UNIVERSIDADE FEDERAL DE SANTA CATARINA

CAMPUS TRINDADE

INE - DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

INE5408 - ESTRUTURAS DE DADOS

Alunos: João Victor Cabral Machado e Pedro Alfeu Wolff Lemos

22204113 **22200373**

Relatório do Projeto 1

Florianópolis

2023

Sumário

1. Introdução ao problema

O projeto 1 aborda a aplicação de conceitos de estruturas lineares na resolução/processamento de arquivos XML relacionadas à operação de um robô aspirador. Os problemas a serem resolvidos neste projeto envolvem a determinação da área que pode ser limpa pelo robô aspirador e a validação da área que foi limpa em um ambiente.

Para a realização do projeto utilizamos as estruturas de dados, fila e pilha, para representar e manipular os dados de cenários de ação do robô para organizarmos elas de maneira fácil e eficiente. As ações do robô foram dadas pelos arquivos XML, que contém matrizes binárias que representam as ações do robô.

O problema 1 envolve a verificação de aninhadas e fechamento correto das tags do arquivo XML. O objetivo do problema é garantir que as tags sejam fechadas sem terem sido previamente abertas e que todas as marcações estejam devidamente aninhadas.

O segundo problema envolve a determinação da área que o robô precisa limpar em um ambiente que é determinado por uma matriz dada pelo arquivo XML. A matriz é formada por vários 0 e 1, 0 indicando que a área não precisa ser limpa e 1 indicando que a área precisa ser limpa.

2. Algoritmos

    2.1 Algoritmos Do Primeiro Exercício

Primeiro criamos uma estrutura que tem como papel armazenar o resultado da verificação.

```
// Estrutura para armazenar o resultado da verificação XML
struct ResultadoVerificacao {
    queue<string> fila;    // Fila de linhas do XML
    bool certoErrado;      // Indica se o XML está formatado corretamente
};
```

Ela tem dois atributos, o primeiro uma fila de strings e o segundo um booleano que vai verificar se o arquivo XML vai ser corretamente analisado ou não.

```cpp
// Algoritmo 1
// Função para verificar a validade do arquivo XML
ResultadoVerificacao VerificarXML(const string& xmlFilename) {
    ResultadoVerificacao resultado;
    string linha;
    queue<string> fila;
    stack<string> pilhaVer;

    ifstream arquivo(xmlFilename);
    if (!arquivo.is_open()) {
        resultado.certoErrado = false; // O arquivo não pôde ser aberto
        return resultado;
    }

    while (getline(arquivo, linha)) {
        fila.push(linha);

        for (int i = 0; i < linha.length(); i++) {
            if (linha[i] == '<' && linha[i + 1] != '/') {
                string palavra = "";
                int j = 1;
                while (linha[i + j] != '>') {
                    palavra += linha[j + i];
                    j++;
                }
                pilhaVer.push(palavra);
            }
            else if (linha[i] == '<' && linha[i + 1] == '/') {
                string palavra = "";
                int j = 2;
                while (linha[i + j] != '>') {
                    palavra += linha[i + j];
                    j++;
                }
                if (!pilhaVer.empty()) {
                    if (pilhaVer.top() == palavra) {
                        pilhaVer.pop();
                    }
                    else {
                        resultado.certoErrado = false;  // As tags não correspondem
                        resultado.fila = fila;
                        return resultado;
                    }
                }
            }
        }
    }
    arquivo.close();

    if (!pilhaVer.empty()) {
        resultado.certoErrado = false;  // Tags não fechadas adequadamente
        resultado.fila = fila;
    }
    else {
        resultado.certoErrado = true;  // XML válido
        resultado.fila = fila;
    }

    return resultado;
}
```

A função VerificarXML é o ponto inicial do exercício 1. Ela é responsável por verificar e validar o arquivo XML. Ela retorna uma estrutura ResultadoVerificacao, que vai carregar o resultado. Ela começa declarando uma fila para armazenar as linhas do XML e uma pilha para rastrear todas as tags XML.

A função lê o arquivo XML linha por linha usando um loop, adicionando cada linha à fila, também dentro desse loop há um for aninhando que analisa cada caractere na linha para ver as tags XML. Classificando ela em tags de abertura e tags de fechamento.

Para as tags de abertura, ele vai extrair o nome da tag e vai empilhar ele na pilha pilhaVer. E para as tags de fechamento, ele vai extrair o nome da tag e verificar se corresponde à última tag na pilha. Caso as tags não correspondam, a função define o booleano como false, armazena a fila no resultado e retorna.

Após isso, a função verifica se a pilha ainda tem elementos dentro, se ela estiver vazia, isso indica que todas as tags foram fechadas corretamente e a função define o resultado.certoErrado como true, finalizando o programa. Caso a função ainda não esteja vazia, isso indica que algumas das tags não foram fechadas, e a função define o resultado.certoErrado como false, e armazena a fila no resultado. Se o booleano do resultado der true, a função vai retornar o objeto resultado com as informações sobre a verificação do arquivo XML.

Em conclusão, o primeiro exercício do projeto era criar uma função que vai verificar a formatação das tags em um documento XML e fornecer o feedback sobre sua validade. Caso o XML estiver formatado corretamente, o resultado.certoErrado será true, caso contrário ele será false, e a fila poderá ser usada para depurar e identificar os problemas no arquivo.

2. Algoritmos

2.2 Algoritmos Do Segundo Exercício

Começamos o exercício 2 criando uma estrutura que vai ser encarregada de armazenar as informações do cenário. Ela possui cinco atributos: O nome do cenário, da área, altura, largura, coordenada x, coordenada y.

```
// Estrutura para armazenar informações de um cenário
struct InformacoesCenario {
    string nomeCenario;    // Nome do cenário
    int area;              // Área do cenário
    int altura;            // Altura do cenário
    int largura;           // Largura do cenário
    int x;                 // Coordenada X do robô
    int y;                 // Coordenada Y do robô
};
```

```cpp
int convertStringToInt(const string& str) { // Função para converter uma string em um inteiro
    try {
        return stoi(str);
    }
    catch (const invalid_argument& e) {
        throw runtime_error("erro");
    }
}

// Funções para extrair informações do XML
void PegarNomeCenario(queue<string>& fila, string& nomeCenario) {
    string linha;
    size_t pos;

    while (!fila.empty()) {
        linha = fila.front();

        if (linha.find("<nome>") != string::npos) {
            pos = linha.find("<nome>") + 6;
            nomeCenario = linha.substr(pos, linha.find("</n", pos) - pos);
            return;
        }
        else if (linha == "</cenario>") {
            break;
        }
    }
}

void PegarAlturaCenario(queue<string>& fila, int& altura) {
    string linha;
    size_t pos;

    while (!fila.empty()) {

        linha = fila.front();
        if (linha.find("<altura>") != string::npos) {
            pos = linha.find("<altura>") + 8;
            altura = convertStringToInt(linha.substr(pos, linha.find("</a", pos) - pos));
            return;
        } else if (linha == "</dimensoes>") {
            break;
        }
    }
}
```

```cpp
void PegarLarguraCenario(queue<string>& fila, int& largura) {
    string linha;
    size_t pos;

    while (!fila.empty()) {
        linha = fila.front();
        if (linha.find("<largura>") != string::npos) {
            pos = linha.find("<largura>") + 9;
            largura = convertStringToInt(linha.substr(pos, linha.find("</l", pos) - pos));
            return;
        }
        else if (linha == "</dimensoes>") {
            break;
        }
    }
}

void PegarCoordenadaX(queue<string>& fila, int& x) {
    string linha;
    size_t pos;

    while (!fila.empty()) {

        linha = fila.front();
        if (linha.find("<x>") != string::npos) {
            pos = linha.find("<x>") + 3;
            x = convertStringToInt(linha.substr(pos, linha.find("</x", pos) - pos));
            return;
        } else if (linha == "</robo>") {
            break;
        }
    }
}

void PegarCoordenadaY(queue<string>& fila, int& y) {
    string linha;
    size_t pos;

    while (!fila.empty()) {
        linha = fila.front();

        if (linha.find("<y>") != string::npos) {
            pos = linha.find("<y>") + 3;
            y = convertStringToInt(linha.substr(pos, linha.find("</y", pos) - pos));
            return;
        }
        else if (linha == "</robo>") {
            break;
        }
    }
}
```

```cpp
// Função para criar uma matriz vazia com a altura e largura fornecidas
vector<vector<char>> CriarMatriz(int altura, int largura) {
    return vector<vector<char>>(altura, vector<char>(largura, '0'));
}

// Função para calcular a área de uma matriz com base nas coordenadas X e Y do robô
int CalcularAreaCenario(vector<vector<char>>& matriz, int x, int y) {
    if (matriz[x][y] != '0') {
        int altura = matriz.size();
        int largura = matriz[0].size();
        queue<pair<int, int>> coordenadas;
        coordenadas.push({ x, y });
        matriz[x][y] = '2'; // Marcar a posição inicial como visitada

        int area = 1; // Área inicial é 1, pois estamos começando de uma célula válida

        while (!coordenadas.empty()) {
            int i = coordenadas.front().first;
            int j = coordenadas.front().second;
            coordenadas.pop();

            // Verificar e adicionar células vizinhas válidas
            if (i + 1 < altura && matriz[i + 1][j] == '1') {
                coordenadas.push({ i + 1, j });
                matriz[i + 1][j] = '2';
                area++;
            }
            if (i - 1 >= 0 && matriz[i - 1][j] == '1') {
                coordenadas.push({ i - 1, j });
                matriz[i - 1][j] = '2';
                area++;
            }
            if (j + 1 < largura && matriz[i][j + 1] == '1') {
                coordenadas.push({ i, j + 1 });
                matriz[i][j + 1] = '2';
                area++;
            }
            if (j - 1 >= 0 && matriz[i][j - 1] == '1') {
                coordenadas.push({ i, j - 1 });
                matriz[i][j - 1] = '2';
                area++;
            }
        }

        return area;
    }

    return 0; // Se a célula de início for '0', a área é zero
}
```

```cpp
// Função para processar as informações de um cenário
InformacoesCenario ProcessarCenario(queue<string>& fila) {
    InformacoesCenario informacoes;
    informacoes.nomeCenario = "SemNome"; // Valor padrão se o nome não for encontrado
    while (fila.front() != "<cenario>") {
        fila.pop();
    }
    fila.pop();
    PegarNomeCenario(fila, informacoes.nomeCenario);
    fila.pop();
    PegarAlturaCenario(fila, informacoes.altura);
    PegarLarguraCenario(fila, informacoes.largura);
    fila.pop();
    PegarCoordenadaX(fila, informacoes.x);
    PegarCoordenadaY(fila, informacoes.y);
    fila.pop();

    vector<vector<char>> matriz = CriarMatriz(informacoes.altura, informacoes.largura);

    // Apagar linha coordenadas
    fila.pop();

    for (auto& linha : matriz) {
        string linhaMatriz = fila.front();
        for (size_t j = 0; j < informacoes.largura; j++) {
            linha[j] = linhaMatriz[j];
        }
        fila.pop();
    }

    informacoes.area = CalcularAreaCenario(matriz, informacoes.x, informacoes.y);
    return informacoes;
}
```

A primeira função do exercício 2, convertStingtoInt, recebe uma string e tenta convertê-la em um inteiro usando a função stoi. Caso a conversão falhar por causa de algum argumento inválido, ela lança uma exceção runtime_error.

As funções PegarNomeCenario, PegarAlturaCenario, PegarLarguraCenario, PegarCoordenadaX, PegarCoordenadaY, servem para extrairmos as informações necessárias para realizar as outras operações do código. Elas usam find para localizar as tags no arquivo XML e extraem os valores.

A função CriarMatriz serve para criarmos uma matriz vazia com as dimensões que extrairmos do arquivo XML com as funções PegarAlturaCenario e PegarLarguraCenario.

A função CalcularAreaCenario serve para calcular a área do cenário com as bases nas coordenadas x e y do robô. Ela usa uma busca em largura para contar o número de casas com o valor 1 a partir da posição do robô. Caso o robô já tenha visitado a casa, elas são marcadas com o valor 2, e a área é incrementada com o valor de cada casa visitada.

A função ProcessarCenario é responsável por processar as informações completas de um cenário. Ela faz o uso das funções e estruturas anteriores para extrair todas as informações necessárias para calcular a área do cenário.

Em conclusão, no segundo exercício do projeto, trabalhamos com a estrutura Fila para ordenar e organizar as linhas que são lidas do arquivo XML e para verificar se as casas da matriz já foram limpadas ou não.

2. Algoritmos

2.3 Função main

```
int main() {
    string xmlFilename;
    cin >> xmlFilename;

    ResultadoVerificacao resultado = VerificarXML(xmlFilename);
    if (!resultado.certoErrado) {
        cout << "erro" << endl;  // O arquivo XML não é válido
        return 1;
    }

    queue<string> fila = resultado.fila;

    while (!fila.empty()) {
        if (fila.front() == "<cenario>") {
            InformacoesCenario informacoes = ProcessarCenario(fila);
            cout << informacoes.nomeCenario << " " << informacoes.area << endl;
        }
        else {
            fila.pop();
        }
    }

    return 0;
}
```

Como a função main estava presente tanto no exercício 1 como no 2, decidimos falar sobre ela no final do relatório.

No exercício 1, ela serve para quando chamamos a função de verificar XML ela salva na variável ResultadoVerificacao, e após isso confere se o valor do atributo resultado.certoErrado está correto ou não.

No exercício 2, ela serve para verificar o estado da fila. Ela tem uma condição de parada caso a fila esteja vazia, e enquanto o primeiro elemento da fila for diferente do cenario, removemos a linha da fila. Caso essa linha for uma tag de fechamento, limpamos a fila para trabalhar com um novo cenário. No fim, removemos a tag de fechamento e limpamos a área e imprimimos o resultado.

3. Dificuldades

Tivemos dificuldade para lidarmos com a lógica das matrizes e como faríamos para trabalhar com os índices. Além disso, as condições de parada foram uma dificuldade, pois não sabíamos como formular uma condição de parada que fosse compatível com os vários loops que o programa tem.

4. Referências

https://cplusplus.com/doc/tutorial/files/

https://cplusplus.com/reference/string/string/

https://en.cppreference.com/w/cpp/container/queue