Relatório de Buscas

1. Introdução

:Definição:

Um problema de busca é definido por um conjunto de estados, um estado inicial, um estado final, um função boleana que nos indica se um estado é um estado final, ou não, e uma função que para um estado nos mapea um conjunto de novos estados, ou rejeita outros se equivalentes.

:Descrição:

Um problema de busca é representado por uma relação binária. Considerando R como uma relação binária e T uma máquina de Turing, então T calcula R se:

- Se $\exists x, y$ R(x,y) $\rightarrow$ T aceita x;
- Se $\forall x, y$ $\neg$R(x,y) $\rightarrow$ T não aceita x.

:Principais métodos utilizados para resolver problemas:

Os principais métodos de busca são a busca não informada e a busca informada.

Busca não informada:

Busca em Largura(BFS)

Busca de Custo Uniforme

Busca em Profundidade(DFS)

Busca Limitada em Profundidade

Busca em Profundidade Iterativa(IDFS)

Busca Biderecional

Busca informada:

Gulosa(Greedy)

A*

Estas estratégias são classicadas quanto à sua complexidade(tempo e memória necessários), completude (se

encontra a solução para o objetivo) e otimilidade (se encontra a melhor solução para um problema).

2. Estratégias de Procura

a) Procura não guiada:

i. DFS

Neste caso em específico, o DFS retorna o caminho para a solução recursivamente em profundidade, seguindo por um caminho “cego” até chegar a uma configuração sem mais filhos e retornar String vazia) ou até“”(chegar a uma configuração igual à configuração final esperada.

Para evitar que o algoritmo siga por ciclos, foi implementado um HashSet que guarda as configurações já visitadas.

Como podemos verificar com os resultados obtidos, o DFS não é um algoritmo eficaz para este problema, pois segue um caminho aleatório, ou seja, acaba por ter complexidade espacial e temporal altas e não retorna um caminho propriamente ideal. Complexidade temporal de O (|V| + |E|) onde |V| é o número de vértices e |E| o número de arestas. Quando à complexidade espacial é de O (|V|) onde de novo | V| é o número de vértices.

ii. BFS

O BFS, ao contrário do DFS retorna uma solução ideal e percorre os nós a partir do nó raiz, analisando os filhos todos da raiz antes de passar para o nivel abaixo e repete o processo
até encontrar o resultado esperado ou até chegar ao nó mais à direita do nivel de profundidade máxima.

Como no DFS, foi usado um HashSet para evitar ciclos e a complexidade é a mesma que o DFS.

ii. IDFS

A pesquisa iterativa limitada em profundidade ou IDFS é diferente dos restantes algoritmos pois pode percorrer a árvore várias vezes com profundidades diferentes, começando de 0 até a uma profundidade limite, até encontrar a solução ou chegar ao limite de profundidade.

Este algoritmo retorna a solução ideal e tem uma complexidade temporal e espacial menor que o DFS.

(b) Procura guiada:

- Para a pesquisa guiada usamos heurísticas para ajudar as decisões dos algoritmos a cada iteração. Uma heurística é uma estratégia usada onde parte da informação é ignorada de modo a tentar tornar o processo ou algoritmo mais rápido. É usado numa tentativa de poupar memória e tempo na execução de um algoritmo. Existem várias heurísticas, mas para este projeto foi sugerido o uso de duas, a do somatório das peças fora de lugar e a *Manhattan Distance.*
- *A pesquisa gulosa ou greedy é um tipo de pesquisa guiada que tenta resolver o problema proposto fazendo a escolha mais acertada mas não olha para o panorama todo,visto que define o peso de cada nó com a heuristica e segue sempre pelo de menor peso.*
- *Já o A* não só se baseia na heuristica para determinar o peso de cada nó como também na profundidade do nó que segue.*

→ Os algoritmos de procura guiada são de facto melhores em termos de complexidade e de tempos de execução, mas nem sempre retornam a solução ideal pois a escolha do momento pode levar a possibilidades desfavorecedoras.

1. **Descrição da Implementação**

Neste projeto utilizamos a linguagem Java porque neste trabalho era necessário criar classes o que é mais fácil com esta. Também em Java dispomos de várias APIs que facilitam a

implementação dos algoritmos, pois são mais fáceis de usar em Java.

Neste programa usamos algumas APIs do Java, sendo estas: HashSets(DFS, IDFS, BFS, A*), Queues(BFS), Priority Queues(Greedy, A*), Scanners(função *main*) e outras mais usuais tais como *java.util.Arrays* e *java.lang.Math*, por exemplo.

Tal como referido anteriomente criamos também uma class *Node* que difere do Node do java em alguns atributos.

1. Resultados

Configuração inicial de teste: **6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15**

Configuração final de teste: **14 6 12 9 7 2 5 11 8 4 13 15 3 0 10 1**

Estratégia	Tempo(seg)	Espaço(MB)	Profundidade
DFS	228.4	45.2	80
BFS	2.3	4.1	10
IDFS	1.70	0.56	10
Greedy	1.35	0	10
A*	1.48	0	10

2. Conclusão

Apesar de todos os métodos chegarem a uma solução para esta configuração, houve dois que se destacaram, o Greedy e o A*, ambos chegaram a um resultado sem utilizar espaço em memória. O Greedy foi considerado o melhor algoritmo para esta configuração porque conseguiu ser mais rápido do que o A*. Denota-se que o DFS é bastante ineficiente, apenas conseguiu chegar a um resultado passados 228 segundos após o algoritmo começar a procura com uma profundidade de 80 passos.