UNIVERSIDADE DE SÃO PAULO – USP
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO

TORRES DE HANÓI

Alunos:
Adams Vietro Codignotto da Silva - 6791943
Ana Clara Kandratavicius Ferreira - 7276877
Frederico Facco - 8532206

São Carlos
2014

1 Introdução

Neste trabalho, iremos resolver o famoso problema das Torres de Hanói, utilizando uma representação em grafo e técnicas aprendidas em IA.

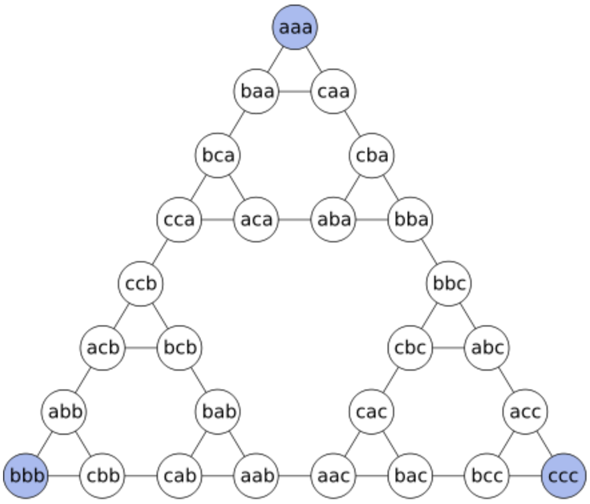
2 Modelagem do problema

A meta é obter o número mínimo de movimentos. O número mínimo pode ser obtido utilizando recorrência.

Tomando n como número de discos, a, b e c como os pinos e $H(n,a,b,c)$ como a quantidade de movimentos para passar do pino a para o pino b, utilizando o auxiliar c, temos:

$$H(1,a,b,c) = 1 \quad (a \rightarrow c)$$
$$H(n,a,b,c) = H(n-1,a,b,c), \quad (a \rightarrow b), H(n-1,c,b,a)$$

Podemos dizer então que para um número n de discos, podemos gerar um grafo contendo todas as possibilidades, com 3^n nós.



Grafo de possibilidades com 3 discos

2.1 Recorrência

Podemos dividir o problema da Torre de Hanói de acordo com o número de discos:

- Para n=1, basta 1 movimento
- Para n=2, precisamos de 3 movimentos (trivial)
- Para n=3, podemos resolver o problema para 2 discos (3 movimentos), mover o disco maior para o pino restante (1 movimento), e movemos os outros 2 discos para o pino final (3 movimentos) totalizando 7 movimentos.
- Para n=4, Resolvemos o problema para três discos(7 movimentos), depois movemos o maior disco (1 movimento), após isso trazemos os três discos que já estão no outro pino para cima do maior disco (7 movimentos), totalizamos 15 movimentos.

Podemos perceber que temos a seguinte seqüencia:

- $1 = 2^1 - 1$
- $3 = 2^2 - 1$
- $7 = 2^3 - 1$
- $15 = 2^4 - 1$

Ou seja, temos $2^n - 1$ movimentos necessários para uma quantidade n de pinos.

3 Implementação

O problema foi implementado na linguagem C. Para compilação, pode ser usado o comando make all no terminal de um sistema Linux, utilizando o arquivo makefile presente, e executar utilizando o comando make run. No Windows, uma alternativa é utilizar um compilador como o CodeBlocks, criar um projeto, incluir todos os arquivos .c e .h e compilar normalmente.

3.1 Entrada

Como entrada, apenas é preciso digitar o número de discos a serem utilizados (variando de 1 a 20). Mais que 20 discos são possíveis, mas devido a grande quantidade de memória e processamento utilizado, não é recomendável, além de tornar o programa instável.

3.2 Saída

A saída exibirá a quantidade de estados (nós) que o BFS e o DFS visitaram, e o caminho que ambos encontraram. Após isso, irá mostrar os estados a serem executados pelo A*, ou seja, os vértices visitados do grafo, onde haverão k linhas representando os k-1 movimentos (pois a primeira é o estado inicial) com n números variando de 1 a 3, onde n é a quantidade de discos desejados. Cada número representa em qual pino o disco deve estar presente na jogada k, ou seja, uma saída *1 3* quer dizer que o disco 1 deve estar no pino 1 e o disco 2 deve estar no pino 3.

3.3 Estruturas

Para representar tal grafo, foi utilizado uma representação de grafo com listas de adjacências para conservar memória.

Também foram implementadas estruturas adicionais necessárias, como uma fila de prioridade e uma pilha.

3.4 Buscas Cegas

Utilizamos o algoritmo de buscas DFS e BFS como buscas cegas, pois são métodos mais comuns e simples de serem implementados, além de percorrerem todo o grafo.

3.5 Heurísticas

Para percorrer o grafo eficientemente, utilizamos o algoritmo A*. Como heurísticas, utilizamos duas funções:

- Um contador de movimentos, onde a cada estado que passamos é incrementado.
- A distância do estado até o estado desejado, onde o peso atribuído ao nó é a quantidade de discos que ainda não estão na posição final (no terceiro pino)

4 Experimentos e Resultados

O algoritmo A* se mostra bem próximo do ideal, como podemos observar pela tabela abaixo, onde mostra a quantidade de nós visitados.

Número de Discos	BFS	DFS	A*	Ideal
1	1	1	1	1
2	6	8	3	3
3	24	26	8	7
4	70	74	15	15
5	232	220	25	24

Porém, o A* necessita de quase 3 vezes mais memória que o DFS ou o BFS, pois necessita guardar o peso de todos os vértices, bem como seus antecessores e se já foram visitados.

5 Conclusões

Para este problema, em particular para o modo em que foi modelado, o algoritmo DFS sempre encontrará a solução ideal. Porém, o algoritmo A* se mostrou muito mais eficiente em questão de nós visitados, mesmo às vezes não exibindo a solução ideal.

Talvez o uso de outra linguagem orientada a objeto, como C++ ou Python, tornaria o código mais limpo e de fácil entendimento. Mas como todos os integrantes possuem maior afinidade com C, esta foi a linguagem escolhida.