MC-202
Backtracking

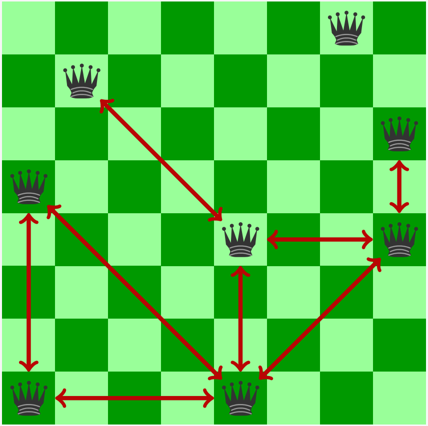
Lehilton Pedrosa
lehilton@ic.unicamp.br

Universidade Estadual de Campinas

Segundo semestre de 2024

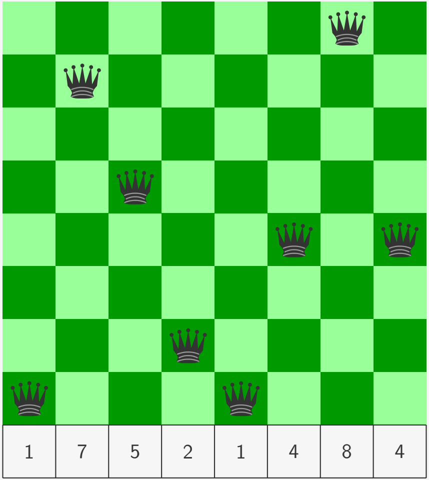
Um problema

Como dispor oito damas em um tabuleiro de xadrez, sem posições de ameaça?



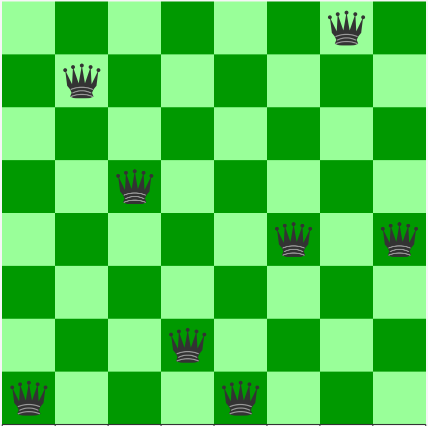
Testando (quase) todas as soluções

- Cada coluna dever ter exatamente uma dama



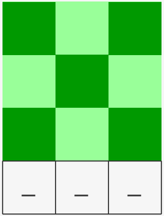
Testando (quase) todas as soluções

- Cada coluna dever ter exatamente uma dama
- Representamos uma disposição com um vetor



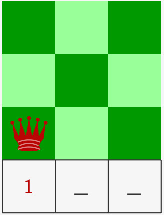
1	7	5	2	1	4	8	4

Enumerando



Enumerando disposições de tamanho 3:

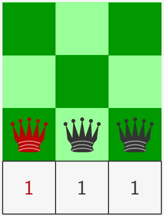
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição

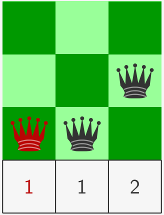
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

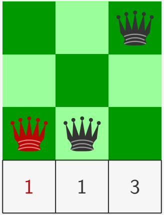
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

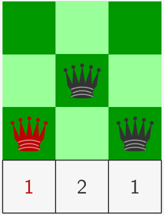
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

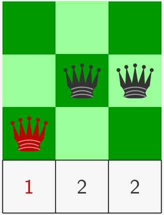
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

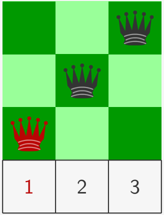
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

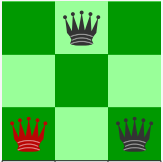
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

Enumerando

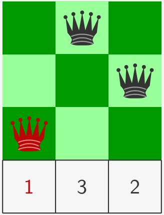


1	3	1

Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

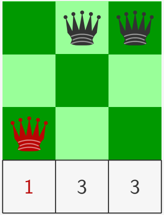
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

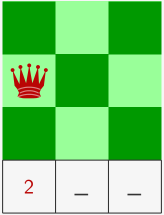
Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2

Enumerando



Enumerando disposições de tamanho 3:

1. fixamos a primeira posição
2. listamos todas os sufixos de tamanho 2
3. repetimos para as outras possibilidades

Enumerando em geral

Como imprimir todas as disposições com prefixo dado?

Enumerando em geral

Como imprimir todas as disposições com prefixo dado?

0			$m-1$				
1	3	3	1				

Enumerando em geral

Como imprimir todas as disposições com prefixo dado?

0			$m-1$				
1	3	3	1				

Vamos escrever uma função que receba um vetor:

Enumerando em geral

Como imprimir todas as disposições com prefixo dado?

0			$m-1$				
1	3	3	1				

Vamos escrever uma função que receba um vetor:

1. com valores fixos até uma posição $m-1$

Enumerando em geral

Como imprimir todas as disposições com prefixo dado?

0			$m-1$				
1	3	3	1	_	_	_	_

Vamos escrever uma função que receba um vetor:

1. com valores fixos até uma posição $m-1$
2. com posições abertas de m até $n-1$

Programando

Como imprimir todas disposições (recursivamente)

```
void enumerar(int vetor[], int m, int n) {
```

Programando

Como imprimir todas disposições (recursivamente)

```
void enumerar(int vetor[], int m, int n) {
  // se todas posições estão fixas, só há uma combinação
  if (n == m) {
    imprimir_vetor(vetor, n);
    return ;
  }

```

Programando

Como imprimir todas disposições (recursivamente)

```
void enumerar(int vetor[], int m, int n) {
  // se todas posições estão fixas, só há uma combinação
  if (n == m) {
    imprimir_vetor(vetor, n);
    return ;
  }

  // senão, estendemos o prefixo em uma posição
  for (int i = 1; i <= n; i++) {
    vetor[m] = i
    enumerar(vetor, m + 1, n);
  }
}
```

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido
  - devolve `1` se nenhuma dama ataca outra

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido
  - devolve `1` se nenhuma dama ataca outra
  - devolve `0` caso contrário

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido
  - devolve `1` se nenhuma dama ataca outra
  - devolve `0` caso contrário

- `existe_solucao` recebe um vetor parcialmente preenchido com m posições fixas

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido
  - devolve `1` se nenhuma dama ataca outra
  - devolve `0` caso contrário

- `existe_solucao` recebe um vetor parcialmente preenchido com m posições fixas
  - devolve `1` se existe alguma disposição válida que mantém a posição das m primeiras damas

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido
  - devolve `1` se nenhuma dama ataca outra
  - devolve `0` caso contrário

- `existe_solucao` recebe um vetor parcialmente preenchido com m posições fixas
  - devolve `1` se existe alguma disposição válida que mantém a posição das m primeiras damas
  - devolve `0` caso contrário

Encontrando uma solução

Como verificar se existe uma disposição válida das damas?

- `disposicao_valida` recebe um vetor preenchido
  - devolve `1` se nenhuma dama ataca outra
  - devolve `0` caso contrário

- `existe_solucao` recebe um vetor parcialmente preenchido com m posições fixas
  - devolve `1` se existe alguma disposição válida que mantém a posição das m primeiras damas
  - devolve `0` caso contrário

Exercício: implemente `disposicao_valida`

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
int existe_solucao(int vetor[], int m, int n) {
```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
int existe_solucao(int vetor[], int m, int n) {
  if (n == m) {
    if (disposicao_valida(vetor, n)) {
      imprimir_vetor(vetor, n);
      return 1;
    } else {
      return 0;
    }
  }

```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

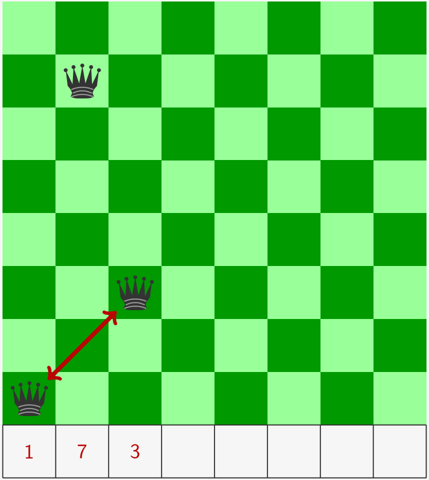
```c
int existe_solucao(int vetor[], int m, int n) {
  if (n == m) {
    if (disposicao_valida(vetor, n)) {
      imprimir_vetor(vetor, n);
      return 1;
    } else {
      return 0;
    }
  }

  for (int i = 1; i <= n; i++) {
    vetor[m] = i;
    if (existe_solucao(vetor, m + 1, n))
      return 1;
  }

```

Enumerando e testando

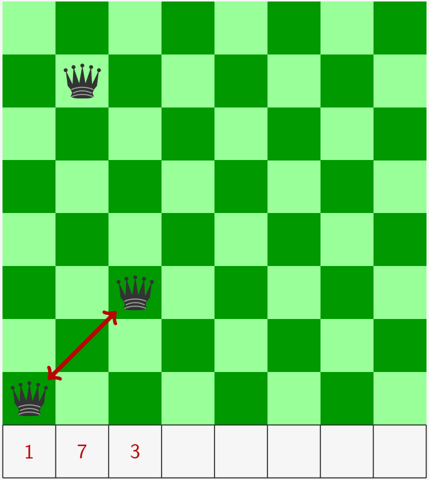
Existe solução com as primeiras damas já dispostas?

```c
int existe_solucao(int vetor[], int m, int n) {
  if (n == m) {
    if (disposicao_valida(vetor, n)) {
      imprimir_vetor(vetor, n);
      return 1;
    } else {
      return 0;
    }
  }

  for (int i = 1; i <= n; i++) {
    vetor[m] = i;
    if (existe_solucao(vetor, m + 1, n))
      return 1;
  }

  return 0;
}
```

Melhorando um pouco

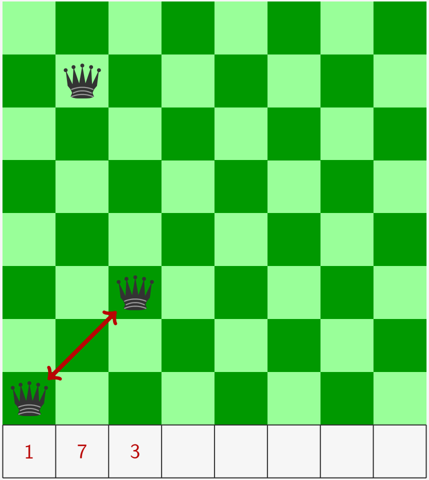


Melhorando um pouco



- alguns prefixos não são viáveis

Melhorando um pouco



- alguns prefixos não são viáveis
- não precisamos testar disposições com esses prefixos

Prefixo viável

Suponha que

Prefixo viável

Suponha que

- `prefixo_viavel` recebe um vetor parcialmente preenchido

Prefixo viável

Suponha que

- `prefixo_viavel` recebe um vetor parcialmente preenchido
  - devolve `1` se nenhuma dama do prefixo ataca outra

Prefixo viável

Suponha que

- `prefixo_viavel` recebe um vetor parcialmente preenchido
  - devolve `1` se nenhuma dama do prefixo ataca outra
  - devolve `0` caso contrário

Ignorando prefixos inviáveis

Se um prefixo é inviável, não continuamos a recursão!

```c
int existe_solucao(int vetor[], int m, int n) {
```

Ignorando prefixos inviáveis

Se um prefixo é inviável, não continuamos a recursão!

```
int existe_solucao(int vetor[], int m, int n) {
  if (!prefixo_viavel(vetor, m))
      return 0;

```

Ignorando prefixos inviáveis

Se um prefixo é inviável, não continuamos a recursão!

```
int existe_solucao(int vetor[], int m, int n) {
  if (!prefixo_viavel(vetor, m))
      return 0;

  if (n == m)
      return 1;

```

Ignorando prefixos inviáveis

Se um prefixo é inviável, não continuamos a recursão!

```
int existe_solucao(int vetor[], int m, int n) {
  if (!prefixo_viavel(vetor, m))
      return 0;

  if (n == m)
      return 1;

  for (int i = 1; i <= n; i++) {
    vetor[m] = i;
    if (existe_solucao(vetor, m + 1, n))
      return 1;
  }

```

Ignorando prefixos inviáveis

Se um prefixo é inviável, não continuamos a recursão!

```c
int existe_solucao(int vetor[], int m, int n) {
  if (!prefixo_viavel(vetor, m))
      return 0;

  if (n == m)
      return 1;

  for (int i = 1; i <= n; i++) {
    vetor[m] = i;
    if (existe_solucao(vetor, m + 1, n))
      return 1;
  }

  return 0;
}
```

Testando prefixo

```
int prefixo_viavel(int vetor[], int m) {
```

Testando prefixo

```
int prefixo_viavel(int vetor[], int m) {
  // para cada coluna do prefixo
  for (int i = 0; i < m - 1; i++) {
```

Testando prefixo

```c
int prefixo_viavel(int vetor[], int m) {
  // para cada coluna do prefixo
  for (int i = 0; i < m - 1; i++) {

    // se está na mesma linha
    if (vetor[i] == vetor[m-1])
      return 0;
```

Testando prefixo

```c
int prefixo_viavel(int vetor[], int m) {
  // para cada coluna do prefixo
  for (int i = 0; i < m - 1; i++) {

    // se está na mesma linha
    if (vetor[i] == vetor[m-1])
      return 0;

    // se está na mesma diagonal
    if ((m - 1) - i == abs(vetor[m-1] - vetor[i]))
      return 0;
```

Testando prefixo

```c
int prefixo_viavel(int vetor[], int m) {
  // para cada coluna do prefixo
  for (int i = 0; i < m - 1; i++) {

    // se está na mesma linha
    if (vetor[i] == vetor[m-1])
      return 0;

    // se está na mesma diagonal
    if ((m - 1) - i == abs(vetor[m-1] - vetor[i]))
      return 0;
  }

  return 1;
}
```

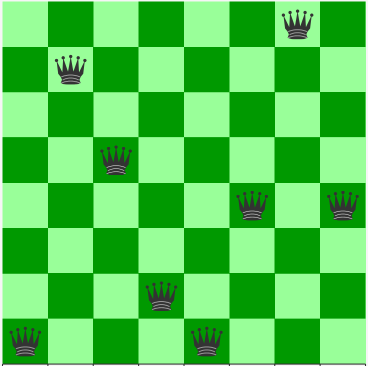
Testando prefixo

```c
int prefixo_viavel(int vetor[], int m) {
  // para cada coluna do prefixo
  for (int i = 0; i < m - 1; i++) {

    // se está na mesma linha
    if (vetor[i] == vetor[m-1])
      return 0;

    // se está na mesma diagonal
    if ((m - 1) - i == abs(vetor[m-1] - vetor[i]))
      return 0;
  }

  return 1;
}
```

Pergunta: por que só precisamos comparar o último elemento do prefixo com os anteriores?

Modificando a estratégia

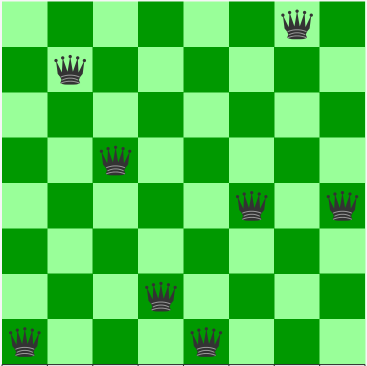
Como diminuir as disposições testadas?



1	7	5	2	1	4	8	4

Modificando a estratégia

Como diminuir as disposições testadas?

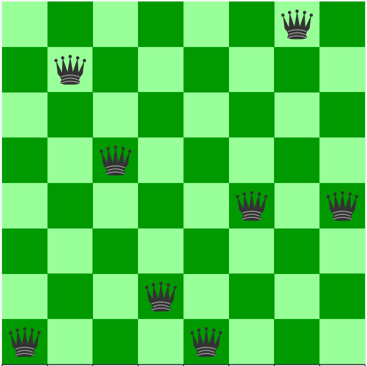


1	7	5	2	1	4	8	4

- cada coluna só deve ter uma dama:

Modificando a estratégia

Como diminuir as disposições testadas?

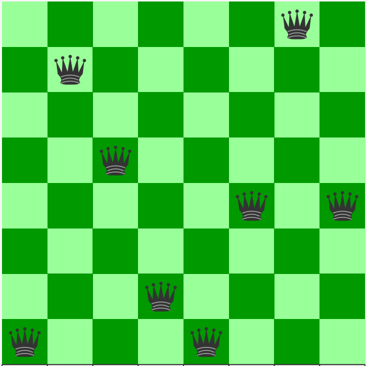


1	7	5	2	1	4	8	4

- cada coluna só deve ter uma dama: ✓

Modificando a estratégia

Como diminuir as disposições testadas?

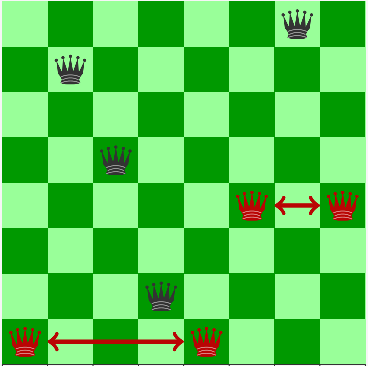


1	7	5	2	1	4	8	4

- cada coluna só deve ter uma dama: ✓
- cada linha só deve ter uma dama:

Modificando a estratégia

Como diminuir as disposições testadas?

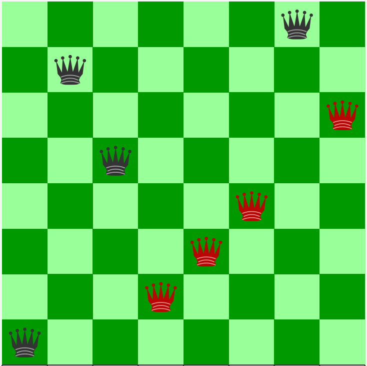


1	7	5	2	1	4	8	4

- cada coluna só deve ter uma dama: ✓
- cada linha só deve ter uma dama: ✗

Modificando a estratégia

Como diminuir as disposições testadas?



1	7	5	2	3	4	8	6

- cada coluna só deve ter uma dama: ✓
- cada linha só deve ter uma dama: ✗

Observação: uma disposição deve ser uma permutação

Permutações

Vamos escreve uma função `permutacoes` que:

Permutações

1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:

Permutações

1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas
- Imprime todas permutações que esse prefixo:

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas
- Imprime todas permutações que esse prefixo:
  - 1, 2, 3, 4, 5, 6, 7, 8

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas

- Imprime todas permutações que esse prefixo:
  - 1, 2, 3, 4, 5, 6, 7, 8
  - 1, 2, 3, 4, 5, 6, 8, 7

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas

- Imprime todas permutações que esse prefixo:
  - 1, 2, 3, 4, 5, 6, 7, 8
  - 1, 2, 3, 4, 5, 6, 8, 7
  - 1, 2, 3, 4, 5, 7, 6, 8

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas

- Imprime todas permutações que esse prefixo:
  - 1, 2, 3, 4, 5, 6, 7, 8
  - 1, 2, 3, 4, 5, 6, 8, 7
  - 1, 2, 3, 4, 5, 7, 6, 8
  - 1, 2, 3, 4, 5, 7, 8, 6

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas

- Imprime todas permutações que esse prefixo:
  - 1, 2, 3, 4, 5, 6, 7, 8
  - 1, 2, 3, 4, 5, 6, 8, 7
  - 1, 2, 3, 4, 5, 7, 6, 8
  - 1, 2, 3, 4, 5, 7, 8, 6
  - 1, 2, 3, 4, 5, 8, 6, 7

Permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Vamos escreve uma função `permutacoes` que:

- Recebe um vetor com n números distintos:
  - m primeiras posições devem ser mantidas
  - posições de m até $n-1$ devem ser permutadas
- Imprime todas permutações que esse prefixo:
  - 1, 2, 3, 4, 5, 6, 7, 8
  - 1, 2, 3, 4, 5, 6, 8, 7
  - 1, 2, 3, 4, 5, 7, 6, 8
  - 1, 2, 3, 4, 5, 7, 8, 6
  - 1, 2, 3, 4, 5, 8, 6, 7
  - 1, 2, 3, 4, 5, 8, 7, 6

Gerando permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Gerando permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Para cada índice i ainda não fixado:

Gerando permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m

Gerando permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m

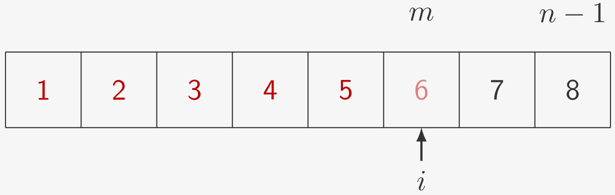
Gerando permutações

					m		$n-1$
1	2	3	4	5	6	7	8

Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

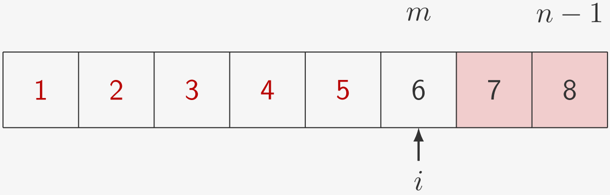
Gerando permutações



Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

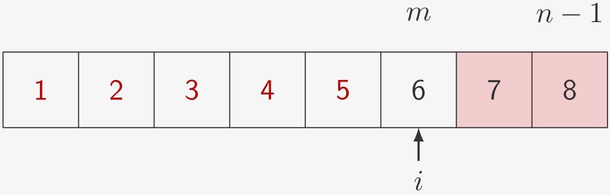
Gerando permutações



Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

Gerando permutações

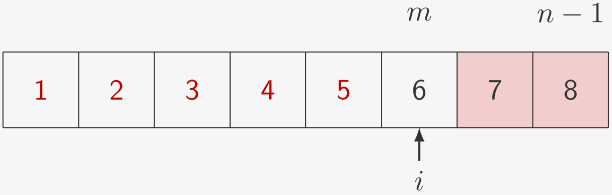


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8

Gerando permutações

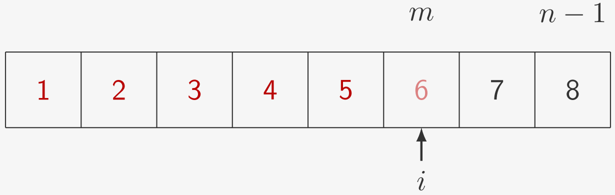


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7

Gerando permutações



Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7

Gerando permutações

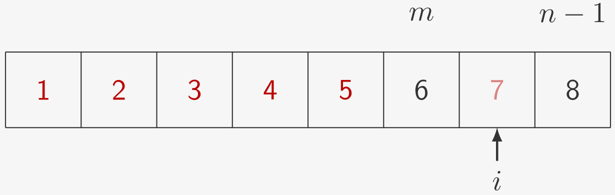
					m		$n-1$
1	2	3	4	5	6	7	8
					$\uparrow$ i		

Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7

Gerando permutações

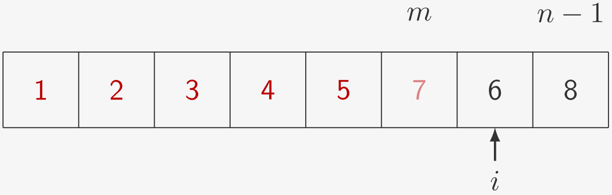


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7

Gerando permutações

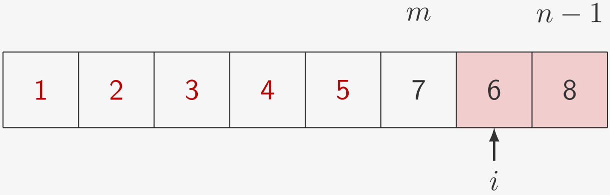


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7

Gerando permutações

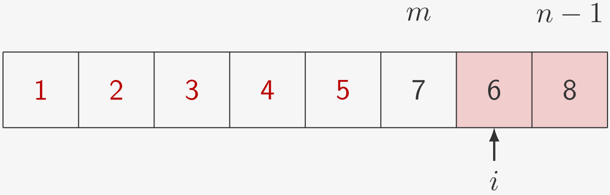


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7

Gerando permutações

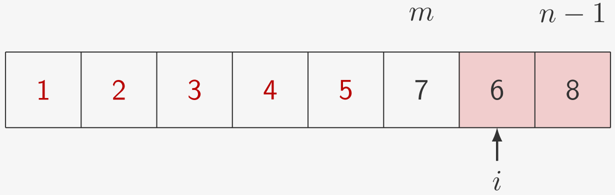


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8

Gerando permutações

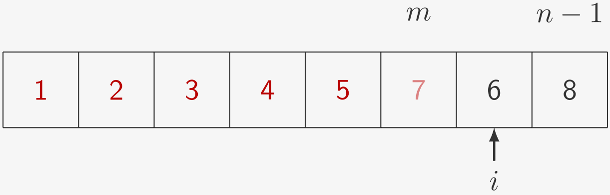


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6

Gerando permutações

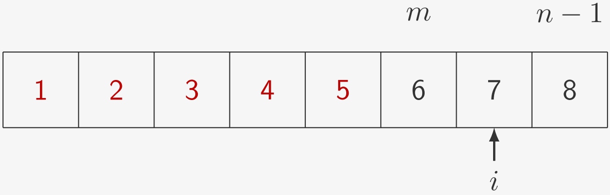


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6

Gerando permutações



Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6

Gerando permutações

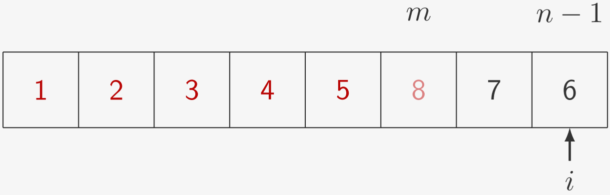
					m		$n-1$
1	2	3	4	5	6	7	8
							$\uparrow$ i

Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6

Gerando permutações

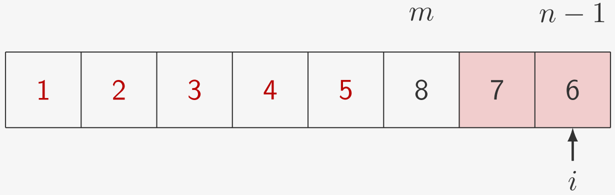


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6

Gerando permutações



Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6

Gerando permutações

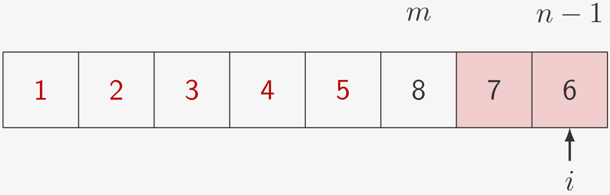
					m		$n-1$
1	2	3	4	5	8	7	6
							i

Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6
- 1, 2, 3, 4, 5, 8, 6, 7

Gerando permutações

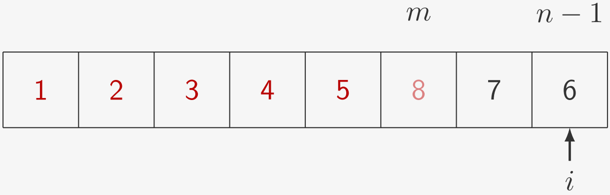


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6
- 1, 2, 3, 4, 5, 8, 6, 7
- 1, 2, 3, 4, 5, 8, 7, 6

Gerando permutações

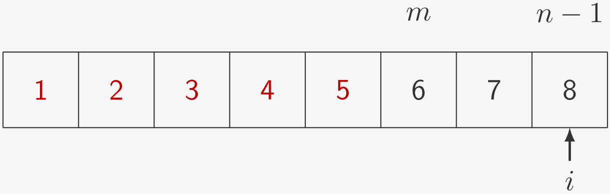


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6
- 1, 2, 3, 4, 5, 8, 6, 7
- 1, 2, 3, 4, 5, 8, 7, 6

Gerando permutações

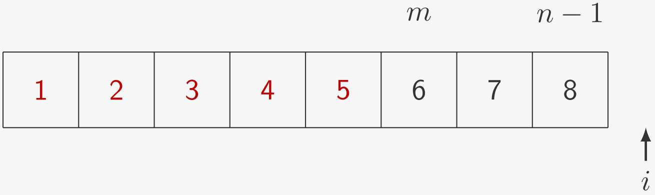


Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6
- 1, 2, 3, 4, 5, 8, 6, 7
- 1, 2, 3, 4, 5, 8, 7, 6

Gerando permutações



Para cada índice i ainda não fixado:

1. trocamos a posição i com a posição m
2. listamos as permutações recursivamente fixando m
3. voltamos às posições originais

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 3, 4, 5, 6, 8, 7
- 1, 2, 3, 4, 5, 7, 6, 8
- 1, 2, 3, 4, 5, 7, 8, 6
- 1, 2, 3, 4, 5, 8, 6, 7
- 1, 2, 3, 4, 5, 8, 7, 6

Programando

Permutando vetores de tamanho n:

Programando

Permutando vetores de tamanho n:

```
void permutacoes(int vetor[], int m, int n) {
```

Programando

Permutando vetores de tamanho n:

```
void permutacoes(int vetor[], int m, int n) {
  // se todo vetor estiver fixo, só há uma permutação
  if (n == m) {
    imprimir_vetor(vetor, n);
    return ;
  }

```

Programando

Permutando vetores de tamanho n:

```
void permutacoes(int vetor[], int m, int n) {
  // se todo vetor estiver fixo, só há uma permutação
  if (n == m) {
    imprimir_vetor(vetor, n);
    return ;
  }

  // senão, então fixa posição m com cada valor livre
  for (int i = m; i < n; i++) {
    troca(&vetor[m], &vetor[i]);
    permutar(vetor, m + 1, n);
    troca(&vetor[m], &vetor[i]);
  }
}
```

Programando

Permutando vetores de tamanho n:

```
void permutacoes(int vetor[], int m, int n) {
  // se todo vetor estiver fixo, só há uma permutação
  if (n == m) {
    imprimir_vetor(vetor, n);
    return ;
  }

  // senão, então fixa posição m com cada valor livre
  for (int i = m; i < n; i++) {
    troca(&vetor[m], &vetor[i]);
    permutar(vetor, m + 1, n);
    troca(&vetor[m], &vetor[i]);
  }
}
```

Exercício: resolver o problema das damas usando permutação

Backtracking

Backtracking ou retrocesso é um algoritmo genérico, com as seguintes propriedades

Backtracking

Backtracking ou retrocesso é um algoritmo genérico, com as seguintes propriedades

- as soluções são construídas incrementalmente

Backtracking

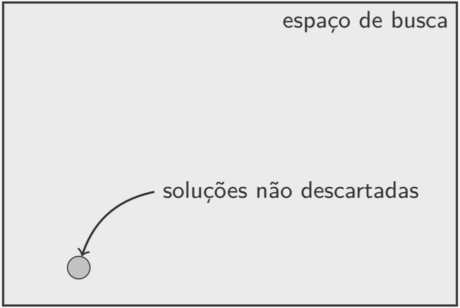
Backtracking ou retrocesso é um algoritmo genérico, com as seguintes propriedades

- as soluções são construídas incrementalmente
- uma solução parcial é descartada tão logo ela se mostre inviável

Backtracking

Backtracking ou retrocesso é um algoritmo genérico, com as seguintes propriedades

- as soluções são construídas incrementalmente
- uma solução parcial é descartada tão logo ela se mostre inviável



Eficiência do Backtracking

- Mais rápido que força Bruta pois eliminamos vários candidatos de uma só vez

Eficiência do Backtracking

- Mais rápido que força Bruta pois eliminamos vários candidatos de uma só vez
- Implementação simples, mas pode ser lento para problemas com muitas soluções parciais

Eficiência do Backtracking

- Mais rápido que força Bruta pois eliminamos vários candidatos de uma só vez
- Implementação simples, mas pode ser lento para problemas com muitas soluções parciais

Como fazer um algoritmo de Backtracking rápido?

Eficiência do Backtracking

- Mais rápido que força Bruta pois eliminamos vários candidatos de uma só vez
- Implementação simples, mas pode ser lento para problemas com muitas soluções parciais

Como fazer um algoritmo de Backtracking rápido?

- O algoritmo para verificar solução parcial deve ser:

Eficiência do Backtracking

- Mais rápido que força Bruta pois eliminamos vários candidatos de uma só vez
- Implementação simples, mas pode ser lento para problemas com muitas soluções parciais

Como fazer um algoritmo de Backtracking rápido?

- O algoritmo para verificar solução parcial deve ser:
  - Bom: evita explorar muitas soluções parciais

Eficiência do Backtracking

- Mais rápido que força Bruta pois eliminamos vários candidatos de uma só vez
- Implementação simples, mas pode ser lento para problemas com muitas soluções parciais

Como fazer um algoritmo de Backtracking rápido?

- O algoritmo para verificar solução parcial deve ser:
  - Bom: evita explorar muitas soluções parciais
  - Rápido: processa cada solução rapidamente

Aplicações para Backtracking

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições
  - Como o Sudoku, por exemplo

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições
  - Como o Sudoku, por exemplo

- Problemas de Otimização Combinatória

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições
  - Como o Sudoku, por exemplo
- Problemas de Otimização Combinatória
  - Conseguimos enumerar as soluções do problema

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições
  - Como o Sudoku, por exemplo

- Problemas de Otimização Combinatória
  - Conseguimos enumerar as soluções do problema
  - Queremos encontrar a de valor mínimo

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições
  - Como o Sudoku, por exemplo
- Problemas de Otimização Combinatória
  - Conseguimos enumerar as soluções do problema
  - Queremos encontrar a de valor mínimo
- Programação Lógica (Prolog, por exemplo)

Aplicações para Backtracking

Para aplicar Backtracking é necessário que o problema tenha um conceito de solução parcial

- Problemas de satisfação de restrições
  - Encontrar uma solução que satisfaça as restrições
  - Como o Sudoku, por exemplo
- Problemas de Otimização Combinatória
  - Conseguimos enumerar as soluções do problema
  - Queremos encontrar a de valor mínimo
- Programação Lógica (Prolog, por exemplo)
  - Prova automática de teoremas

Exercício

Crie um algoritmo que, dado n e C, imprime todas as sequências de números inteiros positivos $x_1, x_2, \ldots, x_n$ tal que

$$x_1 + x_2 + \cdots + x_n = C$$

a) Modifique o seu algoritmo para considerar apenas sequências sem repetições
b) Modifique o seu algoritmo para imprimir apenas sequências com $x_1 \leq x_2 \leq \cdots \leq x_n$