DACC Departamento Acadêmico de Ciência da Computação

FUNDAÇÃO UNIVERSIDADE FEDERAL DE RONDÔNIA

Algoritmos Avançados

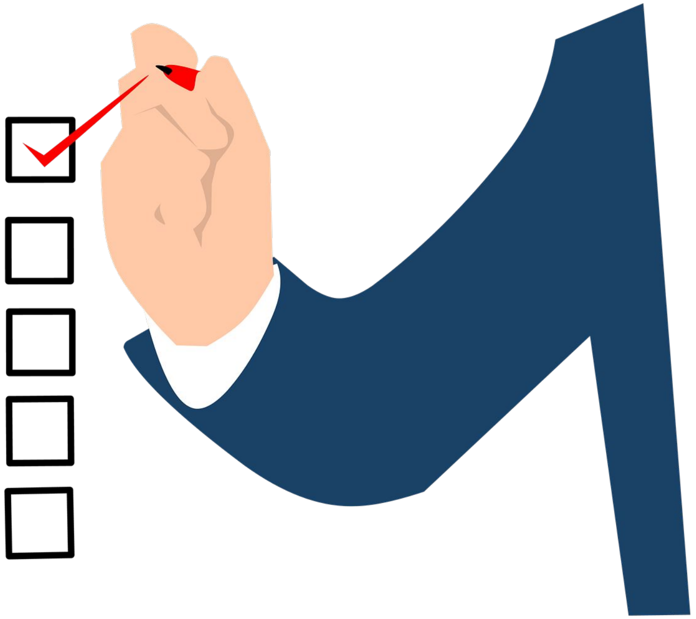
Professor:

Me. Lucas Marques da Cunha

lucas.marques@unir.br

Roteiro

1. Força Bruta e *Backtracking*
   a. O paradigma força bruta
   b. O problema dos doces
   c. Caixeiro Viajante
   d. Criar palavras
   e. *Backtracking*
   f. Busca Backtracking (formato)
   g. Problema das Rainhas
   h. Backtracking com restrições
   i. Coloração de Mapas
2. Desafios



Adaptado: prof. João do E.S. Batista Neto - USP

O paradigma Força Bruta

- Também conhecido como:
  - Busca completa;
  - *Backtracking* recursivo (sem podas);
- **Força Bruta** ou busca completa ou *backtracking* recursivo são sinônimos.
- Consistem de métodos que **<u>vasculham o espaço de busca inteiro</u>** (ou então parte dele) para obter a solução desejada.
  - Durante a busca, podemos podar parte do espaço se entendermos que tais partes não tem o que procuramos!

O paradigma Força Bruta

- A afirmação anterior é contestável.
  - busca completa = força bruta = backtracking recursivo.
- Busca por **força bruta**, computa cada possível solução e seleciona uma que preenche os requisitos.
- ***Backtracking*** é uma extensão da busca por força bruta em que as restrições do problema são avaliadas passo a passo e uma solução encontrada e outras descartadas.

O paradigma Força Bruta

- **Exemplo 01:**
- Sejam N caixas de doces, cada qual com alguma quantidade. Verifique se é possível redistribuir os doces de forma que todas as caixas tenham a mesma quantidade
- Se sim, qual a menor quantidade de doces que se deve redistribuir?

O paradigma Força Bruta

- Primeiro, verifique se o problema tem solução!
  - Σ qtd(i) / N = q (se a somatória for divisível por N)
- Depois calcule o valor mínimo de doces a deslocar:
  - Já pensou na solução?
  - Consegue ver que a solução ótima é força bruta ou a busca completa passando por todas as caixas?
- Escreva o algoritmo...

O paradigma Força Bruta

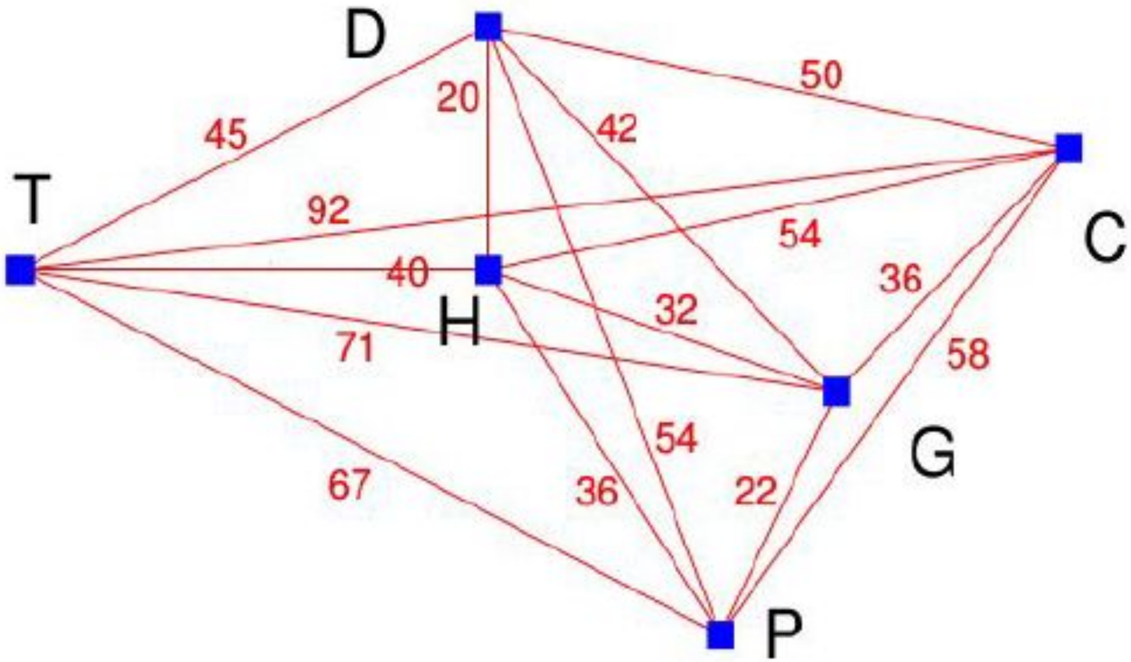
UNIR

- **Exemplo 02: TSP**
- TSP (*Traveling Salesman Problem*) ou o problema do caixeiro viajante.
    - Sendo n cidades e as distâncias entre todos os pares de cidades, calcule o custo mínimo de um **circuito** que começa em uma cidade ‘s’, passando por todas as demais cidades, finalmente retornando a ‘s’

O paradigma Força Bruta

- **Exemplo 02: TSP**
- O TSP é um problema de grafos e lida com um **grafo completo** e procura encontrar um **circuito Hamiltoniano** (aquele que usa todos os vértices uma única vez)
- Portanto, TSP consiste em encontrar um circuito Hamiltoniano de custo mínimo em um grafo completo.

O paradigma Força Bruta



O CAIXEIRO VIAJANTE

	H	P	G	C	D	T
Home (H)	0	36	32	54	20	40
Pet store (P)	36	0	22	58	54	67
Greenhouse (G)	32	22	0	36	42	71
Cleaners (C)	54	58	36	0	50	92
Drugstore (D)	20	54	42	50	0	45
Target (T)	40	67	71	92	45	0

O paradigma Força Bruta

- Forneça uma estratégia de **força bruta** para calcular o circuito mínimo.

O paradigma Força Bruta

- **Solução**
  - Faça uma lista de todos os circuitos possíveis;
    - Para N = 6, temos quantos circuitos possíveis?
      - 5! = 5x4x3x2x1 = 120
      - Calcule o peso de cada um deles;
      - Selecione o menor deles;

O paradigma Força Bruta

- O que acabamos de fazer foi um algoritmo de força bruta!
  - Ele é ótimo ?
    - Sim. Ele encontra a melhor solução;
  - É eficiente?
    - De jeito nenhum…. Se N <= 12, ele passa num programa de competição. Mais que isso, irá exceder o tempo de execução.
- Claro que é possível melhorar o desempenho.
  - Você não acha que cálculos se repetem neste problema?
  - Como podemos tirar proveito disso?

O paradigma Força Bruta

- **Exemplo 3: criar palavras**
- Gere, e imprima, todas as possíveis “palavras” de MAX letras utilizando uma string.
- Cada posição do vetor irá armazenar uma letra → palavra[a,b,c,d,’\0’]. (MAX=4)
- Considere todas as letras do alfabeto, de ‘a’ – ‘z’;
- Podemos sim fazer um algoritmo de **força bruta** para isso.
- **Qual a complexidade?**
  - Arranjos com repetições de 26 letras tomadas MAX a MAX -> 26^{MAX}

O paradigma Força Bruta

- **Exemplo 03: Criar Palavras**
- Vamos pensar na implementação
  - iterativa?
  - recursiva?

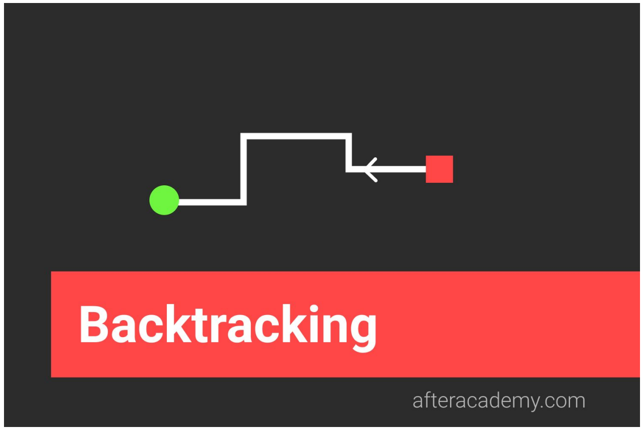
O paradigma Força Bruta

- **Exemplo 03: Criar Palavras**

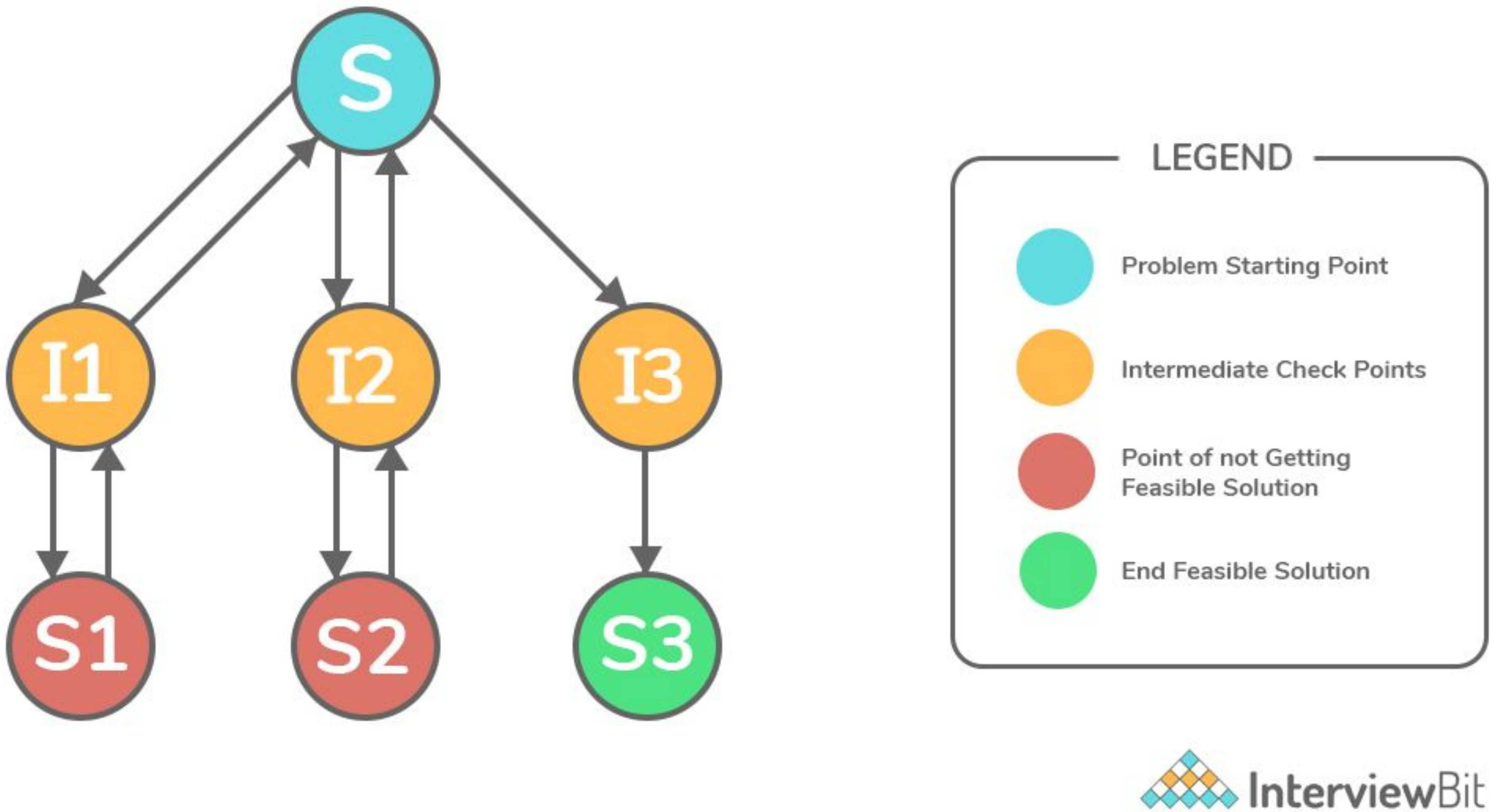
```c
#include<stdio.h>
#define MAX 5

void backtracking(char v[], int k)
{
    char letra;

    if (k == MAX)
        printf("%s\n", v);
    else
        for (letra = 'a'; letra <= 'z'; letra++)
        {
            v[k] = letra;
            backtracking(v, k + 1);
        }
}

int main()
{
    char v[MAX+1];

    v[MAX] = '\0';
    backtracking(v, 0);
    return 0;
}
```

Backtracking

- O algoritmo implementado anteriormente é um clássico **backtracking**.
- Uma **solução recursiva**, mas que também é **força bruta**, mesmo porque não tem como ser diferente neste caso.
- **O problema não traz restrições** e, portanto, nenhum grande desafio.



Backtracking



Busca Backtracking: 'formato'

- Eis um template clássico de *backtracking*!

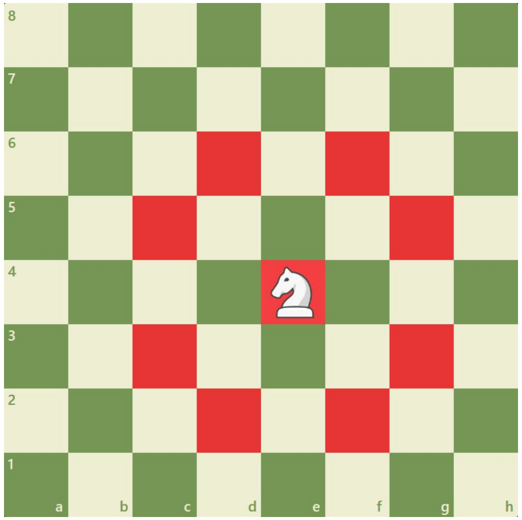
function BACKTRACKING-SEARCH(restricoes) % returns a solution or failure
    return RECURSIVE-BACKTRACKING(*atribuiçoes, restricoes*)

function RECURSIVE-BACKTRACKING(*atribuições, restriçoes*) % returns a solution or failure
    if *atribuições* is complete **then return** *atribuições*
        for each *value* **in** ORDER-DOMAIN-VALUES(*var, atribuições, csp*) **do**
        if *value* is consistent with *atribuições* according to *restriçoes* **then**
            add *{var=value}* to *atribuições*
            result ← RECURSIVE-BACKTRACKING(*atribuições, restriçoes*)
            remove *{var=value}* from *atribuições*
    return *failure*

- **Exemplo 4: Movimentação Cavalo no tabuleiro**
- Encontre um percurso realizado por um cavalo que visite todas as posições, sem passar pela mesma posição 2 vezes;
- Um percurso sempre existe, para um tabuleiro de tamanho 4.

Backtracking

- Com base no template apresentado anteriormente escreva o código:
  - If (*assignment is completed*)?
    - ter visitado todas as casas do tabuleiro!
  - for each value in problem domain.
    - São os 8 possíveis movimentos do cavalo!
  - if (*value is consistent with Constraints*)
    - movimento cai dentro do tabuleiro
    - casa ainda não foi visitada
  - Então
    - armazene resultado parcial;
    - Recursive call
    - remove resultado parcial



Backtracking

```c
#include<stdio.h>
#define SIZE 8

bool marked[SIZE][SIZE];

char moves[8][2] = {-1, -2,
                    -2, -1,
                    -2, 1,
                    -1, 2,
                     1, 2,
                     2, 1,
                     2, -1,
                     1, -2 };

bool valid(char v) {
    return (v >= 0) && (v < SIZE);
}
```

```c
void backtracking(char lin, char col, char k) {
    char new_lin, new_col, i;

    if (k == SIZE*SIZE-1) {
      printf("There exists a path!\n");
    } else
       for (i = 0; i < 8; i++) {
           new_col = col + moves[i][0];
           new_lin = lin + moves[i][1];
           if (valid(new_lin) && valid(new_col) && !marked[new_lin][new_col]) {
               marked[new_lin][new_col] = true;
               backtracking(new_lin, new_col, k+1);
               marked[new_lin][new_col] = false;
           }
       }
}
```

```c
int main() {
    int i, j;
    char c;

    for (i = 0; i < SIZE; i++)
        for (j = 0; j < SIZE; j++)
            marked[i][j] = false;

    marked[SIZE/2][SIZE/2] = true;
    backtracking(SIZE/2, SIZE/2, 0);
}
```

Nro soluções:
Size = 4
Size = 5
Size = 6
....

Backtracking

```
bool finished = FALSE;                       /* found all solutions yet? */

backtrack(int a[], int k, data input)
{
        int c[MAXCANDIDATES];                /* candidates for next position */
        int ncandidates;                     /* next position candidate count */
        int i;                               /* counter */

        if (finished) return;          /* terminate early
        if (is_a_solution(a,k,input))
                process_solution(a,k,input);
        else {
                k = k+1;
                construct_candidates(a,k,input,c,&n
                for (i=0; i<ncandidates; i++) {
                        a[k] = c[i];
                        backtrack(a,k,input);
                }
        }
}
```

Verifica se é solução

Incrementa contagem e/ou imprime a solução e/ou força interrupção do algoritmo ("finished" = TRUE), etc

Preenche o vetor ***c*** com no. "*ncandidates*" de valores possíveis para *a[k]*, dados os *k*-1 valores anteriores, e retorna esse no. Note que não é tão eficiente em termos de memória, pois ***c*** é gerado de uma só vez e armazenado na pilha de recursão.

Backtracking com restrições

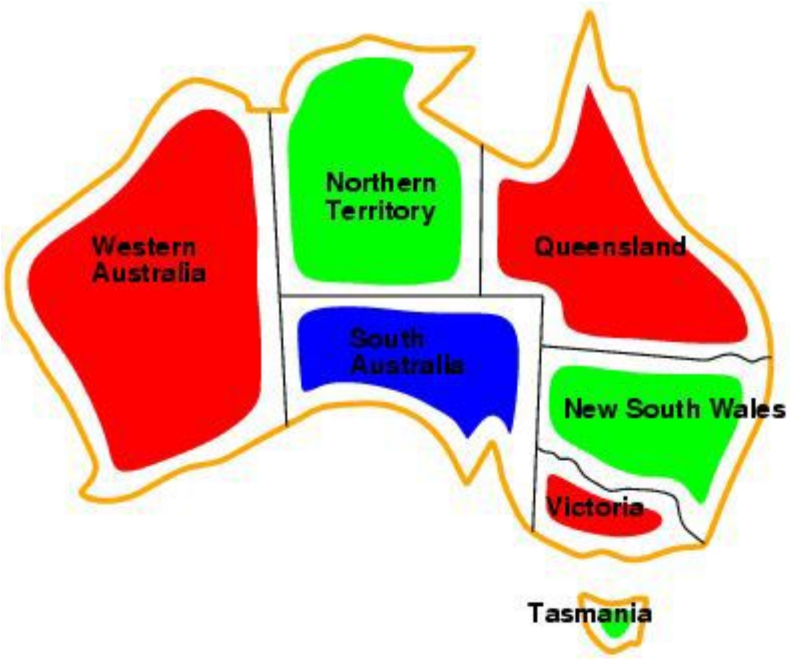
- Há uma série de problemas práticos que impõem restrições (*constraints*) ou condições inerentes ao problema.
  - **Problema das rainhas no tabuleiro, coloração em mapas, etc;**
- Esta classe de problemas é conhecida como CSP (*Constraint Satisfaction Problem*) ou **Problemas com satisfação de Restrições**, uma área bastante estudada na Inteligência Artificial (IA).

Coloração de Mapas

- O objetivo é colorir um mapa utilizando diferentes cores para regiões adjacentes.
- Podemos modelar esse exemplo como um problema de busca.

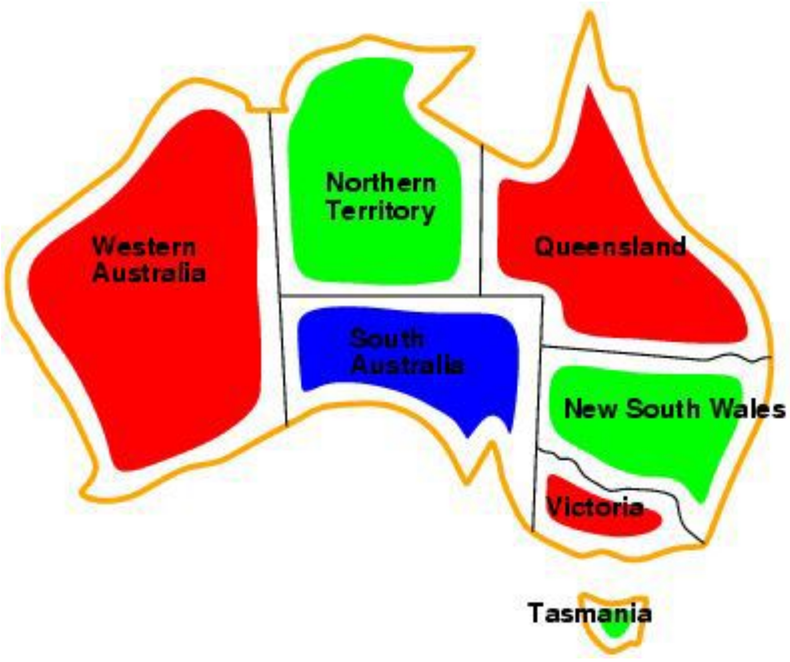
Coloração de Mapas

- Essa modelagem consiste de:
  - **Variáveis:** WA, NT, SA, Q, NSW, V, T
  - **Domínio:** Vermelho, verde e azul;
  - **Restrições:** regiões adjacentes devem possuir cores diferentes



Coloração de Mapas

- **Objetivo:**
  - Atribuir valores a todas as variáveis.
  - Cada atribuição deve respeitar as restrições impostas.



Problemas Combinatoriais

- **Problemas combinatoriais** podem ser modelados dessa maneira.
    - Envolvem encontrar uma atribuição, ordenação ou agrupamento a um conjunto finito de objetos discretos que satisfaz certas condições.
- Diversos problemas em computação são problemas combinatórios e podem ser modelados da mesma maneira.

Problemas Combinatoriais

- **Exemplo: Sudoku**

  - Modelagem:
    - Variáveis: A1...A9, B1...B9, ..., I1...I9
    - Domínio: 1...9
    - Restrições:

Alldiff(A1,A2,A3,A4,A5,A6,A7,A8,A9)

Alldiff (A1,B1,C1,D1,E1,F1,G1,H1,I1)

Alldiff (A1,A2,A3,B1,B2,B3,C1,C2,C3)

....

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Estado inicial

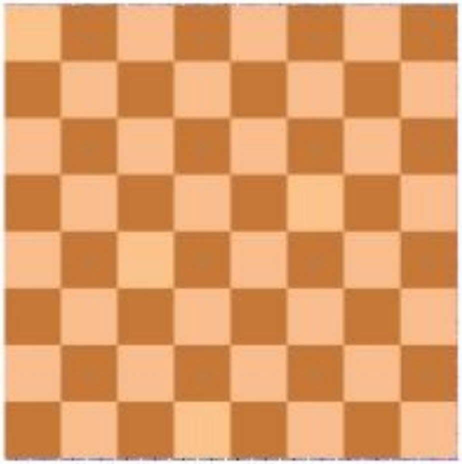
Estado final

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Problemas Combinatoriais

- **Exemplo: Oito Rainhas**

  - Modelagem:
    - Variáveis: A1, A2, ... A8, B1, ... B8, ..., H1, ... H8
    - Domínio: 0 e 1
    - Restrições: Não pode ter duas rainhas na mesma linha, coluna ou diagonal

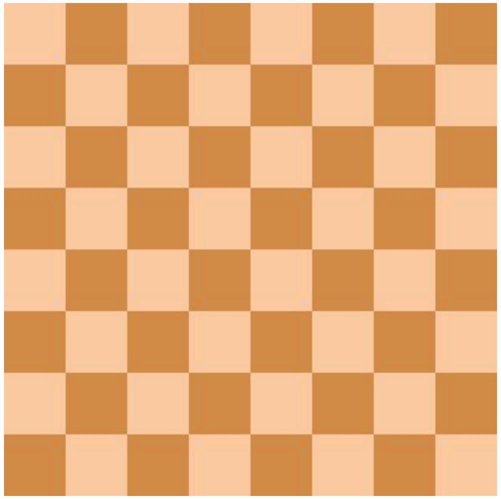


Estado inicial

Estado final

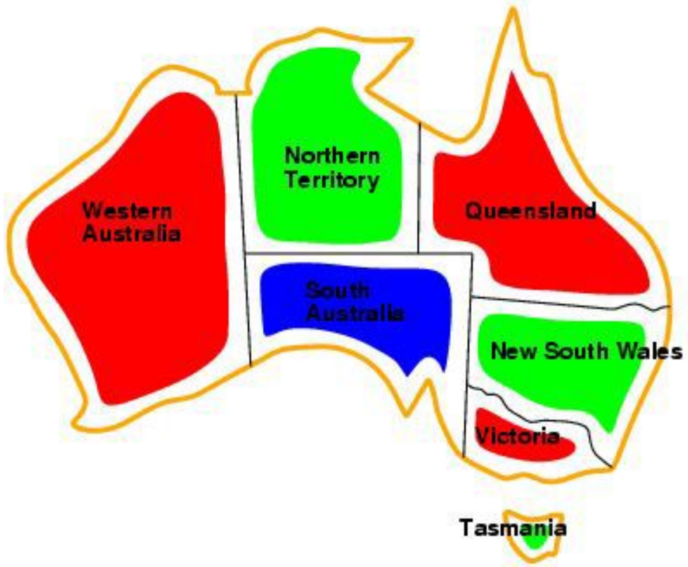
Problema de Satisfação de Restrições (PSR)

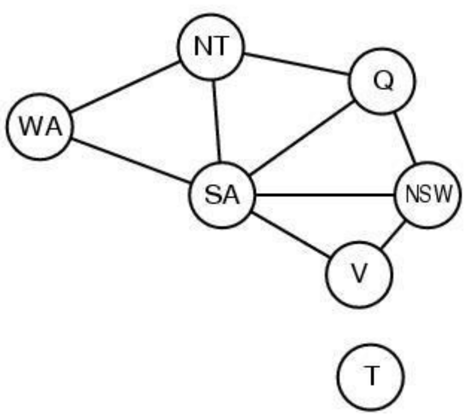
- Pode ser entendido como um problema de busca:
  - **Estado inicial:** nenhuma variável atribuída;
  - **Operador:** atribuir um valor a uma variável livre, dado que não existe conflito.
  - **Estado final:** atribuição completa e consistente
- ***Backtracking:***
  - Busca em profundidade
  - Resolve n-rainhas para n ≈ 25



Como Representar Restrições?

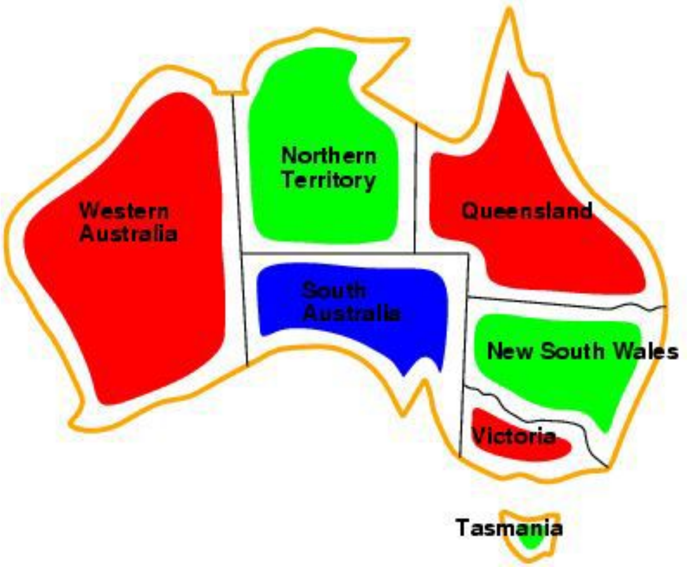
- Restrições podem ser:
- **Unárias:** envolvem uma variável.
  - Ex.: SA ≠ verde
- **Binárias:** envolvem duas variáveis.
  - Ex.: SA ≠ WA
- **Maior-ordem:** envolvem três ou mais variáveis. Ex.: Alldiff(A1...A9)
- Restrições de maior-ordem podem ser quebradas em binárias;
- Grafos é uma representação frequente para as restrições.

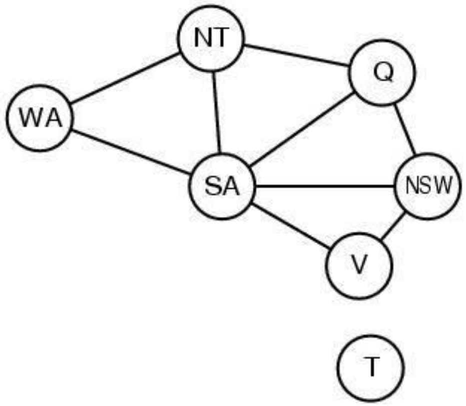




Grafo de Restrições

- **Para PSRs binários:**
  - Nós são variáveis
  - Arestas são restrições
- **Benefícios:**
  - Padrão de representação
  - Funções genéricas de objetivo e sucessor
  - Pode ser utilizado para simplificar a busca
    - Componentes independentes como Tasmânia



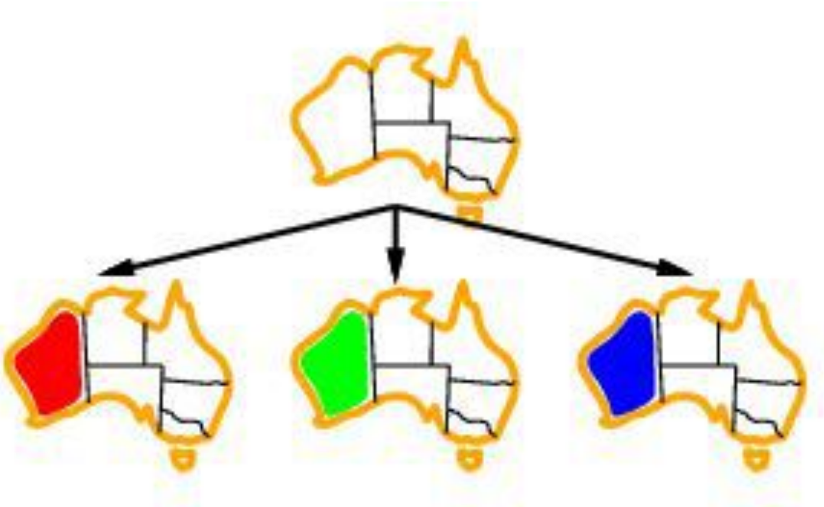


Busca Backtracking

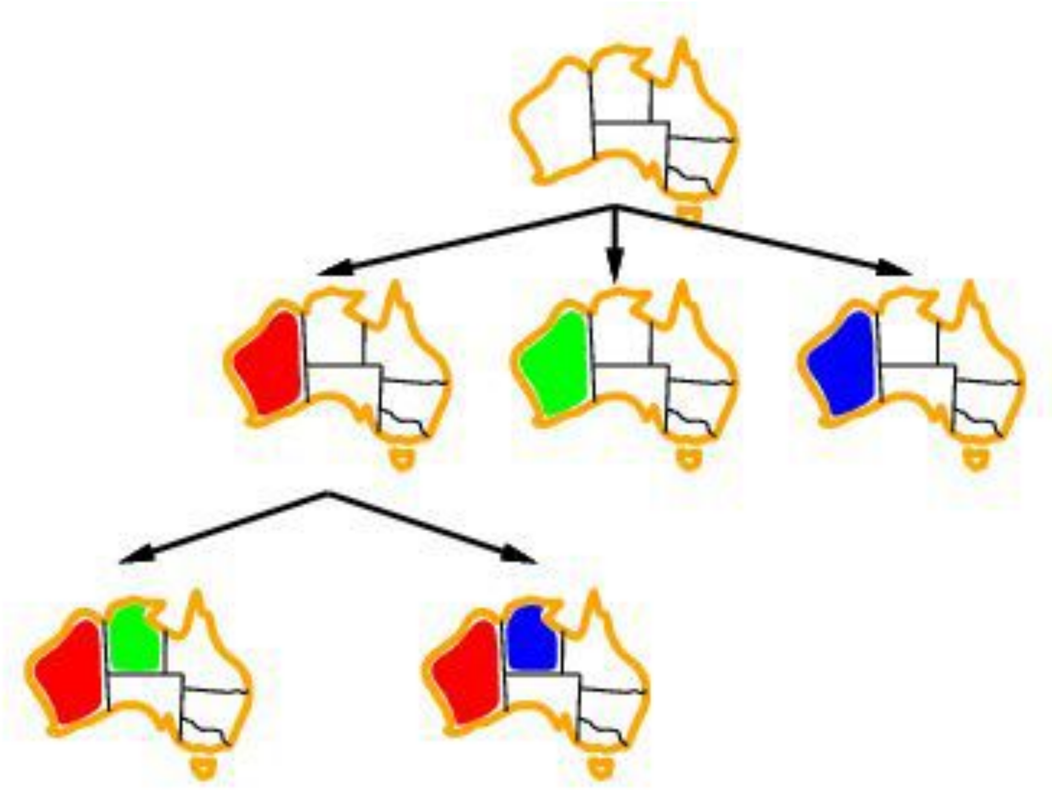
```
function BACKTRACKING-SEARCH(csp) % returns a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) % returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to CONSTRAINTS[csp] then
            add {var=value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure  then return result
            remove {var=value} from assignment
    return failure
```

Busca Backtracking

Busca Backtracking

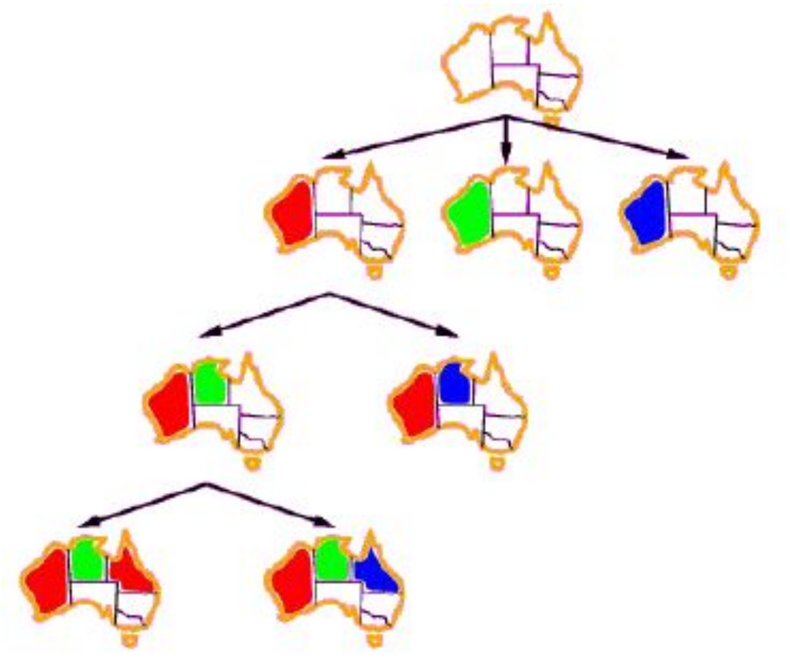


Busca Backtracking



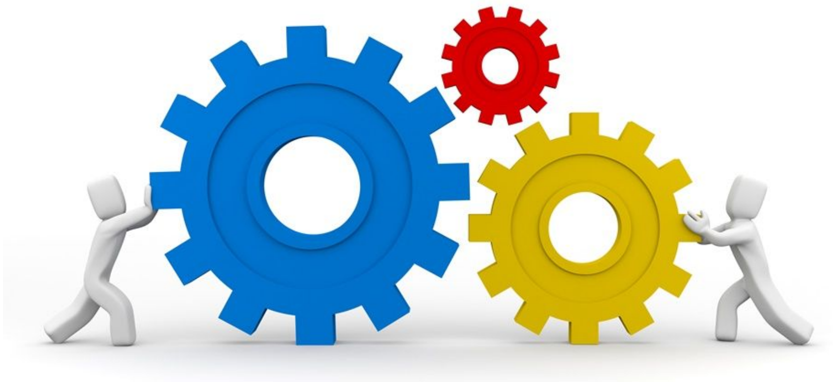
Eficiência do Backtracking

- Complexidade de execução do **backtracking**:
  - n: número de variáveis
  - d: tamanho do domínio
  - Complexidade: $O(d^n)$



Como Melhorar a Eficiência do Backtracking?

- Métodos de propósito geral podem fornecer grandes ganhos de desempenho:
  - Qual variável deve ser a próxima a ser atribuída?
  - Em qual ordem os valores devem ser tentados?
  - Pode-se detectar falhas inevitáveis mais cedo?

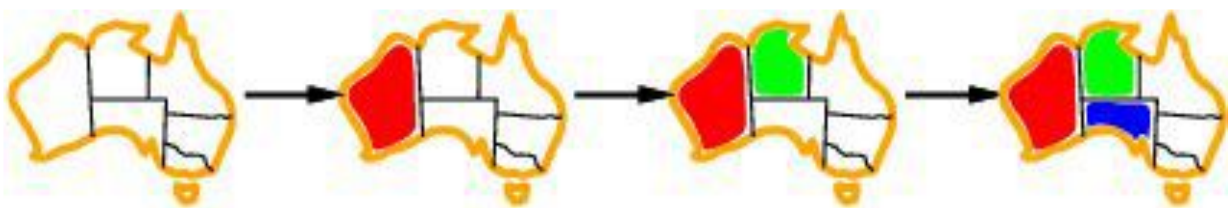


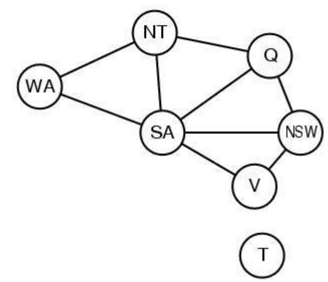
Busca Backtracking

```
function BACKTRACKING-SEARCH(csp) % returns a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) % returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to CONSTRAINTS[csp] then
            add {var=value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure  then return result
            remove {var=value} from assignment
    return failure
```

Valor Restante Mínimo
(*Minimum Remaining Value* (MRV))

- Também conhecido como:
    - Variável mais restrita:
- Escolha a variável com menor número de movimentos legais;

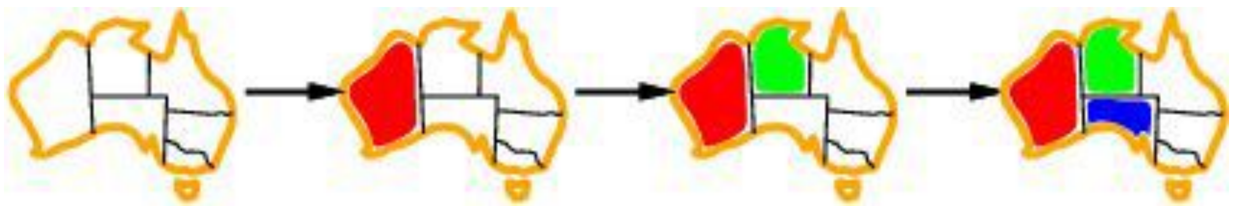


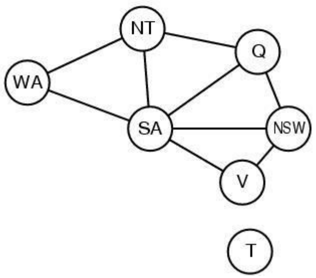


- Escolha a variável com a maior probabilidade de causar uma falha o mais rapidamente possível.
    - Isso provoca podas antecipadas.

Heurística de Grau (vértice)

- Suponha o grafo inicial, não colorido:
- Qual a validade da heurística MVR?
  - **Nenhuma!!! Todas as variáveis podem ser coloridas com 3 cores…**





- **Heurística grau:** selecione a variável que tenha o maior número de restrições a outras variáveis não atribuídas: Vértice de maior grau ! **(diminui o fator de ramificação)**
- No geral, MVR é melhor do que grau, então utiliza-se grau como desempate para MVR.

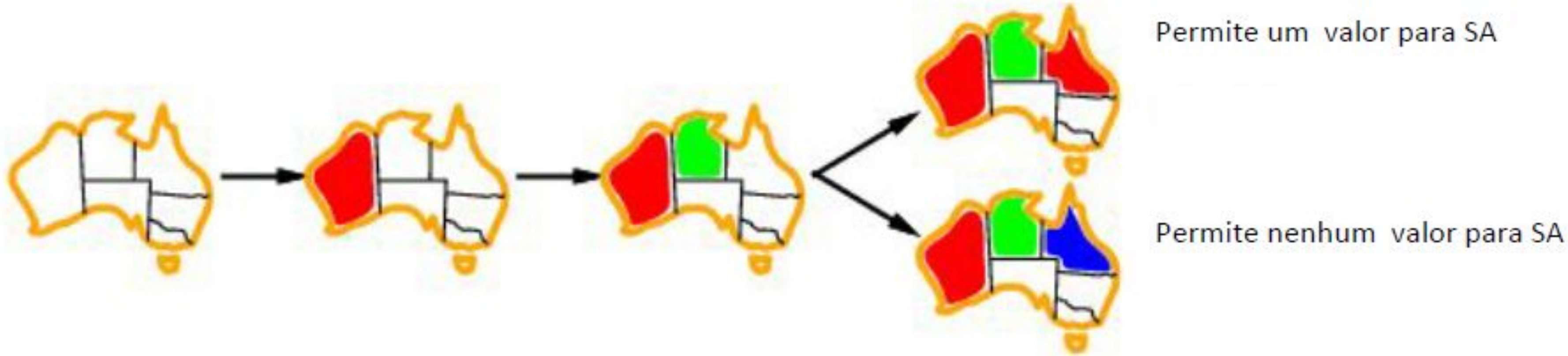
Ordem dos Valores a serem atribuídos

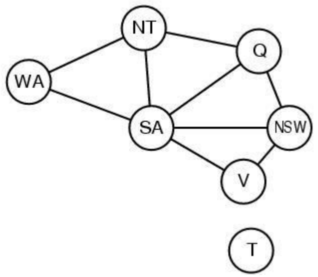
function BACKTRACKING-SEARCH(*csp*) % returns a solution or failure
    return RECURSIVE-BACKTRACKING(*{}* , *csp*)

function RECURSIVE-BACKTRACKING(*assignment, csp*) % returns a solution or failure
    if *assignment* is complete **then return** *assignment*
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
    for each *value* **in** **ORDER-DOMAIN-VALUES**(*var, assignment, csp*) **do**
        if *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] **then**
            add *{var=value}* to assignment
            result ← RECURSIVE-BACKTRACKING(*assignment, csp*)
            if *result* ≠ *failure* **then return** *result*
            remove *{var=value}* from *assignment*
    return *failure*

Valor Menos Restritivo (*least restrictive value*)

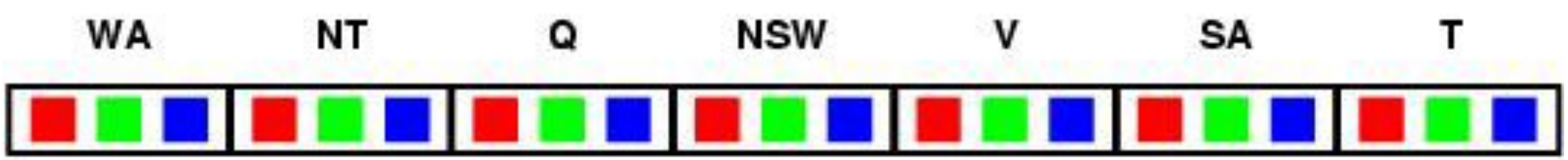
- Dada uma variável, escolha o **valor menos restritivo**:
  - Aquele que remove o menor número de valores para outras variáveis.





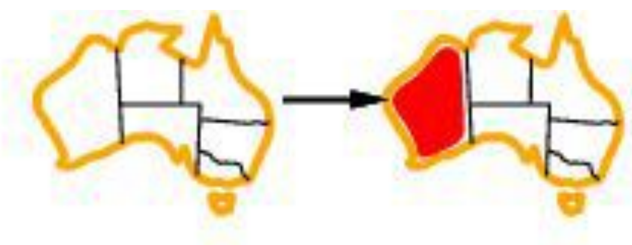
Verificação Adiante (*forward checking*)

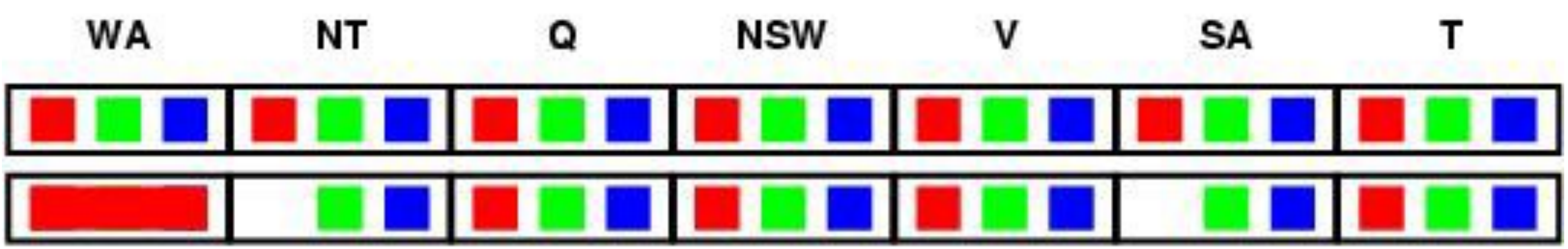
- Manter os valores legais remanescentes para variáveis não atribuídas;
- Retroceder a busca quando uma variável não possuir valores legais.



Verificação Adiante (*forward checking*)

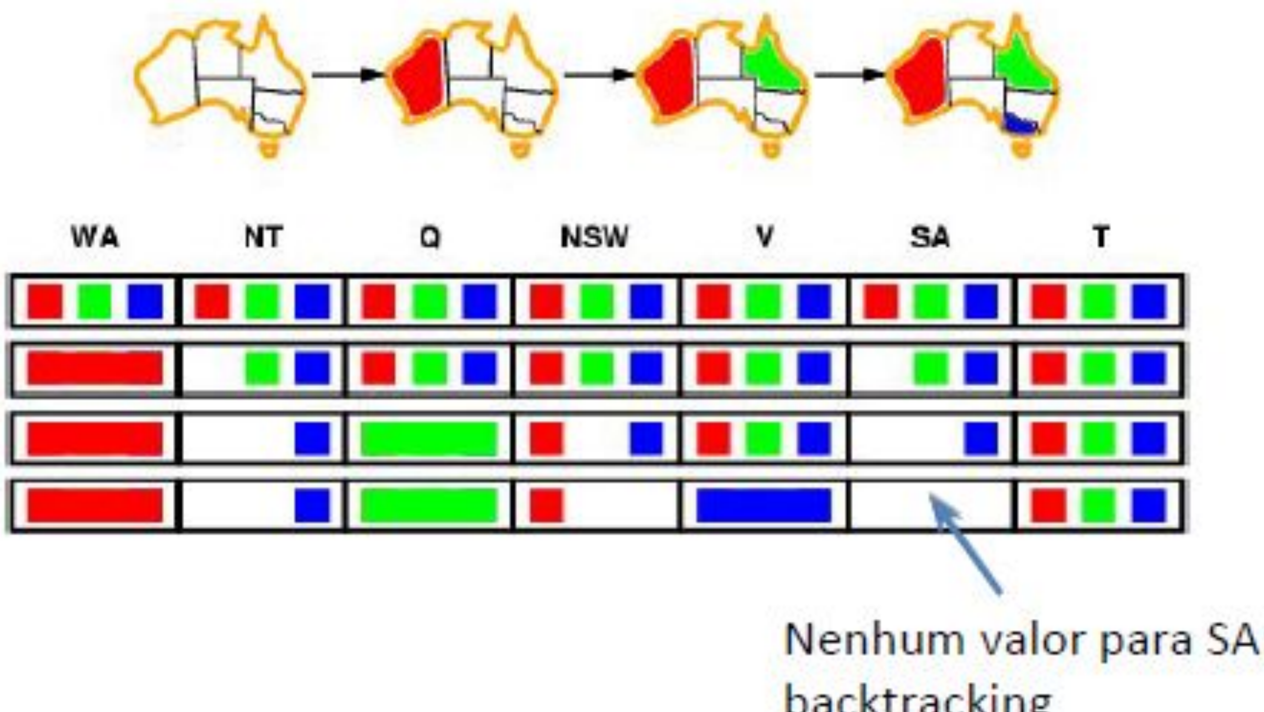
- Manter os valores legais remanescentes para variáveis não atribuídas;
- Retroceder a busca quando uma variável não possuir valores legais.



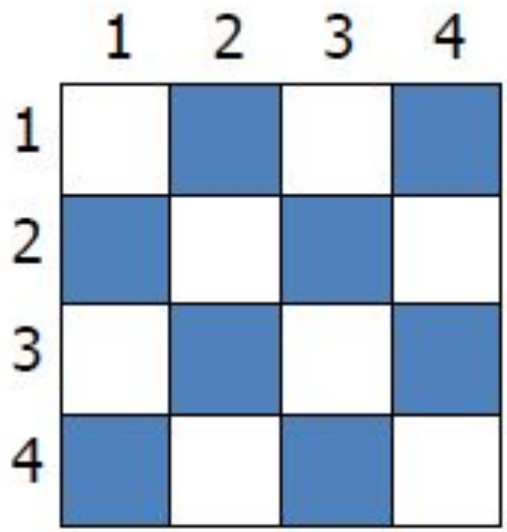


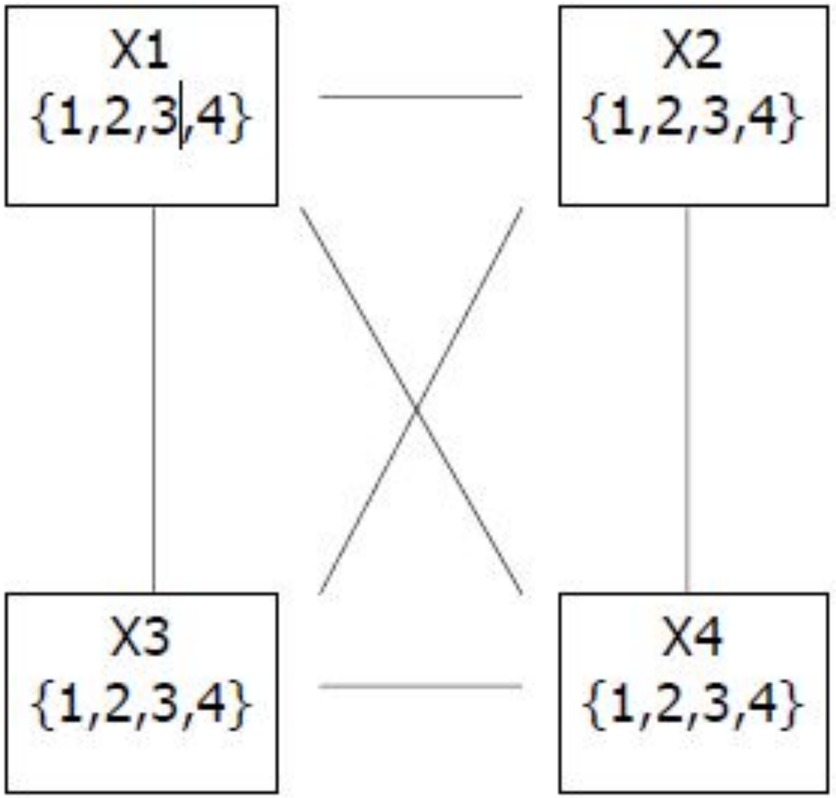
Verificação Adiante (*forward checking*)

- Manter os valores legais remanescentes para variáveis não atribuídas;
- Retroceder a busca quando uma variável não possuir valores legais.

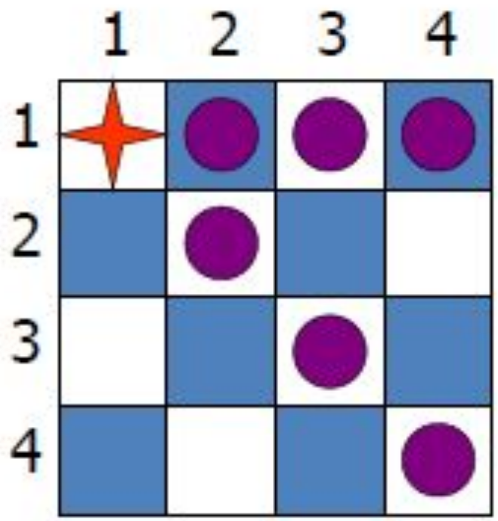


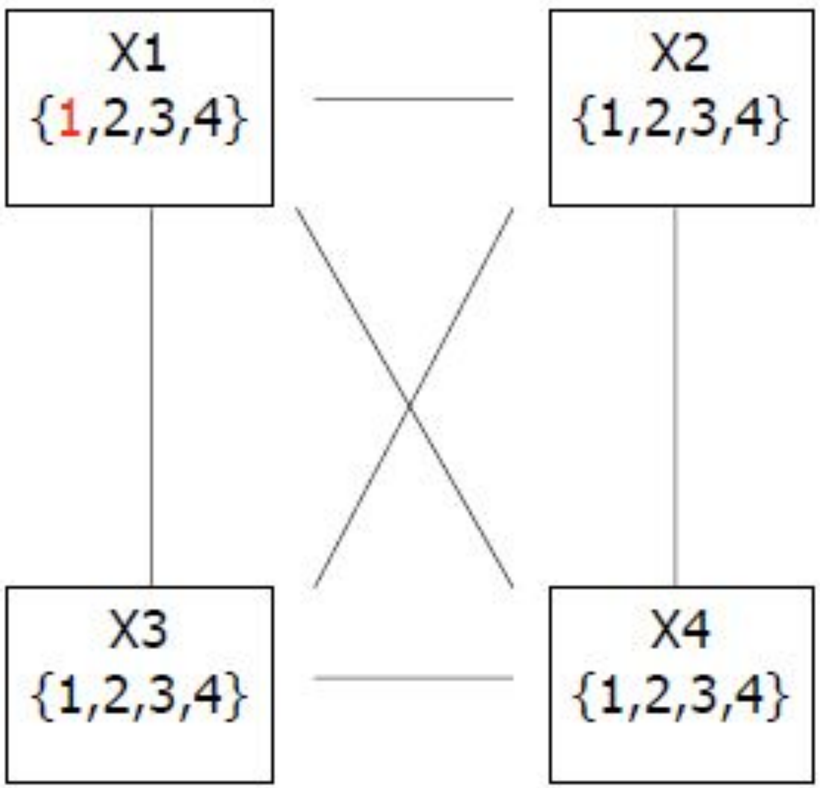
Verificação Adiante: 4-Rainhas



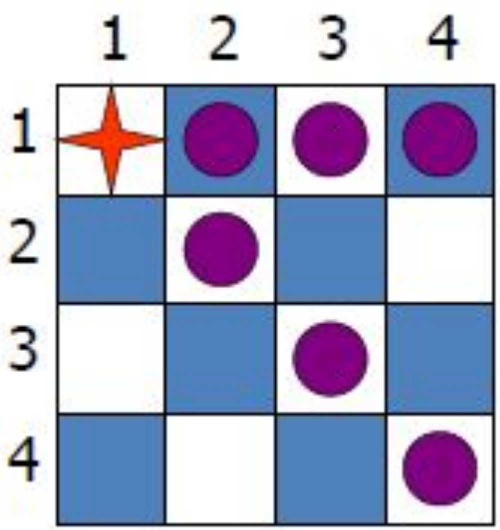


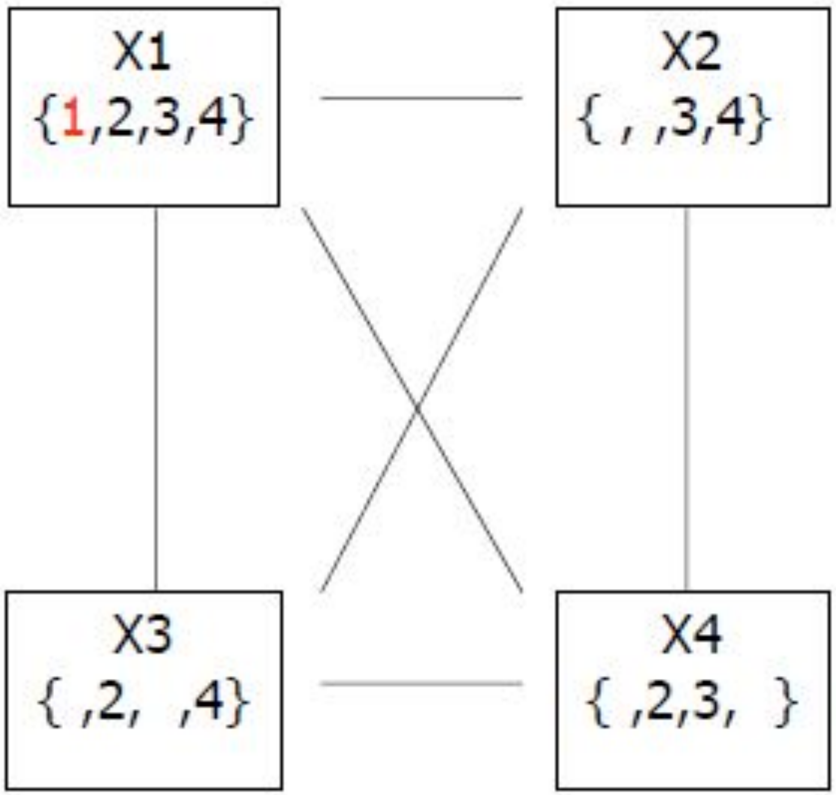
Verificação Adiante: 4-Rainhas



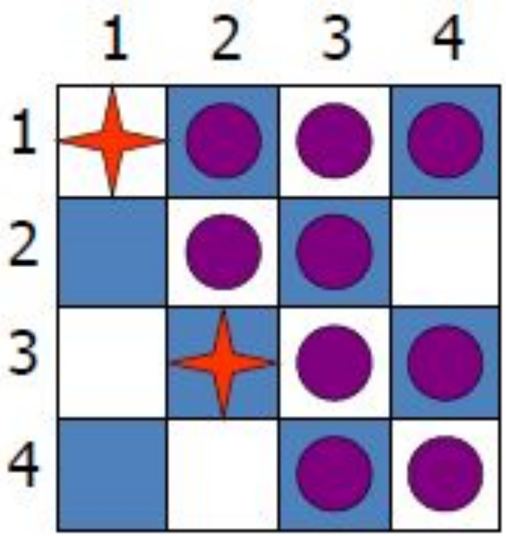


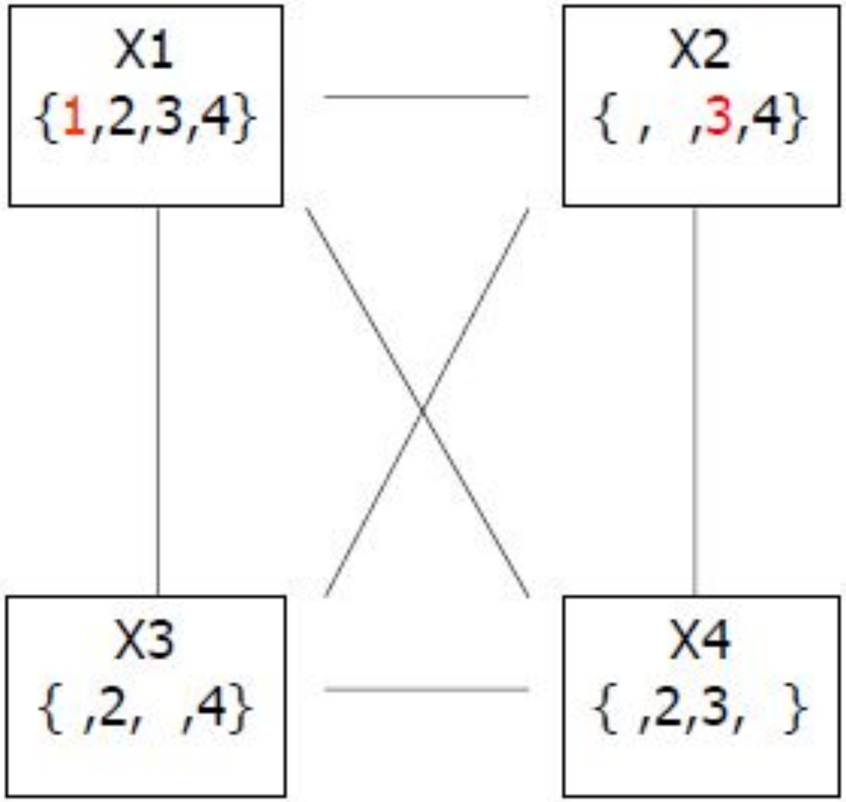
Verificação Adiante: 4-Rainhas



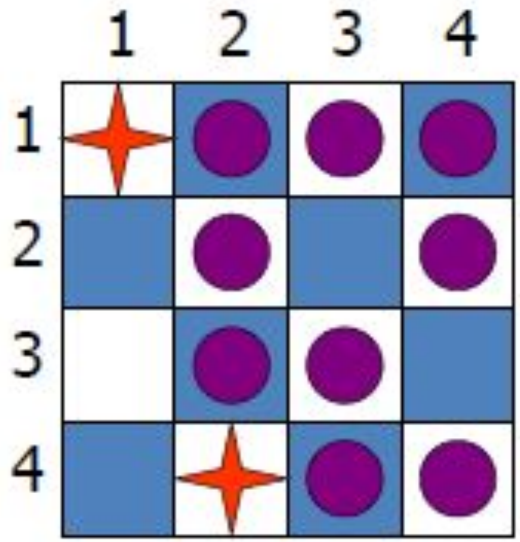


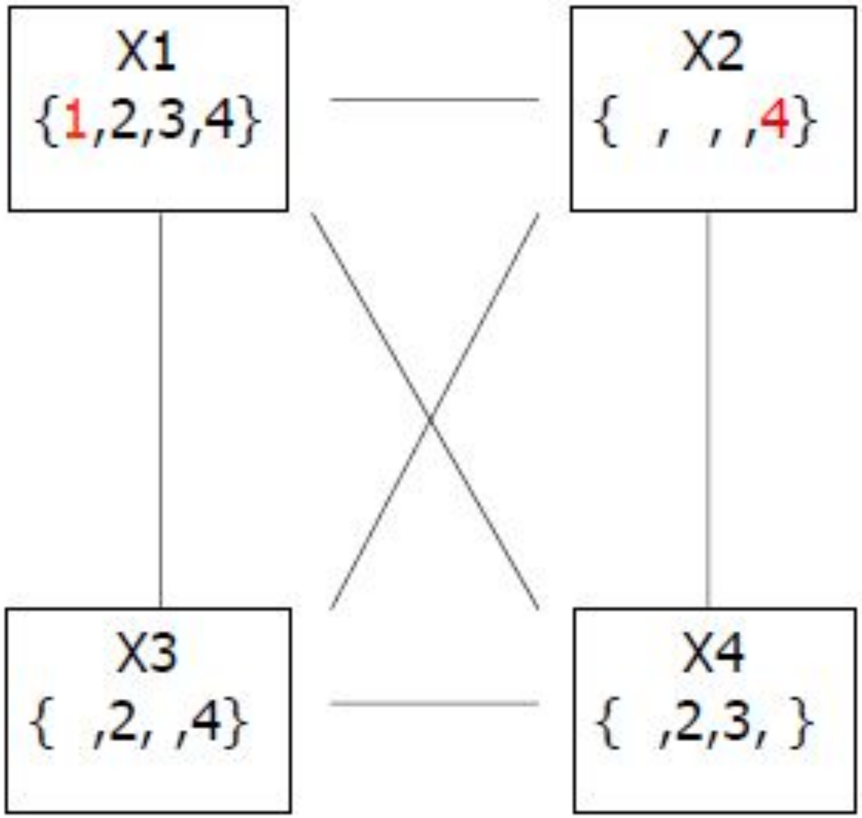
Verificação Adiante: 4-Rainhas



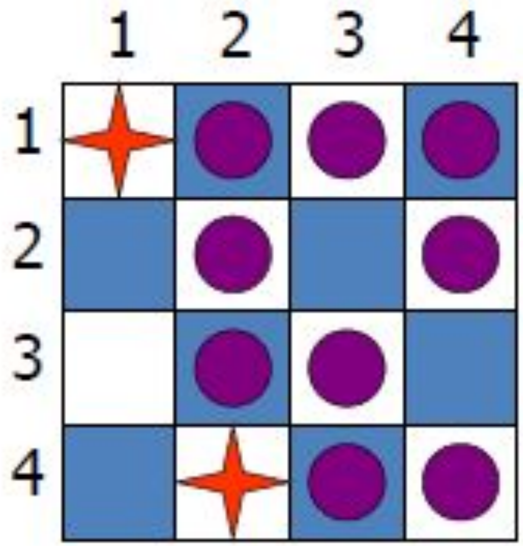


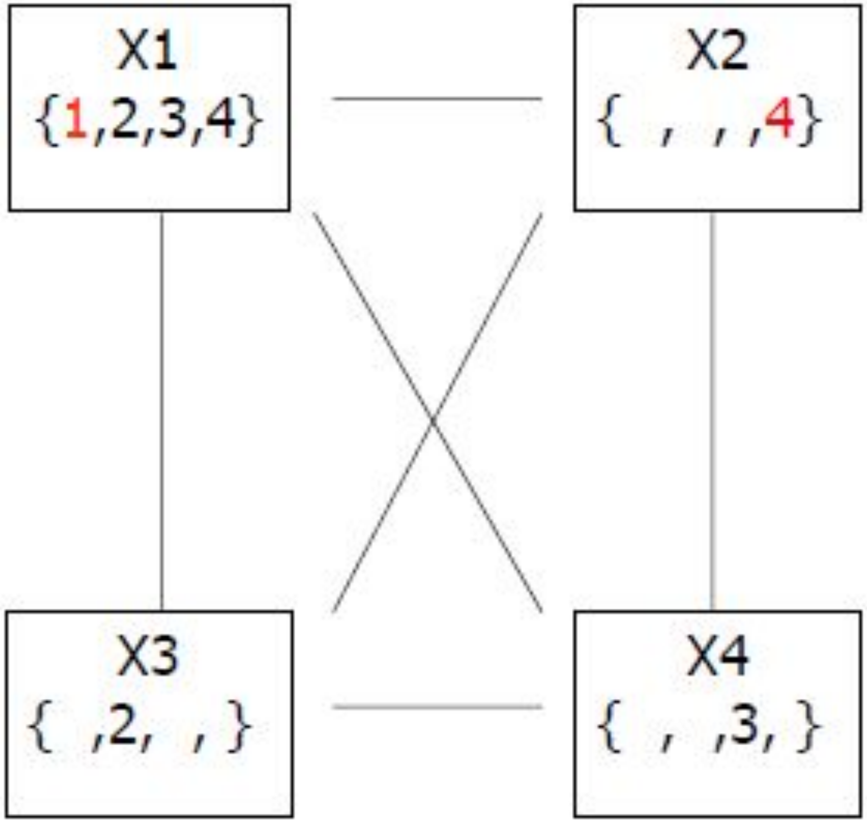
Verificação Adiante: 4-Rainhas



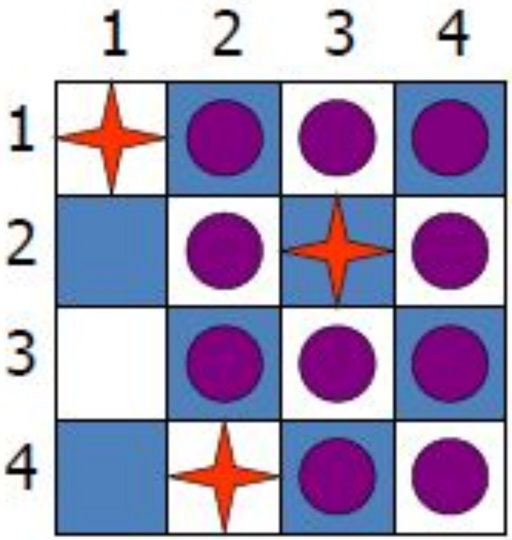


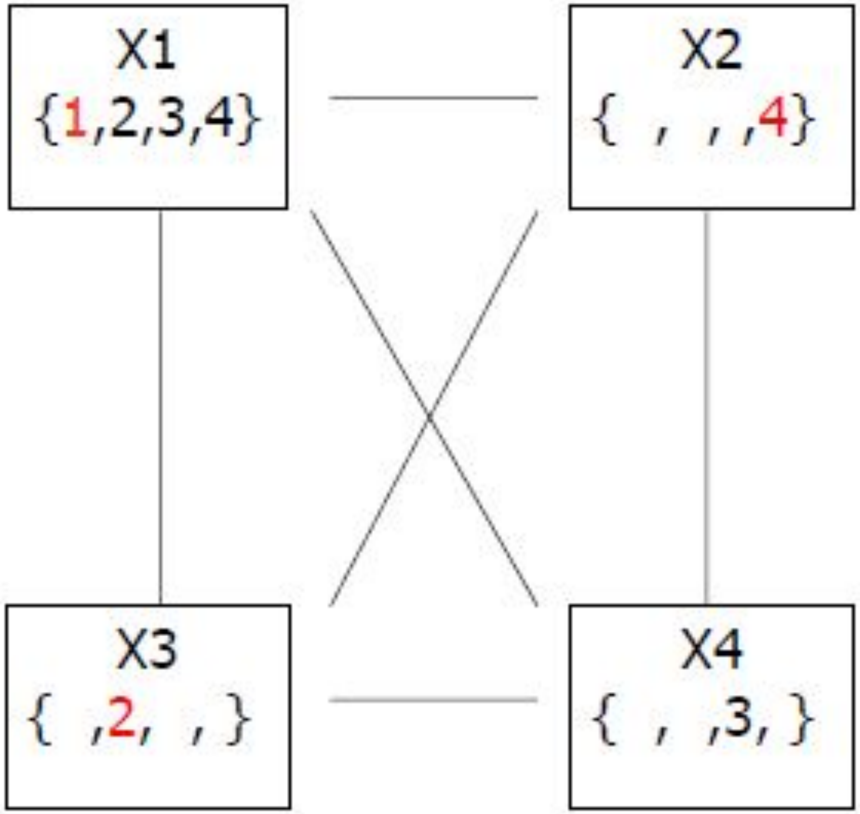
Verificação Adiante: 4-Rainhas



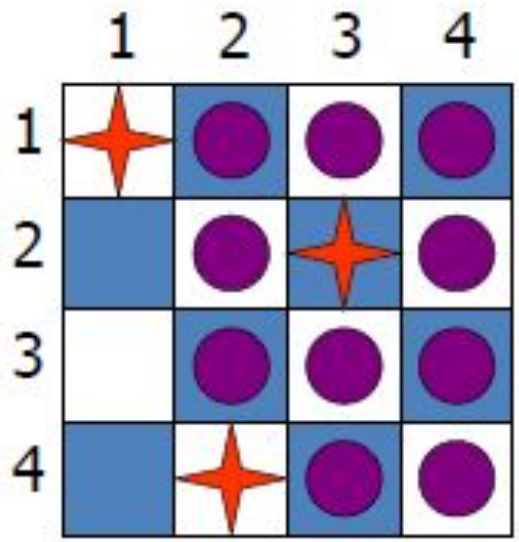


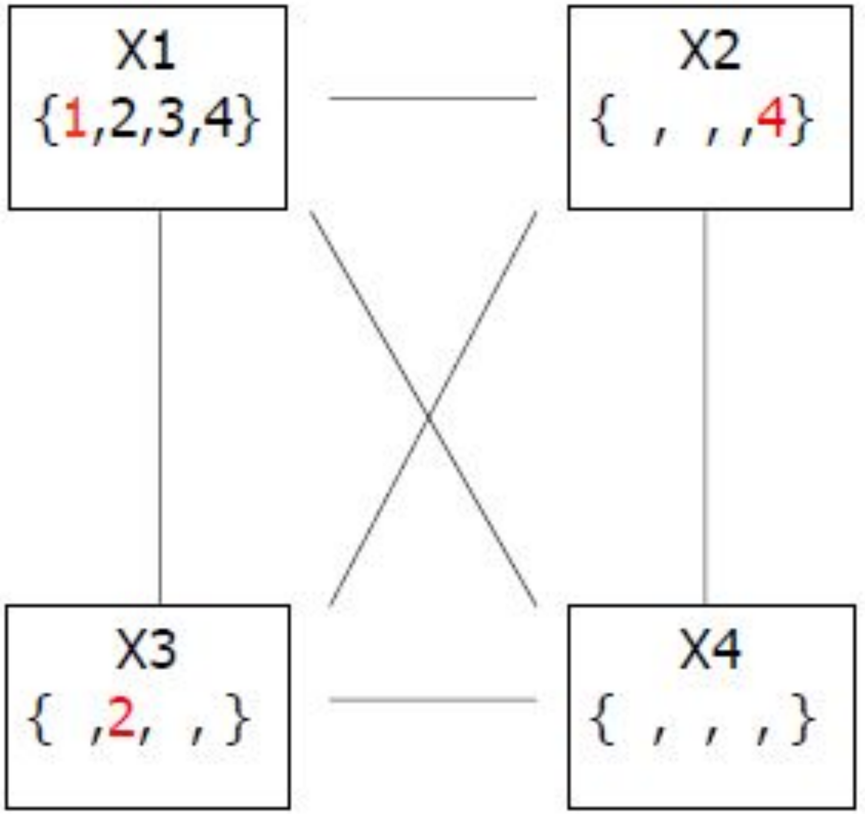
Verificação Adiante: 4-Rainhas



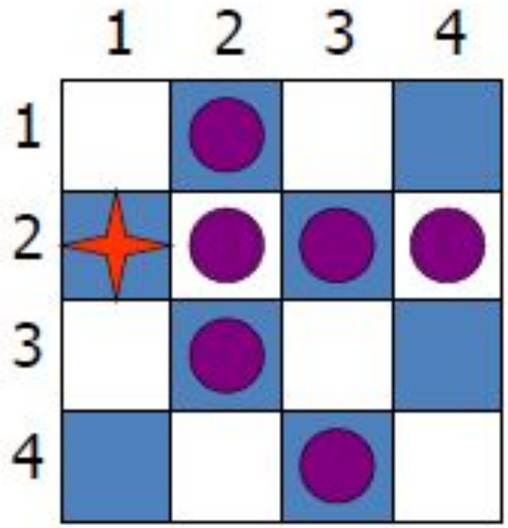


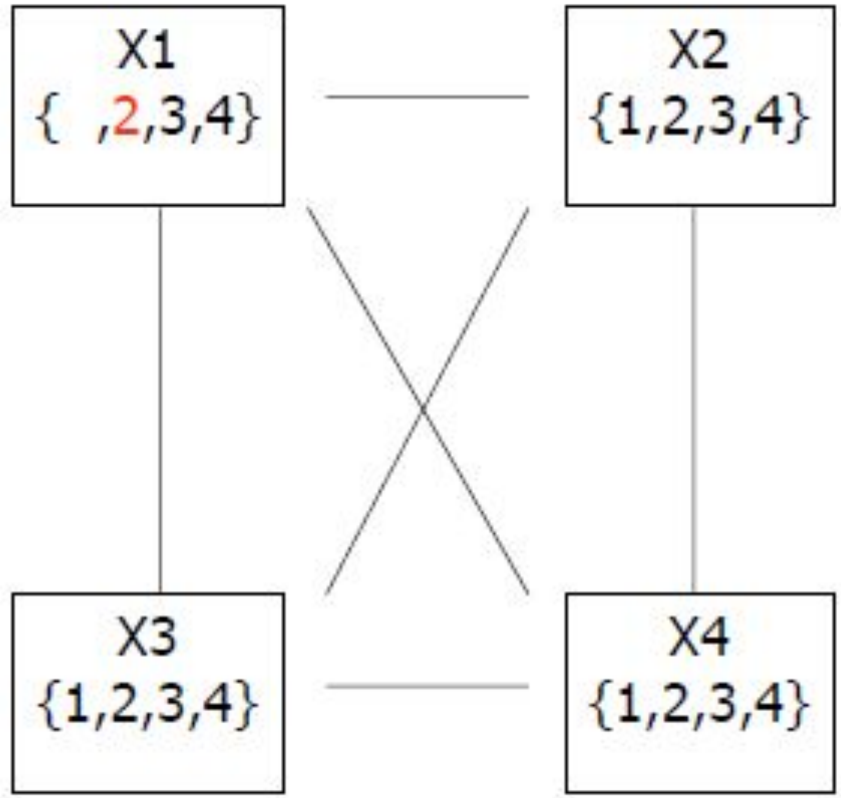
Verificação Adiante: 4-Rainhas



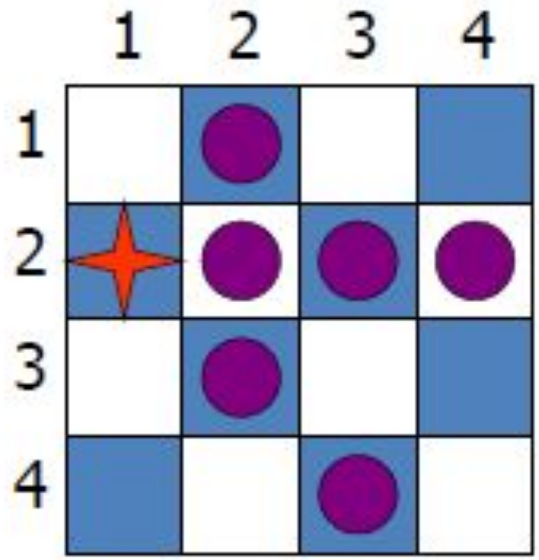


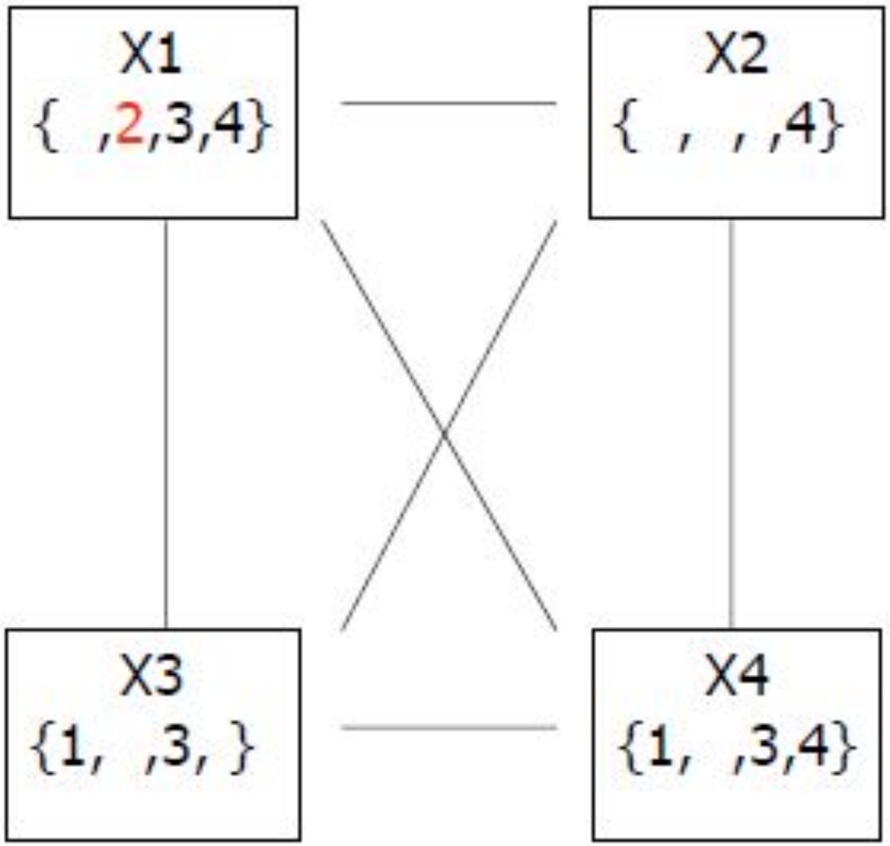
Verificação Adiante: 4-Rainhas



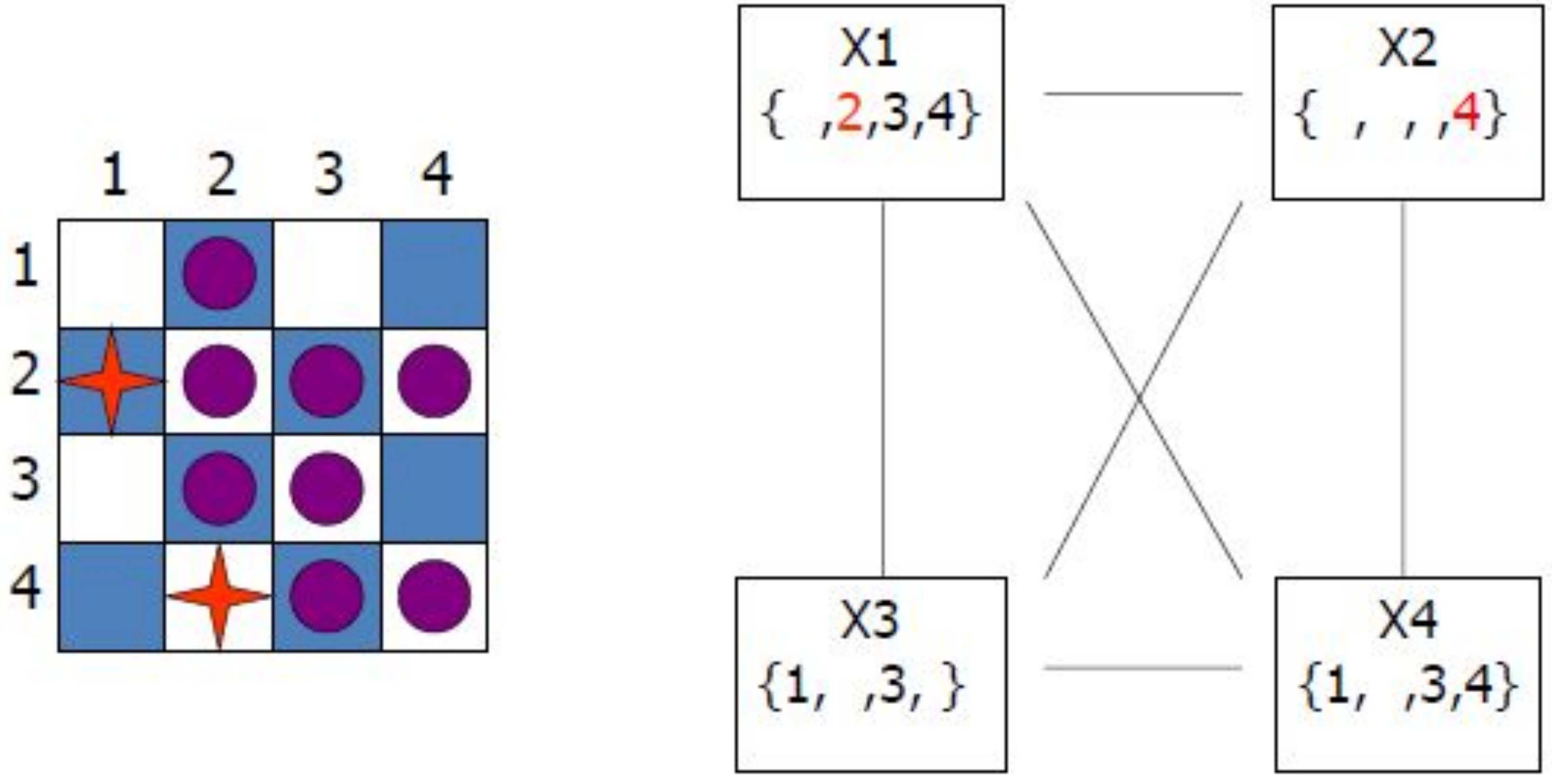


Verificação Adiante: 4-Rainhas

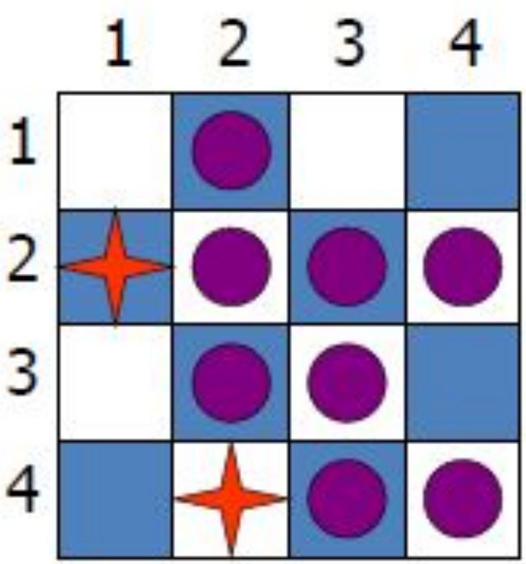


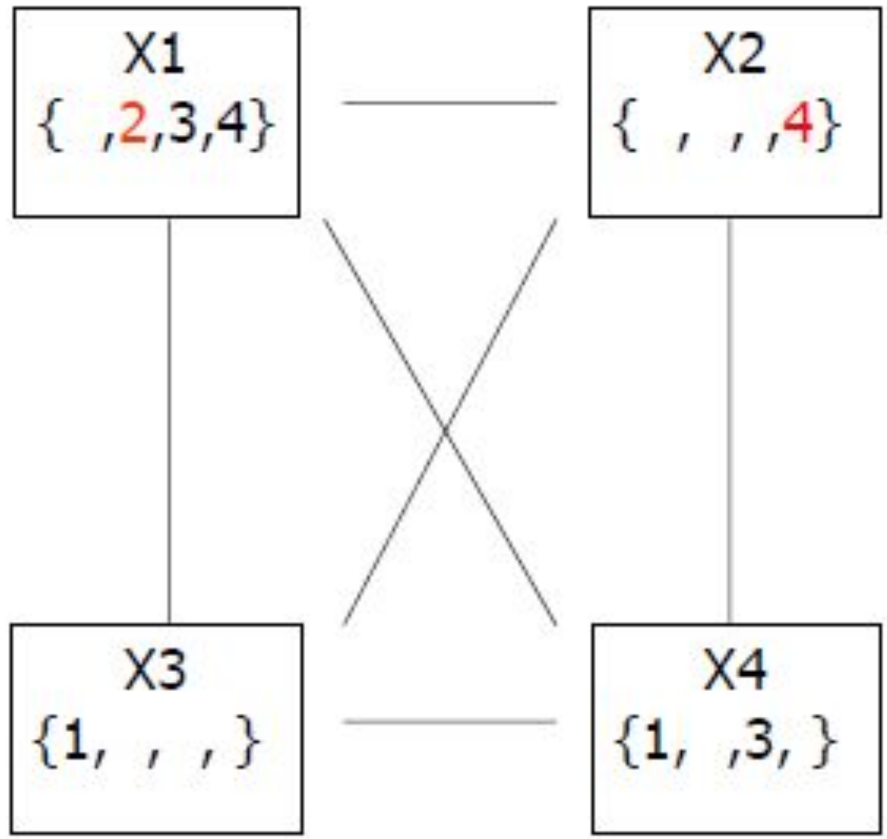


Verificação Adiante: 4-Rainhas

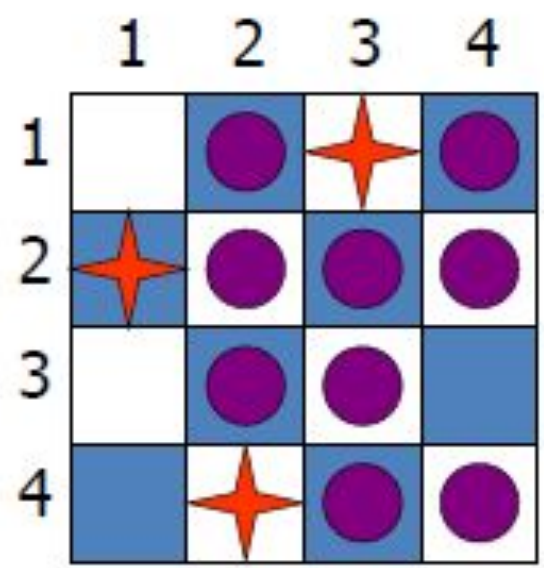


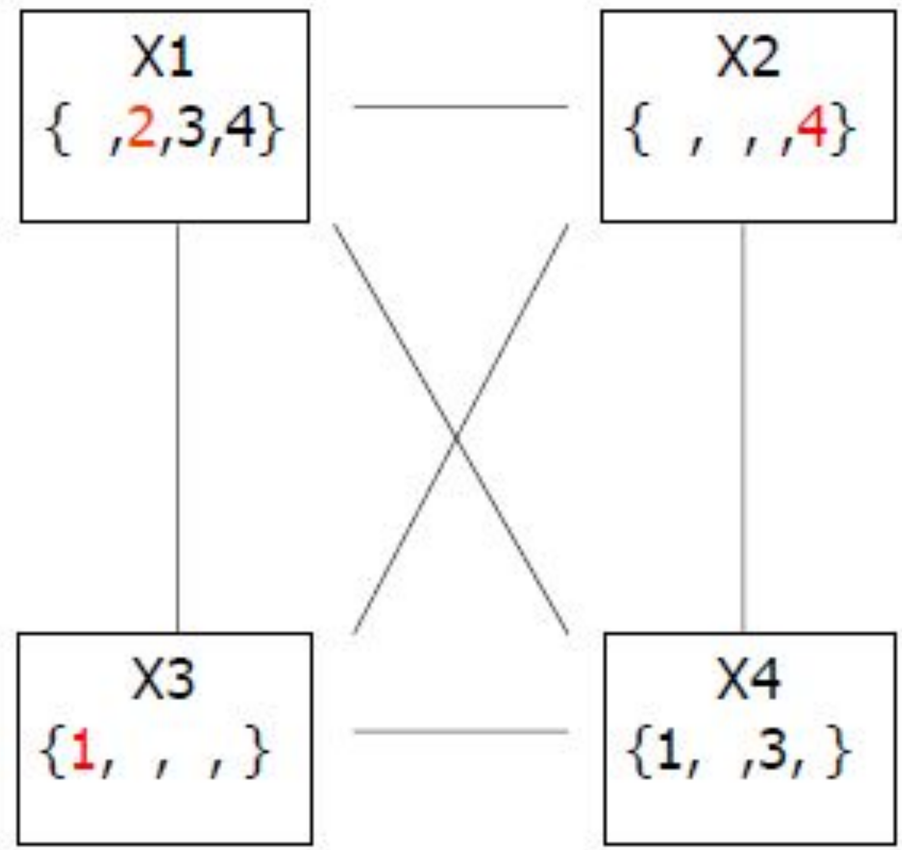
Verificação Adiante: 4-Rainhas



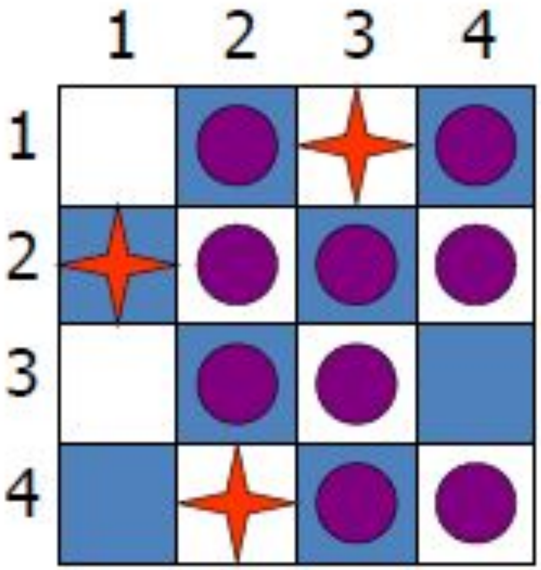


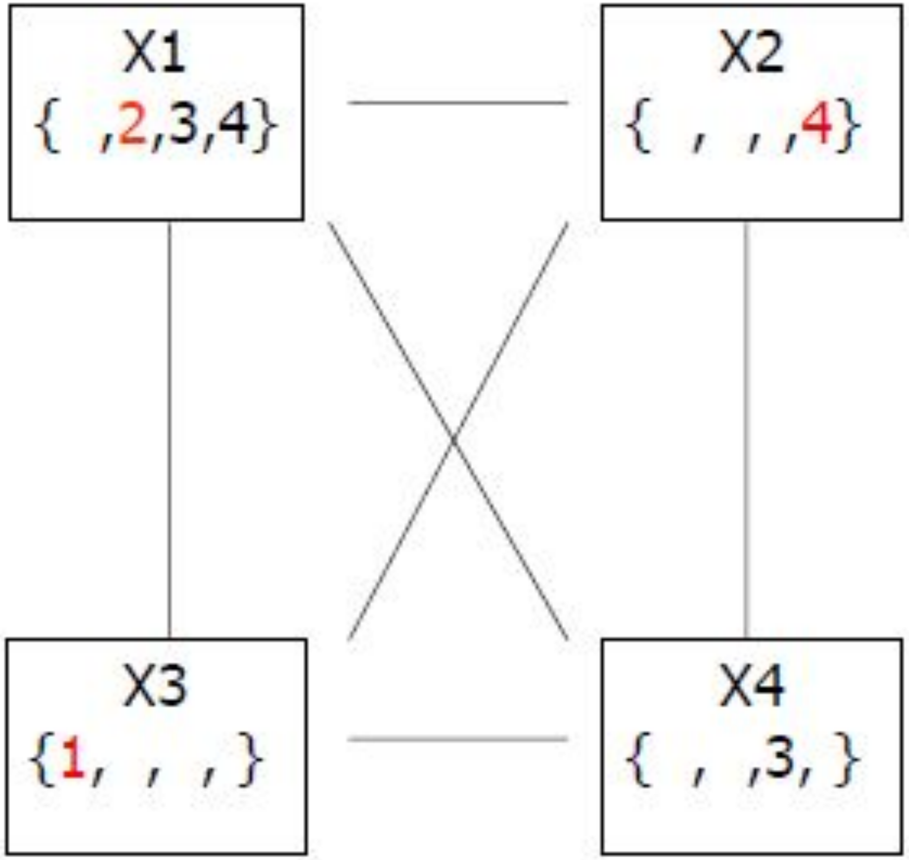
Verificação Adiante: 4-Rainhas



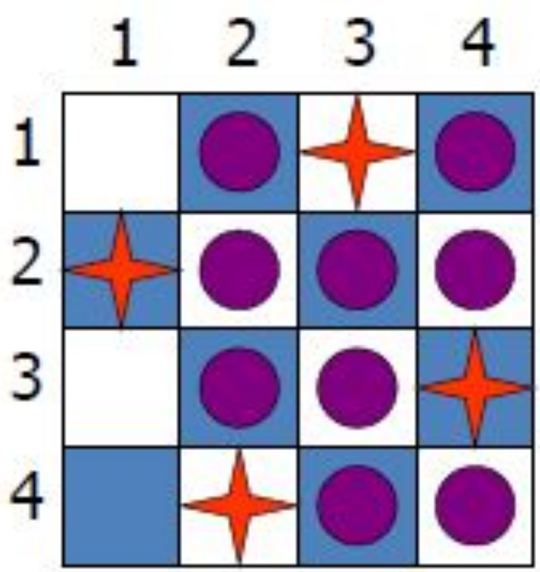


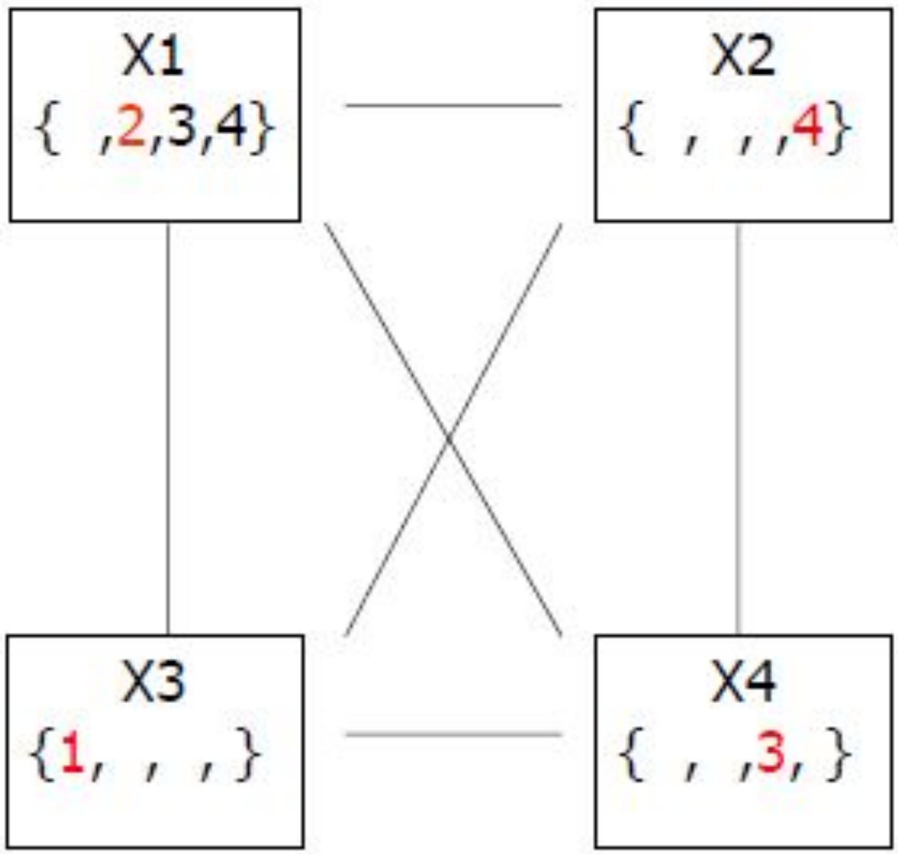
Verificação Adiante: 4-Rainhas





Verificação Adiante: 4-Rainhas





Dúvidas, sugestões?