Universidade Federal de Santa Catarina

Projeto 1 - Relatório

Estrutura de Dados - INE5408

Autores: Julia Macedo de Castro e João Pedro Schmidt Cordeiro

Introdução

A princípio, antes de começarmos a implementar a solução do problema proposto, foi necessário analisar o que precisaríamos fazer. Então foi-se estabelecido que seriam necessárias duas partes principais. Primeiramente, ler o arquivo XML e em seguida organizar o conteúdo em algum tipo de estrutura de organização, a fim de verificar se o arquivo foi construído corretamente e por fim foi necessário implementar o algoritmo de limpeza da matriz.

```
85  int main() {
86
87      char xmlfilename[100];
88
89      std::cin >> xmlfilename;  // entrada
90
91      // Read input file
92      std::string input = read_file(xmlfilename);
93      LinkedList<Cenario> *list_cenarios = new LinkedList<Cenario>;
94
95      // Parce the cenarios and check if the structure of the XML is correct
96      try {
97          parse(input, *list_cenarios);
98      }
99      catch (...) {
100         std::cout << "erro" << std::endl;   // Print erro in the terminal
101         return 0;
102     }
103
104
105     int count = 0;
106     // Check how many spaces the robot will clean
107     for (int i = 0; i < list_cenarios->size(); i++) {
108         Cenario cenario = list_cenarios->at(i);
109         std::cout << cenario.nome << " " << counter(&cenario) << std::endl;
110     }
111
112
113     // std::cout << xmlfilename << std::endl;  // esta linha deve ser removida
114
115     return 0;
116 }
```

Parte 1 (XML)

O passo inicial foi criar uma função *read_file( )* para efetuar a leitura do arquivo. Ela recebe o nome do arquivo a ser lido, retornando, após algumas operações, o arquivo lido em formato de *std::string.* O próximo passo então é a leitura desses dados e organização dos mesmos.

Com alguns dias pensando e intercambiando ideias com os colegas da turma percebemos que seria necessário criar uma estrutura para armazenar as informações de cada cenário então criamos a estrutura *Cenário* que armazena separadamente o nome, altura, largura, posição inicial

e a matriz referentes a um cenário. Visto que cada arquivo possui mais de um cenário, também foi necessário criar uma lista para armazenar cada cenário.

A função *parse( )* é então chamada e recebe como argumentos o conteúdo lido pela função anterior e a lista de cenários (*lista_cenarios*). Com essas informações a função cria uma instância de uma pilha, utiliza um recurso de leitura e manipulação de strings e entra em um laço *while.* A leitura de strings é feita utilizando funções inspiradas na classe String da linguagem C++, contudo adicionando algumas funcionalidades específicas para a aplicação:

get_i( ): pega o caractere atual da string
caracter( ): pega o próximo caractere da string
palavra( ): pega a próxima palavra da string
espera( ): verifica se o próximo caractere é igual a "ch"

O laço irá se repetir até a finalização da leitura da string, e a cada iteração irá estabelecer os limites estipulados pelo carácter "<". Dentro de cada limite irá conter o que chamaremos de *tag*, e esse valor será inserido na pilha.

Caso a próxima *tag* seja uma tag de fechamento, indicada pelos dois caracteres consecutivos, "</", a função entrará em uma condição que irá retirar o elemento do topo da lista e em seguida verificará se é um elemento válido, caso seja, irá checar as demais chaves relativas a cada tag possível, e armazenará no devido lugar estabelecido pela estrutura *Cenário* criada anteriormente.

Também é importante ressaltar que decidimos armazenar a matriz em formato de lista de valores booleanos.

Por fim, após a devida organização dos cenários na lista de cenários podemos ir para a parte 2, relacionada ao tratamento de cada matriz.

Parte 2 (Limpeza pela matriz)

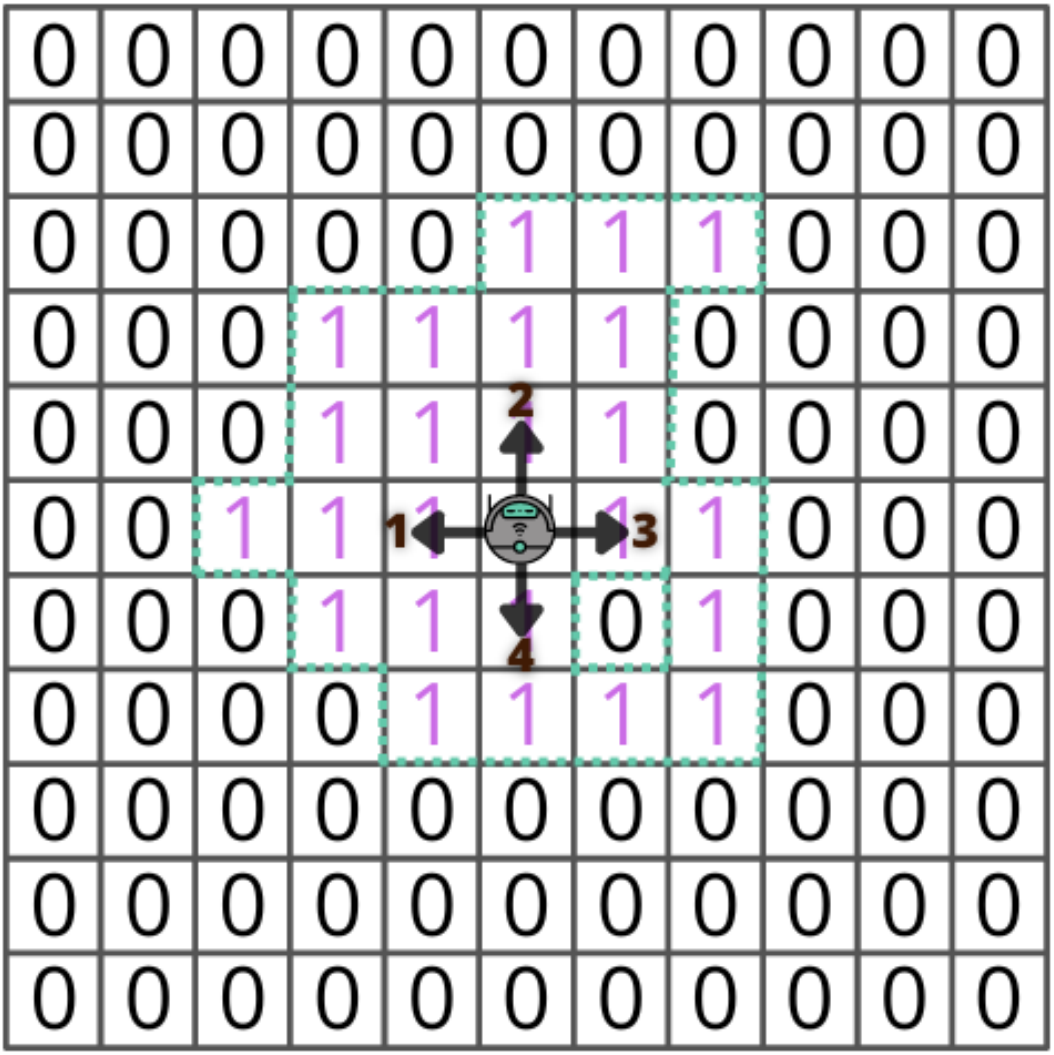
Com os cenários organizados, percorremos a lista e utilizamos a função *counter( )* para verificar quantos espaços o robô deve limpar em cada cenário.

A função *counter( )* conta quantos espaços devem ser limpos em cada cenário. Primeiramente é criado uma matriz preenchida de zeros, da mesma ordem da matriz a ser limpa. Em seguida preenche o espaço inicial do robô e entra em um laço para limpar os demais espaços.

Dentro do laço, ela percorre as matrizes e, utilizando lista encadeada (index list), pegamos os índices dos espaços que serão visitados comparando o mesmo com um array booleano (*matriz_limpa*) a fim de verificar se ainda é necessário limpar o espaço. Recordamos também, que a cada iteração do laço, os pontos adjacentes à posição do robô são verificados, e a *matriz_limpa* tem justamente a funcionalidade de referência, para que um mesmo espaço não seja limpo duas vezes.

Esse algoritmo finaliza quando não possuem mais espaços a serem visitados na *index_list*.

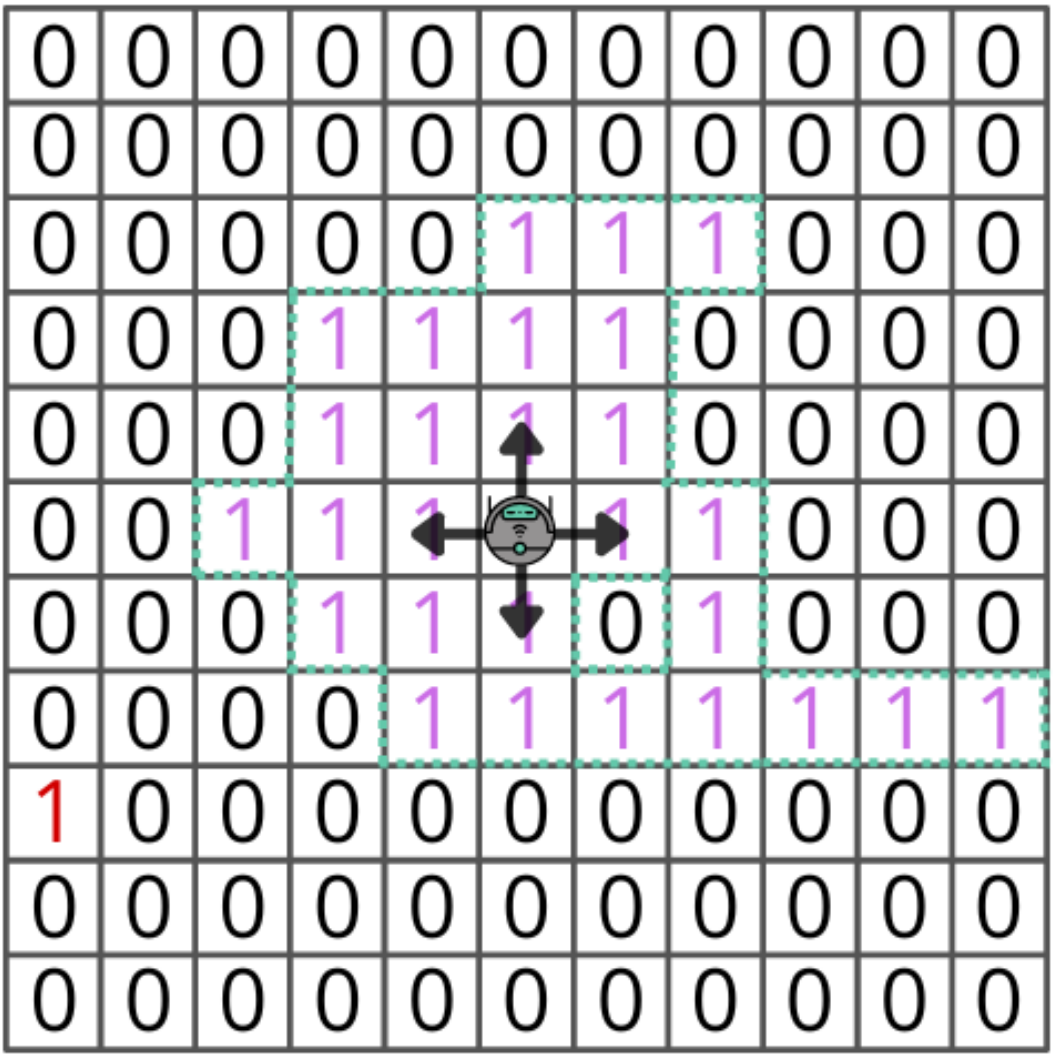
Imagem ilustrando a área que o robô percorre



Conclusão

Em geral, a lógica do projeto em si não era difícil, mas os algoritmos foram um pouco mais complicados de aplicar. Tivemos dificuldades em ajustar alguns erros específicos na área de limpeza da matriz em que o robô estava limpando áreas fora do limite. Por exemplo, se uma posição na extrema direita precisasse ser lida e ao mesmo tempo uma outra, isolada na primeira posição, porém na linha de baixo também seria limpa, quando na verdade não deveria ser.

A imagem indica o número 1 em vermelho, sendo o exemplo do erro que estávamos tendo. Nesse caso o espaço que está na extrema esquerda também seria limpo.



Ao corrigir esse erro e reorganizar o código foi possível completar a tarefa com sucesso.

Referências

- https://cplusplus.com/doc/tutorial/files
- https://cplusplus.com/reference/string/string
- https://github.com/vberlier/tokenstream
- https://cplusplus.com/reference/string/string