SCC-218 Algoritmos Avançados e Aplicações

Algoritmos Gulosos

Os sete pecados capitais

- Gula
- Avareza
- Luxúria
- Ira
- Inveja
- Orgulho
- Preguiça

- Gluttony
- Greed
- Lust
- Wrath
- Envy
- Pride
- Loth

Avareza (Greed) -> Um desejo insaciável por ganhos materiais (dinheiro, status, poder, <u>comida</u>).

Greedy Algorithm - Algoritmo Guloso

- Como definir um algoritmo guloso?
    - É praticamente impossível definir com precisão

Um algoritmo é guloso se este constrói uma solução em pequenos passos, tomando uma decisão "ótima" em cada passo, com o objetivo de atingir uma solução ótima globalmente.

Como saber se a estratégia gulosa funciona?

- Um problema deve exibir as seguintes propriedades
  1. **Ele tem estruturas sub-ótimas**: existe solução ótima para o problema se este contém soluções ótimas para os sub-problemas
  2. **Ele tem propriedade gulosa**: Se fizermos o que parece ser melhor naquele momento, terminaremos com a solução ótimas -> Nunca será preciso reconsiderar escolhas passadas !

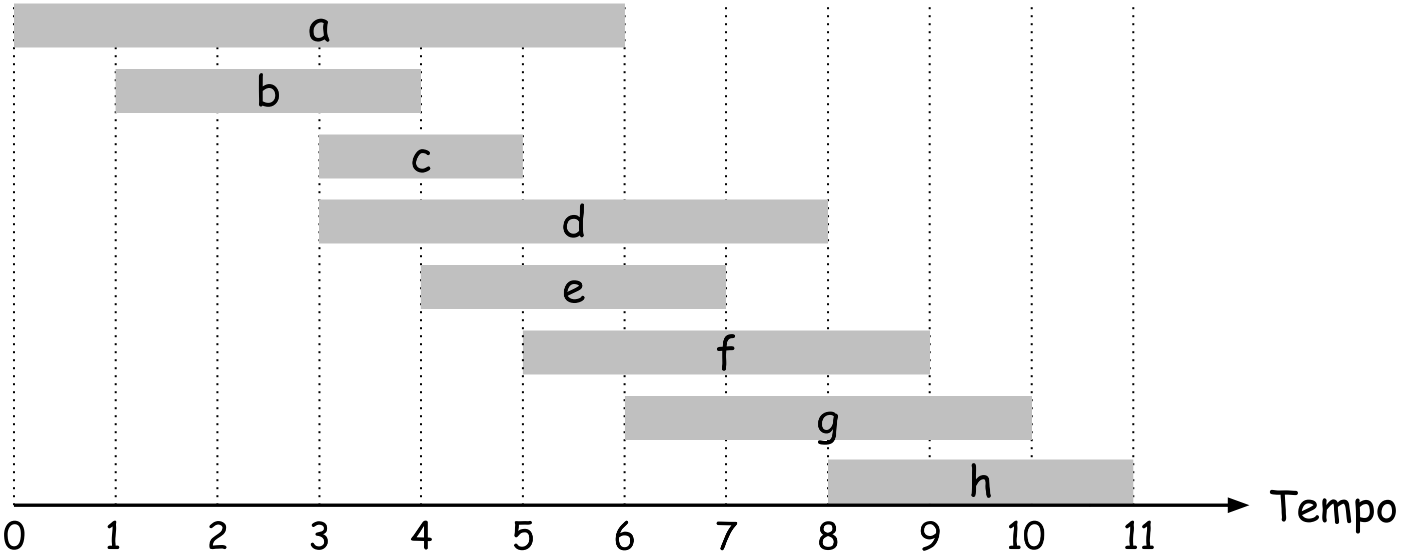
O problema das Moedas

- Dada uma quantia V e uma lista de n moedas, retorne o nro mínimo de moedas que representa V.
  - V = 42.
  - Moedas = {25, 10, 5, 1}
- 42-25 = 17 → 17-10 = 7 → 7-5 = 2 → 2-1 = 1 → 1-1 = 0
- Portanto, 5 moedas
  - **Sub-estruturas ótimas** → 42 = {25,10,5,1,1} : ótimo; 17 = {10,5,1,1} : 4 moedas (ótimo); 7 = {5,1,1}: 3 moedas..
  - **Propriedade Gulosa** → V - x = V' < V; Podemos provar que se "x" é a maior moeda do conjunto, ∄ outra estratégia melhor

Agendamento de Intervalos (Interval Scheduling)

Agendamento de Intervalos.

- Tarefa j começa em s_j e termina em f_j.
- Duas tarefas são compatíveis se não há sobreposição.
- Objetivo: encontre o subconjunto <u>máximo</u> de tarefas mutuamente compatíveis.



Agendamento de Intervalos: Algoritmos Gulosos

Modelo guloso. Considere tarefas em alguma ordem. Cada tarefa é escolhida obedecendo-se o mesmo critério utilizado nas escolhas prévias.

- [Tempo de início mais cedo] Considere tarefas em ordem ascendente de tempo de início s_j.
- [Tempo de fim mais cedo] Considere tarefas em ordem ascendente em tempo de fim f_j.
- [Menor intervalo] Considere tarefas em ordem ascendente de tamanho de intervalo $f_j - s_j$.
- [Menor número de conflitos] Para cada tarefa, conte o número de tarefas em conflito c_j. Agende em ordem ascendente de conflitos c_j.

Agendamento de Intervalos: Algoritmos Gulosos

Modelo guloso. Considere tarefas em alguma ordem. Cada tarefa é escolhida desde que seja compatível com as outras escolhidas previamente.

falha para:

Tempo de início mais cedo

Menor intervalo

Menor número de conflitos

Agendamento de Intervalo

Algoritmo guloso. Considere tarefas em ordem crescente de tempo de término. Cada tarefa é escolhida desde que seja compatível com as outras escolhidas previamente.

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

  jobs selected
A ← φ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

Implementação. O(n log n).

- Guarde a tarefa j* que foi adicionada por último em A.
- Tarefa j é compatível com A se $s_j \geq f_{j^*}$.

Agendamento de Intervalos: Análise

Teorema. Algoritmo guloso é ótimo (retorna A "ótimo").

Pf. (por contradição)

- Assuma que o algoritmo guloso não é ótimo →A tem menos elementos que um conjunto ótimo R ($m > k$). m é o nro de elementos em R e k o nro de elementos em A.

- Seja i_1, i_2, ... i_k o conjunto das tarefas <u>adicionadas</u> em A. (tempo de término em ordem crescente)

- Seja j_1, j_2, ... j_m o conjunto das tarefas em uma solução ótima.

- podemos afirmar que para todo índice $r \leq k$, $f(i_r) \leq f(j_r)$

    - Como $m > k$, existe j_{k+1} no conjunto ótimo

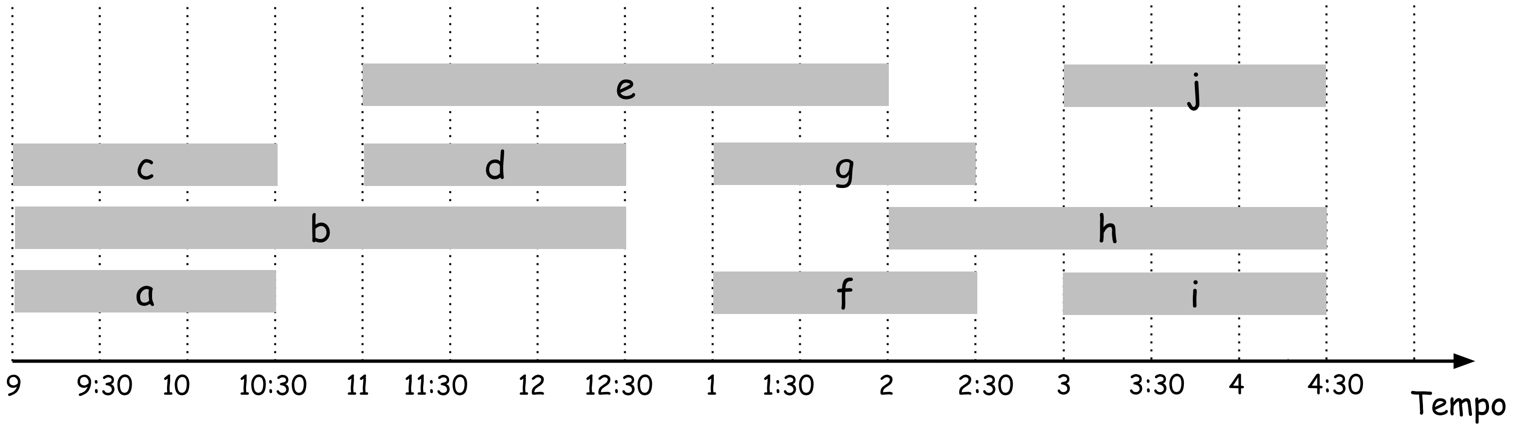
        - este tem início após o fim de ambos j_k e i_k
        - mas após apagar os intervalos não compatíveis com i_1, …, i_k o conjunto em R ainda contém j_{k+1}.
        - Mas o algoritmo guloso termina com o intervalo i_k, quando o conjunto R deveria estar vazio >>> contradição

Particionamento de Intervalos

Particionamento de intervalos.

- Palestra j começa em s_j e termina em f_j.
- Objetivo: encontrar um número mínimo de salas para agendar todas as palestras de forma que duas palestras não ocorram na mesma sala ao mesmo tempo

Ex: Esse agendamento usa 4 salas para agendar 10 palestras. Pode ser menos??

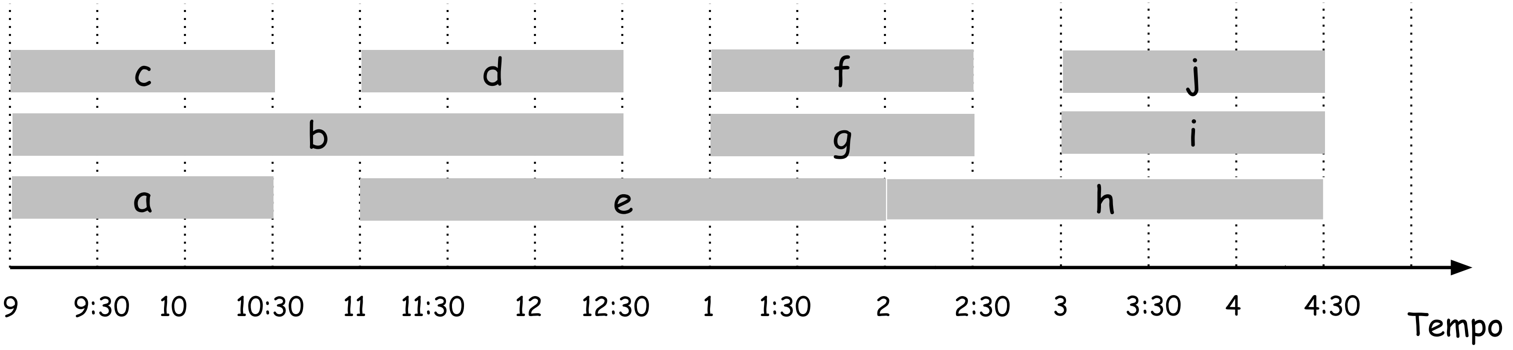


Particionamento de Intervalos

Particionamento de intervalos.

- Palestra j começa em s_j e termina em f_j.
- Objetivo: encontrar um número mínimo de salas para agendar todas as palestras de forma que duas palestras não ocorram na mesma sala ao mesmo tempo

Ex: Esse agendamento usa apenas 3 salas.



Particionamento de Intervalos: Limite inferior da solução ótima

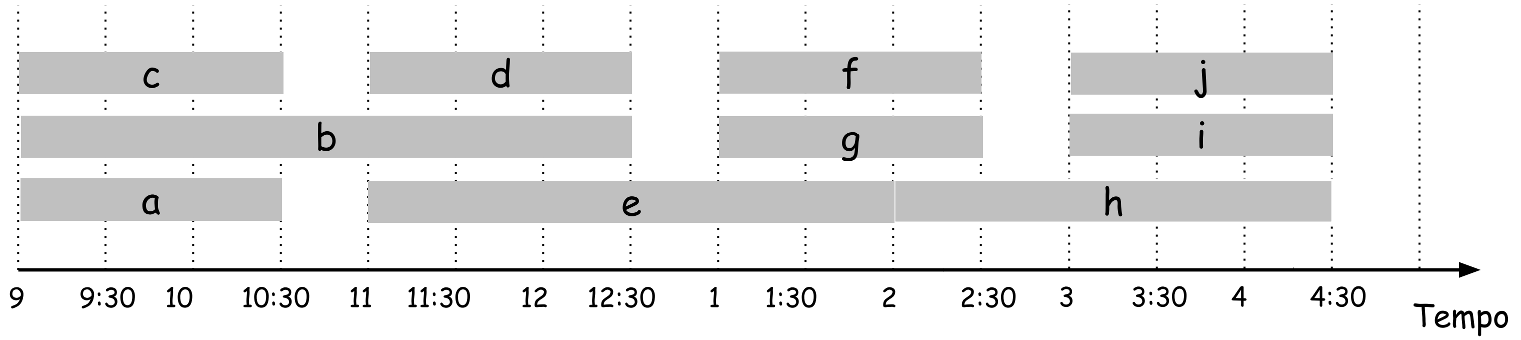
Def. A profundidade de um conjunto é o máximo nro de intervalos que se cruzam em algum ponto na linha do tempo.

Observação importante. Número de salas necessárias ≥ profundidade.

Ex: Profundidade do agendamento abaixo = 3 ⇒ solução ótima.

a, b, c todos se cruzam em 9:30hs

Q. Será que sempre existe um agendamento igual à profundidade dos intervalos?



Particionamento de Intervalos: Algoritmos Gulosos

Algoritmo guloso. Considere palestras em ordem crescente de tempo de início: atribua uma palestra para qualquer sala compatível.

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0   ← number of allocated
          classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementação. O(n log n).

- Para cada sala de aula k, mantenha o tempo final da última tarefa adicionada.
- Mantenha as salas em uma fila de prioridade. (sala com menor f_x fica na frente!)

Particionamento de Intervalos: Análise Gulosa

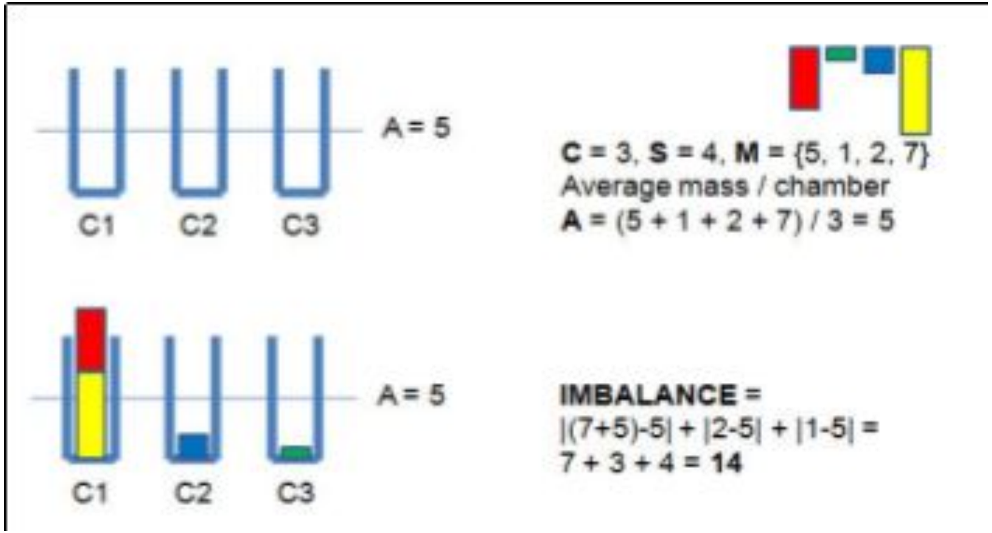
Observação. O algoritmo guloso nunca agenda duas palestras incompatíveis na mesma classe.

Teorema. O algoritmo guloso é ótimo.
Pf.

- Seja d = número de salas que o algoritmo guloso alocou (1..d).
- A sala d está aberta porque tivemos que agendar uma tarefa, digamos j, que era incompatível com todas as d-1 salas.
- Uma vez que nós ordenamos por tempo de início, todas essas incompatibilidades são causadas por palestras que começam antes de s_j.
- Então, temos d palestras sobrepondo-se no tempo $s_j + \varepsilon$.
- Observação importante $\Rightarrow$ todos os agendamentos usam $\geq$ d salas. ▪

Colocando as feras na jaula

- Sejam **C** jaulas, cada qual podendo armazenar 0, 1 ou 2 animais. Existem **S** animais ($1 \leq S \leq 2C$) e uma lista **M** das massas dos S animais. Determine qual jaula deve conter cada animal tal que o <u>desbalanceamento seja mínimo</u>

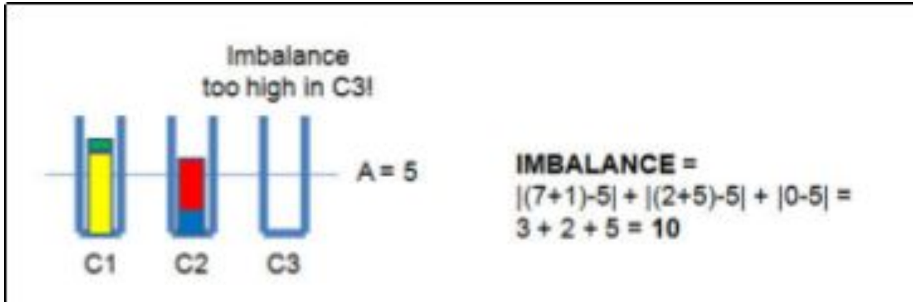


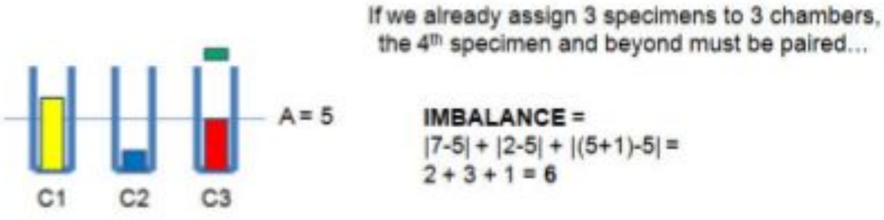
A é o valor médio esperado em cada uma das C jaulas.

O desbalanceamento é a soma das diferenças entre a massa total em cada jaula com relação a A.

Colocando as feras na jaula

- Existe um algoritmo guloso para este problema. Você consegue enxergá-lo?
  - obs 1: se houver uma jaula vazia será normalmente benéfico (e nunca pior) mover um animal de uma jaula com 2, para uma vazia. (jaula vazia aumenta o desbalanceamento)
  - obs 2: Se S > C, então S - C animais devem ser colocados aos pares numa jaula que já contém um animal (o princípio do escaninho!)

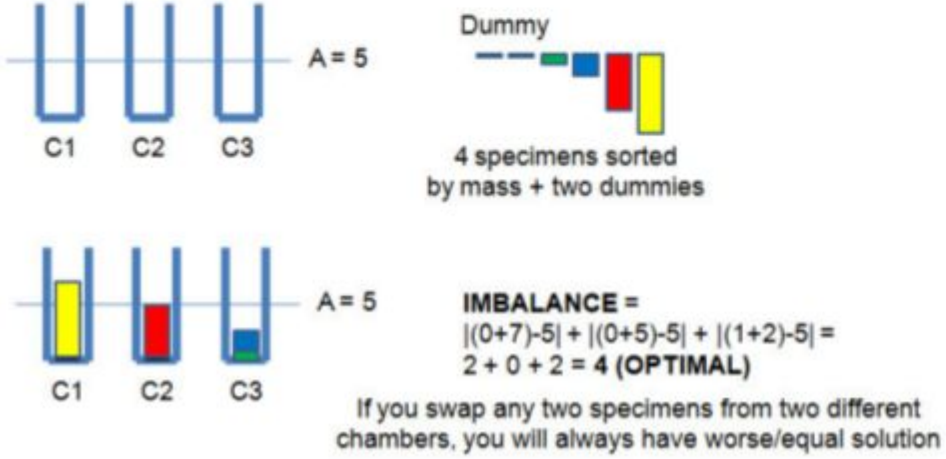




Colocando as feras na jaula

- Segredo está na ordenação !
  - se $S < 2C$, então adicione $2C - S$ animais dummies (de massa 0). Se $C = 3$ e $S = 4$, então crie 2 animais extras de massa 0, tal que $M = \{5,1,2.7\}$ seja $M = \{5,2,1,7,0,0\}$
  - ordene o conjunto: $M = \{0,0,1,2,5,7\}$
  - Temos finalmente uma estratégia gulosa neste ponto
    - forme um par em C_1 com as massas M_1 e M_{C2}
    - forme um par em C_2 com as massas M_2 e M_{C2-1}, e assim por diante
- Esta estratégia gulosa é conhecida por balanceamento de carga ou load balancing.
- Prove que isto funciona !

Colocando as feras na jaula



Compras de Casamento

- Dada uma relação de **I** itens (calça, sapato, blusa, etc) e uma verba **V** limitada, sua tarefa é comprar <u>um item de cada</u>, gastando o máximo possível de sua verba ! Cada item possui preços distintos. O problema pode não ter solução
  - Para V = 20, I = 3
    - item 0 → 6, 4, 8
    - item 1 → 5, 10
    - item 2 → 1, 5, 3, 5
  - Para V = 9, I = 3
    - item 0 → 6, 4, 8
    - item 1 → 5, 10
    - item 2 → 1, 5, 3, 5
  - Qual o primeiro algoritmo que lhe vem à mente?

Compras de casamento

- Se vc pensou o seguinte:
  - selecione, para cada item, aquele com o preço mais alto
  - subtraia da verba este item,
  - repita o processo para os itens restantes
  - Ao final, teremos gasto o máximo possível.
- Se pensou isto, então vc acabou de sugerir um algoritmo guloso:
  - um algoritmo é guloso se ele faz, localmente, a escolha ótima (→selecione o mais caro!) na **esperança** de que, ao final, chegue a uma solução global que seja ótima (→ o menor troco possível ou o máximo dinheiro gasto)
- Por que usamos o termo “esperança” ?
  - voltemos ao exemplo do slide anterior

Compra de casamento

- Exemplo 1
  - Para V = 20, I = 3
    - item 0 → 6, 4, 8
    - item 1 → 5, 10
    - item 2 → 1, 5, 3, 5
- Exemplo 2
  - Para V = 9, I = 3
    - item 0 → 6, 4, 8
    - item 1 → 5, 10
    - item 2 → 1, 5, 3, 5
      - "no solution"
- Exemplo 3
  - Para V = 12, I = 3
    - item 0 → 6, 4, 8
    - item 1 → 5, 10
    - item 2 → 1, 5, 3, 5
  - "no solution" ou 😞😞😞 ???

Algoritmo Guloso

- Analise os casos do slide anterior
- Vc acha que a solução gulosa para este caso é adequada?
  a. para V=20, ele funciona perfeitamente…
  b. para V=9, ele também funciona. Indica que não há solução e, de fato, não há
  c. aqui ele falha retumbantemente. Vai dizer que não tem solução, mas têm
- Programação Dinâmica… (assunto de outras aulas…)