

Trabalho de Implementação I

Alek Frohlich e Gabriel B. Sant'Anna

Versão v1.0

Domingo, 19 de Maio de 2019

Trabalho de Implementação I - Processamento de XML com imagens binárias

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: ``) ou de fechamento com uma `/` antes do identificador (por exemplo: ``).

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 coluna y , ou seja, na coordenada (x, y) , é um conjunto de pixels adjacentes nas coordenadas:

$$\begin{array}{ccc} & (x, y+1) & \\ (x-1, y) & & (x+1, y) \\ & (x, y-1) & \end{array}$$

- ∞ Um **caminho** entre um pixel p_i e outro p_n é uma sequência de pixels distintos $\langle p_1, p_2, \dots, p_n \rangle$, de modo que p_i é **vizinho-4** de p_{i+1} ; sendo $i=1, 2, \dots, n-1$
- ∞ 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 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.

Índice

1. Lista de namespaces

- 1.1. [math](#) (Código de natureza matemática)
- 1.2. [structures](#) (Estruturas de Dados)
- 1.3. [xml](#) (Utilitários para processamento de XML)

2. Lista de componentes

- 2.1. [structures::LinkedList< T >](#) (Fila Encadeada)
- 2.2. [structures::LinkedStack< T >](#) (Pilha Encadeada)

3. Lista de arquivos

- 3.1. [linked_queue.h](#) (Arquivo de declarações e interface da Fila Encadeada)
- 3.2. [linked_queue.inc](#) (Implementações da Fila Encadeada)
- 3.3. [linked_stack.h](#) (Declarações e interface da Pilha Encadeada)
- 3.4. [linked_stack.inc](#) (Implementações da Pilha Encadeada)
- 3.5. [main.cpp](#) (Código do programa principal)
- 3.6. [matrix.cpp](#) (Implementação das funções de processamento de matrizes)
- 3.7. [matrix.h](#) (Declarações das funções de processamento de matrizes)
- 3.8. [xml.cpp](#) (Implementação das funções de processamento de XML)
- 3.9. [xml.h](#) (Declarações das funções de processamento de XML)

Documentação dos namespaces

Lista dos namespaces utilizados com uma breve descrição de seus respectivos métodos:

Referência ao namespace math

Código de natureza matemática. Aqui consta a parte do código responsável por gerenciar a memória de matrizes alocadas através de sua interface. Além disso, o namespace math também disponibiliza a função fundamental para a solução do segundo problema, a contagem de componentes conexos dentro de uma imagem binária, cuja representação aqui é feita através de matrizes.

Funções:

- ∞ `int** matrix_init (int height, int width)`
Inicializa uma matriz com as dimensões especificadas.
- ∞ `void matrix_destroy (int**M, int height)`
Destrói uma matriz e libera a memória que ela ocupava.
- ∞ `int count_shapes (int**E, int height, int width)`
Calcula o número de componentes conexos na matriz usando vizinhança-4.

Documentação das funções

`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 $\Theta(wh)$, onde w e h representam, width e height respectivamente.

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

Parâmetros:

<i>E</i>	Matriz de entrada.
<i>height</i>	Número de linhas da matriz.
<i>width</i>	Número de colunas da matriz.

Retorna:

Um inteiro indicando o número de componentes conexos (formas) encontrados. Zero implica que a matriz é nula/vazia.

Definido na linha 38 do arquivo matrix.cpp.
Referenciado por main().

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

Destrói uma matriz e libera a memória que ela ocupava.

Parâmetros:

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

Definido na linha 32 do arquivo matrix.cpp.
Referenciado por count_shapes() e main().

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:

<i>height</i>	Número de linhas da matriz.
<i>width</i>	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 arquivo matrix.cpp.
Referenciado por count_shapes() e matrix_init().

Referência ao namespace structures

structures

Estruturas de Dados.

Componentes

∞ class [LinkedQueue](#)
Fila Encadeada.

∞ class [LinkedStack](#)
Pilha Encadeada.

Descrição detalhada

Estruturas de Dados.

Referência ao namespace xml

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.

Descrição detalhada

Utilitários para processamento de XML.

Documentação das funções

xml: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:

<i>xml</i>	String contendo o XML.
------------	------------------------

Retorna:

`true` Tags estão balanceadas.
`false` Tags não estão balanceadas.
Definido na linha 21 do arquivo xml.cpp.
Referenciado por main().

extract::xml::extract::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:

<i>origin</i>	String original.
<i>open</i>	Delimitador de abertura.
<i>close</i>	Delimitador de fechamento.
<i>from</i>	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 <code>std::string::npos</code> .

Retorna:

std::string String extraída (sem os delimitadores), vazia quando nada for encontrado.
Definido na linha 62 do arquivo xml.cpp.
Referenciado por `extract()` e `main()`.

extract::xml::extract::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:

<i>origin</i>	String original.
<i>open</i>	Delimitador de abertura.
<i>close</i>	Delimitador de fechamento.

Retorna:

std::string String extraída (sem os delimitadores), vazia quando nada for encontrado.
Definido na linha 79 do arquivo xml.cpp.

Documentação da classe

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

Referência à classe `Template structures::LinkedList< T >`

`structures::LinkedList< T >`
Fila Encadeada.

Membros públicos

- ∞ [`~LinkedList\(\)`](#)
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.

Descrição detalhada

template<typename T>

class structures::LinkedList< T >

Fila Encadeada.

Definido na linha 23 do arquivo linked_queue.h.

Documentação dos Construtores & Destrutor

**~LinkedList:structures::LinkedList< T >structures::LinkedList< T
>::~LinkedListtemplate<typename T > LinkedList::~LinkedList ()**

Destrutor.

Definido na linha 14 do arquivo linked_queue.h.

Documentação dos métodos

**back:structures::LinkedList< T >structures::LinkedList< T
>::backtemplate<typename T > T & LinkedList::back () const**

Acessa o último da Fila.

Definido na linha 68 do arquivo linked_queue.h.

**clear:structures::LinkedList< T >structures::LinkedList< T >::cleartemplate<typename
T > void LinkedList::clear ()**

Limpa a Fila.

Definido na linha 19 do arquivo linked_queue.h.

**dequeue:structures::LinkedList< T >structures::LinkedList< T
>::dequeuetemplate<typename T > T LinkedList::dequeue ()**

Desenfileira.

Definido na linha 34 do arquivo linked_queue.h.

Referenciado por math::count_shapes().

**empty:structures::LinkedList< T >structures::LinkedList< T
>::emptytemplate<typename T > bool LinkedList::empty () const**

Confere se a Fila está vazia.

Definido na linha 76 do arquivo linked_queue.h.

Referenciado por math::count_shapes().

**enqueue:structures::LinkedList< T >structures::LinkedList< T
>::enqueuetemplate<typename T > void LinkedList::enqueue (const T & data)**

Enfileira.

Definido na linha 25 do arquivo linked_queue.h.

Referenciado por math::count_shapes().

front:structures::LinkedList< T >structures::LinkedList< T >:fronttemplate<typename T > T & LinkedList::front () const

Acessa a frente da Fila.

Definido na linha 60 do arquivo linked_queue.h.

size:structures::LinkedList< T >structures::LinkedList< T >:sizetemplate<typename T > std::size_t LinkedList::size () const

Retorna o tamanho da Fila.

Definido na linha 81 do arquivo linked_queue.h.

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

- 00 [linked_queue.h](#)
- 00 [linked_queue.inc](#)

Referência à classe `Template structures::LinkedList< T >`

`structures::LinkedList< T >``structures::LinkedList< T >`

Pilha Encadeada.

Membros públicos

∞ [`~LinkedList\(\)`](#)

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.

Descrição detalhada

`template<typename T>`

`class structures::LinkedList< T >`

Pilha Encadeada.

Definido na linha 23 do arquivo `linked_stack.h`.

Documentação dos Construtores & Destrutor

**~LinkedList:structures::LinkedList< T >structures::LinkedList< T
>::~~LinkedListtemplate<typename T > LinkedList::~~LinkedList ()**

Destrutor.

Definido na linha 14 do arquivo linked_stack.h.

Documentação dos métodos

**clear:structures::LinkedList< T >structures::LinkedList< T
>::cleartemplate<typename T > void LinkedList::clear ()**

Limpa a Pilha.

Definido na linha 19 do arquivo linked_stack.h.

**empty:structures::LinkedList< T >structures::LinkedList< T
>::emptytemplate<typename T > bool LinkedList::empty () const**

Confere se a Pilha está vazia.

Definido na linha 58 do arquivo linked_stack.h.

Referenciado por xml::balanced().

**pop:structures::LinkedList< T >structures::LinkedList< T >::poptemplate<typename T >
T LinkedList::pop ()**

Desempilha.

Definido na linha 31 do arquivo linked_stack.h.

Referenciado por xml::balanced().

**push:structures::LinkedList< T >structures::LinkedList< T >::pushtemplate<typename
T > void LinkedList::push (const T & data)**

Empilha.

Definido na linha 25 do arquivo linked_stack.h.

Referenciado por xml::balanced().

**size:structures::LinkedList< T >structures::LinkedList< T >::sizetemplate<typename T >
std::size_t LinkedList::size () const**

Retorna o tamanho da Pilha.

Definido na linha 63 do arquivo linked_stack.h.

**top:structures::LinkedList< T >structures::LinkedList< T >:toptemplate<typename T >
T & LinkedList::top () const**

Acessa o topo da Pilha.

Definido na linha 50 do arquivo linked_stack.h.

Referenciado por xml::balanced().

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

- ∞ [linked_stack.h](#)
- ∞ [linked_stack.inc](#)

Documentação do arquivo

Lista de todos os arquivos com uma breve descrição:

Referência ao arquivo `linked_queue.h`

`linked_queue.h``linked_queue.h`

Arquivo de declarações e interface da Fila Encadeada.

Componentes

∞ `class structures::LinkedQueue< T >`
Fila Encadeada.

Namespaces

∞ `structures`
Estruturas de Dados.

Descrição detalhada

Arquivo de declarações e interface da Fila Encadeada.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo linked_queue.inc

Linked_queue.inclinked_queue.inc

Implementações da Fila Encadeada.

Descrição detalhada

Implementações da Fila Encadeada.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo linked_stack.h

linked_stack.hlinked_stack.h

Declarações e interface da Pilha Encadeada.

Componentes

∞ class [structures::LinkedStack< T >](#)
Pilha Encadeada.

Namespaces

∞ [structures](#)
Estruturas de Dados.

Descrição detalhada

Declarações e interface da Pilha Encadeada.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo linked_stack.inc

linked_stack.inclinked_stack.inc
Implementações da Pilha Encadeada.

Descrição detalhada

Implementações da Pilha Encadeada.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo main.cpp

main..cpp

Código do programa principal.

Funções

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

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

∞ int [main](#) ()

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

Descrição detalhada

Código do programa principal.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Documentação das funções

main:main.cppmain.cpp:mainint 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 possível abrir o arquivo lido; -1 quando o XML lido é inválido; -2 quando alguma das imagens apresenta dimensões inválidas.

Definido na linha 44 do arquivo main.cpp.

**matrix_init:main.cppmain.cpp:matrix_initstatic int ** matrix_init (int *height*, int *width*,
const std::string & *data*)[static]**

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

Parâmetros:

<i>height</i>	Número de linhas da matriz.
<i>width</i>	Número de colunas da matriz.
<i>data</i>	String contendo os valores colocados na matriz. Whitespace é ignorado.

Retorna:

int** Matriz gerada. Deve ser destruído com [matrix_destroy\(\)](#) para liberar a memória alocada.
Definido na linha 95 do arquivo main.cpp.
Referenciado por main().

Referência ao arquivo matrix.cpp

matrix.cpp

Implementação das funções de processamento de matrizes.

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.

Descrição detalhada

Implementação das funções de processamento de matrizes.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo matrix.h

matrix.h

Declarações das funções de processamento de matrizes.

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.

Descrição detalhada

Declarações das funções de processamento de matrizes.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo xml.cpp

xml.cpp

Implementação das funções de processamento de XML.

Namespaces

∞ [xml](#)

Utilitários para processamento de XML.

Funções

∞ bool [xml::balanced](#) (const std::string &xml)

Confere a validade 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.

Descrição detalhada

Implementação das funções de processamento de XML.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Referência ao arquivo xml.h

xml.h

Declarações das funções de processamento de XML.

Namespaces

∞ [xml](#)

Utilitários para processamento de XML.

Funções

∞ bool [xml::balanced](#) (const std::string &xml)

Confere a validade 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.

Descrição detalhada

Declarações das funções de processamento de XML.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019