

Alien vs Ripley

Ítalo Dell’ Areti

Lucas Paulo Martins Mariz

Raul Formiga Mansur

Coltec/2016

*Especificação do Trabalho Prático (10 pontos) ­*

*Data de entrega: 11/07/16*

*(Nº de participantes: 3)*

*Todos os códigos precisam estar executando corretamente no dia da apresentação.*

*O relatório deve conter: ­*

1. Explicação do problema; ­
2. Modelagem do problema (qual estrutura de dados usada, qual problema mapeado); ­
3. Complexidade do problema;
4. ­ Dificuldades encontradas e soluções tomadas para resolvê-­las; ­
5. O relatório deve ser feito em um documento do Google Docs compartilhado com a professora.

TEMA 1: Alien vs Ripley

Dupla:​ Raul, Lucas P. e Ítalo

A nave Nostromo foi invadida por uma criatura vil e cruel: o Alien! Spoiler Alert: A única sobrevivente da nave, Ripley, está tentando escapar desse labirinto! Porém o Alien pode aparecer em qualquer ponto da Nostromo. Implemente um programa para ajudar a Ripley a traçar um caminho para sair do labirinto. Cuidado, o Alien aparece aleatoriamente em qualquer ponto do labirinto e pode matar a Ripley muito rápido.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Contextualização:

No decorrer da disciplina Algoritmos e Estruturas de Dados, fomos apresentados a desafios envolvendo a manipulação de estruturas mais complexas na linguagem C. Como proposta da professora Virgínia Fernandes Mota, estamos desenvolvendo métodos para a solução do problema descrito no Tema 1: Alien vs Ripley.

Seguido o conteúdo estudado no segundo trimestre, temos grafo como a estrutura base na qual o trabalho será baseado. A proposta é simples: há uma nave no qual Ripley deverá encontrar a saída. Contudo, o alien poderá aparecer durante o percurso, devendo Ripley mudar seu caminho caso queira sobreviver.

Colocando em perspectiva, basicamente deverá haver dois processos:

* Traçar uma rota mais curta para a saída;
* Caso o alien apareça, deve-se mudar a rota para uma na qual o caminho esteja livre.

Sendo assim, podemos definir a seguinte estrutura de dados:

//Estrutura principal do grafo:

struct grafo{

int \*\*matriz; //matriz de adjacências;

char \*\*lados; //lado de cada vértice;

int \*graus; //graus de cada vértice;

int n\_vertices; //quantidade de vértices;

int alien; //onde o alien esta;

int ripley; //onde a ripley esta;

//inicia-se com -1

}typedef struct grafo Grafo;

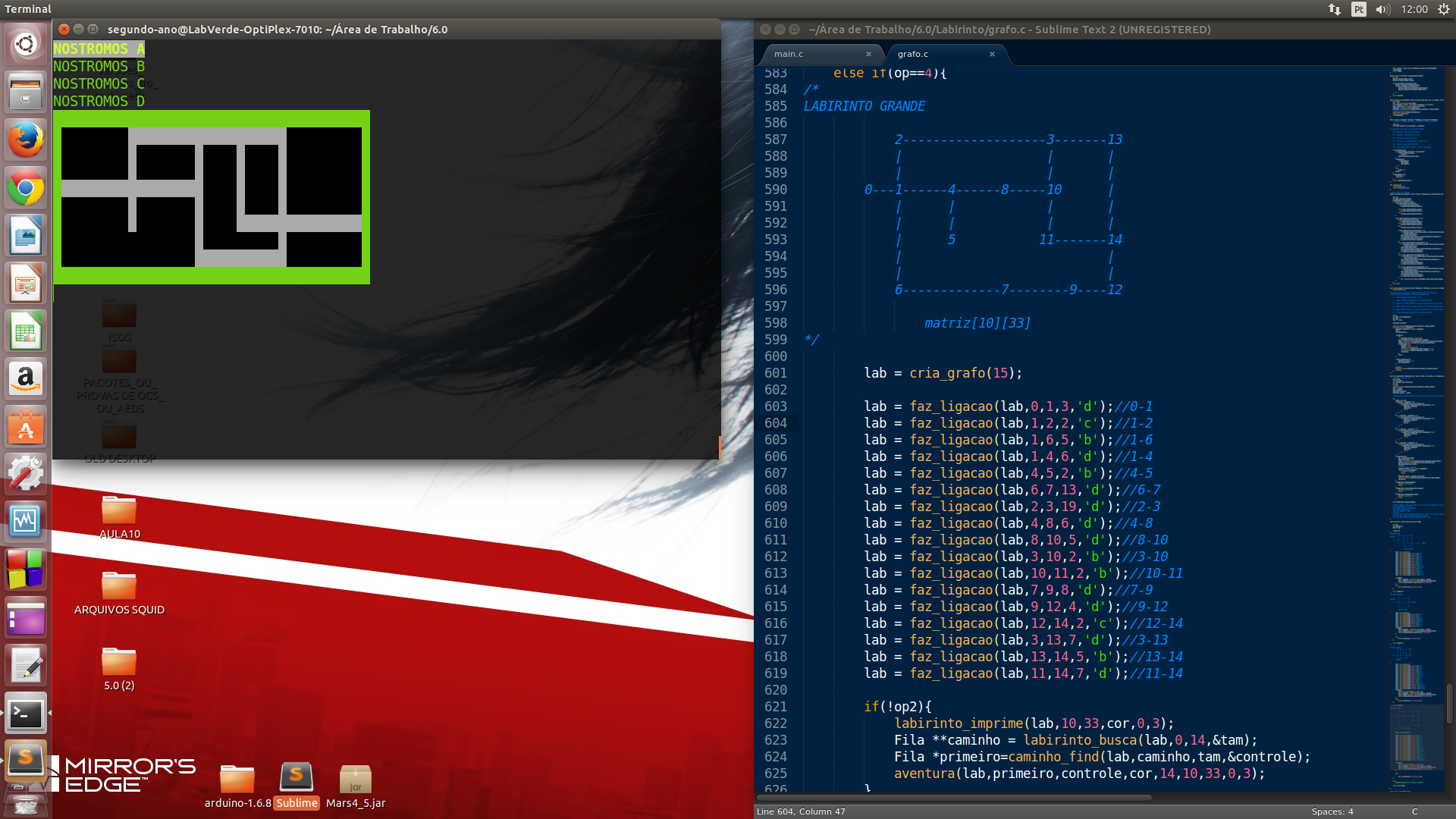
Desta vez, a matriz que representa o grafo está situada na estrutura de forma que, se o valor for 0, ele não existe e, caso contrário, o valor representado nos diz o peso da aresta. A matriz de caracteres armazena dados representado as direções (Ex: ‘d’=direita, ‘c’=cima). Sua existência tem como objetivo facilitar a exploração do grafo, de forma que possamos ter informações mais precisas no que se relaciona as coordenadas dos vértices. O vetor ‘graus’ tem a função de guardar o numero de arestas conectadas a cada vértice (grafo orientado). Já o n\_vertices determina a quantidade total de vértices do labirinto.

Foi implementado o algoritmo para busca de todos os caminhos possíveis a serem tomados pela Ripley, o qual deveria ter em consideração determinadas regras. A alien aparece randomicamente pelo mapa, com a condição de não aparecer em cima da Ripley. O algoritmo por sua vez procura os caminhos nos quais o alien não se encontra e os levam a saída. Dentre esses caminhos, a rota mais curta é selecionada. O caminho é refeito toda vez que o alien aparece na rota a ser seguida.

O algoritmo usado para a busca de caminhos se baseia em uma busca em profundidade, que irá encontrar todos os caminhos possíveis a partir de um determinado vértice, e formará um vetor com filas que contém estas filas. Após gerar todos os caminhos possíveis, verifica-se quais se encaixam na rota desejada, quais não possuem o alien no caminho, e qual é o mais curto. Após escolher um caminho, RIpley o segue, e quando o alien aparece em algum vértice, verifica se ele está presente no caminho, e caso sim, refazemos o algoritmo de busca de caminhos, para encontrar uma nova rota. Existem casos em que o alien pode bloquear todas as possibilidades de Ripley encontrar a saída, e assim, decidimos fazer a Ripley esperar por ele desocupar tal posição.

Existe também, um algoritmo para gerar a matriz que usamos visualmente no trabalho. Ele também se baseia em uma busca em profundidade, que sempre que passa por um vértice, verifica a posição de seu adjacente através da matriz de caracteres do grafo, e preenche em um outra matriz, inicialmente nula, com 1s os caminhos que forem percorridos durante a execução do algoritmo.

Em questão de visualização por parte do usuário, buscamos uma forma mais eficiente de se imprimir a matriz. A solução encontrada foi o caractere espaço, com uma cor diferente. Com isso, a labirinto apresentado acima fica aproximadamente assim:



Levando em consideração o processo de busca em profundidade com a finalidade de encontrar todos os caminhos possíveis a partir de um vértice a saída, temos uma complexidade demarcada de O(n), onde 'n' representa o número de vértices. No entanto, dependendo do caso, dever ser considerado outros fatores, adicionando uma outra incógnita: o número de arestas. A complexidade resultante seria O(max(n,a)), onde o 'a' representa o número de arestas. Sendo assim, se um grafo possui mais arestas do que vértices temos uma complexidade igual O(a), caso contrário, O(n). No pior dos casos, um grafo infinito, a complexidade seria O(|n | + |a|).

Material consultado:

http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/Busca/busca.html

http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/IA/ResolProb/resproblema.html

http://www2.dcc.ufmg.br/livros/algoritmos-java/cap7/transp/completo4/cap7.pdf

Dificuldades Encontradas:

O trabalho descrito anteriormente não foi a versão inicial e final do projeto. Durante o seu desenvolvimento passamos por diversas dificuldades envolvendo buscas em profundidade, complexidade do problema e erros de lógica.

A primeira versão do trabalho é descrita abaixo, apelidada de AoN (Operação Abandonar o Navio) , a qual foi abandonada após diversos problemas acumulados:

“*Inicio da descrição da AoN*

Foi definida a estrutura principal do grafo:

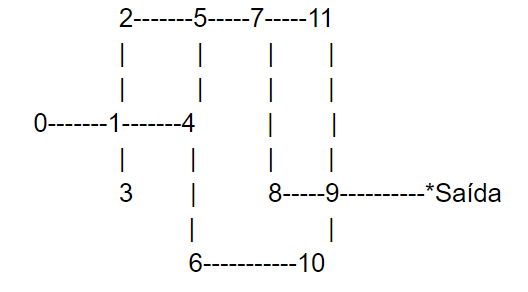
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | struct vertice{  int alien;  int ripley;  int distancia[4];  struct vertice \*adjacente[4];  int identificador[4];  int n;}; typedef struct vertice Vertice; |

O nome da estrutura é vertice. Seguimos esse linha de raciocínio como se cada vértice fosse uma ala da nave. As variáveis inteiras alien e ripley variam entre 0 e 1 indicando se, os respectivos personagens se encontram na posição demarcada da nave. O vetor distancia, indica o quão distante a posição atual está da próxima (é um vetor pois, ao se tratar de duas dimensões, temos quatro direções posições possíveis: esquerda, direita, cima, baixo - nesta ordem).

A estrutura vertice \*adjacente[4] se trata da próxima ala da nave (ou próxima posição em que é possível acessar no labirinto) e, pelos mesmos motivos da distância, é um vetor com quatro posições.

A variável inteira identificador é um vetor e existe com o motivo de representar as quatro direções citadas. Sua função específica se refere a evitar a ocorrência de loops infinitos, como no caso de tentar percorrer todo um labirinto que forma uma figura geométrica. Por fim, a variável n representa o vértice em que se está (Ex: 5).

Montada a estrutura, temos a formação do seguinte labirinto:



Os processos para sua criação são bem distintos. Em um primeiro momento, cada um dos vértices (0-Entrada, 1 a 10-intermediários, 12-Saída) são criados e inseridos no vértice inicial. Logo após, foi construída uma função que, a partir da estrutura de um grafo, fornece a altura e a largura para a criação de uma matriz que represente o labirinto.

Se trata de uma matriz binária. Onde há 1 poderá ser um possível caminho para saída. Para poder transferir os dados do grafo de maneira precisa para a matriz, foi criada a função preenche\_matriz que, ao receber por parâmetros a a largura, altura e o grafo, transfere os dados para uma matiz binária.

O próximo passo seria a realização da função que acharia todos os possíveis caminhos até a saída, de forma que a Ripley escolheria o menor deles desde que o Alien não se encontre em seu trajeto.

Devido a complicação gerada pela forma de realização do trabalho e os constantes problemas envolvendo ponteiros, resolvemos tentar a solução do problema de outra maneira.

Entra em cena então a operação AoN (Abandonar o Navio), em que desistimos, e começamos do zero novamente.

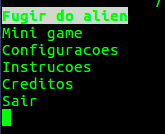
*Fim da descrição da AoN*“

A dificuldade consistia em construir um algoritmo único que, de uma maneira eficiente, percorresse todo o grafo e armazenasse todos os possíveis caminho a saída.

Sendo assim, o trabalho passo desde a versão inicial até a versão 10.0 devido a novas correções e implementações. Apesar de todas as dificuldades encontradas, sentimos que todo o processo foi recompensado (conferir os créditos).

Funcionamento:

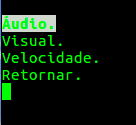
O trabalho funciona com um menu principal, que contém diversas opções:



**Fugir do alien:** Principal ala do trabalho onde os processos de buscas por caminhos e uma fuga automática são realizados.

**Mini game:** Modo jogável onde é possível controlar a Ripley.

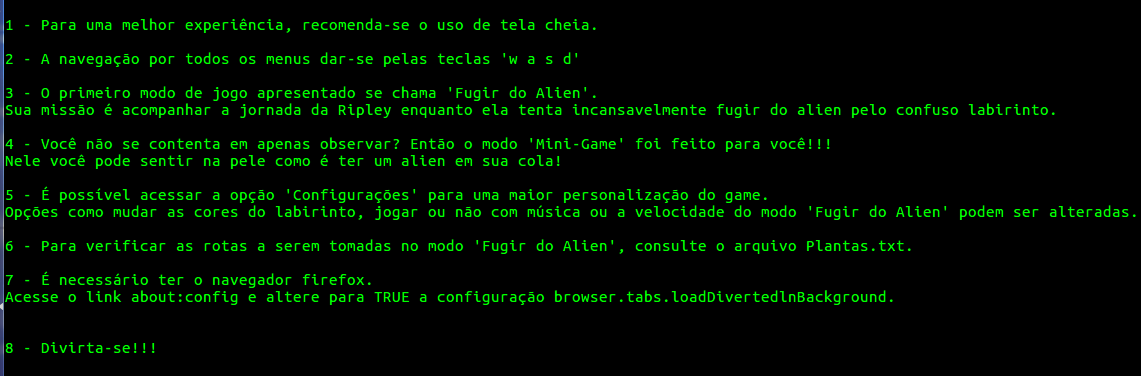
**Configurações:**



Selecionar o modo de jogar ou não com música, as cores do labirinto ou a velocidade do modo “Fugir do Alien”.



**Instruções:**



Conclusão:

Mesmo após diversos problemas encontrados, sentimos que este trabalho fez com que nossos conhecimentos fossem aprimorados. As incontáveis horas dedicadas ao seu desenvolvimento absorvemos lições para que nos tornássemos melhores programadores. Todo o estresse, tempo gasto, horas sentado em frente ao computador, entre outros, nos trouxeram a este epílogo, este pequeno momento, chamado felicidade, fazendo toda a caminhada valer a pena, todo o esforço ser recompensado.