Relatório do Trabalho 2
Algoritmos e Estruturas de Dados II

Douglas Schlatter, Gabriel Panho e João Marmentini

2021

1 Definição do Problema

Este relatório documenta uma possível solução para o problema "O Caranguejo Perdido". A definição desse desafio consiste em encontrar o menor caminho para que o caranguejo chegue a saída do labirinto em que se encontra, seguindo uma série de regras de ambiente e de movimentações.

1.1 Regras de Ambiente

O labirinto representado por uma matriz possui pontos em que o caranguejo não pode atingir devido a estar bloqueado por obstáculos representados pela letra X e caminhos livres representados pelo '.'. O tamanho desta matriz pode variar de 50x50 a 1000x1000. A saída é demarcada pela casa com o caractere 'S' e a posição inicial do caranguejo pelo caractere 'C'. Essas características podem ser observadas na Figura 1.

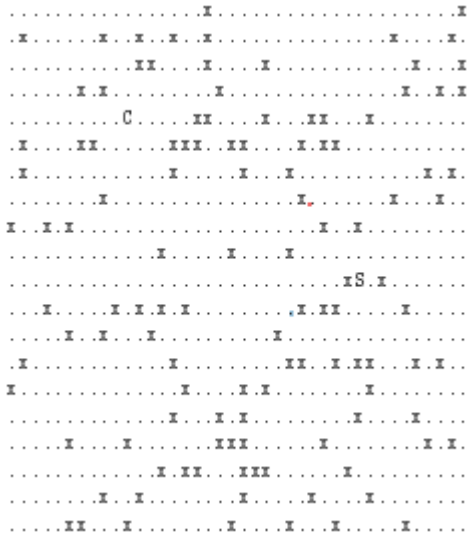

Figura 1 – O caranguejo é representado pela letra C, seu destino/saída é representado pela letra S, posições possíveis de serem ocupadas são representadas pelos pontos e obstáculos por X.

1.2 Regras de movimento

O caranguejo pode movimentar-se uma casas em sentido diagonal ou duas casas para norte, sul, leste e oeste, porém não pode atravessa uma parede para realizar esses movimentos. É possível visualizar essas condições na Figura 2.

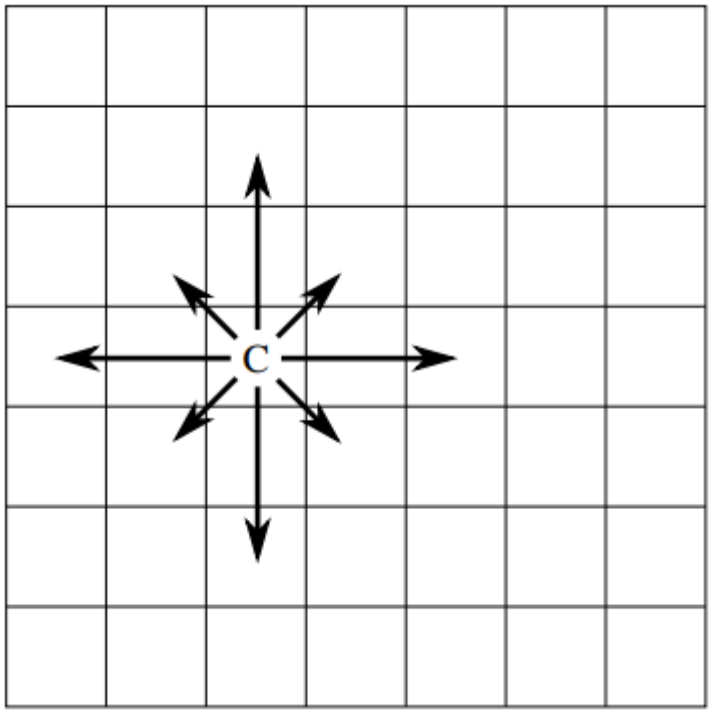

Figura 2 – O caranguejo é representado por "C", a imagem representa as regras de movimento do caranguejo.

Devido às regras de movimento é importante ressaltar que existem determinadas formações de labirinto nas quais o caranguejo não terá capacidade de encontrar a saída. A razão dessa impossibilidade será melhor discutida na secção 2 do relatório.

2 Modelagem do Problema e Hipótese de Solução

Analisando o problema observa-se que devido as limitações de movimento do caranguejo, existem situações em que não será possível atingir a saída do labirinto. A primeira situação em que isto ocorre é quando obstáculos isolam o objetivo, assim o personagem não encontra caminhos para o seu destino. Já a segunda provem de uma propriedade que pode ser notada no problema, devido a movimentação do caranguejo sua posição inicial dita quais posições ele consegue acessar. Isto se da pelo fato de que, caso a posição inicial do caranguejo tenha o somatório de suas posições cardiais um numero resultante par, ele somente poderá ocupar posições que também possuam somatório equivalente a par.O mesmo ocorre caso a posição inicial tenha o resultado de seu somatório em impar. Isso pode ser observado através da figura 3

2.1 Lógica A*

Decidimos por utilizar o algoritmo A* para calcular o melhor caminho entre o caranguejo e o destino. Para acompanhar melhor a explicação da lógica do programa aconselhamos observar a 3. Dentro da lógica desse algoritmo existem dois tipos de nodos, os abertos e os fechados, sendo respectivamente, nodos a serem observados e nodos que já foram observados. Para decidir o

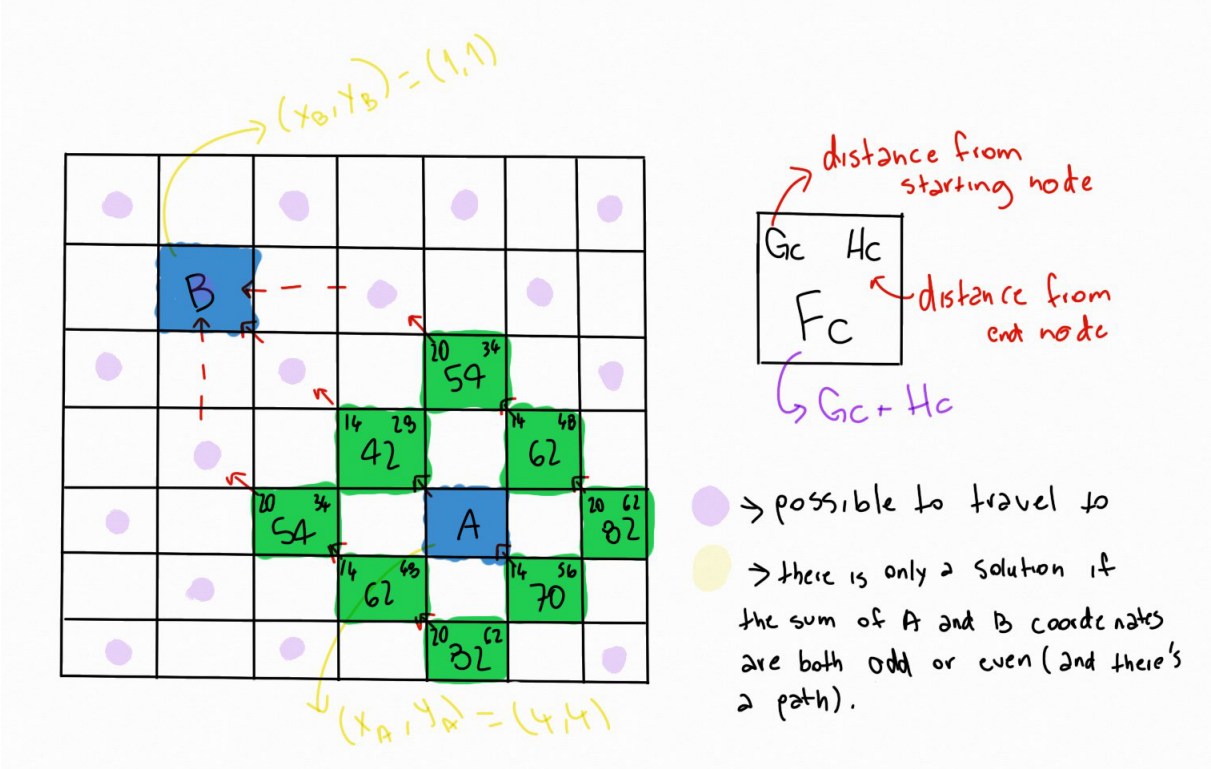

Figura 3 – Nesta imagem "A"representa o caranguejo,ele "nasce"em uma posição com somatório par (par de coordenadas = (4,4)), em roxo pinta-se todos os nodos que o caranguejo pode vir a ocupar.Visto que a saída também possui posição com somatório par (1,1) esse é um exemplo de labirinto em que o personagem tem capacidade de sair, afinal é uma posição que ele pode ocupar.

caminho com menor esforço ou menor distancia é necessário estabelecer um sistema de custo. Estabelecemos, então, que andar para os lados custaria 1 e enquanto na diagonal custaria 1,4. Para não trabalharmos com números flutuantes multiplicamos tudo por 10, assim nossos custos tornam-se 10 e 14. Um ajuste final, visto o problema a ser resolvido, como nosso personagem consegue se mover ou em uma unidade em diagonal, ou em 2 unidades para os lados . Dessa forma os custos reais para os lados é 20,enquanto para as diagonais é 14. Estabelecidas nossas unidades temos que estabelecer os custos dos nodos. O custo de nodo é composto de dois sub-custos somados que são calculados com as unidades já mencionadas. O primeiro consiste da distância do nodo observado da origem, sendo representado pelo custo no canto superior a esquerda na figura 3, posteriori consiste da distancia do nodo observado do destino, representado pelo custo no canto superior a direita na figura 3. Por fim, a soma destes custos gera o custo final, localizado ao meio do nodo na figura 3.

2.2 Pseudocódigo A*

Na figura 4 é possível observar o pseudocódigo do algoritmo A*.

Optou-se pela utilização da linguagem Python pois ela oferece uma ampla variedade bibliotecas úteis que facilitam a solução do problema. Uma bibliotecas utilizada foi a NetworkX, capaz de criar e manipular estruturas e redes de grafos complexas. Também levou-se em conta a experiência prévia dos integrantes com esta linguagem de programação, sua simplicidade e eficiência.

```
OPEN //the set of nodes to be evaluated
CLOSED //the set of nodes already evaluated
add the start node to OPEN

loop
    current = node in OPEN with the lowest f_cost
    remove current from OPEN
    add current to CLOSED

    if current is the target node //path has been found
        return

    foreach neighbor of the current node
        if neighbor is not traversable or neighbor is in CLOSED
            skip to the next neighbor

        if new path to neighbor is shorter OR neighbor is not in OPEN
            set f_cost of neighbor
            set parent of neighbor to current
            if neighbor is not in OPEN
                add neighbor to OPEN
```

Figura 4 – Pseudocódigo do algoritmo A*

3 Metodologia

O primeiro passo para a solução do problema é a leitura dos arquivos de teste. O programa lê apenas um arquivo por execução, e é necessário mudar dentro do código qual arquivo de teste será utilizado. Após isso, criamos uma lista de listas, que conterá cada posição do labirinto, e facilitará a criação das vértices. Em seguida, procuramos nessa matriz a posição inicial do caranguejo e a final e fazemos o teste explicado na seção de Modelagem.

O próximo passo é criar o grafo. A matriz é percorrida linearmente, e nas posições onde é possível o caranguejo se mover e não é um obstáculo, é criado o nodo e uma lista de vizinhos. Nesta lista, checamos os nodos que são possíveis de ser visitados a partir de uma vértice. Após finalizar esse processo, adicionamos as arestas com base na lista de vizinhos mencionada anteriormente.

Então, iniciamos uma variável com o tempo atual e aplicamos o algoritmo de pesquisa A* no grafo recém criado. Esse algoritmo é importado da mesma biblioteca que cria os grafos e seu retorno é uma lista com as vértices do caminho encontrado. Após tratar o resultado dessa lista e mudar tanto as vértices quanto as arestas, imprimimos no terminal o tempo atual menos o tempo iniciado antes do algoritmo de pesquisa.

Por fim, utilizamos a biblioteca Matplotlib para gerar uma imagem representativa do grafo. Esse processo é bastante custoso para o computador, e em alguns casos, impossível de reconhecer as vértices e arestas devido a grande quantidade de nodos.

4 Análise de Resultados

Na tabela abaixo pode-se ver os resultados obtidos após a conversão de cada matriz para grafo e executando o algoritmo A*. Os resultados representados por dois hifens indicam que na matriz recebida não há solução matematicamente possível com base em quantos passos o caranguejo pode dar nas direções. Já os resultados simbolizados pela letra X indicam que seria

possível atingir a saída porém ela está obstruída por algum elemento do ambiente. Na imagem 5 esta a representação gráfica da matriz convertida para grafo com o caminho encontrado através do algoritmo destacado em vermelho.

Tamanho Labirinto	Nodos no menor caminho	Tempo de execução da busca
50x50	22	0.000211s
$75x75_1$	X	X
$75x75_2$	–	–
$75x75_2$	–	–
$75x75_3$	X	X
$75x75_4$	–	–
$75x75_5$	59	0.000783
75x75	–	–
100x100	X	X
250x250	X	X
640x480	242	0.004546
1000x1000	364	0.009400

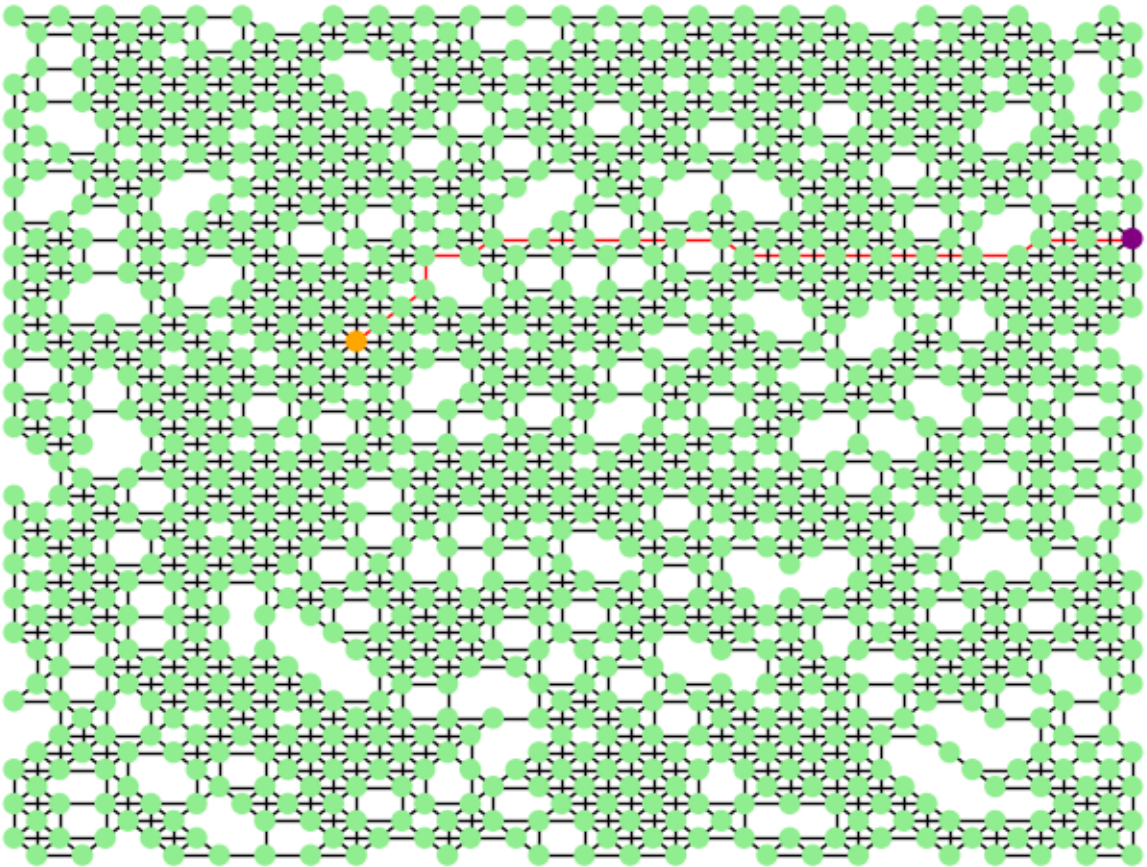

Figura 5 – Teste de tamanho 50x50

5 Conclusão

Este trabalho nos mostrou um pouco das capacidades de solução de problemas utilizando grafos e a importância de estudar algoritmos e estruturas para conseguir desenvolver soluções eficientes e simples, visto que estão cotidianamente presentes.