UNIVERSIDADE FEDERAL DE SANTA
CATARINA CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

VINICIUS HENRIQUE RIBEIRO (23200351)

ENZO AMARAL CUSTODIO (22202688)

Relatório do Trabalho I - Estruturas de Dados

Florianópolis 2024

Introdução

O trabalho envolve implementar um programa que lê um arquivo XML com "cenários" para um robô aspirador guiado por um algoritmo que percorre as vizinhanças. Cada cenário é uma matriz binária onde o robô começa em uma posição inicial e deve varrer posições com o valor "1", verificando as quatro direções possíveis (vizinhança-4). O programa deve contar quantas posições com "1" foram varridas em cada cenário, sem poder avançar para regiões que com valor “0” na matriz, ou seja, fica contido apenas na sua região inicial de “1”s.

O XML precisa ser validado quanto ao aninhamento das tags usando uma pilha (estrutura LIFO) e, em caso de erro de aninhamento no XML, uma mensagem de erro será exibida. A movimentação do robô entre posições é feita pelo algoritmo de componentes conexos, utilizando uma fila (estrutura FIFO).

Validação do XML

A primeira tarefa é realizar a leitura do arquivo XML. Para isso, utilizaremos a nossa classe ArquivoXML.h, nela temos um método responsável para esta atividade. Segue o código:

```
void lerArquivo(string filename) {
    // Abertura do arquivo
    ifstream filexml( s: filename);
    if (!filexml.is_open()) {
        cerr << "Erro ao abrir o arquivo " << filename << endl;
        throw runtime_error("Erro no arquivo XML");
    }

    // Leitura do XML completo para 'texto'
    limparDados();
    char character;
    while (filexml.get( &c: character)) {
        texto_ += character;
    }
}
```

O código apresentado acima é quase idêntico ao fornecido pelo modelo. Usa um objeto ifstream para a leitura e salva o texto em uma string.

O próximo passo é validar o arquivo. Também temos um método específico para isso. Segue a imagem:

```
bool validarXML() {

    string tag = "";
    stack<string> pilhaTags_;
    size_t pos = 0;

    for (; pos < texto_.length(); pos++) {
        if (texto_[pos] == '<') {
            pos++; // estava em <, passa para a primeira letra da tag
            while (texto_[pos] != '>') {
                tag += texto_[pos];
                pos++;
            }
            // pos++;                  // estava em >, passa para a primeira posição fora ou dentro da tag
            if (tag.length() == 0) { // tag não tem nome (vazia)
                descricaoValidadeXML_ = "erro";
                return xmlValido_;
            }
            if (tag == "cenario")
                qtCenario_++;
            if (tag.at( n: 0) != '/') { // tag de abertura. Empilha
                pilhaTags_.push( x: tag);
                tag.clear();
                continue;
            }
            tag.erase( pos: 0,  n: 1);
            if (tag != pilhaTags_.top()) { // tag de fechamento difere da tag de abertura
                descricaoValidadeXML_ = "erro";
                return xmlValido_;
            }

            pilhaTags_.pop(); // desempilha devido ao fechamento da tag
            tag.clear();
        }
    }

    if (pilhaTags_.empty()) {
        xmlValido_ = 1;
        return xmlValido_;
    }

    descricaoValidadeXML_ = "erro";
    return xmlValido_;
}
```

No início temos a declaração de três variáveis:

- tag: responsável por guardar o valor da tag temporariamente;
- pilhaTags: uma pilha da biblioteca std;
- pos: para guardar a posição na texto.

O for itera até o final da string texto. Nele temos um while que percorre a string armazenando o valor da tag na variável auxiliar de seu nome. Terminando de ler, a execução vai para as estruturas condicionais. A primeira verificação é se a tag está vazia, caso seja o XML é inválido. A segunda condição verifica se é um cenário, e caso seja, incrementamos o contador de cenários. A terceira condição verifica se é uma tag de abertura, sendo verdadeira, a tag é empilhada e vamos para a próxima iteração do for. Caso não entre na condição, se trata de um fechamento de tag, verificamos a sua igualdade com a do topo da pilha, se não for igual se trata de um XML inválido, sendo igual nós desempilhamos e seguimos para a próxima iteração.

Ao final do método, após todas as iterações do for, verificamos se a pilha está vazia, caso não esteja se trata de um XML inválido. Não entrando na condição o XML é válido.

Verificação das possíveis direções de avanço do robô aspirador

O segundo problema se trata apenas da aplicação do algoritmo dado no enunciado do trabalho. Segue a sua implementação:

```cpp
size_t resolveQuestao2() {
    // alocação da matriz para aplicação dos algoritimos
    aloca_matriz_dinamicamente();
    inicializa_matriz_numerica();

    queue<Coordenada *> fila_coordenadas;
    // criação da primeira coordenada
    fila_coordenadas.push( x: new Coordenada( x: x_,  y: y_));

    size_t quantidade_ums = 0;

    while (!fila_coordenadas.empty()) {
        Coordenada *ptrCoord = fila_coordenadas.front();

        // vizinho acima (x-1, y)
        if (ptrCoord->X() - 1 >= 0) {
            if (matriz_[(ptrCoord->X() - 1) * largura_ + ptrCoord->Y()] == '1' && matriz_numerica_[ptrCoord->X() - 1][ptrCoord->Y()] == 0) {
                matriz_numerica_[ptrCoord->X() - 1][ptrCoord->Y()] = 1;
                quantidade_ums++;
                fila_coordenadas.push( x: new Coordenada( x: ptrCoord->X() - 1,  y: ptrCoord->Y()));
            }
        }

        // vizinho abaixo (x+1, y)
        if (ptrCoord->X() + 1 < altura_) {
            if (matriz_[(ptrCoord->X() + 1) * largura_ + ptrCoord->Y()] == '1' && matriz_numerica_[ptrCoord->X() + 1][ptrCoord->Y()] == 0) {
                matriz_numerica_[ptrCoord->X() + 1][ptrCoord->Y()] = 1;
                quantidade_ums++;
                fila_coordenadas.push( x: new Coordenada( x: ptrCoord->X() + 1,  y: ptrCoord->Y()));
            }
        }

        // vizinho à esquerda (x, y-1)
        if (ptrCoord->Y() - 1 >= 0) {
            if (matriz_[(ptrCoord->X()) * largura_ + ptrCoord->Y() - 1] == '1' && matriz_numerica_[ptrCoord->X()][ptrCoord->Y() - 1] == 0) {
                matriz_numerica_[ptrCoord->X()][ptrCoord->Y() - 1] = 1;
                quantidade_ums++;
                fila_coordenadas.push( x: new Coordenada( x: ptrCoord->X(),  y: ptrCoord->Y() - 1));
            }
        }

        // vizinho à direita (x, y+1)
        if (ptrCoord->Y() + 1 < largura_) {
            if (matriz_[(ptrCoord->X()) * largura_ + ptrCoord->Y() + 1] == '1' && matriz_numerica_[ptrCoord->X()][ptrCoord->Y() + 1] == 0) {
                matriz_numerica_[ptrCoord->X()][ptrCoord->Y() + 1] = 1;
                quantidade_ums++;
                fila_coordenadas.push( x: new Coordenada( x: ptrCoord->X(),  y: ptrCoord->Y() - 1));
            }
        }

        fila_coordenadas.pop();
        delete ptrCoord;
    }
    qt_ums_ = quantidade_ums;
    return quantidade_ums;
}
```

No início do método temos uma chamada para criar a matriz numérica (a classe Cenario.h tem um atributo para a matriz) e outra para inicializá-la com zeros. Seguindo a execução, temos a declaração de uma fila para o funcionamento do nosso algoritmo. Primeiramente, nós adicionamos a coordenada de origem do robô (Usamos uma classe Coordenada.h que têm dois atributos, x e y, para as coordenadas). Na estrutura de repetição while, nós tiramos o primeiro da fila e passamos pelas 4 estruturas condicionais. Cada uma dessas condições irá checar se há um vizinho com o bit 1. Havendo um vizinho 1, adicionamos a coordenada dele na fila, registramos 1 na matriz numérica e incrementamos o contador de 1s. Ao final nós liberamos a memória da coordenada analisada. Este processo ocorre até que a fila esteja vazia, ou seja, não tenha mais vizinhos para serem analisados.

O main.cpp

No main.cpp temos a leitura do nome do arquivo pelo terminal e a sua validação. Não sendo válido é retornado 1 e impresso na tela a palavra erro. Seguindo, temos um while que irá iterar sobre todos os cenários existentes em texto, aplicando o algoritmo do robô e imprimindo a quantidade 1s varridas.

```cpp
int main() {

    string filename;

    std::cin >> filename;

    ArquivoXML a;
    a.lerArquivo(filename);
    a.validarXML();
    if (!a.XMLValido()) {
        cout << a.descricaoValidadeXML() << endl;
        return 1;
    }

    string texto = a.Texto();

    Cenario c;

    size_t cont = 0;
    while (cont < a.qtCenario()) {
        cont++;
        c.limpa_cenario();
        c.monta_cenario( &: texto,  indice_inicial: c.indice_final());
        cout << c.nome() << ": ";
        c.resolveQuestao2();
        cout << c.qt_ums() << endl;
    }

    return 0;
}
```

Conclusão

A maior dificuldade foi na organização do código e na delegação de responsabilidade das classes. Como os algoritmos foram passados em aula e também disponibilizados no enunciado, não houve grande problema nesta parte. Tivemos alguns problemas de falha de segmentação pela utilização das matrizes e vetores, mas foi superada utilizando o depurador GDB.

Referências

https://en.cppreference.com/w/