# Projeto e Análise de Algoritmos

Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

4 de maio de 2016

# Backtracking

# Conceitos Básicos

- O Backtracking é uma técnica de busca de solução, onde dado um conjunto é necessário fazer várias decisões sobre um conjunto de opções oferecidas.
- O Backtracking é usado quando não se tem informações suficientes para saber o que deve escolher (não é possível definir uma escolha gulosa).
- Cada escolha leva a mais opções.
- Alguma seqüência de decisões, provavelmente mais do que uma, levam à solução do problema.
- Muitas seqüências levam à impossibilidade de continuar.
- A técnica do Backtracking é tentar uma seqüência, se atingir um resultado sem saída, retorna a(s) última(s) escolha(s) e tenta novas opções. Até encontrar uma que seja a solução.

## Ilustrando a técnica

### O Problema do Labirinto

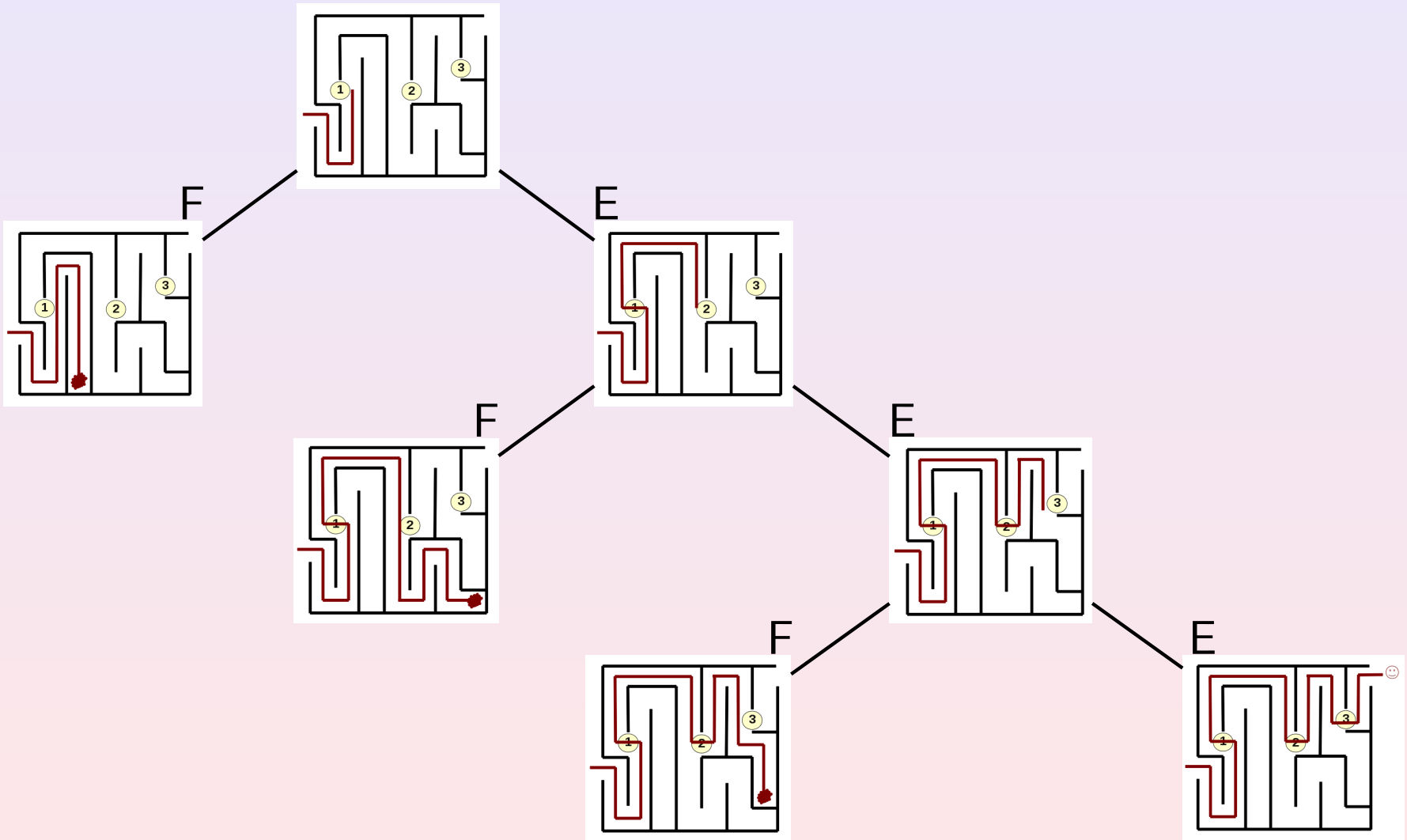
Dado um labirinto, encontrar um caminho da entrada à saída.

- O caminho solução é indicado por um conjunto de decisões tomadas cada vez que há mais de uma opção de caminho a seguir.
- Normalmente, para tomar uma decisão existem duas ou três possíveis escolhas: *Ir para esquerda*; *Ir para direita*; e *Seguir em Frente*.
- Ao atingir um ponto sem saída, é necessário retornar e tentar um caminho não escolhido.
- Eventualmente encontramos uma solução.
- Se voltamos ao início significa que não há solução.

## A árvore de decisão no espaço do problema

- Cada ponto de escolha nos leva a 2 ou mais decisãoes, cada decisão pode levar a outros pontos de escolha, ou a um ponto sem saída, ou à solução.
- Cada ponto de escolha pode ser mapeado como um vértice interno de uma árvore, cada filho desta árvore é uma possível decisão que tomamos.
- As folhas da árvore são os pontos sem saída, que não servem como solução, ou possíveis soluções, não há restrição a uma única solução nem mesmo que haja solução.
- Um problema não é resolvido por uma estrutura de dados árvore. A busca pela solução é vista como uma varredura em profundidade na árvore de decisão que representa o espaço do problema.

# Ilustrando a busca pela solução: F → Em Frente; E → Ir Esquerda



## Formalizando a técnica

- As soluções são representadas por um vetor de solução: $V = \{v_1, v_2, v_3, ..., v_n\}$.
- Cada elemento $v_i$ do vetor define uma escolha sobre um conjunto de opções
  $S = \{< \text{opção 1} >, < \text{opção 2} >, .., < \text{opção p} >\}$.
- Inicia-se com um vetor vazio.
- Em cada etapa acrescenta-se um valor ao vetor.
- O novo valor é avaliado. Se atingir uma configuração de $V$ que necessariamente não atinge uma solução, o último valor é eliminado e explora-se um novo candidato.

## Restringindo o espaço de solução

- Ao contrário da Força Bruta, o Backtracking não precisa testar toda e qualquer possibilidade de configuração, ele respeita restrições definidas para o problema:
- Restrições Explícitas: Regras que restringem as soluções possíveis para cada $v_i$, está relacionado com o problema e o espaço que o $v_i$ ocupa na solução.
- Restrições Implícitas: Relacionamento entre o $v_i$ e todos os $v_r$, com $r < i$.

# Solução genérica baseada no backtracking

## Algoritmo: Backtracking

**Entrada:** $S$ (Escolhas de decisão), $V$ (conjunto de decisões) e $i$ (nível).
**Saída:** O status da solução

```
Algoritmo BACKTRACKING(S, V, i)
    solucao ← VerificarSolucao(V)
    se solucao = RESOLVIDO então
        Retorna RESOLVIDO
    senão
        i ← i + 1
        escolha ← Decidir(S, i)
        enquanto solucao ≠ RESOLVIDO e escolha ≠ ∅ faça
            V ← V + escolha
            O ← CriarOpcoes(V, i)
            solucao ← Backtracking(O, V, i)
            se solucao ≠ RESOLVIDO então
                V ← V − escolha
                escolha ← Decidir(S, i)
        Retorna solucao
```

## Problema de quebra-cabeça

- O Backtracking é um algoritmo importante para resolver problemas de quebra-cabeças ou outros jogos.
- São vários os movimentos, peças, jogadas, ... que podem ser feitas a cada vez.
- Uma jogada leva a outra até o final, que pode representar uma solução, ou simplesmente um beco sem saída.
- Se for um beco sem saída é necessário retornar e tentar outra jogada.
- Se houver solução, eventualmente encontramos a solução.

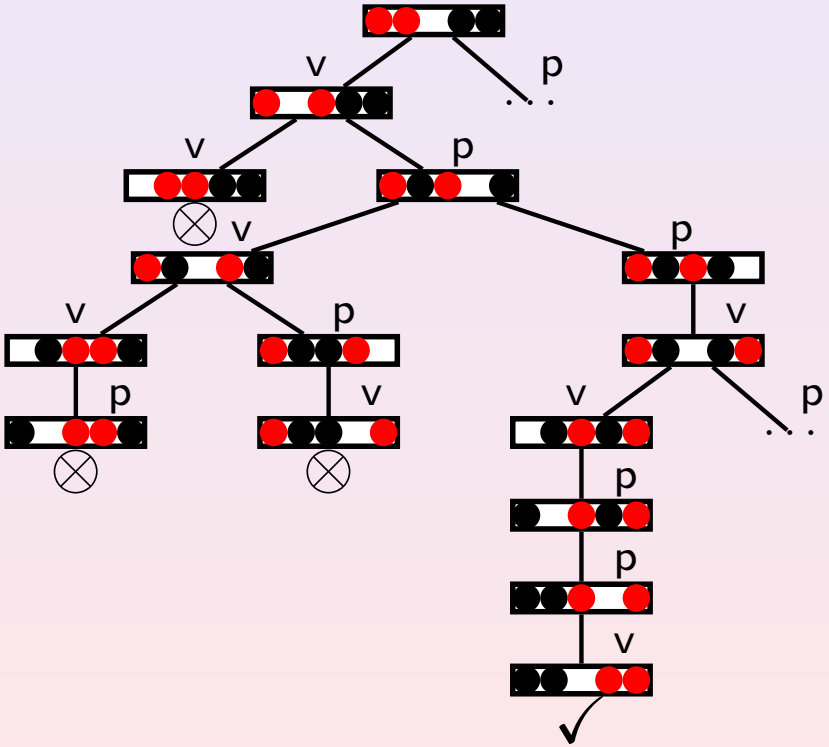
## Jogo das bolas coloridas

Este jogo pode ser encontrado em várias versões e vários tamanhos, esta é uma delas:

### O Jogo: Definição

O jogo inicia com as bolas vermelhas à esquerda e pretas à direita, existe um espaço que separa o conjunto de bolas vermelhas do conjunto de bolas pretas. As bolas vermelhas so se movem para a direita e as bolas pretas só se movem para a esquerda.
Os movimentos possíveis são: Uma bola pode deslizar para o espaço vazio ao seu lado. Uma bola vermelha pode pular uma única bola preta se houver espaço vazio ao lado da bola preta. Uma bola preta pode pular uma única bola vermelha se houver um espaço vazio ao lado da bola vermelha.
O jogo termina quando as bolas vermelhas atingem a direita e as bolas pretas a esquerda.

Árvore do espaço de solução: observe que não abrange todas as opções, pois uma solução foi encontrada antes.



## Parametrizando a solução

- $S = \{v : vermelho, p : preto\}$ fila de opções de escolha para os movimentos.
  - Observe que nem sempre haverá duas escolhas, mas no máximo são estas duas.
- $V = \emptyset$. Inicialmente não há uma proposta de solução.
- $Q = \{v, v, x, p, p\}$. Quadro do jogo, em sua configuração inicial. As funções $Atualiza(Q, escolha)$ e $Desfaz(Q, escolha)$ altera o quadro conforme a jogada.
- $Jogar(Q, O, V)$. Algoritmo de backtracking que procura a solução.
- $CriarOpcoes(Q) \rightarrow O$. Função que retorna uma fila com as escolhas de um quadro.
- $Desenfileira(S) \rightarrow escolha$. Função que pega uma escolha da fila de opções.
- $Verificar(Q) \rightarrow solucao \in \{RESOLVIDO, ERRADO\}$. Função que verifica se o jogo está concluído ou não.

## Algoritmo para o jogo usando backtracking

**Algoritmo** JOGAR(S,V)
    $solucao \leftarrow Verificar(Q)$
    **se** $solucao = RESOLVIDO$ **então**
        **Retorna** *RESOLVIDO*
    **senão**
        $escolha \leftarrow Desenfileira(S)$
        **enquanto** $solucao \neq RESOLVIDO$ **e** $escolha \neq \emptyset$ **faça**
            $V \leftarrow V + escolha$; $Atualiza(Q, escolha)$
            $O \leftarrow CriarOpcoes(Q)$
            $solucao \leftarrow Jogar(Q, O, V)$
            **se** $solucao \neq RESOLVIDO$ **então**
                $V \leftarrow V - escolha$; $Desfaz(Q, escolha)$
                $escolha \leftarrow Desenfileira(S)$
        **Retorna** *solucao*

Fica como exercício construir as funções indicadas no algoritmo!

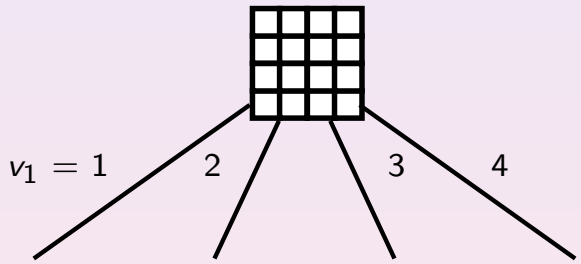
## Descrição do problema:

**Problema: 8 Rainhas**

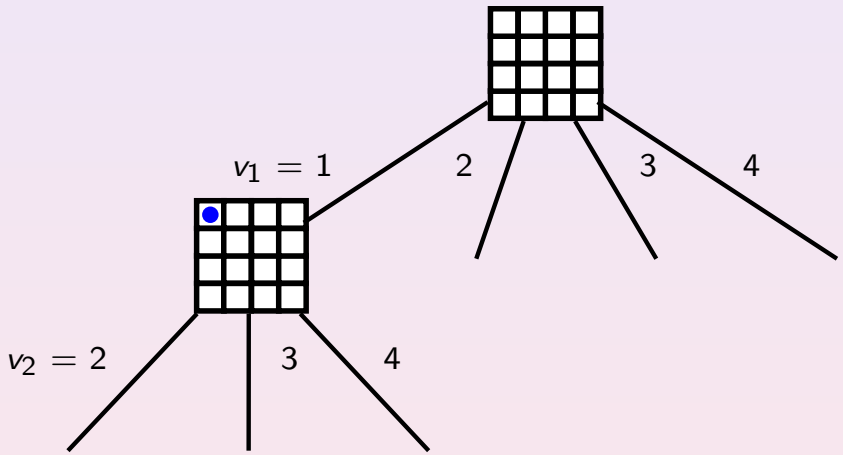
Colocar 8 rainhas em um tabuleiro de xadrez de modo que nenhuma rainha ataque uma outra.

- Solução: $V = \{v_1, v_2, \ldots, v_8\}$, onde $v_i$ indica a coluna em que será colocada a rainha da linha $i$.
- Força Bruta: Tempo da solução: $C_8^{64} = \frac{64!}{8! \cdot 63!} = 4.426.165.368$.
- Regras explícitas: cada rainha é colocada em uma linha distinta; Tempo da solução: $8^8 = 16.777.216$
- Regras implícitas: Tempo da solução: $< 8! = 40.320$
  - $v_i \neq v_j, \forall (i, j)$;
  - Duas rainhas não podem ocupar a mesma diagonal

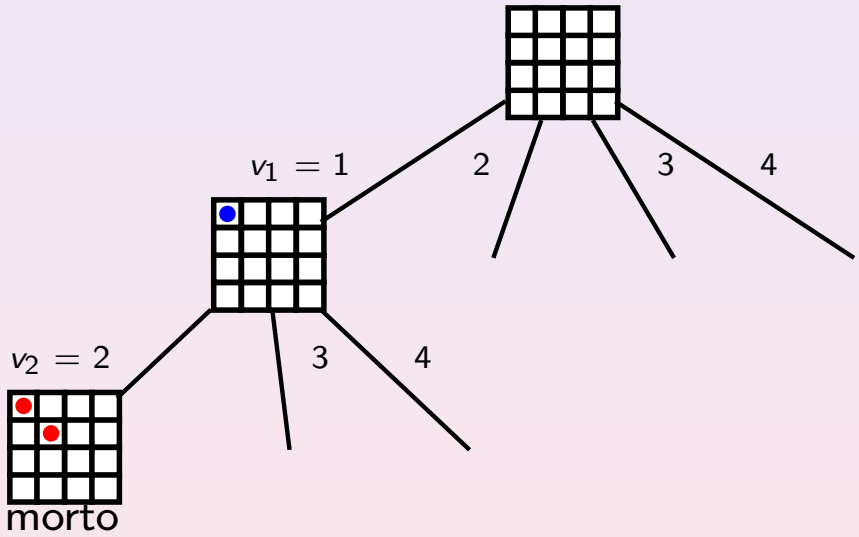
## Exemplificando para um tabuleiro $4 \times 4 : V = \emptyset$



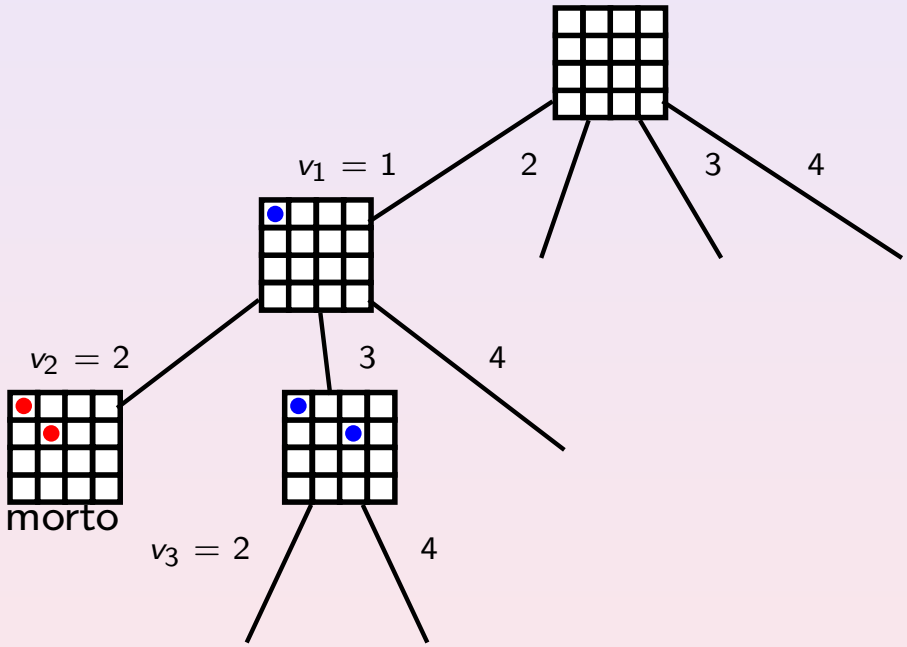
## Exemplificando para um tabuleiro $4 \times 4 : V = \{1\}$



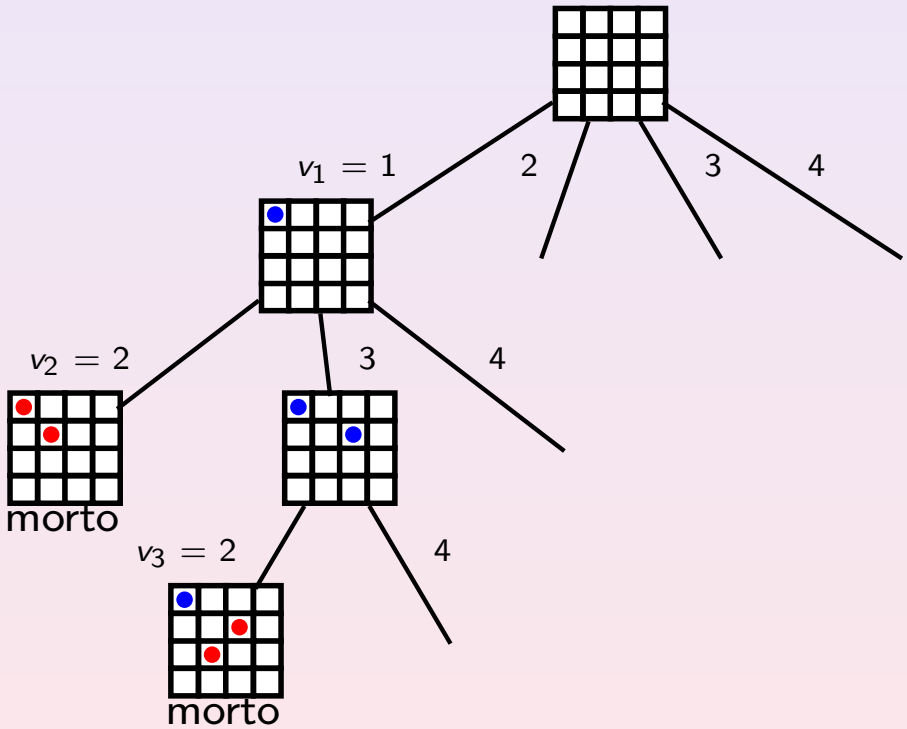
## Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 2\}$



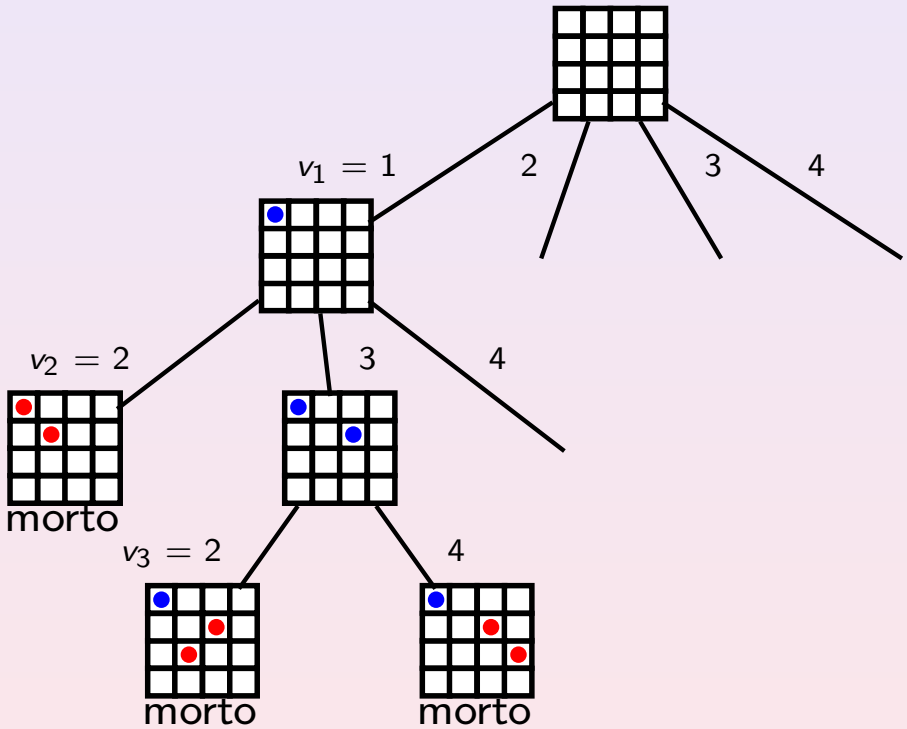
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 3\}$



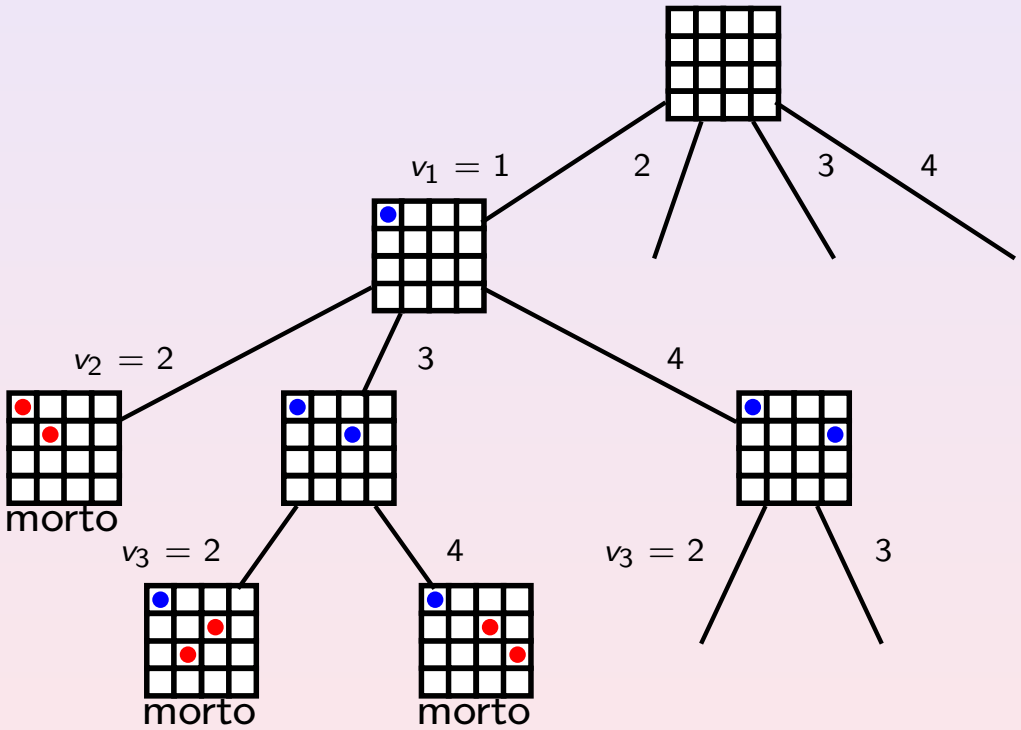
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 3, 2\}$



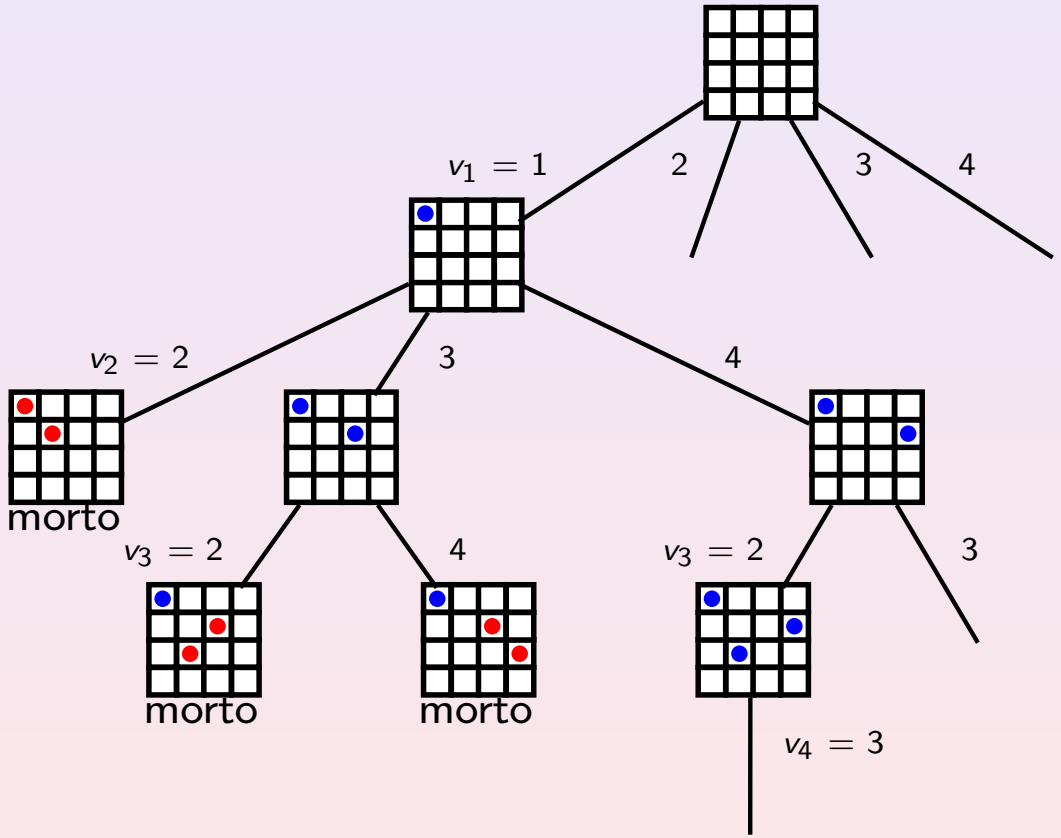
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 3, 4\}$



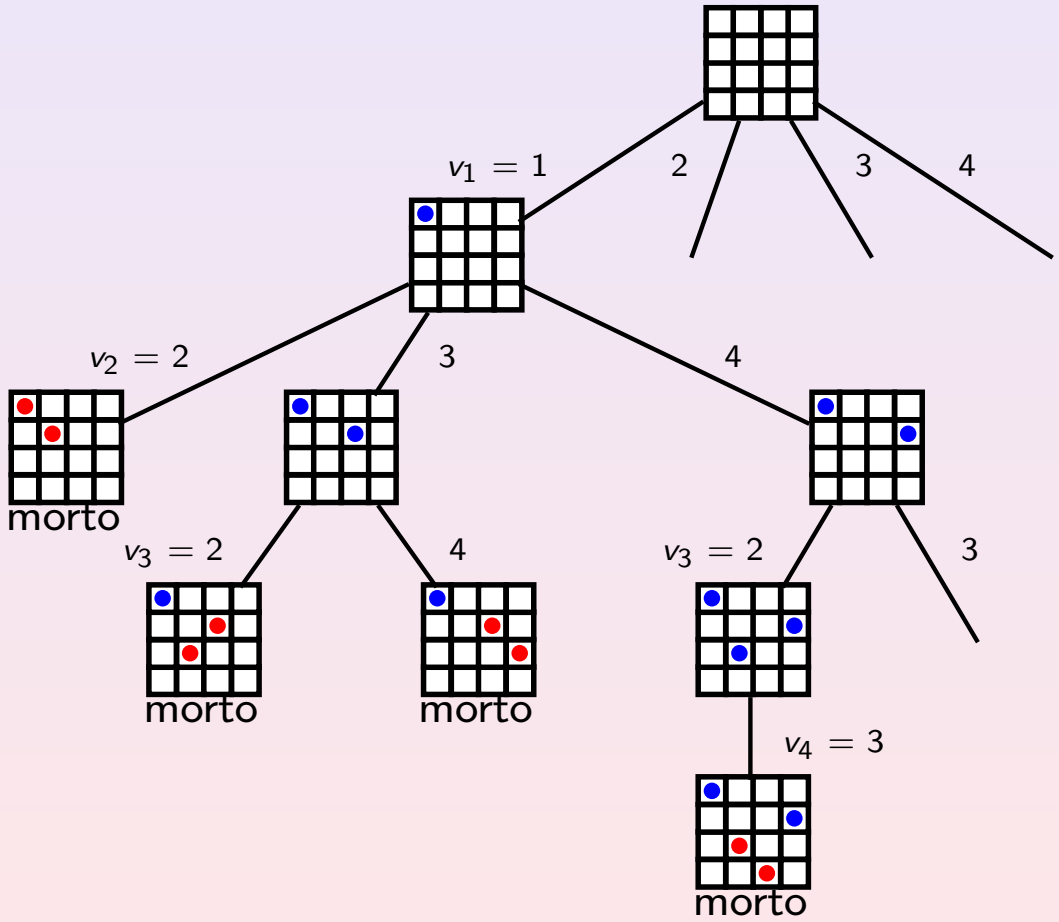
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 4\}$



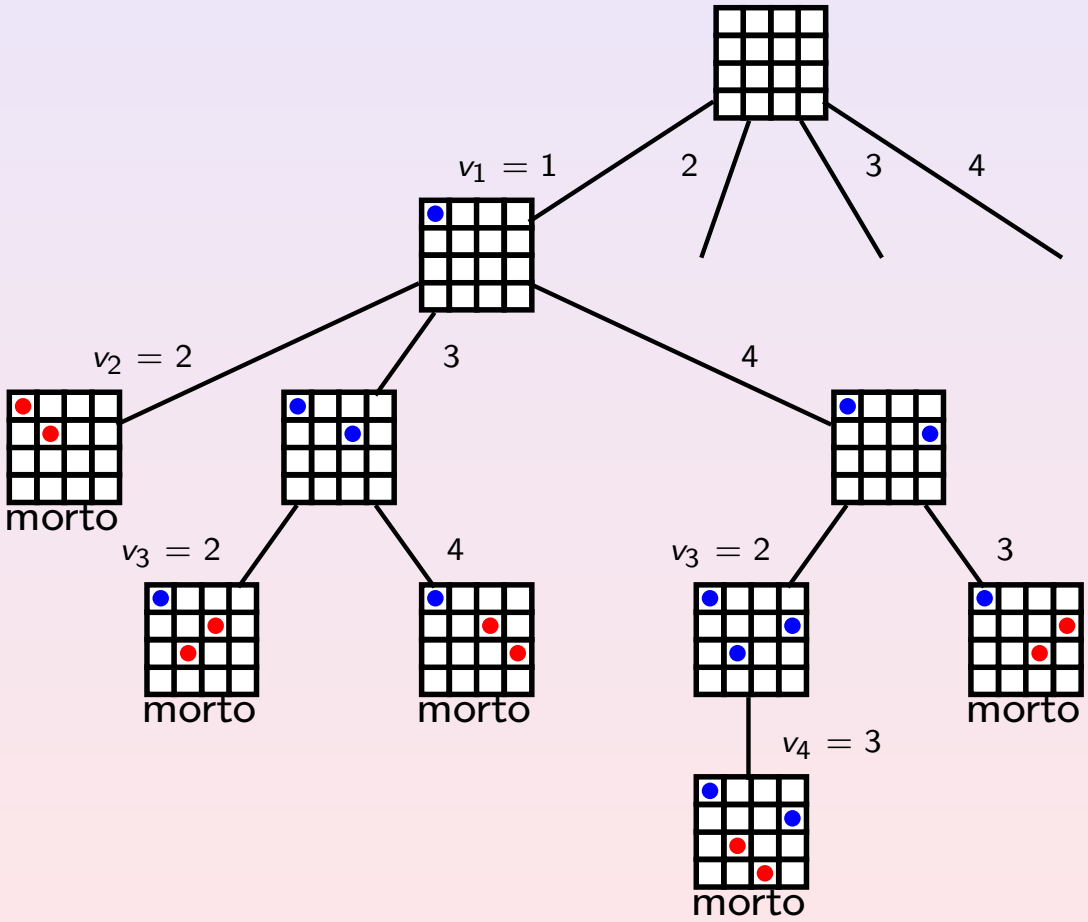
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 4, 2\}$



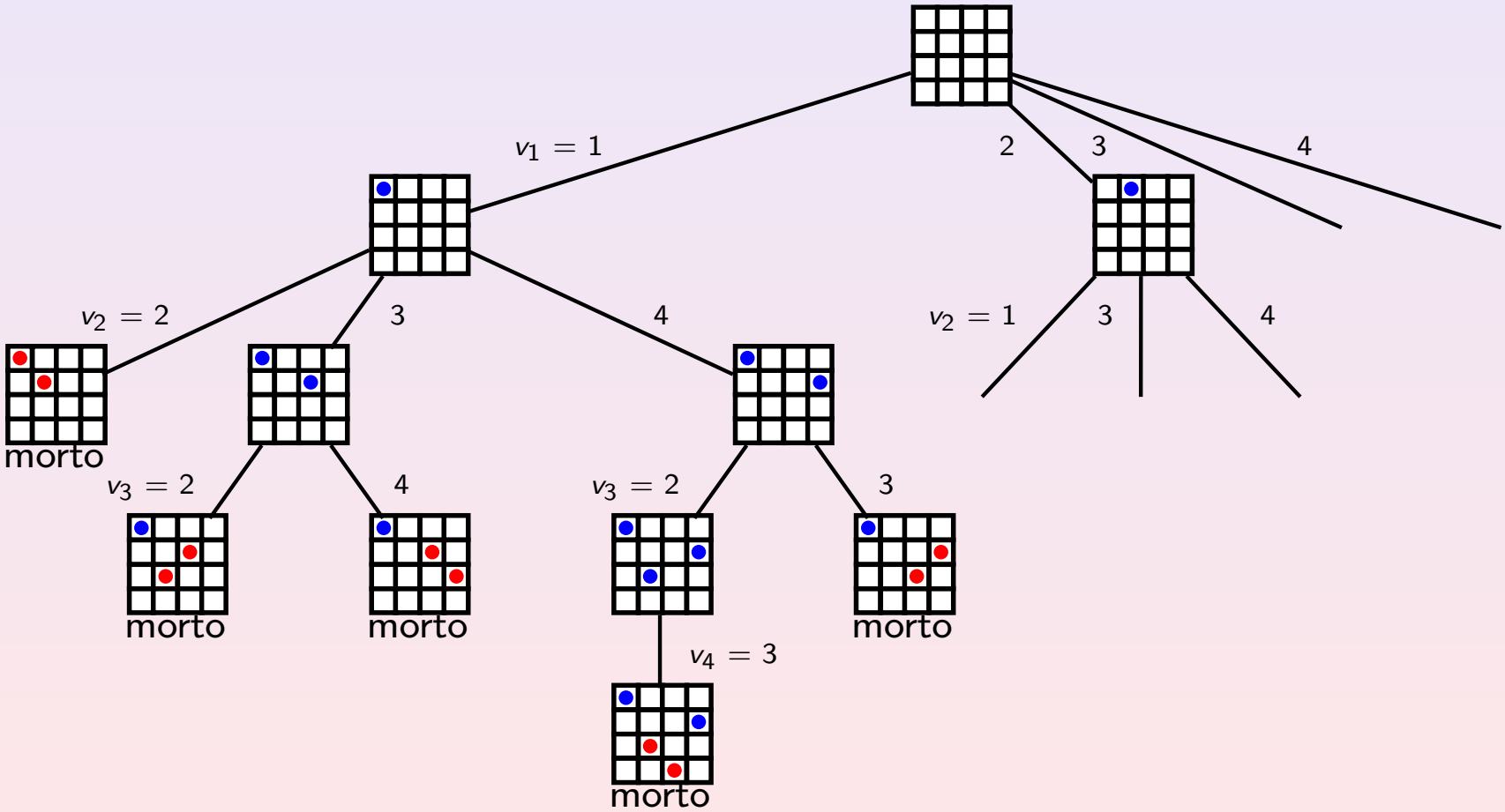
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 4, 2, 3\}$



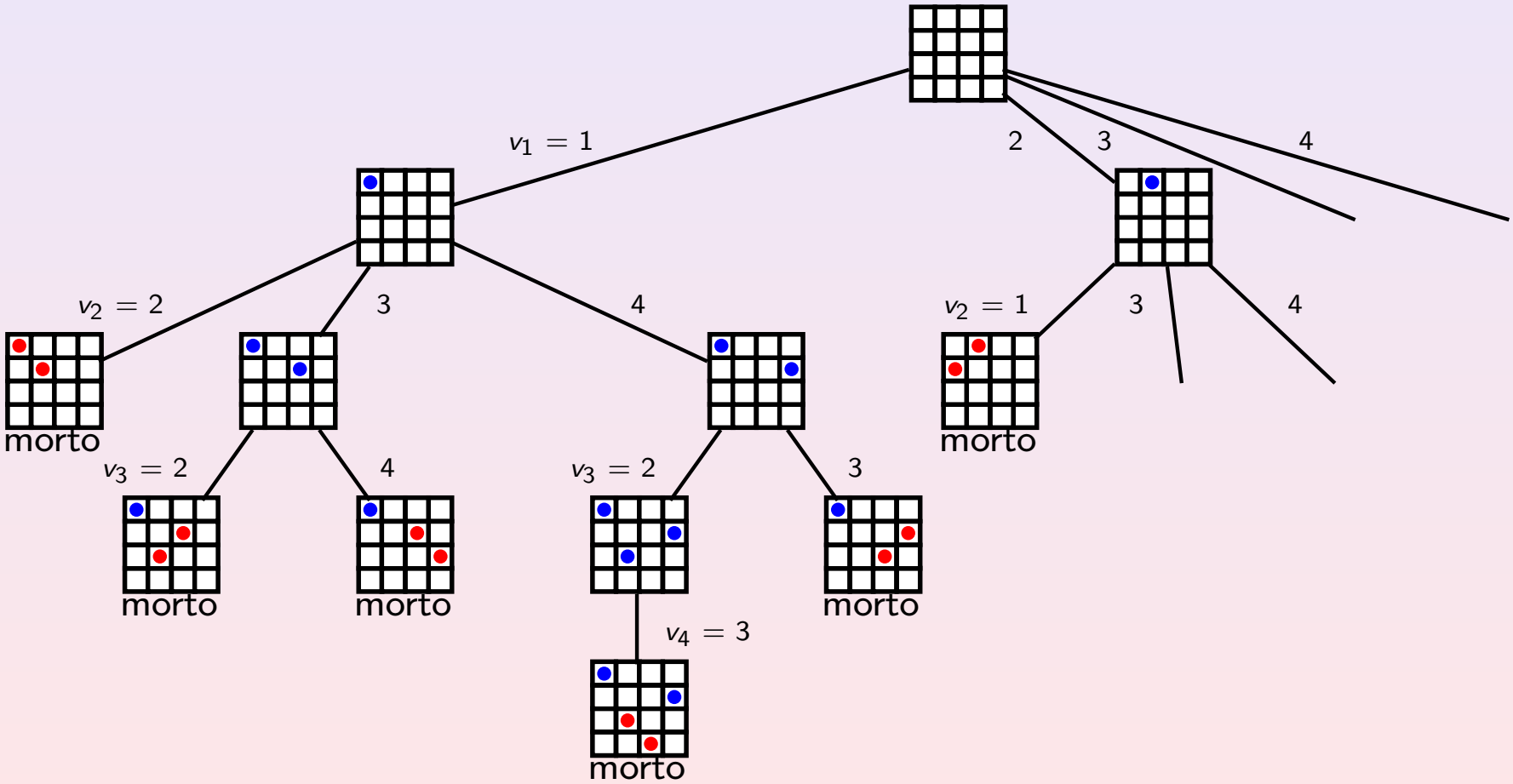
# Exemplificando para um tabuleiro $4 \times 4 : V = \{1, 4, 3\}$



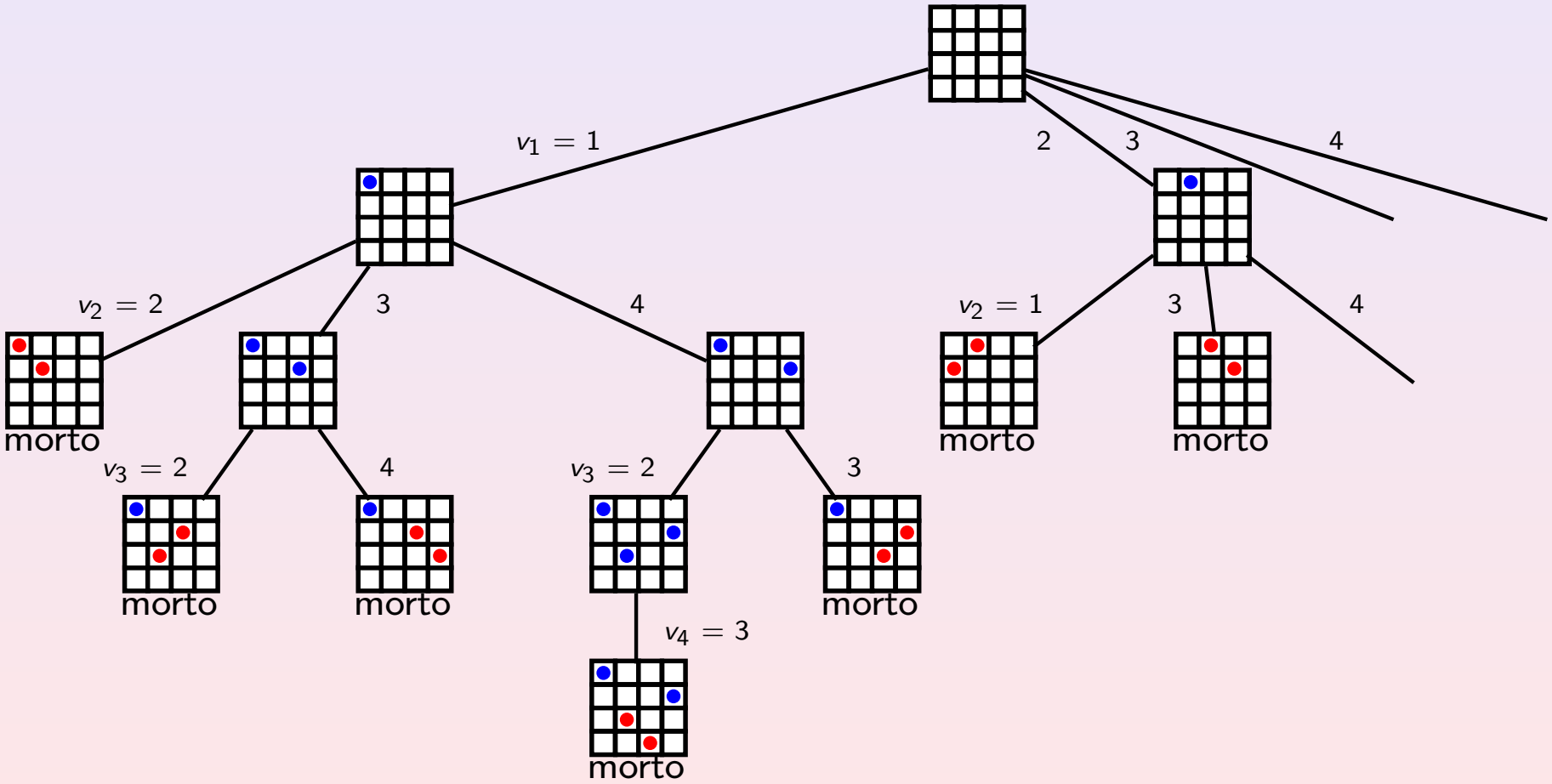
# Exemplificando para um tabuleiro $4 \times 4 : V = \{2\}$



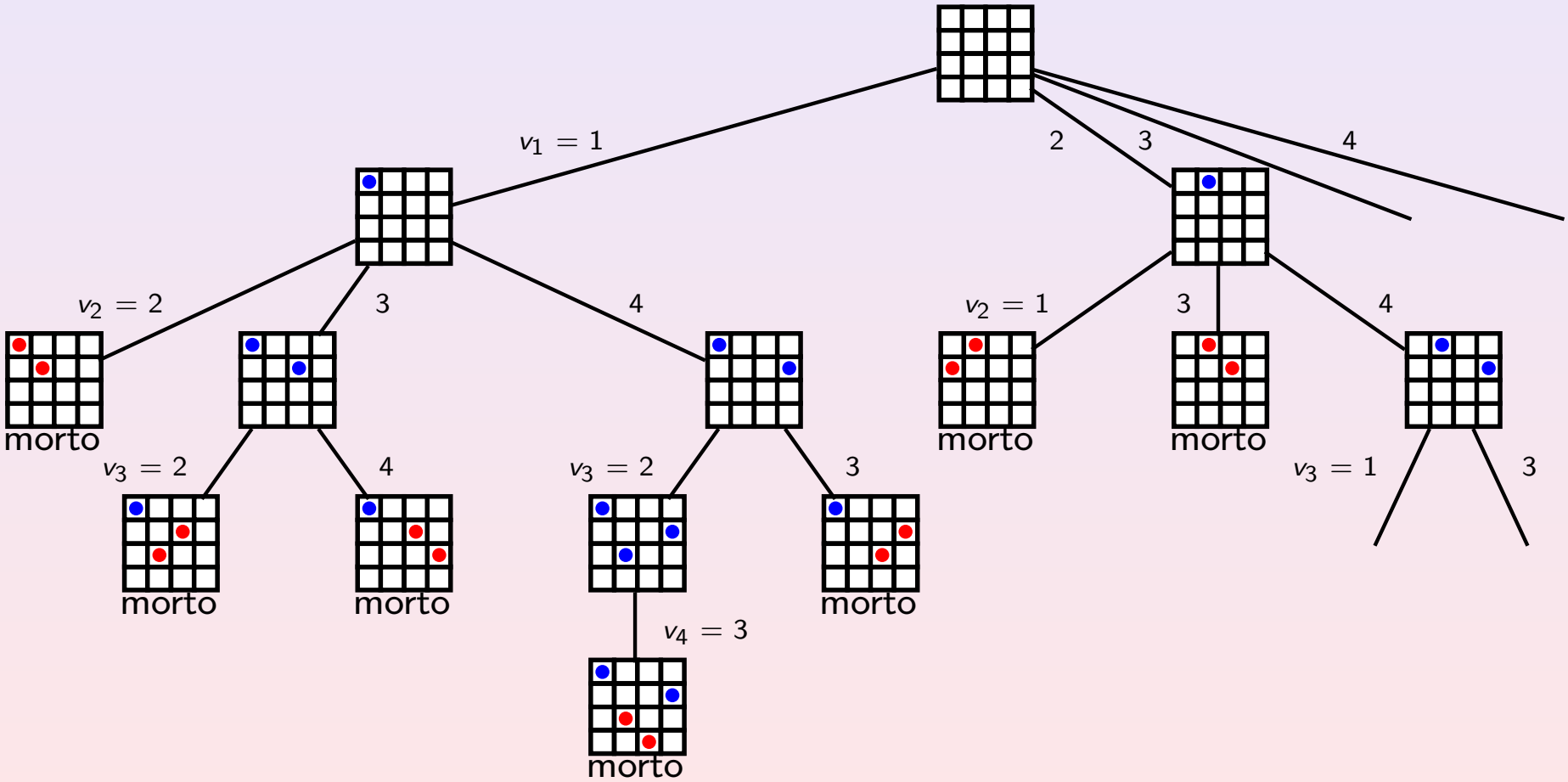
# Exemplificando para um tabuleiro $4 \times 4 : V = \{2, 1\}$



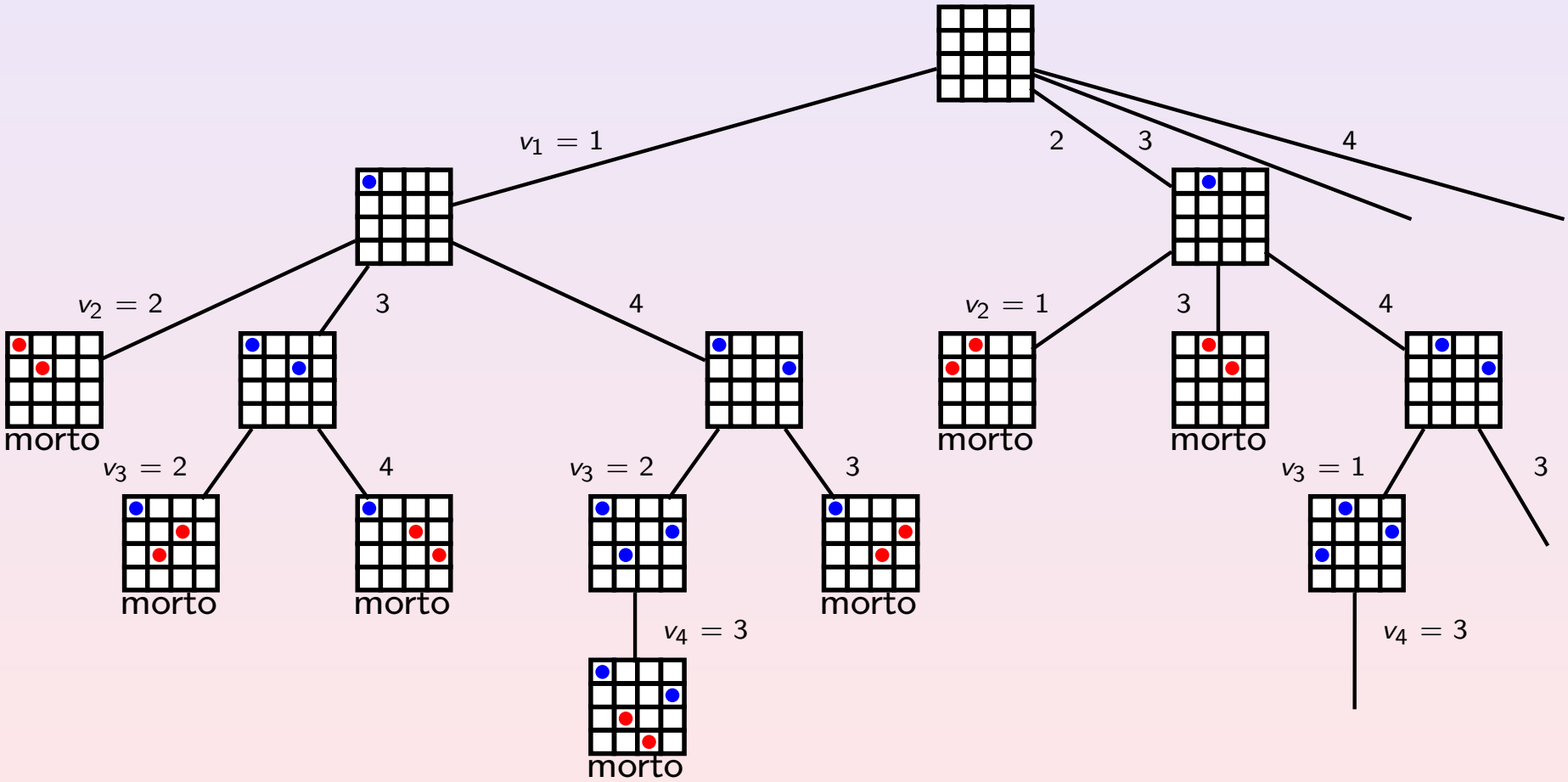
# Exemplificando para um tabuleiro $4 \times 4 : V = \{2, 3\}$



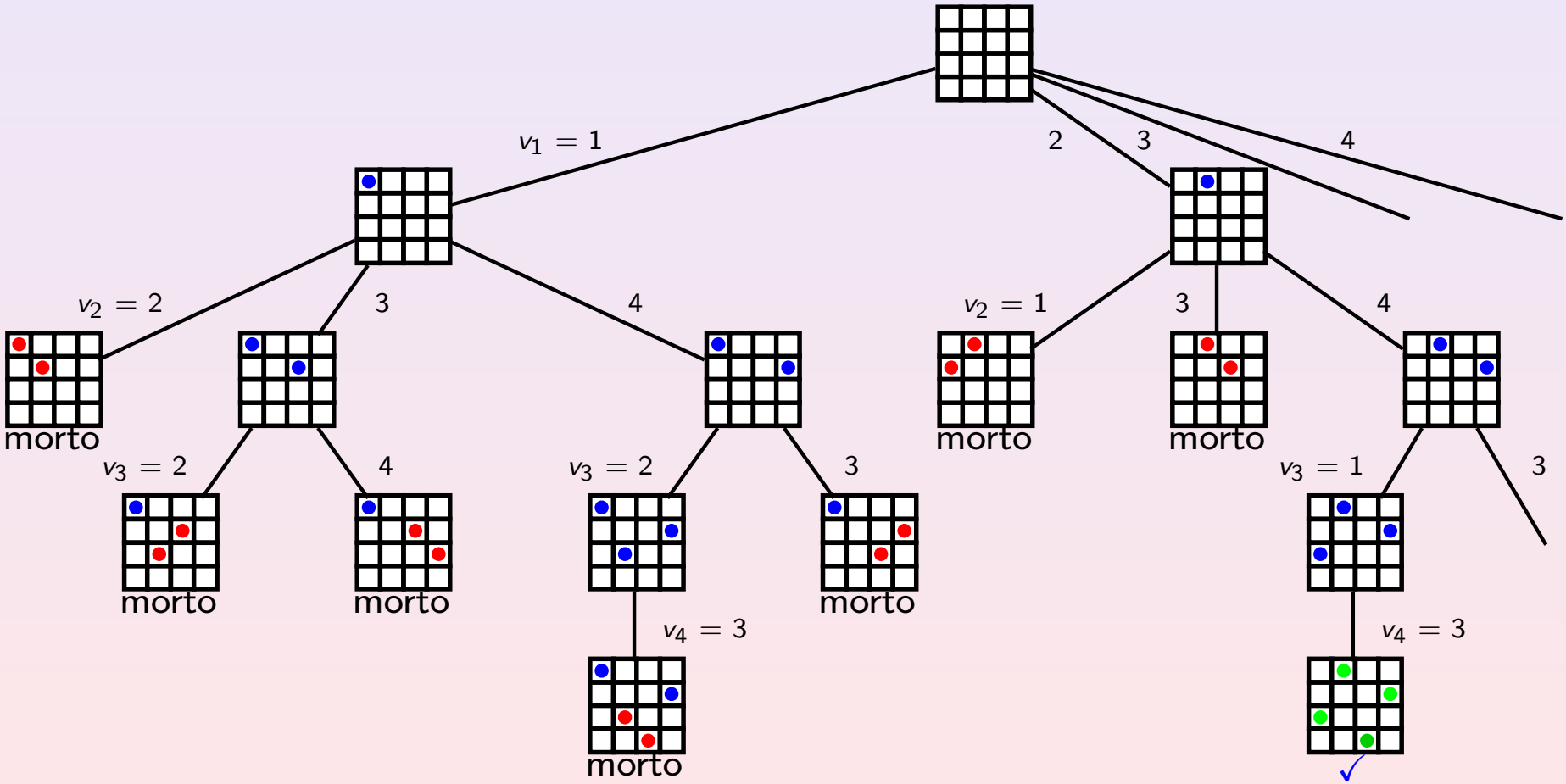
# Exemplificando para um tabuleiro $4 \times 4 : V = \{2, 4\}$



# Exemplificando para um tabuleiro $4 \times 4 : V = \{2, 4, 1\}$



# Exemplificando para um tabuleiro $4 \times 4 : V = \{2, 4, 1, 3\}$



## Parametrizando a solução

- $S = \{1, 2, 3, 4\}$ fila de opções de escolha para os movimentos.
  - Observe que nem sempre haverá todas escolhas.
  - Podemos restringir fazendo: $i \neq j$, $i$ e $j$ não estão na mesma linha. $v_i \neq v_j$, $i$ e $j$ não estão na mesma coluna e $|v_i - v_j| \neq |i - j|$, $i$ e $j$ não estão na mesma diagonal.
- $V = \emptyset$. Inicialmente não há uma proposta de solução.
- $Rainha(O, V)$. Algoritmo de backtracking que procura a solução.
- $CriarOpcoes(V) \rightarrow O$. Função que retorna uma fila com as escolhas a partir do que já foi colocado.
- $Verificar(V) \rightarrow solucao \in \{RESOLVIDO, ERRADO\}$. Função que verifica se o jogo está concluído ou não.

## Algoritmo para o jogo usando backtracking

**Algoritmo** RAINHA(S,V)
    $solucao \leftarrow Verificar(V)$
    **se** $solucao = RESOLVIDO$ **então**
        **Retorna** $RESOLVIDO$;
    **senão**
        $escolha \leftarrow Desenfileira(S)$
        **enquanto** $solucao \neq RESOLVIDO$ **e** $escolha \neq \emptyset$ **faça**
            $V \leftarrow V + escolha$
            $O \leftarrow CriarOpcoes(V)$
            $solucao \leftarrow Rainha(O, V)$
            **se** $solucao \neq RESOLVIDO$ **então**
                $V \leftarrow V - escolha$
                $escolha \leftarrow Desenfileira(S)$
        **Retorna** $solucao$

Fica como exercício construir as funções indicadas no algoritmo!

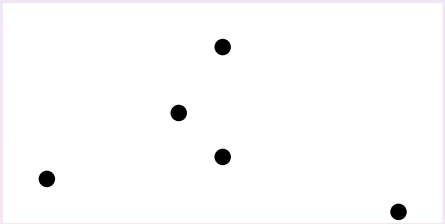
## Problema da Envoltória Convexa

Dado um conjunto de pontos no plano, construir o menor polígono que contenha todos os pontos.

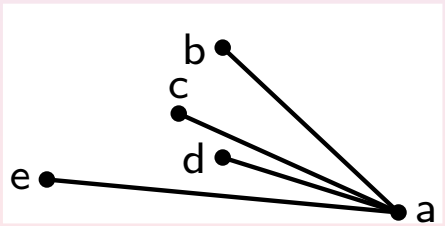
- Uma forma de interpretar a solução seria colocar uma fita elástica no entorno dos pontos, a envoltória fica determinada pelos pontos que formam os vértices.
- Para a solução por Backtracking deve-se considerar o ponto com o maior valor para a coordenada $x$. Deve-se ordenar os demais pontos em ângulos crescentes com relação ao ponto extremo.
- Fazer uma varredura nos pontos de acordo com a ordem criada, adicionando cada um como uma solução parcial e fazendo um backtracking para atingir um determinado ponto se algum ângulo for superior a $180^o$.

## Exemplificando a solução

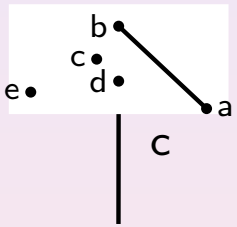
- vamos considerar os seguintes pontos



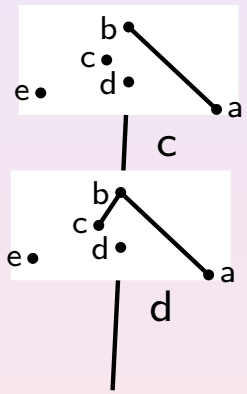
- A ordenação resulta em:



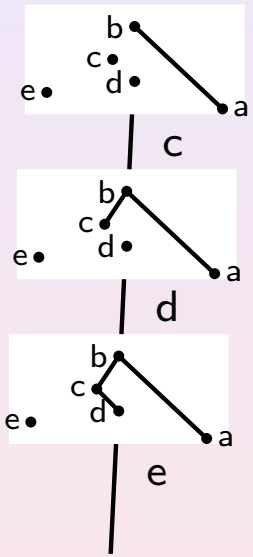
# Aplicando o algoritmo no exemplo



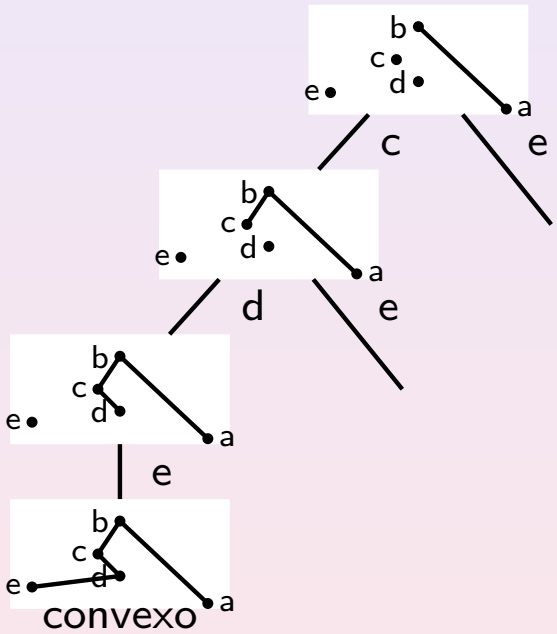
# Aplicando o algoritmo no exemplo



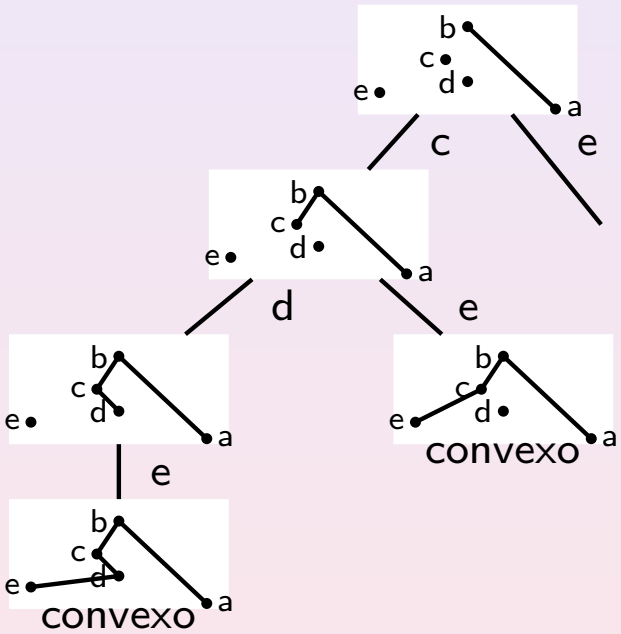
# Aplicando o algoritmo no exemplo



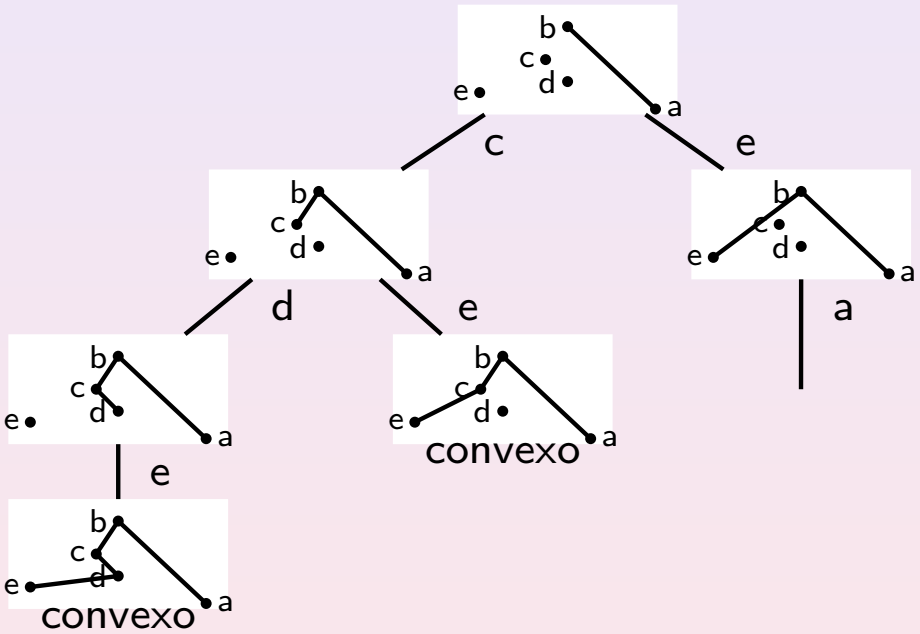
# Aplicando o algoritmo no exemplo



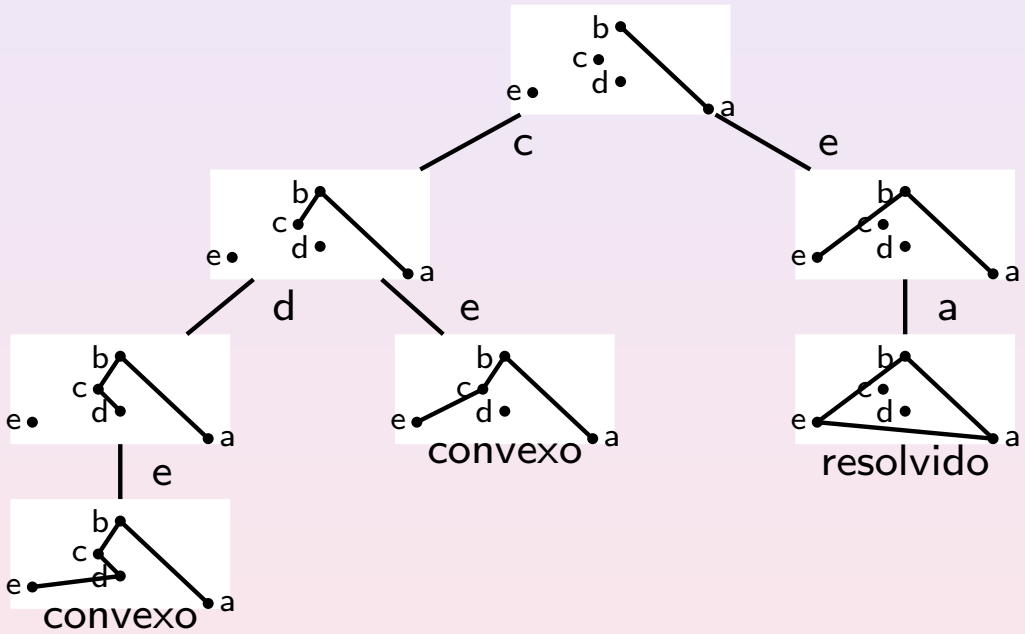
# Aplicando o algoritmo no exemplo



# Aplicando o algoritmo no exemplo



# Aplicando o algoritmo no exemplo



## Vantagens e Desvantagens

- A técnica de Backtracking é melhor que a força bruta, pois nesta técnica se consideram as restrições da construção.
- Muitas soluções não são pesquisadas pois a árvore do espaço de soluções não é completa, alguns ramos não foram seguidos.
- Ainda assim, o algoritmo não é eficiente. O problema das $n$-rainhas é $O(n!)$, o que é péssimo, porém é melhor que $O(n^n)$
- Ao aplicar a técnica é necessário tomar cuidado para que a seqüência de soluções não retorne a um ponto do problema já visitado (imagine um labirinto que possua uma praça).
- Esta é uma técnica a ser aplicada em último recurso.

## Atividades

1. Fazer a lista 6.