Trabalho de Implementação I

v1.0

Gerado por Doxygen 1.8.17

Chapter 1

Trabalho de Implementação I - Processamento de XML com imagens binárias

Objetivo

Este trabalho consiste na utilização de **estruturas de dados lineares**, vistas até o momento no curso, e aplicação de conceitos de **pilha** e/ou **fila** para o processamento de arquivos **XML** contendo **imagens binárias**. A implementação deverá resolver dois problemas (listados a seguir), e os resultados deverão ser formatados em saída padrão de tela de modo que possam ser automaticamente avaliados no VPL.

Primeiro problema: validação de arquivo XML

Para esta parte, pede-se exclusivamente a **verificação de aninhamento e fechamento das marcações** (tags) no arquivo XML (qualquer outra fonte de erro pode ser ignorada). Um identificador (por exemplo: `img`) constitui uma marcação entre os caracteres $<$ e $>$, podendo ser de abertura (por exemplo: $<$`img`$>$) ou de fechamento com uma `/` antes do identificador (por exemplo: $<$`/img`$>$).

Como apresentando em sala de aula, o algoritmo para resolver este problema é baseado em pilha (**LIFO**):

- Ao encontrar uma marcação de abertura, empilha o identificador.
- Ao encontrar uma marcação de fechamento, verifica se o topo da pilha tem o mesmo identificador e desempilha. Aqui duas situações de erro podem ocorrer:
  - Ao consultar o topo, o identificador é diferente (ou seja, uma marcação aberta deveria ter sido fechada antes);
  - Ao consultar o topo, a pilha encontra-se vazia (ou seja, uma marcação é fechada sem que tenha sido aberta antes);
- Ao finalizar a análise (parser) do arquivo, é necessário que a pilha esteja vazia. Caso não esteja, mais uma situação de erro ocorre, ou seja, há marcação sem fechamento.

Segundo problema: contagem de componentes conexos em imagens binárias representadas em arquivo XML

Cada XML contém imagens binárias, com altura e largura definidas respectivamente pelas marcações `<height>` e `<width>`, e sequência dos pixels com valores binários, de intensidade **0 para preto** ou **1 para branco**, em modo texto (embora fosse melhor gravar 1 byte a cada 8 bits, optou-se pelo modo texto por simplicidade), na marcação `<data>`.

Para cada uma dessas imagens, pretende-se **calcular o número de** ∗**componentes conexos**∗∗∗ **usando** ∗∗**vizinhança-4**. Para isso, seguem algumas definições importantes:

- A ***vizinhança-4*** de um pixel na linha *x* e coluna *y*, ou seja, na coordenada ∗∗(x, y)∗∗, é um conjunto de pixels adjacentes nas coordenadas:

```
          (x, y+1)
(x-1, y)            (x+1, y)
          (x, y-1)
```

- Um ***caminho*** entre um um pixel p_1 e outro p_n é uma sequência de pixels distintos ∗∗<p_1,p_2,...,p_n>∗∗, de modo que p_i é **vizinho-4** de p_{i+1}; sendo i=1,2,...,n-1
- Um pixel *p* é ***conexo*** a um pixel *q* se existir um **caminho** de *p* a *q* (no contexto deste trabalho, só há interesse em pixels com intensidade 1, ou seja, brancos).
- Um ***componente conexo*** é um *conjunto maximal* (não há outro maior que o contenha) *C* de pixels, no qual **quaisquer dois pixels** selecionados deste conjunto *C* são **conexos**.

Para a determinação da quantidade de componentes conexos, antes é necessário atribuir um **rótulo** inteiro e crescente (1, 2, ...) para cada pixel de cada componente conexo. Conforme apresentado em aula, segue o algoritmo de rotulação (*labeling*) usando uma fila (**FIFO**):

- Inicializar `rótulo` com 1.
- Criar uma matriz `R` de zeros com o mesmo tamanho da matriz de entrada `E` lida.
- Varrer a matriz de entrada `E`.
  - Assim que encontrar o primeiro pixel de intensidade **1 ainda não visitado** (igual a **0** na mesma coordenada em `R`).
    - Inserir `(x,y)` na fila.
    - Na coordenada `(x,y)` da imagem `R`, atribuir o `rótulo` atual.
  - Enquanto a fila não estiver vazia:
    - Remover `(x,y)` da fila.
    - Inserir na fila as coordenadas dos quatro vizinhos que estejam dentro do domínio da imagem (não pode ter coordenada negativa ou superar o número de linhas ou de colunas), com intensidade **1** (em `E`) e ainda não tenha sido visitado (igual a **0** em `R`).
      - Na coordenada de cada vizinho selecionado, na imagem `R`, atribuir o `rótulo` atual.
  - Incrementar o `rótulo`.
- O conteúdo final da matriz `R` corresponde ao resultado da rotulação. A **quantidade de componentes conexos**, que é a resposta do segundo problema, é igual ao último e **maior *rótulo* atribuído**.

Chapter 2

Índice dos namespaces

2.1 Lista de namespaces

Lista dos namespaces com uma breve descrição:

Chapter 3

Índice dos componentes

3.1 Lista de componentes

Lista de classes, estruturas, uniões e interfaces com uma breve descrição:

Chapter 4

Índice dos ficheiros

4.1 Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

Chapter 5

Documentação dos namespaces

5.1 Referência ao namespace math

Código de natureza matemática.

Funções

- int ∗∗ matrix_init (int height, int width)

  Inicializa uma matriz com as dimensões especificadas.
- void matrix_destroy (int ∗∗M, int height)

  Destroi uma matriz e libera a memória que ocupava.
- int count_shapes (int ∗∗E, int height, int width)

  Calcula o número de componentes conexos na matriz usando vizinhança-4.

5.1.1 Descrição detalhada

Código de natureza matemática.

5.1.2 Documentação das funções

5.1.2.1 count_shapes()

```
int math::count_shapes (
            int ** E,
            int height,
            int width )
```

Calcula o número de componentes conexos na matriz usando vizinhança-4.

Utiliza a técnica de rotulação de formas, para tal criando uma matriz temporária do mesmo tamanho da de entrada: este algoritmo utiliza memória na ordem O(w∗h).

Cada "pixel" é processado em uma fila (FIFO) de tamanho dinâmico, assim como seus vizinhos e assim por diante ate percorrer todos os caminhos do componente.

Parâmetros

E	Matriz de entrada.
height	Número de linhas da matriz.
width	Número de colunas da matriz.

Retorna

int Número de componentes conexos (formas) encontrados. Zero implica que a matriz é nula/vazia.

Definido na linha 38 do ficheiro matrix.cpp.

Referenciado por main().

5.1.2.2 matrix_destroy()

```
void math::matrix_destroy (
            int ** M,
            int height )
```

Destroi uma matriz e libera a memória que ocupava.

Parâmetros

M	Matriz anteriormente inicializada por matrix_init().
height	Número de linhas da matriz. Deve ser o mesmo valor usado em sua inicialização.

Definido na linha 32 do ficheiro matrix.cpp.

Referenciado por count_shapes() e main().

5.1.2.3 matrix_init()

```
int ** math::matrix_init (
            int height,
            int width )
```

Inicializa uma matriz com as dimensões especificadas.

A matriz é dada na forma de um array de arrays onde todos os elementos são inicializados com zero, uma matriz nula.

Parâmetros

height	Número de linhas da matriz.
width	Número de colunas da matriz.

Retorna

int** Matriz gerada. Deve ser destruído com matrix_destroy() para liberar a memória alocada.

Definido na linha 21 do ficheiro matrix.cpp.

Referenciado por count_shapes() e matrix_init().

5.2 Referência ao namespace structures

Estruturas de Dados.

Componentes

- class LinkedQueue

  Fila Encadeada.
- class LinkedStack

  Pilha Encadeada.

5.2.1 Descrição detalhada

Estruturas de Dados.

5.3 Referência ao namespace xml

Utilitários para processamento de XML.

Funções

- bool balanced (const std::string &xml)

  Confere a validez da estrutura do XML contido na string.
- std::string extract (const std::string &origin, const std::string &open, const std::string &close, std::size_t &from)

  Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados a partir de uma dada posição.
- std::string extract (const std::string &origin, const std::string &open, const std::string &close)

  Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados.

5.3.1 Descrição detalhada

Utilitários para processamento de XML.

5.3.2 Documentação das funções

5.3.2.1 balanced()

```
bool xml::balanced (
            const std::string & xml )
```

Confere a validez da estrutura do XML contido na string.

A validação consiste em verificar se as tags estão balanceadas, ou seja, se para cada tag fechada houve seu par de abertura como última tag processada; e se todas as tags abertas foram devidamente fechadas. Para tal, este algoritmo utiliza uma pilha (LIFO) de tamanho dinâmico.

Parâmetros

xml	String contendo o XML.

Retorna

true Tags estão balanceadas.

false Tags não estão balanceadas.

Definido na linha 21 do ficheiro xml.cpp.

Referenciado por main().

5.3.2.2 extract() [1/2]

```
std::string xml::extract (
            const std::string & origin,
            const std::string & open,
            const std::string & close )
```

Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados.

Parâmetros

origin	String original.
open	Delimitador de abertura.
close	Delimitador de fechamento.

Retorna

std::string String extraída (sem os delimitadores), vazia quando nada for encontrado.

Definido na linha 79 do ficheiro xml.cpp.

5.3.2.3 extract() [2/2]

```
std::string xml::extract (
            const std::string & origin,
            const std::string & open,
            const std::string & close,
            std::size_t & from )
```

Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados a partir de uma dada posição.

Parâmetros

origin	String original.
open	Delimitador de abertura.
close	Delimitador de fechamento.
from	Índice por onde iniciar a busca na string original, este será alterado para a posição após o final do delimitador de fechamento encontrado. Se nada for encontrado, recebe o valor de `std::string::npos`.

Retorna

std::string String extraída (sem os delimitadores), vazia quando nada for encontrado.

Definido na linha 62 do ficheiro xml.cpp.

Referenciado por extract() e main().

Chapter 6

Documentação da classe

6.1 Referência à classe Template structures::LinkedQueue< T >

Fila Encadeada.

`#include <linked_queue.h>`

Membros públicos

- ~LinkedQueue ()
  Destrutor.
- void clear ()
  Limpa a Fila.
- void enqueue (const T &data)
  Enfileira.
- T dequeue ()
  Desenfileira.
- T & front () const
  Acessa a frente da Fila.
- T & back () const
  Acessa o último da Fila.
- bool empty () const
  Confere se a Fila está vazia.
- std::size_t size () const
  Retorna o tamanho da Fila.

6.1.1 Descrição detalhada

template<typename T>
class structures::LinkedQueue< T >

Fila Encadeada.

Definido na linha 15 do ficheiro linked_queue.h.

6.1.2 Documentação dos Construtores & Destrutor

6.1.2.1 ~LinkedQueue()

```
template<typename T >
LinkedQueue::~LinkedQueue ( )
```

Destrutor.

Definido na linha 13 do ficheiro linked_queue.inc.

6.1.3 Documentação dos métodos

6.1.3.1 back()

```
template<typename T >
T & LinkedQueue::back ( ) const
```

Acessa o último da Fila.

Definido na linha 67 do ficheiro linked_queue.inc.

6.1.3.2 clear()

```
template<typename T >
void LinkedQueue::clear ( )
```

Limpa a Fila.

Definido na linha 18 do ficheiro linked_queue.inc.

6.1.3.3 dequeue()

```
template<typename T >
T LinkedQueue::dequeue ( )
```

Desenfileira.

Definido na linha 33 do ficheiro linked_queue.inc.

Referenciado por math::count_shapes().

6.1.3.4 empty()

```
template<typename T >
bool LinkedQueue::empty ( ) const
```

Confere se a Fila está vazia.

Definido na linha 75 do ficheiro linked_queue.inc.

Referenciado por math::count_shapes().

6.1.3.5 enqueue()

```
template<typename T >
void LinkedQueue::enqueue (
            const T & data )
```

Enfileira.

Definido na linha 24 do ficheiro linked_queue.inc.

Referenciado por math::count_shapes().

6.1.3.6 front()

```
template<typename T >
T & LinkedQueue::front ( ) const
```

Acessa a frente da Fila.

Definido na linha 59 do ficheiro linked_queue.inc.

6.1.3.7 size()

```
template<typename T >
std::size_t LinkedQueue::size ( ) const
```

Retorna o tamanho da Fila.

Definido na linha 80 do ficheiro linked_queue.inc.

A documentação para esta classe foi gerada a partir dos seguintes ficheiros:

- linked_queue.h
- linked_queue.inc

6.2 Referência à classe Template structures::LinkedStack< T >

Pilha Encadeada.

`#include <linked_stack.h>`

Membros públicos

- ~LinkedStack ()
  Destrutor.
- void push (const T &data)
  Empilha.
- T pop ()
  Desempilha.
- T & top () const
  Acessa o topo da Pilha.
- bool empty () const
  Confere se a Pilha está vazia.
- std::size_t size () const
  Retorna o tamanho da Pilha.
- void clear ()
  Limpa a Pilha.

6.2.1 Descrição detalhada

template<typename T>
class structures::LinkedStack< T >

Pilha Encadeada.

Definido na linha 23 do ficheiro linked_stack.h.

6.2.2 Documentação dos Construtores & Destrutor

6.2.2.1 ~LinkedStack()

```
template<typename T >
LinkedStack::~LinkedStack ( )
```

Destrutor.

Definido na linha 13 do ficheiro linked_stack.inc.

6.2.3 Documentação dos métodos

6.2.3.1 clear()

```
template<typename T >
void LinkedStack::clear ( )
```

Limpa a Pilha.

Definido na linha 18 do ficheiro linked_stack.inc.

6.2.3.2 empty()

```
template<typename T >
bool LinkedStack::empty ( ) const
```

Confere se a Pilha está vazia.

Definido na linha 57 do ficheiro linked_stack.inc.

Referenciado por xml::balanced().

6.2.3.3 pop()

```
template<typename T >
T LinkedStack::pop ( )
```

Desempilha.

Definido na linha 30 do ficheiro linked_stack.inc.

Referenciado por xml::balanced().

6.2.3.4 push()

```
template<typename T >
void LinkedStack::push (
             const T & data )
```

Empilha.

Definido na linha 24 do ficheiro linked_stack.inc.

Referenciado por xml::balanced().

6.2.3.5 size()

```
template<typename T >
std::size_t LinkedStack::size ( ) const
```

Retorna o tamanho da Pilha.

Definido na linha 62 do ficheiro linked_stack.inc.

6.2.3.6 top()

```
template<typename T >
T & LinkedStack::top ( ) const
```

Acessa o topo da Pilha.

Definido na linha 49 do ficheiro linked_stack.inc.

Referenciado por xml::balanced().

A documentação para esta classe foi gerada a partir dos seguintes ficheiros:

- linked_stack.h
- linked_stack.inc

Chapter 7

Documentação do ficheiro

7.1 Referência ao ficheiro linked_queue.h

```
#include <cstdint>
#include <stdexcept>
#include "linked_queue.inc"
```

Componentes

- class structures::LinkedQueue< T >

  Fila Encadeada.

Namespaces

- structures

  Estruturas de Dados.

7.2 Referência ao ficheiro linked_queue.inc

Implementações da Fila Encadeada.

7.2.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright

7.3 Referência ao ficheiro linked_stack.h

Implementações da Fila Encadeada.

```
#include <cstdint>
#include <stdexcept>
#include "linked_stack.inc"
```

Componentes

- class structures::LinkedStack< T >

  Pilha Encadeada.

Namespaces

- structures

  Estruturas de Dados.

7.3.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright

7.4 Referência ao ficheiro linked_stack.inc

Implementações da Fila Encadeada.

7.4.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright

7.5 Referência ao ficheiro main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <cctype>
#include "xml.h"
#include "linked_queue.h"
#include "matrix.h"
```

Funções

- static int ** matrix_init (int height, int width, const std::string &data)

  Copyright [2021] <Rafael Nilson Witt>
- int main ()

  Programa principal, realiza a leitura e processamento dos XMLs e conta o número de componentes conexos nas imagens contidas nos mesmos.

7.5.1 Documentação das funções

7.5.1.1 main()

```
int main ( )
```

Programa principal, realiza a leitura e processamento dos XMLs e conta o número de componentes conexos nas imagens contidas nos mesmos.

Resultados de cada imagem são disponibilizados na saída padrão.

Retorna

int Algum dos seguintes códigos de erro: 0 quando não houver erros; 1 quando não foi possivel abrir o arquivo lido; -1 quando o XML lido é inválido; -2 quando alguma das imagens apresenta dimensões inválidas.

Definido na linha 36 do ficheiro main.cpp.

7.5.1.2 matrix_init()

```
static int ** matrix_init (
            int height,
            int width,
            const std::string & data )  [static]
```

Copyright [2021] <Rafael Nilson Witt>

Inicializa uma matriz de inteiros a partir da string que a representa.

Parâmetros

height	Número de linhas da matriz.
width	Múmero de colunas da matriz.
data	String contendo os valores colocados na matriz. Whitespace é ignorado.

Retorna

int** Matriz gerada. Deve ser destruido com matrix_destroy() para liberar a memória alocada.

Definido na linha 87 do ficheiro main.cpp.

Referenciado por main().

7.6 Referência ao ficheiro matrix.cpp

Implementações da Fila Encadeada.

```
#include "matrix.h"
#include <cassert>
#include <utility>
#include "linked_queue.h"
```

Namespaces

- math

  Código de natureza matemática.

Funções

- int ∗∗ math::matrix_init (int height, int width)

  Inicializa uma matriz com as dimensões especificadas.
- void math::matrix_destroy (int ∗∗M, int height)

  Destroi uma matriz e libera a memória que ocupava.
- int math::count_shapes (int ∗∗E, int height, int width)

  Calcula o número de componentes conexos na matriz usando vizinhança-4.

7.6.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright

7.7 Referência ao ficheiro matrix.h

Implementações da Fila Encadeada.

Namespaces

- math

  Código de natureza matemática.

Funções

- int ** math::matrix_init (int height, int width)

  Inicializa uma matriz com as dimensões especificadas.
- void math::matrix_destroy (int **M, int height)

  Destroi uma matriz e libera a memória que ocupava.
- int math::count_shapes (int **E, int height, int width)

  Calcula o número de componentes conexos na matriz usando vizinhança-4.

7.7.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright

7.8 Referência ao ficheiro README.md

7.9 Referência ao ficheiro xml.cpp

Implementações da Fila Encadeada.

```
#include "xml.h"
#include <string>
#include <cstddef>
#include "linked_stack.h"
```

Namespaces

- xml

  Utilitários para processamento de XML.

Funções

- bool xml::balanced (const std::string &xml)

  Confere a validez da estrutura do XML contido na string.
- std::string xml::extract (const std::string &origin, const std::string &open, const std::string &close, std::size_t &from)

  Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados a partir de uma dada posição.
- std::string xml::extract (const std::string &origin, const std::string &open, const std::string &close)

  Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados.

7.9.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright

7.10 Referência ao ficheiro xml.h

Implementações da Fila Encadeada.

```
#include <string>
#include <cstddef>
```

Namespaces

- xml

  Utilitários para processamento de XML.

Funções

- bool xml::balanced (const std::string &xml)

  Confere a validez da estrutura do XML contido na string.
- std::string xml::extract (const std::string &origin, const std::string &open, const std::string &close, std::size_t &from)

  Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados a partir de uma dada posição.
- std::string xml::extract (const std::string &origin, const std::string &open, const std::string &close)

  Extrai, a partir de uma string original, a substring que existir entre o primeiro par de delimitadores encontrados.

7.10.1 Descrição detalhada

Implementações da Fila Encadeada.

Autor

Rafael Nilson Witt

Versão

1.0

Data

2021-03-23

Copyright