


PANC: Projeto e Análise de Algoritmos

Aula 02: Introdução a Projeto e Análise de Algoritmos

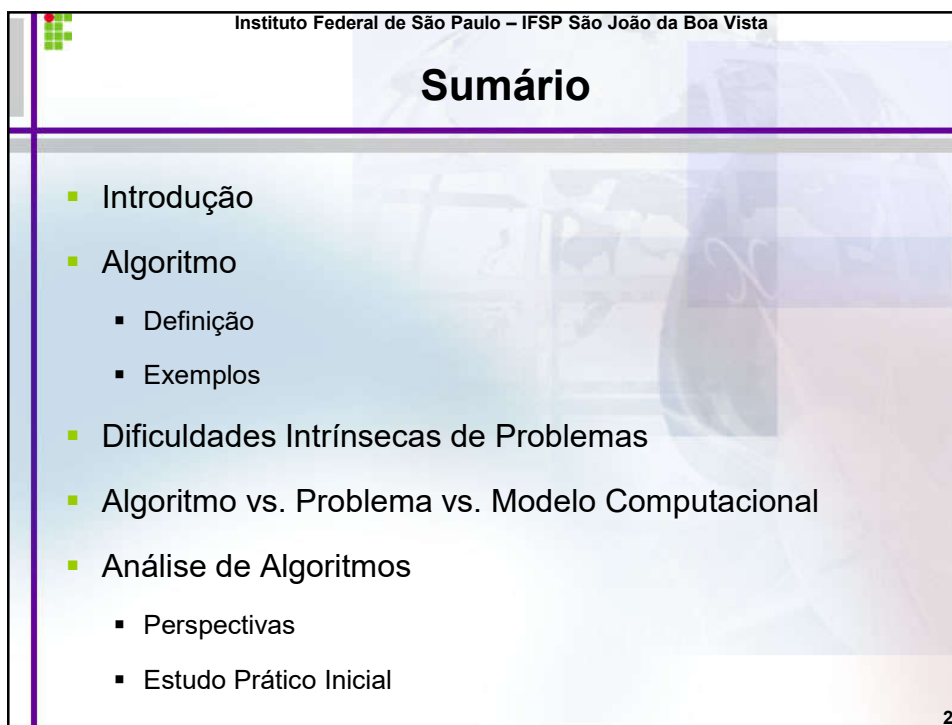
Breno Lisi Romano


11 de Fevereiro de 2020

<http://sites.google.com/site/blromano>

 **Instituto Federal de São Paulo – IFSP São João da Boa Vista**
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista



 Instituto Federal de São Paulo – IFSP São João da Boa Vista

Sumário

- Introdução
- Algoritmo
 - Definição
 - Exemplos
- Dificuldades Intrínsecas de Problemas
- Algoritmo vs. Problema vs. Modelo Computacional
- Análise de Algoritmos
 - Perspectivas
 - Estudo Prático Inicial

2

Introdução

- O que veremos nesta disciplina?
 - Como provar a “**Corretude**” de um algoritmo
 - Estimar a **quantidade de recursos** (tempo, memória) de um algoritmo = **Análise de Complexidade**
 - Técnicas e ideias gerais de projeto de **algoritmos**: divisão e conquista, programação dinâmica, algoritmos gulosos, etc.
 - Tema recorrente: natureza **recursiva** de vários problemas
 - A **difículdade intrínseca** de vários **problemas**: inexistência de soluções eficientes

3

A importância dos Algoritmos

- **Exemplos de aplicações para o uso/desenvolvimento de algoritmos “eficientes”?**
 - projetos de genoma de seres vivos
 - rede mundial de computadores
 - comércio eletrônico
 - planejamento da produção de indústrias
 - logística de distribuição
 - computação científica
 - imagens médicas
 - computação gráfica
 - processamento digital de imagens
 - games e filmes, ...

4

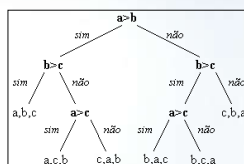
Algoritmo (1)

- **Definição?**
- **Definição Informal 01:**
 - É qualquer **procedimento computacional** bem definido que toma algum valor ou conjunto de valores como **entrada** e **produz** algum valor ou conjunto de valores como **saída**.
 - É uma **sequência de passos computacionais** que transformam uma **entrada** na **saída**
- **Definição Informal 02:**
 - Uma **ferramenta** para **resolver** um **problema** computacional bem especificado
 - O enunciado do **problema** computacional **especifica**, em termos gerais, o relacionamento entre **entrada** e **saída**
 - O **algoritmo** descreve um **procedimento computacional** para alcançar esse relacionamento da entrada com a saída

5

Exemplo 01: Ordenação de 03 Valores

- **Problema de Ordenação:**
 - Ordenar 03 números inteiros de maneira não decrescente
- **Entrada:**
 - a, b, c
- **Saída:**
 - $a < b < c \mid a < c < b \mid b < a < c \mid b < c < a \mid c < a < b \mid c < b < a$
- **Instância de um Problema:**
 - Uma instância de um problema consiste em uma entrada específica para qual se deseja calcular uma solução para o problema, por exemplo:
 - Entrada: 30, 11, 25
 - Saída: 11, 25, 30



```

Função Ordenar03Valores()
{
  Ler a, b, c
  if (a > b)
  {
    if (b > c) imprimir (a > b > c)
    else
    {
      if (a > c) imprimir (a > c > b)
      else imprimir (c > a > b)
    }
  }
  else
  {
    if (b > c)
    {
      if (a > c) imprimir(b > a > c)
      else imprimir(b > c > a)
    }
    else imprimir(c > b > a)
  }
}
  
```

6

Exemplo 02: Ordenação

- **Problema de Ordenação:**
 - Ordenar uma sequência de números de maneira não decrescente
- **Entrada:**
 - Uma sequência de n números $\langle a_1, a_2, \dots, a_n \rangle$
- **Saída:**
 - Uma permutação $\langle a'_1, a'_2, \dots, a'_n \rangle$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq \dots \leq a'_n$
- **Instância de um Problema:**
 - Entrada: $\langle 31, 41, 59, 25, 41 \rangle$
 - Saída: $\langle 25, 31, 41, 41, 59 \rangle$

7

Exemplo 03: Primalidade

- **Problema da Primalidade:**
 - Determinar se um dado número inteiro é primo
- **Entrada:**
 - Número Inteiro
- **Saída:**
 - Decisão: É primo ou Não é primo!
- **Instância de um Problema:**
 - Entrada: 9411461 / Saída: É primo
 - Entrada: 8411461 / Saída: Não é primo

8

Algoritmo (2)

- **Definição Formal 01:**

- Conjunto das **regras** e **procedimentos lógicos** perfeitamente **definidos** que levam à **solução de um problema** em um número de etapas (Dicionário Houaiss da Língua Portuguesa, 2001, 1a edição)

- **Definição Formal 02:**

- *The term algorithm is used in computer science to **describe a problem-solving method** suitable for **implementation** as a **computer program** (Algorithms in C, Sedgewick, 1998, 3rd edition)*

- **Observações:**

1. O número de **etapas** deve ser **finito**: se um algoritmo levar décadas, séculos ou milênios para executar, ele é impraticável
2. Existem **diferentes modelos computacionais** nos quais os algoritmos podem ser implementados

9

Algoritmo (3)

- Podemos **descrever** um **algoritmo** de diferentes maneiras:

- implementando-o em **linguagem de máquina** diretamente executável em hardware
- em **português**
- usando uma linguagem de **programação de alto nível**: C, Pascal, Java, Python, etc
- em um **pseudocódigo** de alto nível → Livro do Cormen

- Nesta disciplina, usaremos essencialmente as **duas últimas alternativas** nesta disciplina

10

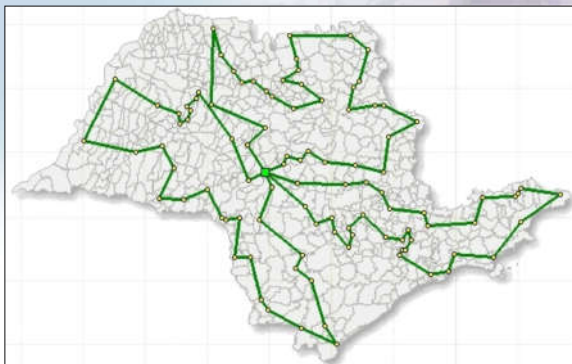
Dificuldade Intrínseca de Problemas (1)

- Infelizmente, existem certos **problemas** para os quais **não se conhece algoritmos eficientes** capazes de resolvê-los → Problemas NP-completos
 - Curiosamente, **não foi provado** que tais algoritmos não existem! Interprete isso como um desafio para inteligência humana.
 - Esses problemas tem a **característica notável** de que **se um deles admitir um algoritmo “eficiente”** então todos admitem algoritmos “eficientes”
- **Por que se preocupar com esses problemas?**
 - Problemas dessa classe surgem em inúmeras situações práticas

11

Dificuldade Intrínseca de Problemas (2)

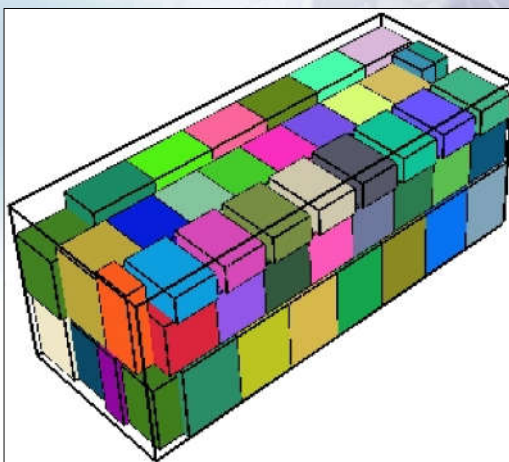
- **Vehicle Routing:** Calcular as rotas dos caminhões de entrega de uma distribuidora de bebidas no estado de São Paulo, minimizando a distância percorrida



12

Dificuldade Intrínseca de Problemas (3)

- **Bin Packing 3D:** calcular o número mínimo de containers para transportar um conjunto de caixas com produtos



13

Dificuldade Intrínseca de Problemas (4)

- **Facility Location:** calcular a localização e o número mínimo de antenas de celulares para garantir a cobertura de uma certa região geográfica



- É importante saber identificar quando estamos lidando com um problema desta natureza!!!

14

Algoritmo vs. Problema vs. Modelo Computacional

- Um programa pode ser entendido como um **algoritmo implementado** em uma **determinada linguagem** para solucionar um **problema computacional** específico em um **modelo computacional** em particular.
- Observações importantes:
 - Nem todo problema computacional é solucionável
 - Diferentes problemas possuem diferentes níveis de dificuldade
 - Diferentes algoritmos possuem diferentes níveis de complexidade
 - Dependendo do modelo computacional utilizado, pode não haver algoritmo possível para determinado problema
 - Cada instrução executada em um modelo computacional possui um custo de tempo
 - Cada dado armazenado em um modelo computacional possui um custo de espaço

15

Modelo Computacional (1)

- **Condição ideal (irreal):** os computadores têm velocidade de processamento e memória infinita → Qualquer algoritmo é igualmente bom e esta disciplina é inútil
- **O mundo real:** há computadores com velocidade de processamento na ordem de bilhões de instruções por segundo e trilhões de bytes em memória
 - Mas ainda assim existe uma limitação na velocidade de processamento e memória dos computadores
- Neste caso, faz muita diferença ter um **bom algoritmo (eficiente*)**

* virtude ou característica de (alguém ou algo) ser competente, produtivo, de conseguir o melhor rendimento com o mínimo de erros

16

Modelo Computacional (2)

- Vamos pensar sobre o Modelo Computacional para **Ordenação de Arrays** com **n elementos**:
 - **Computador A**: Executa 1G (10^9) instruções/segundo
 - **Computador B**: Executa 10M (10^7) instruções/segundo
 - Ou seja, A é **100 vezes** mais rápido que B
- **Algoritmo 01**:
 - Implementado em A
 - Excelente Programador em Linguagem de Máquina
 - Executa $2.n^2$ instruções
- **Algoritmo 02**:
 - Implementado em B
 - Programador Iniciante em Linguagem de Alto Nível
 - Executa $50 n \cdot \log n$ instruções

17

Modelo Computacional (3)

- O que acontece quando ordenamos um vetor de um milhão de elementos (10^6)? **Qual algoritmo é mais rápido?**
- **Algoritmo 1 na Máquina A:**

$$E_a = \frac{2.(10^6)^2 \text{ instruções}}{(10^9) \text{ instruções por segundo}} \rightarrow 2000 \text{ segundos}$$
- **Algoritmo 2 na Máquina B:**

Obs: Entenda $\log(\cdot)$ como sendo $\log_2(\cdot)$ ou $\lg(\cdot)$

$$E_b = \frac{50.(10^6 \log 10^6) \text{ instruções}}{(10^7) \text{ instruções por segundo}} \rightarrow 200 \text{ segundos}$$
- Ou seja, B foi **VINTE VEZES** mais rápido do que A
 - Se o vetor tiver 10 milhões de elementos, esta razão será de 2.3 dias para 20 minutos!

18

Modelo Computacional (4)

- O uso de um **algoritmo adequado** pode levar a ganhos extraordinários de desempenho
- Isso pode ser tão importante quanto a arquitetura de hardware (**modelo computacional**)
- A melhora obtida pode ser tão significativa que não poderia ser obtida simplesmente com o avanço da tecnologia
 - As melhorias nos algoritmos produzem avanços em componentes básicos das aplicações
 - Mas para analisarmos os algoritmos, precisamos pensar independente do modelo computacional
- **Conclusão:** Precisamos padronizar o Modelo Computacional

19

Modelo Computacional Genérico - Arquitetura de von Neumann

- Um único processador (ou unidade lógico aritmética)
- Memória RAM
- Operações sequenciais, não há paralelismo
- Instruções:
 - Aritméticas (soma, subtração, multiplicação, divisão, resto, piso e teto)
 - Movimentação de dados (carregar, armazenar e copiar)
 - Controle (desvio condicional e incondicional, chamada e retorno de rotinas)
- Cada instrução tem tempo de execução constante, embora possam ser diferentes
- Com base nas operações definidas, outras operações mais complexas podem ser derivadas
- Algumas áreas “cinzas” como exponenciação são tratadas como tempo constante também

20


Análise de Algoritmo (1)

- “Algorithm analysis usually means ‘give a big-O figure for the running time of an algorithm’ (Of course, a big- Θ would be even better). This can be done by getting a big-O figure for parts of the algorithm and then combining these figures using the sum and product rules for big-O. Another useful technique is to pick an elementary operation, such as additions, multiplications or comparisons, and observing that the running time of the algorithm is big-O of the number of elementary operations. Then, you can analyze the exact number of operations as function of n in the worst case. This is easier to deal with because it is an exact function of n and you don’t have the messy big-O symbols to carry through your analysis”
- - Ian Parberry, *Problems on Algorithms*



21

Análise de Algoritmo (2)

- **Definição:**
 - Analisar um algoritmo significa prever os recursos que ele necessitará para sua execução. Os mais importantes são tempo e espaço
- **Tempo de Execução:**  Estudaremos esta questão!
 - É o número de instruções primitivas executadas pelo algoritmo
- **Espaço:**
 - É de fato o espaço necessário para armazenar dados durante a execução do algoritmo
- **Utilidade:**
 - Com base na análise, podemos identificar a **eficiência** de cada **algoritmo** para um determinado problema
 - A análise é realizada levando em consideração um determinado modelo computacional

22

Perspectivas

- **Definição:**

- Além do ambiente computacional, o comportamento de um algoritmo pode variar de acordo com o comportamento da entrada (tamanho, estrutura, etc.), o que gera diferentes perspectivas

- **Melhor Caso:**

- A entrada está organizada de maneira que o algoritmo levará o tempo mínimo para resolver o problema

- **Pior Caso:**

- A entrada está organizada de maneira que o algoritmo levará o tempo máximo para resolver o problema

- **Caso Médio:**

- A entrada está organizada de maneira que o algoritmo levará um tempo médio para resolver o problema

23

Análise e Perspectivas

- As **análises** se concentram geralmente no pior caso e no caso médio:

- O **pior caso** nos dá uma ideia de quão ruim pode ser o comportamento do algoritmo - cálculo razoavelmente simples, nos dá uma garantia de que o algoritmo não poderá ser mais lento. Pode ser crucial para aplicações críticas.
- O **caso médio** nos dá uma ideia de como o algoritmo se comportará em boa parte dos casos: determinar qual é o comportamento médio pode ser mais complexo, envolvendo probabilidades

24

Funções Tipicamente Utilizadas

■ Representação:

- O **comportamento do algoritmo** é expressado como uma **função matemática** definida sobre o **tamanho da entrada**, denotada por $T(n)$
 - Por exemplo, ordenar 3 números é mais rápido que ordenar 1000 usando o mesmo algoritmo, porém, ambos seguem uma mesma função de crescimento do tempo
- Geralmente, o **tempo de execução aumenta** com o aumento da **entrada**

■ Análise Assintótica:

- A análise deve ser simplificada, e para a expressão da complexidade em cada perspectiva utilizamos a análise assintótica
- Estamos interessados mais no **comportamento**, na **taxa de crescimento do tempo de execução** do que de fato na precisão da função utilizada para expressar a complexidade

25

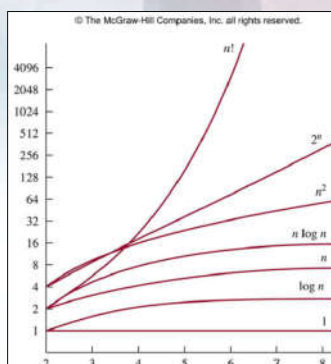
Medida de Complexidade e Eficiência de Algoritmos

- Um **algoritmo** é chamado **eficiente** se a função que mede sua **complexidade** de tempo é limitada por um **polinômio** no **tamanho da entrada**

- Por exemplo: n , $3n - 7$, $4n^2$, $143n^2 - 4n + 2$, n^5

■ Mas por que polinômios?

- Resposta: **polinômios** são funções bem **comportadas!!!**



26

Estudo Prático: Análise Inicial da Complexidade – Discussão sobre o Tempo de Execução

Problema:

- Dado um vetor A de n números inteiros, determine o maior valor entre eles.

Tópicos da Análise:

- Quais operações são relevantes?
- Quais são os limitantes superior e inferior para este problema?
- Melhor Caso? Pior Caso? Caso Médio?

- Como **calcular** “grosseiramente”, em função de n, o **número máximo de operações** realizadas por este algoritmo?

```

1 int arrayMax(int A[ ], int n)
2 {
3     int currentMax = A[0];
4     for(int i=1; i<n; i++){
5         if(A[i] > currentMax)
6             currentMax = A[i];
7     }
8     return currentMax;
9 }

```

27

Conclusões

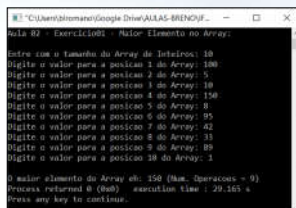
- Na **análise de complexidade** de um **algoritmo** estamos mais interessados no seu **comportamento geral** do que em outros detalhes que podem depender da máquina, do sistema operacional, da linguagem, dos compiladores, etc...
- Procura-se **medir a complexidade** de um algoritmo em função de um **parâmetro do problema**, geralmente, o **tamanho da entrada**
- Alguma **operação** (ou conjunto de operações) devem **balizar a análise de complexidade** de um algoritmo

28

Trabalhos para Casa (1)

■ Exercício 01 – Máximo Elemento de um Array:

- Implementar um Algoritmo em Linguagem C que identifique o elemento máximo de um array
 - Deve-se ler um array de tamanho N fornecido pelo usuário
 - Deve-se criar uma função que retorna o elemento máximo de um array → imprimir na console
- Deve-se imprimir também no console quantas operações foram necessárias para verificar o elemento máximo



```

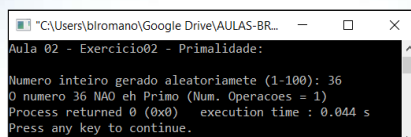
C:\Users\biromano\Google Drive\AULAS-BRENDO...
Aula 02 - Exercício01 - Máximo Elemento no Array:
Entre com o tamanho do Array de Inteiros: 10
Digite o valor para a posição 1 do Array: 100
Digite o valor para a posição 2 do Array: 5
Digite o valor para a posição 3 do Array: 10
Digite o valor para a posição 4 do Array: 150
Digite o valor para a posição 5 do Array: 8
Digite o valor para a posição 6 do Array: 95
Digite o valor para a posição 7 do Array: 42
Digite o valor para a posição 8 do Array: 31
Digite o valor para a posição 9 do Array: 89
Digite o valor para a posição 10 do Array: 1
O maior elemento do Array eh: 150 (Num. Operacoes = 9)
Process returned 0 (0x0)   execution time : 29.155 s
Press any key to continue.
  
```

29

Trabalhos para Casa (2)

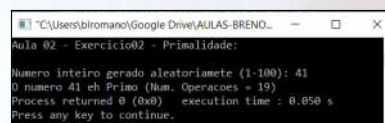
■ Exercício 02 – Primalidade:

- Implementar um Algoritmo em Linguagem C que gera um número inteiro aleatório
- Deve-se criar uma função que retorna se o número inteiro é primo ou não
- Deve-se imprimir no console quantas operações foram necessárias para verificar a primalidade do número gerado



```

C:\Users\biromano\Google Drive\AULAS-BR...
Aula 02 - Exercício02 - Primalidade:
Numero inteiro gerado aleatoriamente (1-100): 36
O numero 36 NAO eh Primo (Num. Operacoes = 1)
Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
  
```



```

C:\Users\biromano\Google Drive\AULAS-BRENDO...
Aula 02 - Exercício02 - Primalidade:
Numero inteiro gerado aleatoriamente (1-100): 41
O numero 41 eh Primo (Num. Operacoes = 19)
Process returned 0 (0x0)   execution time : 0.050 s
Press any key to continue.
  
```

30

Trabalhos para Casa (3)

Exercício 03 – Ordenação:

- Implementar um Algoritmo em Linguagem C que gera um Array de números inteiros aleatórios de tamanho N
- Deve-se criar uma função que ordene o Array gerado e imprima o resultado no Console (Array Ordenado) – Utilizar Pseudocódigo abaixo para Ordenação
 - Sugestão: Criar uma função para impressão do Array
- Deve-se imprimir também no console quantas operações foram necessárias na comparação e troca de um elemento do Array para ordená-lo

```

ORDENA
1  para j ← 2 até n faça
2    chave ← A[j]
3    ▷ Insere A[j] no subvetor ordenado A[1..j-1]
4    i ← j - 1
5    enquanto i ≥ 1 e A[i] > chave faça
6      A[i+1] ← A[i]
7      i ← i - 1
8    A[i+1] ← chave

```

```

C:\Users\Breno\Documents\Desenvolvimento\Breno\IFSP\Aula 02 - Exercício 03 - Ordenação
Digite um tamanho do Array de Inteiros: 10

Ordenacao
Array Desord[ ] = 46 05 01 40 15 77 54 53 82 95 - 65
Array Desord[ ] = 46 05 01 40 15 77 54 53 82 95 - 85
Array Desord[ ] = 46 05 01 40 15 77 54 53 82 95 - 49
Array Desord[ ] = 46 05 01 40 15 77 54 53 82 95 - 75
Array Desord[ ] = 15 46 46 45 55 77 54 53 82 95 - 77
Array Desord[ ] = 75 46 46 45 77 95 54 53 82 95 - 54
Array Desord[ ] = 15 46 46 54 65 77 95 53 82 95 - 33
Array Desord[ ] = 15 46 46 53 54 65 77 95 82 95 - 82
Array Desord[ ] = 15 46 46 53 54 65 77 82 95 95 - 95
Array Desord[ ] = 15 46 46 53 54 65 77 82 95 95
Array Ord[ ] = 15 46 46 53 54 65 77 82 95 95
Numero de Operacoes para Ordenacao: 25
Processo returned 0 (0x0) execution time : 2.363 s
Press any key to continue.

```

31

PANC: Projeto e Análise de Algoritmos

Aula 02: Introdução a Projeto e Análise de Algoritmos

Breno Lisi Romano

Dúvidas???

<http://sites.google.com/site/blromano>

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

32