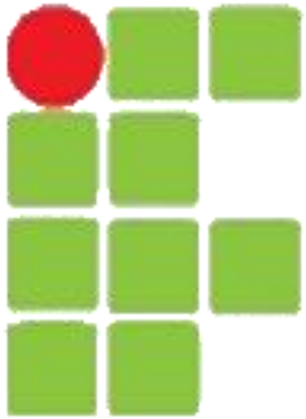


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA

Apresentação da disciplina

Mirko Perkusich

IFPB - Campus Monteiro

Curso: Análise e Desenvolvimento de Sistemas

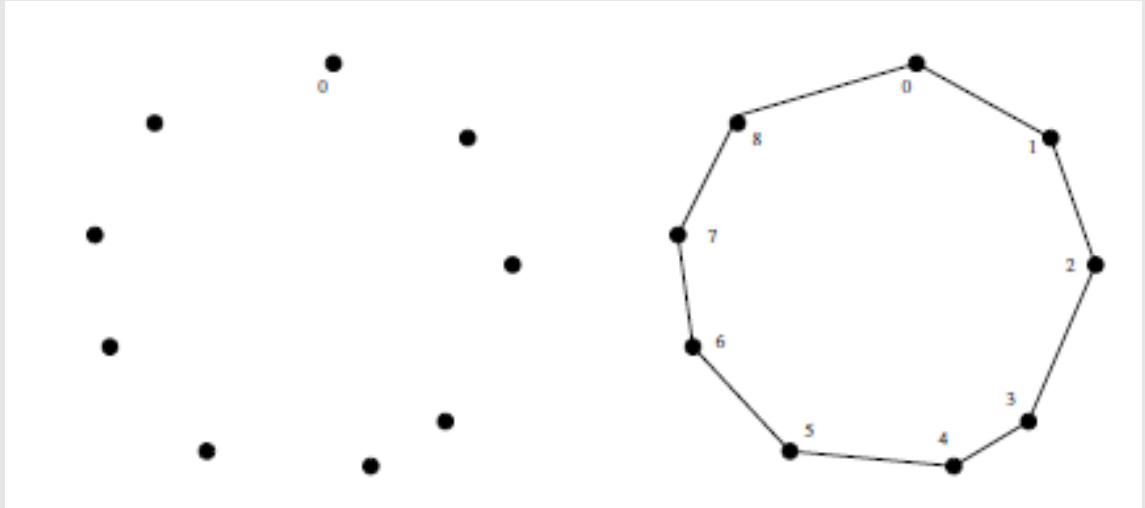
Disciplina: Análise de Algoritmos

Objetivos desta aula

Motivação

- Otimização da rota de um robô
  - Em uma indústria, há um braço robótico responsável por soldar peças em placas de circuitos. Robôs são caros. Então, precisamos minimizar o tempo para a soldagem ser executada. Para simplificar, vamos assumir que o tempo necessário para se movimentar entre dois pontos seja proporcional à distância entre os mesmos.

Problema: Otimização da rota de um robô
Entrada: Um conjunto S de pontos em um plano
Saída: Qual o caminho mais curto de visita a todos os pontos em S?

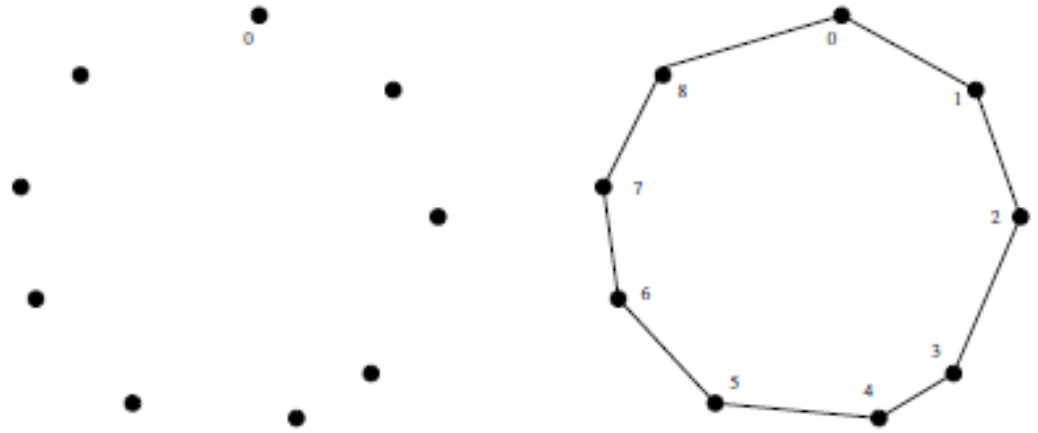


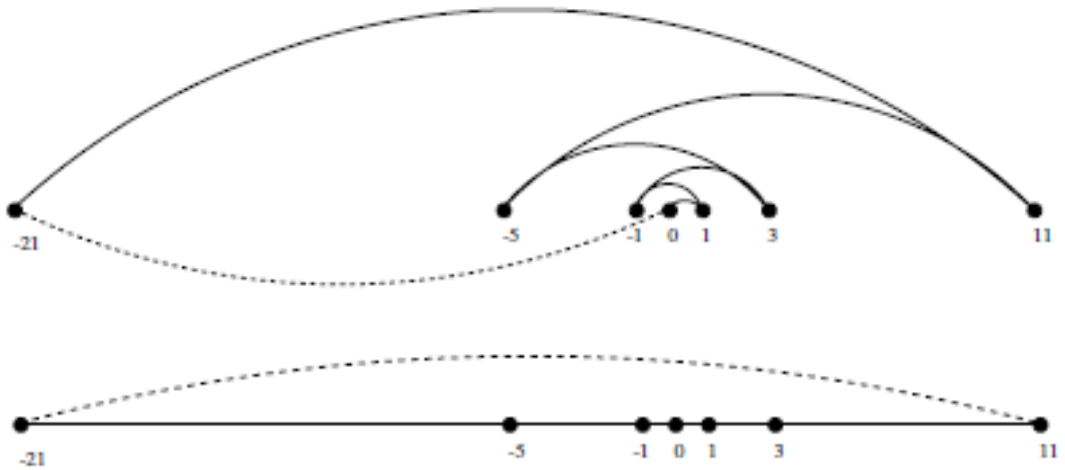
Exemplo de instância para o problema.

Solução

- Heurística de “vizinho mais próximo”
  - Começando em um ponto *p0*, partir para o ponto mais próximo *p1*. De *p1*, partir para o mais próximo de *p1*, excluindo *p0* como possível candidato. Repetir o processo até que não haja mais pontos não visitados, onde retornamos para *p0* fechando o *tour*.

Resultado

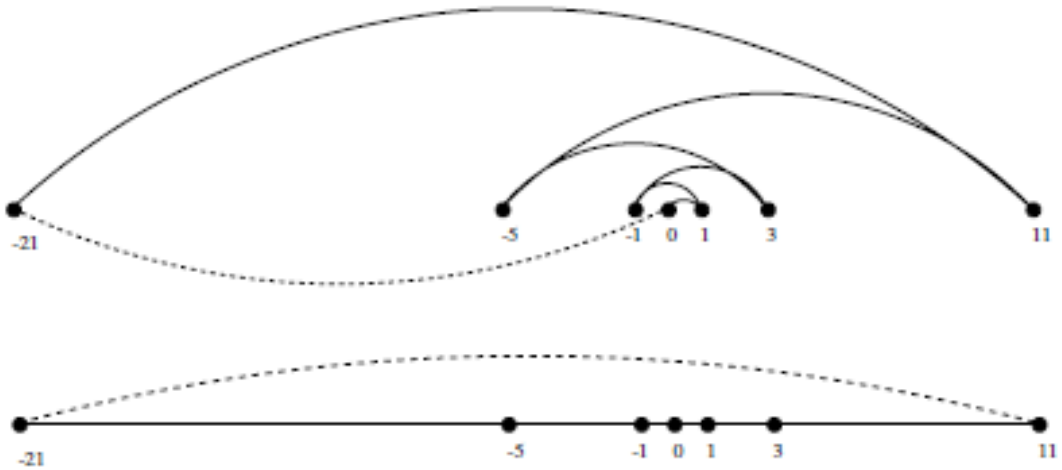


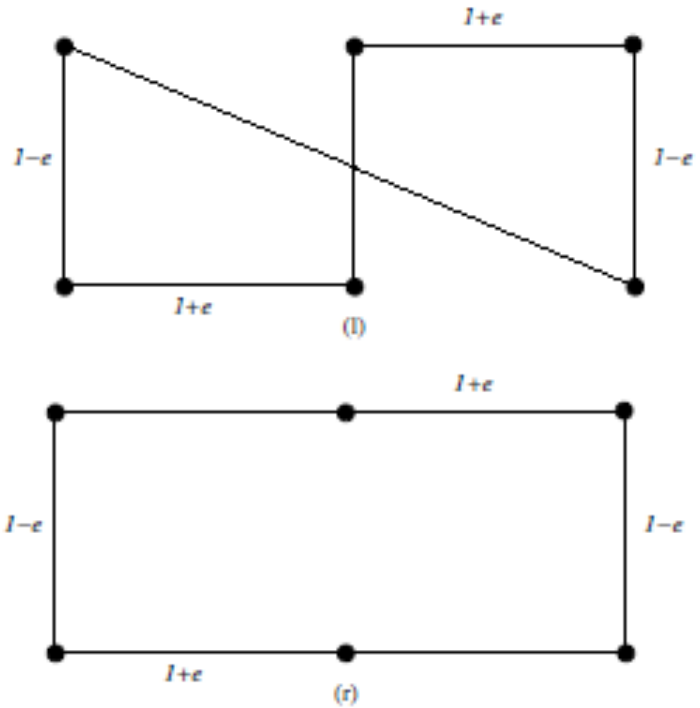


Solução

- Heurística do “par mais próximo”
  - “Repetidamente, conectar o par mais próximo de endpoints os quais não criarão um problema, tais como uma terminação prematura do ciclo. Cada vértice inicia como um cadeia de vértice única (“singleton”). Depois de fundir todas as soluções calculadas, teremos uma única cadeia contendo todos os pontos. Conectando os endpoints restantes, fechamos o ciclo. A cada passo, teremos um conjunto de vértices únicos e cadeias de vértices disjuntas para fundir.”

Resultado





Mirko Perkusich AA 7

Solução

- Que tal, força bruta?
  - Testar todas os caminhos possíveis! Retornamos o mais curto!
- Corretude
  - OK
- Simplicidade
  - OK
- Eficiência
  - NOK
  - Para um problema com 20 pontos, precisaríamos testar 20! = 2.432.902.008.176.640.000…
  - Um problema real tem em torno de 1000 pontos...
- Problema clássico
  - Problema do caixeiro viajante!

Motivação

- Um problema bem mais simples...
  - Escreva um programa para imprimir o número 300.000 de Fibonnaci.

```
Fib(n)
if n ≤ 1 then return n
return Fib(n – 1) + Fib(n - 2)
```

Objetivo

Provar a corretude de algoritmos recursivos e não-recursivos

Analisar algoritmos quanto à sua complexidade temporal

Entender quais técnicas de construção de algoritmos devem ser usadas para os mais diversos problemas

Identificar e relacionar problemas reais com os problemas clássicos da teoria dos grafos

Diferenciar as classes de problemas P e NP e classificar problemas reais em relação a estas classes

Ementa

Corretude de algoritmos

Análise de complexidade de algoritmos

Análise Assintótica

Técnicas de construção de algoritmos: divisão e conquista, método guloso, programação dinâmica, *backtracking* e *branch-and-bound*. Introdução à teoria dos grafos

As classes P e NP. NP-completude.

Conteúdos Programáticos

Unidade	Assunto
I	Função dos algoritmos no desenvolvimento de sistemas
II Corretude de algoritmos	Prova por indução
	Invariantes de laço
	Corretude de algoritmos recursivos
	Corretude de algoritmos não-recursivos
III Análise da complexidade de algoritmos	Análise assintótica
	Complexidade de tempo
	Análise de algoritmos simples
	Análise de algoritmos de ordenação

Conteúdos Programáticos

IV Técnicas de construção de algoritmos	Divisão-e-conquista
	O método guloso
	Programação dinâmica
	Backtracking e *branch-and-bound*
V Introdução à teoria dos grafos	Conceitos de grafos
	Análise de algoritmos sobre grafos
VI Classes de problemas	A classe P
	A classe NP
	NP-completude
	A questão P versus NP

O que é um algoritmo?

"Procedimento computacional bem definido que recebe algum valor, ou conjunto de valores, como ***entrada*** e produz algum valor, ou conjunto de valores, como ***saída***." (Cormen, 2009)

"Uma ferramenta para resolver ***problemas computacionais*** bem definidos." (Cormen, 2009)

Exemplo

"Ordenar uma sequência de números de forma crescente." **Problema da ordenação**

Ordenação

Entrada: Uma seqüência de n números $\langle a_1, a_2, \ldots, a_n \rangle$.

Saída: Uma reordenação da seqüência de entrada $\langle a'_1, a'_2, \ldots, a'_n \rangle$, onde $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Problemas Computacionais

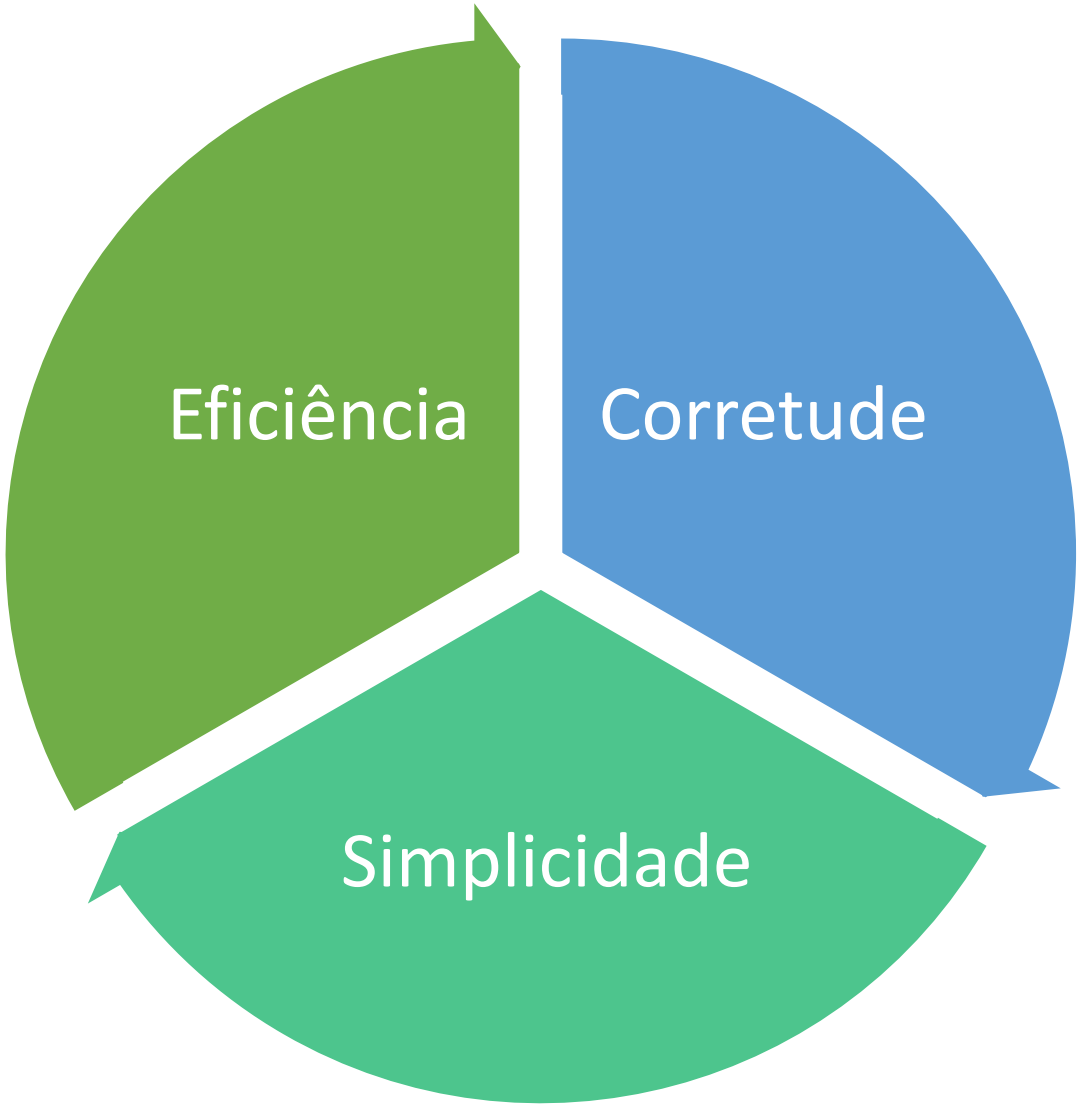
Para o exemplo em questão, com entrada <31, 41, 59, 26, 41, 58>, o algoritmo de ordenação deve retornar <26, 31, 41, 41, 58, 59>

Esta entrada é um exemplo de ***instância*** para o problema da ordenação

Instância de um problema

- “uma entrada, satisfazendo as restrições da declaração do problema, necessária para computar a solução do problema” (Cormen, 2009)

Avaliação de algoritmos



Avaliação de algoritmos

- Corretude
  - Um algoritmo está correto se para toda entrada (legal) ele produz a saída correta (ótima).

Avaliação de algoritmos

- Simplicidade
  - Fácil de ser entendido
  - Fácil de implementar
  - Fácil de manter

Avaliação de algoritmos

- Eficiência
  - Tempo - Quanto tempo leva para produzir a saída correta?
  - Espaço - Quanto espaço de memória é necessário?

Principais propriedades na elaboração de algoritmos

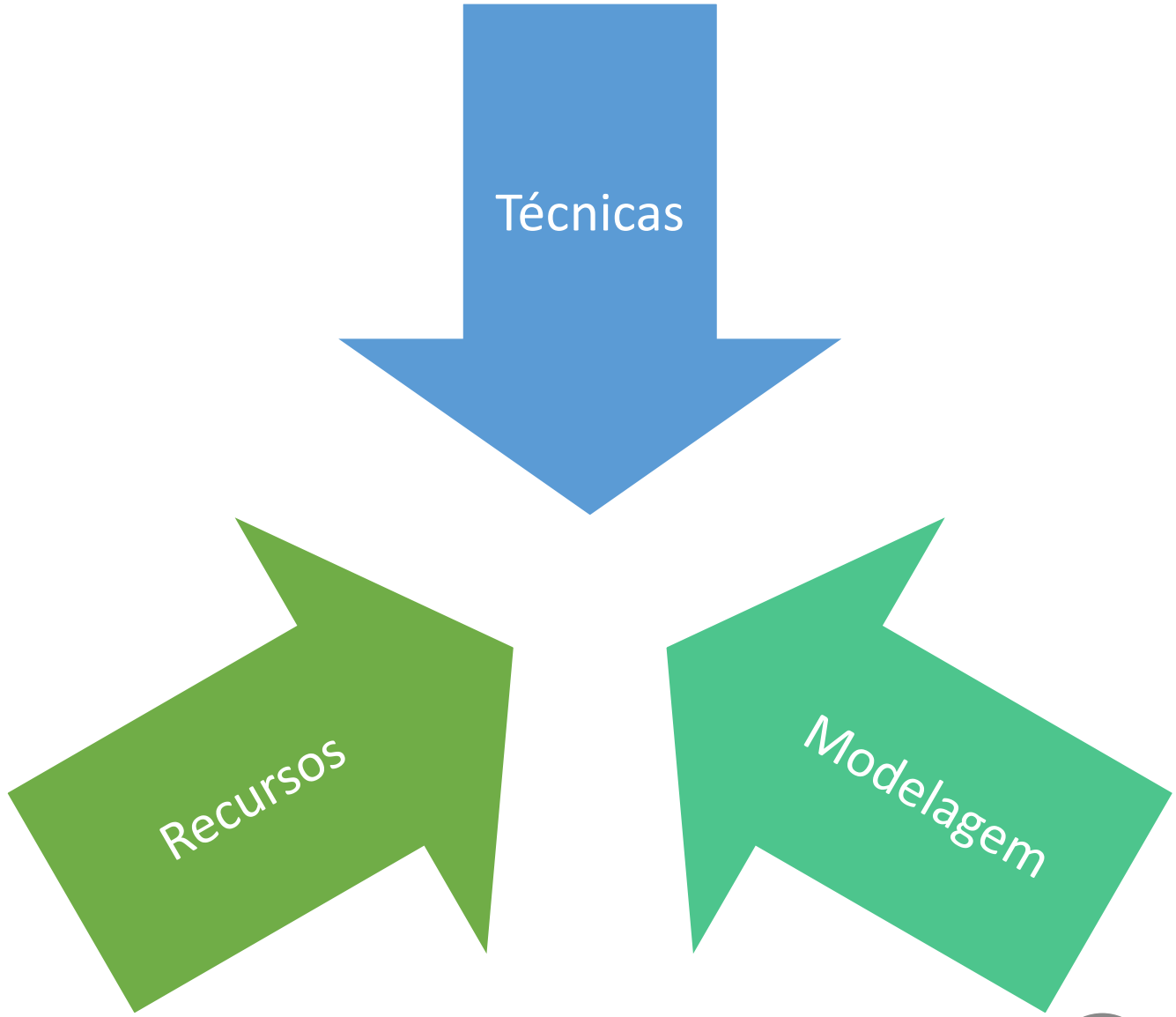
1. Fazer o algoritmo ter um tempo de execução sempre aceitável e

2. Ser a solução ótima ou provavelmente boa para o problema em todos os casos

Problema	• Nem sempre são ***alcançáveis***
Solução	• Heurísticas • um conjunto de regras e métodos que conduzem à descoberta, à invenção e à resolução de problemas

Consequência	• não cumprir uma dessas propriedades!
	  • encontrar boas soluções a maioria das vezes, mas sem garantias de que sempre encontrará ou
	  • processamento rápido, mas sem provas de que será rápido para todas as situações

O que faz um programador ser bom?



O que faz um programador ser bom?

- Técnicas
  - Entende fundamentos de técnicas de construção de algoritmos, tais como: estruturas de dados, programação dinâmica, busca em grafos, *backtracking*, e heurísticas.

O que faz um programador ser bom?

- Modelagem
  - Sabe transformar um problema real (muitas vezes, complexo e confuso) em um problema simples que pode ser atacado de forma algorítmica

O que faz um programador ser bom?

- Recursos
  - Não reinventa a roda! Utiliza algoritmos já existentes como base para suas soluções!

O que vimos hoje?

O que veremos na próxima aula?

Corretude de algoritmos

Análise de algoritmos