Solução eficiente para a resolução do Problema das N-Rainhas

Marco Fernandes, Miguel Alho

***Resumo* – Existem muitas formas de resolver o Problema das N-Rainhas, desde métodos de "força bruta" a técnicas de "backtracking", sendo que estas são ineficientes para grandes valores de N. Neste artigo apresentamos uma solução simples e eficiente que funciona na maioria dos casos.**

***Abstract* – There are several ways of solving the N-Queens Problem, ranging from "brute force" techniques to backtracking, even though these tend to be computationally inefficient. In this article we present a simple solution that works in most cases.**

I. Introdução

À partida, o Problema das N-Rainhas pode parecer um simples jogo: como colocar N rainhas num tabuleiro de xadrez, com N linhas e N colunas, sem que qualquer uma possa atacar outra, dado que, no xadrez, as rainhas têm liberdade de movimentos na horizontal, vertical e diagonal. Na realidade, este problema corresponde à modelação de um problema de cobertura máxima [3]: a solução garante que cada rainha possa ser acedida por qualquer das principais direcções em sua volta (verticais, horizontais e diagonais) sem que haja qualquer interferência. Dito de outra forma, não pode haver mais que uma rainha em qualquer coluna, linha ou diagonal.

As aplicações para um algoritmo deste tipo são vastas, desde testes de VLSI, controlo de tráfego aéreo, sistemas de comunicação, encaminhamento de dados e controlo de carga em computadores multiprocessador, gestão de recursos computacionais, processamento óptico paralelo, compressão de dados, entre outras [4].

Nas secções seguintes, será feita uma sucinta abordagem histórica ao problema, descrita uma solução recursiva, apresentada a solução proposta e avaliada a sua eficiência.

II. Problema

O problema das N-Rainhas, inicialmente proposto para apenas 8 rainhas, foi introduzido por um jogador de xadrez de nome Max Bazzel em 1848 e publicado em 1850 no *Illustrierte Zeitung* [4]. Desde então, captou a atenção de muitos matemáticos e, mais recentemente, tem sido discutido no âmbito das ciências computacionais, sendo usado como exemplo de algoritmos de "backtracking", de geração de permutações, de redes neuronais, de problemas de satisfação de restrições e de paradigmas de "divisão e conquista", entre outros [4].

Há diversas abordagens para a resolução do problema, entre as quais:

- "Força bruta" – uma solução possível é, obviamente, testar todas as combinações possíveis de N rainhas em N^2 posições. Mesmo para um valor de N igual a 10, as combinações possíveis são superiores a 10^{13}. Uma simplificação possível é considerar que, após posicionar uma rainha, o tabuleiro restante fica uma dimensão mais pequeno (retirando a coluna e linha da rainha escolhida). Deste modo, as combinações possíveis são reduzidas a N!, o que representa ainda um grande esforço computacional ($10! > 3x10^6$).
- Abordagem recursiva com "backtracking" [5] – colocam-se as rainhas, uma de cada vez e uma por coluna, verificando se cada uma não é atacada. Se for, move-se a rainha para a linha seguinte. Caso já não existam posições seguras, volta-se à rainha da coluna anterior e procura-se nova posição. Se a rainha estiver segura, continua-se o processo na primeira linha da coluna seguinte. Este é, provavelmente, o algoritmo mais popular, já que tem uma eficiência aceitável e é relativamente simples de implementar.
- Algoritmo permutativo [4] – colocam-se as N rainhas e efectuam-se sucessivas permutações, verificando se a sua disposição é (ou não) uma solução válida. Tem semelhanças com o primeiro método, embora faça a restrição de uma rainha por coluna. Uma variante é colocar inicialmente as rainhas em posições com conflitos mínimos (mas possíveis) e, posteriormente, fazer permutações sucessivas até encontrar uma solução.

III. Algoritmo Recursivo

Numa primeira abordagem a este problema, no âmbito de um projecto da disciplina de Computação Gráfica do 5º ano da L.E.E.T., foi-nos sugerido o uso de uma solução recursiva para o problema. Na realidade, era pedido para apenas apresentar uma solução para o problema (que deveria ser representada graficamente com o OpenGL). Como verificado anteriormente, uma abordagem de força bruta seria possível, mas extremamente ineficiente, ainda que o objectivo fosse apenas demonstrar uma solução. Pretendia-se obter soluções para valores de N

relativamente elevados e com um tempo de cálculo reduzido.

Assim, foi criada uma função para colocar rainhas, de forma segura, numa determinada coluna e a partir de uma dada linha:

```
bool Place(int coluna, int linha)
{
    bool colocado = false;
    do {
        if(Seguro(coluna,linha)) {
            posicoes[coluna] = linha;
            colocado = true;
            if (coluna < N_RAINHAS-1)
               colocado = Place(coluna+1,0);
            if (colocado)
               return colocado;
            else
               posicoes[coluna] = -1;
        }
        linha++;
    } while(linha < N_RAINHAS);
    return false;
}
```

E uma função auxiliar para validar uma colocação:

```
bool Seguro(int coluna, int linha)
{
    for(int j=0; j<coluna; j++) // linha
        if(posicoes[j]==linha)
            return false;
    if(linha && coluna) // diagonal positiva
        for(int col = coluna-1,row = linha-1;
                    row+1&&col+1;col--,row--)
            if(posicoes[col]==row)
                    return false;
    if(coluna) // diagonal negativa
        for(int col = coluna-1,row = linha+1;
            row<N_RAINHAS&&col+1;col--,row++)
            if(posicoes[col]==row)
                    return false;
    return true;
}
```

Desenvolvidas estas funções, o cálculo de uma solução para o problema é feito simplesmente da forma:

```
Place(0,0);
```

A colocação de rainhas com este algoritmo está demonstrado na Figura 1.

Embora o algoritmo funcione, o tempo de cálculo cresce exponencialmente com N, apresentando largos minutos de processamento para valores superiores a 30.

IV. SOLUÇÃO MAIS EFICIENTE

Para valores de N elevados a solução recursiva torna-se bastante lenta. Pretendia-se obter (ainda) mais velocidade

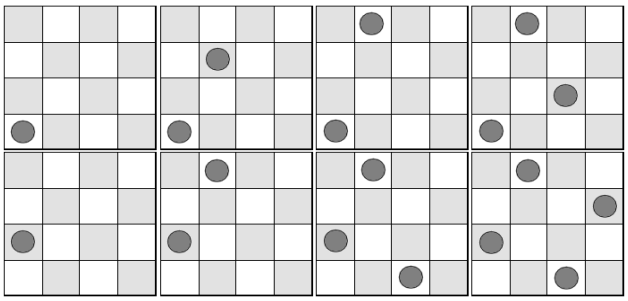

Figura 1 – Sequência de posições para N=4

e eficiência. Procurou-se, então, verificar se havia alguma propriedade da qual se pudesse tirar partido para simplificar os algoritmos. Para isso, partiu-se da solução para N = 4 e procuraram-se soluções que tivessem algum tipo de semelhança ou padrão para tabuleiros maiores.

Uma característica comum a algumas soluções de vários tabuleiros é a de as primeiras rainhas estarem dispostas espaçadas de 2 linhas até atingir o topo, estando a primeira colocada na 2ª linha. Se se adoptar uma estratégia de colocação inicial destas rainhas, um algoritmo recursivo pode trabalhar com apenas aproximadamente metade do tabuleiro. Tirando partido desta característica, reduz-se drasticamente o tempo de processamento. Esta redução é tanto maior se se tiver em conta que, usualmente, esta disposição origina uma disposição idêntica (quase diagonalmente simétrica) nas rainhas restantes.

Apesar de útil, é necessário ter em conta que nem sempre é possível aplicar esta técnica. De facto, existem excepções: N = 9, 15, 21, 27, 33, ..., ou seja, N=6k+3, k=1,2,3,... Para estes casos, é necessário adaptar o algoritmo de forma a que sejam também processados correctamente.

Quando não é possivel implementer o procedimento particular volta-se a aplicar o algoritmo recursivo, fazendo uso da capacidade de “backtracking” para voltar para as colunas anteriores quando necessário. O algoritmo usado é então o seguinte:

```
void Calcular()
{
    for(unsigned c=0;c<N_RAINHAS;c++)
        posicoes[c] = -1;
    // primeiras, de 2 em 2
    for(int linha=1,coluna=0;
        linha<N_RAINHAS; linha+=2,coluna++)
           posicoes[coluna] = linha;
    // processo restante
    if(!Place(coluna,0))
        while(!Place(coluna-1,
            posicoes[coluna-1]+1))
            coluna--;
}
```

Não é necessário testar se `posicoes[coluna-1]+1` é menor que N, já que para N=6k+3 isso se verifica sempre (N é ímpar e posição seguinte – [(2r-1)+1] – é par) e, para esses valores de N, a primeira chamada à função `Place` retorna sempre `true`.

V. Resultados

Os tempos de execução, obtidos num Pentium IV, a 2.6GHz e com 512 MB de memória, para valores de N até 27, encontram-se na Tabela 1 e na Figura 2.

Da análise do gráfico/tabela, pode-se concluir que o método é bastante eficaz. No pior caso, o algoritmo desenvolvido demora menos de meio segundo para determinar as posições das rainhas. Apenas num caso pontual (N=21 – uma das excepções onde o algoritmo não é directo) se revela mais lento. Testámos o nosso algoritmo até N=39 (também caso particular de resto de divisão 3), valor para o qual o programa não obteve uma solução num tempo razoável.

(6k+3 ↓)

	4	**5**	**6**	**7**	**8**	**9**
MM	$1{,}1\times10^{-6}$	$2{,}0\times10^{-6}$	$2{,}3\times10^{-6}$	$3{,}9\times10^{-6}$	$4{,}7\times10^{-6}$	$1{,}5\times10^{-4}$
AT	$4{,}5\times10^{-6}$	$2{,}7\times10^{-6}$	$3{,}4\times10^{-5}$	$9{,}2\times10^{-6}$	$2{,}1\times10^{-4}$	$8{,}6\times10^{-5}$
	10	**11**	**12**	**13**	**14**	**15**
MM	$7{,}7\times10^{-6}$	$1{,}1\times10^{-5}$	$1{,}2\times10^{-5}$	$1{,}6\times10^{-5}$	$3{,}7\times10^{-5}$	$9{,}0\times10^{-4}$
AT	$2{,}7\times10^{-4}$	$1{,}5\times10^{-4}$	$9{,}6\times10^{-4}$	$4{,}5\times10^{-4}$	$9{,}4\times10^{-3}$	$7{,}5\times10^{-3}$
	16	**17**	**18**	**19**	**20**	**21**
MM	$2{,}4\times10^{-5}$	$3{,}1\times10^{-5}$	$3{,}3\times10^{-5}$	$4{,}1\times10^{-5}$	$4{,}3\times10^{-5}$	$1{,}7\times10^{-1}$
AT	$6{,}3\times10^{-3}$	$3{,}7\times10^{-3}$	$3{,}2\times10^{-2}$	$2{,}2\times10^{-2}$	1,9	$9{,}4\times10^{-2}$
	22	**23**	**24**	**25**	**26**	**27**
MM	$5{,}5\times10^{-5}$	$6{,}6\times10^{-5}$	$6{,}9\times10^{-5}$	$8{,}2\times10^{-5}$	$9{,}2\times10^{-5}$	$4{,}2\times10^{-1}$
AT	19,1	$3{,}1\times10^{-1}$	5,3	$6{,}7\times10^{-1}$	6,0	7,37

Tabela 1 - Tempos de computação das posições em segundos (MM - algoritmo eficiente; AT- algoritmo recursivo tradicional)

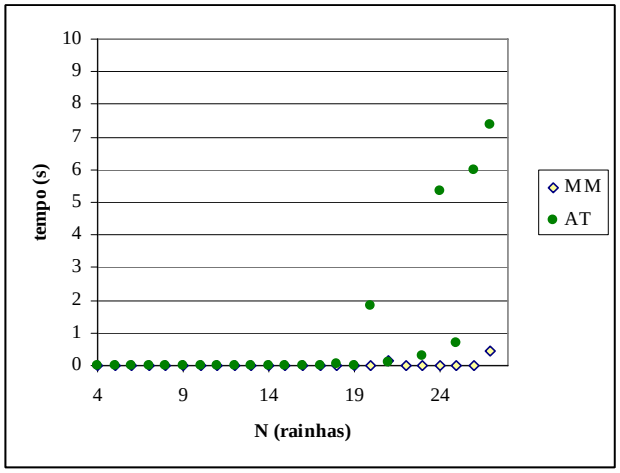

Figura 2 – Comparação gráfica dos algoritmos (para N=19, o ponto do algoritmo tradicional foi omitido)

VI. Conclusões

Efectivamente, o algoritmo funciona muito bem para os casos onde N é diferente de 6k+3, com k=1,2,3,... O programa apenas nos dá uma solução, mas esse era, de facto, o objectivo inicial. Mesmo para valores de N consideráveis, com este algoritmo é possível encontrar uma solução sem o recurso a um computador, de uma forma simples e eficaz.

Aquando da pesquisa de informação para a escrita deste artigo, foi descoberta a publicação [1] de um algoritmo semelhante ao que aqui descrevemos. O autor desse artigo descreve esta solução para um caso particular (N=30) e afirma que resolve para todos os casos em que o resto da divisão de N por 6 é 0,1,4 ou 5. Para resto 2 (N=6k+2=2r) existe [2] uma solução semelhante:

- inicia-se no canto (0, 0) e coloca-se a rainha da coluna seguinte 2 linhas acima: (1 , 2), (2, 4) ... (r-2, N-4), (r-1 , N-2) ;
- permutam-se as rainhas nas casas (0, 0) e (r-2, N-4) para (r-2, 0) e (0, N-4), respectivamente;
- a outra metade do tabuleiro é preenchida por via de uma simetria axial (rotação de 180º em torno do exio normal ao tabuleiro).

Para o caso de N=6k+3, a solução é a mesma que para o caso de resto 2 com a rainha N colocada no ultimo canto (N-1, N-1). Assim é possivel obter uma solução rápida para todos os N possíveis.

Existem ainda outras soluções eficientes e que funcionam para valores de N extremamente elevados [3] de forma muito rápida.

Agradecimentos

Agradece-se ao Prof. Joaquim Madeira o incentivo para a escrita deste artigo e os comentários durante a sua elaboração.

Referências

[1] Kenneth Evans, “The N Queens Problem” , OR Newsletter, Jan. 97, pp13-14.

[2] J. Beasley, “The Mathematics of Games”, Oxford U. Press, New York, 1989.

[3] Rok Sosic and Jun Gu, “Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem” , IEEE Transactions on Knowledge and Data Engineering, Vol. 6. , No 5, October 1994, pp 661-668.

[4] C. Erbas, S. Sarkeshik and M. M. Tanik, “Different Perspectives of the N-Queens Problem”, Proc. of the 1992 ACM Annual Conference on Communications, April 1992, pp 99-108.

[5] F. Carrano, P. Helman and R. Veroff, “Data Abstractions and Problem Solving with C++ — Walls and Mirrors”, Addison-Wesley, 1998.