Aula 04 – Decidibilidade

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação

Introdução

- TMs podem detectar padrões em strings, reconhecer linguagens e computar funções.
- Mas muitos outros problemas interessantes não se encaixam diretamente nas classes de problemas acima.
- **Estes *slides***: problemas de decisão e sua relação com máquinas de Turing.
- **Objetivos**: introduzir conceitos essenciais para a discussão de decidibilidade.

Referências

Chapter 11 & Section 12.1
T. Sudkamp
Chapter 4 – Decidability
M. Sipser
Chapter 5 – Decidable and Undecidable Languages
A. Maheshwari

Problemas de Decisão

- Um problema de decisão **P** é um conjunto de perguntas relacionadas para as quais há uma resposta: sim ou não.
- *Exemplos*:
  1. Um número natural n é primo?
     Uma questão para cada $n \in \mathbf{N}$:

$$\begin{array}{ccl} p_0 & : & \text{O número 0 é primo?} \\ p_1 & : & \text{O número 1 é primo?} \\ p_2 & : & \text{O número 2 é primo?} \\ \vdots & : & \qquad\vdots \end{array}$$

  2. Dada a linguagem fixa L, uma string qualquer x está em L?
     $\Rightarrow$ Uma questão para cada $x \in \Sigma^*$.
  3. Uma TM qualquer M para para uma entrada qualquer x?
     $\Rightarrow$ Uma questão para cada par $[M, x]$.

Problemas de Decisão

- Cada pergunta $p \in \mathbf{P}$ é uma instância do problema **P**.
- Um problema de decisão **P** é decidível (*decidable*) se existe um algoritmo que termina e determina a resposta apropriada para todas as instâncias $p \in \mathbf{P}$.
- Um problema de decisão **P** é semi-decidível (*semi-decidable*) se existe um algoritmo que termina e determina a resposta apropriada para as instâncias $p \in \mathbf{P}$ para as quais a resposta é sim.
- Um problema de decisão **P** é dito indecidível (*undecidable*) se **P** não for decidível ou semi-decidível.

Procedimento Efetivo

Um algoritmo que resolve um problema de decisão deve ser:

1. Correto (*sound*): produz a resposta sim para todas as instâncias $p \in \mathbf{P}$ cujo resultado esperado é positivo.
2. Completo (*complete*): produz a resposta não para todas as instâncias $p \in \mathbf{P}$ cujo resultado esperado é negativo.
3. Mecânico: formado por uma sequência finita de instruções que podem ser realizadas sem exigir intuição, perspicácia ou adivinhação.
4. Determinístico: sempre realiza a mesma computação para entradas iguais.

- Se existe um algoritmo para um problema **P** que é correto e completo então **P** é decidível.
- Nesse caso o algoritmo é um procedimento efetivo.
- Se existe um algoritmo para um problema **P** que é correto mas não é completo então **P** é semi-decidível.

TMs e Problemas de Decisão

- TM é a nossa representação de máquina abstrata de computação.
- No paradigma das TMs, procedimentos efetivos são especificados por TMs que sempre terminam.
- Se a condição de completude for retirada, TMs podem tratar qualquer problema semi-decidível.
- Para usar TMs para resolver problemas de decisão, estes precisam ser representados como a string de entrada da máquina.
- Aqui: foco em TMs para aceite de linguagens.
- $\Rightarrow$ Um problema de decisão **P** necessariamente precisa estar representado como um problema de aceite (reconhecimento) de linguagem.

TMs e Problemas de Decisão

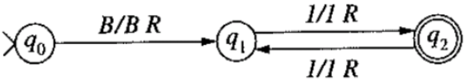
- A construção de uma solução usando TM para um problema de decisão envolve duas etapas:
  1. Escolha de como representar as instâncias do problema como uma string.
  2. Execução da máquina com a instância codificada para obter a resposta de sim/não.
- A etapa 1 requer uma seleção do alfabeto e representação das strings de entrada.
- As propriedades da representação são então usadas na construção da máquina que resolve o problema.

TMs e Problemas de Decisão

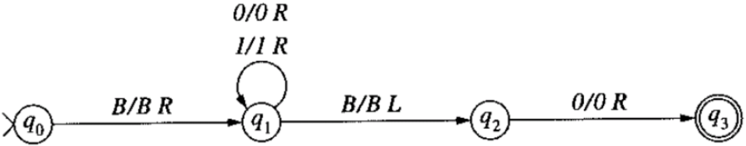
Instâncias de P	**Entrada da TM**	**Resposta**
$p_1 \rightarrow$	$w_1 \rightarrow$	sim/não
$p_2 \rightarrow$	$w_2 \rightarrow$	sim/não
$p_3 \rightarrow$	$w_3 \rightarrow$	sim/não
$\vdots$ Representação	$\vdots$ Computação da TM	$\vdots$
$p_i \rightarrow$	$w_i \rightarrow$	sim/não
$\vdots$	$\vdots$	$\vdots$

Representação de Problemas de Decisão

- Representações distintas de um mesmo problema de decisão resultam em TMs distintas.
- Exemplo: Determinar se um número natural é par.
- TM para a representação unária dos números:



- TM para a representação binária dos números:



- Um problema de decisão **P** tem uma solução (por TM) se existe ao menos uma combinação de representação de **P** e TM M que resolve **P**.

Problema da Pertinência

- O problema de pertinência para uma linguagem L é o problema de decisão fundamental.
- **Pertinência**: uma string de entrada $x \in \Sigma^*$ pertence a L?
- O problema **Pertinência** é
  - decidível se e somente se L é recursiva; ou
  - semi-decidível se e somente se L é recursivamente enumerável.
- Em muitos caso associamos a mesma denominação do problema à linguagem L.
- $\Rightarrow$ L é decidível se o problema é decidível.
- $\Rightarrow$ L é semi-decidível se o problema é semi-decidível.
- $\Rightarrow$ L é indecidível se o problema é indecidível.

Observação Sobre Não-Determinismo

Proposição

O problema de pertinência para L é decidível se e somente se existe uma NTM M com $L(M) = L$, aonde todas as computações de M terminam.

- Por que a proposição acima é verdadeira?
- $\Rightarrow$ Porque já foi visto que sempre é possível construir uma DTM M' equivalente a M.
- Além disso, se M sempre termina então M' também sempre termina.

Exemplo 11.2.2 (Sudkamp)

- Vamos usar não-determinismo para mostrar que o problema **PDG** (*Path in Directed Graph* – Caminho em Grafo Direcionado) é decidível.
- Problema **PDG**: determinar se existe um caminho entre um nó v_i e um nó v_j em um grafo direcionado G.
- Um grafo direcionado $G = (\mathrm{N}, \mathrm{A})$ é formado por um conjunto de nós $\mathrm{N} = \{v_1, \ldots, v_n\}$ e por um conjunto de arcos $\mathrm{A} \subseteq \mathrm{N} \times \mathrm{N}$.
- Como representar um grafo como uma string?
- Primeiro ponto: qual é o alfabeto de entrada?
- $\Sigma = \{0, 1\}$.

Exemplo 11.2.2 (Sudkamp)

- 1 continua sendo usado para representação unária de números.
- 0 é um símbolo especial que é usado como separador.
- Um nó v_k é codificado como 1^{k+1}.
- Notação: $en(v_k)$.
- Um arco $[v_s, v_t]$ é codificado como $en(v_s)0en(v_t)$.
- A string 00 é usada para separar arcos.
- A entrada da máquina é formada pela representação de G seguidos dos nós v_i e v_j.
- Não é preciso representar os nós de forma explícita, eles podem ser inferidos a partir dos arcos.
- Assim, a representação do grafo G é uma sequência de arcos codificados.

Exemplo 11.2.2 (Sudkamp)

- Seja $G = (\mathrm{N}, \mathrm{A})$ com:

$$\mathrm{N} = \{v_1, v_2, v_3\} \qquad \mathrm{A} = \{[v_1, v_2], [v_1, v_1], [v_2, v_3], [v_3, v_2]\}$$

- O grafo G é representado pela string $R(G) = 1101110011011001110111100111110111$.
- Uma computação para determinar se há um caminho de v_3 para v_1 em G começa como a entrada $R(G)0001111011$.
- Usa-se 000 para separar a codificação do grafo e os nós para análise.
- Uma NTM M de duas fitas é projetada para resolver o problema.

Exemplo 11.2.2 (Sudkamp)

Sumário das ações de M:

1. Testa se a entrada está correta.
   Se não estiver, M para e rejeita a string.
2. Entrada tem a forma $R(G)000en(v_i)0en(v_j)$.
   Se $v_i = v_j$, M para em um estado de aceite.
3. Escreve $en(v_i)0$ na fita 2.
4. Seja v_s o nó mais à direita na fita 2.
   Escolha um arco $[v_s, v_t]$ de forma não-determinística.
   Se não existe um arco ou v_t já está na fita, M para em um estado não-final.
5. Se $v_t = v_j$, M para em um estado de aceite.
   Senão, escreve $en(v_t)0$ na fita 2 e repete passo 4.

Por que L(M) é recursiva?

$\Rightarrow$ Passo 4 garante construção de caminhos acíclicos.

$\Rightarrow$ Máquina M sempre para.

Exemplo 11.2.2 (Sudkamp)

- Um problema de decisão é frequentemente definido através:
  - da descrição das suas instâncias; e
  - da condição que deve ser satisfeita para obtenção de uma resposta positiva.
- Assim, o problema do exemplo pode ser descrito como abaixo.

PDG (Caminho em Grafos Direcionados)

Input: Grafo direcionado $G = (\mathrm{N}, \mathrm{A})$, nós $v_i, v_j \in \mathrm{N}$

Output: sim; se existe um caminho em G de v_i até v_j
não; caso contrário.

Exercício

Um grafo direcionado é cíclico se ele possui ao menos um ciclo. Usando a representação de grafo direcionado do exemplo apresentado, projete uma TM que decide se um grafo é cíclico.

Uma Máquina Universal

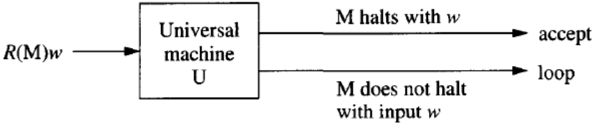
- Modelo de programa armazenado (MPA): $\sim$1945, permitiu o desenvolvimento prático de computadores.
- Computadores anteriores era dedicados para uma única tarefa.
- MPA permitiu a carga das instruções junto com os dados.
- Até agora, as TMs são como os computadores primitivos: projetadas para uma única função.
- Existe uma versão equivalente do MPA para TMs.
- Máquina de Turing Universal (UTM), 1936.
- von Neumann se baseou na UTM para projetar o MPA.

Uma Máquina Universal

- Uma UTM U simula as computações de uma TM arbitrária M.
- Analogia: M é um algoritmo descrito em software enquanto U é o hardware de um computador.
- A partir de agora: foco em TMs que aceitam por parada.
- Precisamos passar uma TM como entrada para uma computação.
- Em outras palavras, é preciso associar cada TM M com uma string R(M), a representação de M.
- A entrada para U é R(M)w, aonde w é a string que M deve processar.

Uma Máquina Universal

Funcionamento da UTM U pode ser descrito pelo diagrama:



- Se M para e aceita a entrada *w*, U faz o mesmo.
- Se M fica em loop para a entrada *w*, o mesmo acontece com U.
- A máquina U é dita universal porque qualquer TM M pode ser simulada por U.

Representação de uma TM

- Consideramos somente TMs com $\Sigma = \{0, 1\}$ e $\Gamma = \{0, 1, B\}$.
- Não é uma restrição já que qualquer TM pode ser simulada por uma TM com esses alfabetos (Hopcroft & Ullman, 1979).
- Estados das TM são ordenados pelos naturais: $Q = \{q_0, \ldots, q_n\}$.
- Lembrando que uma TM M é totalmente definida pela sua função de transição.
- Uma transição em uma TM padrão tem a forma $\delta(q_i, x) = [q_j, y, d]$, onde $q_i, q_j \in Q$; $x, y \in \Gamma$; $d \in \{L, R\}$.
- Os elementos de M são codificados usando strings de 1's.
- $en(z)$: codificação do símbolo z.

Representação de uma TM

Símbolo z da TM M	0	1	B	q_n	L	R
Codificação $en(z)$ para TM U	1	11	111	1^{n+1}	1	11

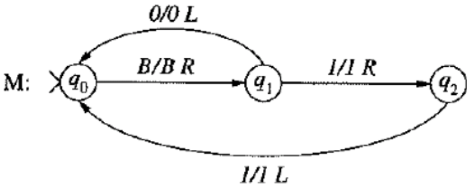
- Transição $\delta(q_i, x) = [q_j, y, d]$ é codificada como

$$en(q_i)0en(x)0en(q_j)0en(y)0en(d)$$

- 00: separador de transições.
- 000: início e fim de R(M).

Exemplo 11.5.1 (Sudkamp)

M não para com strings começando com 0, aceita todas as outras.



Transition	Encoding
$\delta(q_0, B) = [q_1, B, R]$	101110110111011
$\delta(q_1, 0) = [q_0, 0, L]$	1101010101
$\delta(q_1, 1) = [q_2, 1, R]$	110110111011011
$\delta(q_2, 1) = [q_0, 1, L]$	1110110101101

R(M):
00010111011011101100110101010100110110111011011001110110101101000

Determinando se uma string é uma TM

- Dada uma string $u \in \{0,1\}^*$.
- Verifique se u consiste de:
  - um prefixo 000;
  - seguido de uma sequência finita de transições codificadas separadas por 00; e
  - um sufixo 000.
- Se sim então u é a representação de alguma TM M (determinística ou não).
- M é determinística se a combinação de cada estado e símbolo em cada transição codificada é distinta.

Uma UTM U Determinística de 3-Fitas

- A computação começa com a entrada na fita 1.
- Ideia: se entrada tem a forma R(M)w, então a computação de M é simulada na fita 3.
- Computação de U segue os passos abaixo.

1. Se a entrada não tem a forma R(M)w ou se M não é determinística, U entra em loop.
2. Escreve w no início da fita 3.
3. Escreve 1 (codificando q_0) na fita 2.
4. Simula uma transição na fita 3: seja x o símbolo corrente na fita 3 e q_i o estado na fita 2.
    1. Busca uma transição aplicável na fita 1.
       Se não há, para e aceita.
    2. Se há uma transição aplicável:
        1. Escreve novo estado na fita 2.
        2. Escreve símbolo de saída na fita 3.
        3. Move cabeça da fita 3 como especificado na transição.
5. Computação continua com passo 4.

Uma Máquina Universal

Teorema 11.5.1 (Sudkamp)

A linguagem $L_H = \{R(M)w \mid M \text{ para com entrada } w\}$ é recursivamente enumerável.

- A UTM U aceita strings da forma $R(M)w$ que é a representação de uma DTM M e M para quando executada com entrada w.
- Para todas as outras strings, a computação de U não termina.
- Portanto, $L(U) = L_H$.
- L_H é aceita por uma TM $\Rightarrow$ recursivamente enumerável.
- L_H é conhecida como a linguagem do Problema da Parada.

Exemplo 11.5.2 (Sudkamp)

O problema abaixo é decidível.

Halts on *n*'th Transition Problem

Input: DTM M, string *w*, natural *n*

Output: sim; se a computação de M para entrada *w* realiza exatamente *n* transições antes de parar.
não; caso contrário.

Cuidado: esse é uma versão simplificada do Problema da Parada que é indecidível na forma geral.

Exemplo 11.5.2 (Sudkamp)

- Modifique U para U′ adicionando uma quarta fita para registrar o número de transições na simulação de M.
- Represente uma instância do problema como R(M)w0001^{n+1}.
- Computação de U′ segue os passos abaixo.

1. Se a entrada não termina com 0001^{n+1}, U′ rejeita.
2. Escreve 1^n na fita 4. Posiciona cabeça no início da fita 4. Apaga 0001^{n+1} da fita 1.
3. Se string na fita 1 não tem a forma R(M)w, U′ rejeita.
4. Copia w na fita 3. Escreve $en(q_0)$ na fita 2.
5. Simula M, usando a fita 4 como contador de transições de M (mover para direita).
   - Se M termina e U′ lê branco na fita 4. U′ para e aceita.
   - Se M termina sem um branco na fita 4, ou se M não terminaria mas há um branco na fita 4, U′ para e rejeita.

O Problema da Parada

- Sabemos que L_H é recursivamente enumerável.
- Mas L_H é recursiva?
- Em outras palavras, o problema abaixo é decidível?

Halting Problem (for TMs)

Input: DTM M com alfabeto de entrada Σ e uma string $w \in \Sigma^*$.

Output: sim; se a computação de M para com a entrada w.
não; caso contrário.

O Problema da Parada

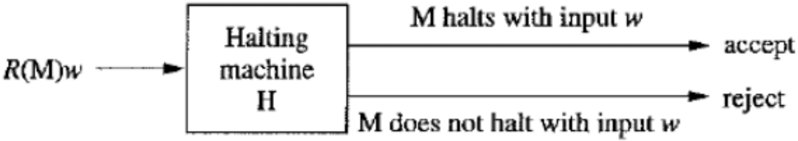
Teorema 12.1.1 (Teorema de Church – 1935)

O Problema da Parada (para TMs) é indecidível.

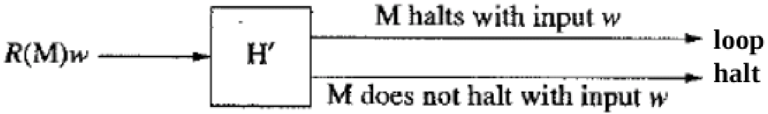
- Primeiramente desenvolvido por Alonzo Church, baseado no seu método de computação: λ-*calculus*.
- Desenvolvida de forma independente por Turing (1936). A prova original de Turing é por contradição, usando diagonalização.
- O argumento é um pouco mais complicado que o usual, por isso vamos usar uma prova alternativa.
- Vamos usar a ideia de uma UTM como visto anteriormente.

O Problema da Parada

Assuma (para efeitos do argumento de contradição) que existe uma UTM H que resolve o Problema da Parada.

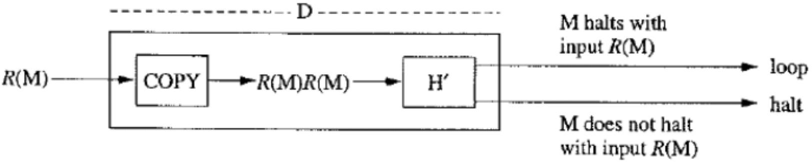


- Vamos usar H para construir uma máquina "impossível" D.
- Isso requer alguns passos.
- A TM H' é a mesma que H, exceto que H' entra em loop quando H aceita.

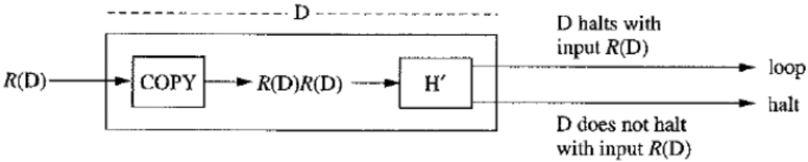


O Problema da Parada

Usamos a TM COPY (que copia a string, i.e., entrada *u* vira *uu*) juntamente com H' para construir a TM D.



Agora considere a computação de D com entrada *R*(D).



A TM D para com entrada *R*(D)?

O Problema da Parada

- Construímos uma TM D que para com a entrada $w = R(\mathrm{D})$ se, e somente se, D não para com essa entrada.
- Isso é obviamente uma contradição.
- Como todos os passos da prova envolveram TMs que são factíveis, a conclusão é que a suposição inicial está errada.
- Assim, é necessário rejeitar a suposição inicial de que existe uma TM H que resolve o Problema da Parada.
- Isto conclui a prova.

Breve Discussão Sobre a Prova Original

- Diagonalização: construa uma tabela aonde todas as entradas são TMs sobre $\Sigma = \{0, 1\}$ em alguma sequência $\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \ldots$.
- Entradas na tabela. Linha i, coluna j:
- 1 se M_i para quando executada com entrada $R(\mathrm{M}_j)$.
- 0 caso contrário.
- A TM D é construída sobre os elementos da diagonal, i.e., D não pode ser encontrada na tabela.
- No entanto, a tabela deveria conter todas as TMs.
- $\Rightarrow$ Contradição.

A Linguagem do Problema da Parada

Corolário 12.1.2 (Sudkamp)

A linguagem $L_H = \{R(M)w \mid M \text{ para com entrada } w\}$ não é recursiva.

Corolário 12.1.3 (Sudkamp)

As linguagens recursivas são um subconjunto próprio das linguagens recursivamente enumeráveis.

Corolário 12.1.4 (Sudkamp)

A linguagem $\overline{L_H}$ não é recursivamente enumerável.

Lambda Calculus – Outro Mecanismo de Computação

- *Lambda calculus*: inventado por Alonso Church nos anos da década de 1930.
- É um modelo (mecanismo) de computação universal equivalente a TMs.
- É uma teoria de funções como fórmulas e um sistema para manipular expressões formadas por funções.
- Exemplo:
  - Uma descrição usual na matemática: "Seja a função $f(x) = x^2$. Então considere $A = f(5)$."
  - Em *lambda calculus* as funções podem ser anônimas (lambdas), o que permite escrever:

$$A = (\lambda x.x^2)(5) \quad .$$

- Uma grande vantagem da notação lambda é permitir o uso de funções de alta ordem (*high-order funcions*), i.e., funções cujas entradas e/ou saídas são outras funções.

Lambda Calculus – Funções de Alta Ordem

- Suponha uma função qualquer f. A composição de f com ela mesma, isto é, $f \circ f$ é dada por

$$\lambda x.f(f(x)) \quad .$$

- A operação que mapeia f para $f \circ f$ é dada por

$$\lambda f.\lambda x.f(f(x)) \quad .$$

- A avaliação de funções de alta ordem tende a ficar complexa rapidamente.
- Exemplo. Qual o valor da expressão abaixo?

$$((\lambda f.\lambda x.f(f(x)))(\lambda y.y^2))(5)$$

- Resposta: A composição de $f(y) = y^2$ com ela mesma gera $g(y) = y^4$, e $g(5) = 5^4 = 625$.

Lambda Calculus – Histórico

- O *lambda calculus* foi inventado para investigar o conceito de computabilidade de funções.
- Exatamente a mesma motivação para a criação das máquinas de Turing.
- Em paralelo, Gödel inventou as funções μ-recursivas, que capturam o conceito de função computável por TM.
- Juntos, Gödel e Church mostraram que as funções computáveis no *lambda calculus* são exatamente as funções μ-recursivas.
- Isso quer dizer que *lambda calculus* e máquinas de Turing são mecanismos de computação equivalentes!
- Ou seja, tudo que se faz com um, se faz com outro, e vice versa.
- A noção de computabilidade transcendeu os mecanismos.
- Isso levou à tese a seguir.

Tese de Church-Turing

Tese de Chuch-Turing – Formulação 1

Tudo que é computável é computável por máquina de Turing.

Tese de Chuch-Turing – Formulação 2

Não existe um mecanismo de computação mais poderoso que máquina de Turing.

Tese de Chuch-Turing – Formulação 3

Qualquer outro mecanismo de computação que for criado será equivalente à máquina de Turing.

- As três formulações são equivalentes, mas a última leva ao conceito de *Turing complete*.
- Mostrar que uma LP ou outro mecanismo é *Turing complete*: mostrar que é equivalente à máquina de Turing.

Tese de Church-Turing

- A investigação das propriedades de computabilidade geraram variados métodos/formalismos para realização de computação algorítmica.
- Transformação de strings: sistemas de Post (1936), sistemas de Markov (1961).
- Avaliação de funções: funções μ-recursivas (Gödel, 1931; Kleene, 1936), *lambda calculus* (Church, 1930s).
- Máquinas abstratas de computação: *Register Machines* (Shepherdson, 1963), máquinas de Turing.
- A Tese de Church-Turing afirma que todos os formalismos acima são equivalentes.

Tese de Church-Turing

- A Tese de Church-Turing não é um teorema matemático, e portanto, não foi (não pode ser) provado formalmente.
- No entanto, ele pode ser refutado!
- Basta criar um mecanismo de computação mais poderoso que TM.
- Apesar do grade número de mecanismos já criados, a tese nunca foi refutada.
- $\Rightarrow$ Forte evidência de que ela é verdadeira.
- Vamos usar bastante essa tese daqui em diante: dar uma descrição de alto nível de uma TM com o entendimento que a TM poderia ser construída, se necessário.

Aula 04 – Decidibilidade

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação