Trabalho de Implementação I

-

Processamento de XML com imagens binárias

Alek Frohlich e Gabriel B. Sant'Anna

Versão v1.0
19/05/2019

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.**

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:
  - O identificador é diferente (ou seja, uma marcação aberta deveria ter sido fechada antes);
  - 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.

Contagem de Componentes Conexos

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, 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** .

Conforme apresentado em aula, segue o algoritmo de rotulação (*labeling*) usando uma fila (**FIFO**):

- Inicializar `rótulo` com 1 e criar uma matriz `R` com o mesmo tamanho da matriz de entrada `E` lida.
- Enquanto varre 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 e 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 com intensidade **1** e ainda não tenha sido visitado (igual a **0** em `R`).
    - Na coordenada de cada vizinho selecionado, 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 dos namespaces

Lista de namespaces

Lista dos namespaces com uma breve descrição:

math (Código de natureza matemática) 4

structures (Estruturas de Dados) 5

xml (Utilitários para processamento de XML) 6

Índice dos componentes

Lista de componentes

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

structures::LinkedQueue< T > (Fila Encadeada) 7

structures::LinkedStack< T > (Pilha Encadeada) 9

Índice dos arquivos

Lista de arquivos

Lista de todos os arquivos com uma breve descrição:

linked_queue.h (Arquivo de declarações e interface da Fila Encadeada) 11

linked_queue.inc (Implementações da Fila Encadeada) 11

linked_stack.h (Declarações e interface da Pilha Encadeada) 12

linked_stack.inc (Implementações da Pilha Encadeada) 12

main.cpp (Código do programa principal) 13

matrix.cpp (Implementação das funções de processamento de matrizes) 14

matrix.h (Declarações das funções de processamento de matrizes) 14

xml.cpp (Implementação das funções de processamento de XML) 15

xml.h (Declarações das funções de processamento de XML) 15

Documentação dos namespaces

Referência ao namespace math

Código de natureza matemática.

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 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 arquivo matrix.cpp.

Referenciado por main().

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 arquivo matrix.cpp.

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 arquivo matrix.cpp.

Referência ao namespace structures

Estruturas de Dados.

Componentes

class LinkedQueue

Fila Encadeada.

class LinkedStack

Pilha Encadeada.

Referência ao namespace xml

Utilitários para processamento de XML.

Documentação das funções

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 arquivo xml.cpp.

Referenciado por main().

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 arquivo xml.cpp.

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 arquivo xml.cpp.

Documentação das classes

Referência à classe Template structures::LinkedQueue< T >

Fila Encadeada.
`#include <linked_queue.h>`

Membros públicos

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.

Documentação dos métodos

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

Acessa o último da Fila.

Definido na linha 68 do arquivo linked_queue.h.

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

Limpa a Fila.

Definido na linha 19 do arquivo linked_queue.h.

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

Desenfileira.

Definido na linha 34 do arquivo linked_queue.h.

Referenciado por math::count_shapes().

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

Confere se a Fila está vazia.

Definido na linha 76 do arquivo linked_queue.h.

Referenciado por math::count_shapes().

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

Enfileira.

Definido na linha 25 do arquivo linked_queue.h.

Referenciado por math::count_shapes().

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

Acessa a frente da Fila.

Definido na linha 60 do arquivo linked_queue.h.

template<typename T > std::size_t LinkedQueue::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:

linked_queue.h
linked_queue.inc

Referência à classe Template structures::LinkedStack< T >

Pilha Encadeada.
`#include <linked_stack.h>`

Membros públicos

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.

Documentação dos métodos

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

Limpa a Pilha.
Definido na linha 19 do arquivo linked_stack.h.

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

Confere se a Pilha está vazia.
Definido na linha 58 do arquivo linked_stack.h.
Referenciado por xml::balanced().

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

Desempilha.
Definido na linha 31 do arquivo linked_stack.h.
Referenciado por xml::balanced().

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

Empilha.

Definido na linha 25 do arquivo linked_stack.h.

Referenciado por xml::balanced().

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

Retorna o tamanho da Pilha.

Definido na linha 63 do arquivo linked_stack.h.

template<typename T > T & LinkedStack::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 dos arquivos

Referência ao arquivo linked_queue.h

Arquivo de declarações e interface da Fila Encadeada.

Componentes

class structures::LinkedQueue< T >
Fila Encadeada.

Namespaces

structures
Estruturas de Dados.

Autor:
Alek Frohlich, Gabriel B. Sant'Anna

Versão:
1.0

Data:
2019-05-19

Copyright:
Copyright (c) 2019

Referência ao arquivo linked_queue.inc

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

Declarações e interface da Pilha Encadeada.

Componentes

class structures::LinkedStack< T >
Pilha Encadeada.

Namespaces

structures
Estruturas de Dados.

Autor:
Alek Frohlich, Gabriel B. Sant'Anna

Versão:
1.0

Data:
2019-05-19

Copyright:
Copyright (c) 2019

Referência ao arquivo linked_stack.inc

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

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.

Autor:

Alek Frohlich, Gabriel B. Sant'Anna

Versão:

1.0

Data:

2019-05-19

Copyright:

Copyright (c) 2019

Documentação das funções

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 44 do arquivo main.cpp.

static 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:

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 95 do arquivo main.cpp.

Referência ao arquivo 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.

Autor:
Alek Frohlich, Gabriel B. Sant'Anna

Versão:
1.0

Data:
2019-05-19

Copyright:
Copyright (c) 2019

Referência ao arquivo matrix.cpp

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 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 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.

Autor:
Alek Frohlich, Gabriel B. Sant'Anna

Versão:
1.0

Data:
2019-05-19

Copyright:

Referência ao arquivo xml.cpp

Implementação das funções de processamento de XML.

Autor:
Alek Frohlich, Gabriel B. Sant'Anna

Versão:
1.0

Data:
2019-05-19

Copyright: