Algoritmos Gulosos

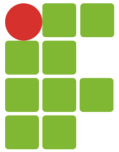
Tópicos Especiais em Algoritmos - Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga

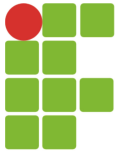
Sumário

1 Introdução



Algoritmos Gulosos

- Um algoritmo é dito **guloso** se ele faz uma escolha local esperando chegar na solução ótima global.
- Em alguns casos os algoritmos gulosos funcionam bem: a solução é curta e eficiente.
- Em muitos outros casos, a solução gulosa não funciona.

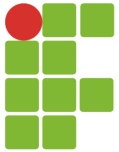


Algoritmos Gulosos

Correção de Algoritmos Gulosos

Para que um problema admita uma solução gulosa ele deve possuir a seguinte propriedade:

- Possui subestrutura ótima: a solução ótima do problema pode ser decomposta em soluções ótimas de subproblemas.
- Ele tem que ter a propriedade gulosa: ao realizar uma escolha gulosa e proceder a resolver o problema que restou utilizando a mesma estratégia, chega-se na solução ótima.



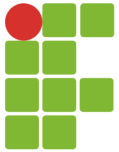
Algoritmos Gulosos

O Problema do Troco

Para contextualizar os conceitos, vamos falar de um problema clássico, *o problema do troco*.

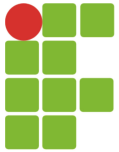
- Entrada: um sistema de moedas $C = (c_0, c_1, \ldots, c_{n-1})$ e um valor de troco a ser pago V.
- Saída: o menor número de moedas necessário para pagar o valor V.

Observação: existe um número ilimitado de moedas de cada tipo.



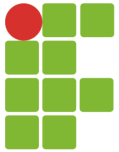
O Problema do Troco: Estratégia Gulosa

- Uma estratégia gulosa para resolver este problema é a seguinte:
  - Utilize a moeda com maior valor possível que é menor ou igual do que a quantia V a ser paga. Suponha que essa moeda seja c_j.
  - Utilize a mesma estratégia para o subproblema de valor $V - c_j$
- Por exemplo se $C = (25, 10, 5, 1)$ e $V = 42$ a estratégia seria:
  - $V = 42$: utilize uma moeda de 25, resta 17.
  - $V = 17$: utilize uma moeda de 10, resta 7.
  - $V = 7$: utilize uma moeda de 5, resta 2.
  - $V = 2$: utilize uma moeda de 1, resta 1.
  - $V = 1$: utilize uma moeda de 1.



O Problema do Troco: Estratégia Gulosa

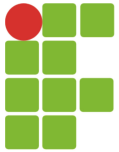
- Esta estratégia, para este sistema de moedas, alcança a solução ótima.
- Para $V = 42$, utilizamos um total de 5 moedas.



O Problema do Troco: Estratégia Gulosa

Analisando melhor o problema para este sistema de moedas, temos que:

- Ele tem subestrutura ótima: para resolver o problema com $V = 42$, utilizamos a solução ótima da solução do subproblema $V = 17$, que considera as moedas $10, 5, 1, 1$.
- Ele tem a propriedade gulosa: para cada quantidade V, podemos escolher a maior moeda que é menor ou igual a V que, ao resolver o problema que restou da mesma forma, chegaremos em uma solução ótima global.



O Problema Do Troco: Estratégia Gulosa

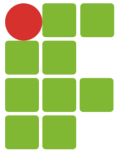
Algorithm 1: GREEDY-COIN-CHANGE(C,V)

Input: $C[0, n-1]$,V

Output: Número mínimo de moedas para pagar V

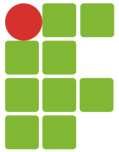
```
coin ← 0
SORT(C)
for( i ← n − 1; i ≥ 0; i − − )
    coin ← coin + ⌊V/C[i]⌋
    V ← V mod C[i]
return coin
```

- Complexidade: $\Theta(n \lg n)$ ou $\Theta(n)$ se C estiver ordenado.



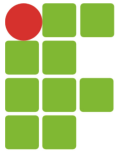
O Problema do Troco: Estratégia Gulosa

- A estratégia gulosa nos fornece uma solução muito simples, mas ela só funciona com um **sistema canônico** de moedas, que é o caso da maioria dos sistemas financeiros.
- Caso $C = (4, 3, 1)$ e $V = 6$, a estratégia gulosa nos daria como solução as moedas $4, 1, 1$, enquanto a solução ótima consiste de duas moedas de 3.
- Nem sempre uma solução gulosa vai funcionar para qualquer problema.



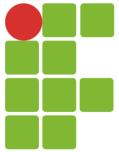
Algoritmos Gulosos

- Comentaremos em seguida uma série de problemas que permitem uma solução gulosa.



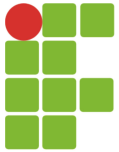
Sumário

2 Problemas



Sumário

2 Problemas
- UVa 410
- UVa 10382



UVa 410: Station Balance

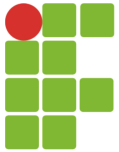
- Suponha que existam $1 \leq c \leq 5$ câmaras, em que cada qual pode armazenar 0, 1 ou 2 itens.
- Ao todo existem $1 \leq n \leq 2c$ itens que devem ser inseridos nas câmaras, cada qual com a sua respectiva massa $M = (m_0, \ldots m_{n-1})$.
- Tome A como a média das massas, isto é:

$$A = \frac{\sum_{i=0}^{n-1} M[i]}{n}$$

- A métrica de **desequilíbrio** é dada por:

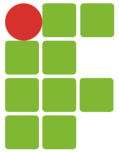
$$\sum_{i=1}^{c} |X[i] - A|$$

em que $X[i]$ é a soma das massas dos itens que estão na câmara i.



UVa 410: Station Balance

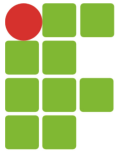
- A solução para este problema pode ser aplicada em um contexto muito prático: balanceamento de carga em servidores.
- As massas podem ser vistas como os recursos exigidos por cada máquina virtual enquanto as câmaras podem corresponder aos recursos físicos disponíveis.



UVa 410: Station Balance

O problema do balanceamento de carga consiste em, dados c e M, encontrar a disposição dos itens nas câmaras que minimiza o *desequilíbrio*.

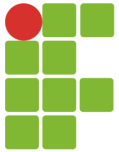
- Entrada: $1 \leq c \leq 5$, $M = (m_0, \ldots, m_{n-1})$ com $1 \leq n \leq 2c$.
- Saída: a disposição dos itens na câmara que configuram o menor desequilíbrio e o valor de desequilíbrio propriamente dito.



UVa 410: Station Balance

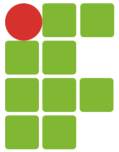
Observações

- É interessante não deixar câmaras vazias. Caso existam câmaras vazias, é sempre interessante tomar um item de uma câmara com 2 itens e inserí-lo em uma câmara vazia.
- Se $n > c$, existirão câmaras com mais de um item.



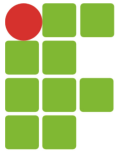
UVa 410: Station Balance

- A estratégia gulosa é simples: caso uma câmara deva acomodar mais de 1 item, este item deve ser pareado com outro item de modo que a soma das massas destes dois itens esteja o mais próximo possível da média das massas.



UVa 410: Station Balance

- Para facilitar a solução, se $n < 2c$, iremos adicionar itens de massa nula até que alcançemos os $2c$ itens.
- Ordenaremos as espécimes pela massa.
- Para cada câmara i, inseriremos nesta câmara os itens de massa m_i e m_{2c-i-1}.



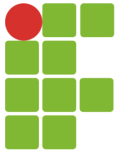
UVa 410: Station Balance

Algorithm 2: GREEDY-LOAD-BALANCE(M, c)

Input: $M[0, n-1], c$

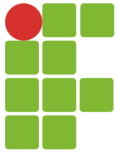
Output: A disposição dos itens que minimiza o desequilíbrio e o valor de desequilíbrio propriamente dito

```
M.RESIZE(2c)
for( i ← n; i < 2c; i++ )
    M[i] ← 0
SORT(M) // Tempo Θ(c lg c)
for( i ← 0; i < c; i++ )
    C[i].PUSH(M[i])
    C[i].PUSH(M[2c − i − 1])
imbalance ← CALC-IMBALANCE(C,M) // Tempo Θ(c)
return (C, imbalance)
```



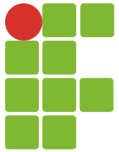
UVa 410: Station Balance

- Complexidade de pior caso: $\Theta(c \lg c)$.



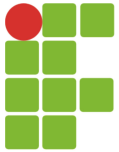
UVa 410: Station Balance

- Uma dica que funcionou neste problema e que pode ser aplicada em muitos outros é a ordenação.
- Ordenar a entrada pode ajudar a resolver o problema.



Sumário

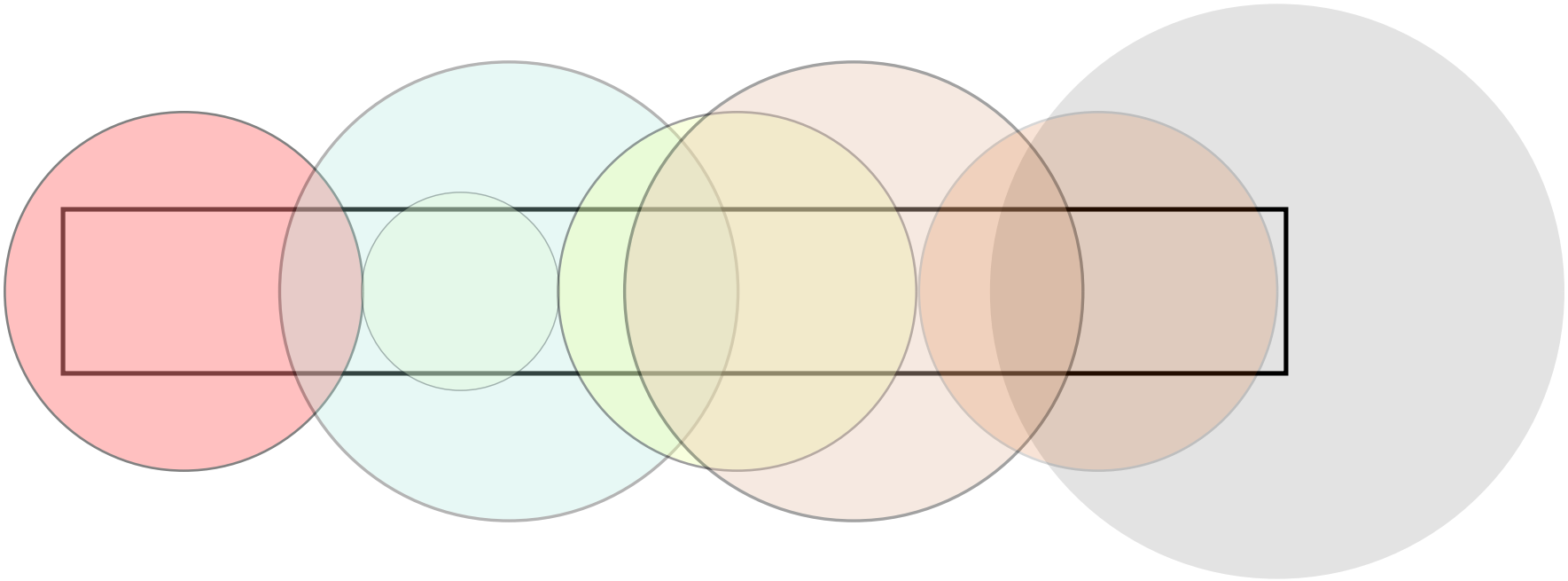
2 Problemas
- UVa 410
- UVa 10382

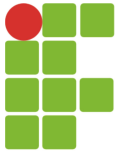


UVa 10382: Watering Glass

- Suponha uma faixa de grama retangular, de comprimento l e largura w.
- Suponha n irrigadores, cada qual com o um raio $R = (r_0, \ldots, r_{n-1})$ de irrigação.
- Cada irrigador é instalado na linha que divide o retângulo horizontalmente em uma determinada posição p_i, $0 \leq i < n$ à direita do início da faixa de grama.
- Qual o mínimo de irrigadores que devem ser utilizados para irrigar a faixa de grama?

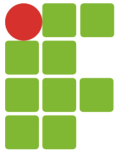
UVa 10382: Watering Glass





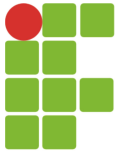
UVa 10382: Watering Glass

- Entrada: l, w, R, P.
- Saída: número mínimo de irrigadores para irrigar a faixa retangular $l \times w$.



UVa 10382: Watering Glass

- Como o número de irrigadores $n \leq 10^4$, uma abordagem baseada em busca completa não vai funcionar.
- Temos 2^{10000} subconjuntos possíveis de irrigadores. Inviável.

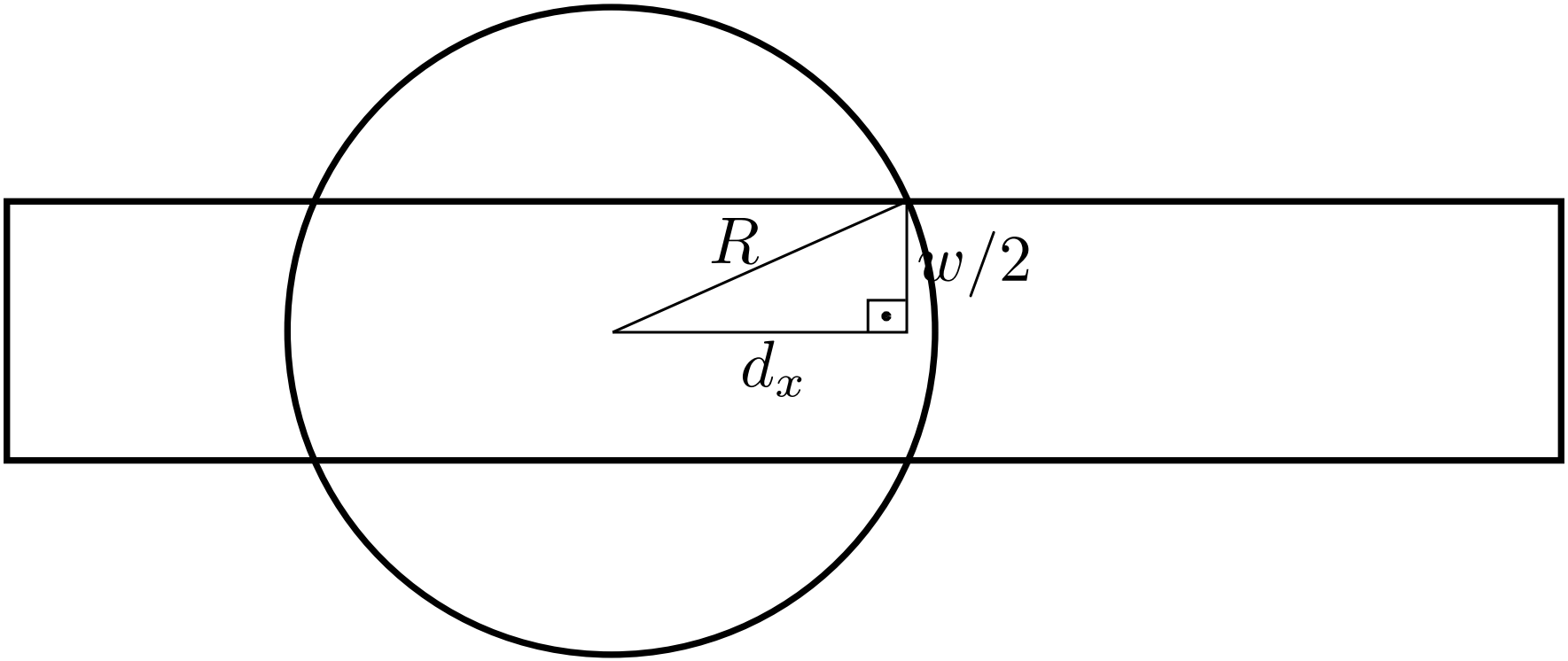


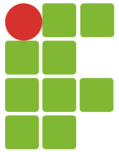
UVa 10382: Watering Glass

- Fazendo uma análise trigonométrica, é possível que cada irrigador com centro à uma distância p_i do início da faixa de grama, cobre o intervalo $[p_i - dx_i, p_i + dx_i]$ da faixa de grama, em que:

$$dx_i = \sqrt{r_i^2 - \left(\frac{w}{2}\right)^2}$$

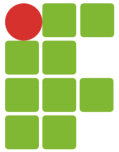
UVa 10382: Watering Glass





UVa 10382: Watering Glass

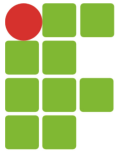
- Transformamos um problema geométrico em um problema mais simples: o de interseção de subintervalos.



UVa 10382: Watering Glass

Estratégia Gulosa

- Calcular os intervalos $[p_i - dx_i, p_i + dx_i]$ de todos os círculos.
- Ordenar os intervalos pelo início e, em caso de empate, em ordem decrescente de término.
- Insira o primeiro intervalo na solução.
- Para cada intervalo ou enquanto a faixa de grama não tiver sido coberta, pegue aquele intervalo mais distante da origem de modo que não haja uma descontinuidade de cobertura da faixa da grama e o inclua na solução. Caso não exista tal intervalo, não existe solução.
- **Observação:** círculos cujo raio é inferior a $w/2$ devem ser desconsiderados.



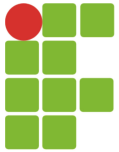
UVa 10382: Watering Glass

Estratégia Gulosa

- Complexidade: $\Theta(n \lg n)$.

Sumário

3 Considerações Finais



Considerações Finais

- Provar a correção da solução gulosa leva tempo e nem sempre é viável caso você não possua este recurso.
- Se o tamanho da entrada for suficientemente pequena para acomodar uma solução em busca completa ou de programação dinâmica, utilize elas em vez da solução gulosa, uma vez que esta nem sempre se aplica a todos os problemas.
- Ordenar a entrada pode ajudar a pensar na escolha gulosa.
- Algoritmos gulosos normalmente são mais simples e eficientes em contrapartida aos elaborados usando outra abordagem.