UNIVERSIDADE DE SÃO PAULO
SSC0230 - INTELIGÊNCIA ARTIFICIAL

TRABALHO PRÁTICO 1 - RELATÓRIO

PAULO H. BODNARCHUKI DA CRUZ.....9790944
EDUARDO ZABOTO MIROLLI.....9778501
BEATRIZ MONTEIRO.....9778619

SÃO CARLOS
2019

SUMÁRIO

INTRODUÇÃO

O primeiro trabalho prático da disciplina de Inteligência Artificial (SCC0230), consiste na implementação de 4 tipos de busca: Depth First Search(DFS), Breadth First Search(BFS), Best First Search e A* Search. As duas primeiras são buscas do tipo cega e as demais, do tipo informada. O intuito aqui é o de analisar como todas elas se comportam quando, dado um labirinto de tamanho $n \times m$, precisam encontrar um caminho entre uma posição de origem e uma posição de destino. Quanto à movimentação, é permitido deslocar-se em qualquer uma das 8 direções, porém a um custo de $\sqrt{2}$ nas diagonais(↖, ↗, ↘ e ↙) e 1 nas demais direções(←, ↑, → e ↓). Esses custos, bem como os tempos decorridos, número médio de elementos nos caminhos gerados pelas execuções, dentre outras métricas, serão altamente relevantes para comparar o desempenho e eficiência dessas buscas e também para inferir os contextos em que elas são mais adequadas. Por fim, os códigos para atingir o objetivo descrito foram implementado em linguagem C, C++ e shell script(para o cálculo automatizado das métricas e geração aleatória dos labirintos) estão todos contidos no arquivo .zip, onde também se encontra esse relatório além de um guia que explica como compilar e executar o trabalho.

DESCRIÇÃO DAS IMPLEMENTAÇÕES

Padrão da Entrada e Saída de Dados

Todas as 4 buscas utilizam o mesmo padrão de entrada: um arquivo inicial contendo, na primeira linha, o número de linhas e de colunas do labirinto e, nas linhas seguintes, o labirinto, fornecido segundo um padrão de caracteres. O caractere "$\#$" indica os obstáculos. O caractere "$\_$" indica os caminhos livres para serem percorridos. O caractere "I" indica a posição por meio da qual as buscas devem ser iniciadas. E, por fim, o caractere "F" indica a posição de destino/objetivo que deve ser alcançada.

Ao final de cada execução, um vetor de posições $i\,x\,j$ do labirinto é impresso na tela. Esse vetor representa um dos caminhos que foi encontrado

pela busca utilizada. Vale citar que, dependendo do algoritmo, esse caminho gerado não é necessariamente o melhor.

Depth First Search

Na Depth First Search(ou DFS) implementada no trabalho, foi necessário, antes de qualquer coisa, encontrar, por meio da função `depth_first_search()`, as posições de origem e destino contidas na matriz `lab`, gerada a partir do arquivo de entrada. Em seguida ambas as informações são passadas por parâmetro, para a função recursiva `depth_walk()`. Essa função irá percorrer o labirinto checando as 8 direções ao redor de cada posição analisada, verificando se é possível ir para ↑, para ↗, para →, para ↘,para ↓, para ↙, para ←, ou para ↖ , nessa ordem. Ao encontrar a primeira direção que leve a uma casa que exista e que não tenha sido processada ainda, ela é empilhada no vetor `path` e a função `depth_walk()` é novamente chamada para essa posição e assim, sucessivamente até que se chegue ao destino ou a um beco sem saída. Na ocorrência desse último evento, as posições que geraram esse caminho são retiradas de `path` na volta da recursão e a matriz `lab` é marcada com "`X`" nesses lugares. Por conta disso os custos com esse caminho não precisam ser contabilizados.

Quando o algoritmo chega recursivamente na posição de destino, a variavel `end_flag` é setada para `1`, sinalizando que um caminho foi encontrado com sucesso. Depois, mais dois eventos ocorrem: na volta da recursão, as posições que levaram de fato até o destino são marcadas com caracteres "`o`" e mantidas no vetor `path` e os custos vão sendo contabilizados pela variável `costs`. Ao final desse processo, já é possível a impressão do caminho, usando `path`, do custo necessário para percorrê-lo e o tempo decorrido.

Breadth First Search

Nessa busca, assim como na Depth First Search, a partir de uma posição inicial, encontrada pela função `init_point( )`, um dos 8 vizinhos da posição (↑, ↗, →,↘,↓,↙, ←, ↖) é escolhido para dar seguimento à caminhada. A função `breadth_walk( )` é responsável por analisar os vizinhos de uma posição e inseri-los em uma fila (enquanto a busca em profundidade utiliza uma pilha ou recursão), além de marcá-los com '0'. As

iterações do algoritmo continuam até que a fila esteja vazia ou a posição final tenha sido encontrada.

Best First Search

Bastante semelhante à busca em largura (breadth search), a `best_first_search( )` é uma busca informada, ou seja, que tem informação da posição final, a qual o algoritmo precisa encontrar. Utilizando as mesmas técnicas da busca em largura, a função, primeiramente, encontra as posições iniciais e finais do labirinto. Em seguida, a partir da posição inicial, avalia todos os seus vizinhos (↑, ↗, →, ↘, ↓, ↙, ←, ↖) e os insere em uma fila. A diferença consiste no fato de que, enquanto a breadth search utiliza uma fila comum, a best first (ou busca A) utiiliza uma fila de prioridades. O valor de comparação é uma heurística, que neste trabalho é o cálculo do quadrado da a distância euclidiana entre a posição e o ponto final. Para selecionar o próximo vizinho a ser percorrido, a fila de prioridade é desempilhada e o algoritmo continua a iteração até a fila estar vazia ou a posição final ter sido encontrada.

A* Search

Na A* Search, é calculada a distância euclidiana(heurística utilizada para essa busca) entre os próximos possíveis passos (↑, ↗, →, ↘, ↓, ↙, ←, ↖) até o ponto destino, o próximo ponto será o que possuir a menor distância.

Na implementação, as distâncias euclidianas são calculadas para todos os vizinhos ainda não visitados. Assim que definido o próximo espaço, um novo nó com os valores do vértice é criado na lista *`path`* e o valor da tabela é trocado por 4, indicando um espaço agora inválido.

O programa foi implementado de maneira iterativa e o algoritmo se repete até chegar no ponto destino.

PROCEDIMENTOS

Para analisar a performance de cada uma das 4 buscas, foram utilizadas as seguintes métricas: média dos tempos de execução, média dos custos, número médio de elementos que formam o caminho até o destino. Cada uma delas foi obtida considerando 5 tamanhos diferentes de labirintos: `5x5`, `10x10`, `20x20`, `50x50` e `150x150`. E Para cada um desses tamanhos, foram gerados 5 labirintos diferentes e de forma aleatória usando o código `random_maze.cpp`(contido na pasta mazes), totalizando 25 casos de teste para cada uma das 4 buscas.

Como o tempo, o custo do caminho gerado e o número médio de elementos que formam o caminho são calculados no próprio código fonte do trabalho, pôde-se calcular as 3 métricas da mesma forma: através da média aritmética das 5 execuções de cada tamanho de labirinto. Dessa forma foram obtidos 5 valores de média de tempo de execução, 5 de custos e 5 de quantidade de elementos no caminho.

ANÁLISE DOS RESULTADOS DAS EXECUÇÕES

Depth First Search

A tabela abaixo mostra os valores obtidos do procedimento descrito anteriormente para a Depth First Search.

	TAMANHOS				
	5x5	**10x10**	**20x20**	**50x50**	**150x150**
Média de custos	12.680	38.600	146.560	839.322	9888.800
Média de tempo decorrido (em ms)	0.242	0.260	0.280	0.502	2.435
Número médio de elementos que formam o caminho até o destino	12	36	128	740	8726

Nota-se por esses dados que a DFS tende a ser muito custosa a partir de labirintos com tamanho 20x20. Tanto isso é fato que ao aumentar expressivamente o tamanho, o custo aumentou exponencialmente. Isso é devido principalmente ao fato de essa busca não usar heurística, como é o caso da A* e Best-First. Por conta disso, ela produz um caminho muito extenso, algo que acaba inviabilizando o uso desse tipo de busca para problemas em que a prioridade é ter baixo custo e gerar um caminho curto. Quanto ao tempo decorrido, ela se mostrou como sendo a busca menos eficiente em todos os casos, com exceção dos labirintos `150x150`, em que ela teve um tempo médio de quase 1 milissegundo a menos do que os outros algoritmos. Embora a diferença se restrinja a casas decimais, a busca por largura teve todas as suas médias consideravelmente menores do que o método Best First Search, que teve o desempenho entorno de 0,2 segundos mais lento.

Breadth First Search

A tabela abaixo mostra os valores obtidos do procedimento descrito anteriormente para a Breadth First Search.

	TAMANHOS				
	5x5	**10x10**	**20x20**	**50x50**	**150x150**
Média de custos	12	69.440	271.200	2246.800	18900.600
Média de tempo decorrido (em ms)	0.066	0.099	0.136	0.664	3.026

Mantendo as piores métricas entre todas as buscas, a busca por largura é definitivamente a pior. A vantagem da busca em relação às buscas informadas é inexistente. Já com relação à busca em profundidade, ela só aparece quando a posição final está posicionada em posições contrárias à ordem de escolha dos vizinhos, em tabuleiros pequenos. Em todos os outros casos, obteve um desempenho pior. Pode servir para casos onde cobertura de área seja interessante.

Best First Search

A tabela abaixo mostra os valores obtidos do procedimento descrito anteriormente para a Best First Search.

	TAMANHOS				
	5x5	**10x10**	**20x20**	**50x50**	**150x150**
Média de custos	2.760	24.080	18.680	510.320	4481.760
Média de tempo decorrido (em ms)	0.0974	0.144	0.141	0.829	3.430

Obtendo os melhores resultados em comparação com as buscas não informadas, a best first search é consagrada por conseguir utilizar heurísticas que otimizam a procura pela solução do labirinto. Em todos os aspectos (utilização de memória, processamento e tempo), a best first possui apenas uma desvantagem: necessitar da posição final de antemão. Dispondo disso, pelos resultados obtidos, o custo pode ser reduzido em cerca de 4 vezes.

A* Search

A tabela abaixo mostra os valores obtidos do procedimento descrito anteriormente para a A* Search.

	TAMANHOS				
	5x5	**10x10**	**20x20**	**50x50**	**150x150**
Média de custos	3.840	9.800	18.800	54.880	167.520
Média de tempo decorrido (em ms)	0.219	0.243	0.289	0.291	0.420
Número médio de elementos que formam o caminho até o destino	4	9	16	45	138

Nota-se que existe uma relativa regularidade no comportamento da A*, até mesmo para casos mais complexos como o da matriz `150x150`, que teve um leve aumento de quase 0,2 milissegundos em relação às demais situações. Isso faz com que esse tipo de busca seja uma opção bem vinda em labirintos muito extensos, tanto em termos de tempo quanto de custos, já que ela sempre encontra o melhor caminho. Entretanto, para labirintos menores ela apresentou um desempenho semelhante a DFS e inferior às buscas BFS e Best-First, com uma diferença de aproximadamente de 0,15 e 0,1 milissegundos entre elas, respectivamente. Esse fator somado à sua implementação complexa, torna o seu uso, nesse contexto, desaconselhável.

Considerações Finais

Através da implementação dos 4 tipos de busca propostas no trabalho e da análise das métricas a elas associadas, foi possível medir e avaliar o desempenho e performance delas isoladamente em diferentes contextos, o que por consequência permitiu inferir em quais deles esses algoritmos são mais adequados. A busca A*, por exemplo se mostra mais rápida para labirintos muito grandes e também quando o objetivo em questão é o de encontrar o menor caminho, ou seja, aquele que tenha o menor custo. Mesmo assim, sua implementação tem maior complexidade. A BFS e Best-First, por outro lado, se mostraram boas opções em termos de tempo a se considerar quando o labirinto tem uma dimensão menor. No caso da Best-First ainda, ela produz caminho com custos bem menores que a BFS, em todos os casos de teste. Já a DFS, acabou por apresentar resultados piores em termos de tempo, custo e até número médio de elementos no caminho gerado para casos maiores. Entretanto teve desempenho razoável e comparável a A* para os menores, fator esse que somado à sua implementação de menor complexidade faz com que ela seja uma opção de algoritmo válida.