UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
CURSO DE GRADUAÇÃO EM TECNOLOGIA DA INFORMAÇÃO

Cleiane Clementino Bondade

Gabriel Guilherme Cavalcanti da Costa

Relatório de desenvolvimento de codificador e decodificador de código de barras usando Elixir.

Natal
2024

Sumário

1 INTRODUÇÃO

Esse relatório apresenta o processo de desenvolvimento de um programa usando Elixir para a codificação e decodificação de códigos de barras utilizados em boletos bancários.

Quanto à funcionalidade de codificação o programa deve receber como entrada um conjunto de dados que incluam código do banco, moeda, data de vencimento do boleto, valor do boleto e informações de convênio para cada boleto e deve ter como saída o código de barras e linha digitável de cada boleto.

Na funcionalidade de decodificação o programa deve receber como entrada uma lista de códigos de barras de boletos e deve retornar como saída os dados de código do banco, moeda, data de vencimento do boleto, valor do boleto, informações de convênio e linha digitável de cada boleto.

A codificação e decodificação dos dados se basearam nas instruções apresentadas do documento “Especificações Técnicas para Confecção de Boleto de Pagamento do Banco do Brasil”, disponível em:

https://www.bb.com.br/docs/pub/emp/empl/dwn/Doc5175Bloqueto.pdf

1.1 Tabela de participação

A implementação do programa foi dividida entre os membros do grupo, com a participação representada na tabela a seguir.

Nome	**Participação**	**Exemplos de participação**
Cleiane Bondade	10	LinhaDigitável, FatorVencimento.
Gabriel da Costa	10	CodigoDeBarras, FatorVencimento.

Nesse sentido, há módulos (como Linha Digitavel e CodigoDeBarras) em que a participação de ambos os membros da equipe só se deu de forma bastante pontual, e há módulos (como FatorVencimento) que foram implementados, na maior parte, de forma conjunta.

2 IMPLEMENTAÇÃO

Após a leitura das especificações, realizada em conjunto pelos membros do grupo, foram determinados blocos de funcionalidades que poderiam ser implementadas para fornecer uma solução.

Em primeiro lugar foram pensadas as funcionalidades relacionadas ao processo de codificação de dados em código de barras e linha digitável. Somente após a finalização dessa funcionalidade é que se iniciou o desenvolvimento da decodificação.

2.1 Codificação

Foram pensadas, inicialmente, divisões que incluíssem funções para: calcular o dígito verificador de cada bloco da linha digitável, calcular o dígito verificador do código de barras, converter data de vencimento em fator de vencimento, posicionar os dados do código de barras para a ordenação necessária na linha digitável, formatar a linha digitável para a apresentação padrão (que inclui pontos, e espaços em brancos), posicionar os dados codificados na ordenação necessária para o código de barras, ler os dados de um arquivo de entrada.

Ao longo da implementação outras funções foram utilizadas como auxiliares para melhor organizar os fluxos de processamento nas funções citadas no parágrafo anterior, e, além disso, foi-se percebendo a necessidade de novas funcionalidades para a completude da solução final.

Desde o início da implementação ficou determinada a criação de testes, para cada uma das funcionalidades principais, imediatamente após a implementação da primeira versão de cada função. Assim, todo o processo de desenvolvimento aqui relatado foi pontuado pela criação dos testes das respectivas funções associadas.

A ordem em que as funcionalidades foram implementadas priorizou a implementação daquelas com menos dependência de outras funções, assumindo-se, para isso, que, ao ser chamada, as funções chamadoras (a serem posteriormente implementadas) seriam responsáveis por entregar a entrada no formato adequado (já definido) e usar a saída gerada para executar as operações necessárias, definindo-se, assim, previamente, como os dados devem estar nesse dado momento do processamento. Nesse sentido, foram implementadas,

inicialmente, funções que realizam um processamento muito específico, e posteriormente funções mais gerais que coordenam e organizam aspectos mais amplos do programa.

As primeiras funcionalidades implementadas foram os cálculos dos dígitos verificadores dos blocos da linha digitável e do código de barras, e seus respectivos testes (como em todas as demais funções)

Dada uma lista de inteiros que representa um bloco da linha digitável (um bloco, nesse caso, sendo entendido como uma das três partes iniciais da linha digitável, com cada bloco separado por um espaço em branco), a função calcula o dígito verificador do bloco. A escolha por receber uma lista de inteiros deu-se porque é preciso efetuar operações aritméticas em cada elemento (ou dígito) do bloco, assim, caberia às outras partes do programa prover a entrada em um formato já pronto para a realização das operações. De forma semelhante para o código de barras, dada uma lista de inteiros que representa o código de barras (sem o dígito verificador), calcular o dígito verificador.

Como se pode observar, essas funções não utilizam nenhuma outra função do programa como parte de seus códigos, mas pressupõem a implementação de funções que vão, por exemplo, ler os dados do arquivo de entrada, converter os dados lidos para valores numéricos, organizar os dados codificados nas ordenações necessárias para o código de barras e linha digitável, etc. Essas funções serão apresentadas posteriormente.

A implementação das funções que realizam os cálculos dos dígitos verificadores foi distribuída entre os integrantes do grupo, assim como as demais funções do programa, resultado tem-se a função dvBloco no módulo LinhaDigitavel e a função dvCodigoBarras no módulo CodigoDeBarras.

A seguir foi implementada a funcionalidade de gerar o fator de vencimento a partir de uma data, o que, novamente, não depende de outras funcionalidades do programa.

Recebendo uma data como uma string no formato dia/mês/ano ou dia-mês-ano, dia e mês com dois dígitos cada e ano com 4, foi utilizada a biblioteca Timex para converter a string para o tipo Date e poder operar com o dado (subtraindo datas).

A função conta a diferença de dias entre a data informada e a data base definida em 03/07/2000, indo até o máximo de 9999, após o que reseta para 1000, valor que representa a diferença de 0 dias em relação à data base do ciclo. Ou seja, ao se chegar a uma diferença de 10000 dias em relação à última data base, inicia-se um novo ciclo, com uma nova data base, a qual possui diferença de 10000 dias para a data base anterior. O novo ciclo, por exemplo, inicia-se em 22/02/2025, assim, essa data terá fator de vencimento 1000, enquanto que 21/02/2025 terá fator de vencimento 9999.

Essa funcionalidade foi desenvolvida de forma conjunta pela equipe, sendo implementada na função fatorVencimento do módulo FatorVencimento (arquivo codigodebarras.ex).

As próximas funcionalidades foram a de ordenar a linha digitável, ler o arquivo de entrada e a de ordenar o código de barras.

A de ordenar para a linha digitável recebe como parâmetro uma lista de inteiros que representa o código de barras e gera uma nova lista reorganizando os elementos da lista de inteiros para deixá-los na ordem adequada da linha digitável. Sua implementação se deu na função ordenar, módulo LinhaDigitavel.

A escolha pela lista de inteiros deu-se pelo fato de que tanto funcionalidades anteriores quanto posteriores a esta precisarem efetuar operações aritméticas sobre cada dígito do código de barras, de forma que esse formato foi considerado adequado para as soluções pensadas.

Foi utilizada uma estratégia de "construir" a linha digitável de trás para frente, de forma a executar a concatenação de novos blocos de dígitos sempre no início da lista já construída até o momento, para otimizar o desempenho (para não ser necessário percorrer a cada concatenação todos os elementos de uma lista crescente para apensar os novos elementos ao final).

Logo a seguir foi implementada a função que realiza a leitura de um arquivo de texto contendo os dados de input para a geração de um código de barras e converte os dados para um map que será usado eventualmente para gerar o código de barras. Inicialmente essa funcionalidade considerava apenas um boleto, sendo posteriormente adaptada para receber mais de um.

A função considera que o arquivo contém somente os dados necessários, sendo 5 por boleto, na seguinte ordem: código do banco, moeda, data de vencimento(no formato dia/mês/ano ou dia-mês-ano), valor do boleto (com duas casas decimais, separadas por vírgula, conteúdo do convênio), cada dado separado por uma quebra de linha.

Em sua versão final, é possível inserir dados de vários boletos, mantendo o formato definido no parágrafo anterior e com cada novo boleto iniciando na linha imediatamente posterior à última linha do boleto anterior, exemplo:

001
9
21/08/2032
1,00
05009401448160606809350 31
002
8
22/08/2032
2,00
05009401448160606809350 31

Assim, a função tem como saída uma lista com maps gerados para cada boleto. A função que concretiza essa funcionalidade é a lerRegistros do módulo EstradaSaida (arquivo io.ex).

Em paralelo, foi implementada a função que inicia e coordena o processo de geração de um código de barras, incluindo a ordenação correta dos dados.

Recebendo um map com os dados obtidos da leitura do arquivo de entrada, essa função concatena na ordem correta os dígitos que representam os dados do código de barras, sendo necessário, para isso, utilizar funções já implementadas, como a de gerar o fator de vencimento a partir da data de vencimento recebida. Além disso, também é necessária para a correta ordenação dos dados a formatação do valor do boleto para o formato adequado para o código de barras (sem vírgula e com zeros à esquerda até complementar 10 dígitos).

Essa função também converte a string gerada na ordenação inicial para a lista de inteiros (para ser passada para as funções mencionadas anteriormente que

precisam operar sobre cada dígito). Chama a função para calcular o dígito verificador e o insere na posição adequada. Implementada na função gerar do módulo CodigoDeBarras.

Para completar as funcionalidades relacionadas à linha digitável foram criadas funções para coordenar o cálculo do dígito verificador para cada bloco e a inserção do resultado na posição correta (inserirDvs); para converter a lista (de inteiros) final resultante da anterior em uma string (toString), adicionando a formatação adequada (pontos e espaços em branco nas posições necessárias), e a que dá início ao processo de geração da linha digitável (gerar), a qual recebe uma lista de inteiros que representa o código de barras e chama as funções para ordenar a lista, inserir os dígitos verificadores e, por fim, converter a linha digitável para string. Todas essas funções tendo sido implementadas no módulo LinhaDigitavel.

Para iniciar o processo de codificação como um todo (tanto código de barras quanto linha digitável) foi implementada a função codificador no módulo App, chamando a função para leitura do arquivo de input, passando os dados para a função de geração do código de barras, geração da linha digitável e saída do resultado da codificação. Essa função usa, ainda, a biblioteca barlix, para, após o código de barras gerado, gerar a imagem que o representa e salvá-la na pasta imagens.

Antes de iniciar o desenvolvimento de funções necessárias à decodificação algumas funções foram refatoradas, fosse para otimizar o desempenho, com base nos tempos de execução exibidos nos testes automáticos, fosse para melhorar a legibilidade do código.

2.1 Decodificação

Para o processo de decodificação também foi criada no módulo App uma função (decodificador) que inicia o processo, chamando uma nova função para ler os dados do arquivo de entrada e passando esses dados para o decodificador do código de barras.

A leitura para decodificação é feita por uma nova função (lerCodigos) do módulo EntradaSaida, isso porque o formato esperado da entrada é diferente. Para a decodificação espera-se uma sequência de dígitos que representa um código de barras. Mais de um código de barras pode ser inserido no arquivo de input, cada um

separado do anterior por uma quebra de linha. O conteúdo é retornado como uma lista de strings.

A função decodificador do módulo CodigoDeBarras recebe uma string representando o código de barras e extrai dela os dados do boleto, para isso é necessário uma nova função (decode) no módulo FatorVencimento, para decodificar o fator de vencimento do código de barras, convertendo-o para uma data e uma função para formar o valor do boleto como moeda.

A decodificação do fator de vencimento para data utilizou, novamente, a biblioteca Timex. A função realiza soma de dias, baseados no fato de vencimento, à data base, até obter uma data igual ou posterior ao dia atual, e então, usando Timex, formata esse valor para uma string.

Aqui cabe apontar que se optou por converter o fator de vencimento para a próxima data de vencimento maior ou igual à data atual. Assim, por exemplo, se o fator de vencimento indicar 9701, a data poderia ser, por exemplo, 29/04/2024 ou 19/12/2048, nesse caso, o algoritmo vai indicar 19/12/2048.

A função stringParaMoeda foi utilizada para formatar o valor, em string, como moeda (com vírgula separando as casas decimais, sendo duas casas decimais).

Por fim, o decodificador também precisa converter o código inicial para uma lista de inteiros para poder gerar a linha digitável. Os dados decodificados e a linha digitável são, então, retornados como um map.

3 LIMITAÇÕES

No projeto desenvolvido cabe destacar 4 limitações importantes.

A primeira é que não existe uma funcionalidade que, recebendo uma representação em imagem de um código de barras, decodifique os padrões para recuperar as informações do boleto.

A segunda é que não é efetuado nenhum tipo de processamento sobre os dados relacionados ao convênio.

Embora nas especificações que se se baseou o programa haja instruções para os processamentos que devem ser realizados sobre os dados de convênio, ou seja, os dados dos dígitos 20 a 44 do código de barras, que incluem identificação do tipo de convênio e cálculo de dígito verificador (em algumas situações), nessa implementação esse campo não é decodificado, não se faz a identificação dos dados, não se faz o cálculo ou verificação do dígito verificador. Esses dígitos são apenas ordenados corretamente para a geração da linha digitável, mas nenhum outro processamento é realizado sobre eles.

A terceira limitação é que não se tratou os casos em que o valor do documento é superior a R$ 99.999.999,99. Nesses casos a indicação é de que o valor deve avançar sobre o fator de vencimento, suprimindo-o do código de barras. Assim, o projeto considera apenas valores inferiores a esse montante, e a utilização de valores superiores acarretará na geração de um resultado incorreto para as especificações estabelecidas.

A quarta limitação é que não há verificação das entradas fornecidas para o programa. Espera-se que as entradas sejam fornecidas corretamente, o que inclui caminho e nome correto do arquivo entrada, formato dos dados, tamanho, ordem, etc. A utilização de dados inadequados pode resultar em erros (não tratados), resultados inesperados ou incorretos.

4 UTILIZAÇÃO

O código do programa pode ser baixado em: https://github.com/gabriel-guilherme/codigo_de_barras_elixir .

Para executar o programa será necessário instalar as dependências com mix deps.get.

Após compilado, é possível rodar os testes usando mix test. São 17 testes, e um deles, a função decode do módulo FatorVencimento, é esperado que falhe. A falha nesse teste se dará porque, como já foi mencionado, o decode irá retornar sempre uma data igual ou posterior à data atual, assim, no dia de apresentação desse trabalho, 09/05/2024, ao testar o decode para 9711 o programa retornará corretamente 09/05/2024, e o teste passará; no entanto, após essa data, se não

houver alteração no código do teste, o teste permanece esperando o resultado 09/05/2024, mas a função retornará uma data futura, do próximo ciclo.

Os arquivos de entrada utilizados nos testes são os arquivos de texto na pasta test: input_codigos_test.txt, input_codigos2_test.txt, input_test.txt, input_tests2.txt. Os dois primeiros fornecem dados de teste para o decodificador, ou seja, código de barras para serem decodificados. O primeiro apenas um código de barras, e o segundo dois. Os arquivos seguintes fornecem dados de teste para codificador, ou seja, registros contendo as informações do código do banco, moeda, data de vencimento, valor do boleto e dados de convênio, o primeiro deles apenas um registro e o segundo dois.

Para executar o programa no modo iterativo, foram disponibilizados, na raiz do projeto, dois arquivos que podem ser usados como exemplo, o input.txt e o decode.txt. Assim, pode-se executar a funcionalidade de codificação usando App.codificador("input.txt") e a de decodificação usando App.decodificador("decode.txt"). Outros arquivos com os dados podem ser criados e utilizados como parâmetro para o App.codificador e o App.decodificador.