Projeto I - relatório
</Verificação de Cenários e Determinação de Área Limpa por um Robô Aspirador>
Nomes: Jessica Regina dos Santos e Myllena da Conceicao Correa
Matrículas: 22100626 e 22104061
Semestre: 2023.2

</Resumo/Introdução>
Nosso relatório descreve a implementação de um programa em C + + que processa arquivos XML contendo matrizes binárias que representam cenários de ação de um robô aspirador.
Nosso programa resolve dois problemas propostos pelos professores:

1. A validação de arquivo XML e determinação das áreas que o robô deve limpar. Para a validação desses arquivos são utilizados algoritmos baseado em pilha (LIFO) para verificar o aninhamento e o fechamento correto das tags XML.
2. A determinação das áreas a serem limpas pelo robô, com base na identificação de componentes conexos em matrizes binárias. A implementação utiliza estruturas lineares, como pilhas e filas, demonstrando a lógica envolvida em cada parte do código.

</Primeiro problema: Validação de Arquivo XML>
Para validar o arquivo XML, implementamos um algoritmo baseado em pilha (LIFO) para verificar o aninhamento e o fechamento correto das tags XML.
O código começa com a leitura do arquivo XML e a utilização de uma pilha de strings para rastrear as tags abertas.
Abaixo, pontuamos as partes mais essenciais do código em ordem de execução:

- ➔ A inicialização de uma pilha de strings (stack<string> pilha) para rastrear as tags XML abertas;

```
61 int main() {
62     stack<string> pilha; // cria uma pilha de strings
```

- ➔ A leitura do nome do arquivo XML de entrada (cin >> xmlfilename);

```
64     char xmlfilename[100]; // para armazenar o nome do arquivo xml
65
66     cin >> xmlfilename;  // entrada
```

- ➔ A abertura do arquivo XML para leitura (arquivo.open(xmlfilename));

```
68      ifstream arquivo; // cria uma variável do tipo ifstream
69      string linha; // uma string para armazenar a linha
70      arquivo.open(xmlfilename); // abrindo o arquivo
```

➔ A leitura do arquivo linha a linha e processamento de cada caractere;

```
72      if (arquivo.is_open()) { // if para saber se o arquivo es
73          while (getline(arquivo, linha)) { // enquanto consegu
74              size_t tamanho = linha.size(); // pegamos o taman
75              for (size_t i = 0; i < tamanho; i++) { // for par
76                  string forPilha = ""; // uma variável para co
77                  if (linha[i] == '<' && linha[i + 1] != '/') {
78                      i++; // para tirar o '<'
79                      while (linha[i] != '>') { // enquanto o c
80                          if (!isspace(linha[i])) { // se for d
81                              forPilha.push_back(linha[i]); //
82                          }
83                          i++; // incrementa o índice
84                      }
85                      pilha.push(forPilha); // adicionamos à pi
86                      forPilha = ""; // "limpamos" a string
87                  }
```

➔ <u>Lógica principal do algoritmo</u>: Consiste na verificação de abertura e fechamento de tags XML. Ao encontrar uma marcação de abertura, empilhamos o identificador da tag. Ao encontrar uma marcação de fechamento, verificamos se o topo da pilha contém o mesmo identificador. Caso contrário, imprimimos "erro" e encerramos o programa.

```
89                  if (linha[i] == '<' && linha[i + 1] == '/') {
90                      i += 2;
91                      while (linha[i] != '>') {
92                          if (!isspace(linha[i])) {
93                              forPilha.push_back(linha[i]);
94                          }
95                          i++;
96                      }
97                      if (forPilha == pilha.top()) {
98                          //cout << pilha.top() << endl;
99                          pilha.pop();
100                     } else {
101                         cout << "erro" << endl;
102                         exit(1);
103                     }
104                 }
105             }
106         }
```

```
108         if (!pilha.empty()) {
109             cout << "erro" << endl;
110             exit(1);
111         }
```

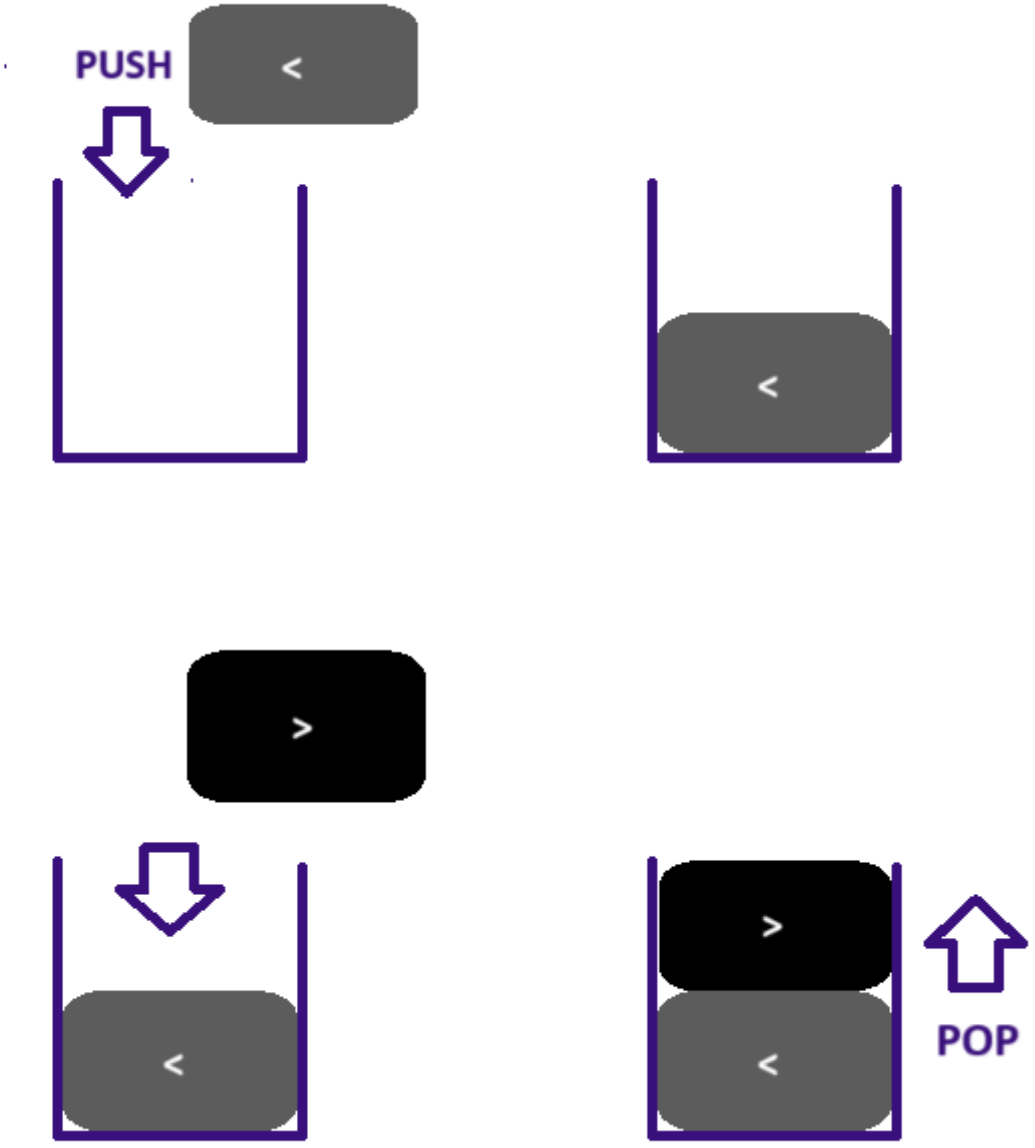
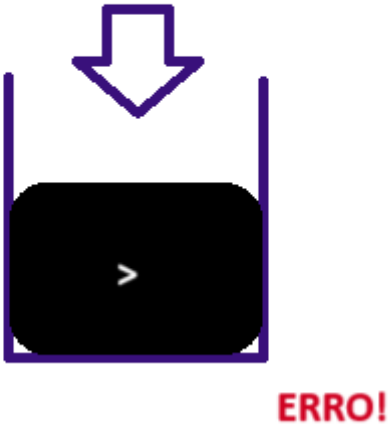
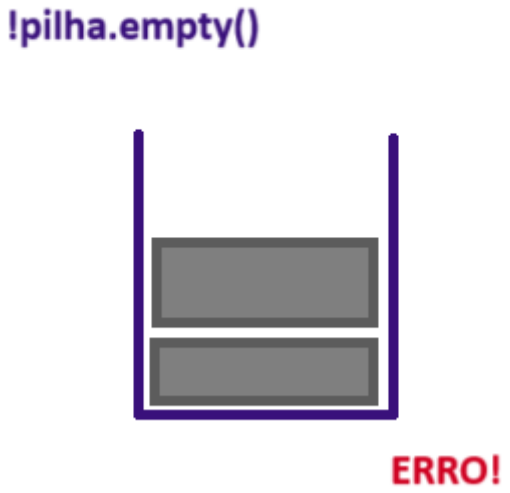


Figura 1.1: Exemplo de Validação de Arquivo XML.



ERRO!

Figura 1.2: Exemplo de Tag de Fechamento sem Tag de Abertura



ERRO!

Figura 1.3: Exemplo Pilha não vazia no final da execução do programa

</Segundo problema: Algoritmo de Determinação de Área do Espaço a Ser Limpo>

O segundo problema envolve a determinação da área que o robô deve limpar com base na identificação de componentes conexos na matriz binária.

O algoritmo de reconstrução de componente conexo é implementado usando uma fila (FIFO). Abaixo, explicamos as principais etapas do algoritmo:

- ➔ A inicialização da matriz R de zeros, com o mesmo tamanho da matriz de entrada;

```
184                 } else if (forPilha == "matriz") {
185                     // Encontrou a tag <matriz>, indica que estamos dentro da seção <matriz>
186                     isMatrizSection = true;
187                     int **M = new int*[Nlinha];  // Aloca dinamicamente a matriz M
188                     int **R = new int*[Nlinha];  // Aloca dinamicamente a matriz R
189 
190                     for (int i = 0; i < Nlinha; ++i) {
191                         M[i] = new int[Ncoluna];
192                         R[i] = new int[Ncoluna];
193                     }
194 
195                     // Lê e preenche as matrizes M e R com os valores da linha
196                     for (int i = 0; i < Nlinha; ++i) {
197                         getline(arquivo, linha);
198                         for (int j = 0; j < Ncoluna; ++j) {
199                             if (linha[j] != '<'){
200                                 M[i][j] = linha[j] - '0';
201                                 R[i][j] = 0;
202                             } else {
203                                 break;
```

➔ A inserção da coordenada (x, y) do robô na fila e marcação dessa coordenada como 1 em R.

➔ <u>Enquanto a fila não estiver vazia:</u>

- Remoção de um par de coordenadas (x, y) da fila.
- Verificação dos quatro vizinhos que estão dentro dos limites da matriz, têm valor 1 (na matriz de entrada) e ainda não foram visitados (valor 0 em R).

```
while (!fila.empty()) {
    pair<int, int> temp = fila.front();
    fila.pop();

    int l = temp.first;
    int c = temp.second;

    if (l >= 0 && l < Nlinha && c >= 0 && c < Ncoluna) {
        if (M[l][c] == 1 && R[l][c] == 0) {
            R[l][c] = 1;
            //cout << "Coloquei 1 na posição (" << l << ", " << c << ")" <<

            fila.push(make_pair(l - 1, c)); // Vizinho acima
            fila.push(make_pair(l + 1, c)); // Vizinho abaixo
            fila.push(make_pair(l, c - 1)); // Vizinho à esquerda
            fila.push(make_pair(l, c + 1)); // Vizinho à direita
        }
    }
}
```

- Marcação desses vizinhos como 1 em R.
- Contagem dos elementos marcados como 1 na matriz R para determinar a área do componente conexo.

```
int contador = 0;
for (int i = 0; i < Nlinha; ++i) {
    for (int j = 0; j < Ncoluna; ++j) {
        if (R[i][j] == 1) {
            contador+= 1;
        }
    }
}
cout << nome << " " << contador << endl;
return;
```

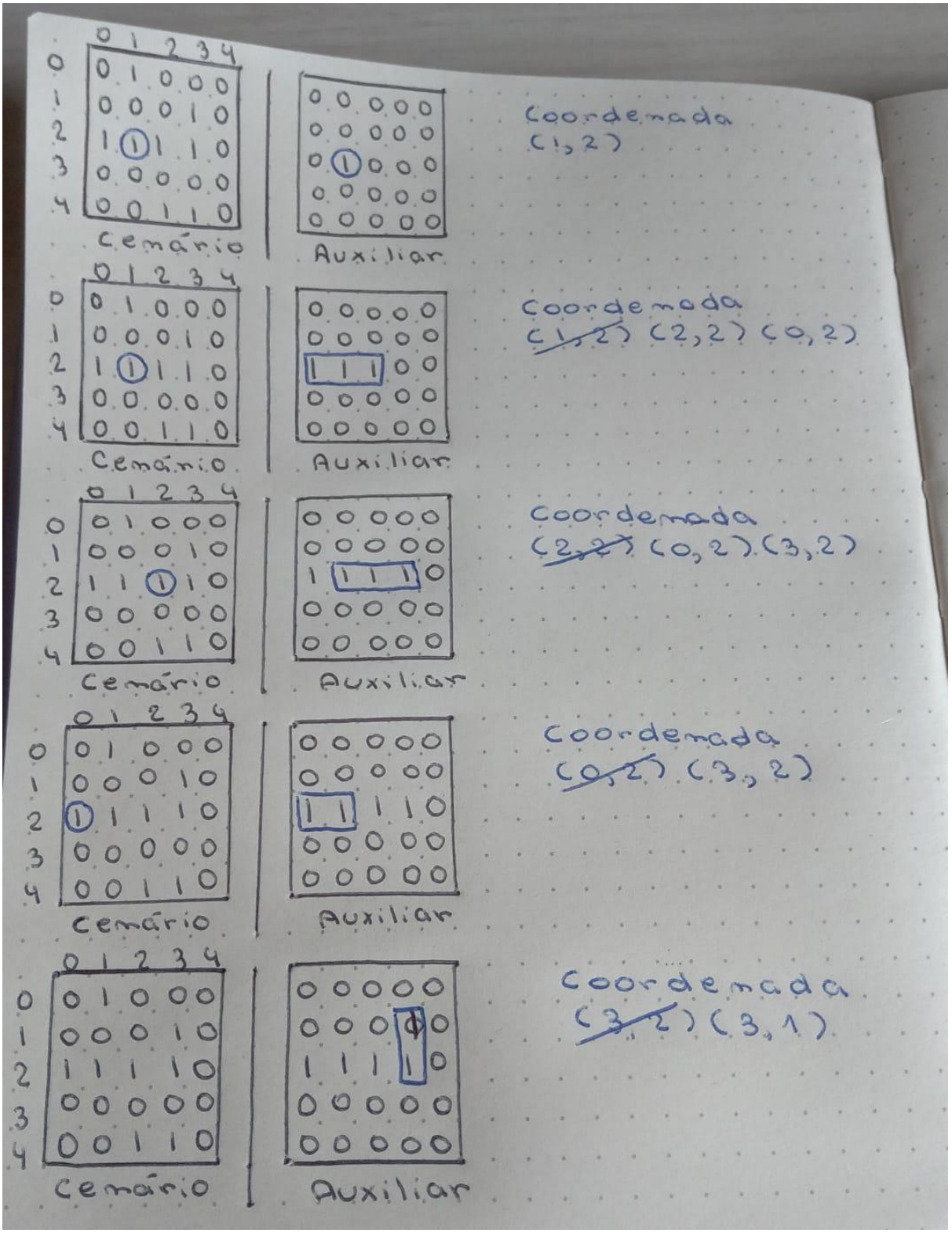


Figura 2: Exemplo de Determinação de Área a Ser Limpa pelo Robô.

</Dificuldades no Desenvolvimento do Projeto>

Durante o desenvolvimento do projeto, enfrentamos algumas dificuldades, como:

- A compreensão do formato XML: A interpretação do formato XML e a extração das informações necessárias dos arquivos foram desafiadoras inicialmente;
- A compreensão e alocação dinâmica de matrizes: A compreensão e alocação dinâmica de matrizes para armazenar as informações das matrizes binárias exigiu um cuidado e estudo extra;
- Implementação dos algoritmos: A implementação dos algoritmos para validação de XML e determinação da área do espaço a ser limpa exigiu entendimento sólido das estruturas de dados estudadas ao longo do semestre e lógica de programação;
- Testes e depuração: A verificação da lógica e a depuração do código para lidar com diferentes cenários de entrada foram desafios adicionais, assim como desenvolver em sistema Linux.

</Referências>

Durante o desenvolvimento do projeto, algumas referências foram consultadas:

1. O próprio material da disciplina INE5408, da sala/laboratórios/Moodle;
2. Documentação da linguagem C++: http://www.cplusplus.com/doc/tutorial/files/
3. Alocação dinâmica de matrizes: https://www.inf.ufpr.br/roberto/ci067/14_alocmat.html

</Conclusões>

O Projeto I abordou a resolução de dois problemas relacionados ao processamento de arquivos XML que representam cenários de ação de um robô aspirador.

A implementação foi realizada em C++ e utilizou estruturas lineares, pilhas e filas, para validar o arquivo XML e determinar a área que o robô deve limpar.

O algoritmo de validação baseado em pilha garante que as tags XML estejam corretamente aninhadas e fechadas, enquanto o algoritmo de determinação da área utiliza uma fila para encontrar componentes conexos.

O projeto demonstra a aplicação de conceitos de estruturas de dados e algoritmos em situações do mundo real, proporcionando uma experiência prática na resolução de problemas de processamento de dados. Partes do código-fonte, da lógica de implementação e figuras ilustrativas foram fornecidos neste relatório para facilitar a compreensão do processo.