Universidade Federal de São João del-Rei

Trabalho Prático para disciplina de Grafos

Problema das 8 rainhas.

Higor Alves

São João del Rei
2019/Novembro

O Problema

O problema das N Rainhas é um quebra-cabeça clássico que pergunta “quantas maneiras diferentes você pode encaixar N rainhas em um tabuleiro de xadrez N * N sem que nenhuma delas esteja em posição de atacar uma à outra?”. Além de ser um excelente quebra-cabeças, também é muito útil em ciência da computação ao demonstrar técnicas de otimização e complexidade de tempo em programas.

Abordagem

Optei por iniciar esse problema escrevendo soluções das quais não se usa um grafo para que os conceitos do quebra cabeça fossem praticados, escrevendo várias funções que testaram quais posições em um determinado tamanho de tabuleiro *N* onde *R* rainhas possam ser colocadas em um determinado local no tabuleiro sem criar conflitos. Para resolver foi usado recursão junto com o conceito de conjuntos independentes, para achar onde se poderia colocar cada rainha no tabuleiro, verificamos dentro da modelagem do grafo quais dos nossos vértices são caracterizados pelo conceito apresentado, assim ele retorna e plota "Graficamente" dentro do nosso tabuleiro mostrando as posições das quais se caracterizam como resposta para o problema.

Funções de ajuda

Foi criada uma função de ajuda chamada:

createChess()

Esta função nos ajuda a criar um grafo no tamanho *N* por *N* já preenchendo todas as condições entre os vértices de um tabuleiro, resultando numa matriz de *N por N* onde podemos verificar as ligações dos vértices, para esta criação fazemos uso de dois laços de repetição.

Função de Solução

A função usada para desenvolver a solução usando por meio de um conjunto solução foi:

conjIndependente()

Nela chamamos a função ***tentativaConjunto*** para tentarmos verificar se um vértice se enquadra ou não na nossa solução, ou seja se a e b são vértices quaisquer de um conjunto independente, não há aresta entre a e b, esta função é recursiva para que possamos achar os vértices necessários e depois concatenando os vértices num vetor de soluções, caso seja verdade nos retorna o *index* do array onde a rainha pode ser colocada, se não for possível nos retorna o valor falso, após executar essa função pelo tamanho da matriz temos

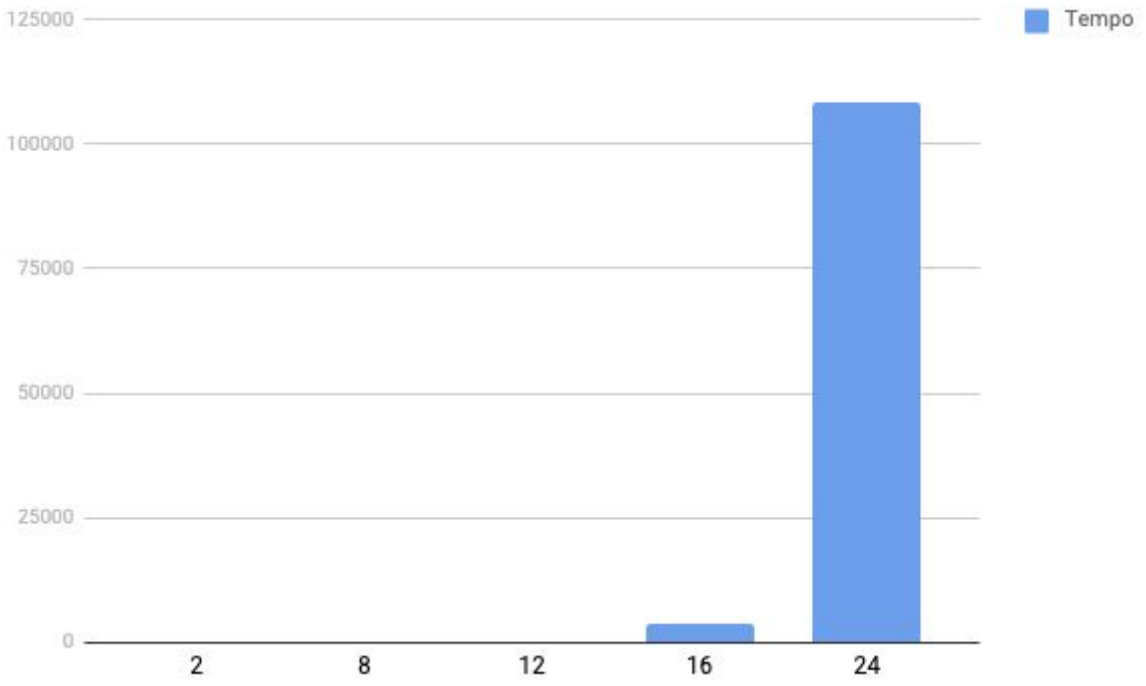
todas as soluções possíveis para um determinado tabuleiro, estes dados são jogados dentro de um vetor de posições para que possamos plotar graficamente nosso tabuleiro e as posições onde as rainhas podem aparecer sem conflitos.

Análise dos resultados

Nesta seção apresento alguns resultados executando o algoritmo implementado com tamanhos de tabuleiro diversos, estes são:

- 2
- 8
- 12
- 16
- 24

O gráfico abaixo apresenta o tempo em segundos gasto por cada tabuleiro de tamanho *N.*



Podemos analisar o tempo gasto como algo exponencial ou seja quanto maior for o tabuleiro maior será o tempo gasto para achar a solução com o algoritmo implementado. Verificamos também que devido a ser uma linguagem de alto nível temos um maior aumento no tempo gasto que mesmo algoritmo implementado em uma linguagem de baixo nível, devemos observar que a resolução utilizando a modelagem de grafos é mais custosa neste algoritmo, do que utilizar um que faça as verificações em tempo real, esta comparação pode ser realizada executando o arquivo **not-graph-sol.js**, onde que com um tabuleiro 10 por 10 já visualizamos uma enorme diferença de em média 10 segundos.

A partir da análise dos resultados apresentados aqui, notamos que usando um algoritmo exponencial não é a melhor solução para resolver o problema das N Rainhas, pois quanto maior o tabuleiro mais tempo nosso código leva para achar uma solução, na literatura é possível achar resoluções mais rápidas e em outras linguagens, por exemplo a linguagem C.

Limitações da solução

Grandes limitações foram percebidas tanto pela linguagem quanto pela resolução adotada, devido nosso algoritmo ser exponencial, a partir de um tamanho de tabuleiro de 13 por 13, encontramos uma demora enorme na resolução do problema levando a realizar somente o teste com um tamanho de tabuleiros limitados. Um algoritmo mais otimizado usando a mesma linguagem poderia ser encontrado para diminuir o tempo exponencial da solução, ou até mesmo executar paralelismo nas chamadas recursivas.

Conclusão

Concluímos que podemos implementar de maneira simples uma solução se baseando nos conceitos básicos de grafos, para acharmos mais de uma solução para um mesmo tamanho de tabuleiro, mostrando como é importante a forma que estruturamos nosso grafo e a função solução.