Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

Algoritmos gulosos

DEVSUPERIOR

https://devsuperior.com.br

1

Algoritmo Guloso

https://devsuperior.com.br

Prof. Dr. Nelio Alves

2

Algumas estratégias de solução de problemas…

- Busca Completa
- Dividir e Conquistar
- Programação Dinâmica
- Algoritmo Guloso

Cada uma tem suas vantagens e desvantagens, e serão melhores dependendo do contexto do problema.

Por que esse nome?

Em inglês, essa técnica se chama “Greedy Algorithm”.

Os Sete Pecados Capitais

1. Lust	1. Luxúria
2. Gluttony	2. Gula
3. Greed	3. Avareza
4. Sloth	4. Preguiça
5. Wrath	5. Ira
6. Envy	6. Inveja
7. Pride	7. Orgulho

Algoritmo Guloso - Greedy Algorithm

Definição

É um algoritmo que resolve o problema em pequenos passos, fazendo a escolha que parece ser a melhor no momento, esperando que se atinja uma solução ótima globalmente.

- Problemas de otimização, onde queremos maximizar ou minimizar um valor.
- Nunca volta e refaz uma decisão.

Quando podemos usar algoritmos Gulosos?

Um problema deve ter as seguintes propriedades para que o algoritmo funcione:

1. Subestruturas ótimas

A solução ótima para o problema contém soluções ótimas para os sub-problemas.

2. Propriedade gulosa

Fazendo a escolha que parece ser melhor no momento, você chegará em uma solução ótima. Nunca será preciso reconsiderar escolhas passadas.

Problema exemplo

Dada uma lista de números, diga qual é a maior soma possível, tomando K números desse conjunto.

[~~3~~, ~~6~~, -1, ~~5~~, -7, 2]
k = 3

Abordagem Gulosa: escolhendo um número a cada passo, qual parece melhor?

k = 1
soma = 6

k = 2
soma = 6 + 5 = 11

k = 3
soma = 6 + 5 + 3 = 14

Por que é uma boa solução gulosa?

1. Subestruturas ótimas

k = 1
soma = 6 (ótimo)

k = 2
soma = 6 + 5 = 11 (ótimo)

k = 3
soma = 6 + 5 + 3 = 14
(ótimo)

2. Propriedade gulosa

Conseguimos maximizar a soma ao fim apenas escolhendo o melhor número a cada passo.

Algoritmo Guloso - Problemas Clássicos

https://devsuperior.com.br

Prof. Dr. Nelio Alves

O problema das moedas (Coin Change Problem)

Dado um valor V e um conjunto de N moedas, ache o número mínimo de moedas que representa V. Considere que sempre é possível representar V com o conjunto, e que podemos tomar um número infinito de moedas.

Guloso: Como escolher a melhor moeda a cada passo?

V = 37
[25, 10, 5, 1]

Passo 1
37 - 25 = 12

Passo 2
12 - 10 = 2

Passo 3
2 - 1 = 1

Passo 4
1 - 1 = 0

37 = {25, 10, 1, 1}

4 moedas

Por que funciona?

1. Subestruturas ótimas

- 37 = {25, 10, 1, 1} (ótimo)
- 12 = {10, 1, 1} (ótimo)
- 2 = {1, 1} (ótimo)

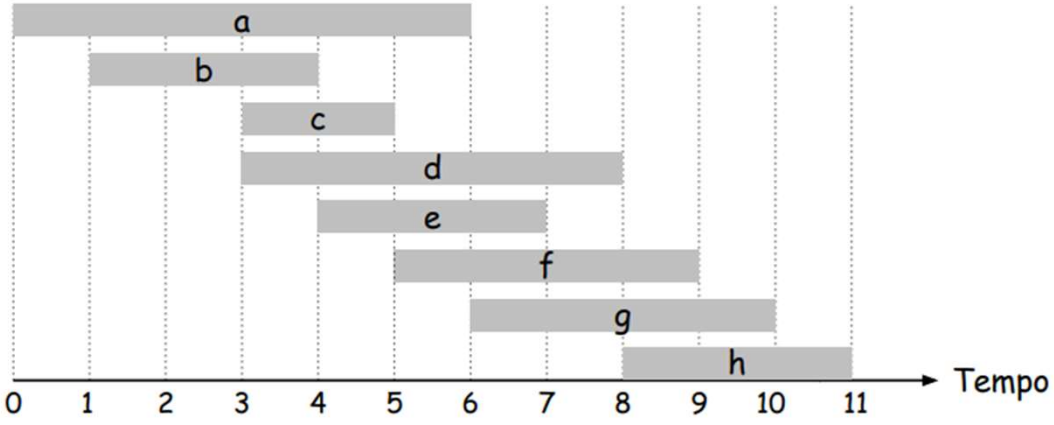
2. Propriedade gulosa

- É possível provar que, a cada passo, escolher a maior moeda é a melhor decisão.

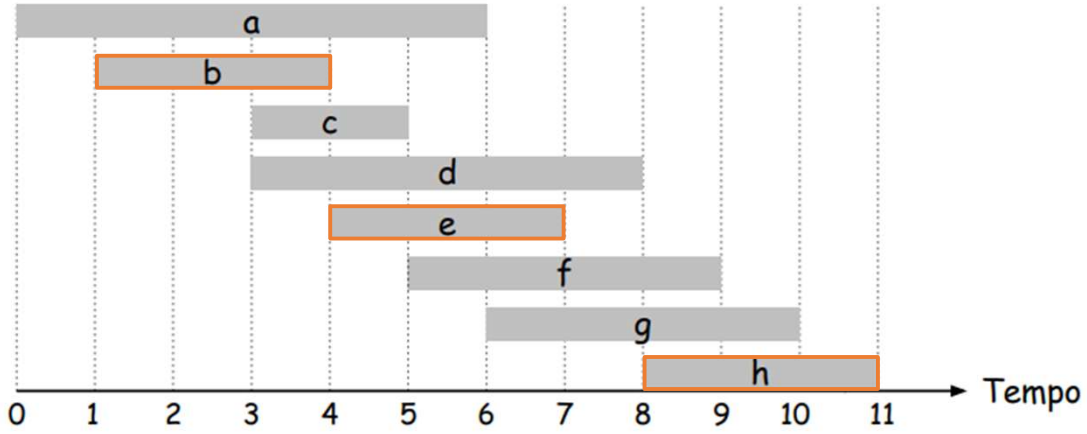
OBS: Para outro conjunto, pode não ser verdade!

Agendamento de Intervalos (Interval Scheduling)

Suponha que tenhamos N tarefas para realizar, cada uma com um tempo de começo e um tempo de fim. Ao começar uma tarefa, a executamos até o seu fim, e só então podemos começar outra. Qual o número máximo de tarefas que é possível realizar?



Qual o máximo de tarefas que é possível realizar?



13

Guloso: A cada momento, que tarefa devemos escolher?

O algoritmo falhará para:

- A que começa mais cedo

- A mais curta

- A que possui menos conflitos

14

Algoritmo guloso. Considere as tarefas em ordem crescente de **tempo de término**. Uma tarefa é selecionada se não tem sobreposição com outra selecionada anteriormente.

Implementação. O(nlogn)

1. Ordenar tarefas crescentemente por tempo de término
2. Percorrer tarefas nessa ordem
3. Guardar última tarefa **i** adicionada na solução
4. Tarefa **j** da lista é adicionada se **comeco_ j > fim_i**

Para os propósitos dessa aula, omitiremos a prova.

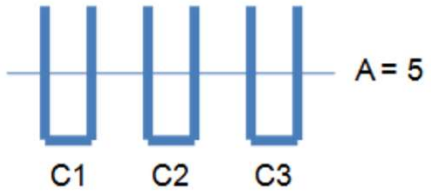
Colocando as feras na jaula

Suponha que existem C (1 ≤ C ≤ 5) jaulas que podem abrigar 0, 1 ou 2 feras, S (1 ≤ S ≤ 2C) feras e uma lista M das massas das S feras. Determine qual jaula deve conter cada fera para minimizar o 'desbalanceamento'.

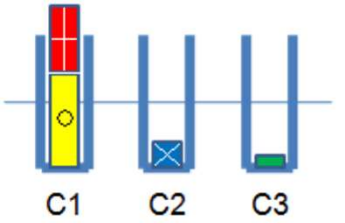
$A = (\sum_{j=1}^{S} M_j)/C$, a média de massa nas jaulas

$\text{Imbalance} = \sum_{i=1}^{C} |X_i - A|$, a soma das diferenças entre a massa total de cada jaula e a média, Xi é a massa na jaula i

Exemplo:



C = 3, **S** = 4, **M** = {5, 1, 2, 7}
Average mass / chamber
A = (5 + 1 + 2 + 7) / 3 = 5

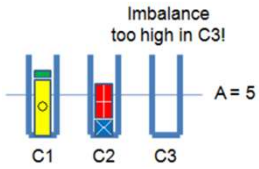


IMBALANCE =
|(7+5)-5| + |2-5| + |1-5| =
7 + 3 + 4 = **14**

Imagens do livro Competitive Programming 3, Halim

17

Como formular algoritmo guloso para o problema?



IMBALANCE =
|(7+1)-5| + |(2+5)-5| + |0-5| =
3 + 2 + 5 = **10**

Situação 1: muito desbalanceado em C3

IMBALANCE =
|7-5| + |2-5| + |(5+1)-5| =
2 + 3 + 1 = **6**

Situação 2: já existem 3 feras em 3 jaulas, devemos escolher onde alocar a 4

Algumas observações:

1. Se existe jaula vazia, nunca é ruim mover uma fera de uma jaula com 2 feras para a jaula vazia. Jaula vazia aumenta desbalanceamento.
2. Se S > C, então S - C feras devem ser colocadas em pares em uma jaula que já contém um animal. (Princípio da Casa dos Pombos)

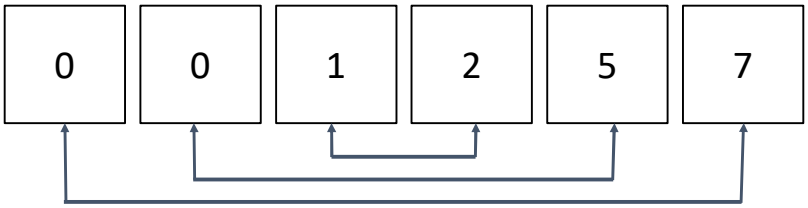
Imagens do livro Competitive Programming 3, Halim

18

Como formular algoritmo guloso para o problema?

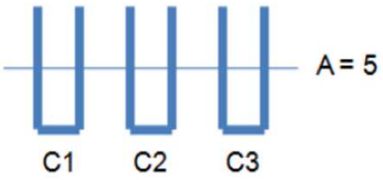
A principal ideia é que a solução pode ser simplificada com **ordenação**!

1. Se S < 2C, adicione 2C - S feras fictícias com massa 0.
2. Se C = 3, S = 4 e M = {5, 1, 2, 7}, adicionamos duas feras extras de massa 0, ficando com M = {5, 1, 2, 7, 0, 0}
3. Ordenamos o conjunto, M = {0, 0, 1, 2, 5, 7}
4. Pareamos as feras da seguinte forma:



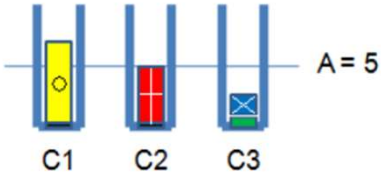
Para os propósitos dessa aula, omitiremos a prova.

Exemplo de solução ótima



Extras

4 feras ordenadas por massa
+ 2 extras



IMBALANCE =
|(0+7)-5| + |(0+5)-5| + |(1+2)-5| =
2 + 0 + 2 = **4 (OPTIMAL)**

Trocando quaisquer duas feras de jaulas diferentes, a solução sempre será igual ou pior

Imagens do livro Competitive Programming 3, Halim

Algoritmo Guloso - Revisão e Panorama

https://devsuperior.com.br

Prof. Dr. Nelio Alves

O que é um algoritmo guloso?

- Técnica que busca resolver o problema tomando a melhor decisão a cada passo, sem voltar atrás.
- Usado em problemas de maximização e minimização.
- Temos uma função objetivo para maximizar/minimizar:
  - Maximizar lucro
  - Minimizar distância até um local
  - Minimizar desbalanceamento

Vantagens

- **Simplicidade:** fácil de entender e implementar
- **Eficiência:** geralmente complexidade de tempo linear
- **Pouca memória:** já que tomam decisões pensando apenas no estado atual
- **Aplicabilidade:** ampla aplicação em diversos problemas

Desvantagens

- **Corretude:** nem sempre dá a melhor resposta
- **Dependência da estrutura:** precisa exibir propriedade gulosa para ser ótimo
- **Difícil de provar corretude:** desafiador provar que sempre dá a resposta correta

Trade-offs de um Algoritmo Guloso

i. Otimalidade vs. Eficiência

Escolhas ótimas a cada passo podem não levar a uma otimização global. Porém, a simplicidade e eficiência os tornam atrativos em certos cenários.

ii. Simplicidade vs. Acurácia

A natureza direta dos algoritmos gulosos os tornam fáceis de entender e implementar. Porém, a simplificação pode dar uma solução aproximada em vez de exata.

iii. Sem Backtracking vs. Busca Exaustiva

A falta de backtracking simplifica seu design, mas pode não dar a melhor solução. Métodos exaustivos podem explorar mais opções mas são caros computacionalmente.

Trade-offs de um Algoritmo Guloso

iv. Otimização Local vs. Otimização Global

Otimização local melhora a solução a cada passo, mas não garante a melhor solução global. Otimização global precisa considerar todo o espaço de solução, o que podem ser caro computacionalmente.

v. Soluções Exatas vs. Heurísticas

Heurísticas priorizam achar uma boa solução rápido, sendo apropriados para problemas de larga escala. No entanto, isso vem ao custo de não se garantir a melhor solução.

vi. Propósito geral vs. Específico para o problema

Alguns algoritmos são feitos para estruturas de problemas específicos e podem não ser bom em outros contextos. É preciso muito cuidado para adaptar e escolher algoritmos baseado nas características do problema.

Problemas reais e algoritmos famosos

Algoritmo de Dijkstra e a internet

Busca achar o menor caminho em uma rede, e pode ser usado em comunicações globais para achar o menor caminho para os pacotes de dados.

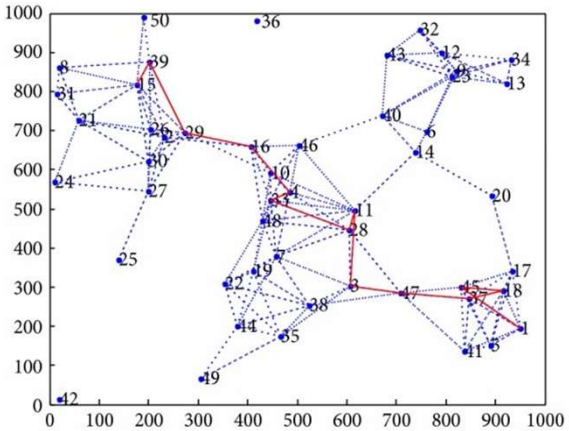
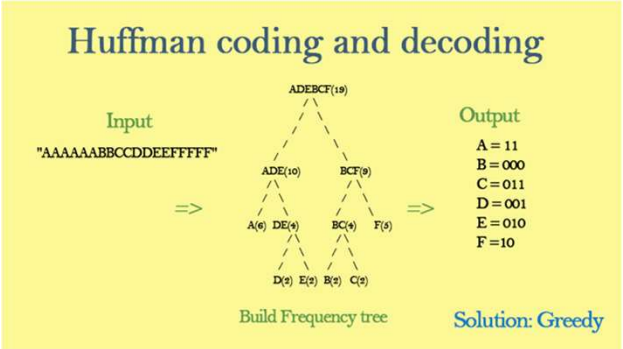


Imagem:Menor caminho em uma rede

Problemas reais e algoritmos famosos

Compressão de Dados com o Código de Huffman

Consegue representar a mesma informação com menos memória, a tornando mais eficiente de ser transportada.



"ABC" ➡ 3 Byte ➡ 24 bits

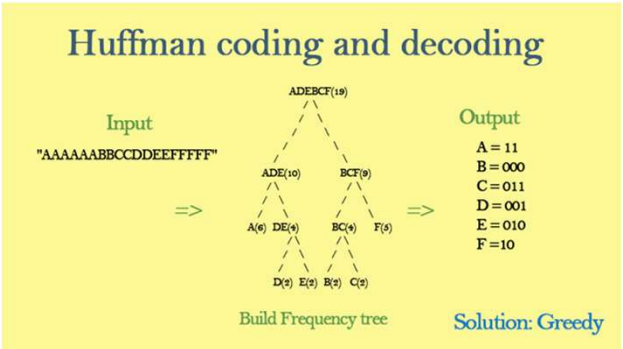
A : 00 B : 01 C : 10 ➡ 000110 ➡ 6 bits

27

Problemas reais e algoritmos famosos

Compressão de Dados com o Código de Huffman

Consegue representar a mesma informação com menos memória, a tornando mais eficiente de ser transportada.



"ABC" ➡ 3 Byte ➡ 24 bits

A : 00 B : 01 C : 10 ➡ 000110 ➡ 6 bits

28