em uma solução gulosa sem piorar seu custo.

Problema do escalonamento em uma única máquina

Neste problema temos uma única máquina e diversas tarefas por realizar.

- Cada tarefa j tem um peso $w_j \geq 0$ e uma duração $l_j > 0$

  $\Sigma \rightarrow$ prioridade

Uma solução é uma ordem (permutação) das tarefas

- e o objetivo é encontrar uma solução que minimize
  - a soma ponderada dos tempos de término, i.e., $\sum_{j=1}^{n} w_j t_j$
- sendo t_j o tempo em que j é concluído.

f.o.

Note que t_j é a soma dos tempos de execução l_i

- de toda tarefa i que vem antes de j mais o próprio l_j

Exemplo: considere três tarefas com durações $l_1 = 1$, $l_2 = 2$ e $l_3 = 3$

- Quais seus tempos de término nos seguintes escalonamentos?

a) || #3 | #2 | #1 |

$t_1 = 6 \quad t_2 = 5 \quad t_3 = 3$

b) || #1 | #2 | #3 |

$t_1 = 1 \quad t_2 = 3 \quad t_3 = 6$

- Supondo que os pesos são $w_1 = 3$, $w_2 = 2$, $w_3 = 1$
  - qual o valor da função objetivo em cada caso?

a) $3 \cdot 6 + 2 \cdot 5 + 1 \cdot 3 = 31$

b) $3 \cdot 1 + 2 \cdot 3 + 1 \cdot 6 = 15$

Algoritmo guloso para escalonamento

Lembrando que queremos minimizar a soma ponderada dos tempos de término,

- i.e., $\sum_{j=1}^{n} w_j t_j$

- Vamos fazer um projeto a partir da generalização de casos particulares.

Casos particulares:

a. toda tarefa tem o mesmo peso,
b. toda tarefa tem a mesma duração.

Exemplos:

a. duas tarefas $\ell_1 = 1,\ \ell_2 = 2,\ w_1 = w_2 = 1$

$\boxed{\#1}\boxed{\#2}$ $1.1 + 1.3 = 4$ $\boxed{\#2}\boxed{\#1}$ $1.2 + 1.3 = 5$

b. duas tarefas $\ell_1 = \ell_2 = 1,\ w_1 = 1$ e $w_2 = 2$

$\boxed{\#1}\boxed{\#2}$ $1.1 + 2.2 = 5$ $\boxed{\#2}\boxed{\#1}$ $1.2 + 2.1 = 4$

→ menor duração

Qual seria um bom critério guloso em cada caso?

a. itens mais curtos antes, pois a duração de uma tarefa impacta
   - o tempo de término de toda tarefa que vem depois dela.
b. itens mais pesados antes, pois o peso multiplica o tempo de término.

Generalizando e resolvendo conflitos:

- Se uma tarefa é mais curta e mais pesada que outra, ela deve vir antes.
- Mas, o que fazer se uma tarefa i é mais curta e outra j é mais pesada,
  - i.e., $l_i < l_j$ e $w_j > w_i$?

Uma ideia é combinar duração e peso em uma única pontuação

- que aumenta com o peso e diminui com a duração,
  - e então escalonar as tarefas em ordem decrescente de pontuação.
- Quais funções atendem esses critérios? Duas possibilidades são:

$$f_1(i) = w_i - l_i \qquad\qquad f_2(i) = w_i / l_i$$

- Note que, no máximo um desses critérios está correto. Talvez nenhum esteja! !!

Vamos analisar um cenário em que eles tenham comportamento distinto entre si,

- para colocá-los a prova. São os chamados contra-exemplos.

$l_i = 1$ e $w_i = 3$ $\qquad\qquad$ $l_j = 2$ e $w_j = 5$

descartamos a f_1

$f_1(i) = 3 - 1 = 2$

$\wedge$

$f_1(j) = 5 - 2 = 3$

|| #j | #i |

$3.3 + 5.2 = 19$

$f_2(i) = 3/1 = 3$

$\vee$

$f_2(j) = 5/2 = 2.5$

|| #i | #j |

$3.1 + 5.3 = 18$

Como vimos, nossa escolhida foi a função $f(i) = w_i/l_i$

- que chamaremos de densidade.

- Mas, será que ela é correta, i.e., um algoritmo que escalona as tarefas
  - em ordem decrescente seguindo essa função
    - sempre devolve um escalonamento ótimo?
- Isso não é óbvio, mas nesse caso é verdade,
  - e nós vamos demonstrar!
- Note que, para esse problema, "ótimo" significa "de custo mínimo".

Antes da demonstração de corretude, vamos analisar a eficiência do algoritmo.

- Sendo n o número de tarefas, observe que o algoritmo
  - tem um laço para calcular a densidade das tarefas. $O(n)$
  - Então ele ordena as tarefas em ordem decrescente de densidade, $O(n \lg n)$
    - e esta ordem é a solução devolvida.
- Portanto, o algoritmo tem complexidade de tempo $O(n \lg n)$

Em questão de memória, a implementação mais simples

- usa um vetor de tamanho n para armazenar as densidades.
- No entanto, isso pode ser evitado se
  - inserirmos o cálculo da densidade nas funções de ordenação.
- Quiz: como fazer isso?

Prova de corretude: vamos fazer a prova usando um argumento de troca

- e a técnica de prova será por contradição.

Para simplificar, vamos supor que não ocorrem empates, i.e.,

- para todo par de tarefas i, j temos $f(i) \neq f(j)$ lembrando que $f(i) = w_i / l_i$

σ = escalonamento do nosso alg.

Por contradição, $\exists\, \sigma^* \neq \sigma : \sigma^*$ é ótimo

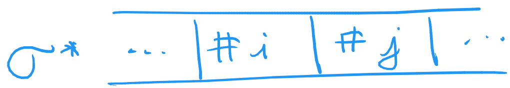
Pra simplificar, renomear tarefas seguindo σ,

i.e., $\frac{w_1}{l_1} > \frac{w_2}{l_2} > \frac{w_3}{l_3} > \dots > \frac{w_n}{l_n}$

Como $\sigma^* \neq \sigma$ existe uma inversão em σ^*,

i.e., $\exists\, i, j$ adjacentes em $\sigma^* : j < i$

e no σ^* temos i antes do j



Como $j < i$ temos $\frac{w_j}{l_j} > \frac{w_i}{l_i} \Rightarrow w_j l_i > w_i l_j$

Identificamos uma inversão em σ^*, i.e., $\sigma^* = \cdots \,|\, \#i \,|\, \#j \,|\, \cdots$ e $j < i$

O que acontece se invertermos i e j em σ^*?

$$\sigma^*_{\text{antes}} = \boxed{\cdots} \,|\, \#i \,|\, \#j \,|\, \boxed{\cdots} \xrightarrow[\text{de } i \text{ e } j]{\text{troca}} \sigma^*_{\text{depois}} = \boxed{\cdots} \,|\, \#j \,|\, \#i \,|\, \boxed{\cdots}$$

1°) nada muda p/ tarefas antes de i e j || 2°) nada muda p/ tarefas depois de i e j

$$c(\sigma^*_{\text{antes}}) = \sum_{\substack{k \text{ antes} \\ \text{de } i \text{ e } j}} w_k t_k + \sum_{\substack{k \text{ depois} \\ \text{de } i \text{ e } j}} w_k t_k + w_i t_i + w_j t_j$$

$$c(\sigma^*_{\text{depois}}) = \sum_{\substack{k \text{ antes} \\ \text{de } i \text{ e } j}} w_k t_k + \sum_{\substack{k \text{ depois} \\ \text{de } i \text{ e } j}} w_k t_k + w_i t'_i + w_j t'_j$$

$$t'_i = t_i + l_j$$

$$t'_j = t_j - l_i$$

$$\boxed{\Delta c(\sigma^*)} = c(\sigma^*_{\text{depois}}) - c(\sigma^*_{\text{antes}}) = w_i t'_i - w_i t_i + w_j t'_j - w_j t_j = \boxed{w_i l_j - w_j l_i}$$

$$\boxed{\Delta c(\sigma^*) = c(\sigma^*_{depois}) - c(\sigma^*_{antes}) = w_i l_j - w_j l_i} \quad *_1$$

$$\sigma^*_{antes} = \boxed{\cdots \mid \# i \mid \# j \mid \cdots} \xrightarrow[\text{de } i \text{ e } j]{\text{troca}} \sigma^*_{depois} = \boxed{\cdots \mid \# j \mid \# i \mid \cdots}$$

$$\frac{w_j}{l_j} > \frac{w_i}{l_i} \Rightarrow w_j l_i > w_i l_j \Rightarrow w_i l_j - w_j l_i < 0 \quad *_2$$

$*_1$ e $*_2 \Rightarrow \Delta c(\sigma^*) < 0$ o que é um absurdo ■

Se considerarmos que podem ocorrer empates de densidade, podemos usar o mesmo argumento de troca, mas a prova deixa de ser por contradição (já que uma desinversão nem sempre diminui o custo) e passa a ser por indução, mostrando que após um certo número de trocas saímos de σ^* e chegamos em σ sem aumentar o custo.