**FUNDAÇÃO CENTRO DE ANÁLISE PESQUISA E INOVAÇÃO TECNOLÓGICA**

**FACULDADE FUCAPI (INSTITUTO DE ENSINO SUPERIOR FUCAPI)**

**COORDENAÇÃO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO**

**HUSSAMA IBRAHIM ISMAIL**

**SOFTWARE SIMULADOR DE TÉCNICAS DE ESCALONAMENTO   
de SISTEMAS em TEMPO REAL**

**MANAUS**

**2013**

**HUSSAMA IBRAHIM ISMAIL**

**SOFTWARE SIMULADOR DE TÉCNICAS DE ESCALONAMENTO**

**DE SISTEMAS EM TEMPO REAL**

Monografia apresentada ao curso de graduação em Engenharia de Computação da Faculdade Fucapi (Instituto de Ensino Superior Fucapi), como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

**Orientador: Mikhail Yasha Ramalho Gadelha, MSc.**

**MANAUS**

**2013**

**DEDICATÓRIA**

Dedico essa obra à minha família, em especial minha mãe, Isley Maria da Conceição Ramalho Gomes, por estar sempre do meu lado, nos momentos de vitória e derrota, e cujo amor é incondicional. A meu pai, Aldamir Gadelha, por ter me acompanhado nessa jornada e à minha esposa, Loma Brito do Nascimento, por sua paciência, compreensão, ajuda e pelo seu amor.

**AGRADECIMENTOS**

Agradeço primeiramente a Deus por me ajudar a conseguir todas as minha vitórias.

A meus amigos da Fucapi, Matheus Cavalcante, Alexsandro Sorato, Almino Júnior, Genival Caldeira, Marcus Brandt, Guilherme Juqueira, João Evangelista, Ronaldo Araújo, por sua amizade e companheirismo nesses 5 anos de faculdade.

A meus amigos da Ufam, Jeferson Bentes, Evandro Teixeira, Nathan Almeida, Marcos Correia, Heitor Carlos, Wellyghan Júnior, Diego Aquino, Fredyson Costa, Jhony Braga, Rogério Ramos, Northon Farias, Lucas Marques,Victor Loyola, pela amizade e por agüentar junto comigo noites em claro pra terminar os trabalhos a tempo.

Aos meus amigos brasileiros que conheci na Hungria, Maurício Andreis, Guilherme Becker, Victor Nicolau, Gabriel Brocki, Danillo Guilhon e Samir Salles, pela sua amizade e ajuda nos dias fora do Brasil.

Aos meus colegas e integrantes do projeto ZAGAIA, que contribuíram direta ou indiretamente para a realização desta obra.

Ao meu orientador, Prof.o Alexandre Duarte, pelo suporte no desenvolvimento dessa monografia.

*“Mas, como está escrito: Nem olhos viram, nem ouvidos ouviram, nem jamais penetrou em coração humano o que Deus tem preparado para aqueles que o amam.”*

1 Coríntios 2:9

**RESUMO**

Em um mundo altamente conectado através dos mais diversos meios de comunicação, como o celular e a internet é extremamente relevante que as informações sigilosas, ao trafegarem por redes públicas, cheguem íntegras e sigilosas aos seus destinatários. O presente projeto propõe uma otimização de um novo método de encriptação de dados por, mim proposto em um trabalho anterior, utilizando imagens, que se aproveita da distribuição aleatória dos valores de intensidade de pixels da imagem. Para validar a otimização, foram realizados testes comparativos, com os representantes dos principais algoritmos simétricos e assimétricos, respectivamente, de encriptação utilizados atualmente, o AES e o RSA, além do algoritmo original. Para tal, fez-se uso de mensagens aleatórias e textos na língua Portuguesa. A otimização mostrou-se eficiente em certos casos onde o algoritmo original mostra-se bastante lento. Com a otimização houve um ganho foi de até 60% no tempo de encriptação do algoritmo. Além disso, em tais casos houve um aumento de até 12% da entropia do arquivo encriptado.

Palavras-chave: criptografia, imagens, AES, RSA.

**ABSTRACT**

In a highly connected world through several means of communication such as cell phones and internet, is extremely important that sensitive information, trafficking on public networks, reach their recipients intact and confidential. In this project is proposed an optimization of a new method of data encryption, using images, which takes advantage of the random distribution of intensity values of image’s pixels. To validate the optimization, comparative tests were made with the main symmetric and asymmetric encryption algorithms currently used, AES and RSA, respectively, and the original algorithm. The optimization was efficient in certain cases where the original algorithm was very slow. With the optimization there was a gain of at most 60% of encryption time. Besides that, in those cases there was at most 12% of encrypted file entropy.

Key-words: cryptography, images, AES, RSA.

**LISTA DE FIGURAS**

[Figura 1: Representação dos parâmetros de uma tarefa de tempo real.. .......18](#_Toc278285462)

[Figura 2: Exemplo de tarefa periódica assumindo os valores: (r0 = 0, C = 6, T = 10, D = 10).....18](#_Toc278285462)

[Figura 3: Exemplo de tarefa esporádica sendo ativada no instante de tempo 14........................18](#_Toc278285462)

[Figura 4: Exemplo de escalonamento preemptivo.....................................................................18](#_Toc278285462)

[Figura 5: Exemplo de escalonamento não preemptivo...............................................................18](#_Toc278285462)

[Figura 6: Exemplo de cálculo do tempo máximo de resposta.....................................................18](#_Toc278285462)

[Figura 7: Exemplo de escalonamento com a técnica RM para um conjunto de tarefas t1 (r0=0, C=3, D=T=20), t2 (r0=0, C=2, D=T= 5), t3 (r0=0, C=2, D=T=10)..........................................18](#_Toc278285462)

[Figura 8: Exemplo de escalonamento com a técnica DM para um conjunto de tarefas t1 (r0=0, C=3, D=7 e T=20), t2 (r0=0, C=2, D=4 e T= 5), t3 (r0=0, C=2, D=9 e T=10)............................18](#_Toc278285462)  
Figura 9:[Exemplo de escalonamento com a técnica EDF para um conjunto de tarefas t1 (r0=0, C=3, D=7 e T=20), t2 (r0=0, C=2, D=4 e T= 5), t3 (r0=0, C=2, D=9 e T=10)............................18](#_Toc278285462)  
[Figura 10: Exemplo de escalonamento com a técnica *Round Robin* para um conjunto de tarefas A (r0=0, C=25, D=T=100), B (r0=0, C=20, D=T= 80), C (r0=0, C=30, D=T=100), D (r0=0, C=20, D=T=80).........................................................................................................................18](#_Toc278285462)  
[Figura 11: Exemplo de escalonamento com a técnica LL para um conjunto de tarefas t1 (r0=0, C=3, D=7 e T=20), t2 (r0=0, C=2, D=4 e T=5), t3 (r0=0, C=1, D=8 e T=10).........................................................................................................................................18](#_Toc278285462)

**LISTA DE ALGORITMOS**

[Algoritmo 1: Encriptação. 37](#_Toc278285521)

[Algoritmo 2: Decriptação. 39](#_Toc278285522)

[Algoritmo 3: Encriptação com coordenadas utilizando n-bits sem sinal. 43](#_Toc278285523)

[Algoritmo 4: Decriptação com coordenadas utilizando n-bits sem sinal. 44](#_Toc278285524)

[Algoritmo 5: Encriptação com procura de pixels próximos. 48](#_Toc278285525)

[Algoritmo 6: Decriptação com procura de pixels próximos. 49](#_Toc278285526)

[Algoritmo 7: Encriptação otimizada com lista de valores representáveis. 51](#_Toc278285527)

**LISTA DE TABELAS**

[Tabela 1: Substituições efetuadas no Código de César. 24](#_Toc278285528)

[Tabela 2: Valores representados pelos diferentes tipos de inteiros. 42](#_Toc278285529)

[Tabela 3: Valores de inteiros sem sinal de tamanho entre 9 e 15 bits. 43](#_Toc278285530)

[Tabela 4: Especificações das imagens utilizadas para teste. 52](#_Toc278285531)

[Tabela 5: Resultados obtidos na encriptação do Gênesis. 64](#_Toc278285532)

[Tabela 6: Comparativo entre o método proposto, AES e RSA. 66](#_Toc278285533)

# LISTA DE ABREVIATURAS E SÍMBOLOS

STR Sistema de Tempo Real

RM *Rate Monotonic*

DM *Deadline Monotonic*

EDF *Earliest Deadline First*

LL *Least Laxity*

C Tempo de Computação

D *Deadline*

T Período

r0 Tempo de Ativação

UI *User Interface*

JDK *Java Development Kit*

# SUMÁRIO

[1 INTRODUÇÃO 14](#_Toc278285941)

[1.1 OBJETIVOS 15](#_Toc278285942)

[1.1.1 Objetivo Geral 15](#_Toc278285943)

[1.1.2 Objetivo Específico 15](#_Toc278285944)

[1.2 METODOLOGIA 15](#_Toc278285945)

[1.3 ESTRUTURAÇÃO DO TRABALHO 16](#_Toc278285946)

[2 REPRESENTAÇÃO DE IMAGENS E TEXTOS EM UM COMPUTADOR 17](#_Toc278285947)

[2.1 REPRESENTAÇÃO DE TEXTO 17](#_Toc278285948)

[2.2 REPRESENTAÇÃO DE IMAGENS 18](#_Toc278285949)

[2.2.1 Processo de Digitalização de Imagem 19](#_Toc278285950)

[2.2.2 Imagens Digitais 20](#_Toc278285951)

[3 CRIPTOGRAFIA 22](#_Toc278285952)

[3.1 OBJETIVOS DA CRIPTOGRAFIA 22](#_Toc278285953)

[3.3 MODELO DE UM SISTEMA CRIPTOGRÁFICO DE CHAVE SIMÉTRICA 23](#_Toc278285954)

[3.4 MÉTODOS DE CRIPTOGRAFIA DE CHAVE SIMÉTRICA 24](#_Toc278285955)

[3.4.1 Técnica de Substituição 24](#_Toc278285956)

[3.4.2 Cifra de Transposição 25](#_Toc278285957)

[3.4.3 Principais Algoritmos de Criptografia de Chave Simétrica 26](#_Toc278285958)

[3.4.3.1 Data Encryption Standard (DES) 27](#_Toc278285959)

[3.4.3.3 Advanced Encryption Standard (AES) 29](#_Toc278285960)

[3.5 MODELO DE UM SISTEMA CRIPTOGRÁFICO DE CHAVE ASSIMÉTRICA 32](#_Toc278285961)

[3.6 MÉTODOS DE CRIPTOGRAFIA DE CHAVE ASSIMÉTRICA 34](#_Toc278285962)

[3.6.1 RSA 34](#_Toc278285963)

[3.7 ENTROPIA 35](#_Toc278285964)

[4 CRIPTOGRAFIA UTILIZANDO IMAGENS 36](#_Toc278285965)

[4.1. MÉTODO DE CRIPTOGRAFIA UTILIZANDO IMAGENS 36](#_Toc278285966)

[4.1.1. Processo de Encriptação 36](#_Toc278285967)

[4.1.2. Processo de Decriptação 38](#_Toc278285968)

[4.2. VANTAGENS E DESVANTAGENS DO MÉTODO 39](#_Toc278285969)

[4.2.1. Alta variabilidade do Arquivo Encriptado 39](#_Toc278285970)

[4.2.2. Tamanho do Arquivo Encriptado 42](#_Toc278285971)

[4.2.3. Tamanho da Imagem-chave 44](#_Toc278285972)

[4.2.4. Falta de Coordenadas para Representar Valores 45](#_Toc278285973)

[4.3. OTIMIZAÇÃO NO ALGORITMO 49](#_Toc278285974)

[5 EXPERIMENTOS E RESULTADOS 52](#_Toc278285975)

[5.1 GRUPO DE EXPERIMENTO 1 53](#_Toc278285976)

[5.1.1 Imagem de Teste 1: aerial.pgm 53](#_Toc278285977)

[5.1.2 Imagem de Teste 2:boats.pgm 54](#_Toc278285978)

[5.1.3 Imagem de Teste 3:bridge.pgm 55](#_Toc278285979)

[5.1.4 Imagem de Teste 4:D108.pgm 56](#_Toc278285980)

[5.1.5 Imagem de Teste 5: f16.pgm 57](#_Toc278285981)

[5.1.6 Imagem de Teste 6: girl.pgm 58](#_Toc278285982)

[5.1.7 Imagem de Teste 7: lena.jpg 59](#_Toc278285983)

[5.1.8 Imagem de Teste 8: peppers.pgm 60](#_Toc278285984)

[5.1.9 Imagem de Teste 9: pp1209.pgm 61](#_Toc278285985)

[5.1.10 Imagem de Teste 10: zelda.pgm 62](#_Toc278285986)

[5.2 GRUPO DE EXPERIMENTO 2 63](#_Toc278285987)

[5.3 GRUPO DE EXPERIMENTO 3 66](#_Toc278285988)

[6 DISCUSSÃO DOS RESULTADOS E CONCLUSÕES 67](#_Toc278285989)

[7 REFERÊNCIAS BIBLIOGRÁFICAS 69](#_Toc278285990)

[ANEXOS 71](#_Toc278285991)

[ANEXO A – TABELA ASCII E ISO/IEC 8859-1 72](#_Toc278285992)

[ANEXO B – BANCO DE IMAGENS 73](#_Toc278285993)

# 1 INTRODUÇÃO (TODO)

No decorrer da história da humanidade, o homem sempre precisou se comunicar de forma segura, seja enviando mensagens confidenciais através de redes de comunicações para aliados durante a guerra, como rádio, ou mais recentemente enviando relatórios confidenciais por email para um superior na empresa.

O mundo atualmente é movido por informações. As operações sobre elas devem ocorrer de forma segura. A cada dia são desenvolvidas técnicas para escutar e modificar tais informações. O meu trabalho vem no sentido de contribuir com mais um algoritmo para prover serviços de criptografia.

A criptografia surgiu para tentar evitar que informações sigilosas sejam recuperadas e alteradas por intrusos. Com ela é possível prover serviços de segurança para informações em arquivos, mas também em canais de comunicação, quando existem informações importantes trafegando por canais inseguros, como ocorre com a *internet*. [1]

No presente projeto será implementado uma otimização no algoritmo de criptografia de dados utilizando imagens [1]. A principal motivação deste trabalho é a busca pela melhoria no algoritmo original no sentido de diminuir o tempo de encriptação para determinados tipos de imagens.

Hoje existem duas classes diferentes de métodos de criptografia: o método de chave simétrica, utilizado desde os tempos do império romano, que utiliza uma mesma chave para encriptar e decriptar uma mensagem (apesar do mesmo conceito, o algoritmo utilizado na Roma antiga é, nos dias de hoje, trivialmente quebrável), e o método proposto em 1976 por pesquisadores do MIT que utiliza duas chaves diferentes, uma para a encriptação e outra para decriptação, chamado método de chave assimétrica. O algoritmo de criptografia de dados utilizando imagens é do tipo “criptografia de chave simétrica”, ou seja, a mesma chave (no caso, a imagem) é utilizada tanto na encriptação quanto na decriptação. A otimização aplica-se no momento em que o algoritmo tenta encriptar um valor que não possui representação na imagem, o que leva o algoritmo original a procura o *pixel* mais próximo representável do *pixel* faltante, enquanto o algoritmo otimizado procura por qualquer *pixel* na imagem para substituir o *pixel* faltante.

A otimização será avaliada e os seus resultados comparados com os resultados obtidos do algoritmo original, em teste envolvendo textos aleatórios e textos reais, em português.

## 1.1 OBJETIVOS

### 1.1.1 Objetivo Geral

Desenvolver uma ferramenta de *software* capaz de simplificar a geração de diagramas temporais para as técnicas de escalonamento clássicas utilizadas em Sistemas de Tempo Real como: *Rate Monotonic*, *Deadline Monotonic*, *Earliest Deadline First*, *Round Robin* e *Least Laxity* que trabalham com tarefas periódicas, e as técnicas de Servidores *Background*, Servidores de *Polling* e Servidores *Sporadic* aplicadas à tarefas esporádicas, abstraindo assim os cálculos e suas complexidades envolvidas durante este procedimento.

### 1.1.2 Objetivos Específicos

1. Efetuar estudo sobre Sistemas de Tempo Real, seus conceitos, sua utilização, suas técnicas clássicas de escalonamento e trabalhos relacionados.
2. Identificar técnicas clássicas de escalonamento que possam ter seu processo de geração de diagrama temporais automatizado.
3. Implementar a lógica e comportamento das técnicas de escalonamento utilizando como entrada tarefas periódicas e esporádicas.
4. Implementar testes unitários que validem a lógica implementada para cada técnica de escalonamento.
5. Desenvolver uma ferramenta de *software* que faça a integração das técnicas, que permita ao usuário inserir tarefas e que apresente o digrama temporal gerado.

## 1.2 METODOLOGIA

O trabalho foi divido em etapas que compreendem:

* Pesquisa Bibliográfica: Tem como base a análise de conteúdo de livros que servem de embasamento teórico sobre o tema Sistemas de Tempo Real, destacando pontos relevantes sobre seu funcionamento, seus conceitos e técnicas clássicas utilizadas para escalonamento de tarefas.
* Implementação: Desenvolvimento de algoritmos para as técnicas de escalonamento escolhidas utilizando linguagem de programação Java em conjunto com JavaFX e também o desenvolvimento de uma aplicação gráfica que permita ao usuário inserir tarefas a serem escalonadas e escolher a técnica desejada, apresentando diagrama temporal gerado.
* Validação: Para cada técnica de escalonamento a geração de testes unitários desenvolvidos com JUnit, utilizando como base nas suas assertivas exercícios resolvidos nos livros e materiais didáticos sobre o tema relacionado.

## 1.3 ESTRUTURAÇÃO DO TRABALHO (TODO)

Este trabalho esta dividido em 6 capítulos, descritos a seguir:

O capítulo 1 apresenta a introdução, descrição do cenário atual, motivação da proposta do algoritmo assim como os objetivos e a metodologia empregada no desenvolvimento.

O capítulo 2 apresenta a forma como são representados textos e imagens em um computador, assim como o processo de aquisição e transformação de uma imagem analógica em digital.

O capítulo 3 apresenta um estudo sobre criptografia, seus princípios e objetivos, além de uma descrição dos modelos de sistemas criptográficos simétricos e assimétricos e seus principais algoritmos.

O capítulo 4 apresenta o método de criptografia de chave simétrica utilizando imagens, da sua concepção à forma mais avançada. Apresenta também a otimização proposta por este projeto.

O capítulo 5 apresenta os resultados obtidos e a comparação entre os principais algoritmos de criptografia, o algoritmo original e o algoritmo otimizado.

O capítulo 6 apresenta as considerações finais do projeto assim como idéias para trabalhos futuros e melhorias no algoritmo original.

# 2 SISTEMAS OPERACIONAIS

Segundo (SILBERSCHATZ, GALVIN, GANGE, 2009, tradução nossa), “Sistema operacional é um programa que gerencia o hardware um computador”. Estes sistemas provêm a base para execução de programas e tarefas e servem como intermediadores entre usuário e máquina.

# 3 SISTEMAS DE TEMPO REAL

Neste momento, diversos sistemas e aplicações de tempo real estão sendo utilizados nas mais variadas atividades da raça humana, atuando em áreas militares, industriais, em sistemas embarcados, controles veiculares, aviões, saúde, entre outras.

Segundo Young (1982, tradução nossa), podemos dizer que STR: “É qualquer atividade de processamento de informações ou sistema que deve responder a um estimulo externo em um período finito e especificado.” .

KOPETZ (2011, p. 2, tradução nossa), também nos dá um complemento a definição de Young: “Um sistema de tempo real é um sistema onde a sua corretude não depende apenas do resultado da sua lógica mas também do tempo físico no qual esse resultado é produzido.”.

Na sua grande maioria de aplicações, os STR são utilizados para monitorar, controlar ou responder um ambiente externo utilizando sensores e atuadores, sendo implementados em sistemas embarcados especializados. Para isso devem ser bem projetados e tolerantes a falhas, pois problemas durante a sua execução podem ocasionar perdas financeiras (i.e., Um sistema automatizado de alimentação de linha de produção robotizada, caso o material não chegue a tempo o sistema pode danificar os robôs) ou até perdas de vidas (i.e., Um controlador de cancela em um cruzamento de uma rodovia e uma ferrovia, se este sistema falhar, pode ocasionar uma grande catástrofe).

## 3.1 TERMINOLOGIAS

Dependendo da área de atuação e tarefa no qual será exercida pelo STR, algumas terminologias são adotadas para esses sistemas, dentre elas:

* *Hard Real-Time*:“São sistemas onde é obrigatório que as respostas ocorram dentro de uma meta temporal estabelecida”.(BURNS; WELLINGS, 2011, tradução nossa)
* *Soft Real-Time*: “Sistemas onde a meta temporal é importante, mas o sistema continua funcionando caso ela seja ocasionalmente esquecida”. (BURNS; WELLINGS, 2011, tradução nossa)
* *Real Real-Time*: “São sistemas que são *Hard Real-time* e que possuem tempos de resposta muito curtos”. (BURNS; WELLINGS, 2011, tradução nossa)
* *Firm Real-Time*:“São sistemas que são *Soft Real-time* e que não se beneficia com uma entrega atrasada do serviço”. (BURNS; WELLINGS, 2011, tradução nossa)

# 4 ESCALONAMENTO

Um sistema de tempo real assim como os sistemas operacionais executam um conjunto de tarefas concorrentes, onde cada tarefa precisa de dados, processamento e acesso à recursos (i.e., Dispositivos de entrada e saída, sensores, atuadores, entre outros), no entanto, para esses tipos de sistema o grande desafio é realizar todas as operações pendentes sem violar a meta temporal de cada tarefa.

Nesse item, discorre-se sobre como acontece o escalonamento de tarefas em sistemas de tempo real.

## 4.1 TAREFAS DE TEMPO REAL

Segundo (COLLET; DELACROIX; KAISER; MAMMERI, 2012, tradução nossa) temos que: “Tarefas de tempo real são entidades básicas executáveis e escalonáveis, e que devem ser periódicas ou aperiódicas”, ou seja, quando falamos em entidades básicas, podemos levar em consideração que são “Segmentos de código de software que deverá ser executada múltiplas vezes com diferentes dados de entrada” (CORDEIRO, 2013).

Cada tarefa é caracterizada pelos seus parâmetros de tempo, e é composta por:

* *r0* (tempo de ativação): Representa o instante de tempo no qual a tarefa será iniciada durante o processo.
* *C* (tempo de computação): Valor obtido a partir do pior caso de computação para execução de determinada tarefa.
* *D* (meta temporal ou *deadline*): Representa o instante de tempo máximo aceitável no qual a tarefa deve ser executada.
* *T* (período, válido apenas para tarefas periódicas): Representa a periodicidade no qual a tarefa será executada. Ao fim do período a tarefa é novamente iniciada.

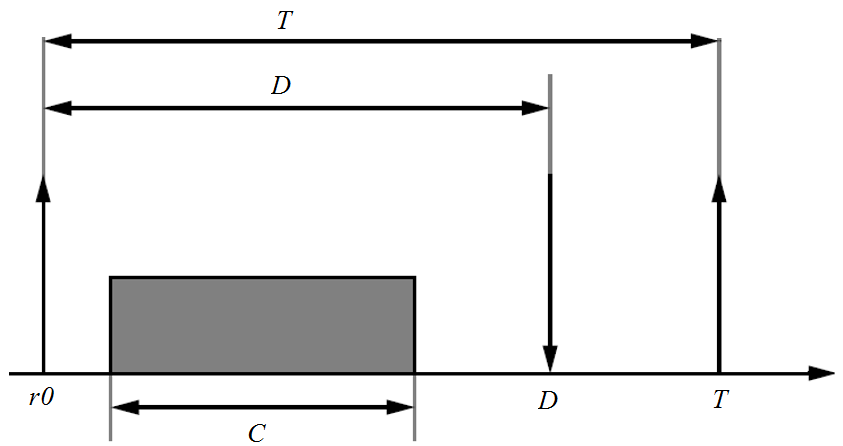


Figura 1: Representação dos parâmetros de uma tarefa de tempo real.

Na Figura 1, podemos observar a representação de uma tarefa de tempo real, onde ela possui um instante inicial *r0,* uma meta temporal *D,* um tempo de computação *C* e um período no qual possuirá uma nova execução *T.*

### 4.1.1 Tarefas Periódicas

Tarefas periódicas são compostas pelos 4 parâmetros (*r0, C, D, T)* e são ativadas a cada instante de tempo T, sendo assim executadas várias vezes durante a execução do sistema.

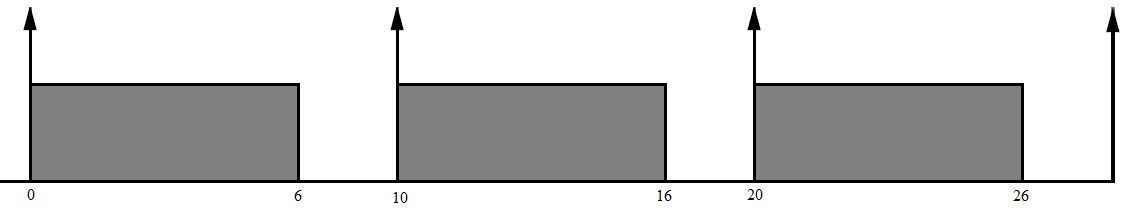
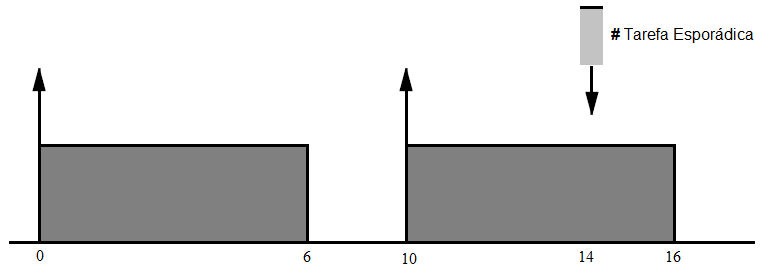


Figura 2: Exemplo de tarefa periódica assumindo os valores: (*r0* = 0, *C* = 6, *T* = 10, *D* = 10).

A representação da Figura 2 nos mostra uma tarefa periódica que é iniciada no instante 0 e que possui período igual a 10, ela se repetirá desta forma durante toda a execução do sistema. Trazendo para um exemplo real, poderíamos ter um sistema que possui uma tarefa de leitura de um sensor de temperatura a cada 10 segundos.

### 4.1.2 Tarefas Aperiódicas (Esporádicas)

São tarefas que ocorrem eventualmente no sistema. Elas podem passar um longo período sem ativação, no entanto, após sua ativação devem ser executadas dentro de um tempo hábil, para atender essas tarefas existem técnicas específicas como servidores *Background*, de *Polling* e *Sporadic*.

  
Figura 3: Exemplo de tarefa esporádica sendo ativada no instante de tempo 14.

O exemplo da Figura 3 nos mostra um cenário com uma atividade periódica assumindo os valores (*r0* = 0, *C* = 6, *T* = 10, *D* = 10), no entanto, no instante de tempo 14, é ativada uma tarefa esporádica. Trazendo para um exemplo real podemos imaginar um sistema que possui uma tarefa de leitura de um sensor a cada 10 segundos, no entanto, no instante 14 é acionado um botão de emergência que solicita a parada do sistema.

## 4.2 ESCALONADOR

O escalonador é um componente que faz parte do o núcleo ou *kernel* de um sistema de tempo real e é responsável por alocar as tarefas seguindo regras e políticas definidas. Para STR, possuímos duas abordagens de escalonadores, que podendo ser estáticos ou dinâmicos.

### 4.2.1 Escalonadores Estáticos

(KOPETZ, p. 240, 2011, tradução nossa) diz que “Um escalonador é chamado estático (ou *pre-run-time*) se ele faz suas decisões de escalonamento em tempo de compilação”.

Esses escalonadores também são chamados de *off-line* e trabalham gerando uma tabela de escalonamento com todas as tarefas e suas execuções já definidas, não sendo possível altera-las durante sua execução.

“Qualquer variação no modelo de tarefas implicará na geração de uma nova tabela de escalonamento” (CORDEIRO, 2013).

### 4.2.2 Escalonadores Dinâmicos

(KOPETZ, p. 240, 2011, tradução nossa) considera que “Um escalonador é considerado dinâmico (ou *on-line*), se ele faz suas decisões de escalonamento em tempo de execução, selecionando uma tarefa de um conjunto de tarefas prontas para serem executadas”.   
 Estes escalonadores também são conhecidos por escalonadores por prioridade, e são mais flexíveis podendo se adaptar a determinados cenários, utilizando políticas criadas durante sua concepção.

## 4.3 TIPOS DE ESCALONAMENTOS

Para o processo de escalonamento podemos adotar duas abordagens, escalonamento preemptivo ou não preemptivo.

### 4.3.1 Preemptivo

Supondo que tenhamos o processo de escalonamento de um conjunto de tarefas (*A*, *B*), durante a execução da tarefa *A*, caso seja ativada uma tarefa *B,* a execução de *A* será interrompida até que *B* tenha sido completada.

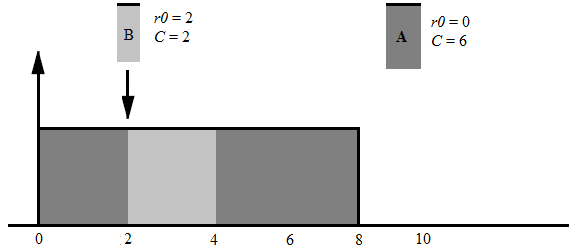


Figura 4: Exemplo de escalonamento preemptivo.

Na figura 4, podemos observar um exemplo de escalonamento preemptivo, onde durante a execução de uma tarefa *A* que possui tempo de computação igual a 6, no instante de tempo 2, uma tarefa *B* de tempo de computação igual a 2 é iniciada, a tarefa *A* é interrompida até que B esteja concluída, isso atrasa o fim da execução da tarefa *A* para o instante de tempo 8.

### 4.3.2 Não Preemptivo

O escalonamento não preemptivo tem seu comportamento oposto ao preemptivo, durante a execução de uma tarefa *A,* caso *B* seja ativada, até ela fica em espera que *A* seja completada.

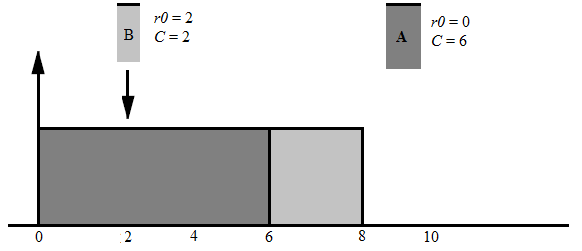


Figura 5: Exemplo de escalonamento não preemptivo.

Conforme figura 5, percebemos que o escalonamento não preemptivo trabalha de maneira diferente ao preemptivo, durante a execução de uma tarefa *A* com *C*=6, uma tarefa *B* com *C*=2 tem sua execução solicitada, no entanto, ela só é realmente executada após a finalização da tarefa *A*, ou seja, no instante de tempo 6.

## 4.4 TÉCNICAS DE ESCALONAMENTO

### 4.4.1 Tarefas Periódicas

#### 4.4.1.1 *Rate Monotonic* (RM)

Segundo (KOPETZ, p.251, 2011, tradução nossa) temos que “*Rate Monotonic* é um algoritmo dinâmico, preemptivo e baseado em prioridades estáticas de tarefas”. Este algoritmo foi publicado em 1973 por Liu, C.L. & J.W. Layland.

“Trata-se de um algoritmo ótimo para sistemas monoprocessador” (CORDEIRO, 2013).

O RM é um algoritmo simples, que trabalha com o escalonamento de tarefas periódicas e adota a política que quanto menor o período de uma tarefa, maior é sua prioridade. Ele não é recomendado para sistemas que possuem tarefas com metas temporais (*deadline*) menores que o período.

Antes de utilizar o algoritmo RM para o escalonamento de um conjunto de tarefas, é recomendo a realização de 2 testes que verificam se é ou não possível efetuar o procedimento, que são: “Teste Suficiente de Escalonabilidade” e “Cálculo do tempo máximo de resposta”.

**Teste Suficiente de Escalonabilidade**: É um teste não obrigatório mas que pode facilmente dizer se o conjunto de tarefas informados pelo usuário pode ser ou não escalonado com a técnica.

Ele é definido por:

Onde dado um conjunto *n* de tarefas é calculado o somatório do tempo de computação (*Ci*)sobre período (*T*)de cada tarefa.

No entanto, caso o teste não seja satisfeito ainda é possível que o conjunto de tarefas seja escalonável, fazendo-se necessário a realização do cálculo do máximo tempo de resposta.

**Cálculo do tempo máximo de resposta**: Por meio do cálculo do tempo máximo de resposta é possível obter um resultado mais correto à respeito da escalonabilidade.

Este mecanismo utiliza um procedimento recursivo, sendo um pouco mais complexa a sua execução.

O cálculo é feito para cada tarefa e é definido por:

Onde *Ii* representa uma interferência feita pelas tarefas de maior prioridade que a tarefa

A interferência *Ii* é calculada por:

Temos que *hp(i)* representa o conjunto de tarefas com maior prioridade que

Dado um conjunto de duas tarefas *t1*(*r0* = 0, *C* = 6.25, *T* = *D* = 25) e *t2* (*r0* = 0, *C* = 6.25, *T* = *D* = 50) possuímos a seguinte execução para o cálculo:

Tarefa T1:

O cálculo é realizado para cada tarefa até que haja uma convergência no resultado.

Após sabermos que é possível escalonar o conjunto de tarefas com a técnica RM, podemos aplicar o algoritmo.



Figura 7: Exemplo de escalonamento com a técnica RM para um conjunto de tarefas *t1* (*r0*=0, *C*=3, *D*=*T*=20), *t2* (*r0*=0, *C*=2, *D*=*T*= 5), *t3* (*r0*=0, *C*=2, *D*=*T*=10).

Observando da figura 7, temos um escalonamento utilizando a técnica RM para as tarefas periódicas *t1* (*r0*=0, *C*=3, *D*=*T*=20), *t2* (*r0*=0, *C*=2, *D*=*T*= 5), *t3* (*r0*=0, *C*=2, *D*=*T*=10). No instante de tempo 0, possuímos as 3 tarefas prontas para execução, no entanto, percebemos que ao aplicar a política de prioridades do algoritmo, a tarefa *t2* é a mais prioritária por possuir menor período, sendo assim a primeira a ser executada, sua execução vai até o instante 2. Logo em seguida utilizando o mesmo critério, a próxima tarefa ser executada é *t3,* que é executada até o instante de tempo 4. Após a execução de *t3*, a tarefa *t1* é iniciada, até o instante 5 no qual é suspensa por preempção para execução da tarefa *t2,* retornando no instante de tempo 7 até 9 e assim sucessivamente.

#### 4.4.1.2 *Deadline Monotonic* (DM)

O algoritmo *Deadline Monotonic* (ou *Inverse Deadline*) é outro algoritmo simples muito parecido com o RM, no entanto, a sua política de prioridades é baseada no deadline de cada tarefa, sendo as tarefas com menores metas-temporais mais prioritárias. Este algoritmo foi publicado em 1982 por Leung e Whitehead.

O DM assim como RM também utiliza os mesmos testes para verificação se o conjunto de tarefas é ou não escalonável para o algoritmo, no entanto, para o teste suficiente de escalonabilidade é feita uma pequena modificação, no qual é substituído o período pelo *deadline,* ficando da seguinte maneira: .

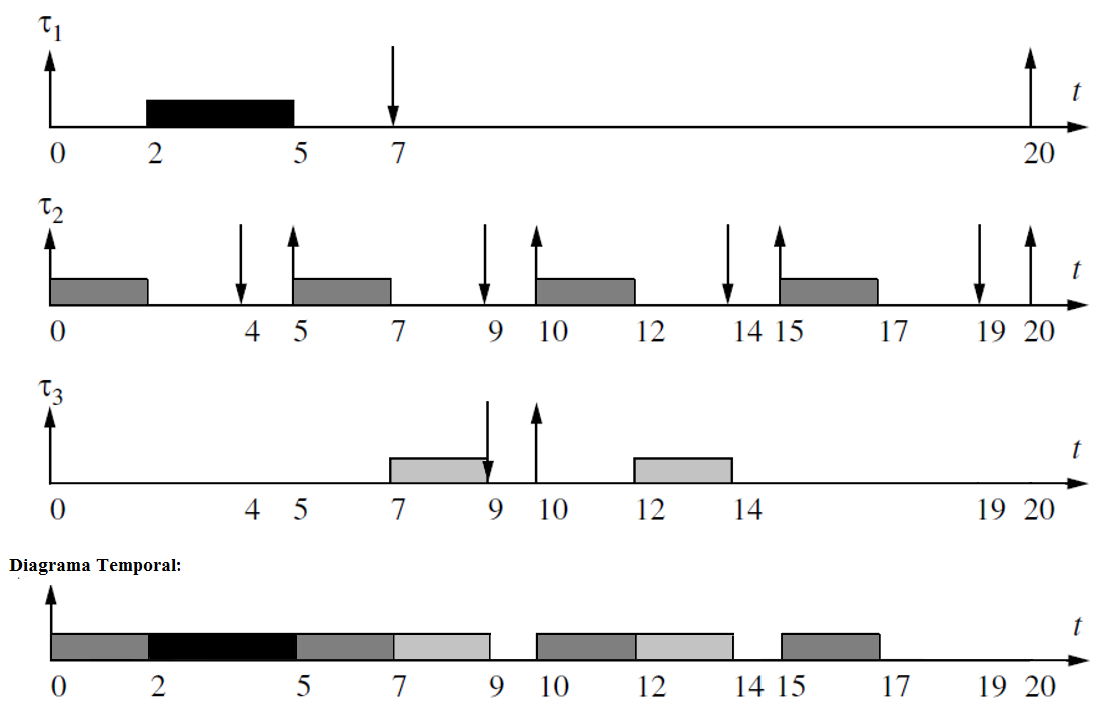


Figura 8: Exemplo de escalonamento com a técnica DM para um conjunto de tarefas *t1* (*r0*=0, *C*=3, *D*=7 e *T*=20), *t2* (*r0*=0, *C*=2, *D*=4 e *T*= 5), *t3* (*r0*=0, *C*=2, *D*=9 e *T*=10).

Conforme a figura 8, percebemos que a ideia de escalonamento com DM, é bem semelhante com RM, no entanto, a principal diferença é na política de prioridades, neste exemplo de escalonamento com 3 tarefas *t1* (*r0*=0, *C*=3, *D*=7 e *T*=20), *t2* (*r0*=0, *C*=2, *D*=4 e *T*= 5), *t3* (*r0*=0, *C*=2, *D*=9 e *T*=10), no instante de tempo 0, as 3 tarefas estão prontas para execução, no entanto, a tarefa *t2* é executada primeiramente por possuir menor *deadline,* logo em seguida a tarefa *t1* é executada. No instante de tempo 5, duas tarefas estão prontas para execução, *t2* e *t3, t2* é executada por possuir maior prioridade então, após ela, *t3* é finalmente executada.

#### 4.4.1.3 *Earliest Deadline First* (EDF)

Assim como o RM e o DM, o EDF é um algoritmo preemptivo, dinâmico, otimizado para sistemas monoprocessador e que se baseia em prioridades dinâmicas. Ele foi proposto em 1973 por Liu e Layland, e diferente do RM que trabalha com prioridade baseada no período e do DM que trabalha com prioridades baseadas no *deadline*, o EDF utiliza suas prioridades baseadas no *deadline* absoluto das tarefas a serem executadas, essas prioridades são modificadas durante o tempo de execução. A tarefa pendente de execução que possuir a sua meta temporal mais próxima de ser alcançada recebe maior prioridade.

Para sabermos se é possível a operação de escalonamento com o algoritmo EDF possuímos um teste de escalonabilidade feito pelo cálculo:

Onde o conjunto de tarefas é escalonável desde que a utilização do processador não ultrapasse 100% (i.e., O valor 1 representa 100%).



Figura 9: Exemplo de escalonamento com a técnica EDF para um conjunto de tarefas *t1* (*r0*=0, *C*=3, *D*=7 e *T*=20), *t2* (*r0*=0, *C*=2, *D*=4 e *T*= 5), *t3* (*r0*=0, *C*=2, *D*=9 e *T*=10).

Na Figura 9, temos um escalonamento utilizando EDF para o mesmo conjunto de tarefas utilizados na Figura 8, e diferente do DM, a prioridade é dada a tarefa com o menor deadline absoluto, ou seja, a que tem sua meta temporal mais próxima de ser alcançada. No instante 0, as 3 tarefas estão prontas para serem executadas, no entanto, *t2* tem prioridade, por ter sua meta temporal mais próxima (i.e., instante de tempo 4), após *t2, t1* é executada devido aos mesmos critérios (i.e., meta temporal no instante de tempo 7 contra instante de tempo 8 da tarefa t3) e assim sucessivamente.

#### 4.4.1.4 *Round Robin*

O algoritmo *Round Robin* diferente dos algoritmos apresentados anteriormente não aplica uma política de prioridades, o que torna fácil a sua implementação, no entanto, lhe faz não recomendável para sistemas de tempo real crítico.

Sua ideia básica é a disponibilização de fatias de tempo (i.e., Definidas pelo projetista durante a criação do projeto) no qual as tarefas serão executadas, ou seja, o processador é alocado para cada tarefa no tempo definido. Caso a tarefa ainda não tenha sido executada por completo, ela aguarda a próxima alocação do processador para sua execução. Caso a tarefa termine a sua execução antes da fatia de tempo o processador é liberado e uma nova tarefa é processada.

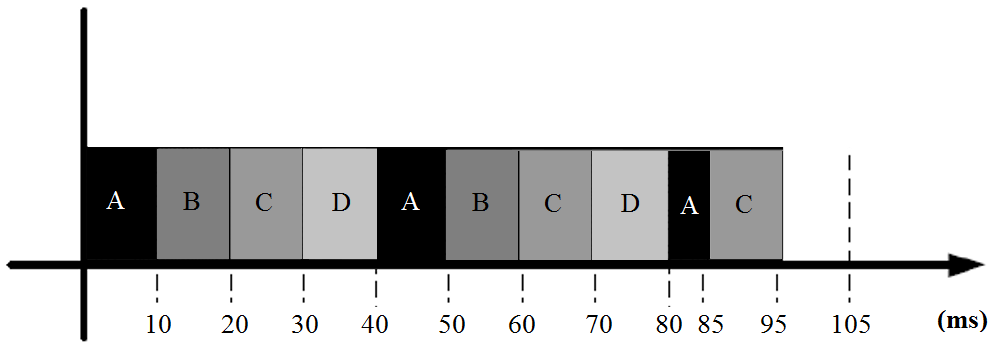


Figura 10: Exemplo de escalonamento com a técnica *Round Robin* para um conjunto de tarefas *A* (*r0*=0, *C*=25, *D*=*T*=100), *B* (*r0*=0, *C*=20, *D*=*T*= 80), *C* (*r0*=0, *C*=30, *D*=*T*=100), *D* (*r0*=0, *C*=20, *D*=*T*=80).

Na figura 10, possuímos um escalonamento utilizando a técnica *Round Robin* para o conjunto de tarefas *A* (*r0*=0, *C*=25, *D*=*T*=100), *B* (*r0*=0, *C*=20, *D*=*T*= 80), *C* (*r0*=0, *C*=30, *D*=*T*=100), *D* (*r0*=0, *C*=20, *D*=*T*=80), e é configurado fatias de tempo de 10ms. Cada tarefa é executada dentro do período de tempo definido. Como destaque na figura temos a tarefa *A* no instante de tempo 80, ela executa por 5ms e é finalizada, liberando o processador para uma nova tarefa.

#### 4.4.1.5 *Least Laxity* (LL)

O algoritmo *Least Laxity* trabalha com prioridades dinâmicas utilizando como critério o menor *laxity* dentre as tarefas pendentes de execução, ou seja, o menor *laxity* possui maior prioridade. O LL é bastante custoso computacionalmente, no entanto, é recomendável para sistemas de tempo real críticos.

O *laxity* é calculado em todos os instantes de tempo e é obtido para cada tarefa pela equação:

Onde (*D*)é o deadline da tarefa, (*S*)é o instante de tempo em que o algoritmo se encontra e (*E*)é o tempo de computação restante para conclusão da tarefa.

Para o exemplo da figura 11, possuímos um conjunto de tarefas *t1* (*r0*=0, *C*=3, *D*=7 e *T*=20), *t2* (*r0*=0, *C*=2, *D*=4 e *T*=5), *t3* (*r0*=0, *C*=1, *D*=8 e *T*=10) e calculando o *laxity* para as tarefas no instante *t=0*, podemos observar:

Logo, a tarefa *t2* possui o menor *laxity* e consequentemente será executada. Caso aconteça igualdade nos valores calculados, o ideal é continuar a tarefa já em execução no instante de tempo anterior.

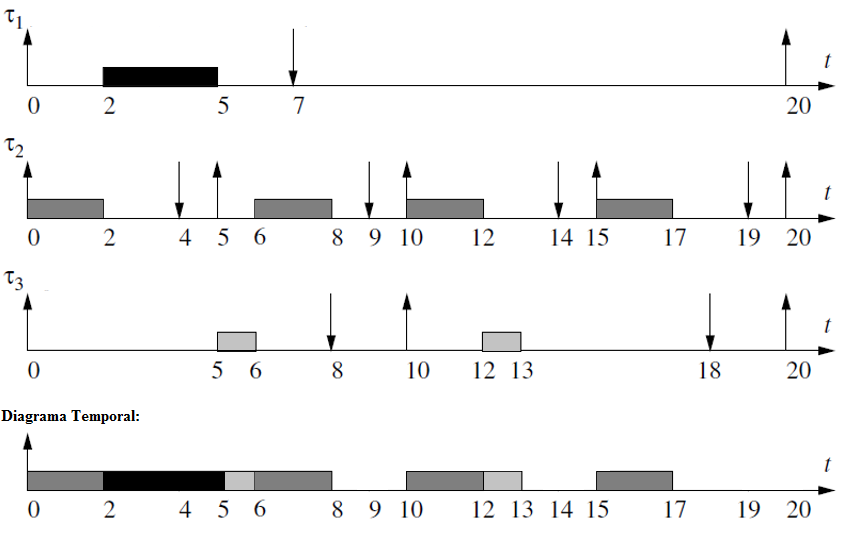


Figura 11: Exemplo de escalonamento com a técnica *LL* para um conjunto de tarefas *t1* (*r0*=0, *C*=3, *D*=7 e *T*=20), *t2* (*r0*=0, *C*=2, *D*=4 e *T*=5), *t3* (*r0*=0, *C*=1, *D*=8 e *T*=10).

### 4.4.2 Tarefas Aperiódicas (Esporádicas)

As técnicas de escalonamento de tarefas aperiódicas apresentadas a seguir trabalham em conjunto com as técnicas descritas anteriormente, aplicando algumas políticas de tratamento para esses tipos de tarefa.

#### 4.4.2.1 Servidores *Background*

Dentre as técnicas de escalonamento de tarefas aperiódicas a mais simples é a servidor *Background*, no entanto, essa simplicidade traz uma baixa performance (i.e., Pode-se ter um alto tempo na resposta da tarefa). Na utilização dessa técnica as tarefas de esporádicas são executadas quando o processador está ocioso, ou seja, quando não se tem nenhuma tarefa periódica pronta para execução.

Caso mais de uma tarefa seja acionada, elas são enfileiradas e executadas nos momentos disponíveis.

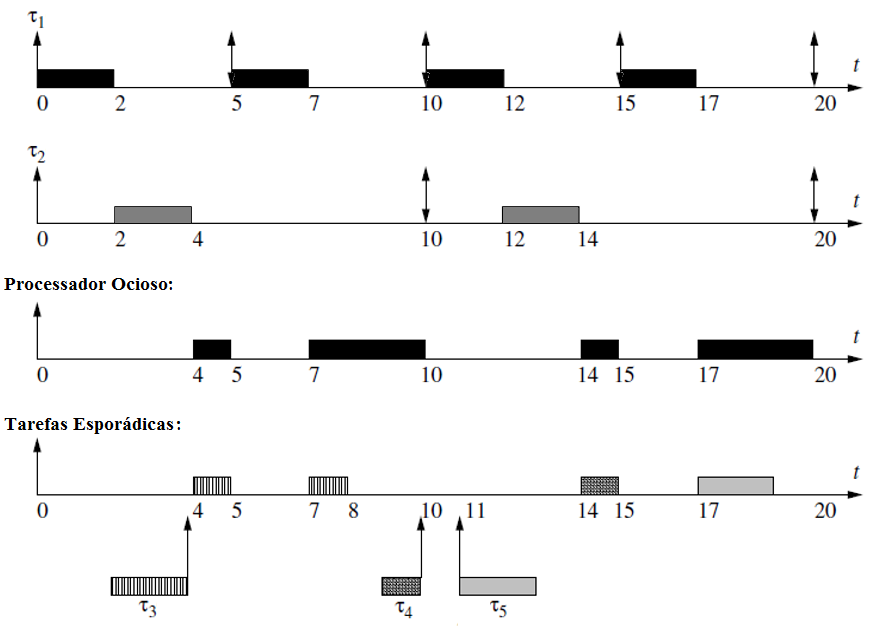


Figura 12: Exemplo de escalonamento com a técnica RM para um conjunto de tarefas periódicas *t1* (*r0*=0, *C*=2, *D*=*T*=5), *t2* (*r0*=0, *C*=2, *D*=*T*=10) e esporádicas *t3* (*r0*=4, *C*=2), *t4* (*r0*=10, *C*=1), *t5* (*r0*=11, *C*=2) utilizando servidor *background*.

Fonte: (COLLET; DELACROIX; KAISER; MAMMERI, p. 34, 2012)

No exemplo da Figura 12, temos o escalonamento de 5 tarefas, 2 periódicas *t1* (*r0*=0, *C*=2, *D*=*T*=5), *t2* (*r0*=0, *C*=2, *D*=*T*=10) e 3 esporádicas *t3* (*r0*=4, *C*=2), *t4* (*r0*=10, *C*=1), *t5* (*r0*=11, *C*=2). Para as tarefas periódicas, observamos que o algoritmo utilizado é o RM, já para as tarefas esporádicas percebemos que sua execução é sempre enquanto o processador está ocioso (i.e., Sem tarefas para executar), e quando temos duas tarefas esporádicas pendentes de execução (*t4* e *t5*) eles são enfileiradas e executadas quando possível.

#### 4.4.2.2 Servidor *Polling*

A técnica de servidor de *polling* trabalha utilizando uma tarefa periódica chamada *Ts* (*Task Server*), onde nessa tarefa são alocadas as tarefas esporádicas. Essa técnica nos fornece um melhor tempo de resposta se comparado a técnica de servidor *background*, pois em intervalos regulares é alocado um *server* para execução das tarefas aperiódicas pendentes, no entanto, ainda não é possível executa-las imediatamente.

Caso o *server* seja acionado e não exista tarefas esporádicas pendentes de execução, sua capacidade é perdida e é dado lugar a outra tarefa periódica pendente de execução seguindo as prioridades do algoritmo de escalonamento.

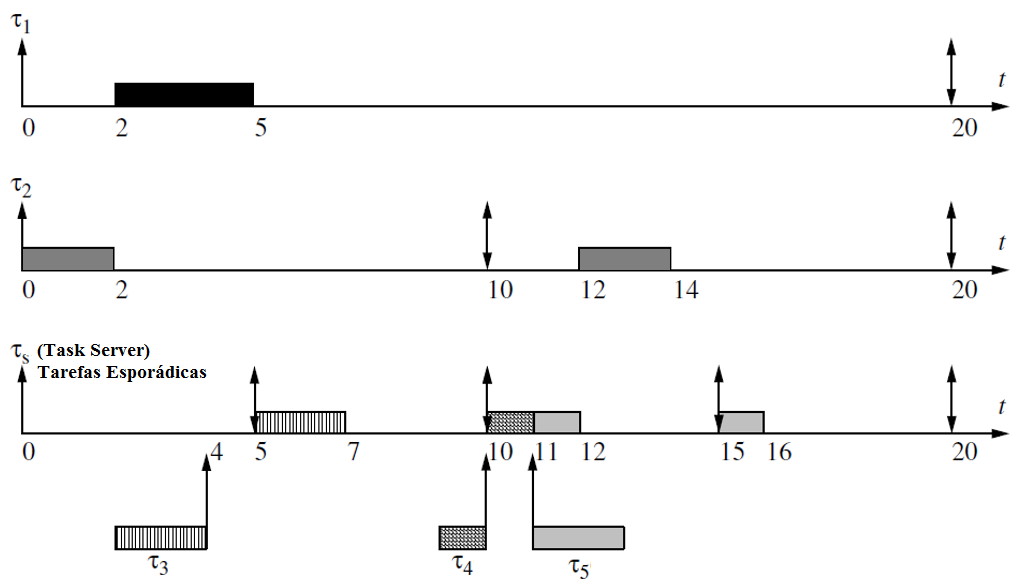


Figura 13: Exemplo de escalonamento com a técnica RM para um conjunto de tarefas periódicas *t1* (*r0*=0, *C*=3, *D*=*T*=20), *t2* (*r0*=0, *C*=2, *D*=*T*=10), *ts* (*r0*=0, *C*=2, *D*=*T*=5) e esporádicas *t3* (*r0*=4, *C*=2), *t4* (*r0*=10, *C*=1), *t5* (*r0*=11, *C*=2) utilizando servidor de *polling*.

Fonte: (COLLET; DELACROIX; KAISER; MAMMERI, p. 34, 2012)

Podemos perceber que na figura 13, a tarefa *ts* é executada por primeiro devido possuir maior prioridade (i.e., Menor período, RM), no entanto, não é executada devido não possuir tarefas esporádicas pendentes de execução.

#### 4.4.2.3 Servidor *Sporadic*

A técnica de servidor *Sporadic* assim como servidores *Background* utiliza uma tarefa periódica (i.e., *Task Server*) para execução das tarefas aperiódicas, no entanto, a capacidade desse servidor não é totalmente restaurada a cada período da tarefa.

Nesta técnica, as tarefas esporádicas possuem alta prioridade, sendo executada no seu instante de ativação, desde que, o servidor possua capacidade para executa-la.

Para esta técnica, a capacidade do servidor só é restaurada no instante de tempo calculado pela formula:

Onde (*t*) equivale ao instante de tempo no qual a tarefa esporádica é executada e (*Ti*) equivale ao período da *Task Server.*

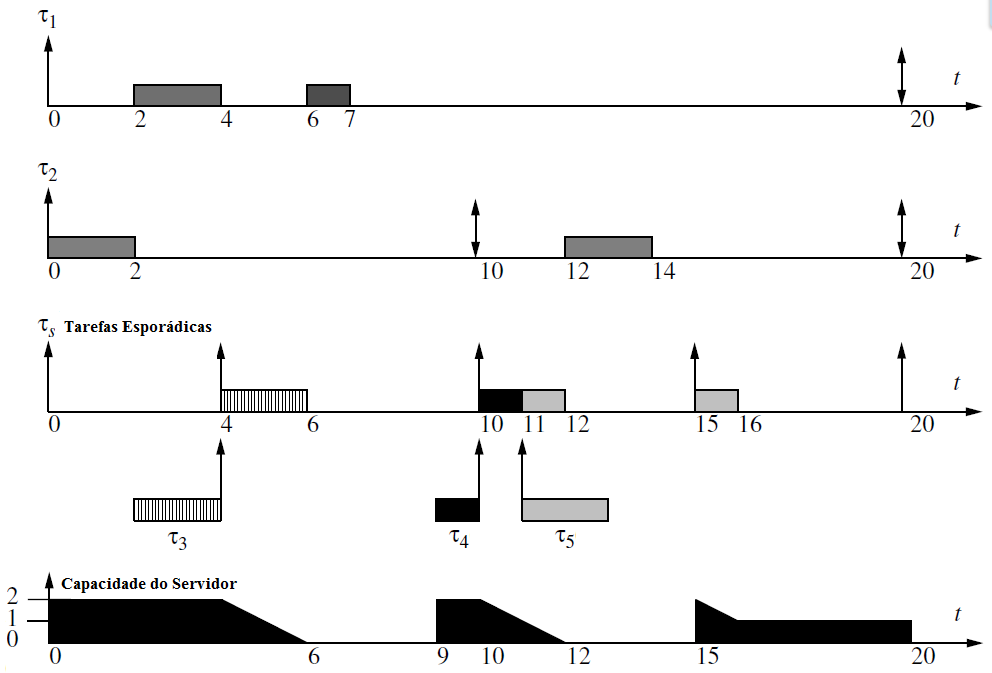


Figura 14: Exemplo de escalonamento de tarefas periódicas *t1 (r0=0, C=3,T=20), t2 (r0=0, C=2, T=10), ts (r0=0, C=2, T=5) e* tarefas esporádicas *t3* (*r0*=4, *C*=2), *t4* (*r0*=10, *C*=1), *t5* (*r0*=11, *C*=2) utilizando servidor *sporadic*.

Fonte: (COLLET; DELACROIX; KAISER; MAMMERI, p. 37, 2012)

Na Figura 14, possuímos um escalonamento de tarefas periódicas e esporádicas utilizando a técnica de servidor *sporadic*. Podemos observar que a capacidade do servidor inicia-se em seu estado máximo até o instante que a tarefa *t3* é iniciada, ela reduz ao valor zero e permanece assim até o instante de tempo 9 (i.e., este valor é obtido pelo cálculo ), o mesmo acontece para a execução das tarefas t4 e t5 no instante de tempo 10.

# 5 DESENVOLVIMENTO DA FERRAMENTA

No desenvolvimento da ferrementa foram implementados os algoritmos de escalonamento para tarefas periódicas: *Rate Monotonic, Deadline Monotonic, Earliest Deadline First, Round Robin, Least Laxity.* Já as políticas de escalonamento de tarefas esporádicas: Servidor *Background*, *Polling* e *Sporadic* foram integradas apenas a técnica EDF.

Para implementação foi utilizada a linguagem Java em conjunto a api JavaFX para geração da UI, MAVEN para gerência das dependências e *build*, e JUnit para desenvolvimento dos testes unitários criados no sistema. Essa implementação foi feita sobre sistema operacional Linux.

## 5.1 TECNOLOGIAS UTILIZADAS

### 5.1.1 Java

Conforme o site da ORACLE (2013): “Java é uma linguagem de programação e uma plataforma de computação lançada pela primeira vez pela Sun Microsystems em 1995. É a tecnologia que capacita muitos programas da mais alta qualidade, como utilitários, jogos e aplicativos corporativos, entre muitos outros”. Ela foi escolhida por ser uma linguagem orientada objetos e multi-plataforma.

### 5.1.2 JavaFX

O JavaFX é um plataforma mantida pela ORACLE para desenvolvimento de interfaces ricas utilizando a linguagem Java. Atualmente se encontra na versão 2 e foi integrada a versão 7 da JDK, já sendo inclusa ao pacote de desenvolvimento.

Com JavaFX é possível escrever uma aplicação e executa-la em ambientes ou dispositivos diferentes (i.e., Desktop, WEB, Mobile, TV, entre outros), sendo esse o principal motivo de sua escolha.

### 5.1.3 Maven

MAVEN é uma ferramenta mantida pela APACHE que é responsável por auxiliar no processo de build e gerência das dependências do projeto. Com o MAVEN não precisamos nos preocupar em anexar *libs* ao nosso repositório, ele possui um arquivo descritor, onde nele são colocadas quais *libs* são utilizadas pela nossa aplicação, e elas são baixadas de repositórios na internet durante o processo de compilação. Permitindo assim fácil troca de versões de *lib* e melhor organização do projeto.

### 5.1.4 JUnit

Conforme o site do JUnit (2013, tradução nossa): “JUnit é um framework simples para escrita de testes repetitivos”. O JUnit nos permite escrever testes unitários utilizando linguagem Java, para testar partes de código analisando resultados esperados por meio de assertivas (i.e., *Asserts*).   
 O JUnit foi incorporado ao projeto para validar os diagramas temporais gerados pelos algoritmos, permitindo que após modificações no algoritmo, testes validem se eles ainda estão gerados valores corretos.

## 5.2 IMPLEMENTAÇÃO DOS ALGORITMOS

Para implementação melhor implementação do projeto e utilização dos recursos da linguagem adotada, foram utilizados alguns conceitos de orientação a objetos como herança, polimorfismo entre outros, que podem ser visualizados no diagrama de classes gerado para a aplicação, conforme figura 15.

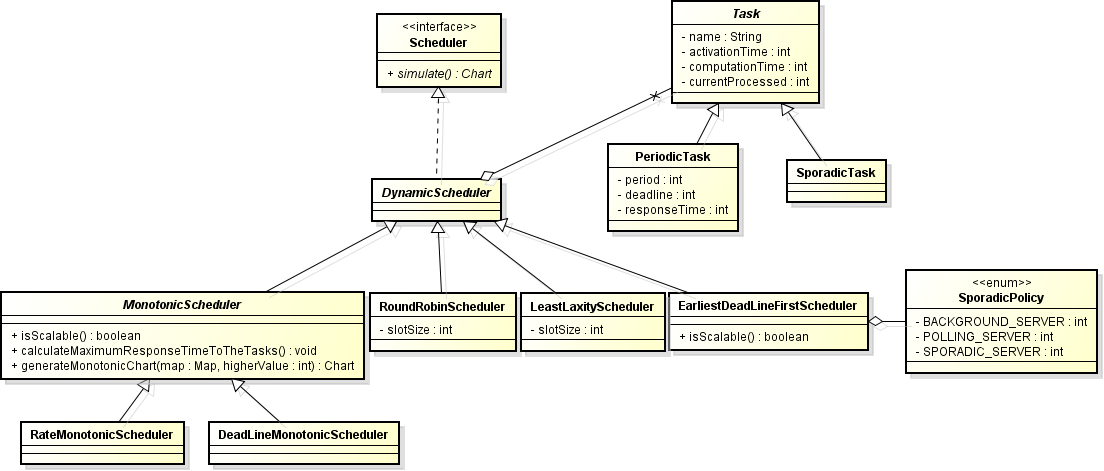


Figura 15: Diagrama de classes gerado para a aplicação.

Como podemos observar na figura 15, foi criada uma interface mais genérica chamada “*Scheduler*” que representa todos os escalonadores de STR e que possuí uma única assinatura de método que é “*simulate*” retornando um objeto do tipo *“Chart*” do JavaFX, abaixo dela temos uma classe abstrata chamada “*DynamicScheduler*” que implementa “*Scheduler*” e que representa os escalonadores dinâmicos, sendo ela composta por uma ou várias instâncias da classe “*Task*”.

Abaixo de “*DynamicScheduler*” temos as classes específicas cada técnica, no qual é composto pelo método “*simulate*” com a implementação da sua lógica correspondente.

Todas as implementações foram feitas pelo autor utilizando estruturas de repetição *for*, *while* e controle de fluxo *if-else*, seguindo a lógica de funcionamento de cada técnica. Nos próximos tópicos serão apresentados fluxogramas com a lógicas em alto nível para o método “*simulate*” implementado por cada escalonador.

### 5.2.1 *Rate Monotonic* (RM)

Para a técnica *Rate Monotonic* foram implementadas duas abordagens: escalonamento com suporte a preempção e sem preempção. As duas abordagens trabalham utilizando como entrada uma lista de tarefas periódicas.

#### 5.2.1.1 *Rate Monotonic* Preemptivo

Conforme podemos observar na figura 16, o algoritmo de escalonamento gerado para o *Rate Monotonic* inicia classificando lista de tarefas pendentes por período (i.e., As tarefas que possuem menor período, possuem maior prioridade. Conforme política de prioridades do algoritmo). Após a ordenação é realizado primeiramente a etapa de testes onde é realizado o teste de escalonabilidade, caso o teste seja satisfeito inicia-se o processamento das tarefas, caso não seja, é realizado o cálculo do tempo máximo de resposta utilizando como profundidade na convergência até 10 tentativas. Se o algoritmo conseguir encontrar os valores, é iniciado o processamento das tarefas, senão o algoritmo é finalizado e é informado ao usuário que não foi possível realizar o cálculo do tempo máximo de resposta.

Após a etapa de testes e caso seja satisfeita, é iniciada a etapa de processamento. Nessa etapa primeiramente é realizada a montagem de um mapa (i.e., chave e valor) onde a chave corresponde ao instante de tempo de ativação e o valor é uma lista de tarefas que são ativadas no instante. Para melhor exemplificar a geração do mapa, supomos que temos 2 tarefas periódicas *t0 (r0=0, C=2, T=D=5)*, *t1 (r0=0, C=2, T=D=10)* e limitando o valor máximo à 20, teríamos um mapa gerado conforme a tabela 01.

|  |  |  |
| --- | --- | --- |
| Instante de Tempo  (chave) | Lista de Tarefas (valor) | |
| 0 | *t0* | *t1* |
| 5 | *t0* |  |
| 10 | *t0* | *t1* |
| 15 | *t0* |  |
| 20 | *t0* | *t1* |

Tabela 01: Representação do mapa com tarefas e tempo de ativação.

Como podemos perceber na tabela 01, o instante de tempo se inicia em 0 e é propagado até um valor definido, no exemplo temos uma progressão de valor 5, no entanto para o algoritmo é gerado checando cada unidade de tempo. É utilizado com base para montagem o período das tarefas, onde no exemplo, percebemos que a tarefa *t0* possuí *T=5*, logo a cada 5 instantes de tempo ela será iniciada novamente.

Para o algoritmo RM o valor máximo utilizado para geração do mapa é o maior período entre as tarefas.

Com o mapa gerado é iniciada uma variável de controle “*position*” com valor 0 e que é incrementada a cada unidade de tempo, ela é responsável por gerenciar o processamento das tarefas, o processamento ocorre até que essa variável atinja o valor do maior período entre as tarefas.

Em cada instante de tempo é verificado se existem tarefas a serem ativadas no mapa, caso existam, elas são movidas para uma lista de tarefas pendentes e caso haja tarefas nessa lista, o algoritmo busca a primeira tarefa da lista, checa se a sua meta temporal já foi atingida e inicia seu processamento sempre observando se outra tarefa não é ativada e sempre incrementando position, caso outra tarefa seja ativada, o algoritmo efetua a preempção da tarefa atual, processa a tarefa que foi ativada em seguida volta a tarefa que estava em execução anteriormente. Após processamento, as tarefas são removidas da lista de tarefas pendentes e o fluxo de verificação de “*position*” é iniciado novamente.

Caso não haja tarefas pendentes de execução o algoritmo incrementa “*position*” em uma unidade de tempo e verifica novamente. Caso ocorram violações de meta temporal o algoritmo é interrompido e é retornado ao usuário a tarefa, o instante de tempo e o gráfico gerado até a violação.

Caso todo o caminho feliz do algoritmo seja executado é retornado para o usuário o diagrama temporal gerado pela técnica.



Figura 16: Fluxograma simplificado para a implementação do algoritmo *Rate Monotonic* preemptivo*.*

#### 5.2.2.2 *Rate Monotonic* Não Preemptivo

A funcionamento do algoritmo RM sem preempção também está dividido em duas etapas, possuindo modificação apenas durante processamento das tarefas.

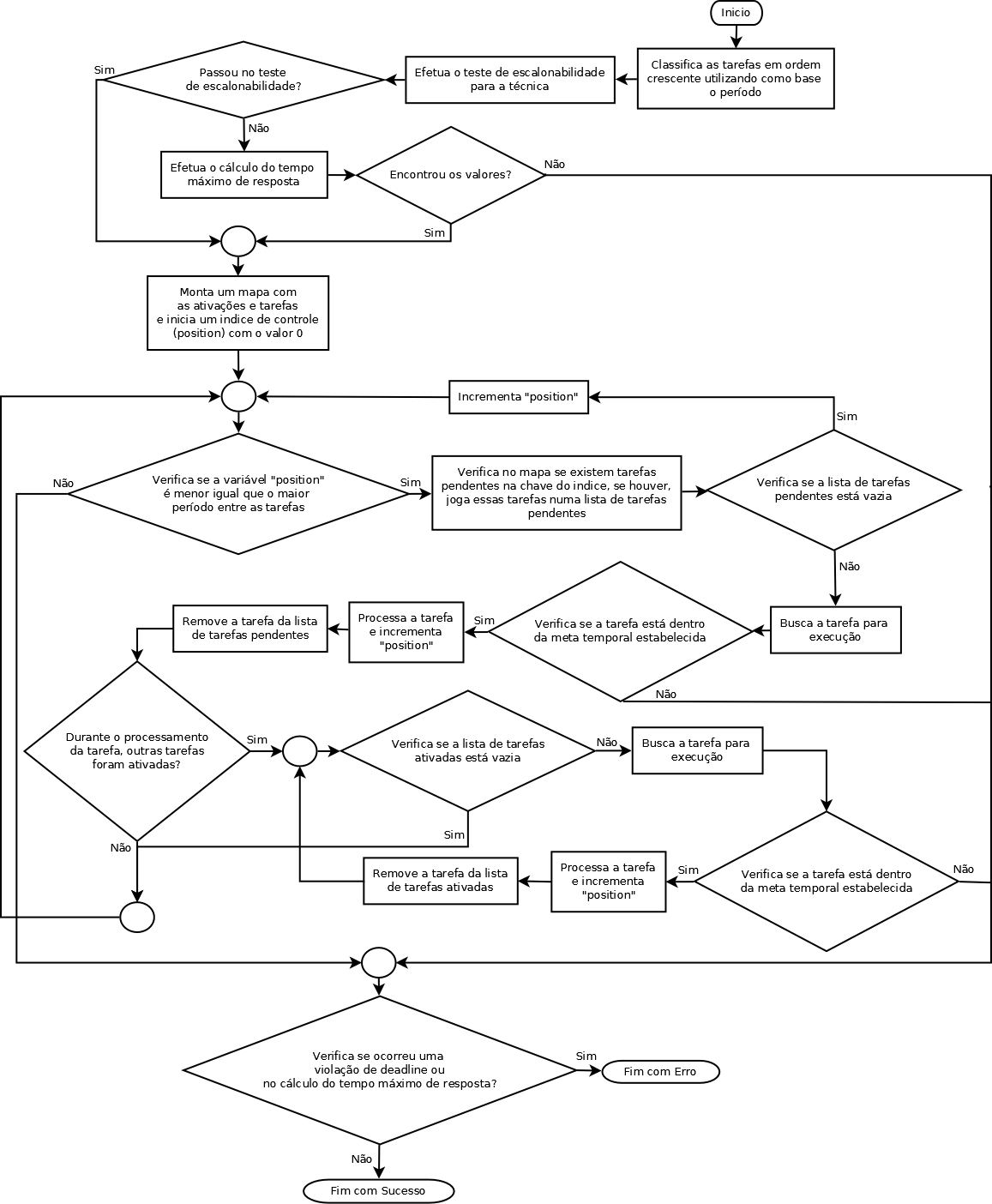


Figura 17: Fluxograma simplificado para a implementação do algoritmo *Rate Monotonic* sem preempção.

Conforme podemos perceber na figura 17, na etapa de processamento também é utilizada uma variável de controle “*position*” e um mapa com as tarefas e suas ativações se interagindo com uma lista de tarefas pendentes. No entanto, durante o processamento de uma tarefa, ao invés de iniciar e observar se outra tarefa é ativada interrompendo o processamento da tarefa atual, as tarefas que são ativadas durante o processamento são enviadas para uma lista de tarefas ativadas e processadas ao final da tarefa corrente.

### 5.2.2 *Deadline Monotonic* (DM)

As implementações geradas para o algoritmo *Deadline Monotonic* são baseadas na implementação do algoritmo *Rate Monotonic*, sofrendo alteração na prioridade das tarefas (i.e., As tarefas com menor *deadline*, possuem maior prioridade. Conforme especificação do algoritmo).

Visando reuso do código gerado, conforme podemos observar na figura 15 foi gerada uma classe abstrata intermediária chamada “*MonotonicScheduler*” e nela foi criado um método “*generateMonotonicChart*” que recebe dois parâmetros, um mapa, que é o mapa de ativações e um valor inteiro que representa o maior valor para geração do gráfico. Esse método que realmente é o responsável por gerar o gráfico para as duas técnicas. O método “*simulate*” de cada técnica gera o mapa, e obtém o maior valor seguindo sua política (i.e., maior *deadline* ou maior período) em seguida chama ele para gerar o diagrama temporal.

As modificações na política de prioridade se comparadas ao RM podem ser observadas nas figuras 18 e 19 que demonstram a implementação do algoritmo DM com e sem preempção.

#### 5.2.2.1 *Deadline Monotonic* Preemptivo

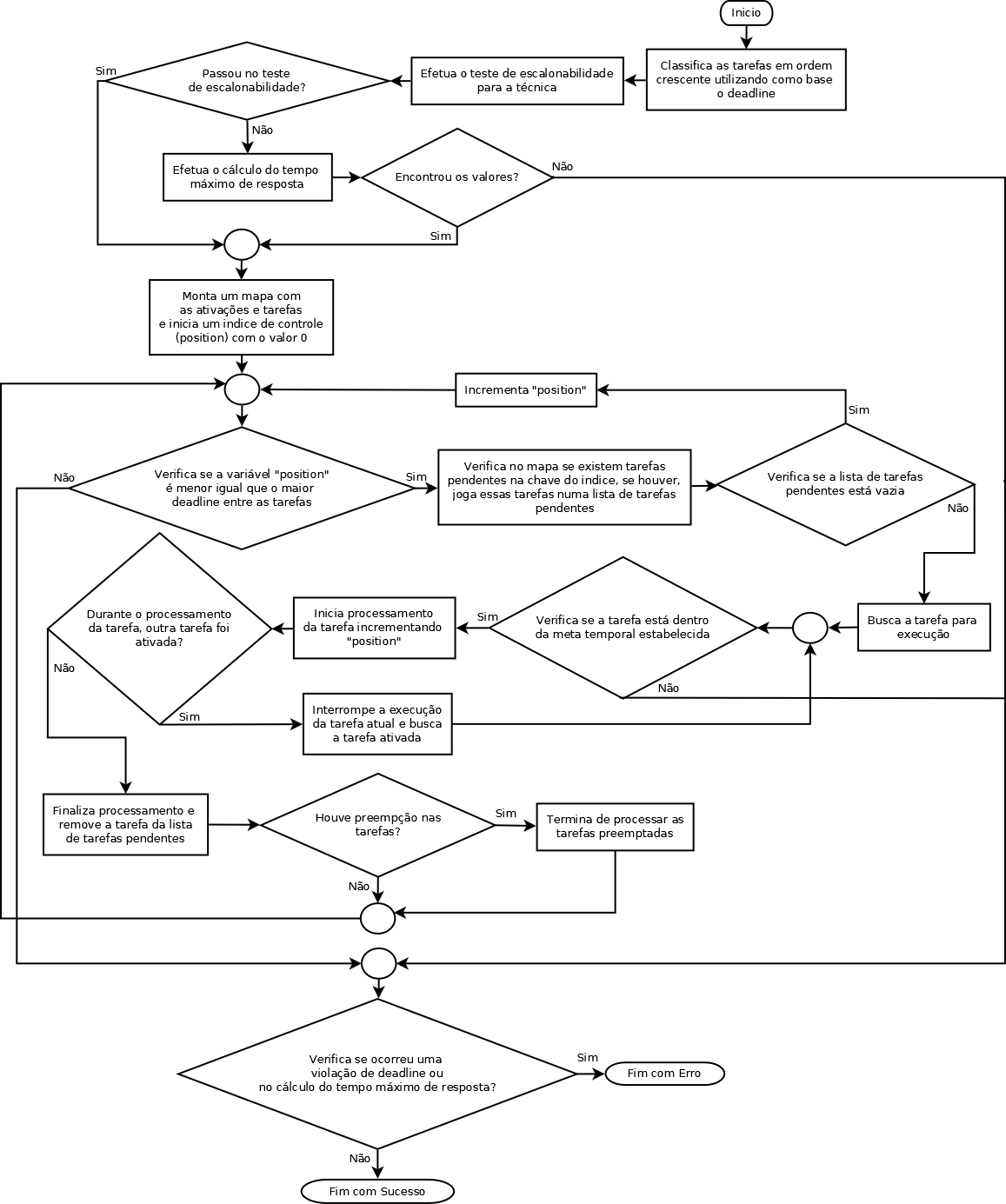


Figura 18: Fluxograma simplificado para a implementação do algoritmo *Deadline Monotonic* preemptivo*.*

#### 5.2.2.2 *Deadline Monotonic* Não Preemptivo

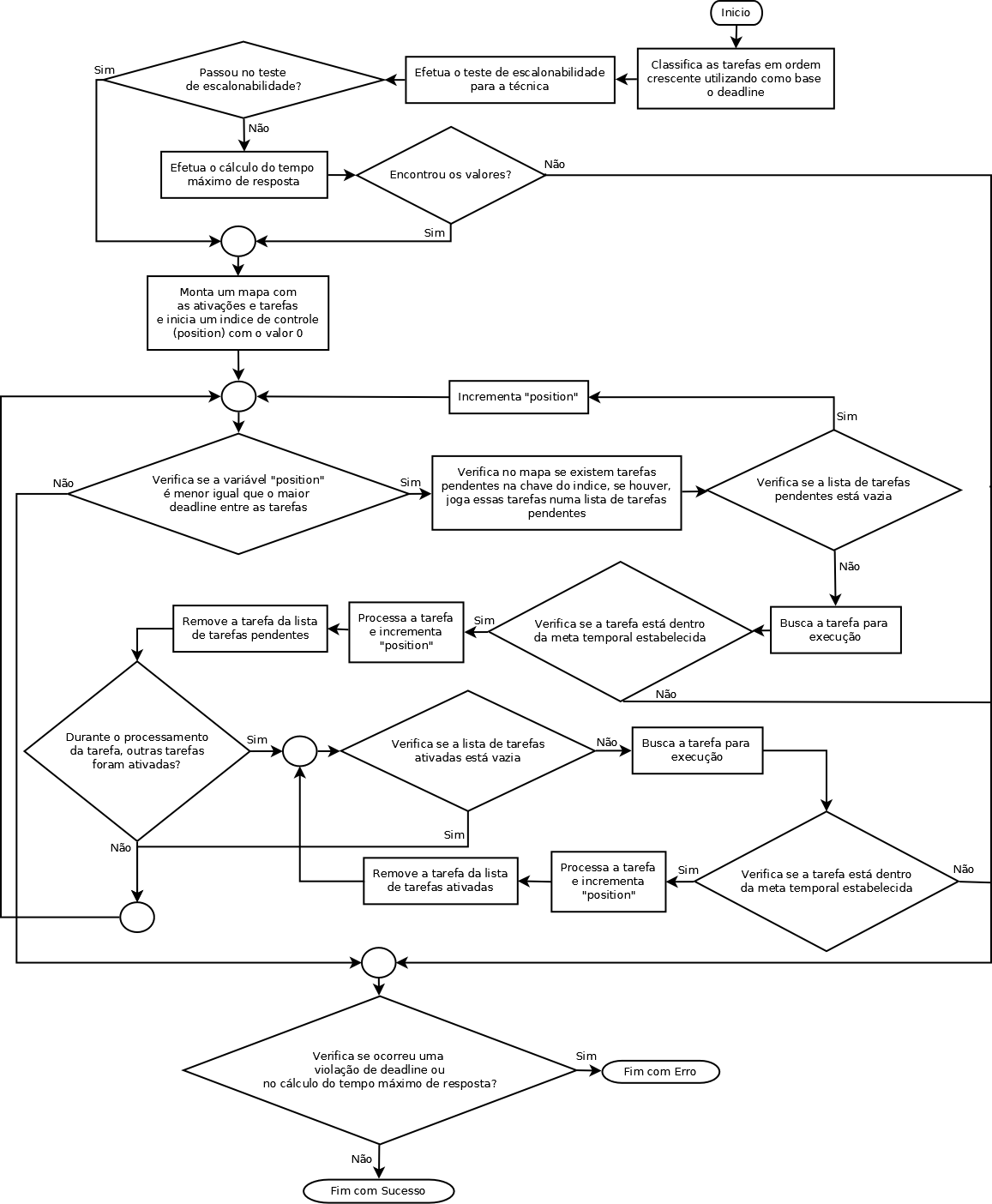


Figura 19: Fluxograma simplificado para a implementação do algoritmo *Deadline Monotonic* não preemptivo*.*

### 5.2.3 *Earliest Deadline First* (EDF)

Para o algoritmo implementado utilizando a técnica *Earliest Deadline First* além do escalonamento utilizando tarefas periódicas, foi implementado também técnicas de suporte a tarefas esporádicas (i.e., Foram implementados as técnicas de servidor *Background*, servidor de *Polling* e servidor *Sporadic*). Para apresentação dessa implementação visando facilitar o entendimento, serão demonstrados fluxogramas desmembrados com a lógica específica de cada combinação, no entanto, para a aplicação um único algoritmo contempla todos os comportamentos.

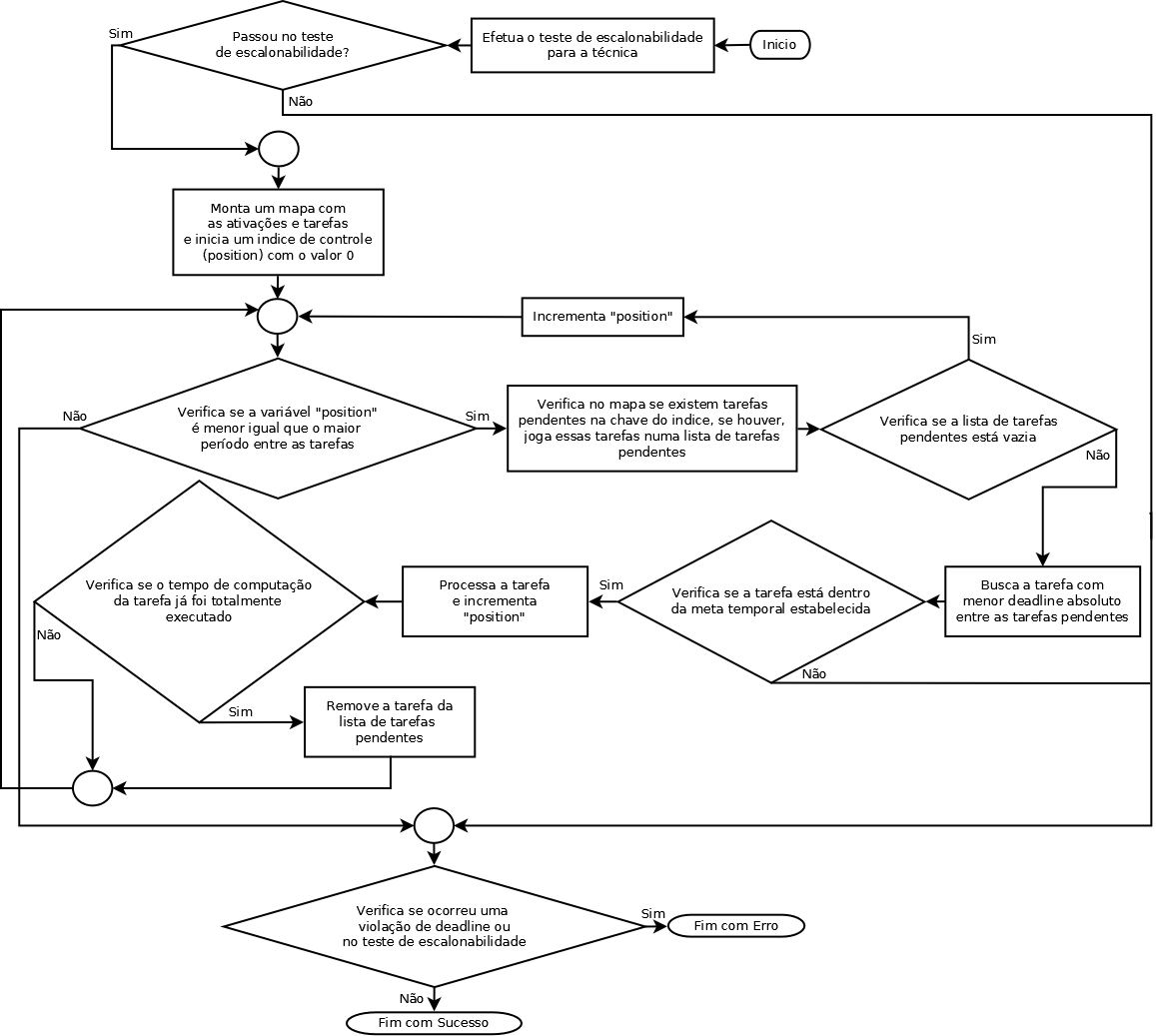


Figura 20: Fluxograma simplificado para a implementação do algoritmo EDFapenas para tarefas periódicas.

Na figura 20 temos um fluxograma simplificado que representa o algoritmo gerado para a técnica EDF utilizando apenas tarefas periódicas. Esse algoritmo utiliza como entrada uma lista de tarefas periódicas.

A execução desse algoritmo tem início com a realização do teste de escalonabilidade para as tarefas, caso não seja satisfeito o algoritmo é encerrado e é informado ao usuário o motivo, caso o teste seja satisfeito é gerado um mapa com as ativações e suas respectivas tarefas e é iniciada uma variável de controle “*position*” com o valor 0 e que é incrementada até o maior valor de período entre as tarefas. Em cada instante de “*position*” é verificado no mapa se existem tarefas a serem ativadas, caso existam elas são movidas para uma lista de tarefas pendentes, após isso é verificado se a lista de tarefas pendentes está vazia, caso esteja vazia, “*position*” é incrementada em uma unidade de tempo e o processo é repetido, caso existam tarefas na lista de tarefas pendentes, é verificado qual delas possuem o menor deadline absoluto, ou seja, qual das tarefas está mais próxima de atingir sua meta temporal, para essa tarefa é verificada se o seu *deadline* já foi ultrapassado, caso não, a tarefa é processada em 1 unidade de tempo, “*position*” é incrementado e caso o seu tempo de computação já tenha sido alcançado ela é removido da lista de tarefas pendentes. Esse processo é realizado até que “*position*” alcance o valor do maior período entre as tarefas.

Caso ocorra uma violação de *deadline*, é informado ao usuário a tarefa, a posição e o gráfico gerado até que a violação ocorresse.

#### 5.2.3.1 Earliest ***Deadline*** First com Servidor *Background*

A implementação do EDF utilizando servidor *Background* adiciona a técnica o suporte a tarefas esporádicas, para esse algoritmo é utilizado como entrada uma lista de tarefas periódicas e esporádicas. Para facilitar a implementação, podemos observar no diagrama de classes da figura 15 que foi gerado um esquema com uma classe abstrata chamada “*Task*” que possui duas implementações “*PeriodicTask*” e “*SporadicTask*”. Observamos também que “*DynamicScheduler*” possuí uma lista de “*Task*”, no entanto, em tempo de execução, caso alguma tarefa esporádica tenha sido adicionada e outra técnica diferente de EDF tenha sido selecionada, é informado ao usuário que somente EDF possui suporte a tarefas esporádicas.

Conforme a figura 21, temos a implementação do algoritmo EDF com suporte ao servidor *background*, podemos observar que a implementação é bem similar a gerada para somente tarefas periódicas, no entanto, ao invés de ser gerado apenas um mapa para as tarefas periódicas, são gerados dois mapas, um para as tarefas periódicas e outro para as tarefas esporádicas. Nesse algoritmo também são geradas duas listas de tarefas pendentes uma para cada tipo respectivo e a lista de tarefas periódicas é que define se as tarefas esporádicas serão executadas (i.e., Caso a lista de tarefas periódicas pendentes esteja vazia é considerado que o processador está ocioso, e neste ponto é que entra a atuação do servidor *background*). Caso o processador esteja ocioso e existam tarefas esporádicas pendentes de execução, elas são executadas e caso completadas removidas da lista de pendências. Lembrando que a execução só é feita uma unidade de tempo por vez. Caso não existam tarefas esporádicas a serem executadas a variável de controle “*position*” é simplesmente incrementada e o processo é repetido.

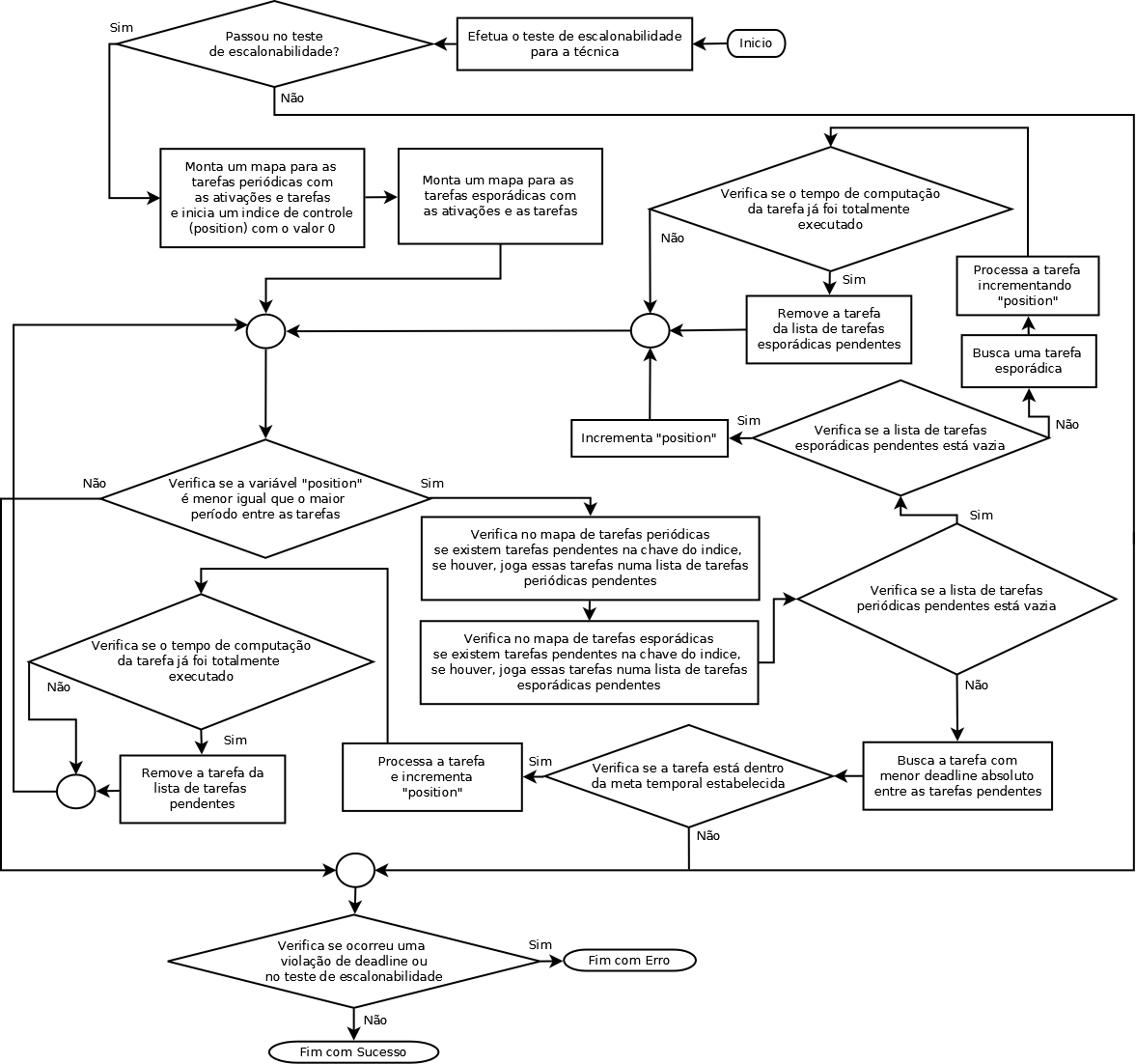


Figura 21: Fluxograma simplificado para a implementação do algoritmo EDFcom suporte a tarefas esporádicas utilizando servidor *background*.

#### 5.2.3.2 *Earliest Deadline First* com Servidor de *Polling*

#### *5.2.3.3 Earliest Deadline First* com Servidor *Sporadic*

### 5.2.4 *Round Robin*

Para a técnica *Round Robin* é necessário que o usuário informe o *Slot Size* que corresponde a fatia de tempo definida pelo projetista. Caso o usuário não informe esse valor é definido por padrão o valor 1. Para o algoritmo é utilizado como entrada: a lista de tarefas periódicas e o tamanho do *slot*.

Na figura 19 temos descrito o processo simplificado utilizado pelo algoritmo para geração do diagrama temporal.

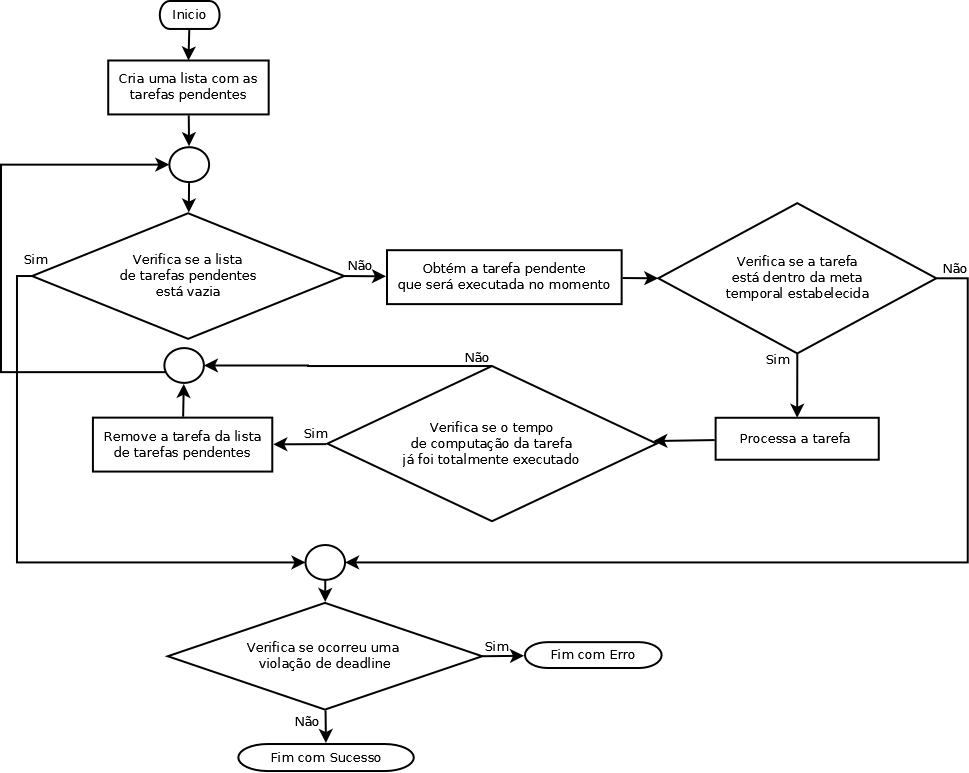


Figura 19: Fluxograma simplificado para a implementação do algoritmo *Round Robin.*

Conforme figura 19, percebemos que o algoritmo implementado para a técnica *Round Robin* fica ativo enquanto houver tarefas pendentes para execução, essas tarefas pendentes são enfileiradas e durante a execução de cada tarefa é sempre verificado se ela ainda está dentro da meta temporal estabelecida, caso não esteja, o algoritmo é finalizado e é lançada uma exceção no Java contendo a posição no eixo X do gráfico, a tarefa e o gráfico gerado. Outro ponto importante para destacar é o processamento da tarefa, caso o tempo de computação restante dela seja maior que o *Slot Size*, é processado apenas *Slot Size,* caso ele seja menor, é processado o que ainda resta na tarefa. Após o processamento, caso essa tarefa ainda não tenha sido completada, ela irá para o final da fila dando vez a próxima.

### 5.2.5 *Least Laxity* (LL)

A implementação do algoritmo *Least Laxity* por padrão não requer um *Slot Size*,no entanto, a fim de facilitar a execução de exemplos didáticos baseados em livros adotamos que ele também usaria este artifício, ficando assim um pouco semelhante ao *Round Robin.* Esse algoritmo utiliza como entrada: uma lista de tarefas pendentes e o tamanho do *slot.* Caso o tamanho do *slot* não seja informado pelo usuário é adotado o valor padrão 1.

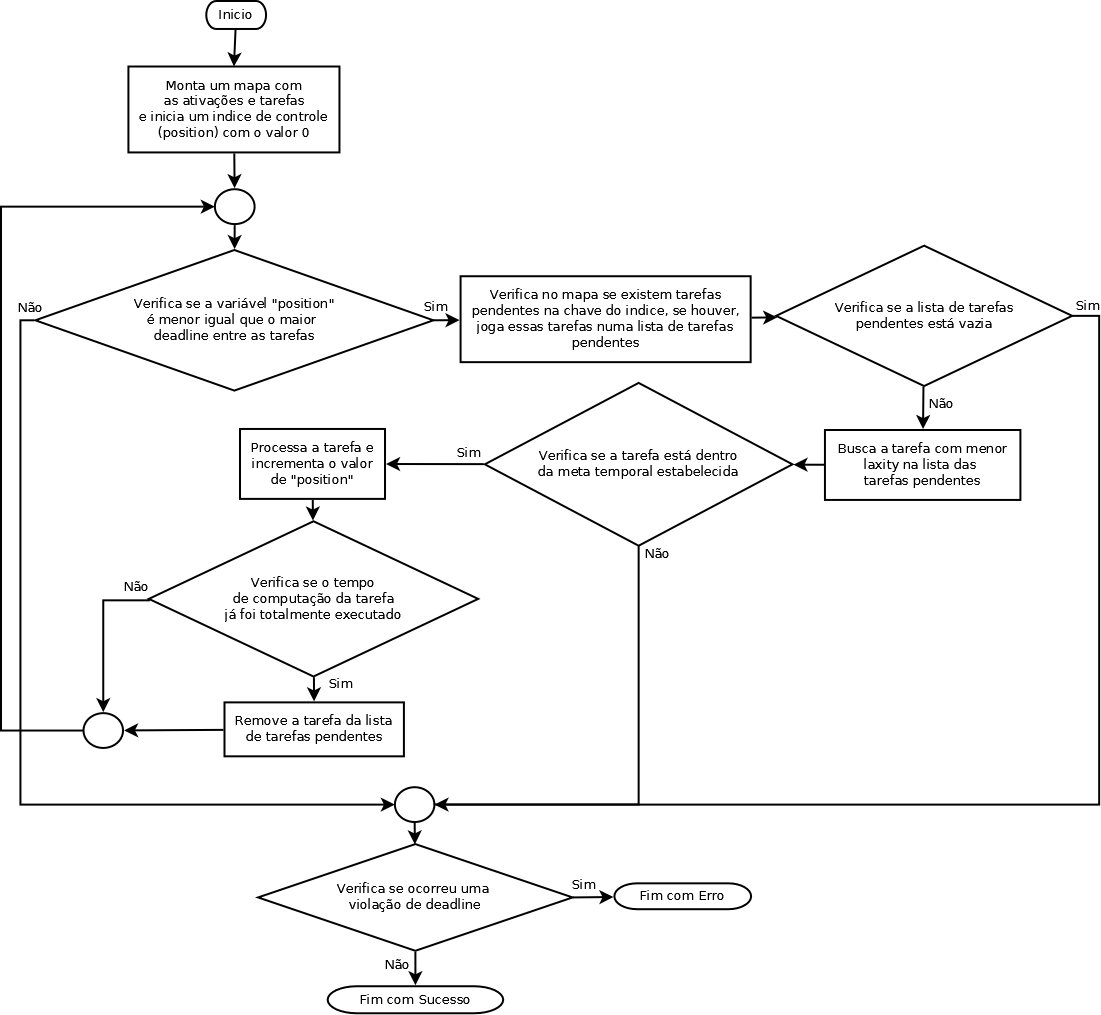


Figura 20: Fluxograma simplificado para a implementação do algoritmo *Least Laxity*.

Na figura 20 podemos observar a ideia básica da implementação gerada para o algoritmo *Least Laxity,* onde a partir de uma lista de tarefas pendentes é gerado um mapa (i.e., chave e valor) onde a chave corresponde ao instante de tempo de ativação e o valor é uma lista de tarefas que são ativadas no instante. Para melhor exemplificar a geração do mapa, supomos que temos 2 tarefas periódicas *t0 (r0=0, C=2, T=D=5)*, t1 (r0=0, C=2, T=D=10) e limitando o valor máximo à 20, teríamos um mapa gerado conforme a tabela 01.

|  |  |  |
| --- | --- | --- |
| Instante de Tempo  (chave) | Lista de Tarefas (valor) | |
| 0 | *t0* | *t1* |
| 5 | *t0* |  |
| 10 | *t0* | *t1* |
| 15 | *t0* |  |
| 20 | *t0* | *t1* |

Tabela 01: Representação do mapa com tarefas e tempo de ativação.

Como podemos perceber na tabela 01, o instante de tempo se inicia em 0 e é propagado até um valor definido, no exemplo temos uma progressão de valor 5, no entanto para o algoritmo, é gerado checando cada unidade de tempo. É utilizado com base para montagem o período das tarefas, onde no exemplo, percebemos que a tarefa *t0* possuí *T=5*, logo a cada 5 instantes de tempo ela será iniciada novamente.

Com o mapa gerado é utilizada uma variável de controle “*position*” que é iniciada com 0 e tem seu valor máximo o maior *deadline* dentre as tarefas, o maior *deadline* também é utilizado como valor limite para geração do mapa. A cada instante de “*position*” é verificado se existem tarefas no mapa, se existirem, essas tarefas são levadas a uma lista de tarefas pendentes no qual é retirada a tarefa com menor *laxity* para processamento. Para essa tarefa é verificado a sua meta temporal e caso esteja dentro do especificado é processado o tamanho do *Slot Size* ou o restante de tempo de computação da tarefa, feito isso, caso ela já tenha sido processada por completo ela é removida da lista de pendentes, “position” é incrementado com o valor processado e o fluxo começa novamente.

# 3 CRIPTOGRAFIA

A palavra criptografia é formada a partir dos termos gregos *kryptos* (escondido, oculto) e *graphé* (grafia, escrita) e é a parte da ciência que torna possível a comunicação segura entre dois agentes, entidades ou sistemas sobre um canal aberto, convertendo informação legível em algo sem sentido, com a capacidade de ser recuperada ao estado original com o uso de processos inversos ou não. [5].

A criptografia estuda de técnicas matemáticas relacionada a aspectos de segurança da informação como confidencialidade, integridade de dados, autenticação de entidade e autenticação de origem de dados. [7]

Além da criptografia é parte da criptologia que também engloba a criptoanálise, que é a arte de obter a informação escondida em um texto cifrado, sem que exista o acesso à chave secreta requerida para tal informação. Juntas, a criptografia e a criptoanálise formam a ciência chamada de Criptologia.

## 3.1 OBJETIVOS DA CRIPTOGRAFIA

Muitos objetivos de segurança podem ser citados, porém existem quatro principais dos quais outros podem ser derivados [6]. São eles:

1. **Confidencialidade**: é um serviço utilizado para esconder o conteúdo da informação de todos que não tem autorização para tê-las. Sigilo é um termo sinônimo à confidencialidade e privacidade. Existem diversas abordagens para prover confidencialidade, desde proteção física a algoritmos matemáticos que tornem os dados ilegíveis.
2. **Integridade de dados**: é um serviço que identifica a alteração não autorizada de dados. Para assegurar a integridade de dados, a entidade deve possuir a possibilidade de detectar a manipulação dos dados por entidades não autorizadas. A manipulação de dados inclui inserções, remoções e substituições.
3. **Autenticação**: é um serviço relacionado à identificação. Essa função é aplicável tanto à entidade quanto informações. Duas entidades começando uma comunicação devem identificar uma à outra. A informação enviada através do canal deve ser autenticada quanto à origem, data de origem, conteúdo dos dados, horário enviado, etc. Por essa razão essa área da criptografia geralmente é subdividida em duas grandes áreas: Autenticação de Entidade e Autenticação de origem de dados. Autenticação de origem de dados implica a integridade dos dados (se as informações forem modificadas, a sua origem foi alterada).
4. **Não-Repúdio**: é um serviço que previne uma entidade negar uma ação ou comprometimento passado. Quando uma disputa surge devido uma entidade negar que algumas ações foram feitas, uma maneira de resolver a situação é necessária. Por exemplo, uma entidade pode autorizar uma compra de propriedade por outra entidade e, mais tarde, negar que tal autorização foi dada. Um processo envolvendo uma terceira entidade confiável é necessário para resolver a disputa.

Um objetivo fundamental da criptografia é implementar adequadamente essas quatro áreas tanto na teoria quanto na prática. A criptografia trata de prevenção e detecção de fraudes e outras atividade maliciosas.

## 3.3 MODELO DE UM SISTEMA CRIPTOGRÁFICO DE CHAVE SIMÉTRICA

O sistema da Figura 7 representa um Modelo de Sistema Criptográfico de Chave Simétrica, onde é utilizada uma mesma chave k, tanto na encriptação quanto na decriptação.

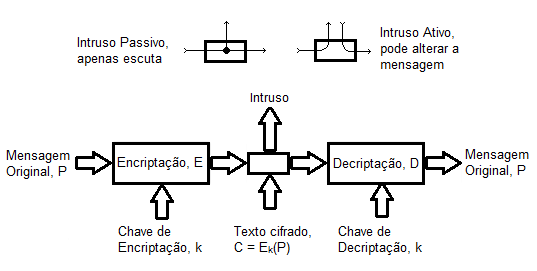


Figura 7: Sistema Criptográfico de Chave Simétrica

O sistema inicia quando se deseja enviar uma mensagem *P* sigilosa para um destino. Essa mensagem original é conhecida como texto simples. No bloco Método de Criptografia, E, essa mensagem é transformada por uma função que é parametrizada por uma chave *k*. A saída desse bloco é chamada texto cifrado *C*. Podemos definir então:

(3.1)

O texto cifrado C é enviado por um canal que não possibilita a segurança da informação. Nesse momento podem existir dois tipos de intrusos tentando recuperar a informação: o intruso passivo e o intruso ativo.

O intruso passivo somente escuta o canal de comunicações e copia todo o conteúdo do texto cifrado. Entretanto, ele não compartilha a chave *k*, e terá certa dificuldade para decifrar o texto cifrado, dependendo do método empregado no primeiro bloco.

O intruso ativo também escuta o canal de comunicações, mas, diferentemente do intruso passivo, ele captura a mensagem, impedindo que a mesma chegue ao destinatário. O objetivo disso é tentar decifrar o texto cifrado e injetar ou modificar as informações contidas no texto cifrado, encriptar essa nova mensagem e enviar ao seu destinatário como se fosse legítima.

No último bloco, Método de Decriptografia, D, é feito o processo inverso ao processo de criptografia, utilizando a mesma chave *k*, para recuperar a mensagem original *P*.

(3.2)

Essa notação sugere que as funções e são simplesmente funções matemáticas, o que é verdade [7].

## 3.4 MÉTODOS DE CRIPTOGRAFIA DE CHAVE SIMÉTRICA

Existem basicamente duas técnicas de criptografia simétrica: a cifra de substituição e a cifra de transposição (ou permutação), também conhecidas como cifras de blocos. Ambos podem atuar *bit*-a-*bit* ou em blocos de dados.

## 3.4.1 Técnica de Substituição

Em um cifra de substituição, cada letra ou grupo de letras é substituído por outra letra ou grupo de letras, de modo a criar um disfarce. Uma das formas de cifra de substituição mais antiga é a chamada Código de César. Foi utilizada para enviar mensagens cifradas na Roma antiga por Júlio César. [7,8]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Código de César | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Pi | a | b | c | D | e | f | g | h | I | J | k | l | M | n | o | p | q | r | s | t | u | v | w | x | y | z |
|  | d | e | f | G | h | I | J | k | L | M | n | o | P | q | r | s | t | u | v | w | x | y | z | a | b | c |

Tabela 1: Substituições efetuadas no Código de César.

No código de César, cada letra é substituída por uma letra três posições à frente no alfabeto. Generalizando a cifra de substituição temos:

Onde:

*Ei* = a letra i encriptada

*Pi* = a letra i da mensagem original

*k* = deslocamento de posições no alfabeto

*n* = tamanho do alfabeto

Notamos então que o Código de César é uma cifra de substituição com valor de *k* = 3 e *n* = 26.

A primeira vista podemos pensar que a cifra de substituição é segura e que um intruso teria dificuldade de testar todas as possíveis combinações. Porém, técnicas de criptoanálise modernas podem descobrir a mensagem utilizando um texto de volume cifrado relativamente pequeno. Para tal, o criptanalista necessita saber o idioma em que a mensagem original foi escrita e faz uma análise de freqüência de ocorrência de letras, diagramas e trigramas da mensagem cifrada, que é igual à freqüência do idioma original. Essa forma de cifra de substituição é chamada cifra de substituição monoalfabética. [6]

### 3.4.2 Cifra de Transposição

Na cifra de substituição, as letras são trocadas por outras letras para disfarçar as palavras, porém a ordem das letras na palavra continua a mesma. Na cifra de transposição, porém não há troca de letras, mas a reordenação delas. Além disso, a chave utilizada não contém nenhuma letra repetida.

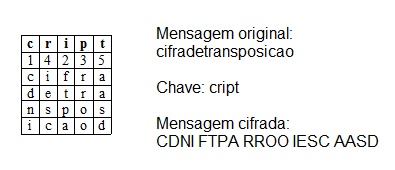


Figura 8: Exemplo de cifra de Transposição.

A Figura 8 mostra um exemplo de cifra de transposição. A chave utilizada na encriptação é “cript”. Na cifra de transposição, a chave tem dois objetivos: o primeiro é definir o número de colunas em que será escrito a mensagem original e a numeração das colunas. A numeração das colunas é feita crescentemente, iniciando com 1 na coluna que possui a letra da chave mais próxima do início do alfabeto, assim por diante, até que se tenha feito a seqüência numérica para todas as letras da chave.

Após a numeração das colunas, a mensagem original é escrita de tal forma que cada letra da mensagem ocupe uma célula da linha, caso a linha tenha sido totalmente preenchida e a mensagem não tenha chegado ao fim, continua-se a escrever a mensagem na próxima linha. Ao fim, caso a mensagem tenha acabado e células de uma linha ficaram vazias, preenchem-se tais células com letras quaisquer. Na Figura 8, a mensagem não ocupou todos os espaços da última linha e adicionou uma letra d ao final.

Após o preenchimento da tabela, escrevem-se as colunas na ordem da numeração inicial. Espaçam-se igualmente os grupos de letras para tentar dificultar onde inicia e termina cada palavra da mensagem original.

Para decifrar a mensagem cifrada utilizando a chave basta fazer o processo inverso, numerando as colunas a partir da chave e escrevendo os grupos de letras em cada coluna e lendo a mensagem nas linhas.

A quebra de uma mensagem cifrada possui uma facilidade maior devido ao fato de que as letras continuam com a mesma freqüência de ocorrência do idioma original, pois só foram reordenadas.

### 3.4.3 Principais Algoritmos de Criptografia de Chave Simétrica

Algoritmos modernos utilizam as mesmas idéias básicas das cifras de substituição e transposição, porém a ênfase é diferente. Antigamente, os métodos de criptografia eram simples e a segurança estava contida no tamanho muito longo da chave, porém atualmente a idéia é diferente: os algoritmos atuais são feitos de forma a possuir chaves relativamente pequenas, porém tão complexos e emaranhados que um criptoanalista, nem mesmo com quantidades enormes de texto cifrado de sua própria escolha, seja capaz de quebrar o código.

Os algoritmos mais utilizados atualmente para transmissão de dados criptografados são os algoritmos DES e AES. Ambos são padrões norte-americanos definidos pela NIST (*National Institute of Standards and Technology*) através das FIPS (*Federal Information Processing Standard*) PUB 46-3 e PUB 197, para TDES e AES, respectivamente.

#### 3.4.3.1 Data Encryption Standard (DES)

O algoritmo DES foi desenvolvido em 1976 pela IBM. Foi adotado como padrão de encriptação de dados pelos Estados Unidos e, posteriormente, em muitos lugares do mundo,. A forma original do DES não é mais segura devido ao aumento de poder computacional crescente no mundo. Dessa forma, em 1998, foi substituído pelo TDEA (*Triple Data Encryption Algorithm*), que aplica o DES três vezes em cada bloco de dados.

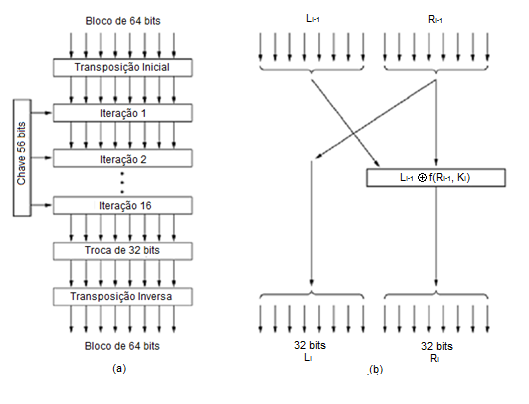


Figura 9: Algoritmo DES. (a) Passos do algoritmo. (b) Detalhe de uma iteração.

Na Figura 9(a), é mostrado o algoritmo do DES. O algoritmo inicialmente quebra uma mensagem em blocos de 64 bits e criptografa cada bloco separadamente. Ao final, o algoritmo junta todos os blocos.

O algoritmo é parametrizado por uma chave de 56 bits e possui 19 estágios distintos [7]. O passo inicial é uma transposição no bloco da mensagem original de 64 bits, e o último passo é o inverso da transposição. O penúltimo bloco faz uma troca dos 32 bits à esquerda pelos 32 bits à direita. Os 16 blocos intermediários são essencialmente idênticos, parametrizados pela chave. Na decriptação, somente aplica-se o algoritmo na ordem inversa.

Os 16 blocos intermediários são parametrizados por chaves diferentes, derivadas da chave original. A chave original de 56 bits é dividida em duas partes de 28 bits e cada parte é rotacionada à esquerda um número de bits que depende do número atual do bloco de iteração.

O funcionamento desses blocos intermediários é mostrado na Figura 9(b). Cada bloco usa duas entradas de 32 bits e produz duas saídas de 32 bits. Como mostrado na figura, a saída da direita é uma cópia da entrada da esquerda. A saída da direita é formada pelo resultado do OU-EXCLUSIVO *bit*-a-*bit* aplicado entre a entrada da esquerda e uma função f que tem como entrada, a entrada da direita e a chave rotacionada desse bloco. A complexidade do algoritmo reside no procedimento interno dessa função, chamada Função de Feistel.

A Função de Feistel possui quatro estágios: expansão, mistura da chave, substituição e permutação, como é mostrada na Figura 10. Na expansão, a entrada da direita de tamanho 32 bits é expandida, de acordo com uma transposição fixa e de uma regra de duplicação, em 48 bits. Esses 48 bits passam por um processo de OU-EXCLUSIVO com a chave específica do bloco. Esse estágio é chamado de mistura de chave.

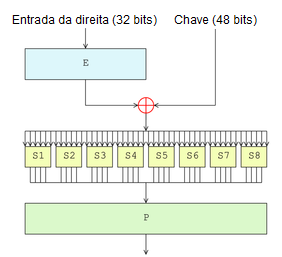


Figura 10: Função de Feistel

No estágio de substituição, esses 48 bits são divididos em 8 grupos de 6 bits, que passam por um processo de substituição nas caixas S1 a S8 mostradas na Figura 10. Cada caixa mapeia um grupo de 6 bits em grupos de 4 bits. É utilizado um método de transformação não-linear no mapeamento que proporciona a segurança do algoritmo[6]. No último estágio, de permutação, esses 8 grupos de 4 bits passam por uma permutação fixa.

#### 3.4.3.3 Advanced Encryption Standard (AES)

Com o passar do tempo, o DES, mesmo com o TDEA, tornou-se relativamente fraco frente a nova geração de computação paralela. Em 1998, o computador *EFF DES Cracker*, custando em torno de US$250.000, quebrou uma mensagem cifrada em DES em 56 horas [9]. Em 2006, o computador COPACOBANA, quebrou uma mensagem cifrada em DES em 6,4 dias, com um custo de US$10.000 [10]. O sucessor do COPACOBANA, RIVYERA, diminui o tempo para menos de um dia [11].

Em 1997, o NIST patrocinou um concurso de criptografia que iria eleger o novo padrão de criptografia a ser utilizado pelas agências norte-americanas. Foram feitas 15 propostas e organizadas conferências públicas para apresentação e análise de cada algoritmo a procura de falhas. Em outubro de 2000, o algoritmo Rijndael (de Joan Daemen e Vicent Rijmen) foi escolhido para ser o próximo padrão de encriptação[12], denominado AES [7]. O algoritmo Rijndael é mostrado na Figura 11.

O algoritmo aceita chaves de tamanho 128, 192 ou 256 *bits*. O tamanho da chave define também o número de rodadas, Nr, da Figura 11, a qual varia de 10 para chave de 128 *bits* até 14 para chave de 256 *bits*. No entanto, diferente do DES, todas as operações são feitas utilizando *bytes*, possibilitando implementações eficientes, tanto em hardware quanto em software [7].

O algoritmo codifica blocos de 16 *bytes*, e durante cada rodada o estado atual da mensagem cifrada é armazenado em uma matriz de *bytes* chamado estado, cujo tamanho depende do tamanho da chave, sendo 4 x 4 *bytes* para chave de tamanho 128 bits.

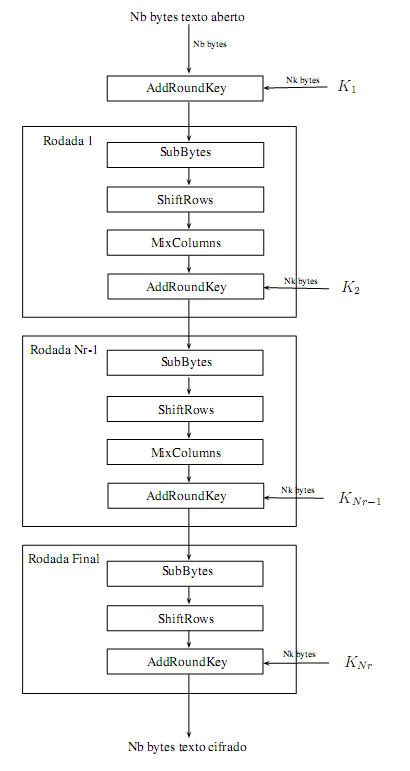


Figura 11: Fluxograma do algoritmo Rijndael.

Fonte: [13]

O processo de encriptação inicia expandindo a chave em Nr+1 vetores do mesmo tamanho que o estado. Um será utilizado no início do cálculo e os outros serão utilizados durantes as Nr rodadas (um por rodada). Em seguida, o texto é copiado para a matriz estado, preenchendo as colunas seqüencialmente. Além disso, antes de começar as rodadas, é feita uma operação OU EXCLUSIVO entre a matriz estado e a primeira chave expandida. Esse processo é chamado AddRoundKey e também utilizado ao final de cada rodada.

As rodadas seguintes são dividas em quatro etapas: SubBytes, ShiftRows, MixColumns e AddRoundKey. Na última rodada, porém, a operação MixColumns não é realizada. Na etapa SubBytes, mostrada na Figura 12, uma substituição não-linear irreversível opera em cada *byte* da matriz estado. Cada byte é usado como índice para um bloco de substituição, a fim de substituir o seu valor pelo conteúdo dessa entrada na cifra de substituição [7]. Basicamente, é uma cifra de substituição monoalfabética.

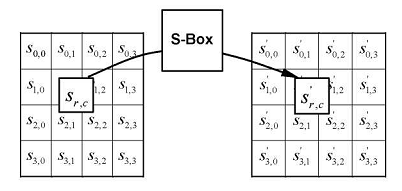


Figura 12: Operação SubBytes.

Fonte: [5]

Na etapa ShiftRows, cada linha da matriz estado é girada para esquerda, um número de vezes que representa a linha na matriz, ou seja, a linha 0 é girada 0 *byte* para a esquerda (não se altera), a linha 1 é girada 1 *byte* para esquerda e assim por diante, conforme a Figura 13.

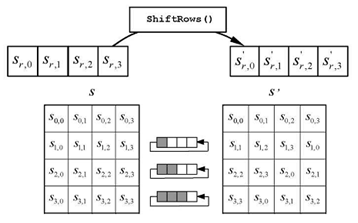


Figura 13: Operação ShiftRows.

Fonte: [5]

Na etapa MixColumns, mostrada na Figura 14, cada coluna da matriz estado é misturada independentemente. Essa mistura é realizada pela multiplicação da matriz estado com uma matriz constante. Embora isso possa parecer um *overhead* no processo de encriptação, existe um algoritmo que permite calcular cada elemento da nova coluna utilizando duas pesquisas em *lookup tables* e três operações XOR [7].

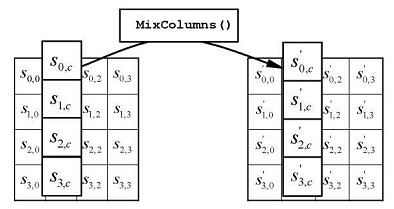


Figura 14: Operação MixColumns.

Fonte: [5]

Na última etapa, AddRoundKey, é efetuada a mesma operação que ocorreu ante do início das rodadas porém com a chave expandida correspondente da rodada, conforme pode ser visto na Figura 15.

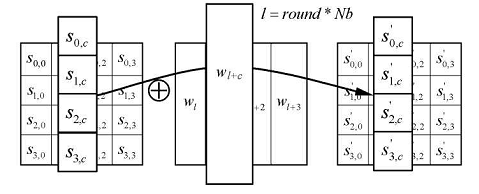


Figura 15: Operação AddRoundKey.

Fonte: [5]

O Rijndael foi desenvolvido para prover ao usuário tanto velocidade quanto segurança. Em termos de velocidade, um boa implementação em software em uma máquina de 2 GHz pode alcançar uma taxa de encriptação de 700 Mbps [7]. E, em termos de segurança, mesmo que seja desenvolvida uma máquina com um bilhão de processadores paralelos, cada um avaliando uma chave por picossegundo, levaria 1010 anos para pesquisar o espaço total de chaves [7].

## 3.5 MODELO DE UM SISTEMA CRIPTOGRÁFICO DE CHAVE ASSIMÉTRICA

O modelo de criptografia assimétrica foi proposta em 1976, por dois pesquisadores da Universidade de Harvard, Diffie e Hellman. A sua proposta baseava-se no fato de que por mais sólido que fosse um método de encriptação, a distribuição de chaves tornava o sistema fraco, pois caso um intruso roubasse a chave, o sistema se tornaria inútil [7].

Em sua proposta inicial, um sistema de criptografia assimétrico deveria atender aos seguintes requisitos:

1. É extremamente difícil deduzir *D* a partir de *E*.
2. *E* não pode ser decifrado por um ataque de texto simples escolhido.

O primeiro requisito diz que se aplicado o processo de decriptação *D* em um texto cifrado *E(P)* deve-se obter o texto original. O segundo é auto-explicativo. O terceiro é importante, pois o intruso pode testar o processo de encriptação indefinidamente utilizando a chave pública [14]. Um sistema criptográfico de chave assimétrica é mostrado na Figura 16.

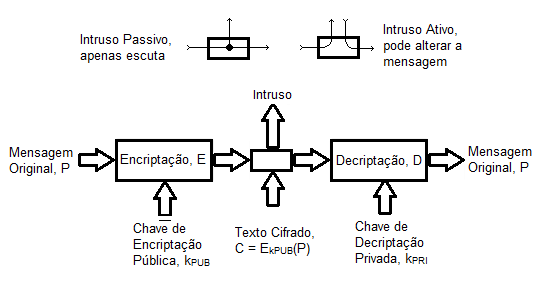


Figura 16: Sistema Criptográfico de Chave Assimétrica.

O sistema criptográfico de chave assimétrica é semelhante ao sistema criptográfico simétrico no sentido de que uma mensagem é criptografada por um processo de encriptação, passa por um meio onde pode ser recuperada e/ou alterada por um intruso e recebida e decriptada por outro usuário.

A diferença entre os dois sistemas reside no fato de que duas chaves são utilizadas no sistema, uma pública e uma privada. A chave pública, como o próprio nome diz, é de conhecimento público, qualquer usuário pode ter acesso à chave. A chave privada é possuída somente por um usuário. A chave pública e a privada são relacionadas entre si de tal forma que uma mensagem encriptada utilizando a chave pública só pode ser decriptada utilizando a chave privada.

## 3.6 MÉTODOS DE CRIPTOGRAFIA DE CHAVE ASSIMÉTRICA

Existem diversos algoritmos de criptografia assimétrica conhecidos, dos quais os mais importantes são [6]:

* RSA: considerado o mais seguro, foi proposto em 1978 por três pesquisadores do MIT. Baseia-se no problema da fatoração de números muito grandes.
* ElGamal: proposto em 1985 por Taher Elgamal. Baseia-se no problema do logaritmo discreto.
* Rabin: proposto em 1979 por Michael O. Rabin. Assim com o RSA baseia-se no problema da fatoração de números muito grandes.
* McEliece: proposto em 1978 por Robert McEliece. Baseia-se na dificuldade de decodificar códigos lineares.

### 3.6.1 RSA

O algoritmo mais utilizado na atualidade para encriptação de informações foi proposto em 1978, por três pesquisadores do MIT, Ron Rivest, Adi Shamir e Leonard Adleman. O nome RSA deriva da primeira letra do sobrenome dos seus criadores.

Desde que foi proposto, sobreviveu a todas as tentativas de quebra e é considerado um algoritmo bastante forte, porém sua principal desvantagem é exigir uma chave de pelo menos 1024 *bits*, em comparação aos 128 *bits* dos algoritmos simétricos.

O algoritmo funciona da seguinte forma [7]:

1. Escolha dois números primos muito grandes, *p* e *q* (geralmente, 1024 *bits*).
2. Calcule *n = p x q* e *z = (p -1) x (q -1)*
3. Escolha um número *d* tal que *z* e *d* sejam primos entre si.
4. Encontre e de forma que *(e x d) mod z = 1*

Com esses parâmetros calculados, o algoritmo inicia dividindo a mensagem original em blocos, de tal forma que bloco de mensagem *P* fique no intervalo 0≤ *P* < *n*. Isso pode ser feito agrupando-se cada bloco de mensagem em *k* bits, onde *k*  é o maior inteiro para 2*k* < n.

A partir disso, para encriptar a mensagem P, calcule e para decriptar, calcule . A chave pública é o número *e* e a chave privada é o número *d*. Uma outra vantagem do método, além da segurança, é que, por exemplo, *p* e *q* , então , logo cada bloco poderia encriptar até 1024 *bits*, ou 128 caracteres de 8 *bits*, em comparação aos 8 *bits* do DES e aos 16 *bits* do AES [7].

Em termos de segurança, considerando o melhor algoritmo conhecido de fatoração e um computador com tempo de instrução de 1s, o mesmo levaria 1025 anos para fatorar um número de 500 *bits*.

## 3.7 ENTROPIA

Os conceitos fundamentais da teoria da informação, criado por Claude Shannon em 1948, são as definições de como medir a incerteza de um experimento aleatório. Um quantificador matemático definido para medir a quantidade de informação em uma mensagem é chamado entropia, onde a informação em uma mensagem é a quantidade de “surpresa” na mensagem. Nenhuma “surpresa” significa nenhuma informação. [15,16]

A entropia é definida como a quantidade média de bits necessários para representar uma mensagem, usando a melhor codificação possível. Supondo uma mensagem *n*, tal que:

com probabilidade de ocorrência:

A entropia é definida por [17]:

Por exemplo, supondo que possuímos uma mensagem *N* que possa assumir dois valores {masculino, feminino}, com igual probabilidade de ocorrência. Logo, a entropia será:

Nesse caso, somente é necessário um bit para representar uma mensagem.

Uma mensagem aleatória possui alto valor de entropia, pois não é possível prever qual valor de cada bit da mensagem. Na criptografia, é necessário que a entropia de uma mensagem encriptada seja a maior possível, pois quanto maior a entropia, mais aleatória é mensagem e quanto mais aleatória é a mensagem, mais segura é a encriptação .

# 4 CRIPTOGRAFIA UTILIZANDO IMAGENS

O algoritmo de criptografia de dados utilizando imagens baseia-se nas seguintes premissas:

1. Uma imagem digital geralmente possui 256 valores possíveis que descrevem a intensidade de um pixel em cada coordenada da imagem, com valores variando entre 0 (preto total) e 255 (branco total).
2. Assim também é a tabela ASCII, como vista no anexo A, a qual é utilizada para representar todos os caracteres imprimíveis, possuindo valores que variam entre 0 e 255.

Vale lembrar que nem todos os valores da tabela ASCII são imprimíveis por uma impressora. Os valores entre 0 e 32 da tabela ASCII são valores de controle, utilizados como bits de sinalização, como o ASCII-10 (*Line-Feed*) utilizado para mover o cursor para linha seguinte sem mudar a coluna onde se encontra o cursor, o ASCII-11 (*Carriage Return*), utilizado para mover o cursor para a coluna zero enquanto continua na mesma linha, entre outros.

## 4.1. MÉTODO DE CRIPTOGRAFIA UTILIZANDO IMAGENS

Partindo dessa semelhança entre o nível de intensidade de cinza de um pixel em uma imagem e o conjunto de caracteres da tabela ASCII, o algoritmo de criptografia utilizando imagens propõe que o processo de encriptação de um vetor de caracteres consista no mapeamento do caractere em uma determinada posição aleatória na imagem, que possua o valor de intensidade de pixel igual ao valor do caractere na tabela ASCII.

### 4.1.1. Processo de Encriptação

O algoritmo é descrito abaixo:

Encriptação

1. Ler mensagem original *msg* de um arquivo
2. Ler imagem-chave *img*
3. Inicia mensagem cifrada *msgCript* nula
4. **Para** cada caractere *c* na *msg* **faça**
   1. Retorne uma lista *list* de par de coordenadas *(x,y)* onde *img(x,y)* = *c*
   2. Escolha aleatoriamente uma posição *k* na *list*
   3. Adicione ao fim de *msgCript* o par de coordenadas de *list(k)*
5. **Fim para**
6. Salve *msgCript* em um arquivo

Algoritmo 1: Encriptação.

No processo de encriptação, o Algoritmo 1 necessita da imagem que será a chave para decriptar. Chamaremos essa imagem de imagem-chave.

A partir do vetor de caracteres que serão criptografados, o algoritmo irá encriptar os caracteres um a um. Para cada caractere, o algoritmo irá selecionar um par de coordenadas aleatórias, que possua o valor de intensidade de pixel igual ao valor correspondente do caractere de entrada na tabela ASCII.

Após selecionar o par de coordenadas equivalente ao caractere, esse par é adicionado ao vetor de saída. Ao final, o vetor de saída, ou seja, o vetor de caracteres encriptados será um conjunto de pares de coordenadas.

Por exemplo, vamos supor que para um vetor de entrada que possua os caracteres “ab”, desejamos criptografar utilizando a Figura 17.



Figura 17: Imagem de teste.

Durante o processo de encriptação, o algoritmo verifica os valores de cada caractere, como podemos ver na tabela no Anexo A, os valores que representam os caracteres “a” e “b” são 97 e 98, respectivamente.

A partir desses valores, o algoritmo irá procurar na imagem todas as coordenadas que possuem valor de intensidade de pixel igual a 97 (para encriptar “a”) e 98 (para encriptar “b”), e escolher aleatoriamente um par de coordenadas para representar cada um dos caracteres.

Vamos supor que as coordenadas escolhidas estejam sendo mostradas na Figura 18 em vermelho.

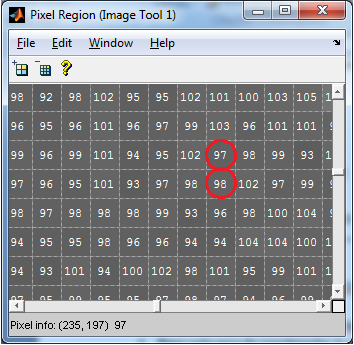


Figura 18: Coordenadas escolhidas aleatoriamente para encriptar a mensagem "ab", as coordenadas do círculo superior são mostradas no canto.

Logo, o vetor final encriptado vai ser o par de coordenadas do círculo superior e o par de coordenadas do círculo inferior. O processo é mostrado abaixo.

|  |  |  |
| --- | --- | --- |
| **Vetor de Entrada = [ a b ]** | | |
| Representação tipo char | a | b |
| Representação tipo inteiro | 97 | 98 |
| Representação coordenadas | [ 235 197 ] | [ 235 198 ] |
| **Vetor de Saída = [ 235 197 235 198 ]** | | |

### 4.1.2. Processo de Decriptação

O algoritmo é descrito abaixo:

Decriptação

1. Ler mensagem cifrada *msgCript*
2. Ler imagem-chave *img*
3. Inicia mensagem decriptada *msgDecript* nula
4. **Para** cada par de coordenadas *(x,y)* na *msgCript* **faça**
   1. Retorne o valor do pixel *p* na coordenada *img(x,y)*
   2. Adicione ao fim de *msgDecript* o valor *p*
5. **Fim para**
6. Salve *msgDecript* em um arquivo

Algoritmo 2: Decriptação.

Para o processo de decriptação, o Algoritmo 2 necessita da mesma imagem-chave utilizada para encriptar o arquivo original.

A partir do vetor de caracteres criptografados, ou seja, do vetor de coordenadas, o algoritmo irá decriptar cada par de coordenadas uma a uma. Para cada par de coordenadas, o algoritmo irá procurar na imagem-chave o valor correspondente do pixel para cada par de coordenadas.

Após retornar esse valor de pixel, ele será adicionado ao final do vetor de saída, ou seja, o vetor decriptado. Ao final, o vetor decriptado tem que ser igual ao vetor original antes do processo de encriptação.

Como exemplo, tentaremos decriptar a mensagem encriptada anteriormente utilizando a Figura 17.

O algoritmo irá carregar as coordenadas salvas no arquivo encriptado e, para cada par de coordenadas, irá procurar na imagem o valor correspondente do pixel.

|  |  |  |
| --- | --- | --- |
| **Vetor de Entrada = [ 235 197 235 198 ]** | | |
| Coordenadas encontradas | [ 235 197 ] | [ 235 198 ] |
| Valor de intensidade de pixel das coordenadas | 97 | 98 |
| Representação tipo char | a | b |
| **Vetor de Saída = [ a b ]** | | |

Ao final do processo, caso a imagem-chave utilizada para decriptar seja a mesma utilizada para encriptar, o vetor decriptado deve ser o mesmo que o vetor original.

## 4.2. VANTAGENS E DESVANTAGENS DO MÉTODO

### 4.2.1. Alta variabilidade do Arquivo Encriptado

Como descrito no algoritmo de encriptação, para cada caractere lido do vetor de entrada, é retornado uma lista de par de coordenadas na imagem que possuam valor de intensidade de pixel igual ao valor do caractere.

A vantagem disso é que, ao escolher aleatoriamente uma posição na lista, o vetor final encriptado será sempre diferente, o que garante que uma mesma mensagem original terá diferentes versões encriptadas.

Como exemplo, supondo que gostaríamos de encriptar uma mensagem de 10 caracteres utilizando a Figura 19.

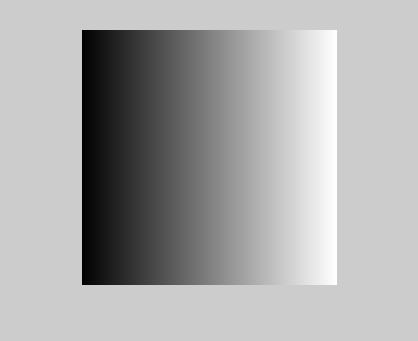


Figura 19: Imagem de dimensões 256x256 com distribuição uniforme de intensidade de pixels.

A Figura 19 possui dimensões 256x256 pixels com todos os valores de intensidade de pixel com o mesmo número de ocorrências, como podemos ver pelo seu histograma na Figura 20. Histograma é o gráfico das freqüências de ocorrência dos caracteres em uma imagem ou texto.

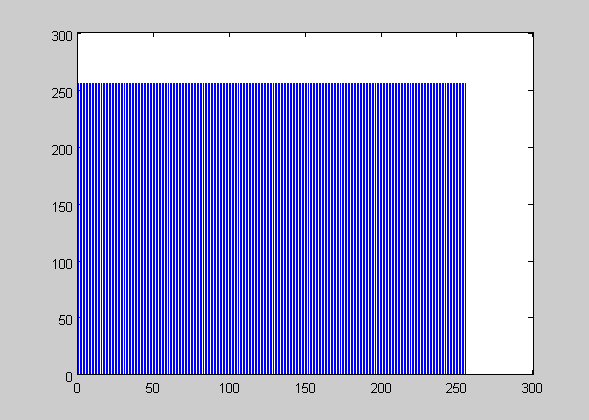


Figura 20: Histograma da Figura 19.

Para uma mensagem com 10 caracteres, o número de possíveis mensagens encriptadas é igual a:

Onde:

NE = Número de possíveis mensagens encriptadas.

k = Tamanho da mensagem.

*Oc*(x,i) = Número de ocorrências do valor *x* na imagem *i.*

Como todos os caracteres possuem o mesmo número de ocorrências na imagem:

E mesmo com todas essas possíveis mensagens encriptadas, a Figura 19 seria capaz de decriptá-las corretamente.

Notamos também que, qualquer imagem pode ser utilizada para decriptar a mensagem encriptada, porém a resposta só será igual se para cada par de coordenadas do vetor encriptado o valor de intensidade de pixel for igual ao da imagem original.

Suponhamos uma mensagem com 10 caracteres encriptada com a Figura 19. E, após a encriptação, desejamos decriptar utilizando a Figura 21, que nada mais é que a Figura 19 com cada linha deslocada pelo seu número (linha 1 possui todos os pixels deslocados de 1 pixel, linha 2 possui todos os pixels deslocados de 2 pixels, etc.).

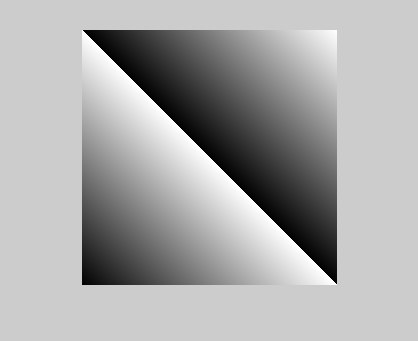


Figura 21: Imagem de dimensões 256x256 com distribuição uniforme de intensidade de pixels.

Como a Figura 21 é a Figura 19 deslocada, podemos afirmar que o histograma é o mesmo da Figura 20, porém somente uma linha é comum às duas imagens, a última linha.

Para que a mensagem encriptada anteriormente seja corretamente decriptada, é necessário que no processo de encriptação tenha selecionado, para cada caractere, as coordenadas da última linha da Figura 19, a probabilidade de isso acontecer é:

Na verdade, a probabilidade dessa mensagem ocorrer é igual à probabilidade de qualquer outra combinação de coordenadas. Apesar da probabilidade de decriptar um caractere é de:

Logo, as chances de decriptar a mensagem corretamente é igual à probabilidade de encriptação, por que no processo de decriptação não existe nenhuma aleatoriedade.

### 4.2.2. Tamanho do Arquivo Encriptado

Se por um lado, o fato de utilizar uma imagem é vantajoso, devido a robustez do algoritmo, por outro lado, notamos um aumento do arquivo encriptado em relação ao arquivo original, em média, o arquivo encriptado é oito vezes maior que original.

Isso acontece porque durante o processo de encriptação, a cada byte de informação lida (um caractere, um valor entre 0 e 255), serão escolhidos um par de coordenadas. Caso esse par de coordenadas seja inteiro, de tamanho 4 bytes, o tamanho final do arquivo seria 8 vezes maior que o original (4 bytes para a posição x e 4 bytes para a posição y).

Para minimizar esse fato podemos utilizar menos bytes para representar a informação, como podemos ver na Tabela 2, o valor inteiro (int) pode representar valores extremamente altos, valores que dificilmente serão alcançados pelas coordenadas de uma imagem.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **bits (bytes)** | **Intervalo de valores** | |
| **Inferior** | **Superior** |
| Int | 32 (4) | -2147483648 | 2147483647 |
| unsigned int | 32 (4) | 0 | 4294967295 |
| short int | 16 (2) | -32768 | 32767 |
| unsigned short int | 16 (2) | 0 | 65535 |
| n-bit int | n (n/8) |  |  |
| unsigned n-bit int | n (n/8) | 0 |  |

Tabela 2: Valores representados pelos diferentes tipos de inteiros.

Outro ponto que podemos notar que é que o tipo inteiro representa também valores negativos, que nunca serão representados por coordenadas de imagem, que possuem valores maiores que 0.

Logo, podemos modificar os algoritmos de encriptografia e decriptografia de forma que, a partir da imagem-chave, seja escolhido o menor número possível de bits para representar os valores de coordenadas. Por exemplo, caso a imagem possua dimensões até 511 pixels, o algoritmo deverá escolher um inteiro sem sinal de 9 bits, como podemos perceber na Tabela 3.

|  |  |  |
| --- | --- | --- |
| Tipo | Intervalo de valores | |
| Inferior | Superior |
| unsigned 09-bits int | 0 | 511 |
| unsigned 10-bits int | 0 | 1023 |
| unsigned 11-bits int | 0 | 2047 |
| unsigned 12-bits int | 0 | 4095 |
| unsigned 13-bits int | 0 | 8191 |
| unsigned 14-bits int | 0 | 16383 |
| unsigned 15-bits int | 0 | 32767 |

Tabela 3: Valores de inteiros sem sinal de tamanho entre 9 e 15 bits.

A vantagem dessa escolha inteligente do algoritmo são duas: o arquivo criptografado terá um tamanho reduzido pois um caractere de tamanho 8 bits, será criptografado em 18 bits (9 bits para cada coordenada).

A outra vantagem é que caso um atacante tente decriptar a mensagem, sem que saiba as dimensões da imagem-chave, terá dificuldade em determinar as diferentes coordenadas.

Inserindo essas modificações no Algoritmo 1 e Algoritmo 2, teríamos para a encriptação:

Encriptação com coordenadas utilizando n-bits

1. Ler mensagem original *msg* de um arquivo
2. Ler imagem-chave *img*
3. Calcule o número de bits *n* máximos necessários para representar as coordenadas de *img*
4. Inicia mensagem cifrada *msgCript* nula com inteiro sem sinal de precisão *n*
5. **Para** cada caractere *c* na *msg* **faça**
   1. Retorne uma lista *list* de par de coordenadas *(x,y)* onde *img(x,y)* = *c*
   2. Escolha aleatoriamente uma posição *k* na *list*
   3. Adicione ao fim de *msgCript* o par de coordenadas de *list(k)*
6. **Fim para**
7. Salve *msgCript* em um arquivo utilizando *n* bits de precisão

Algoritmo 3: Encriptação com coordenadas utilizando n-bits sem sinal.

E para decriptação, temos:

Decriptação com coordenadas utilizando n-bits

1. Ler imagem-chave *img*
2. Calcule o número de bits *n* máximos necessários para representar as coordenadas de *img*
3. Ler mensagem cifrada *msgCript* utilizando *n*-bits para cada coordenada
4. Inicia mensagem decriptada *msgDecript* nula
5. **Para** cada par de coordenadas *(x,y)* na *msgCript* **faça**
   1. Retorne o valor do pixel *p* na coordenada *img(x,y)*
   2. Adicione ao fim de *msgDecript* o valor *p*
6. **Fim para**
7. Salve *msgDecript* em um arquivo

Algoritmo 4: Decriptação com coordenadas utilizando n-bits sem sinal.

Notamos que não são necessárias grandes alterações no algoritmo, a mudança ocorrerá basicamente no momento de escrita do arquivo criptografado (Algoritmo 3) e na leitura do arquivo decriptado (Algoritmo 4).

Com essas alterações o tamanho do arquivo encriptado diminui drasticamente, como será mostrado na próxima seção.

### 4.2.3. Tamanho da Imagem-chave

Em comparação aos algoritmos atuais, uma chave que tenha tamanho de uma imagem, mesmo que considerada pequena, geralmente terá tamanho em *Kilo-Bytes*.

Como exemplo, podemos citar a Figura 19, que possui dimensões 256x256, em escala de cinza, salva em jpeg utiliza 8KB.

Porém, ao tentarmos diminuir o tamanho da imagem, diminuímos o número de coordenadas possíveis para cada caractere, diminuindo assim a aleatoriedade do arquivo criptografado. Apesar disso, ao diminuirmos a imagem-chave, dependendo do número de coordenadas que representem o valor desejado e da aleatoriedade do algoritmo, podemos ainda ter um grande número de possíveis vetores encriptados.



Figura 22: Imagem de dimensões 40x64 com distribuição uniforme de intensidade de pixels.

Suponhamos que desejamos encriptar uma mensagem aleatória com 10 caracteres utilizando a Figura 22.

A Figura 22 possui uma distribuição uniforme de intensidade de pixel igual a 10, ou seja, cada valor de intensidade de pixel ocorre 10 vezes na imagem, logo, se a mensagem é aleatória temos:

Ou seja, mesmo uma imagem com tamanho reduzido ainda pode gerar um grande número de arquivos encriptados diferentes.

Quanto ao tamanho da image-chave, a Figura 22, codificada em jpeg, o tamanho é de 809 bytes, que comparada ao tamanho de uma chave do processo de criptografia de Chave Pública, utiliza uma chave de 1024 bits. O tamanho da Figura 22 é 6,32 vezes maior que uma chave utilizada no algoritmo RSA e 50,56 vezes maior que uma chave utilizada no algoritmo AES.

### 4.2.4. Falta de Coordenadas para Representar Valores

Caso o Algoritmo 3 encontre um valor na mensagem que não possua um par de coordenadas que represente um pixel de mesmo valor de intensidade na imagem, ele irá parar de funcionar. Isso pode acontecer em diversas situações, mesmo com imagens como a Figura 23.



Figura 23: Imagem de teste *bridge*.

O histograma da Figura 23 é mostrado na Figura 24.

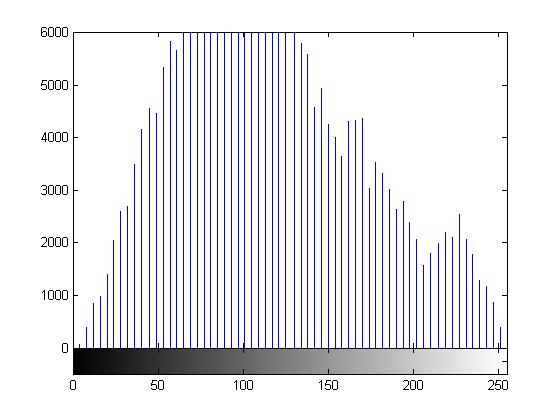


Figura 24: Histograma da Figura 23.

Como pode ser visto, apesar de haver valores de pixels em toda a faixa de valores, existem espaços vazios entre cada feixe, que não possuem representações de valores na imagem.

A solução proposta para esse problema é a inserção de *flags* e informações de distância para o pixel de intensidade mais próxima do desejado existente. A inserção dessas *flags* e informações irão funcionar da seguinte forma:

* Ao iniciar o algoritmo, será verificado se existe pelo menos um par de coordenadas para representar o caractere, caso haja o algoritmo inicia com valor 0, caso não haja o algoritmo irá iniciar com valor 1.
* Caso um par de coordenada tenha sido encontrado, será iniciado o algoritmo com valor 0 e será feito uma nova verificação para o próximo pixel, caso haja um par correspondente, somente adiciona ao texto cifrado. Isso deve ocorrer até encontrar um pixel que não pode ser representado por um par de coordenadas. Quando isso acontecer deve inserir uma *flag* de parada, que é definida como a maior dimensão da imagem + 1.
* Ao inserir uma *flag* de parada, será inserida uma *flag* 1 para sinalizar que não foi encontrada um par de coordenadas para o valor do caractere. Em seguida, o algoritmo deve procurar o próximo valor mais próximo que pode ser representado, dentro do intervalo [0,255]. Após encontrar o par de coordenada que representa a intensidade de pixel mais próxima, deve ser inserido um sinal para representar se ele é menor ou maior que o valor de intensidade encontrado e quantas unidades ele é maior ou menor. Por fim, novamente a *flag* de parada, o valor 0 para representar a coordenada encontrada e o par de coordenadas.

Como exemplo, vamos supor que desejamos encriptar a seqüência [ a d h ], que são equivalente à [ 97 100 104] em valor numérico. Vamos supor também que não exista nenhum pixel na imagem-chave que possua valor de intensidade igual no intervalo [103,107]. Supondo pares de coordenadas quaisquer, o algoritmo irá encriptar a mensagem da seguinte forma (as *flags* utilizadas estão em vermelho):

|  |  |  |  |
| --- | --- | --- | --- |
| **Vetor de Entrada = [ a d h ]** | | | |
| Representação tipo char | a | d | h |
| Representação tipo inteiro | 97 | 100 | 104 |
| Representação coordenadas | [243 412] | [283 398] | ? |
| **Vetor de Saída = [0 243 412 283 398 513 1 45 2 513 0 274 117]** | | | |

Como pode ser visto, o algoritmo inseriu a *flag* 0 e começou a preencher a mensagem cifrada até encontrar o valor que não possuía representação. Nesse momento foi inserida a *flag* de parada (considerando uma imagem de dimensões 512x512, a *flag* será 513), o valor 1 para representar que não foi encontrado o valor desejado.

Foi selecionado valor 45 para representar que o pixel mais próximo encontrado é menor que o atual e 43 para representar que o pixel mais próximo é maior que o atual (esses valores foram escolhidos por representar o sinal aritmético “-” e “+” na tabela ASCII, respectivamente). Inserido o valor 45 para representar que o pixel é menor, inseri-se o quanto ele é menor (2 unidades no exemplo, levando o mais próximo pixel encontrado foi o 102).

Por fim, a *flag* de parada, o valor 0 e as coordenadas do pixel mais próximo encontrado. Caso o pixel sem representação fosse o primeiro da sequência, o vetor de saída ficaria da seguinte forma (com as *flags* em vermelho):

**Vetor de Saída = [1 45 2 513 0 274 117 243 412 283 398]**

O novo processo de encriptação é mostrado no Algoritmo 5.

Encriptação com procura de pixels próximos

1. Ler mensagem original *msg* de um arquivo
2. Ler imagem-chave *img*
3. Calcule o número de bits *n* máximos necessários para representar as coordenadas de *img*
4. Calcule *flag* de parada *escape* como a maior dimensão da imagem + 1
5. Inicia mensagem cifrada *msgCript* nula com inteiro sem sinal de precisão *n*
6. **Para** cada caractere *c* na *msg* **faça**
   1. Retorne uma lista *list* de par de coordenadas *(x,y)* onde *img(x,y)* = *c*
   2. **Se** existir o par coordenadas
      1. Escolha aleatoriamente uma posição *k* na *list*
      2. *encryptedChar* = par de coordenadas de *list(k)*
   3. **Senão**
      1. Retorne o vetor contendo *temp* = [1 (43/45) *dist escape* 0], onde *dist* é a distância para o pixel mais próximo representável.
      2. Retorne uma lista *list* de par de coordenadas *(x,y)* onde *img(x,y)* = *c (+/-) dist*
      3. Escolha aleatoriamente uma posição *k* na *list*
      4. *encryptedChar* = *temp* concatenado com *list(k)*
   4. **FimSe**
   5. **Se** o tamanho de *encryptedChar* == 2 e *msgCript* está vazia
      1. *msgCript* = 0 concatenado com ele mesmo.
   6. **FimSe**
   7. **Se** o tamanho de *encryptedChar* > 2 e *msgCript* não está vazia
      1. *msgCript* = *escape* concatenado com *encryptedChar*
   8. **FimSe**
   9. Adicione ao fim de *msgCript* a variável *encryptedChar*
7. **Fim para**
8. Salve *msgCript* em um arquivo utilizando *n* bits de precisão

Algoritmo 5: Encriptação com procura de pixels próximos.

O processo de decriptação deve ser atualizado também para entender a nova forma de encriptação, como mostrado no Algoritmo 6:

Decriptação com procura de pixels próximos

1. Ler imagem-chave *img*
2. Calcule o número de bits *n* máximos necessários para representar as coordenadas de *img*
3. Ler mensagem cifrada *msgCript* utilizando *n*-bits para cada coordenada
4. Calcule *flag* de parada *escape* como a maior dimensão da imagem + 1
5. Inicia mensagem decriptada *msgDecript* nula
6. Inicia *hasOp* como *false*
7. Inicia i = 1
8. **Enquanto** i < tamanho da mensagem encriptada **faça** 
   1. **Se** msgCript(i) == 0
      1. i = i + 1
      2. **Enquanto** *msgCript*(i) *escape* **faça**
         1. *x* = *msgCript*(i)
         2. *y* = *msgCript*(i+1)
         3. *decryptedChar* = valor do pixel *p* na coordenada *img(x,y)*
         4. **Se** *hasOp =*= *true*
            1. *decryptedChar = decryptedChar* + *corr*
            2. *hasOp* = *false*
         5. **FimSe**
         6. *msgDecript* = *decryptedChar* concatenado com *msgDecript*
         7. i = i + 2
         8. **Se** i > tamanho de *msgCript*
            1. *Break*
         9. **FimSe**
   2. **SenãoSe** msgCript(i) == 1
      1. i = i + 1
      2. *hasOp* = *true*
      3. **Se** *msgCript*(i) == 43
         1. *corr* = msgCript(i+1)\*(-1)
      4. **SenãoSe** *msgCript*(i) == 45
         1. *corr* = msgCript(i+1)
      5. **FimSe**
      6. i = i + 2
   3. **FimSe**
   4. i = i + 1
9. **FimEnquanto**
10. Salve *msgDecript* em um arquivo

Algoritmo 6: Decriptação com procura de pixels próximos.

Outra vantagem de inserir *flags* no algoritmo é que, aliado ao fato de o número de bits utilizados para salvar cada coordenada varia, dá ao intruso infinitas combinações de possíveis tamanhos de imagens-chave, dificultando a quebra da mensagem cifrada.

Além disso, o sistema de *flags* utilizados foi pensado de forma a facilitar um processo de compressão chamado compressão aritmética, que será mais bem comentado nos trabalhos futuros.

## 4.3. OTIMIZAÇÃO NO ALGORITMO

A proposta de melhoria no algoritmo de criptografia utilizando imagens procura melhorar o tempo que o algoritmo leva para encriptar uma mensagem em casos onde a imagem possui poucos valores representáveis. Para tal, o algoritmo foi modificado de forma a não procurar o valor de intensidade de *pixel* mais próximo representável, mas representar o pixel faltante com qualquer valor de pixel representável.

A implementação da otimização foi feita somente no algoritmo de encriptação, e ocorre da seguinte forma:

* No início do algoritmo, quando é gerada a lista de coordenadas, é gerada também uma lista contendo todos os *pixels* que possuem valores de coordenadas na imagem.
* Essa lista é acessada somente quando o algoritmo precisa representar um valor que não possui valor de pixel correspondente. Nesse momento, é escolhido um valor aleatório nessa lista, e a partir desse valor o processo de encriptação continua normalmente.

Essa abordagem traz certas vantagens sobre o algoritmo original, que procura o valor de pixel mais próximo representável:

1. O algoritmo se torna mais rápido, pois não há necessidade de ficar procurando o valor mais próximo superior ou inferior. Com a otimização, o algoritmo precisa somente escolher aleatoriamente um valor de uma lista e utilizá-lo para os cálculos.
2. No algoritmo original, era utilizado sempre o valor mais próximo representável, ou seja, a aleatoriedade diminuía. Por exemplo, supondo que uma imagem possui valores de intensidade de pixel entre 0 e 200, todos os valores acima de 200 iriam encontrar o valor 200 como mais próximo representável, o que iria diminuir a aleatoriedade do algoritmo, pois suas coordenadas iriam se repetir com mais freqüência. Com a otimização, qualquer coordenada seria utilizada para representar o valor faltante, aumentando a segurança.
3. E, por conseqüência do aumento da aleatoriedade, há um aumento da entropia do arquivo encriptado.

O novo processo de encriptação é mostrado no Algoritmo 7.

Encriptação com procura de pixels próximos

1. Ler mensagem original *msg* de um arquivo
2. Ler imagem-chave *img*
3. Calcule o número de bits *n* máximos necessários para representar as coordenadas de *img*
4. Calcule *flag* de parada *escape* como a maior dimensão da imagem + 1
5. Inicia mensagem cifrada *msgCript* nula com inteiro sem sinal de precisão *n*
6. Cria *listNonNull* de valores representáveis da imagem.
7. **Para** cada caractere *c* na *msg* **faça**
   1. Retorne uma lista *list* de par de coordenadas *(x,y)* onde *img(x,y)* = *c*
   2. **Se** existir o par coordenadas
      1. Escolha aleatoriamente uma posição *k* na *list*
      2. *encryptedChar* = par de coordenadas de *list(k)*
   3. **Senão**
      1. Escolha uma posição aleatória na lista *listNonNull*.
      2. Retorne o vetor contendo *temp* = [1 (43/45) *dist escape* 0], onde *dist* é a distância para o pixel aleatório escolhido no passo anterior.
      3. Retorne uma lista *list* de par de coordenadas *(x,y)* onde *img(x,y)* = *c (+/-) dist*
      4. Escolha aleatoriamente uma posição *k* na *list*
      5. *encryptedChar* = *temp* concatenado com *list(k)*
   4. **FimSe**
   5. **Se** o tamanho de *encryptedChar* == 2 e *msgCript* está vazia
      1. *msgCript* = 0 concatenado com ele mesmo.
   6. **FimSe**
   7. **Se** o tamanho de *encryptedChar* > 2 e *msgCript* não está vazia
      1. *msgCript* = *escape* concatenado com *encryptedChar*
   8. **FimSe**
   9. Adicione ao fim de *msgCript* a variável *encryptedChar*
8. **Fim para**
9. Salve *msgCript* em um arquivo utilizando *n* bits de precisão

Algoritmo 7: Encriptação otimizada com lista de valores representáveis.

# 5 EXPERIMENTOS E RESULTADOS

Todos os algoritmos apresentados nesse capítulo foram implementados utilizando o programa MATLAB®. O principal motivo da utilização da ferramenta MATLAB® deve-se ao fato de, apesar de ser mais lento que implementações em linguagens como C ou C++, facilita o trabalho com imagens dos mais diversos formatos, pois o mesmo faz o *parser* dos cabeçalhos automaticamente para o usuário.

O MATLAB® (*Matrix Laboratory*) é uma linguagem de alto nível e ambiente de desenvolvimento interativo de alto desempenho voltado para o cálculo numérico. Ele foi desenvolvido para trabalhar com matriz, o que facilita o desenvolvimento de aplicações que trabalhem com imagens.

Os experimentos a seguir foram divididos em 3 grupos. No primeiro grupo de experimentos, foram geradas 50 mensagens randômicas de tamanho variável entre 9000 e 10000 caracteres para serem encriptadas, com valores entre 0 e 255. No segundo grupo de experimentos, foi utilizado um texto real, o texto Gênesis da Bíblia Cristã. O texto possui 187071 caracteres, entre o ASCII-10 e ASCII-252. Serão comparados valores de tempo de encriptação e entropia, entre o algoritmo original e o algoritmo otimizado. Os valores de tamanho de mensagem encriptada não serão comparados, pois o algoritmo otimizado não muda a estrutura do arquivo encriptado, mas a forma de encriptá-los. O terceiro grupo de experimentos foi comparado os resultados do algoritmo original, do algoritmo otimizado, AES e RSA na encriptação do Gênesis da Bíblia Cristã.

No primeiro e segundo grupo de experimentos foram utilizadas 10 imagens de um banco de imagem público (anexo B). Para os testes foi utilizado um computador com Intel® Core 2 Duo 2.20 GHz e 4GB de memória RAM, Windows 7 Home Premium 32 bits. As especificações das imagens podem ser vistas na Tabela 4.

|  |  |  |
| --- | --- | --- |
| Imagem-chave (formato) | Dimensões (*pixels*) | Tamanho (*bytes*) |
| Primeira (aerial.pgm) | 512x512 | 257K |
| Segunda (boats.pgm) | 576x720 | 406K |
| Terceira (bridge.pgm) | 512x512 | 257K |
| Quarta (D108.pgm) | 640x640 | 401K |
| Quinta (f16.pgm) | 512x512 | 257K |
| Sexta (girl.pgm) | 576x720 | 406K |
| Sétima (Lena.jpg) | 512x512 | 43K |
| Oitava (peppers.pgm) | 512x512 | 257K |
| Nona (pp1209.pgm) | 512x512 | 257K |
| Décima (zelda.pgm) | 576x720 | 406K |

Tabela 4: Especificações das imagens utilizadas para teste.

## 5.1 GRUPO DE EXPERIMENTO 1

Para cada imagem será mostrado um gráfico comparativo entre os resultados obtidos a partir do algoritmo original e o algoritmo otimizado e uma breve análise dos resultados.

### 5.1.1 Imagem de Teste 1: aerial.pgm

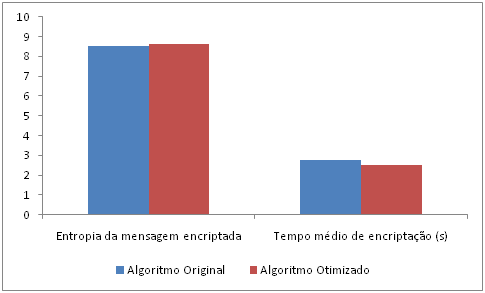


Figura 25: Resultados obtidos a partir da Figura 40 (Anexo A).

Como pode ser visto na Figura 41 (Anexo A), a primeira imagem de teste possui representação de quase todos os valores na faixa de 0 a 255, logo os ganhos no algoritmo são pequenos.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 9% de diminuição no tempo de encriptação e 2% de aumento da entropia.

### 5.1.2 Imagem de Teste 2:boats.pgm

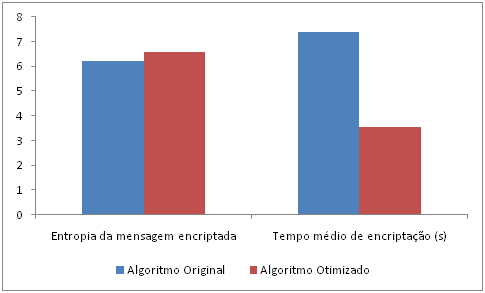


Figura 26: Resultados obtidos a partir da Figura 42 (Anexo A).

Como pode ser visto na Figura 43 (Anexo A), a segunda imagem de teste possui valores somente até o valor 220, logo para todos os caracteres acima desse valor iriam perder muito tempo procurando quais os valores mais próximos representáveis, e todas as vezes o resultado seria o mesmo, o valor mais próximo seria o 220. Isso iria diminuir a entropia, por causa da repetição das coordenadas do valor 220 e, o tempo de encriptação, por causa da procura excessiva.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 52% de diminuição no tempo de encriptação e 5,5% de aumento da entropia.

### 5.1.3 Imagem de Teste 3:bridge.pgm

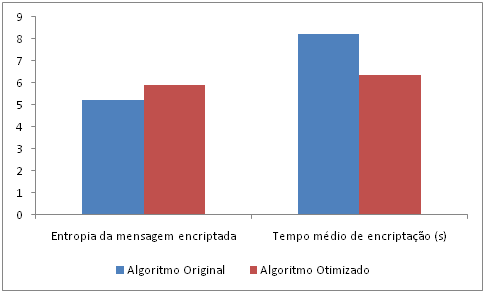


Figura 27: Resultados obtidos a partir da Figura 44.

Como pode ser visto na Figura 45 (Anexo A), a terceira imagem de teste possui valores em toda a faixa entre 0 e 255, porém é composta de várias lacunas igualmente espaçadas, ou seja, diversos valores não possuem representação na imagem, por essa razão o algoritmo executa tão lentamente, pois ao encontrar um valor que não possui representação, o valor mais próximo representável está próximo. Logo, é de se esperar que haja ganhos com o algoritmo otimizado porém não tão altos quanto da segunda imagem mas, maiores que os da primeira imagem. Novamente, a entropia irá aumentar pelo fato de que os valores mais próximos irão se repetir com menos freqüência.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 23% de diminuição no tempo de encriptação e 10% de aumento da entropia.

### 5.1.4 Imagem de Teste 4:D108.pgm

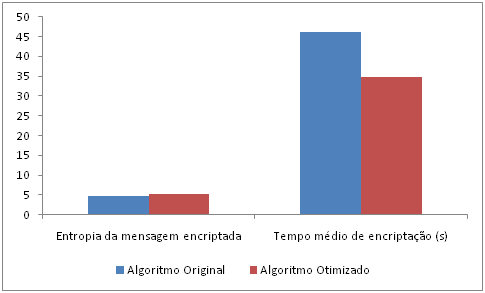


Figura 28: Resultados obtidos a partir da Figura 46.

Como pode ser visto na Figura 47 (Anexo A), a quarta imagem de teste, assim como a terceira imagem de teste, possui valores em toda a faixa entre 0 e 255, porém as lacunas não estão igualmente espaçadas, o que causa o algoritmo demore mais tempo para encontrar o pixel mais próximo representável. Para o algoritmo otimizado, os resultados são semelhantes aos resultados da imagem anterior.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 25% de diminuição no tempo de encriptação e 12% de aumento da entropia.

### 5.1.5 Imagem de Teste 5: f16.pgm

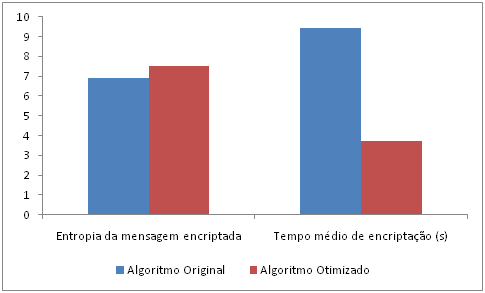


Figura 29: Resultados obtidos a partir da Figura 48.

Como pode ser visto na Figura 49 (Anexo A), a quinta imagem de teste possui valores de intensidade de pixel apenas na faixa entre 31 e 229 e, assim como a segunda imagem de teste, caso o algoritmo tente encriptar um caractere com valor fora dessa faixa, ele irá perder muito tempo procurando pelo pixel mais próximo representável. Para essa imagem, os resultados são semelhantes aos resultados da segunda imagem.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 60% de diminuição no tempo de encriptação e 8% de aumento da entropia.

### 5.1.6 Imagem de Teste 6: girl.pgm

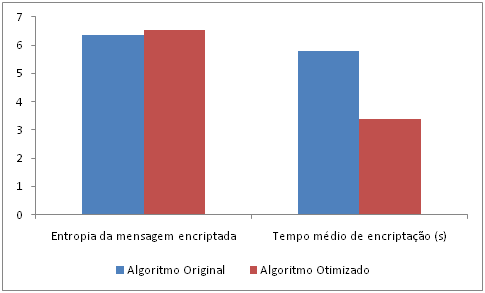


Figura 30: Resultados obtidos a partir da Figura 50.

Como pode ser visto na Figura 51 (Anexo A), a sexta imagem de teste possui valores de intensidade de pixel na faixa entre 1 e 218, logo para valores acima de 218, o algoritmo irá perder muito tempo procurando o valor mais próximo representável, além disso, esse valor será sempre 218, aumentando a repetição das coordenadas do valor 218 e diminuindo a entropia. Para essa imagem, os resultados são semelhantes aos resultados da segunda e quinta imagens.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 41% de diminuição no tempo de encriptação e 2,6% de aumento da entropia.

### 5.1.7 Imagem de Teste 7: lena.jpg

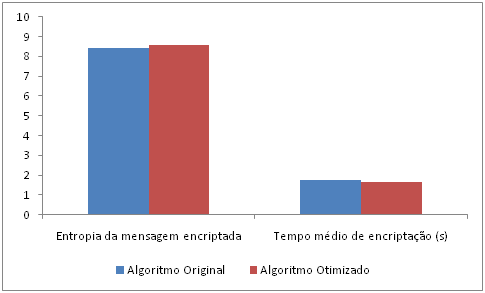


Figura 31: Resultados obtidos a partir da Figura 52.

Como pode ser visto na Figura 53 (Anexo A), a imagem sétima imagem de teste, assim como a primeira imagem de teste, possui valores em quase toda a faixa entre os valores 0 e 255, logo os ganhos no tempo de encriptação e de aumento da entropia são baixos. Para essa imagem, os resultados são semelhantes aos resultados da primeira imagem.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 2% de diminuição no tempo de encriptação e 1% de aumento da entropia.

### 5.1.8 Imagem de Teste 8: peppers.pgm

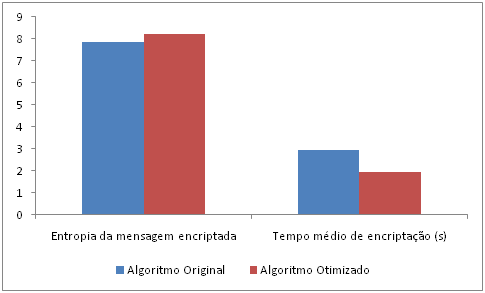


Figura 32: Resultados obtidos a partir da Figura 54.

Como pode ser visto na Figura 55 (Anexo A), a oitava imagem de teste, assim como a imagem anterior, possui valores em quase toda a faixa entre 0 e 255. Apesar dos resultados mostrarem uma diminuição significativa no tempo de encriptação (33%), o tempo médio de diminuição é pequeno, cerca de 1s.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 33% de diminuição no tempo de encriptação e 5% de aumento da entropia.

### 5.1.9 Imagem de Teste 9: pp1209.pgm

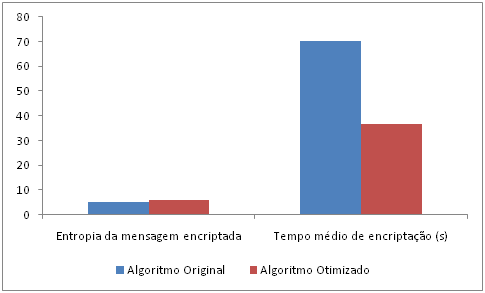


Figura 33: Resultados obtidos a partir da Figura 56.

Como pode ser visto na Figura 57 (Anexo A), a nona imagem de teste possui um histograma com diversas lacunas, o que torna o algoritmo bastante lento, perdendo bastante tempo procurando o *pixel* mais próximo representável. Na verdade, devido ao fato de que, na faixa entre 0 e 255, apenas 64 valores estão presentes, o algoritmo passa mais tempo procurando outros *pixels* para representar um *pixel* faltante, e o algoritmo otimizado mostra bom resultados pois melhora essa busca.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 48% de diminuição no tempo de encriptação e 10% de aumento da entropia.

### 5.1.10 Imagem de Teste 10: zelda.pgm

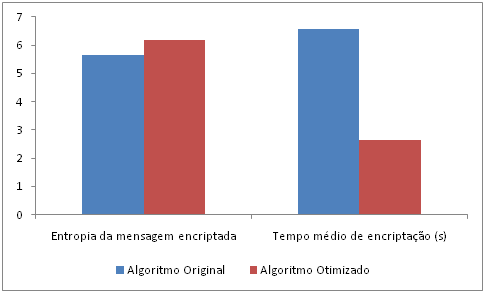


Figura 34: Resultados obtidos a partir da Figura 58.

Como pode ser visto na Figura 59 (Anexo A), a décima imagem teste possui um histograma parecido com o da quinta imagem de teste, porém a décima imagem de teste possui valores entre 8 e 204. Os resultados são semelhantes aos resultados da quinta imagem de teste.

Para esse teste, o algoritmo otimizado apresentou, em média, cerca de 60% de diminuição no tempo de encriptação e 8% de aumento da entropia.

## 5.2 GRUPO DE EXPERIMENTO 2

No grupo de experimento 2, foram utilizadas as mesmas imagens-chave para encriptar uma mensagem. A diferença é que a mensagem a ser encriptada é um texto em português, e não um texto randômico. Essa diferença aparece no histograma da mensagem original, pois ao invés de possuir um histograma que tende à uniformidade, como o randômico, as freqüências de certas letras na língua portuguesa tendem a ser maior que outras. Além disso, em um texto real, os caracteres não imprimíveis não aparecem no texto, com exceção do ASCII-10 e ASCII-11 (*line feed* e *carriage return*, respectivamente). O texto utilizado foi o capítulo Gênesis da Bíblia Cristã e seu histograma pode ser visto na Figura 35.

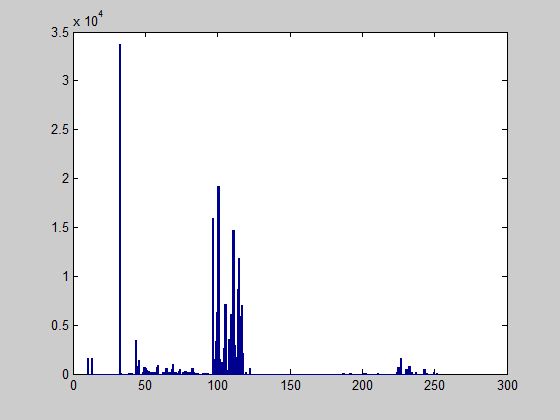


Figura 35: Histograma do capítulo Gênesis da Bíblia Cristã.

Como podemos notar na Figura 35, o caractere que mais se repete é o ASCII-32, o espaço entre palavras. Em seguida, notamos uma concentração na área entre 97 e 122, que são as letras de “a” a “z”, minúsculas. Outros caracteres também aparecem, mas com menor freqüência do que os citados. A quantidade de caracteres contidos no Gênesis é de 187071. Os resultados da encriptação do Gênesis, utilizando o algoritmo original e o algoritmo otimizado, com cada imagem do banco de teste, podem ser visto na Tabela 5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tempo de encriptação  (original)  (s) | Tempo de encriptação  (otimizado)  (s) | Entropia  (original) | Entropia  (otimizado) |
| aerial | 78.80 | 78.52 | 8.82 | 8.83 |
| boats | 112.06 | 99.34 | 6.40 | 6.45 |
| bridge | 317.86 | 289.12 | 5.98 | 6.55 |
| D108 | 763.47 | 686.54 | 5.42 | 5.89 |
| f16 | 199.06 | 176.59 | 6.70 | 7.48 |
| girl | 100.93 | 94.94 | 7.38 | 7.40 |
| lena | 62.89 | 62.12 | 8.94 | 8.94 |
| peppers | 68.94 | 67.68 | 8.85 | 8.88 |
| pp1209 | 1418.35 | 964.61 | 5.25 | 5.85 |
| zelda | 90.34 | 78.54 | 6.89 | 6.96 |

Tabela 5: Resultados obtidos na encriptação do Gênesis.

O comparativo entre o tempo de encriptação utilizando o algoritmo otimizado e o algoritmo original podem ser visto na .

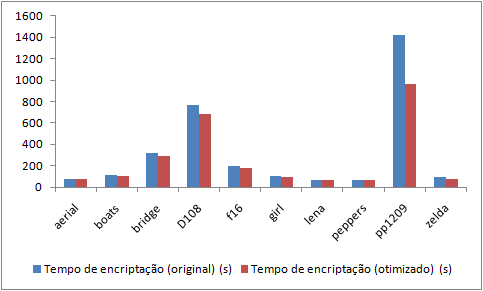


Figura 36: Comparativo entre o tempo de encriptação utilizando o algoritmo original e o algoritmo otimizado.

Assim como nos teste anteriores, houve uma diminuição no tempo de encriptação para todas as imagens. Algumas imagens, que possuem valores em quase toda a faixa, tiveram poucos ganhos, porém em imagens que possuem muitos valores faltantes, que fazem o algoritmo original perder muito tempo procurando o *pixel* mais próximo representável, o algoritmo otimizado mostra-se mais rápido, pois substitui o *pixel* faltante por qualquer *pixel* válido.

A diminuição no tempo de encriptação variou entre 0,35% para a primeira imagem de teste e 32% para a nona imagem de teste.

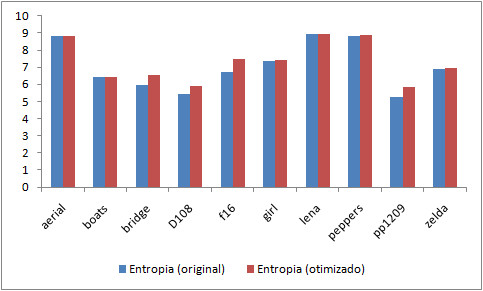


Figura 37: Comparativo entre a entropia do arquivo encriptado utilizando o algoritmo original e o algoritmo otimizado.

O teste comparativo mostrou que, utilizando o algoritmo otimizado, com exceção da sétima imagem de teste que não teve ganho, todas as outras imagens mostraram um aumento da entropia do texto encriptado. Isso se deve ao fato de que, pelo fato de que o algoritmo otimizado escolhe aleatoriamente qualquer *pixel* para representar o *pixel* faltante, há uma diminuição do número de coordenadas repetidas. A sétima imagem não obteve ganho, pois possui valores em quase toda faixa entre 0 e 255, logo a substituição ocorre poucas vezes em relação ao número de caracteres do Gênesis.

Os ganhos de entropia variam entre 0% para a sétima imagem de teste e 10.42% para a quinta imagem de teste.

## 5.3 GRUPO DE EXPERIMENTO 3

No grupo de experimentos 3 foram feitos testes comparativos entre o algoritmo original, o algoritmo otimizado e os algoritmos AES e RSA, de criptografia de chave simétrica e criptografia de chave assimétrica, respectivamente. Os algoritmos utilizados para encriptação AES e RSA estão disponíveis online, em [18] e [19], respectivamente.

Para os experimentos, a mensagem utilizada foi o Gênesis da Bíblia Cristã, assim como no grupo de experimentos 2. Para a comparação de resultados, foi escolhido o melhor resultado do método proposto, encriptação com a sétima imagem-chave (Figura 52).

Os seguintes resultados foram obtidos:

|  |  |  |
| --- | --- | --- |
| Algoritmo | Tempo de encriptação (s) | Entropia |
| AES | 4.58 | 7.998 |
| RSA | 6.64 | 7.989 |
| Algoritmo Original | 62.89 | 8.94 |
| Algoritmo Otimizado | 62.12 | 8.94 |

Tabela 6: Comparativo entre o método proposto, AES e RSA.

Os resultados obtidos utilizando o algoritmo otimizado está próximo do original, pois para esse teste foi escolhido o melhor resultado para a comparação com os algoritmos atuais, RSA e AES. Porém, certas considerações devem ser feitas quanto aos resultados acima.

A grande vantagem dos métodos atuais sobre o proposto sem dúvida é o tempo de encriptação. Os algoritmos são rápidos, pois são constituídos basicamente de cifras de substituição e transposição não-lineares, e não precisam tomar decisões aleatórias sobre tabelas nem procurar por *pixels* próximos representáveis como o algoritmo proposto.

A entropia do arquivo encriptado utilizando AES e RSA é menor, porém o valor ideal de entropia, para ambos os casos, é 8 (oito). O valor está próximo do ideal e, pelo fato de serem constituídos de cifras de substituição e transposição, ou seja, não substituem um caractere por um par de coordenadas, como faz o algoritmo original (e proposto).

Porém, o método proposto possui uma enorme vantagem sobre os algoritmos comparados. Independente do número de vezes que os algoritmos AES e RSA sejam executados tendo como entrada o texto de teste, o resultado, a mensagem encriptada será sempre a mesma, enquanto utilizando o método proposto, o resultado será diferente para cada encriptação, não afetando o processo de decriptação que irá sempre decriptar a mesma mensagem.

# 6 DISCUSSÃO DOS RESULTADOS E CONCLUSÕES

O objetivo deste trabalho foi desenvolver uma otimização do método de encriptação utilizando imagens [20] e avaliar o seu desempenho frente ao algoritmo original. Para alcançar esse objetivo foi proposta uma melhoria no processo de encriptação, no momento em que é necessário representar um *pixel* que não possui valor representável na imagem. A otimização muda o algoritmo para que, ao invés de sempre procurar o *pixel* com valor mais próximo do valor faltante, o algoritmo otimizado procure qualquer pixel pra substituir o faltante.

Para os testes foram utilizadas 10 imagens, cada uma com características diferentes. Algumas possuem valores de níveis de cinza distribuídos em quase toda faixa entre 0 e 255 (imagens com alto contraste) e outras que mostravam muitas lacunas no histograma (vide anexo B).

Foram feitos testes utilizando tanto mensagens aleatórias geradas pelo programa de testes, que gerava mensagens com valores entre 0 e 255 e mensagens de texto reais, como o Gênesis da Bíblia Cristã. Esses testes foram necessários para validar o método frente aos algoritmos atuais.

Em comparação ao algoritmo original, o algoritmo otimizado mostrou-se eficiente nos casos, em que as imagens-chave mostravam histogramas com muitas lacunas, ou seja, muitos valores não possuíam representação na imagem-chave. Nesses casos a imagem diminuiu o tempo de encriptação em até 60% na encriptação dos textos aleatórios e 32% na encriptação do Gênesis da Bíblia Cristã. Além disso, houve um aumento na entropia do arquivo encriptado em até 12% no arquivo encriptado a partir dos textos aleatórios e 10.42% no arquivo encriptado a partir do Gênesis da Bíblia Cristã.

Os testes envolvendo os representantes dos algoritmos simétricos e assimétricos com maior disseminação comercial, RSA e AES, mostraram resultados semelhantes aos resultados obtidos com o algoritmo original, pois o melhor resultado obtido nos testes anteriores foi utilizado na comparação. O método se mostrou mais lento que o RSA e AES, porém o método se mostrou mais eficiente na forma de esconder mensagens, pois, diferentemente dos algoritmos que sempre geram a mesma saída para uma entrada, o método proposto gera diferente saídas para uma mesma entrada, aumentando a dificuldade da quebra da mensagem encriptada.

A criptografia mostra-se imprescindível ao cotidiano do mundo moderno, principalmente em que tange a tecnologia da informação. A busca por algoritmos criptográficos mais eficientes vem convergindo o interesse para diversas pesquisas cientificas. A manipulação e transmissão de imagens em sistemas de comunicação, impulsionou o desenvolvimento da criptografia utilizando imagens, como a criptografia visual e a esteganografia.

O meu trabalho vem contribuir como mais uma alternativa, agora otimizado, para os algoritmos criptográficos de chave privada, utilizando imagens.

Os resultados superaram em muito as minhas expectativas iniciais, compensando a onerabilidade na manipulação de imagens com a leveza dos algoritmos simétricos e maior nível de entropia.

O algoritmo ainda pode ser melhorado para diminuir o tamanho de mensagem encriptada. Para tal, pode-se usar uma forma de compressão. Os testes mostraram que a entropia das mensagens encriptadas, ou seja, o número de bits necessários para representar a informação está acima do ideal. Um compressor por entropia iria diminuir o tamanho da mensagem, aproximando-o do ideal, diminuindo assim o tamanho do arquivo.

A própria estrutura de *flags* possibilita a implementação de um compressor aritmético de dois níveis. Isso traz vantagens e desvantagens, a vantagem é que a compressão serviria como uma segurança a mais na mensagem encriptada, pois um intruso não teria acesso direto à mensagem encriptada. A principal desvantagem dessa melhoria é o *overhead* computacional necessário para comprimir os dados.

O código-fonte esta disponível no repositório online: gitorious.org/cryptography-using-images

# 7 REFERÊNCIAS BIBLIOGRÁFICAS

x

|  |  |
| --- | --- |
| 1. | LIBERTY, J.; HORVATH, D. B. **Sams Teach Yourself C++ for Linux in 21 Days**. 2ª Edição. ed. Indiana: Sams Publishing, 2000. |
| 2. | GOODRICH, M. T.; TAMASSIA, R. **Projeto de Algoritmos**. Porto Alegre: Artmed, 2004. |
| 3. | DEITEL, H. M.; DEITEL, P. J. **C++ How To Program**. 3ª Edição. ed. New Jersey: Prentice-Hall, 2001. |
| 4. | GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2° Edição. ed. New Jersey: Prentice Hall, 2002. |
| 5. | CAMPOS, A. A. N. **Algoritmo de Criptografia AES em Hardware, Utilizando Dispositivo de Lógica Programável (FPGA) e Linguagem de Descrição de Hardware (VHDL)**. Itajubá - MG: [s.n.], 2008. |
| 6. | MENEZES, A.; OORSCHOT, P. V.; VANSTONE, S. **Handbook of Applied Criptography**. 1ª Edição. ed. [S.l.]: CRC Press, 2001. |
| 7. | TANENBAUM, A. S. **Redes de Computadores**. 3 Edição. ed. Manaus: Campus, 2002. |
| 8. | STALLINGS, W. **Cryptography and Network Security Principles and Practices**. 4ª. ed. [S.l.]: Prentice Hall, 2005. |
| 9. | **Electronic Frontier Foundation**. Disponivel em:  <http://w2.eff.org/Privacy/Crypto/Crypto\_misc/DESCracker/HTML/19980716\_eff\_des  \_faq.html>. Acesso em: 7 agosto 2010. |
| 10. | **COPACOBANA**. Disponivel em: <http://www.copacobana.org/>. Acesso em: 7 agosto 2010. |
| 11. | **SciEngines**. Disponivel em: <http://www.sciengines.com/joomla/index.php>. Acesso em: 7 agosto 2010. |
| 12. | NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Advanced Encryption Standard**, 26 novembro 2001. |
| 13. | SOUZA, R. D. A.; OLIVEIRA, F. B. D. O Padrão de Criptografia Simétrica AES, Petrópolis, 5 outubro 2007. |
| 14. | HANKERSON, D.; MENEZES, A.; VANSTONE, S. **Guide to Elliptic Curve Cryptography**. New York: Springer-Verlag New York, 2004. |
| 15. | WAGNER, N. R. The Laws of Cryptography with Java Code, 2003. |
| 16. | CACHIN, C. **Entropy Measures and Unconditional Security in Cryptography**, Zurich, 1997. |
| 17. | VAUDENAY, S. **Introduction to Cryptography - Applications for Communications Security**. New York: Springer Science+Business Media, 2006. |
| 18. | **Matlab Central**. Disponivel em:  <http://www.mathworks.com/matlabcentral/fileexchange/9083-rapidly-encrypt-and-decrypt-using-rsa>. Acesso em: 2010 agosto 7. |
| 19. | **Matlab Central**. Disponivel em:  <http://www.mathworks.com/matlabcentral/fileexchange/8925-rapid-aes-data-encryption-and-decryption>. Acesso em: 2010 agosto 9. |
| 20. | RAMALHO, M. Estudo de Métodos de Criptografia de Dados utilizando Imagens, Manaus, agosto 2010. |
| 21. | **ASCII Table - ASCII and Unicode Characters**. Disponivel em: <http://ascii-table.com/>. Acesso em: 2010 agosto 8. |
| 22. | ISO/IEC. **ISO/IEC 8859-1: Latin Alphabet No. 1**, 1997. |
| 23. | **Braingle**. Disponivel em: <http://www.braingle.com/>. Acesso em: 2010 Julho 29. |
| 24. | NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Data Encryption Standard**, 25 outubro 1999. |

x

# ANEXOS

# ANEXO A – TABELA ASCII E ISO/IEC 8859-1

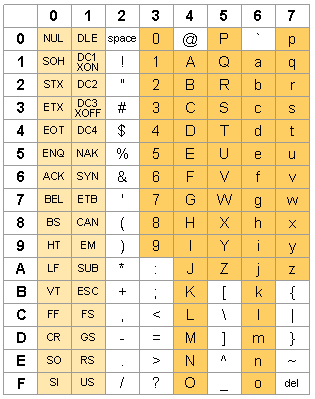


Figura 38: Tabela ASCII original.

Fonte: [21]

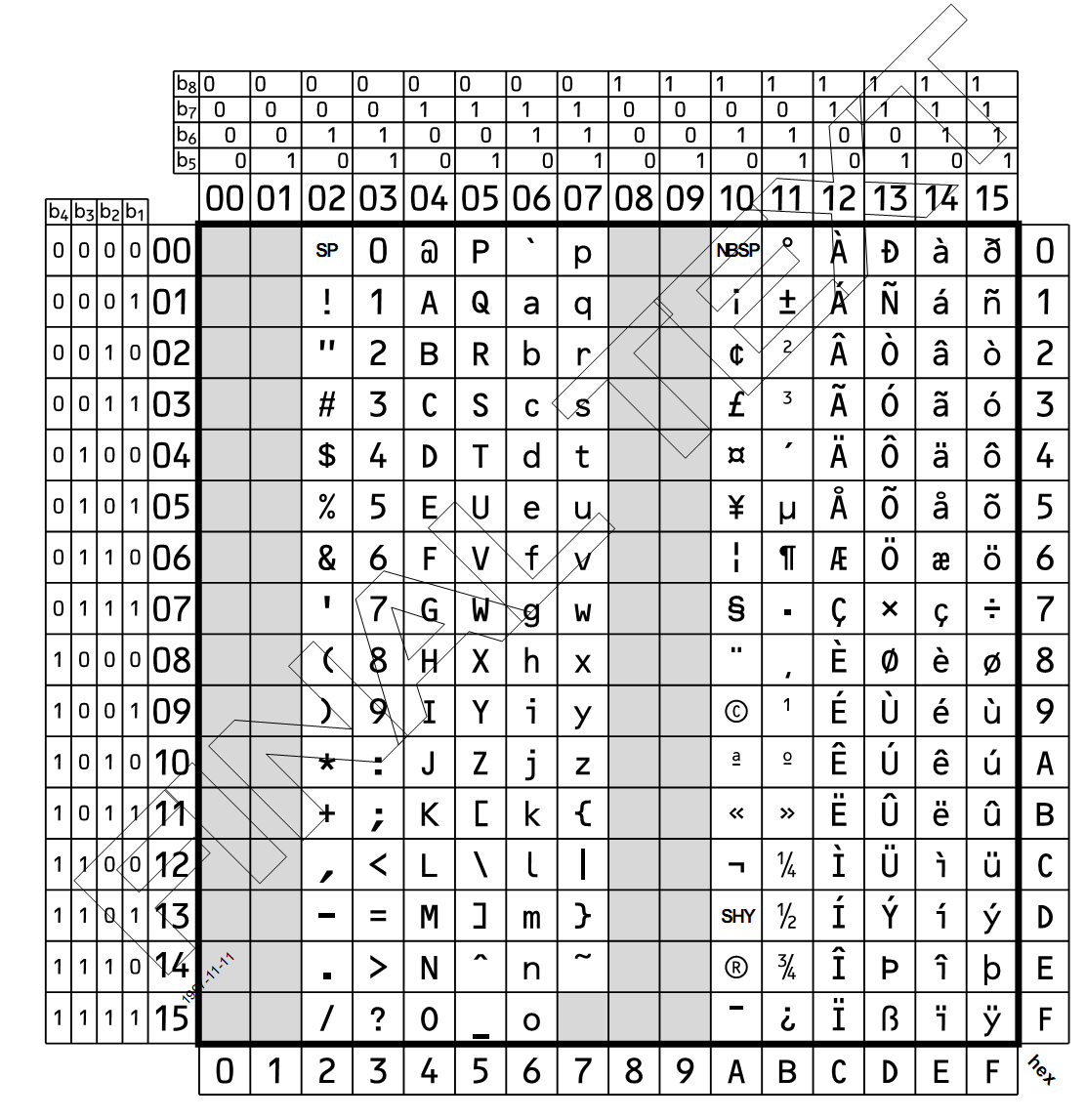


Figura 39: ISO/IEC 8859-1.

Fonte: [22]

# ANEXO B – BANCO DE IMAGENS

No anexo B estão todas as imagens utilizadas nos experimentos. Assim como seus respectivos histogramas.



Figura 40: Primeira imagem-chave utilizada nos experimentos. (aerial.pgm)

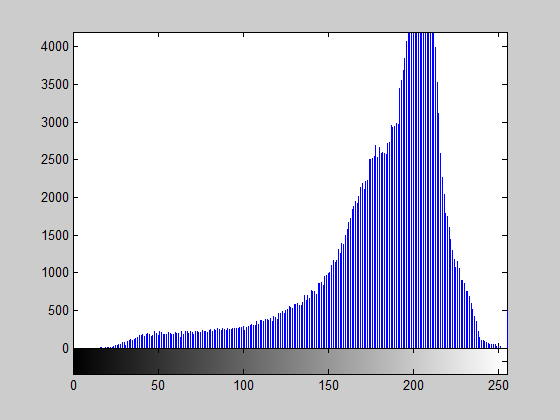


Figura 41: Histograma da imagem aerial.pgm



Figura 42: Segunda imagem-chave utilizada nos experimentos. (boats.pgm)

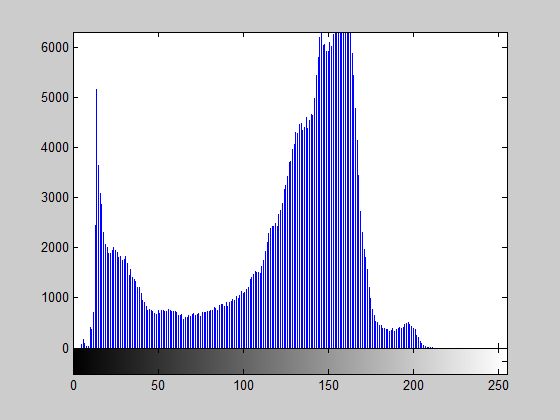


Figura 43: Histograma da imagem boats.pgm



Figura 44: Terceira imagem-chave utilizada nos experimentos. (bridge.pgm)

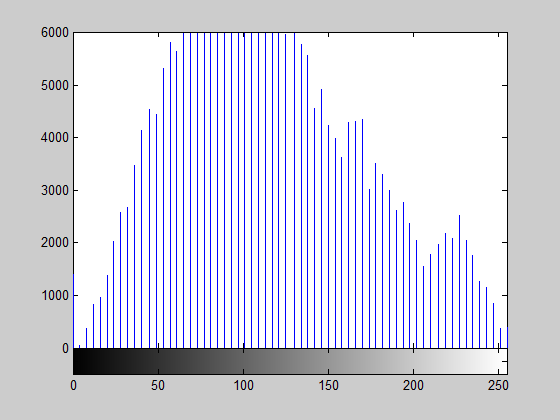


Figura 45: Histograma da imagem bridge.pgm



Figura 46: Quarta imagem-chave utilizada nos experimentos. (D108.pgm)

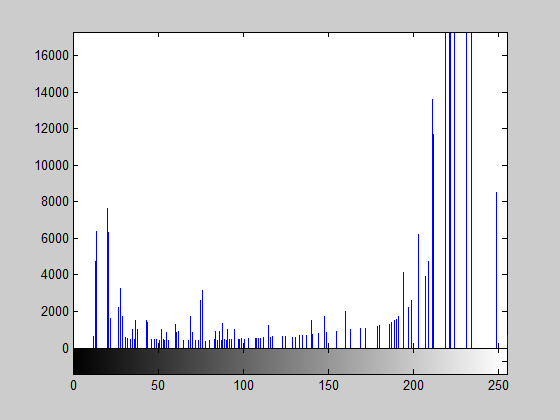


Figura 47: Histograma da imagem D108.pgm



Figura 48: Quinta imagem-chave utilizada nos experimentos. (f16.pgm)

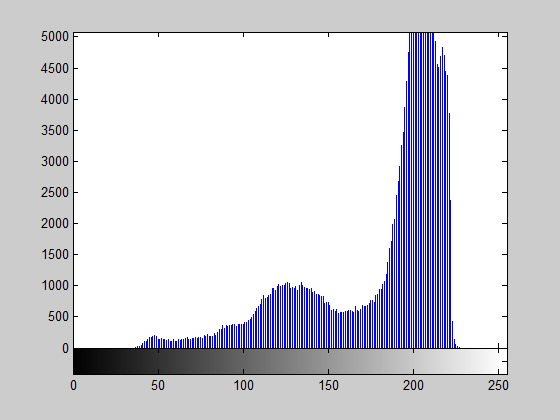


Figura 49: Histograma da imagem f16.pgm



Figura 50: Sexta imagem-chave utilizada nos experimentos. (girl.pgm)

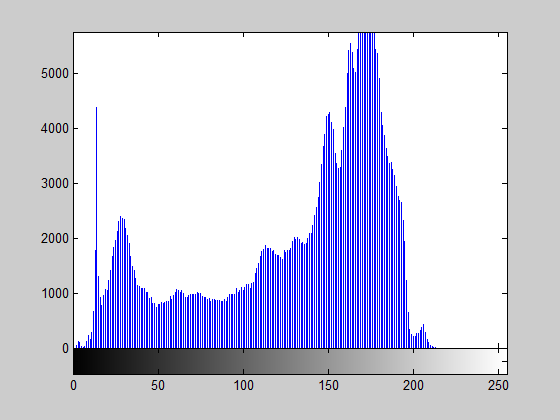


Figura 51: Histograma da imagem girl.pgm



Figura 52: Sétima imagem-chave utilizada nos experimentos. (lena.jpg)

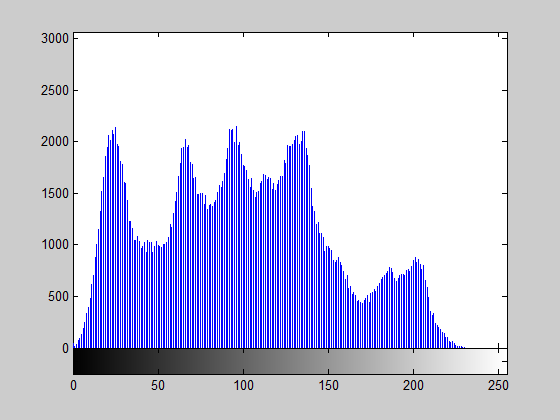


Figura 53: Histograma da imagem lena.jpg



Figura 54: Oitava imagem-chave utilizada nos experimentos. (peppers.pgm)

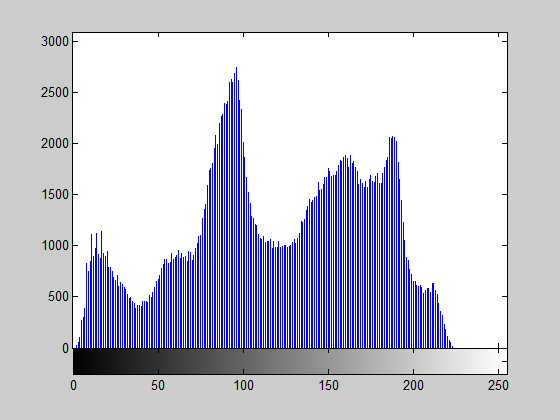


Figura 55: Histograma da imagem peppers.pgm

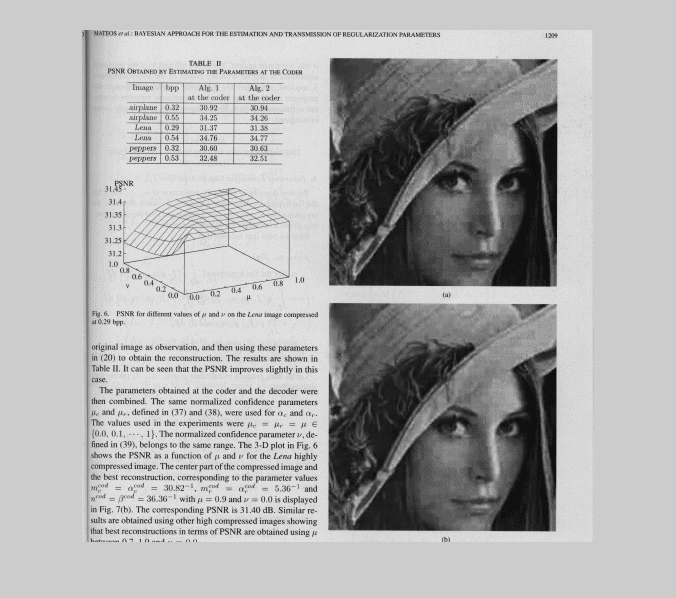


Figura 56: Nona imagem-chave utilizada nos experimentos. (pp1209.pgm)

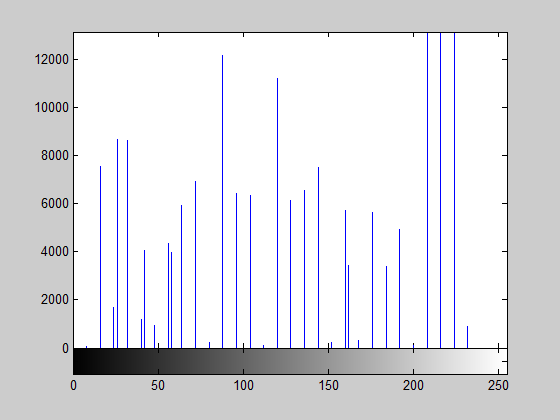


Figura 57: Histograma da imagem pp1209.pgm



Figura 58: Décima imagem-chave utilizada nos experimentos. (zelda.pgm)

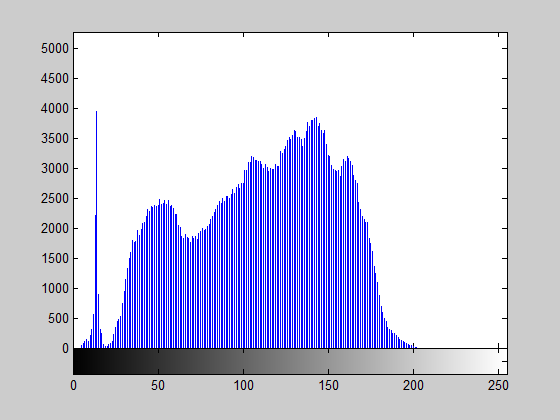


Figura 59: Histograma da imagem zelda.pgm