MC-202
Backtracking

Lehilton Pedrosa

Universidade Estadual de Campinas

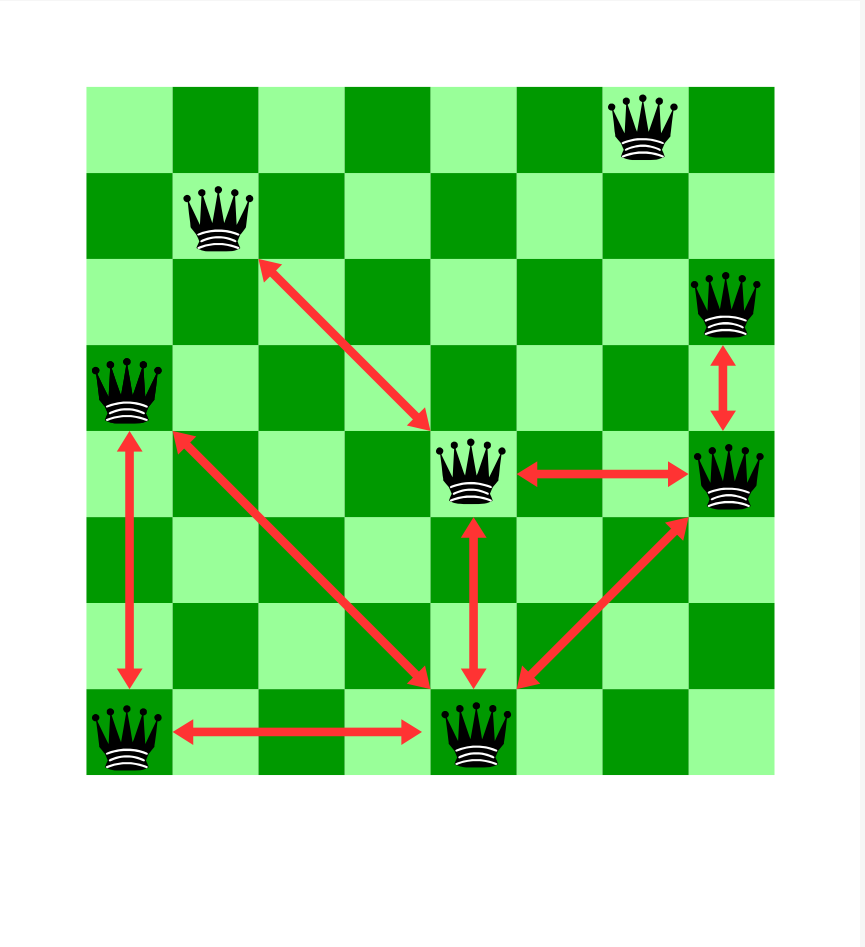
Segundo semestre de 2018

Um problema

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

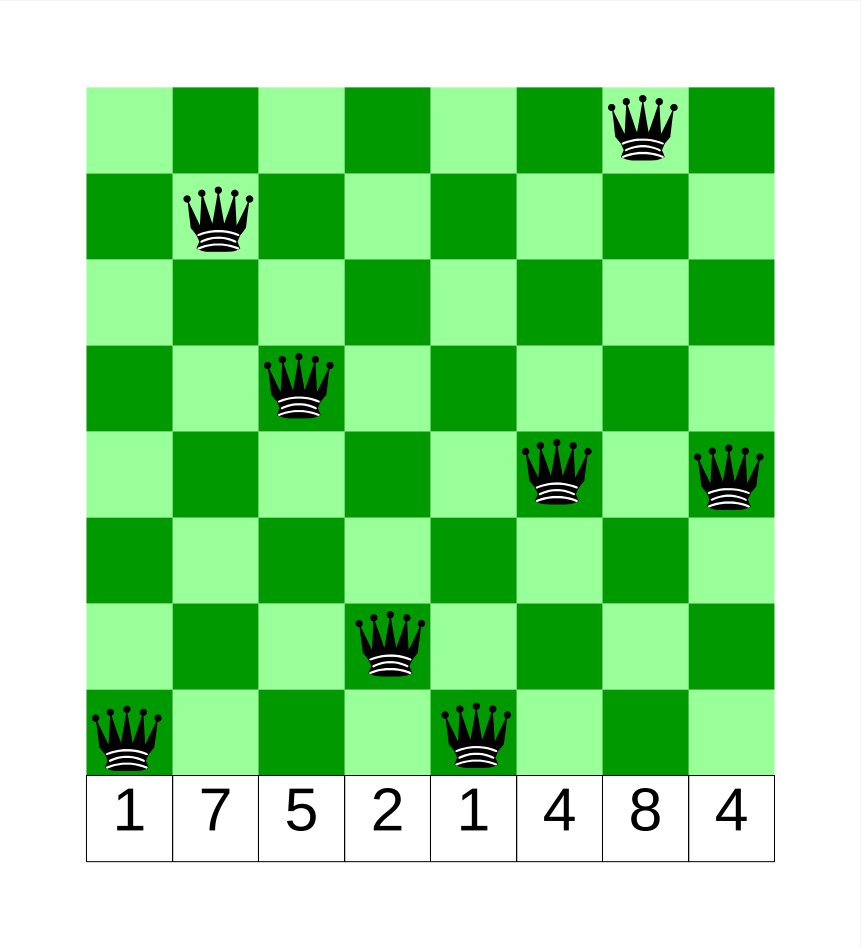
Um problema

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



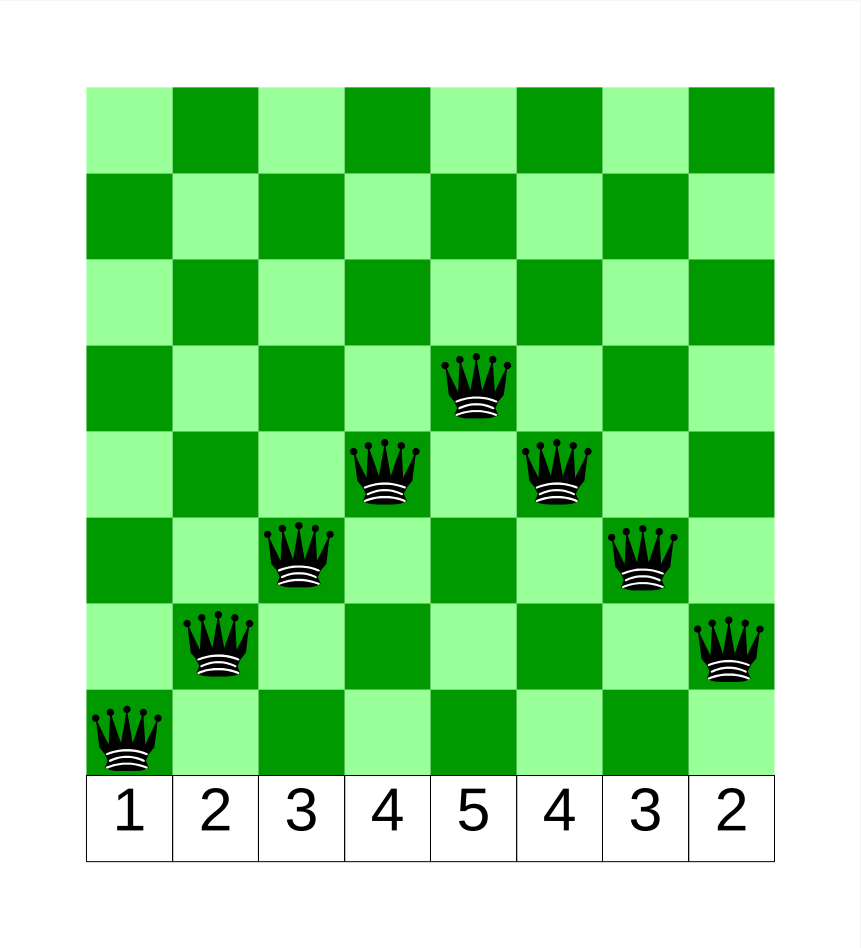
Testando (quase) todas as soluções

- Cada coluna dever ter exatamente uma dama

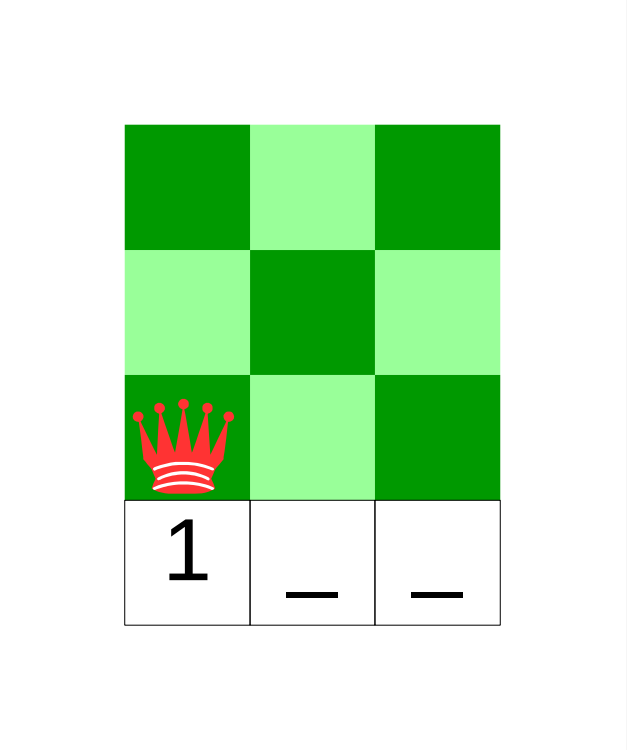


Testando (quase) todas as soluções

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



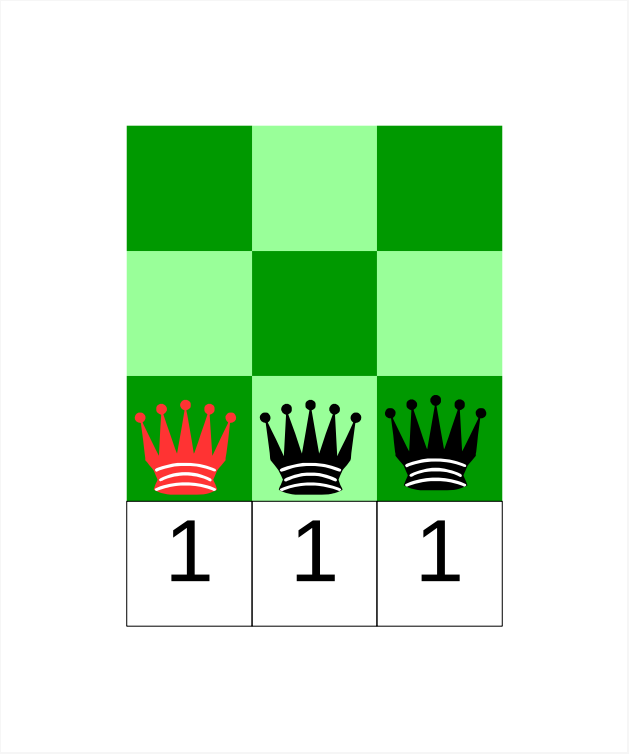
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição

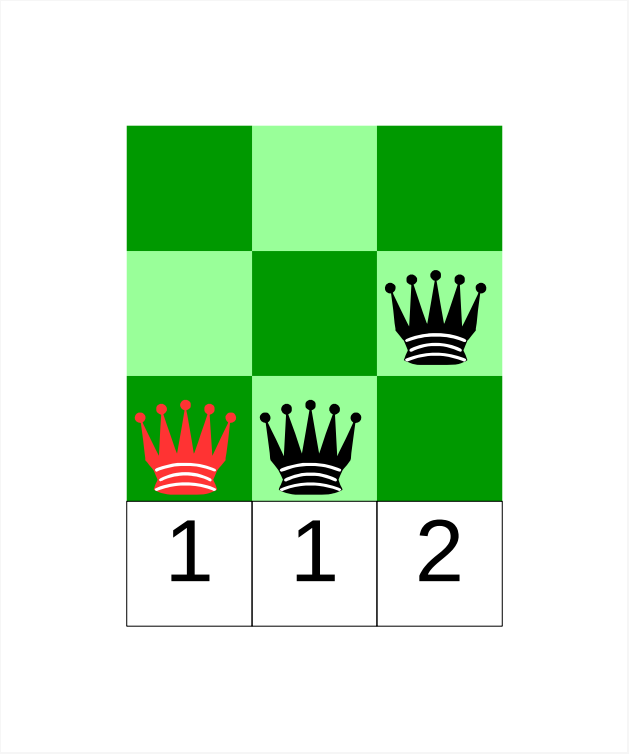
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

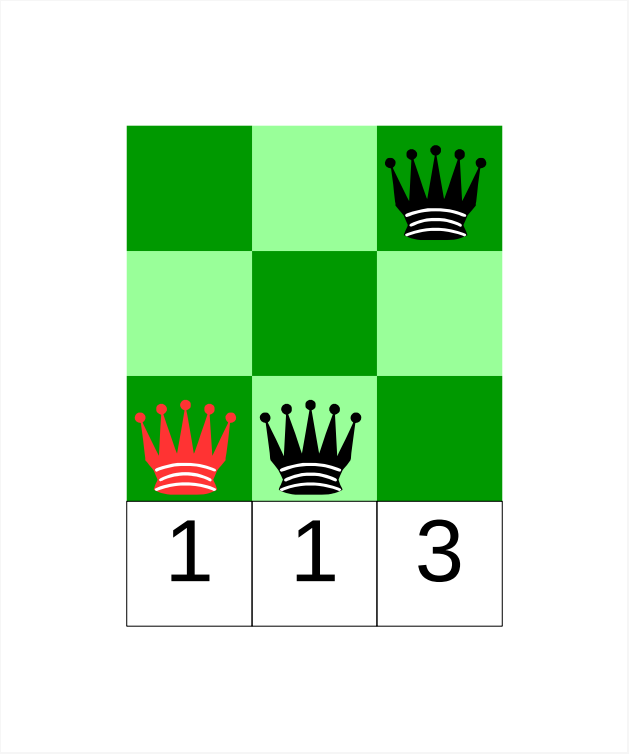
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

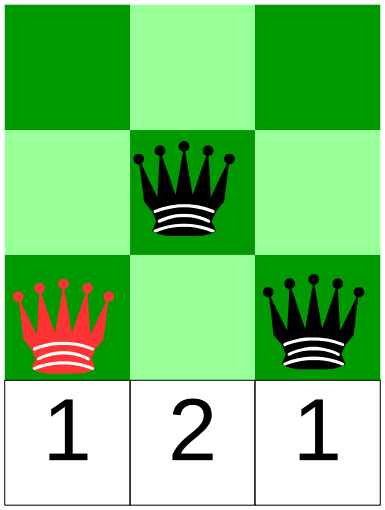
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

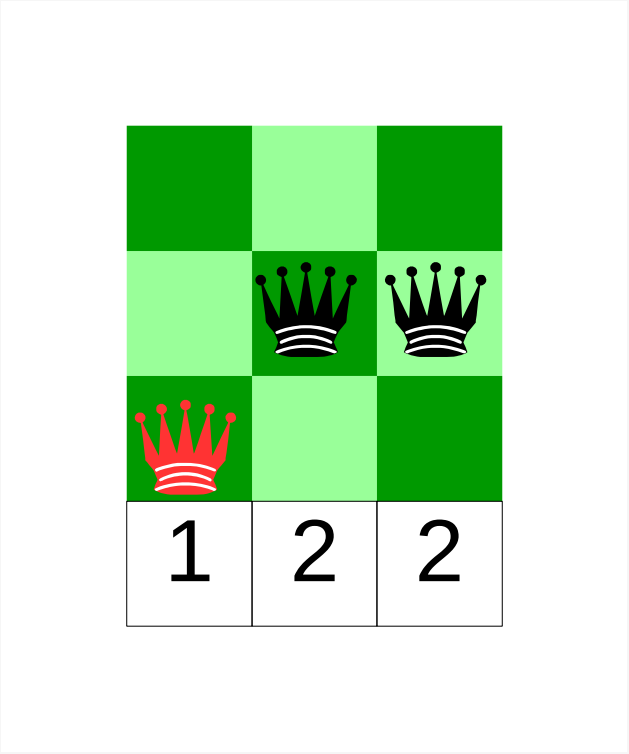
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

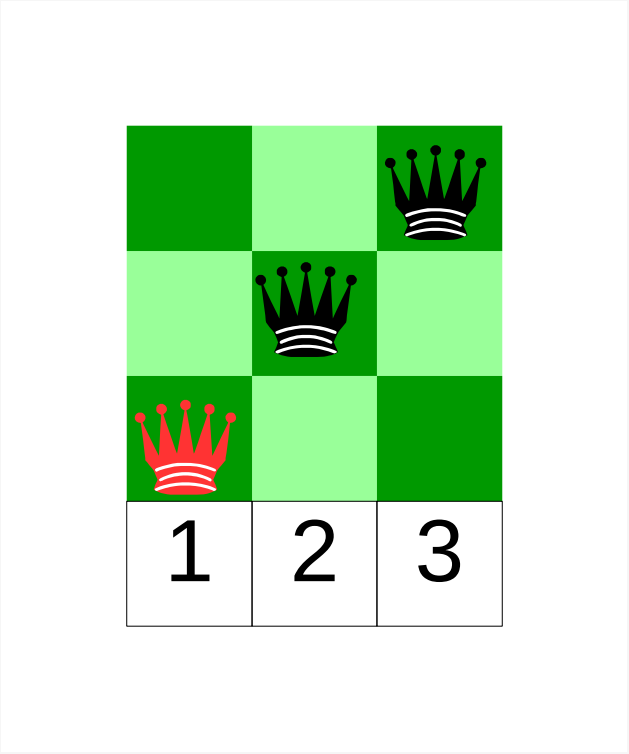
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

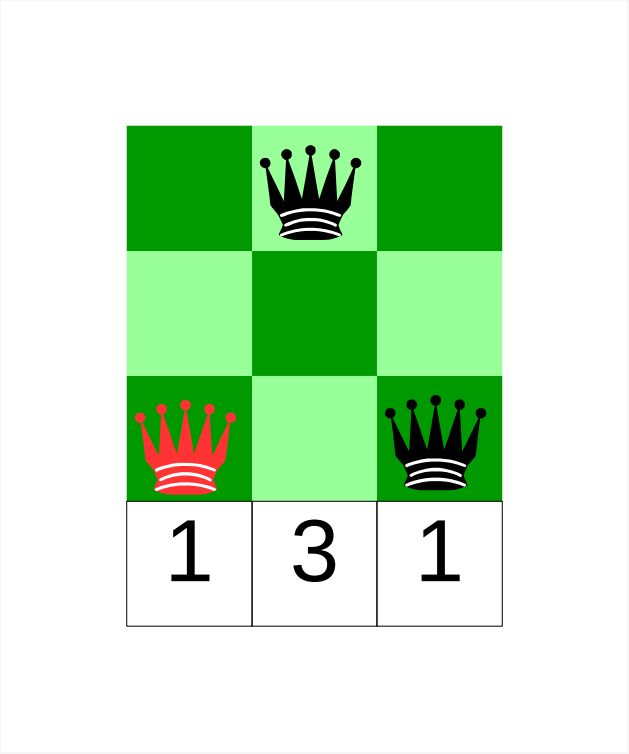
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

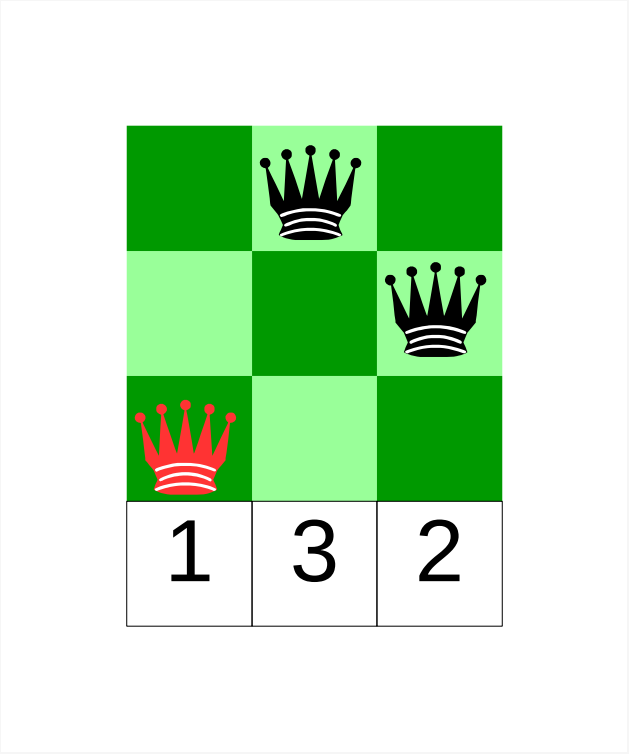
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

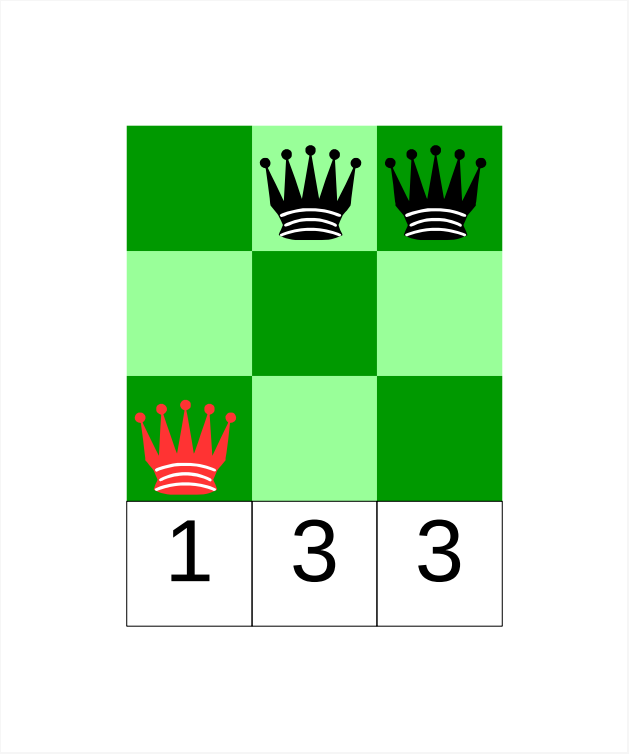
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

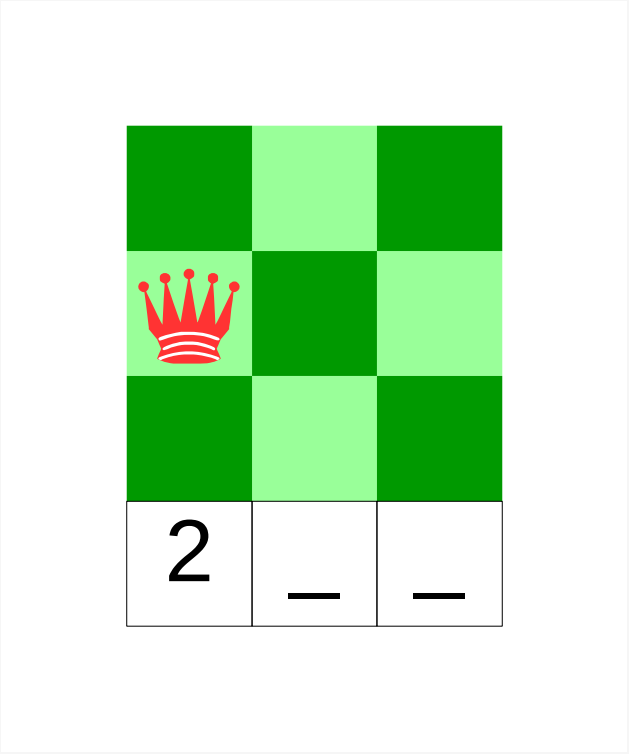
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2

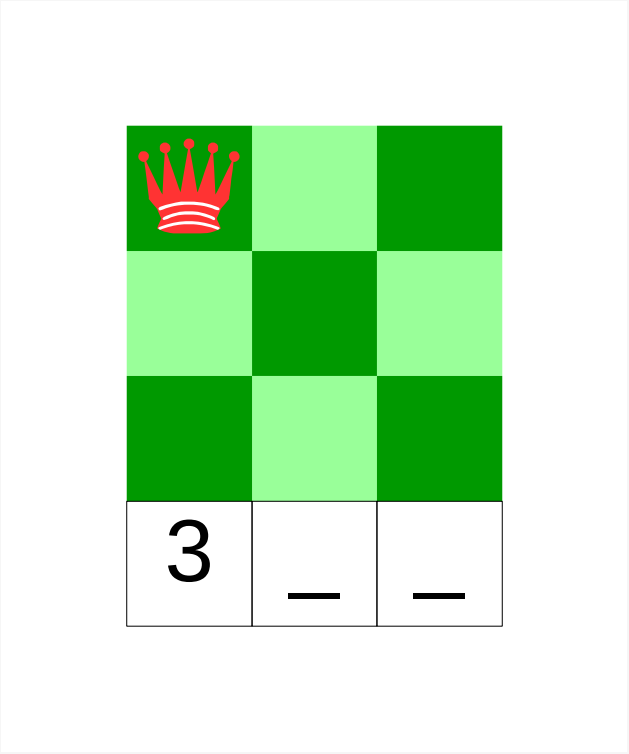
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2
3. repetimos para as outras possibilidades

Enumerando



Enumerando vetor de tamanho 3

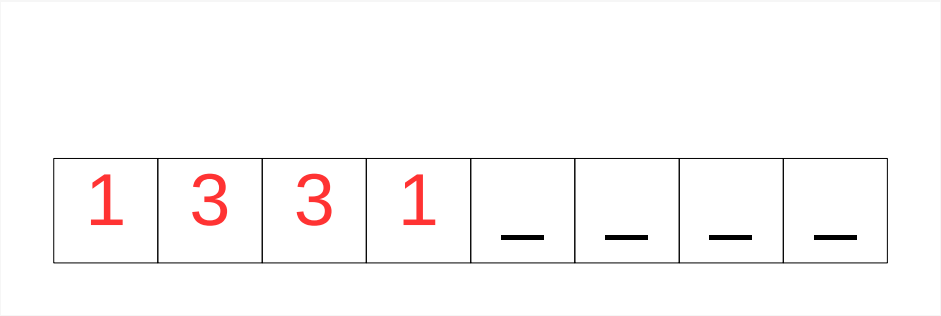
1. fixamos a primeira posição
2. testamos todas as combinações de tamanho 2
3. repetimos para as outras possibilidades

Enumerando em geral

Como imprimir todas as combinações com prefixo dado?

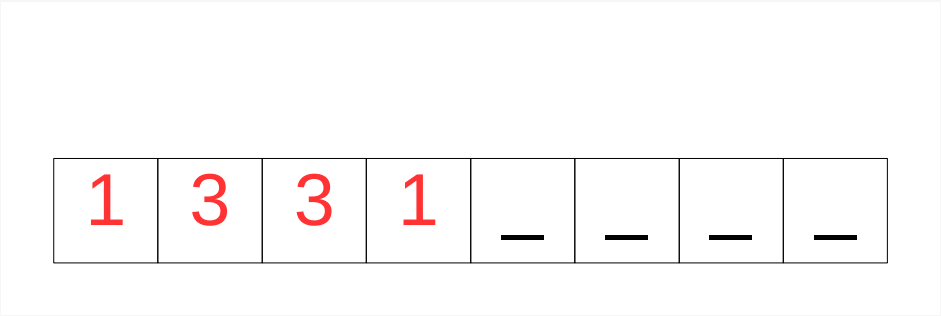
Enumerando em geral

Como imprimir todas as combinações com prefixo dado?



Enumerando em geral

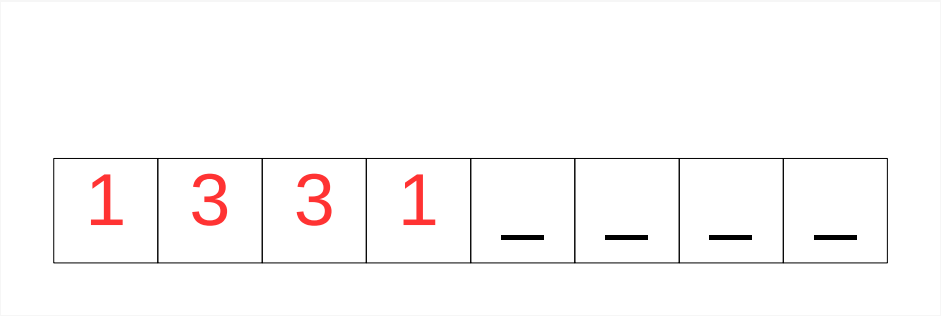
Como imprimir todas as combinações com prefixo dado?



Vamos escrever uma função que receba um vetor:

Enumerando em geral

Como imprimir todas as combinações com prefixo dado?

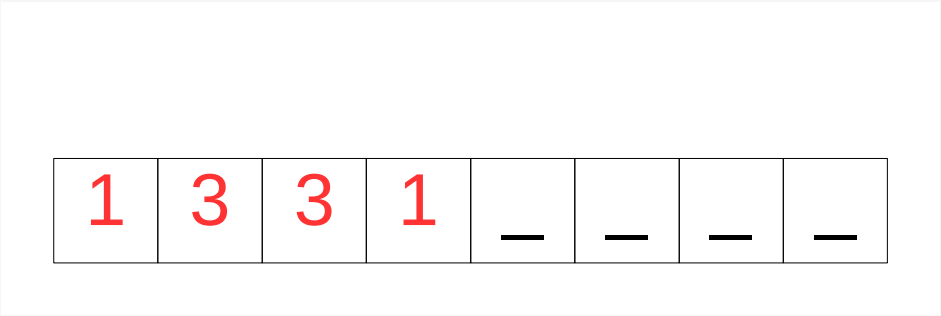


Vamos escrever uma função que receba um vetor:

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

Enumerando em geral

Como imprimir todas as combinações com prefixo dado?



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 combinações (recursivamente)

Programando

Como imprimir todas combinações (recursivamente)

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

Programando

Como imprimir todas combinações (recursivamente)

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

Programando

Como imprimir todas combinações (recursivamente)

```
1 void enumerar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   }
```

Programando

Como imprimir todas combinações (recursivamente)

```
1  void enumerar(int vetor[], int m, int n) {
2    int i;
3    if (n == m) { // todo vetor fixo, só há uma combinação
4      imprimir_vetor(vetor, n);
5    } else {
6      for (i = 1; i <= 8; i++) {
7        vetor[m] = i // fixa primeira
8        enumerar(vetor, m + 1, n); // enumera o resto
9      }
10   }
11 }
```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
1 int existe_sol(int vetor[], int m, int n) {
```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
1 int existe_sol(int vetor[], int m, int n) {
```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      if (eh_disposicao_valida(vetor, n)) {
5        imprimir_vetor(vetor, n);
6        return 1;
7      } else {
8        return 0;
9      }
10   }
```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      if (eh_disposicao_valida(vetor, n)) {
5        imprimir_vetor(vetor, n);
6        return 1;
7      } else {
8        return 0;
9      }
10   } else {
11     for (i = 1; i <= 8; i++) {
12       vetor[m] = i;
13       // ver se existe solução com prefixo
14       if (existe_sol(vetor, m + 1, n))
15         return 1;
16     }
```

Enumerando e testando

Existe solução com as primeiras damas já dispostas?

```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      if (eh_disposicao_valida(vetor, n)) {
5        imprimir_vetor(vetor, n);
6        return 1;
7      } else {
8        return 0;
9      }
10   } else {
11     for (i = 1; i <= 8; i++) {
12       vetor[m] = i;
13       // ver se existe solução com prefixo
14       if (existe_sol(vetor, m + 1, n))
15         return 1;
16     }
17     return 0; // não encontrou nenhuma solução com prefixo
18   }
19 }
```

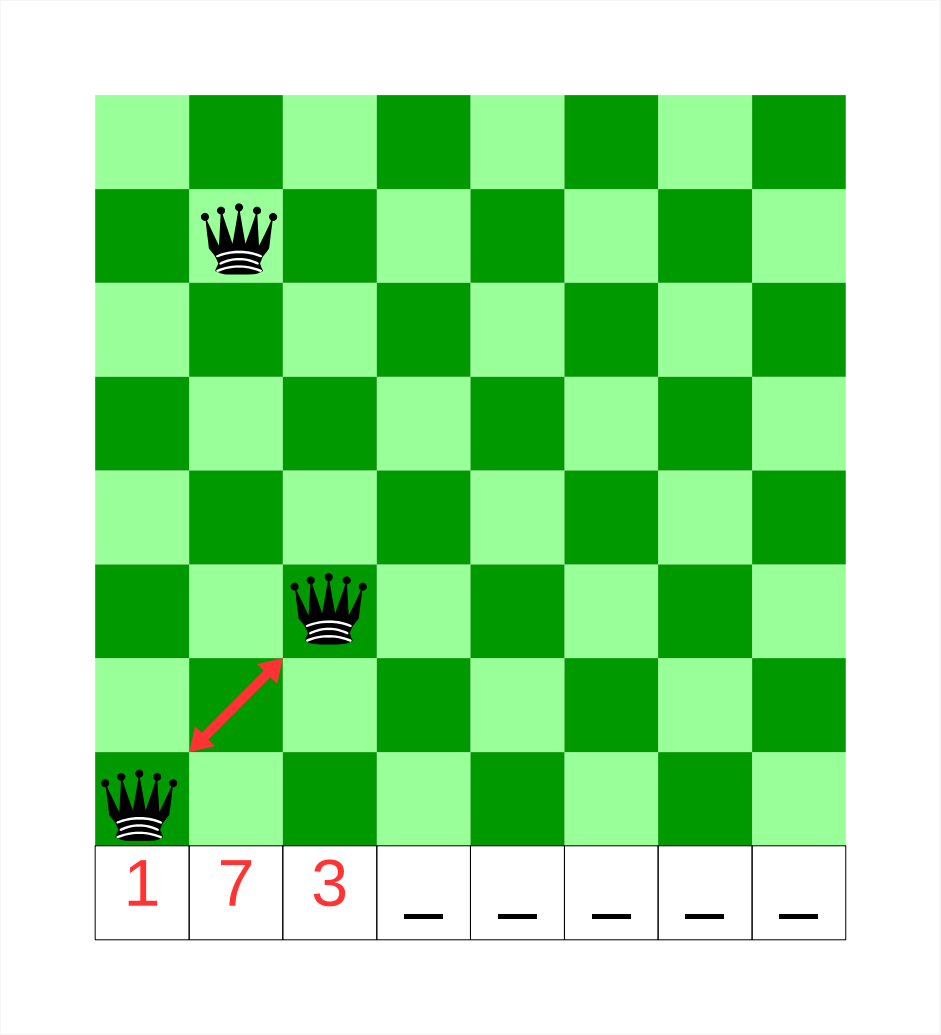
Enumerando e testando

Existe solução com as primeiras damas já dispostas?

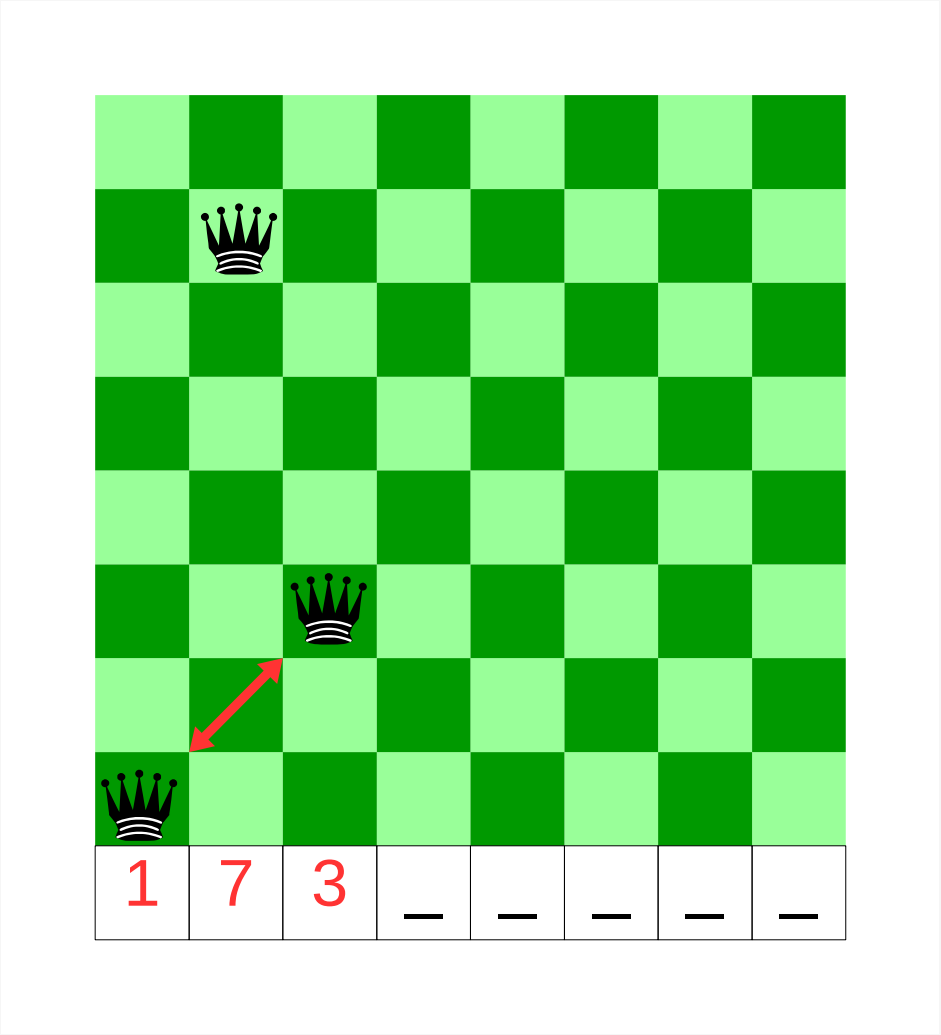
```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      if (eh_disposicao_valida(vetor, n)) {
5        imprimir_vetor(vetor, n);
6        return 1;
7      } else {
8        return 0;
9      }
10   } else {
11     for (i = 1; i <= 8; i++) {
12       vetor[m] = i;
13       // ver se existe solução com prefixo
14       if (existe_sol(vetor, m + 1, n))
15         return 1;
16     }
17     return 0; // não encontrou nenhuma solução com prefixo
18   }
19 }
```

Exercício: implementar `eh_disposicao_valida`

Melhorando um pouco

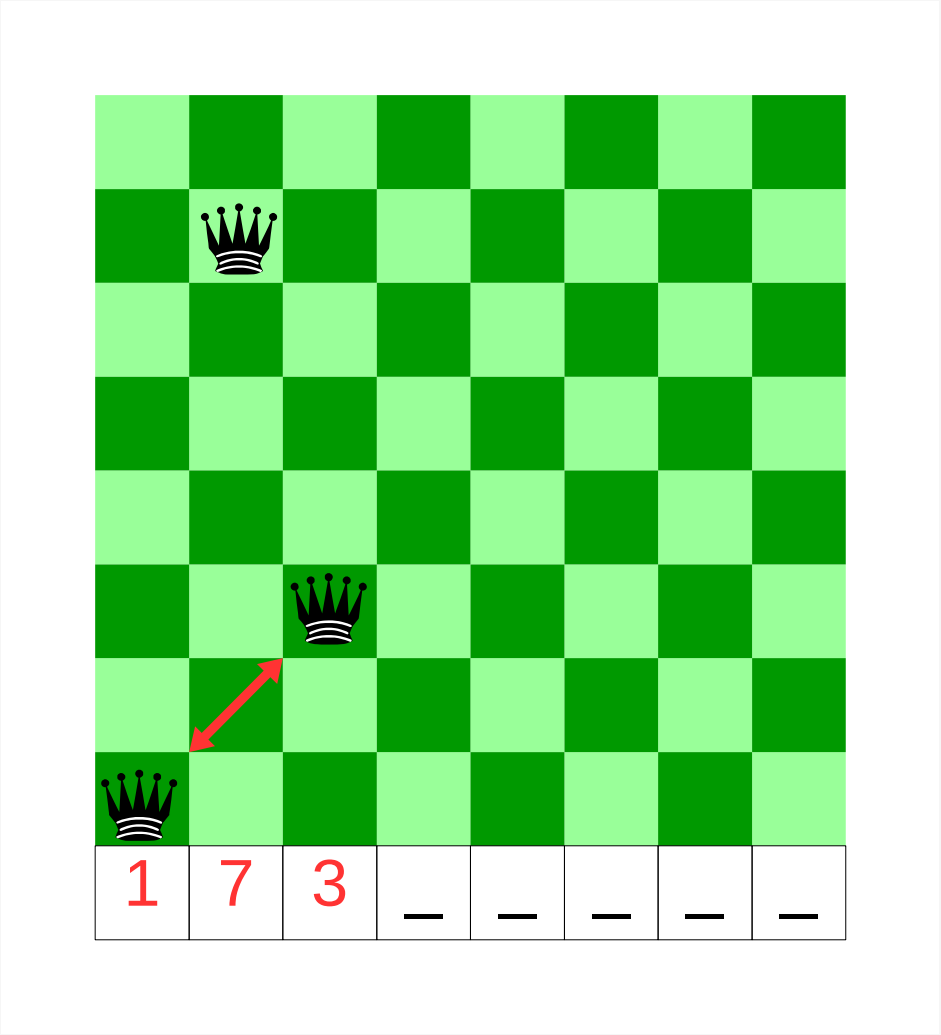


Melhorando um pouco



- alguns prefixos não são viáveis

Melhorando um pouco



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

Melhorando

Podemos sair antes de testar combinações desnecessárias!

Melhorando

Podemos sair antes de testar combinações desnecessárias!

```
1 int existe_sol(int vetor[], int m, int n) {
2   int i;
3   if (n == m) {
4     ...
5   }
```

Melhorando

Podemos sair antes de testar combinações desnecessárias!

```
1 int existe_sol(int vetor[], int m, int n) {
2   int i;
3   if (n == m) {
4     ...
5   }
```

Melhorando

Podemos sair antes de testar combinações desnecessárias!

```
1 int existe_sol(int vetor[], int m, int n) {
2   int i;
3   if (n == m) {
4     ...
5   } else {
6     if (!eh_prefixo_viavel(vetor, m)) {
7       return 0;
8     }
```

Melhorando

Podemos sair antes de testar combinações desnecessárias!

```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      ...
5    } else {
6      if (!eh_prefixo_viavel(vetor, m)) {
7        return 0;
8      }
9      for (i = 1; i <= 8; i++) {
10       ....
11     }
```

Melhorando

Podemos sair antes de testar combinações desnecessárias!

```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      ...
5    } else {
6      if (!eh_prefixo_viavel(vetor, m)) {
7        return 0;
8      }
9      for (i = 1; i <= 8; i++) {
10       ....
11     }
12     return 0; // não encontrou nenhuma solução com prefixo
13   }
14 }
```

Melhorando

Podemos sair antes de testar combinações desnecessárias!

```
1  int existe_sol(int vetor[], int m, int n) {
2    int i;
3    if (n == m) {
4      ...
5    } else {
6      if (!eh_prefixo_viavel(vetor, m)) {
7        return 0;
8      }
9      for (i = 1; i <= 8; i++) {
10       ....
11     }
12     return 0; // não encontrou nenhuma solução com prefixo
13   }
14 }
```

Testando prefixo

```
1 int eh_prefixo_viavel(int vetor[], int m) {
2   int i, lin;
3   lin = vetor[m-1]; // posição do último no prefixo
```

Testando prefixo

```
1 int eh_prefixo_viavel(int vetor[], int m) {
2   int i, lin;
3   lin = vetor[m-1]; // posição do último no prefixo
```

Testando prefixo

```
1 int eh_prefixo_viavel(int vetor[], int m) {
2   int i, lin;
3   lin = vetor[m-1]; // posição do último no prefixo
4   for (i = 0; i < m - 1; i++) {
```

Testando prefixo

```
1 int eh_prefixo_viavel(int vetor[], int m) {
2   int i, lin;
3   lin = vetor[m-1]; // posição do último no prefixo
4   for (i = 0; i < m - 1; i++) {
5     // se está na mesma linha
6     if (vetor[i] == lin)
7       return 0;
```

Testando prefixo

```
1  int eh_prefixo_viavel(int vetor[], int m) {
2    int i, lin;
3    lin = vetor[m-1]; // posição do último no prefixo
4    for (i = 0; i < m - 1; i++) {
5      // se está na mesma linha
6      if (vetor[i] == lin)
7        return 0;
8      // se está na mesma diagonal
9      if ((m - 1) - i == abs(lin - vetor[i]))
10       return 0;
11   }
```

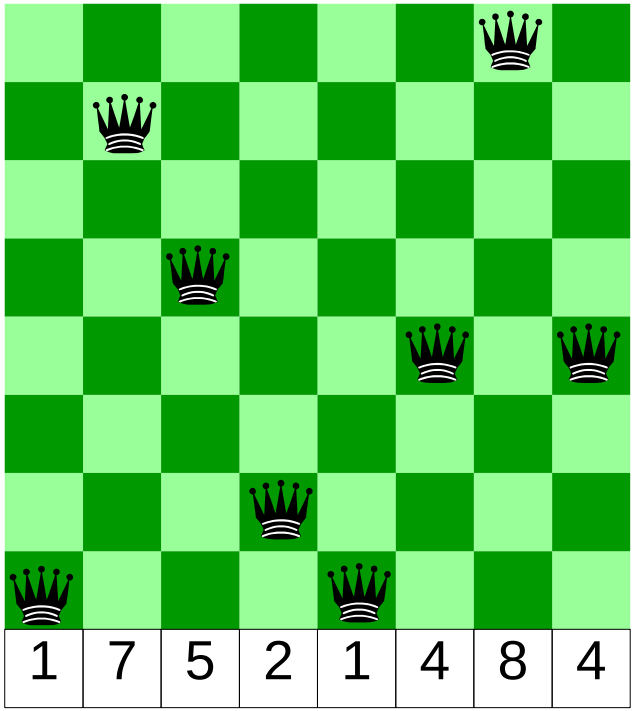
Testando prefixo

```
1  int eh_prefixo_viavel(int vetor[], int m) {
2    int i, lin;
3    lin = vetor[m-1]; // posição do último no prefixo
4    for (i = 0; i < m - 1; i++) {
5      // se está na mesma linha
6      if (vetor[i] == lin)
7        return 0;
8      // se está na mesma diagonal
9      if ((m - 1) - i == abs(lin - vetor[i]))
10       return 0;
11   }
12   return 1;
13 }
```

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

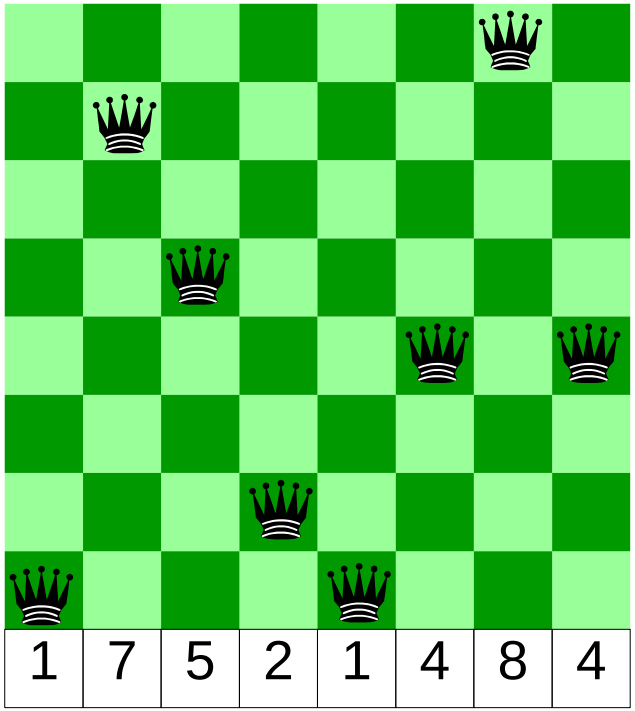
Modificando a estratégia

Como diminuir as combinações testadas?



Modificando a estratégia

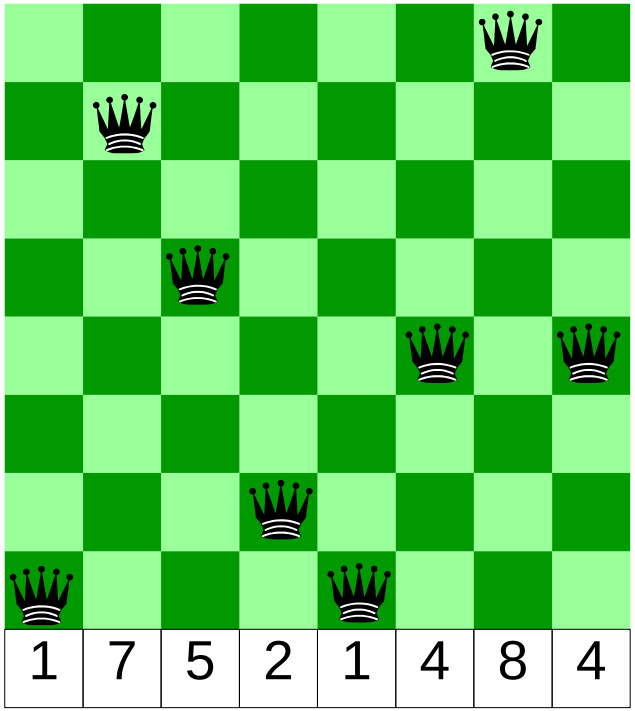
Como diminuir as combinações testadas?



- cada coluna só deve ter uma dama:

Modificando a estratégia

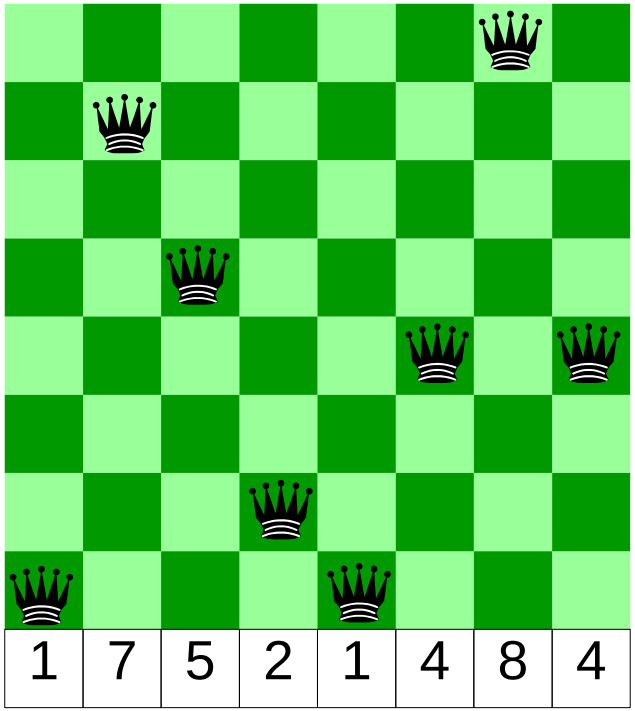
Como diminuir as combinações testadas?



- cada coluna só deve ter uma dama: ✓

Modificando a estratégia

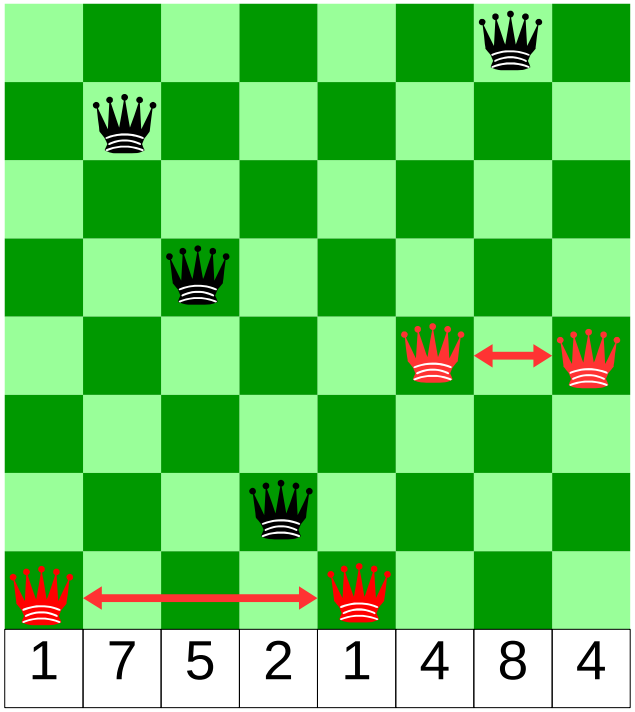
Como diminuir as combinações testadas?



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

Modificando a estratégia

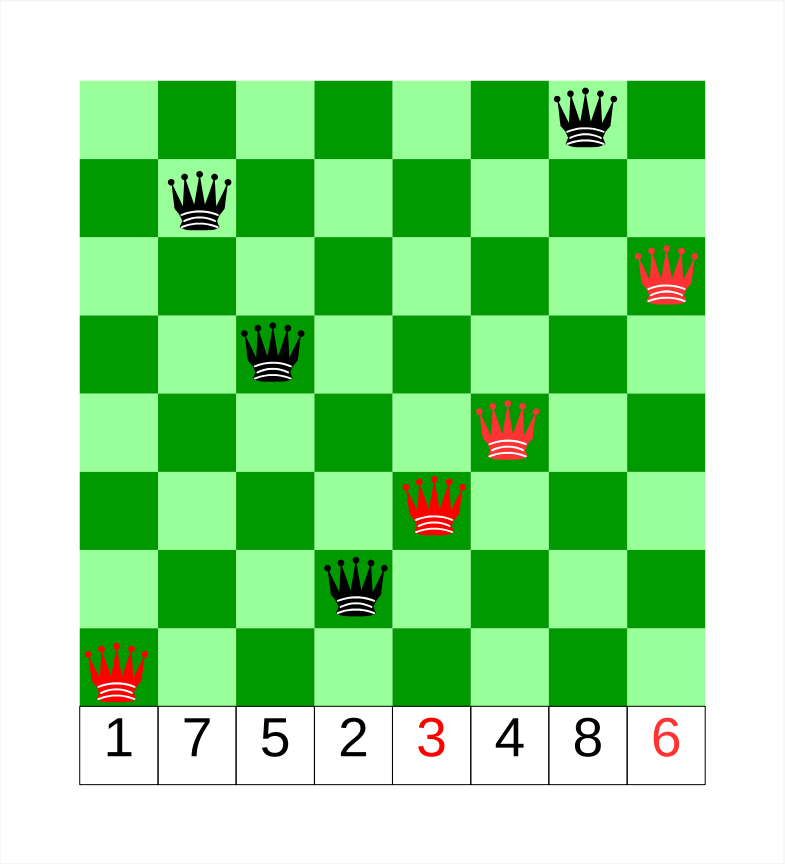
Como diminuir as combinações testadas?



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

Modificando a estratégia

Como diminuir as combinações testadas?

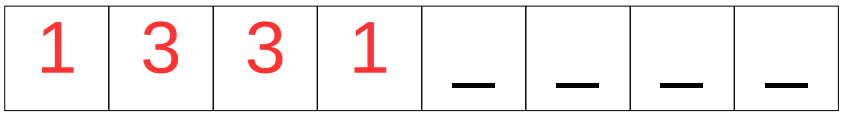


- cada coluna só deve ter uma dama: ✓
- cada linha só deve ter uma dama: ✗

Observação: uma configuração deve ser uma permutação

Gerando permutações

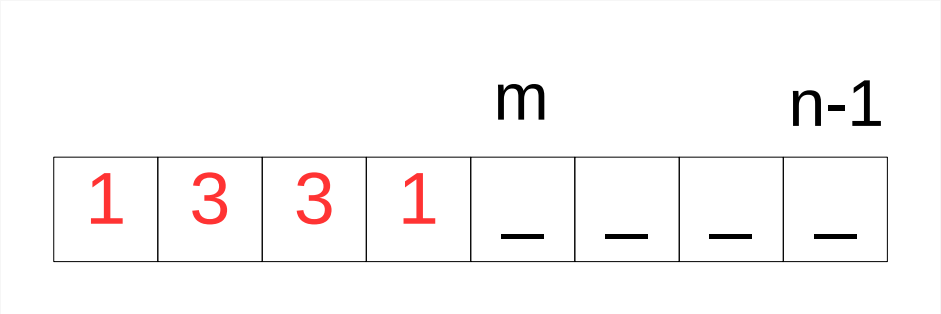
Vamos escrever uma função que receba uma permutação



Gerando permutações

Vamos escrever uma função que receba uma permutação

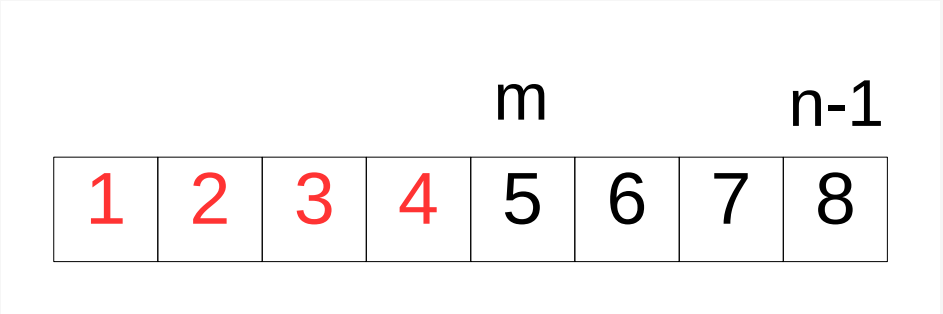
- com valores fixos até uma posição $m - 1$



Gerando permutações

Vamos escrever uma função que receba uma permutação

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

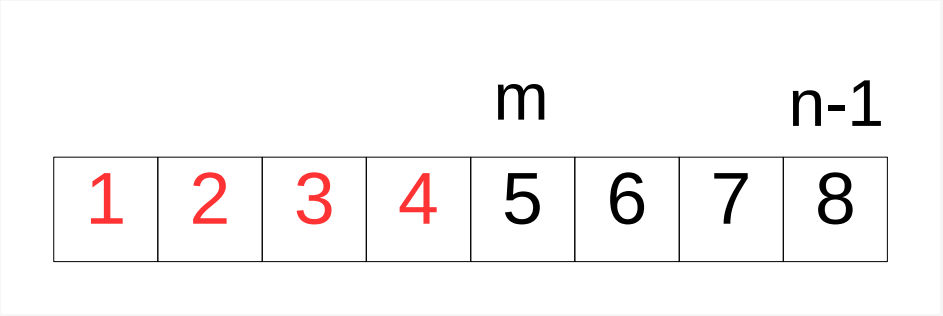


Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.

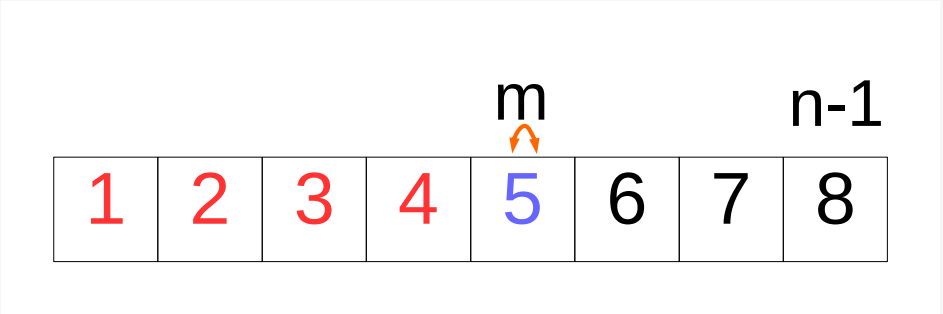


Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



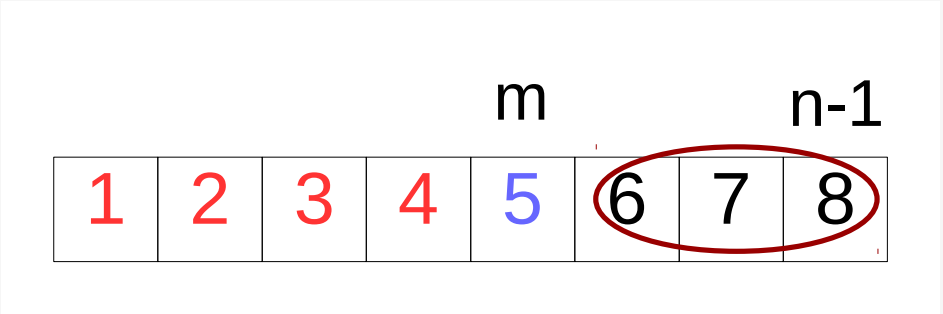
1. fixa primeiro elemento

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



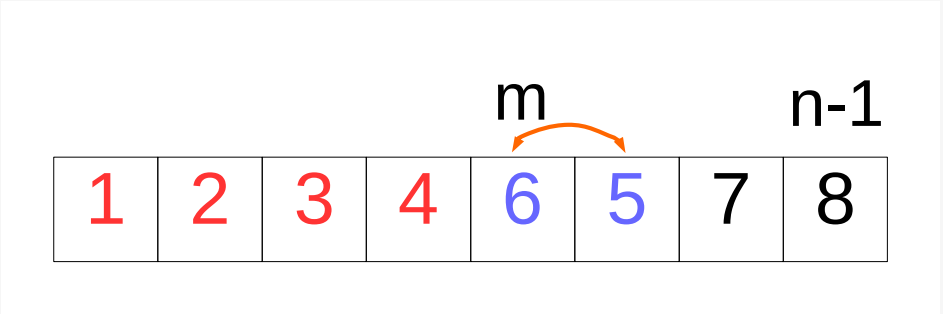
1. fixa primeiro elemento e permuta outros

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



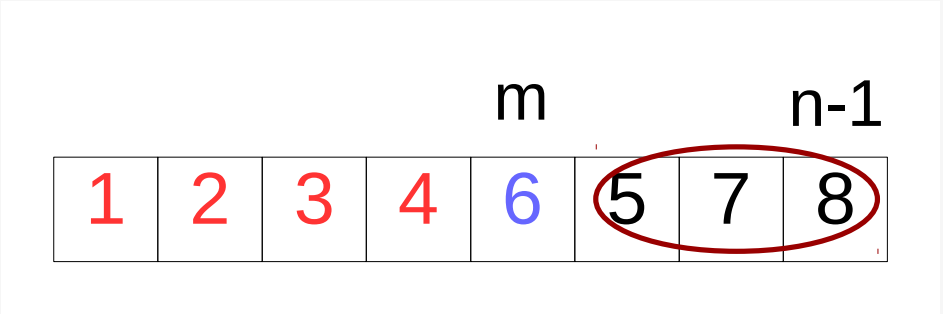
1. fixa primeiro elemento e permuta outros
2. troca primeiro com segundo

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



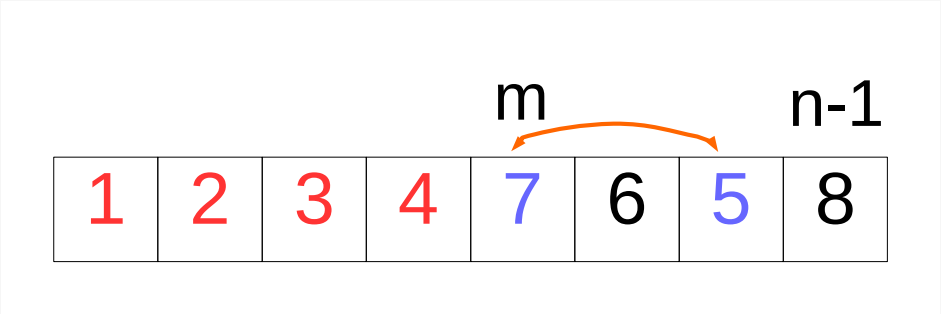
1. fixa primeiro elemento e permuta outros
2. troca primeiro com segundo e permuta outros

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



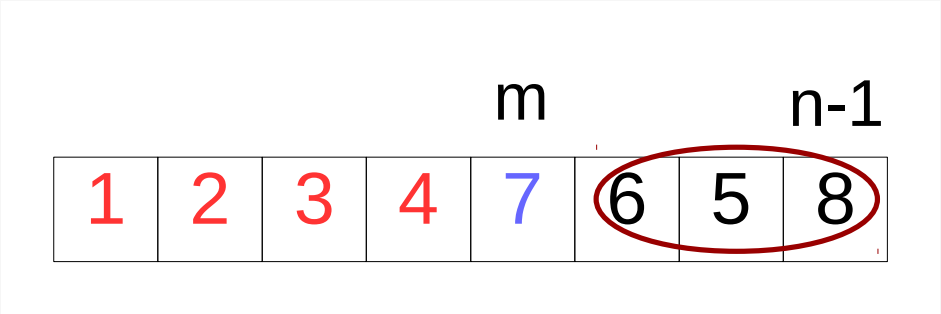
1. fixa primeiro elemento e permuta outros
2. troca primeiro com segundo e permuta outros
3. troca primeiro com terceiro

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



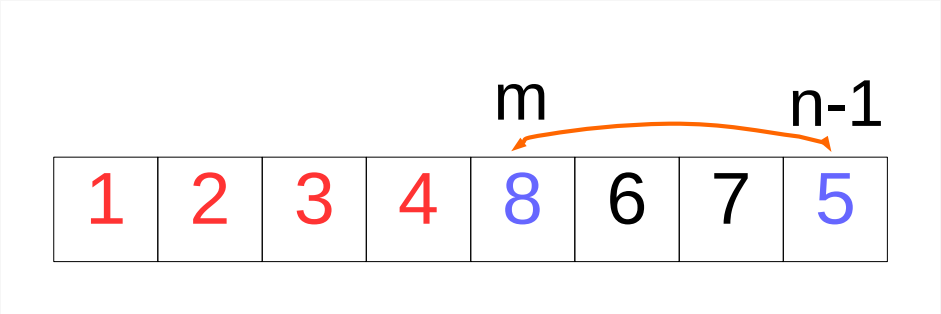
1. fixa primeiro elemento e permuta outros
2. troca primeiro com segundo e permuta outros
3. troca primeiro com terceiro e permuta outros

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



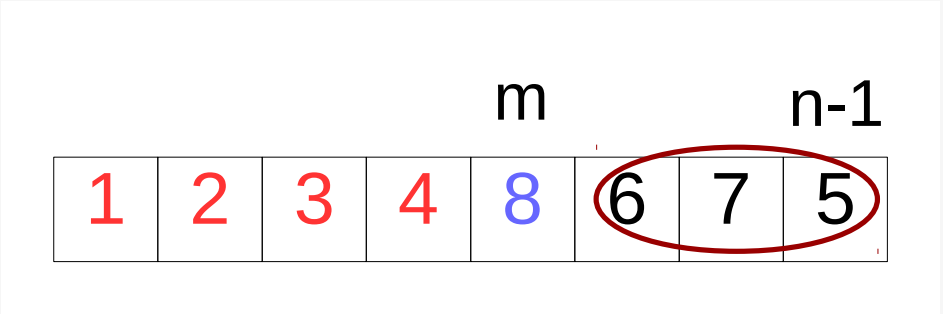
1. fixa primeiro elemento e permuta outros
2. troca primeiro com segundo e permuta outros
3. troca primeiro com terceiro e permuta outros
4. etc...

Gerando permutações

Vamos escrever uma função que receba uma permutação

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

e imprima todas as permutações com prefixo dado.



1. fixa primeiro elemento e permuta outros
2. troca primeiro com segundo e permuta outros
3. troca primeiro com terceiro e permuta outros
4. etc...

Programando

Enumerando vetores de tamanho n:

```
1 void permutar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   }
```

Programando

Enumerando vetores de tamanho n:

```
1 void permutar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   }
```

Programando

Enumerando vetores de tamanho n:

```
1 void permutar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   } else {
6     for (i = m; i < n; i++) {
```

Programando

Enumerando vetores de tamanho n:

```
1 void permutar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   } else {
6     for (i = m; i < n; i++) {
7       troca(&vetor[m], &vetor[i]); //troca e fixa
```

Programando

Enumerando vetores de tamanho n:

```
1 void permutar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   } else {
6     for (i = m; i < n; i++) {
7       troca(&vetor[m], &vetor[i]); //troca e fixa
8       permutar(vetor, m + 1, n); // permuta o resto
```

Programando

Enumerando vetores de tamanho n:

```
1 void permutar(int vetor[], int m, int n) {
2   int i;
3   if (n == m) { // todo vetor fixo, só há uma combinação
4     imprimir_vetor(vetor, n);
5   } else {
6     for (i = m; i < n; i++) {
7       troca(&vetor[m], &vetor[i]); //troca e fixa
8       permutar(vetor, m + 1, n); // permuta o resto
9       troca(&vetor[m], &vetor[i]); //volta ao original
```

Programando

Enumerando vetores de tamanho n:

```
1  void permutar(int vetor[], int m, int n) {
2    int i;
3    if (n == m) { // todo vetor fixo, só há uma combinação
4      imprimir_vetor(vetor, n);
5    } else {
6      for (i = m; i < n; i++) {
7        troca(&vetor[m], &vetor[i]); //troca e fixa
8        permutar(vetor, m + 1, n); // permuta o resto
9        troca(&vetor[m], &vetor[i]); //volta ao original
10     }
11   }
12 }
```

Exercício: implementar para 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

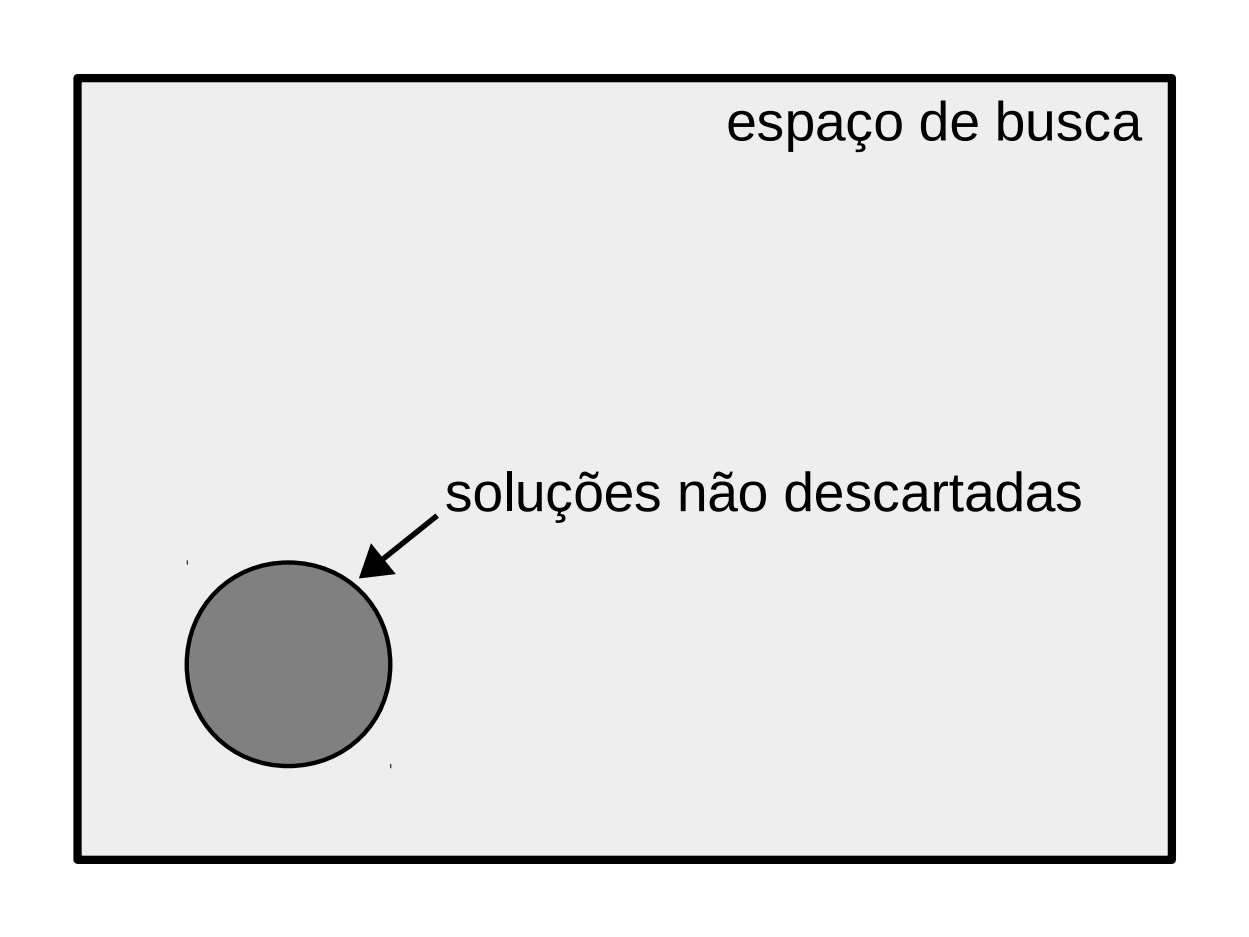
Backtracking ou retrocesso é um algoritmo genérico, com as seguintes propriedades

- 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

- Em geral, mais rápido que a Força Bruta pois eliminamos vários candidatos a solução de uma só vez

Eficiência do Backtracking

- Em geral, mais rápido que a Força Bruta pois eliminamos vários candidatos a solução de uma só vez
- Implementação simples, mas pode ser lento para problemas onde temos muitas soluções parciais possíveis

Eficiência do Backtracking

- Em geral, mais rápido que a Força Bruta pois eliminamos vários candidatos a solução de uma só vez
- Implementação simples, mas pode ser lento para problemas onde temos muitas soluções parciais possíveis

Como fazer um algoritmo de Backtracking rápido?

Eficiência do Backtracking

- Em geral, mais rápido que a Força Bruta pois eliminamos vários candidatos a solução de uma só vez
- Implementação simples, mas pode ser lento para problemas onde temos muitas soluções parciais possíveis

Como fazer um algoritmo de Backtracking rápido?

- É importante ter um algoritmo para decidir se uma solução parcial pode ser estendida para uma solução completa que seja

Eficiência do Backtracking

- Em geral, mais rápido que a Força Bruta pois eliminamos vários candidatos a solução de uma só vez
- Implementação simples, mas pode ser lento para problemas onde temos muitas soluções parciais possíveis

Como fazer um algoritmo de Backtracking rápido?

- É importante ter um algoritmo para decidir se uma solução parcial pode ser estendida para uma solução completa que seja
  - Bom: Evita explorar muitas soluções parciais

Eficiência do Backtracking

- Em geral, mais rápido que a Força Bruta pois eliminamos vários candidatos a solução de uma só vez
- Implementação simples, mas pode ser lento para problemas onde temos muitas soluções parciais possíveis

Como fazer um algoritmo de Backtracking rápido?

- É importante ter um algoritmo para decidir se uma solução parcial pode ser estendida para uma solução completa que seja
  - 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 $x_1 \leq x_2 \leq \cdots x_n$