Universidade Federal de Santa Catarina
INE5408-03208A (20221) - Estruturas de Dados
Aluna: Samantha Costa de Sousa (21106156)

Relatório do Projeto I - processamento de XML com imagens binárias

Para concluir o projeto, utilizei as estruturas de dados pilha e fila (ArrayStack e ArrayQueue), implementadas anteriormente nos VPL's da disciplina. Os algoritmos utilizados na resolução do problema também são baseados na explicação do professor da disciplina durante as aulas e no moodle. Todos os testes passaram com sucesso.

Na **main()**, o que acontece primeiramente é a criação de uma variável *file* do tipo "ifstream" em seguida a leitura do nome do arquivo que será lido e a abertura desse arquivo utilizando *file*. Então, com uso de ostringstream() e demais funções das bibliotecas fstream/sstream, os conteúdos do arquivo lido vão para uma variável do tipo "string" e o arquivo é fechado. A partir daí, começa a primeira validação

Primeiro problema: validação de arquivo XML

Na main(), a função **validate()** é chamada entregando como parâmetro file_contents, que é a string com o conteúdo do arquivo xml.

```
/* Deve retornar falso se abertura+fechamento de tags não estiver correto *
bool validate(string file_contents)
{
```

Essa função

- Cria uma variável do tipo pilha *stack* do tamanho *file_contents.length()* de elementos string e depois disso começa a iterar (com um while que dura enquanto i é menor que *file_contents.length()*) por todos caracteres do arquivo xml.
- Ao encontrar um caractere "<" ela salva sua posição em *tag_start* (começa a abertura da tag)
- Ao encontrar um caractere ">" ela salva sua posição em tag_end, reconhecendo que uma tag acabou de ser fechada.
- Nessa condição da tag ter sido fechada, faz um slice de file_contents para adquirir a tag (logo depois de < e logo antes de />, utilizando a função substr() de string)
- Se tag[0] não é "/", significa que é uma tag de abertura, logo, push na stack.
- Se tag[0] é "/", sendo uma tag de fechamento, a "/" da tag é apagada e a seguinte verificação é feita:
    - Se a pilha está vazia -> Uma tag foi fechada sem ser aberta, retorna **false**.
    - Se a tag é a igual a stack.top() -> Tudo correto, faz stack.pop(), a tag abriu e fechou adequadamente então a verificação pode ser terminada.
    - Se a tag não é igual a stack.top() -> Eu fechei uma tag que não foi a última a ser aberta: Quebra no aninhamento. Retorna **false**.
- Continua o while até acabar o arquivo ou encontrar um erro. Caso passe do while, stack.empty() é retornado, ou seja, se ainda há tag aberta é retornado **false** (erro) e caso não **true** (passa a verificação feita na main)
- Caso aconteça o erro, a main() dá cout "error" e retorna -1, finalzando a execução do programa.

Passada a verificação com sucesso, agora começa a resolução do segundo problema.

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

Antes de começar a solução, é importante ressaltar uma função criada para ajudar a resolução em vários pontos, a **get_tag_content()**

```
/* @param start_position a partir de onde começo a buscar no arquivo
   @param tag qual tag estou buscando o conteúdo
   @return o conteúdo da tag */
string get_tag_content(string file_contents, size_t start_position, string tag)
{
```

Com as informações dadas a ela, e utilizando funções de string (find, length, substr, etc) ela encontra onde começa o conteúdo da tag entregue a ela, encontra onde ele termina (logo antes da tag de fechamento), faz o slice e retorna uma string com somente o conteúdo dentro daquela tag.

De volta a main, variáveis são criadas para percorrer o arquivo adequadamente (começo a partir de 0 e termino um pouco antes de dataset, para não ter erros de procurar imagens novas quando não há mais nenhuma) *Cada iteração no loop while corresponde a uma imagem.

A partir de start_position (que é a variável da iteração), adquiro:

- Altura e largura da imagem com get_tag_content() de height e width, em int (utilizando stoi());
- Conteúdo da imagem (conteúdo dentro da tag <data> utilizando get_tag_content()), apagando todas as quebras de linha;
- Nome da imagem (get_tag_content() em <name>)
- Crio uma matriz com os tamanhos encontrados, onde cada uma de suas entradas é um caractere da imagem (data_content), passado a int com uma conversão de ASCII para número (fazendo -48)

```
/* Criando matriz com a imagem */
int **matriz_data_content = new int *[height];
int contador = 0;
for (int i = 0; i < height; i++)
{
    matriz_data_content[i] = new int[width];
    for (int j = 0; j < width; j++)
    {
        matriz_data_content[i][j] = data_content[contador] - 48; // converte char p int
        contador++;
    }
}
```

A partir daí, possuindo uma matriz que tem todas as entradas da imagem do arquivo xml corretamente, é necessário adquirir o número de labels quando é feita a rotulação, e uma função de rotulação é chamada:

```
int labeling_algorithm(int **matriz_imagem, int height, int width)
{
```

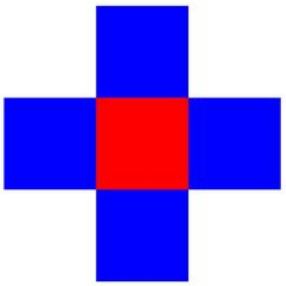
Essa função segue o algoritmo explicado no moodle, seguindo dessa forma:

- Uma matriz de zeros (que terá as labels) com mesma altura e largura da matriz_imagem é criada.
- Uma variável de label (que começa em 1), e uma fila *queue* de elementos tuple (para guardar coordenadas i, j) do tamanho height*width; Depois disso, começa o percurso da matriz_imagem.
- Ao encontrar um pixel de intensidade 1 que não foi rotulado, ele insere na fila e rotula ele com a label atual na matriz de zeros.
- A partir daí, começa uma checagem (enquanto a fila não está vazia) dos vizinhos do local pintado (vizinhança-4, ou seja, não conta diagonais).
- Removendo da fila, pegando as coordenadas de quem foi removido (primeiro vizinho, já que é FIFO) e checando seus vizinhos (inserindo na fila quando o “pixel” pertence a pintura, ou seja, é de intensidade 1 e não foi rotulado, e rotulando na outra matriz).
- Dessa forma, é garantido que o algoritmo checa os vizinhos de todos inseridos na fila, até eles acabarem (fila vazia). O algoritmo também possui condições para a checagem da vizinhança-4 não passar dos extremos da matriz (menor que 0, maior que sua dimensão)

```
/* Percorre matriz */
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        if (matriz_imagem[i][j] == 1 && matriz_zeros[i][j] == 0)
        {
            /* Encontrou pixel de intensidade 1 não visitado*/
            queue.enqueue(make_tuple(i, j)); // insiro na fila
            matriz_zeros[i][j] = label;      // atribuo o rótulo

            while (!queue.empty())
            {
                tuple<int, int> removed = queue.dequeue();
                int coord_i = get<0>(removed);
                int coord_j = get<1>(removed);
                // Primeiro check: não passar dos extremos
                // Segundo check: pixel de intensidade 1 NÃO ROTU

                if (coord_i - 1 >= 0)
                { // Vizinho 1
```



Vizinhos cardeais (4 pontos)

$$M(i-1,j)$$
$$M(i+1,j)$$
$$M(i,j-1)$$
$$M(i,j+1)$$

Terminando esse loop de *!queue.empty()*, a variação label é incrementada (próxima região encontrada será rotulada com outro número), e os loops continuam a percorrer a matriz em busca de regiões pintadas não rotuladas até ela acabar.

A função termina retornando a última label aplicada. Sabendo que as labels começaram em 1, o número de regiões encontradas é esse retorno - 1. Um cout é feito com o nome da imagem e o número de regiões encontradas.

O loop que percorre *file_contents* continua checando por outras imagens se houver alguma, pois start_position receberá a posição logo do fim da tag </img>, ou seja, a próxima iteração é toda feita em referência a depois da última imagem, já que todas as funções de busca usam start_position como referência de "início do arquivo". O loop termina quando as imagens do arquivo acabaram (cheguei na última tag de </img>. O programa finaliza com sucesso.

Conclusão (dificuldades e referências)

As partes mais complicadas foram, 1. Utilização do C++, visto que essa disciplina é minha primeira experiência com a linguagem e não possuo conhecimento de todas as necessidades dela, sintaxe, bibliotecas comuns etc. Os VPL's me familiarizaram um pouco com a sintaxe e ponteiros (que também comecei aprender neste semestre nesta disciplina e na de Programação Concorrente, que utiliza C), mas o projeto foi bem mais desafiante já que não possuía uma estrutura pronta. Optei por fazer uma solução simples, utilizando funções e não criando novas classes, para não complicar meu entendimento das coisas. 2. Operações no arquivo XML. Foram diversas pesquisas e testes para entender as bibliotecas e funções que eu teria que usar, se eu estava utilizando elas corretamente, etc.

A utilização das Estruturas de Dados foi a parte mais simples do projeto, pois eu já possuía as implementações e os algoritmos, tais o motivo de utilizar essas estruturas, foram tranquilos. O algoritmo do segundo problema é bem mais complicado, mas após começar a escrever o código, minhas dúvidas foram sanadas (por exemplo, as instruções diziam "inserir (x,y) na fila" mas aí pensei que era o valor na matriz, mas não fazia sentido para conseguir percorrer, então entendi que o que eu deveria salvar eram as coordenadas.

Alguns sites usados (também utilizei a documentação que estava no moodle e pesquisei dúvidas de erros de sintaxe algumas vezes, normalmente sendo sanadas no stackoverflow):

https://www.bgsu.edu/arts-and-sciences/computer-science/cs-documentation/reading-data-from-files-using-c-plus-plus.html

https://www.delftstack.com/pt/howto/cpp/read-file-into-string-cpp/

https://stackoverflow.com/questions/33434030/c-how-to-store-two-data-types-in-a-vector

https://en.cppreference.com/w/cpp/utility/tuple

Fonte da imagem sobre vizinhança-4: https://slideplayer.com.br/slide/13023413/ (slide sobre processamento de imagens encontrado no google)