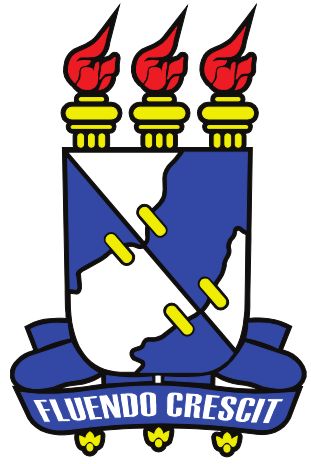


Projeto e Análise de Algoritmos

Backtracking e Branch-and-Bound

Bruno Prado

Departamento de Computação / UFS

Introdução

- Classe de problemas difíceis para algoritmos
  - Complexidade com crescimento exponencial
  - Aplicações demandam tempo de resposta aceitável
  - São problemas muito relevantes para computação

Busca exaustiva $\longleftrightarrow$ *Espaço exponencial*

Introdução

- Como evitar a busca exaustiva por soluções?
  - Utilização de técnicas de backtracking e branch-and-bound para reduzir este espaço de soluções
  - Neste paradigma, as soluções candidatas são geradas de forma incremental, buscando atender as restrições do problema e gerar novas soluções baseadas nestas soluções promissoras
  - Apesar de bons resultados práticos, em uma análise de pior caso estas técnicas podem recair no desempenho de uma busca exaustiva

Introdução

- O que é backtracking?
  - Back = Voltar + Tracking = Encaminhamento
  - O algoritmo constrói um conjunto de soluções parciais, sempre avaliando se as restrições impostas pelo problema são satisfeitas
  - Quando uma solução parcial parece promissora, ou seja, atende as restrições, são geradas novas soluções parciais a partir desta solução
  - Se nenhuma solução obtida atende as restrições, o algoritmo deve retroceder e avaliar a próxima solução parcial ainda não explorada, caso exista

Introdução

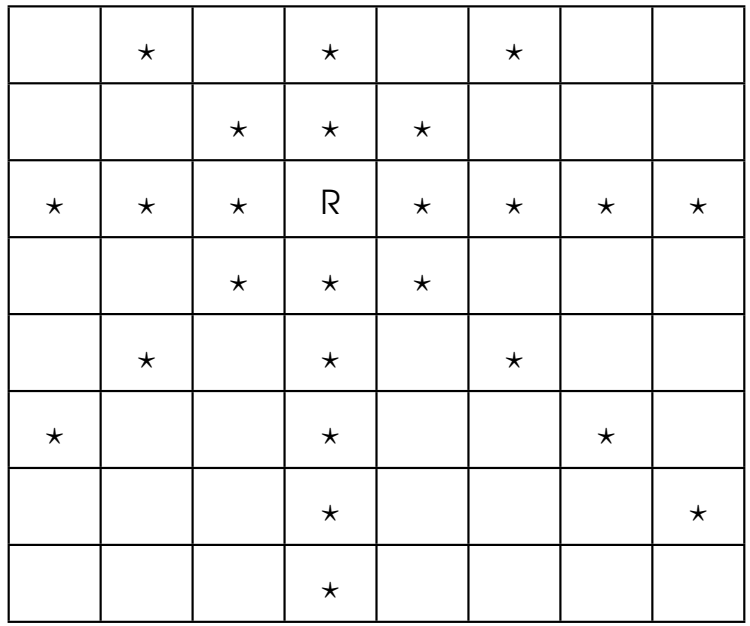
- O que é branch-and-bound?
  - Branch = Desviar + Bound = Limitar
  - Em problemas de otimização, as soluções geradas procuram minimizar ou maximizar alguma métrica do problema, atendendo as restrições do problema
  - As soluções parciais geradas são avaliadas e as opções inválidas são descartadas
  - O valor da melhor solução obtida até o momento armazenada, para que seja verificado se as próximas soluções geradas são melhores

Backtracking

- Princípios de funcionamento
  - Busca em profundidade
    - As soluções candidatas ou promissoras que atendem as restrições são exploradas primeiro
    - É feito o incremento de solução sempre atendendo as regras impostas pelo problema
    - Este processo pode levar a uma solução completa, mas não existe uma garantia
  - Soluções que não atendem as restrições são desprezados durante o processo de busca, reduzindo o espaço de busca do algoritmo

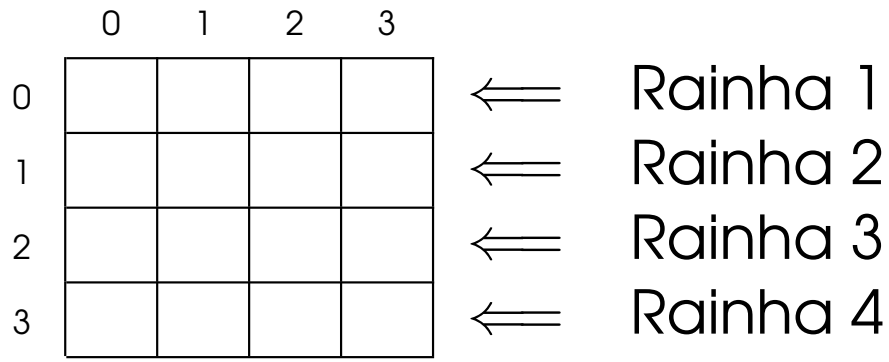
Backtracking

- Problema das *n*-rainhas
  - Consiste em colocar *n* rainhas em um tabuleiro de dimensões $n \times n$, de forma que nenhuma das rainhas estejam em linha de ataque
  - As linhas de ataque das rainhas são definidas por suas linha horizontais, verticais e diagonais do tabuleiro, sem limite de alcance



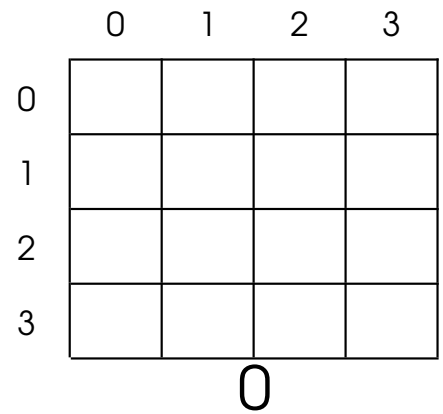
Backtracking

- Problema das 4-rainhas
  - Tabuleiro com dimensões 4×4
  - Cada rainha é posicionada em uma linha



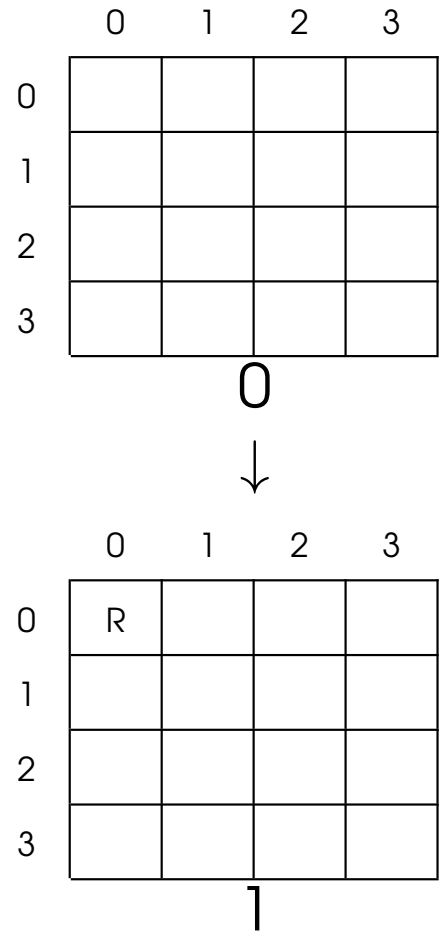
Backtracking

- Algoritmo de busca com backtracking
  - Solução 0: sem nenhuma rainha posicionada
  - Sem violações de ataque das rainhas



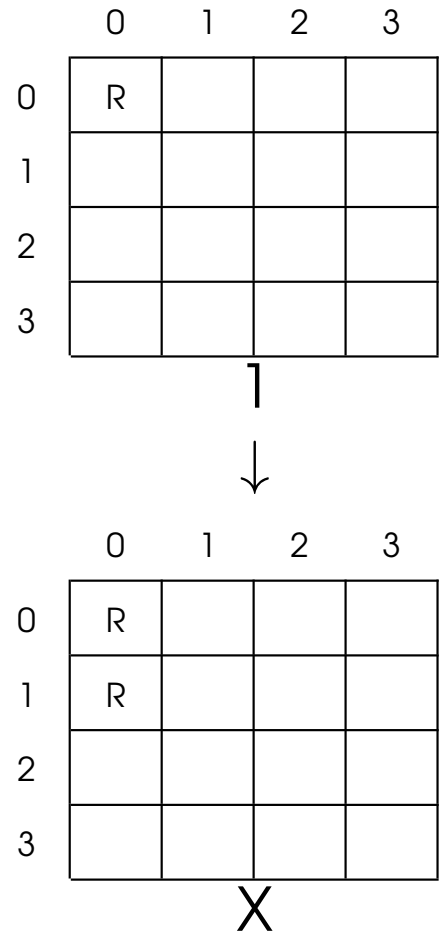
Backtracking

- Algoritmo de busca com backtracking
  - Solução 1: rainha 1 é posicionada em (0, 0)
  - Sem violações de ataque das rainhas



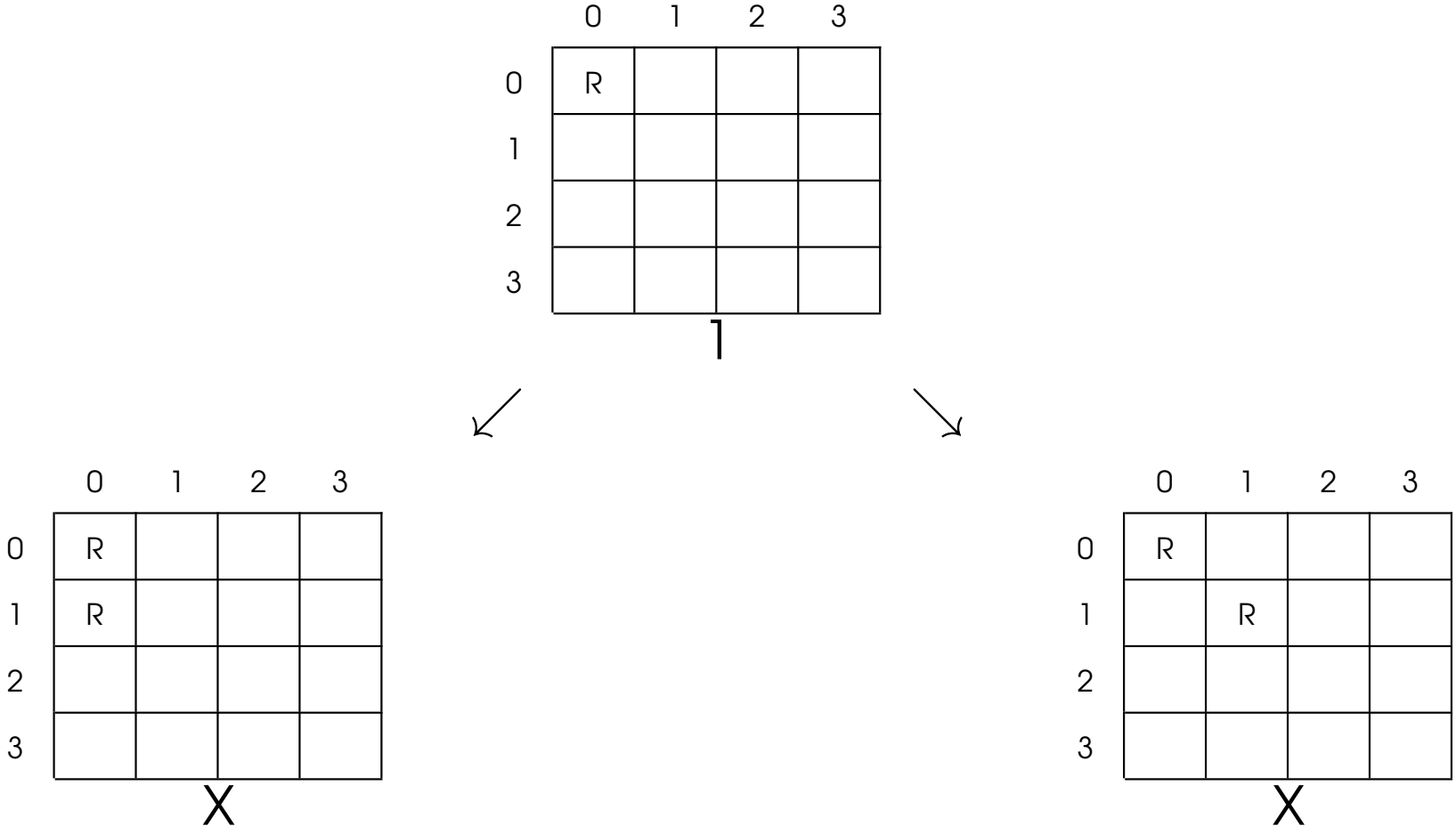
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 2 é posicionada em (1, 0)
  - Conflito: rainhas 1 e 2 em linha de ataque



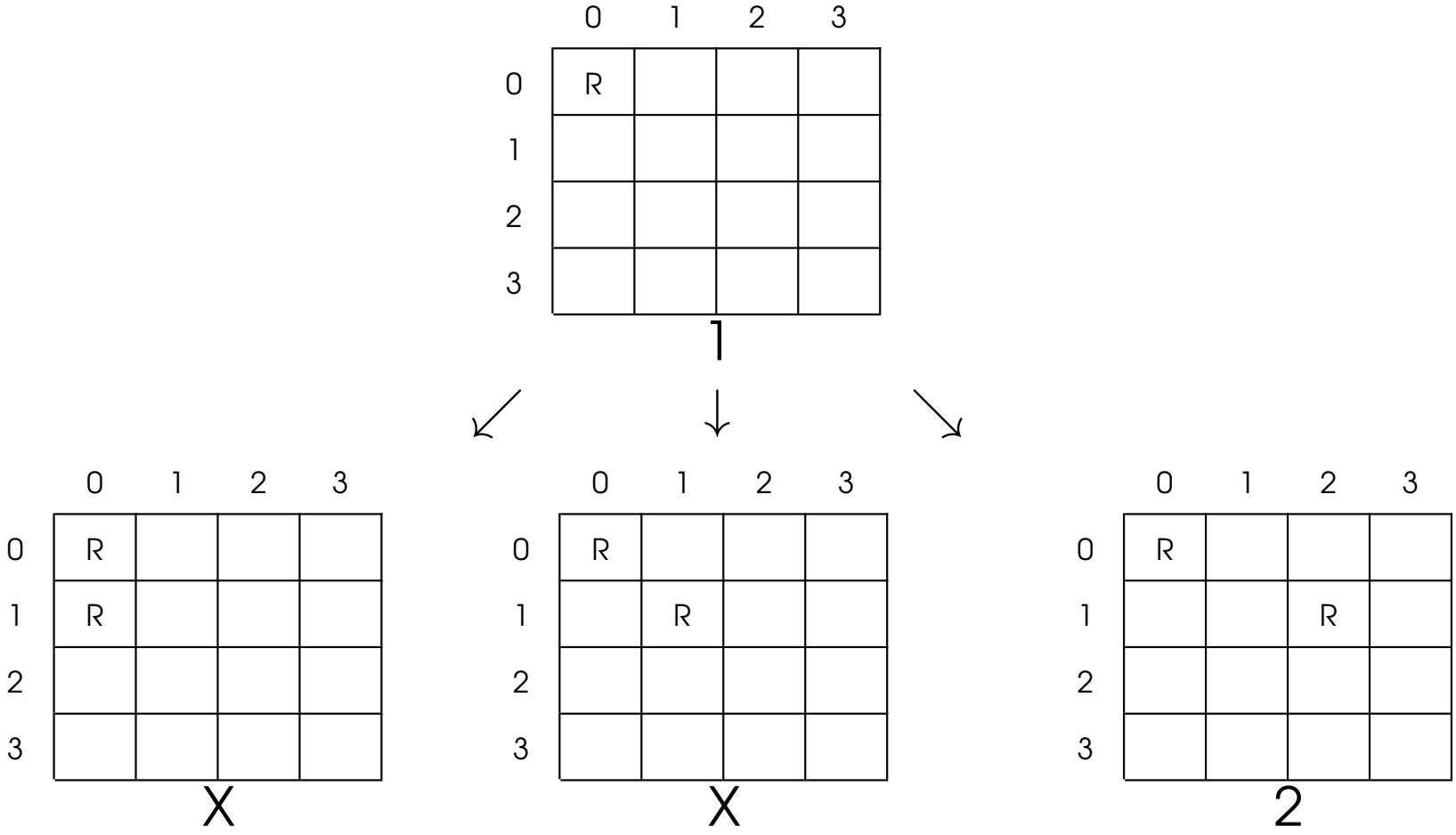
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 2 é posicionada em (1, 1)
  - Conflito: rainhas 1 e 2 em linha de ataque



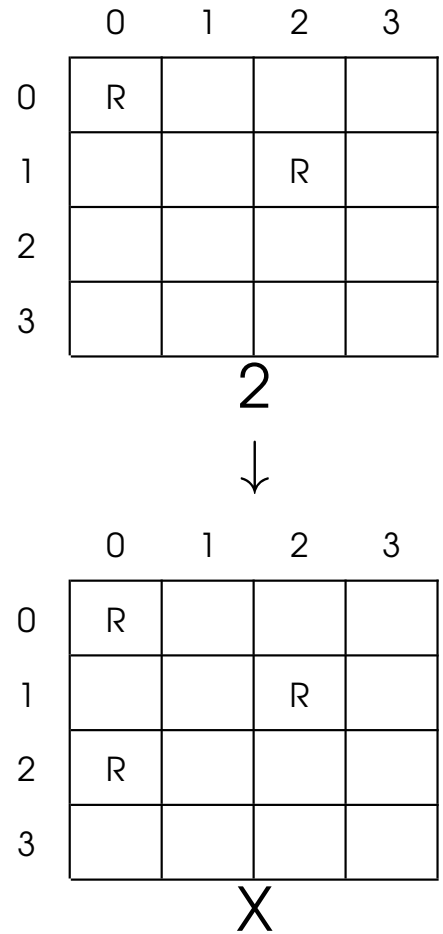
Backtracking

- Algoritmo de busca com backtracking
  - Solução 2: rainha 2 é posicionada em (1, 2)
  - Sem violações de ataque das rainhas



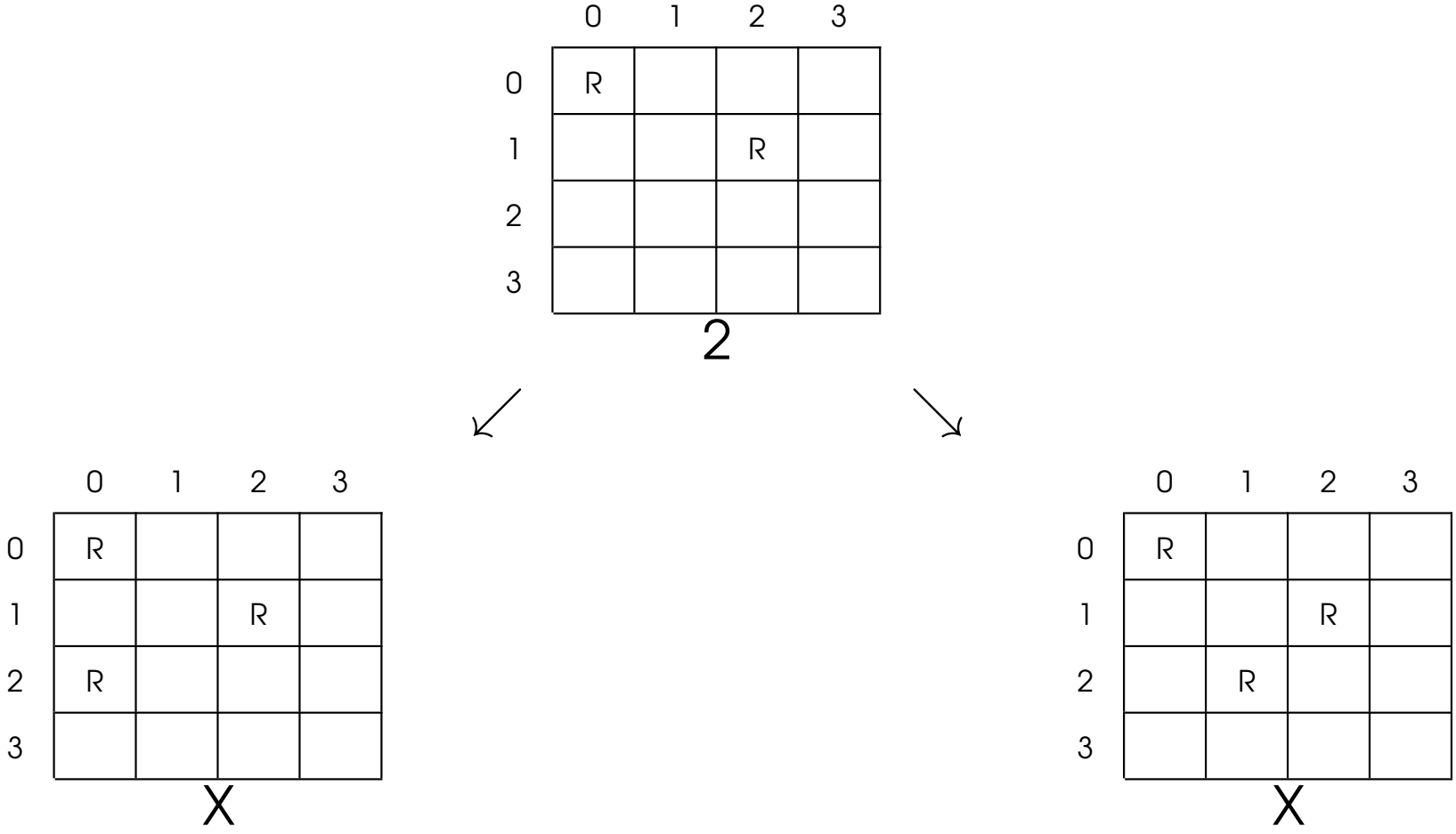
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 3 é posicionada em (2, 0)
  - Conflito: rainhas 1 e 3 em linha de ataque



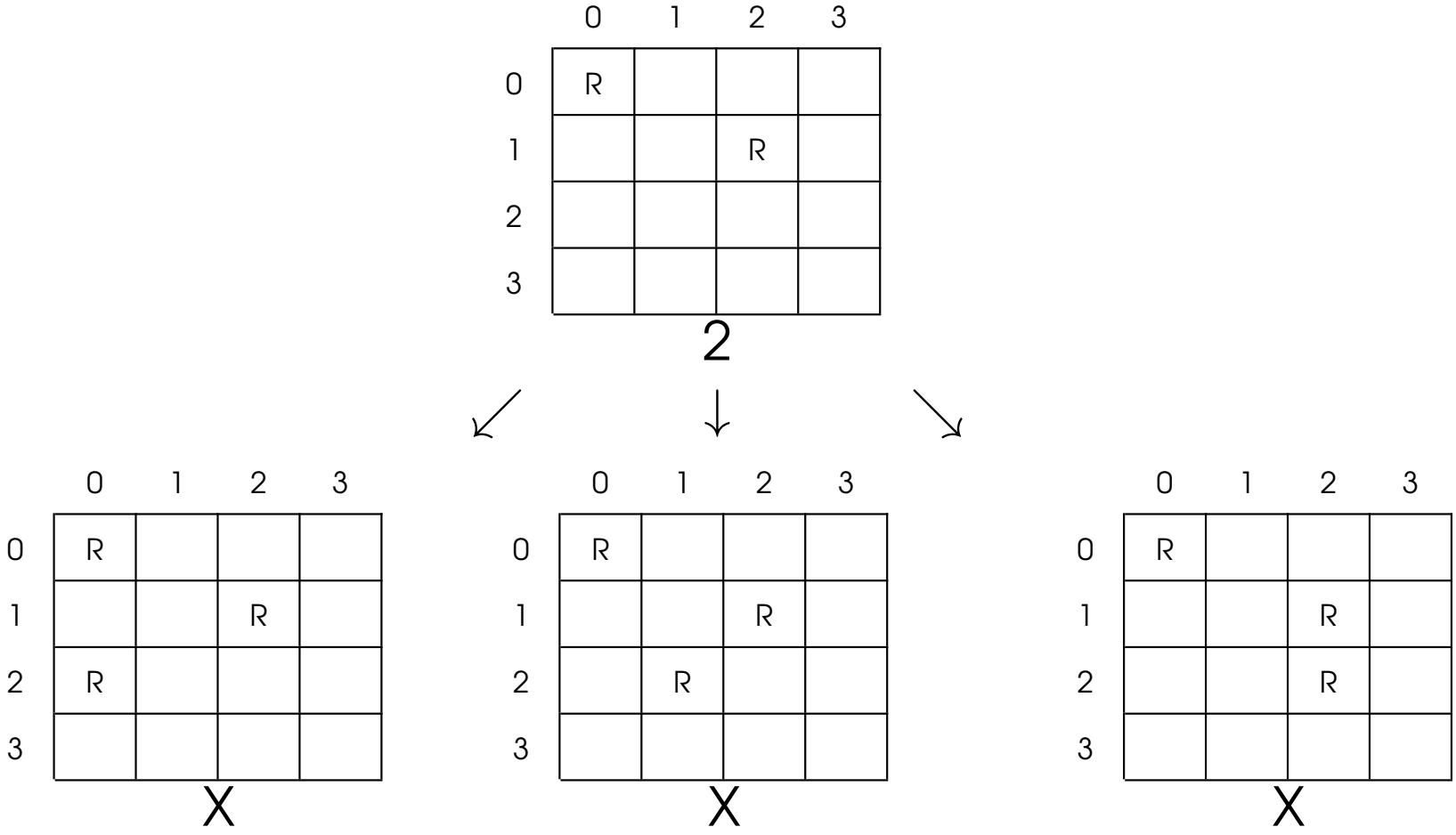
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 3 é posicionada em (2, 1)
  - Conflito: rainhas 2 e 3 em linha de ataque



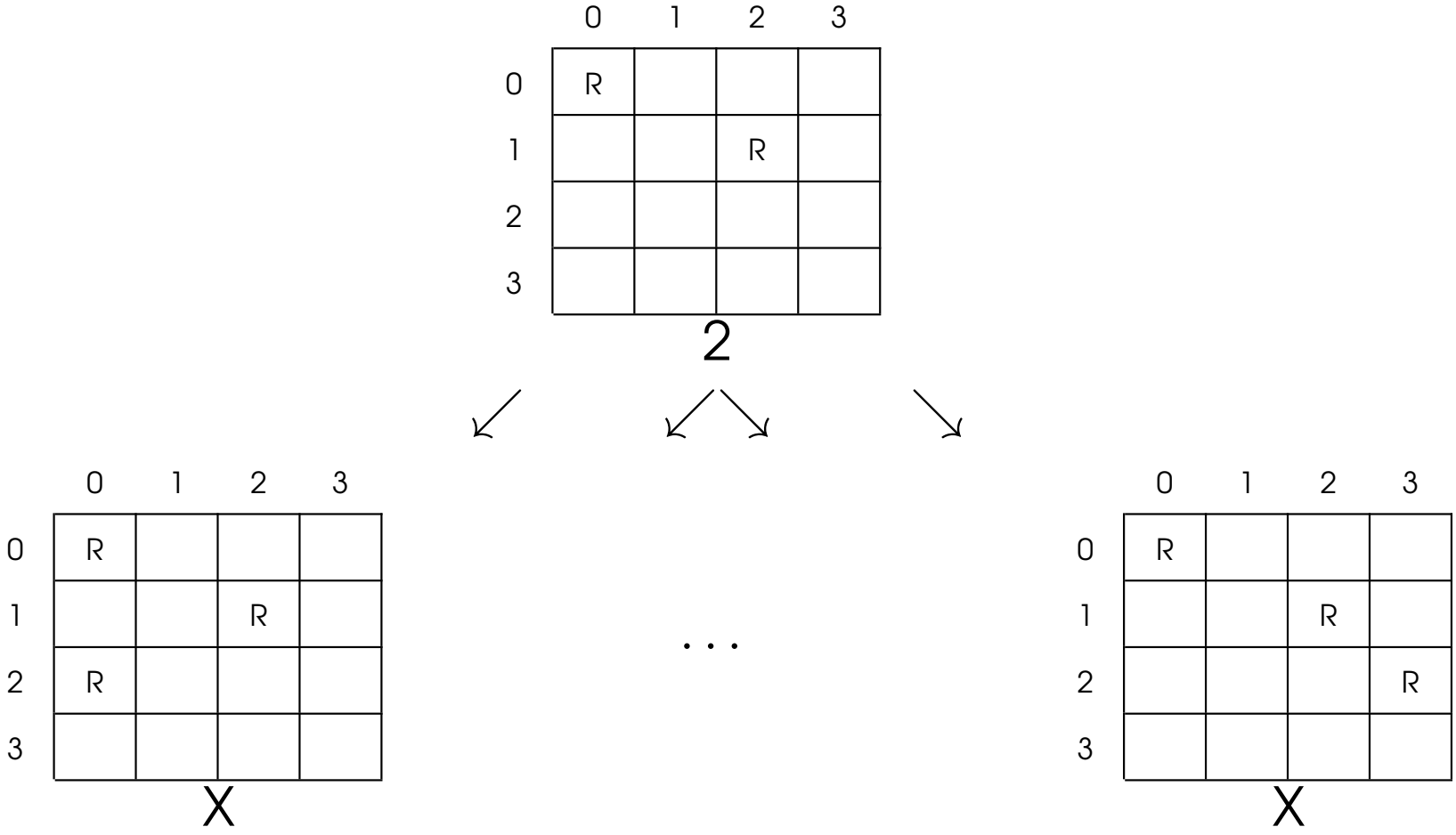
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 3 é posicionada em (2, 2)
  - Conflito: rainhas 1, 2 e 3 em linha de ataque



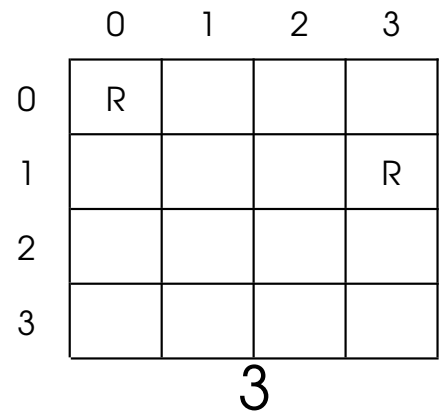
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 3 é posicionada em (2, 3)
  - Conflito: rainhas 2 e 3 em linha de ataque



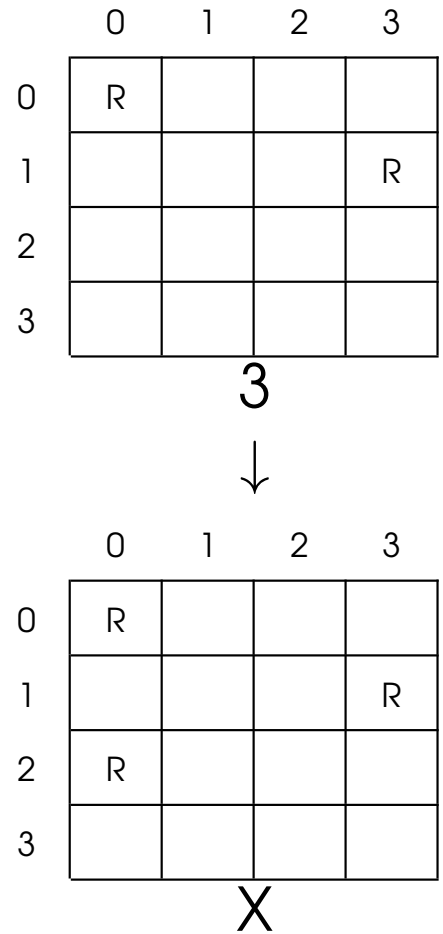
Backtracking

- Algoritmo de busca com backtracking
  - Backtracking: rainha 2 é reposicionada em (1, 3)
  - Sem violações de ataque das rainhas



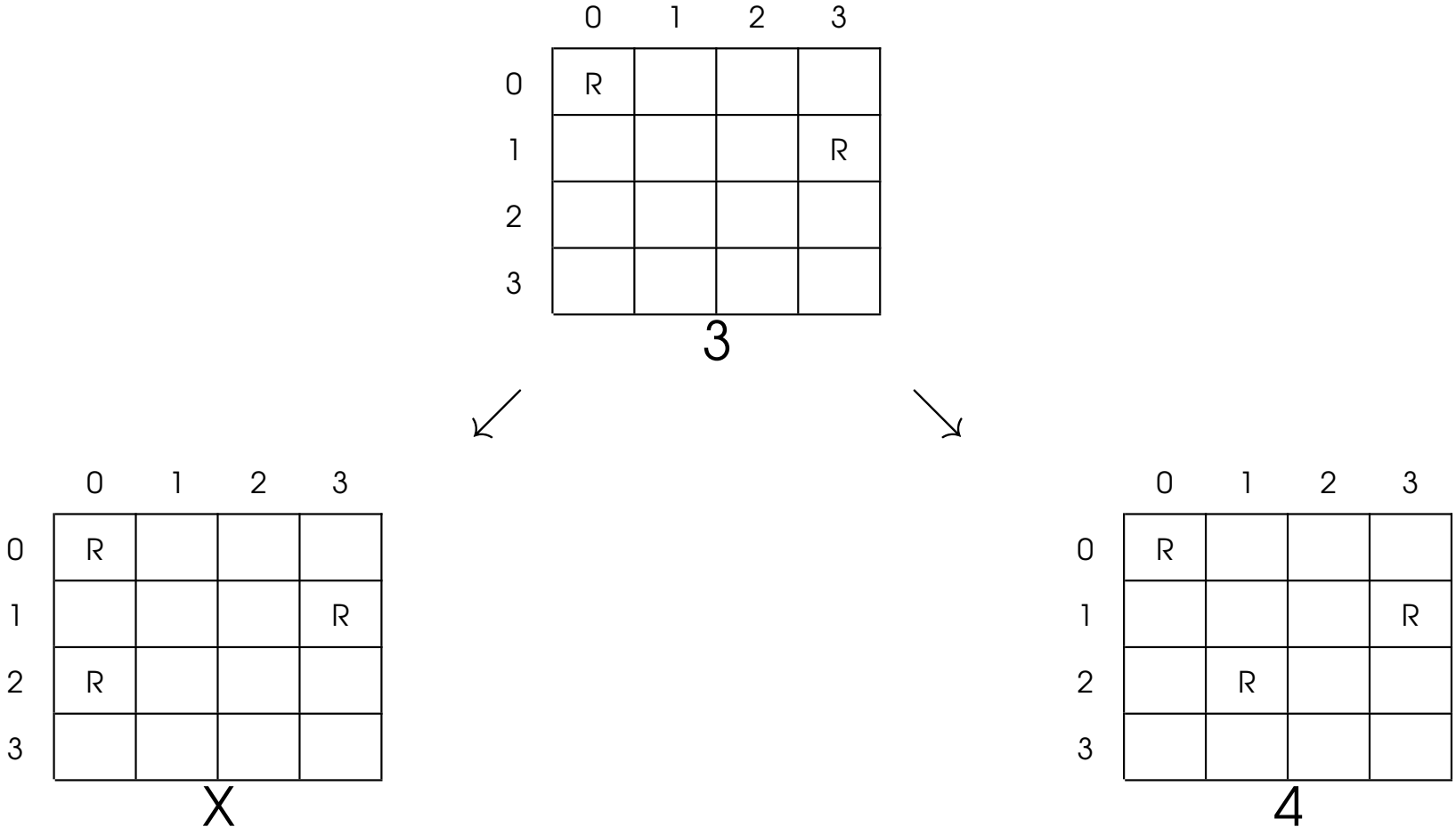
Backtracking

- Algoritmo de busca com backtracking
  - A rainha 3 é posicionada em (2, 0)
  - Conflito: rainhas 1 e 3 em linha de ataque



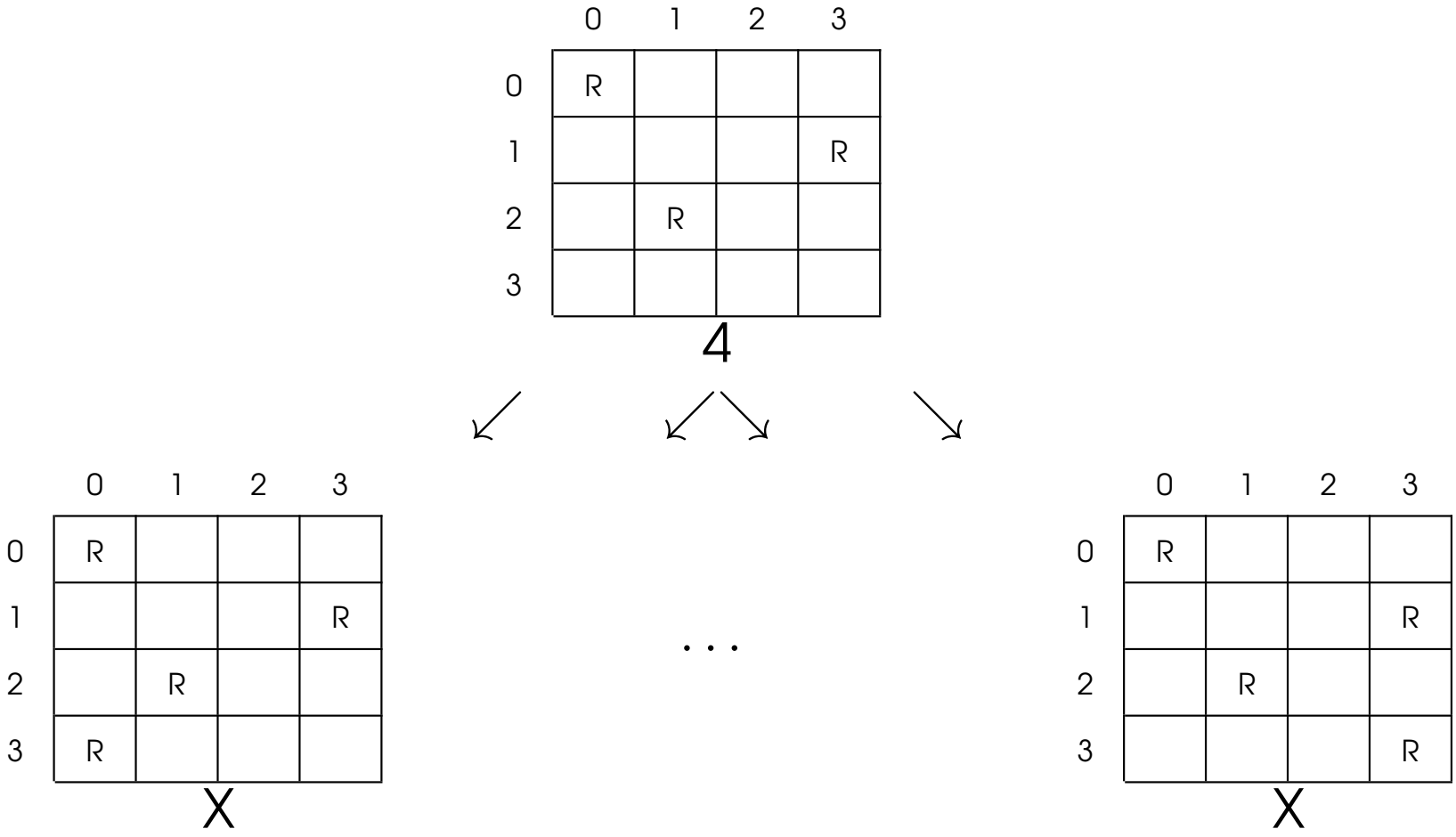
Backtracking

- Algoritmo de busca com backtracking
  - Solução 4: a rainha 3 é posicionada em (2, 1)
  - Sem violações de ataque das rainhas



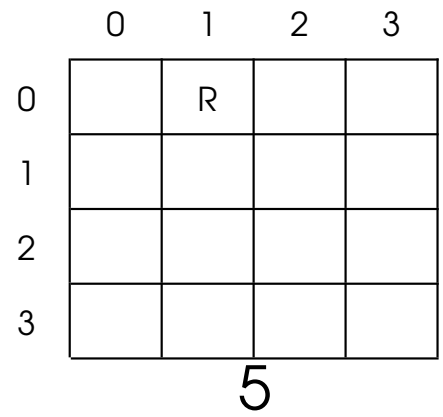
Backtracking

- Algoritmo de busca com backtracking
  - Todos os posicionamentos da rainha 4 geram conflitos de ataque com outras rainhas já posicionadas



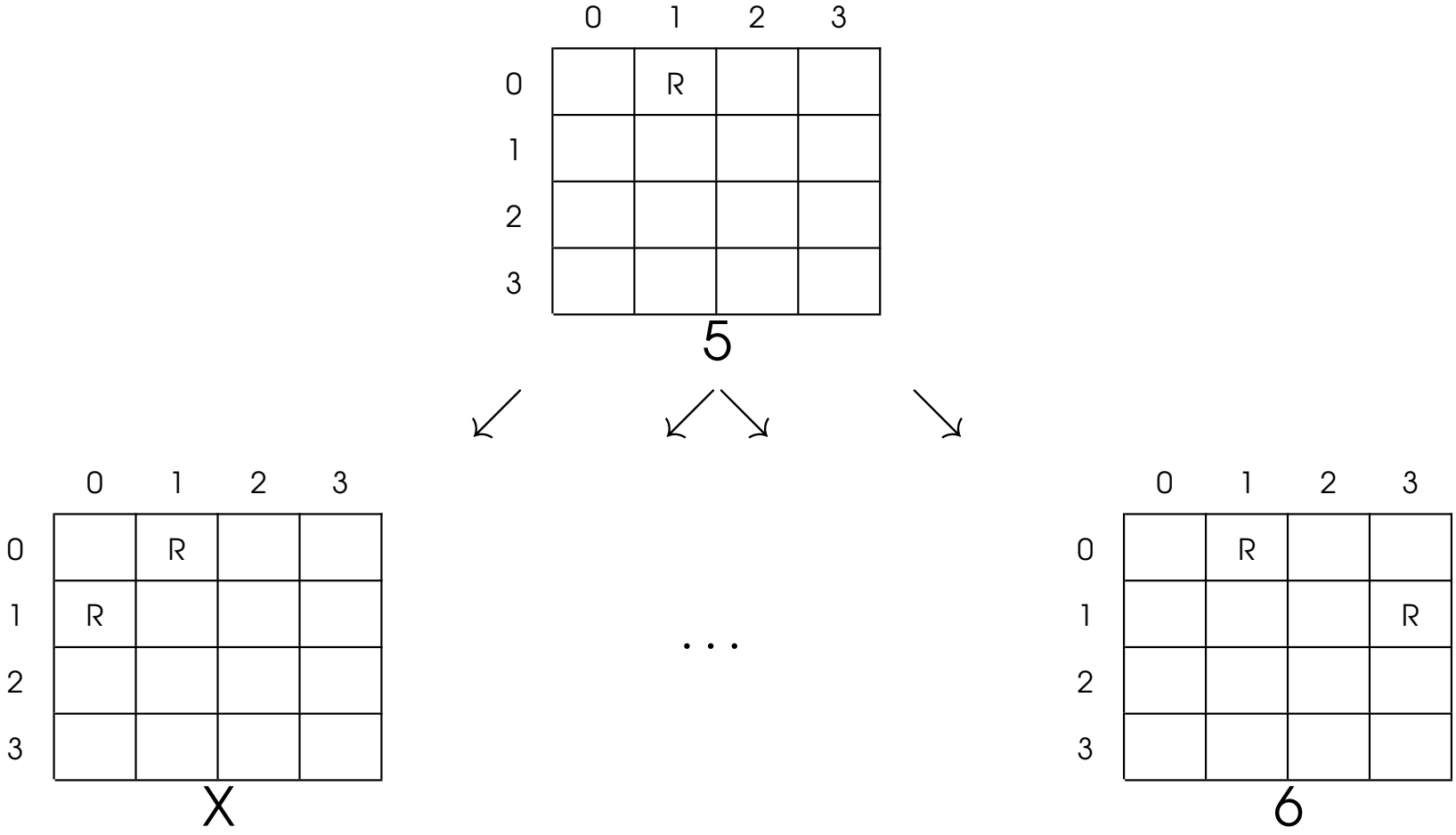
Backtracking

- Algoritmo de busca com backtracking
  - Backtracking: rainha 1 é reposicionada em (0, 1)
  - Sem violações de ataque das rainhas



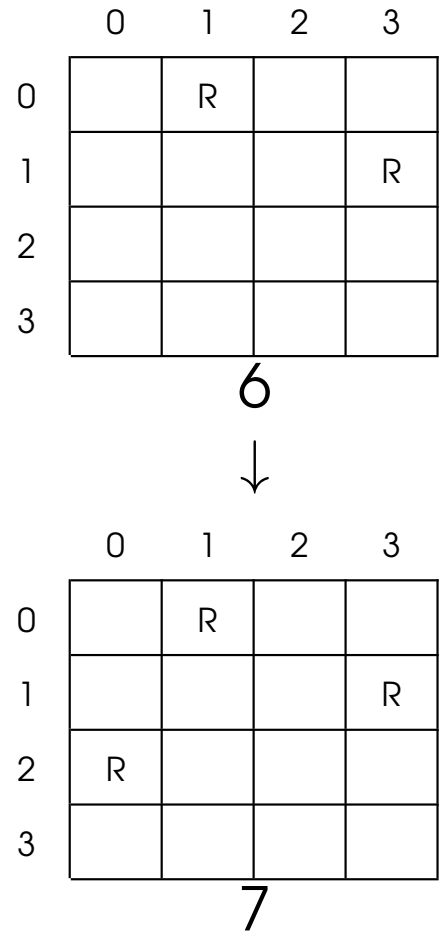
Backtracking

- Algoritmo de busca com backtracking
  - Solução 6: a rainha 2 é posicionada em (1, 3)
  - Sem violações de ataque das rainhas



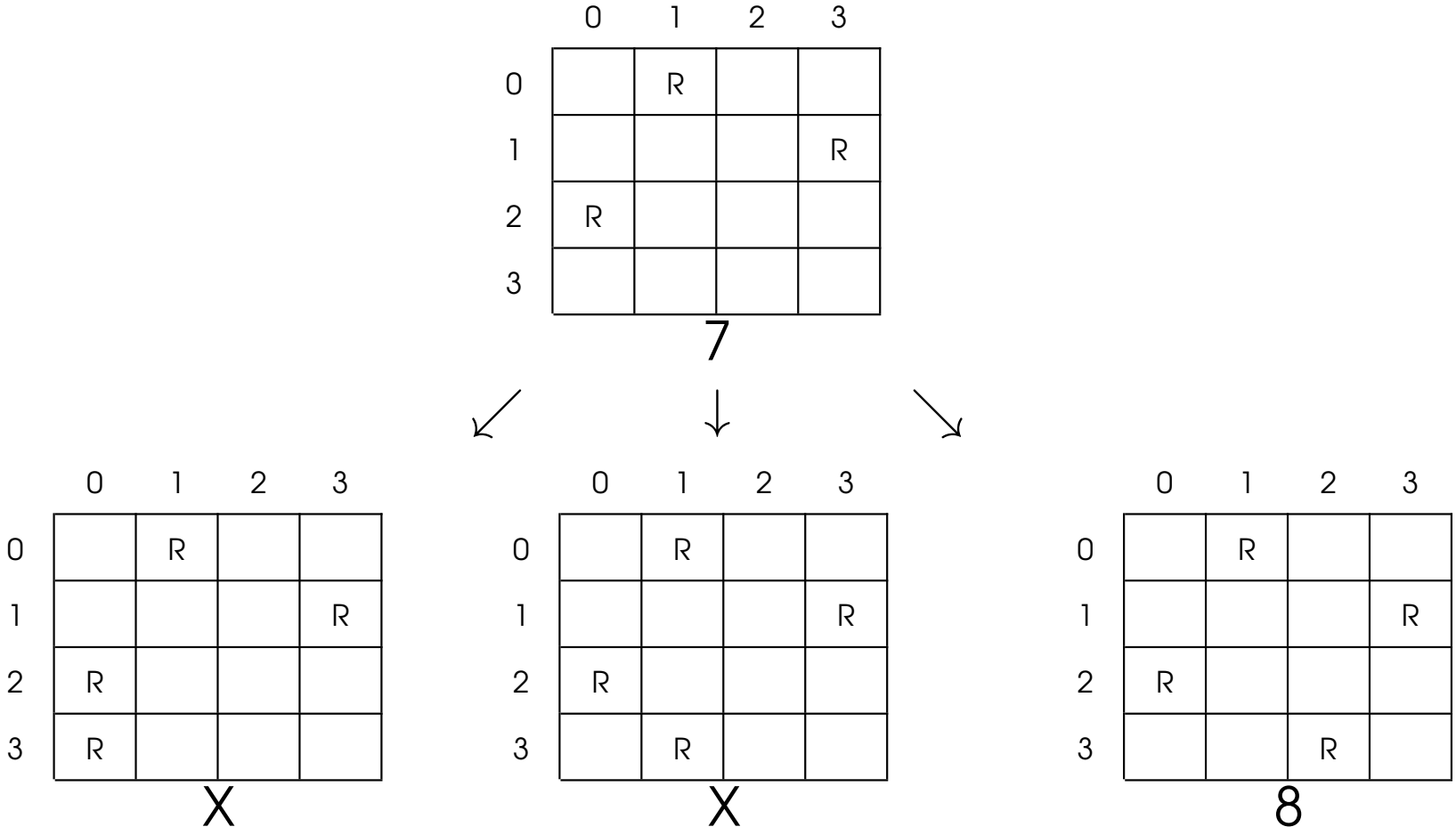
Backtracking

- Algoritmo de busca com backtracking
  - Solução 7: a rainha 3 é posicionada em (2, 0)
  - Sem violações de ataque das rainhas



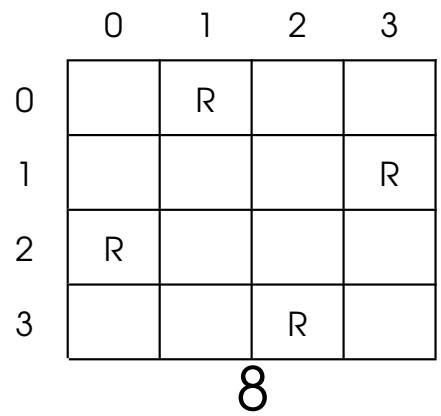
Backtracking

- Algoritmo de busca com backtracking
  - Solução 8: a rainha 4 é posicionada em (3, 2)
  - Sem violações de ataque das rainhas



Backtracking

- Solução para o problema das 4-rainhas
  - Em uma busca exaustiva, seriam necessárias que $n^n = 4^4 = 256$ soluções fossem geradas e avaliadas para verificar as restrições de ataque das rainhas, considerando exatamente uma rainha por linha
  - Utilizando backtracking foram exploradas 8 diferentes soluções, representando cerca de 3% do espaço total de combinações possíveis



Branch-and-bound

- Princípios de funcionamento
  - Esta técnica procura deduzir quais caminhos do espaço de solução não irão conduzir o algoritmo para encontrar uma solução viável
  - Uma solução viável atende todas as restrições do problema, mas não é necessariamente a melhor solução ou solução ótima

Definição de limitantes
inferiores ou superiores
+
Armazenamento do valor
da melhor solução obtida

Branch-and-bound

- Problema de alocação de n pessoas para realizar n trabalhos, onde cada pessoa é paga para realizar um único tipo de trabalho
  - A matriz armazena o quanto cada pessoa recebe para realizar um determinado trabalho
  - Nas linhas da matriz estão representadas as pessoas a, b, c e d e nas colunas os trabalhos 1, 2, 3 e 4

$$Custo = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

O objetivo é minimizar o custo dos trabalhos

Branch-and-bound

- Problema de alocar n pessoas para n trabalhos
  - Como parte da estratégia de minimização, o menor custo de cada linha é selecionado para obtenção do limite inferior do problema
  - No cálculo deste limitante inferior é permitido que uma mesma pessoa realize dois trabalhos, uma vez que o seu propósito é estabelecer um limite e não encontrar uma solução

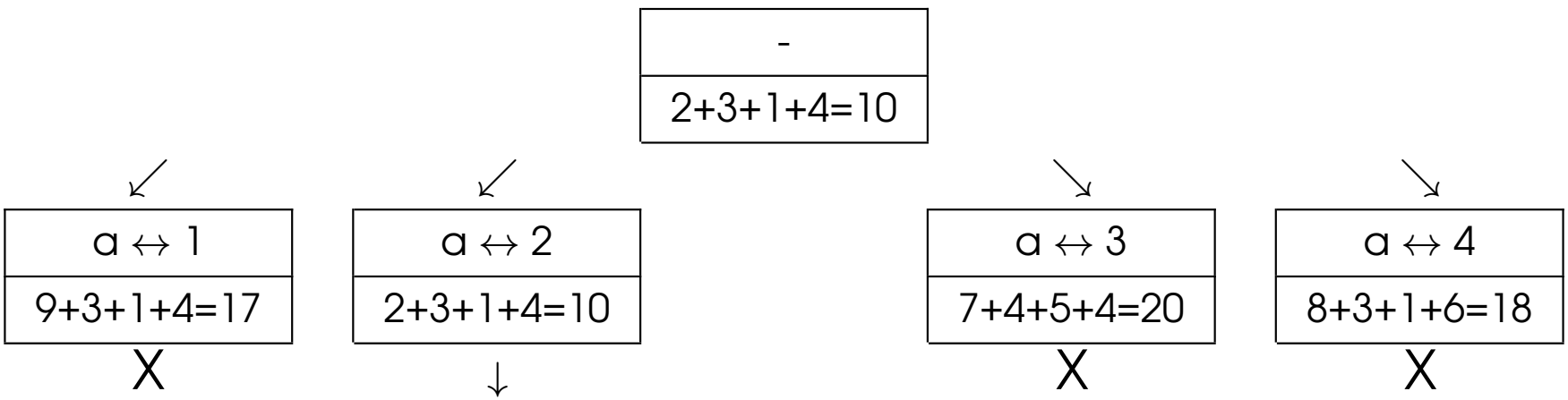
$$Custo = \begin{bmatrix} 9 & \underline{2} & 7 & 8 \\ 6 & 4 & \underline{3} & 7 \\ 5 & 8 & \underline{1} & 8 \\ 7 & 6 & 9 & \underline{4} \end{bmatrix}$$

O limite inferior é $2 + 3 + 1 + 4 = 10$

Branch-and-bound

- Problema de alocar *n* pessoas para *n* trabalhos
  - A primeira solução é gerada alocando a pessoa *a*
  - São geradas soluções para os diferentes trabalhos

$$Custo = \begin{bmatrix} 9 & \underline{2} & 7 & 8 \\ 6 & 4 & \underline{3} & 7 \\ 5 & 8 & \underline{1} & 8 \\ 7 & 6 & 9 & \underline{4} \end{bmatrix}$$

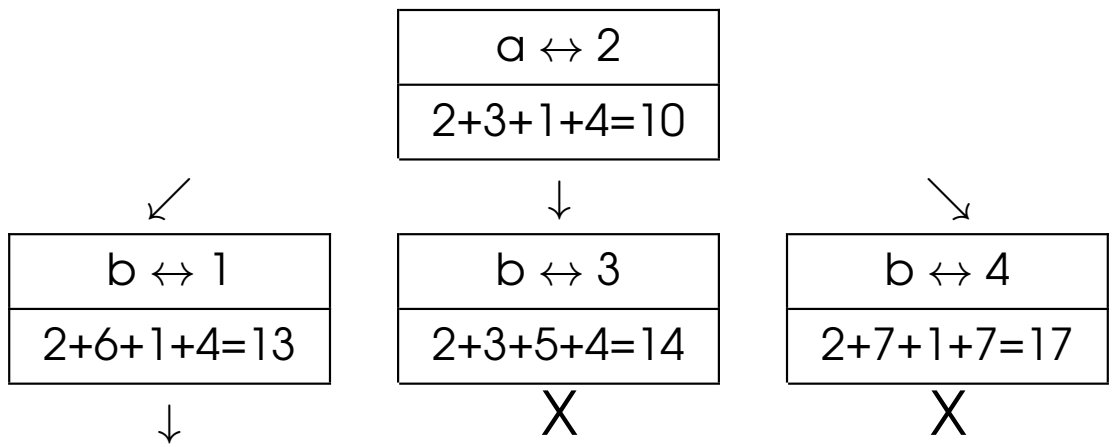


A solução mais promissora possui o menor custo e aloca a pessoa *a* para o trabalho 2

Branch-and-bound

- Problema de alocar *n* pessoas para *n* trabalhos
  - A pessoa *b* é escolhida para próxima alocação
  - São geradas soluções para os diferentes trabalhos

$$Custo = \begin{bmatrix} 9 & \underline{2} & 7 & 8 \\ 6 & 4 & \underline{3} & 7 \\ 5 & 8 & \underline{1} & 8 \\ 7 & 6 & 9 & \underline{4} \end{bmatrix}$$

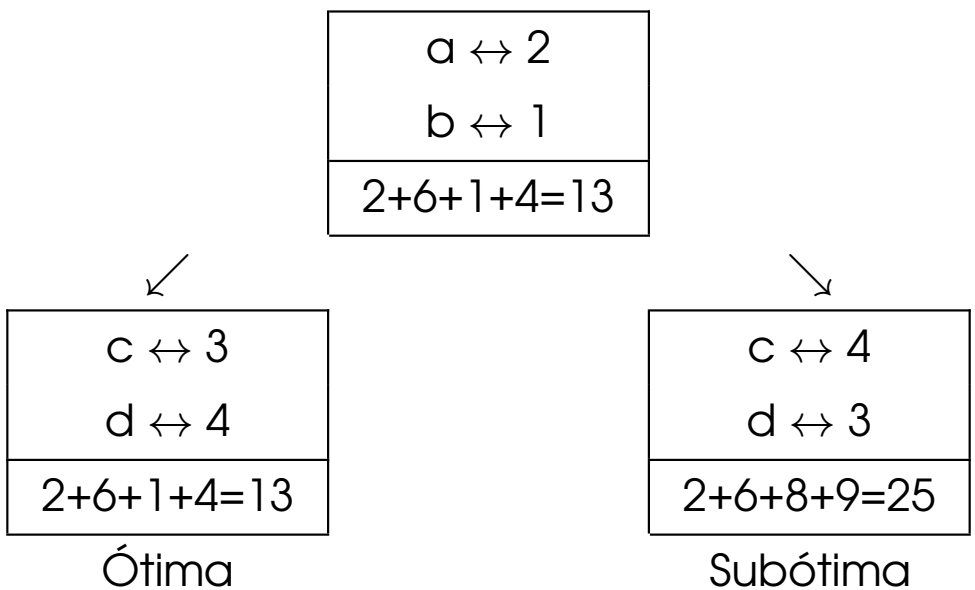


A solução mais promissora possui o menor custo e aloca a pessoa *b* para o trabalho 1

Branch-and-bound

- Problema de alocar n pessoas para n trabalhos
  - A pessoa c é escolhida para próxima alocação
  - São geradas soluções para os diferentes trabalhos

$$Custo = \begin{bmatrix} 9 & \underline{2} & 7 & 8 \\ \underline{6} & 4 & 3 & 7 \\ 5 & 8 & \underline{1} & 8 \\ 7 & 6 & 9 & \underline{4} \end{bmatrix}$$



Branch-and-bound

- Solução para o problema de alocação de pessoas
  - Na busca exaustiva e em um cenário de pior caso, seriam geradas $n! = 4! = 24$ soluções possíveis
  - Aplicando as técnicas de branch-and-bound são geradas apenas 2 soluções, que representam cerca de 8% do espaço total de soluções possíveis

Branch-and-bound

- Problema da mochila com branch-and-bound
  - O limitante superior é definido pela utilização de toda a capacidade W que maximiza o valor armazenado
  - É feito o cálculo da relação entre o valor e o peso $\frac{v_i}{w_i}$ de cada item e sua ordenação na tabela

i	1	2	3	4	5
w_i	2	3	5	1	4
v_i	44	36	55	10	32
v_i/w_i	22	12	11	10	8

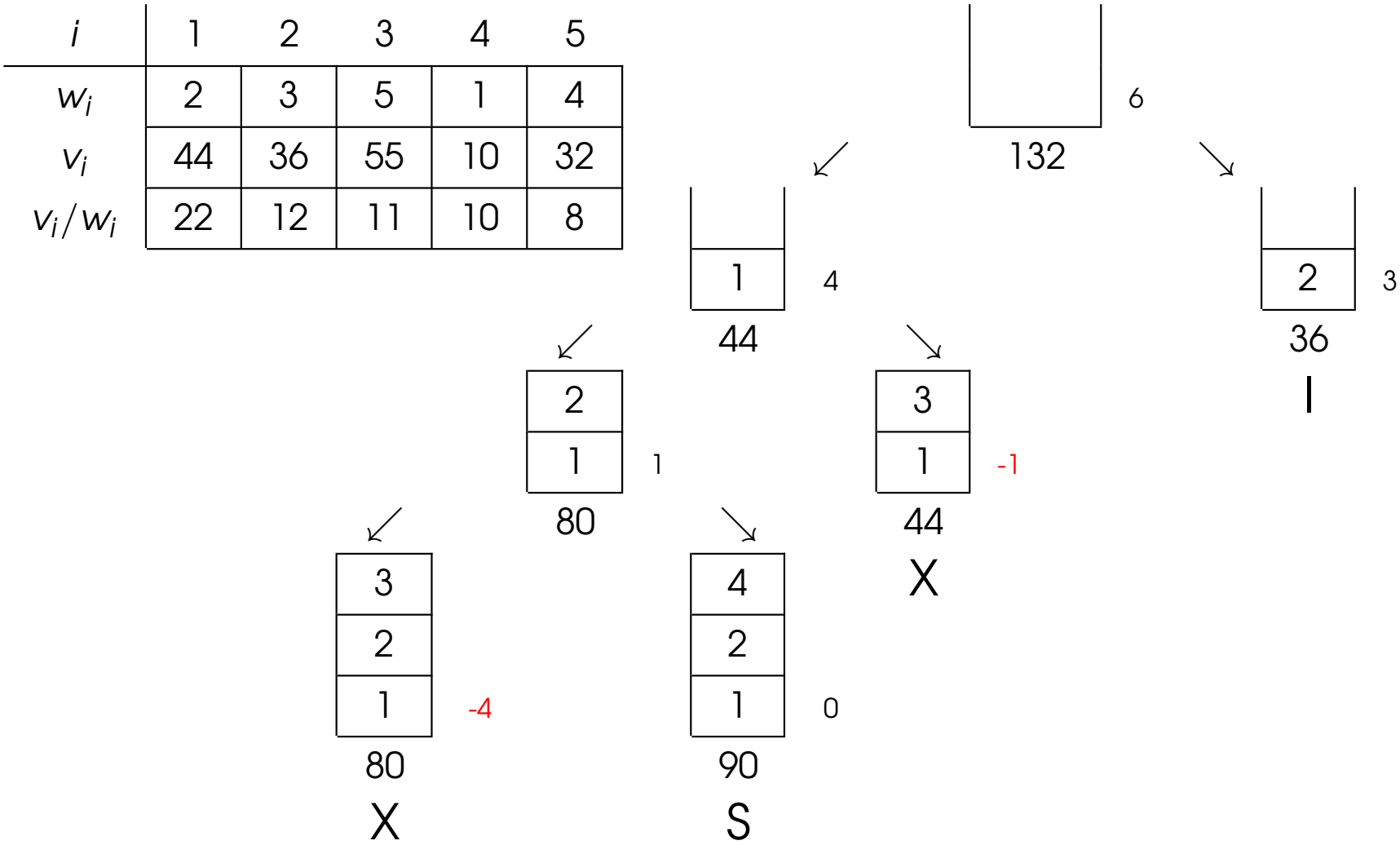
$W = 6$

$$L = max(W \times \frac{v_i}{w_i}) = 132$$

Branch-and-bound

- Problema da mochila com branch-and-bound

i	1	2	3	4	5
w_i	2	3	5	1	4
v_i	44	36	55	10	32
v_i/w_i	22	12	11	10	8



Exercício

- A empresa de tecnologia Poxim Tech está desenvolvendo um robô humanoide que é capaz de se deslocar de forma totalmente autônoma e sem precisar do conhecimento prévio do ambiente físico no qual está localizado
  - Durante o seu deslocamento, que é feito um passo por vez, podem ser realizadas as seguintes operações, listadas em ordem de prioridade
    - Direita (D)
    - Frente (F)
    - Esquerda (E)
    - Trás (T)

Exercício

- A medida que vai explorando o ambiente, o robô cria uma mapa interno para as rotas exploradas
  - Caso uma rota não gere uma solução, outro caminho deve ser escolhido para ser explorado até que a solução seja obtida ou que não existam mais opções
  - Para demonstrar suas habilidades exploratórias, são criados labirintos com exatamente 1 entrada e até 1 saída, com tamanho máximo de 100 por 100 posições
  - É possível que nenhuma rota seja possível para atravessar o labirinto criado, mas quando existe uma saída, é sempre um espaço livre na borda do labirinto que não é o ponto de partida

Exercício

- Formato do arquivo de entrada
  - $\#NL$
  - $[Largura] \times [Altura]$
  - $M_{x,y} = 0 \rightarrow$ *Espaço livre*
  - $M_{x,y} = 1 \rightarrow$ *Parede*
  - $M_{x,y} = X \rightarrow$ *Ponto de partida*

$$\begin{matrix} M_{0,0} & \cdots & M_{0,L-1} \\ \vdots & \ddots & \vdots \\ M_{A-1,0} & \cdots & M_{A-1,L-1} \end{matrix}$$

```
2
5 4
1 1 1 1 1
1 0 0 0 1
1 0 X 0 1
1 1 0 1 1
3 4
1 1 1
1 X 1
1 0 1
1 1 1
```

Exercício

- Formato do arquivo de saída
  - A rota é descrita pela sequência de coordenadas visitadas e operações realizadas
    - Saída do labirinto $\rightarrow$ Espaço livre na extremidade
    - Sem saída $\rightarrow$ Trajeto termina no ponto de partida

```
L0:
INICIO [2,2]
D [2,2]->[2,3]
F [2,3]->[1,3]
E [1,3]->[1,2]
E [1,2]->[1,1]
T [1,1]->[2,1]
BT [1,1]<-[2,1]
BT [1,2]<-[1,1]
BT [1,3]<-[1,2]
BT [2,3]<-[1,3]
BT [2,2]<-[2,3]
T [2,2]->[3,2]
SAIDA [3,2]
L1:
INICIO [1,1]
T [1,1]->[2,1]
BT [1,1]<-[2,1]
SEM SAIDA
```