AED III
Backtracking

Backtracking

Um algoritmo de backtracking começa com uma solução vazia e amplia a solução passo a passo. A pesquisa recursivamente passa por todas as formas diferentes de como uma solução pode ser construída.

Sudoku

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Sudoku

```
#include <bits/stdc++.h>
 // UNASSIGNED é usado por células vazias
#define UNASSIGNED 0

// N é usado para o tamanho do Sudoku. Será NxN
#define N 9

// Esta função encontra uma entrada no Sudoku que não foi numerada
bool FindUnassignedLocation(int grid[N][N], int &row, int &col);

// Checa se é permitido colocar um determinado número na célula encontrada
bool isSafe(int grid[N][N], int row, int col, int num);
```

```
/* Faz um preenchimento, para atribuir valores para todas as células não numeradas, verificando os requisitos para solução do Sudoku (não duplicação nas linhas, colunas e matrizes 3x3) */
bool SolveSudoku(int grid[N][N])
{  int row, col;
// Se não há célula vazia, sucesso!
   if (!FindUnassignedLocation(grid, row, col))
      return true; // successo!
   // considerando dígitos de 1 a 9
   for (int num = 1; num <= 9; num++)
   {
      // if looks promising
      if (isSafe(grid, row, col, num))
      {
         // faz tentativa de preenchimento
         grid[row][col] = num;
          // retorna, se deu certo, prossiga!
         if (SolveSudoku(grid))
            return true;
          // falha, desfaz a inserção e tenta novamente
         grid[row][col] = UNASSIGNED;
      }
   }
   return false; // isto engatilha o backtracking
}
```

```
/* Procura uma célula que não foi preenchida. Se for encontrada, linha e
coluna são retornadas nos ponteiros */

bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
{
   for (row = 0; row < N; row++)
      for (col = 0; col < N; col++)
         if (grid[row][col] == UNASSIGNED)
            return true;
   return false;
}
```

```
// Retorna um booleano que indica se um número já foi utilizado em uma linha específica
bool UsedInRow(int grid[N][N], int row, int num)
{
   for (int col = 0; col < N; col++)
      if (grid[row][col] == num) return true;
   return false;
}
// Retorna um booleano que indica se um número já foi utilizado em uma coluna específica
bool UsedInCol(int grid[N][N], int col, int num)
{
   for (int row = 0; row < N; row++)
      if (grid[row][col] == num) return true;
   return false;
}
// Retorna um booleano que indica se um número já foi utilizado em uma matriz 3x3 específica
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num)
{
   for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++)
         if (grid[row+boxStartRow][col+boxStartCol] == num) return true;
   return false;
}
```

```
/* Retorna um booleano que indica se será permitido colocar um número em uma
determinada célula */

bool isSafe(int grid[N][N], int row, int col, int num)
{
   /* Checa se num já não está na linha definida, na coluna definida ou na matriz 3x3
correspondente */
   return !UsedInRow(grid, row, num) &&
          !UsedInCol(grid, col, num) &&
          !UsedInBox(grid, row - row%3 , col - col%3, num);
}

/* Função para imprimir a matriz toda  */
void printGrid(int grid[N][N])
{
   for (int row = 0; row < N; row++)
   {
      for (int col = 0; col < N; col++)
           printf("%2d", grid[row][col]);
       printf("\n");
   }
}
```

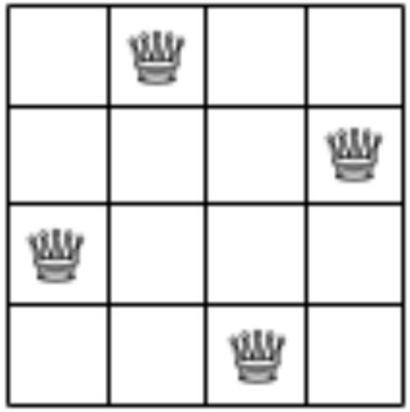
```
int main()
{
   // 0 significa células não numeradas
  int grid[N][N] = {{3, 0, 6, 5, 0, 8, 4, 0, 0},
                    {5, 2, 0, 0, 0, 0, 0, 0, 0},
                    {0, 8, 7, 0, 0, 0, 0, 3, 1},
                    {0, 0, 3, 0, 1, 0, 0, 8, 0},
                    {9, 0, 0, 8, 6, 3, 0, 0, 5},
                    {0, 5, 0, 0, 9, 0, 6, 0, 0},
                    {1, 3, 0, 0, 0, 0, 2, 5, 0},
                    {0, 0, 0, 0, 0, 0, 0, 7, 4},
                    {0, 0, 5, 2, 0, 6, 3, 0, 0}};
if (SolveSudoku(grid) == true)
         printGrid(grid);
   else
        printf("No solution exists");

   return 0;
}
```

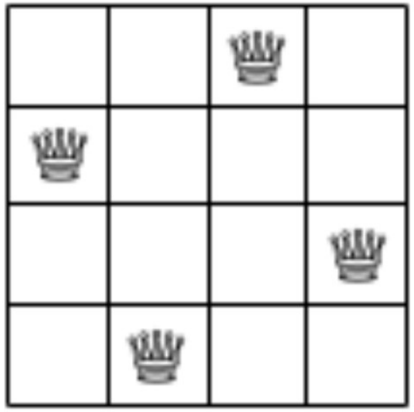
Problema das N-Rainhas

Como exemplo, considere o problema de calcular o número de maneiras em que as rainhas podem ser colocadas em um xadrez n × n para que nenhuma rainha se ataque. Por exemplo, quando n = 4, existem duas soluções possíveis: um algoritmo de backtracking começa com uma solução vazia e amplia a solução passo a passo. A pesquisa recursivamente passa por todas as formas diferentes de como uma solução pode ser construída.

Backtracking

Como exemplo, considere o problema de calcular o número de maneiras em que as rainhas podem ser colocadas em um xadrez n × n para que nenhuma rainha se ataque. Por exemplo, quando n = 4, existem duas soluções possíveis:



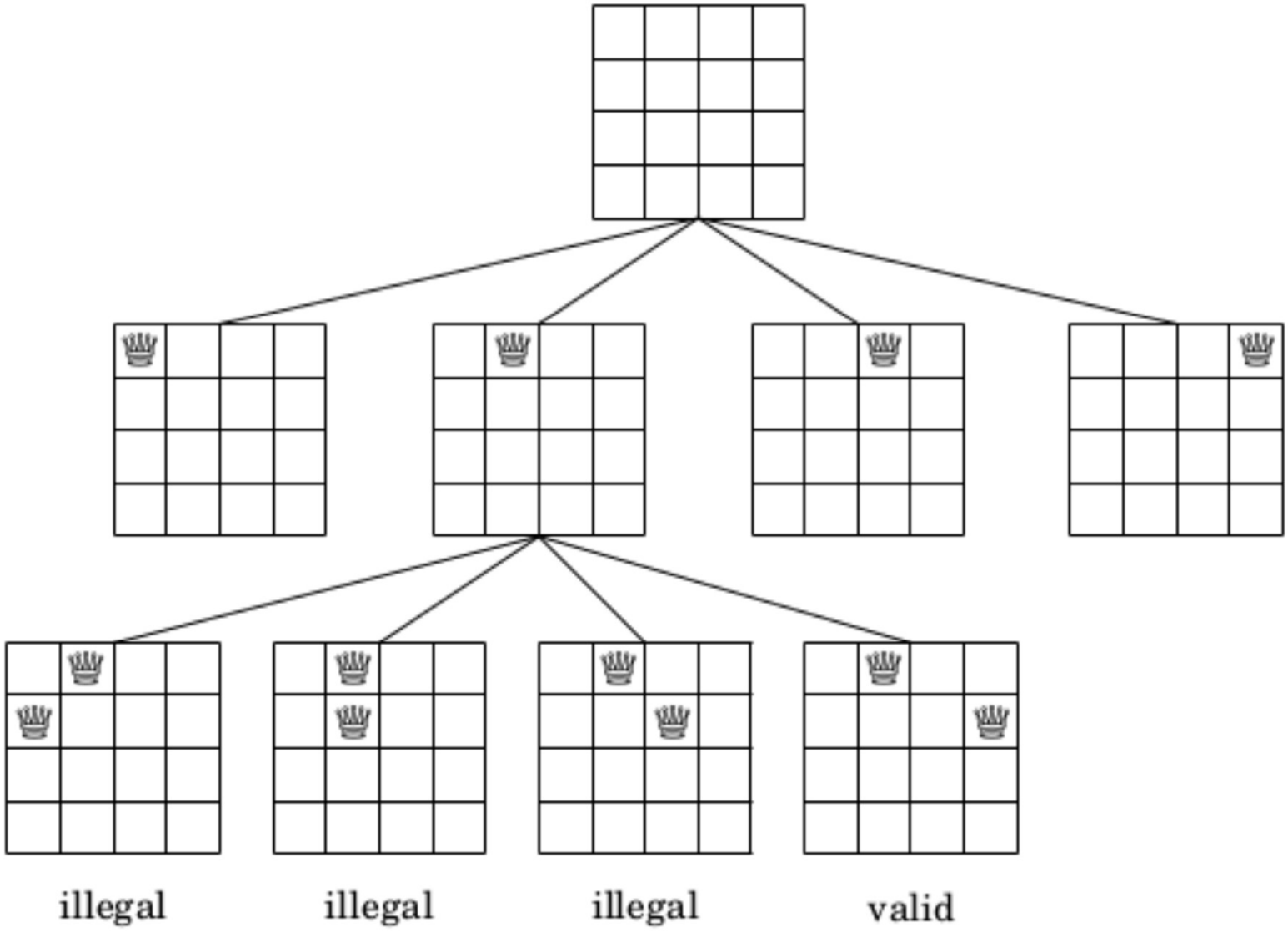


Backtracking

O problema pode ser resolvido usando backtracking colocando rainhas na linha da placa por linha. Mais precisamente, exatamente uma rainha será colocada em cada linha para que nenhuma rainha ataque qualquer das rainhas colocadas antes. Uma solução é encontrada quando todas as n rainhas foram colocadas na placa.

Por exemplo, quando n = 4, algumas soluções parciais geradas pelo algoritmo de backtracking são as seguintes:

Backtracking



Backtracking

No nível inferior, as três primeiras configurações são ilegais, porque as rainhas se atacam. No entanto, a quarta configuração é válida e pode ser estendida para uma solução completa colocando mais duas rainhas na placa. Existe apenas uma maneira de colocar as duas rainhas restantes.

Exercícios

1. Implemente a solução para o problema das N-rainhas utilizando o backtracking.