MC-202 — Aula 6
Retrocesso e recursão mútua

Lehilton Pedrosa

Instituto de Computação – Unicamp

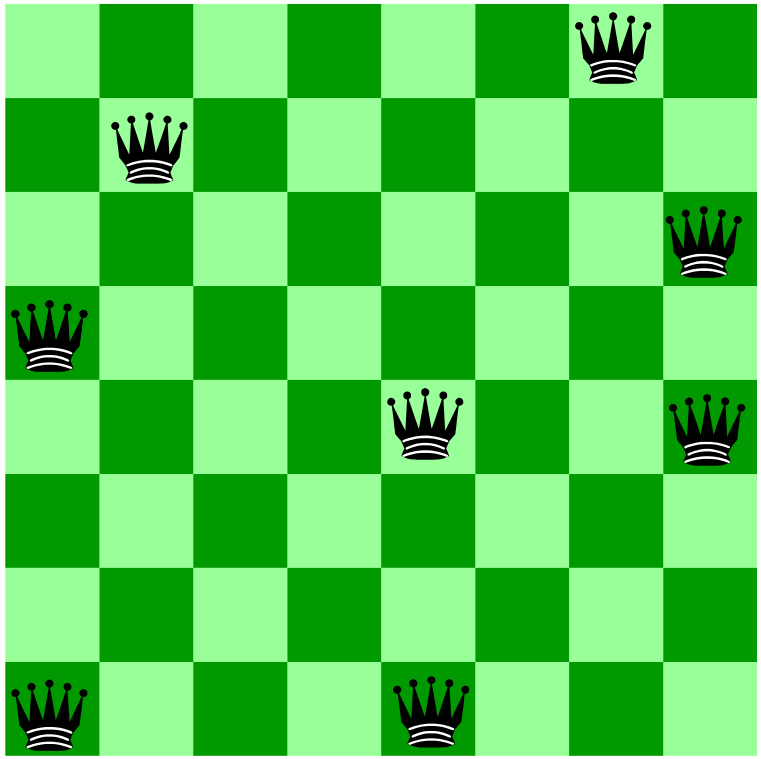
Segundo Semestre de 2015

Roteiro

Um problema

Oito damas

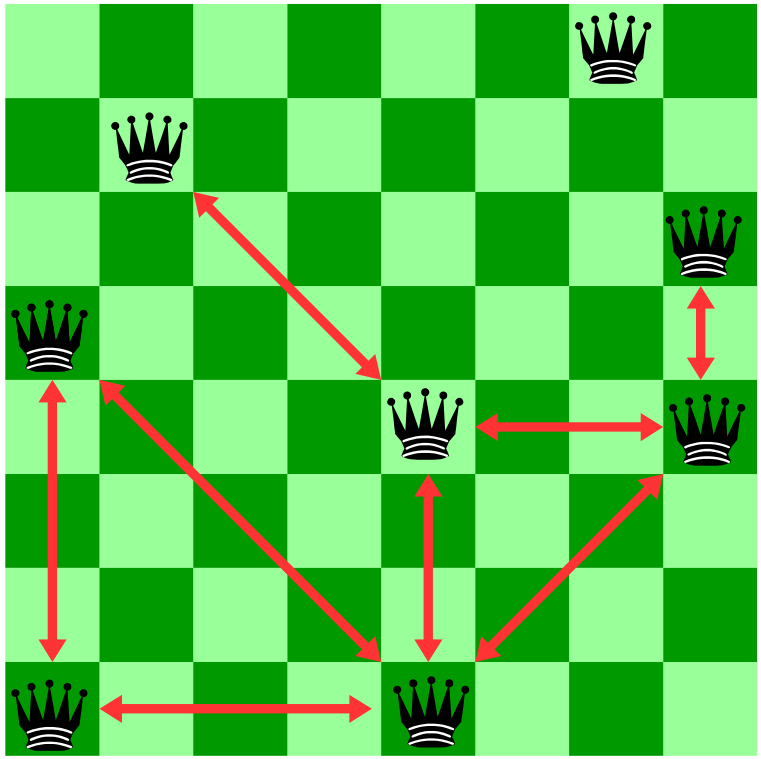
Como dispor oito damas em um tabuleiro de xadrez, sem posições de ameaça?



Um problema

Oito damas

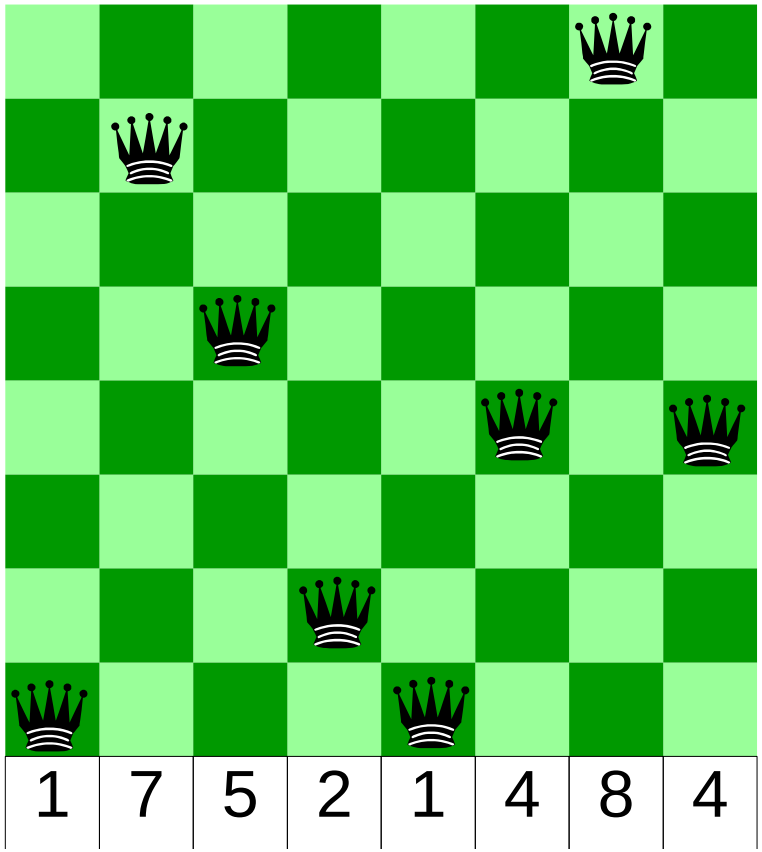
Como dispor oito damas em um tabuleiro de xadrez, sem posições de ameaça?



Testando (quase) todas as soluções

Ideia

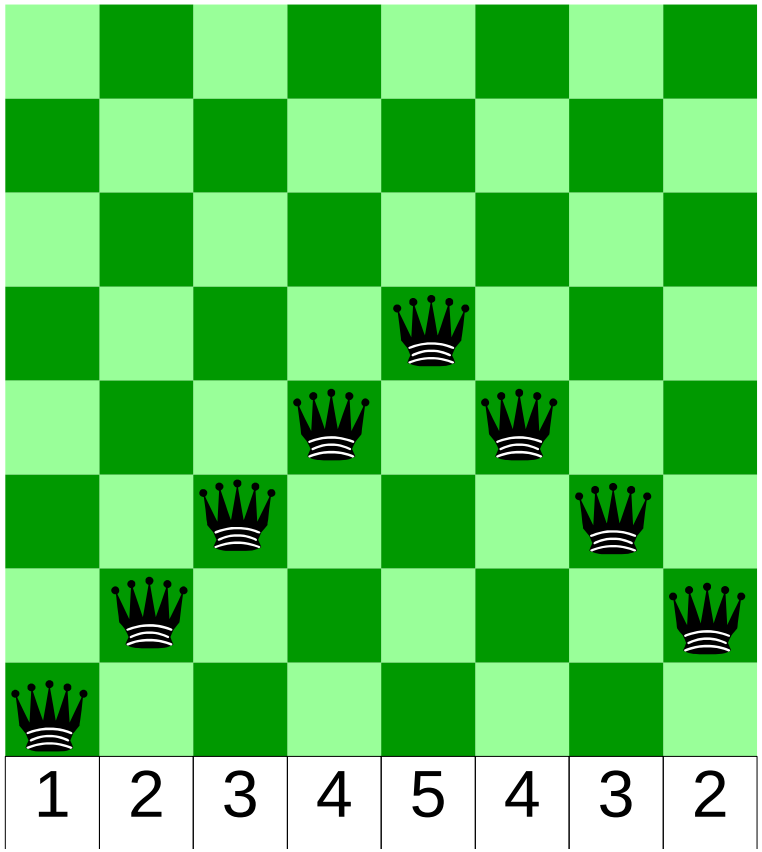
- Cada coluna dever ter exatamente uma dama
- Representamos uma disposição com um vetor



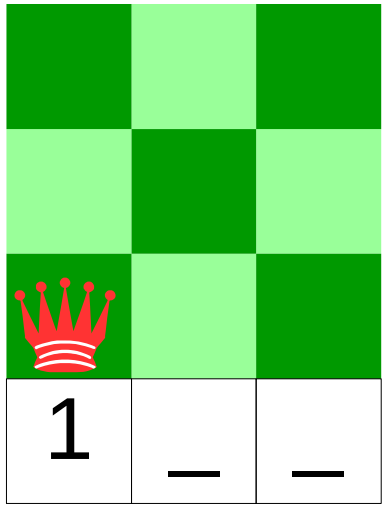
Testando (quase) todas as soluções

Ideia

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



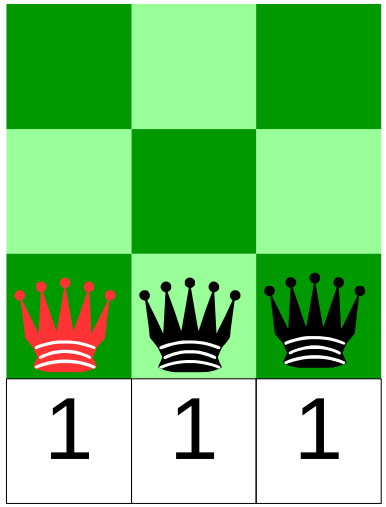
Enumerando



Enumerando vetor de tamanho 3

1. fixamos a primeira posição

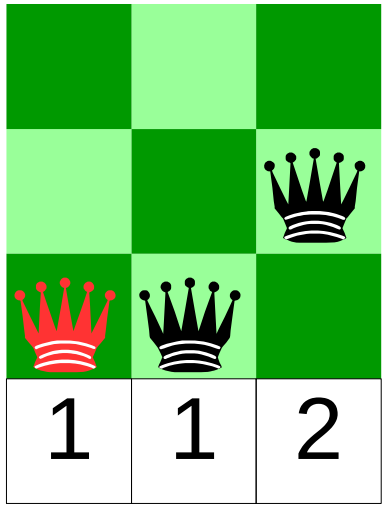
Enumerando



Enumerando vetor de tamanho 3

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

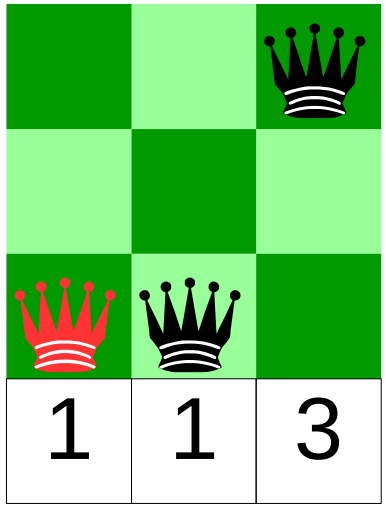
Enumerando



Enumerando vetor de tamanho 3

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

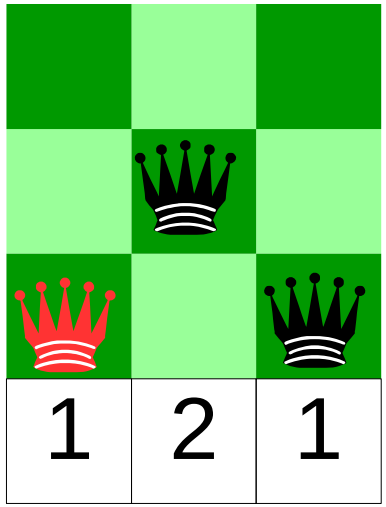
Enumerando



Enumerando vetor de tamanho 3

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

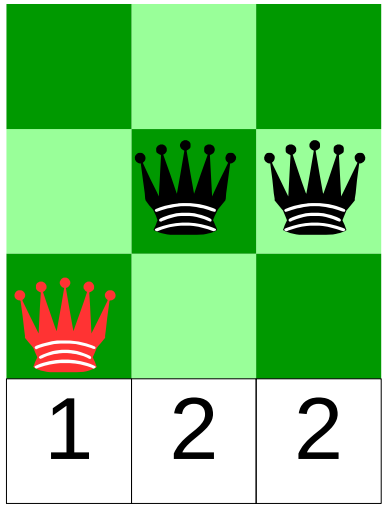
Enumerando



Enumerando vetor de tamanho 3

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

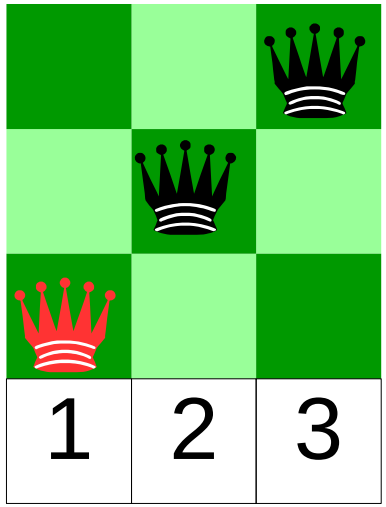
Enumerando



Enumerando vetor de tamanho 3

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

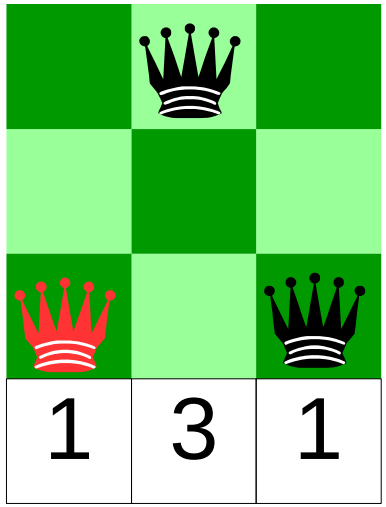
Enumerando



Enumerando vetor de tamanho 3

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

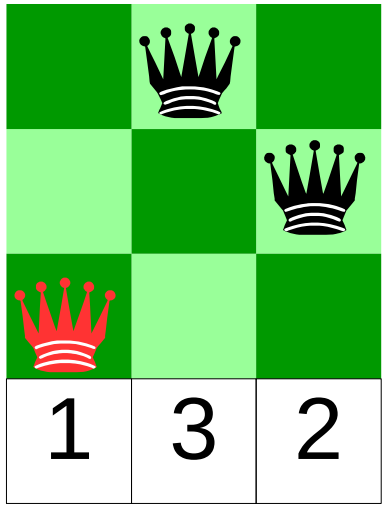
Enumerando



Enumerando vetor de tamanho 3

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

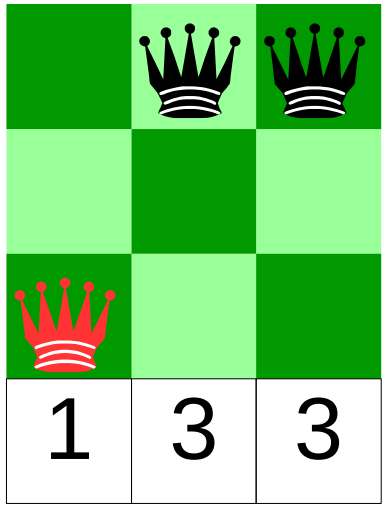
Enumerando



Enumerando vetor de tamanho 3

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

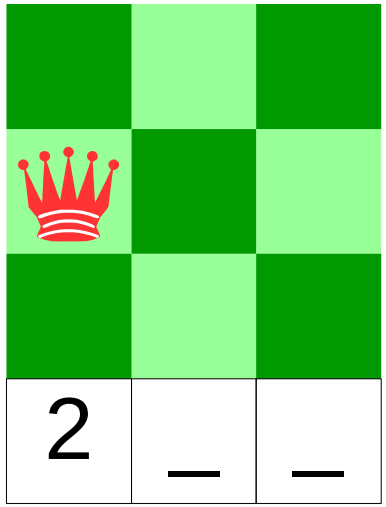
Enumerando



Enumerando vetor de tamanho 3

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

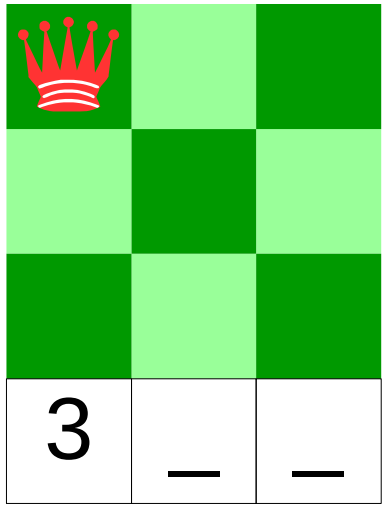
Enumerando



Enumerando vetor de tamanho 3

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

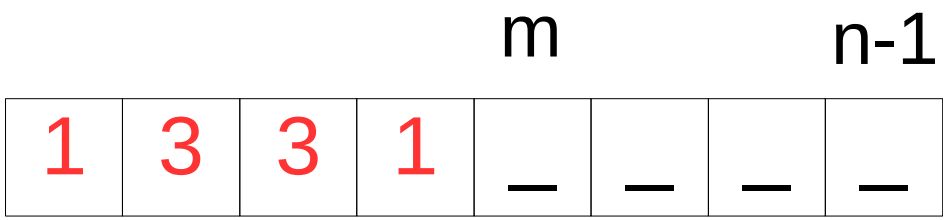
Enumerando



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

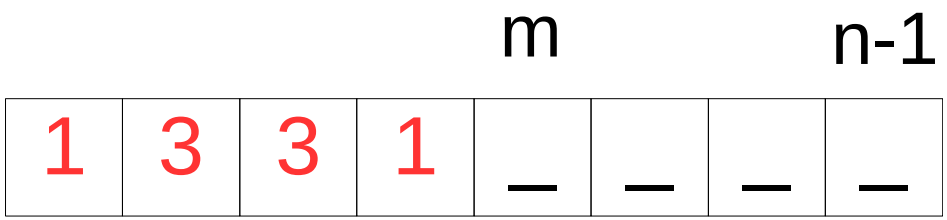


Estratégia

Vamos escrever uma função que receba um vetor:

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

Enumerando em geral



Estratégia

Vamos escrever uma função que receba um vetor:

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

Objetivo: imprimir todas as combinações com prefixo dado.

Programando

Como imprimir todas combinações (recursivamente)

Enumerando vetores de tamanho n

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

Programando

Como imprimir todas combinações (recursivamente)

Enumerando vetores de tamanho n

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

Programando

Como imprimir todas combinações (recursivamente)

Enumerando vetores de tamanho n

```
void enumerar(int vetor[], int m, int n) {
    int i;
    if (n == m) { // todo vetor fixo, só há uma combinação
        imprimir_vetor(vetor, n);
    } else {
        for (i = 1; i <= 8; i++) {
            vetor[m] = i;                    // fixa primeira
```

Programando

Como imprimir todas combinações (recursivamente)

Enumerando vetores de tamanho n

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

Enumerando e testando

Novo objetivo: ver se existe solução com as primeiras damas já dispostas

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

Enumerando e testando

Novo objetivo: ver se existe solução com as primeiras damas já dispostas

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

Enumerando e testando

Novo objetivo: ver se existe solução com as primeiras damas já dispostas

```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        if (eh_disposicao_valida(vetor, n) {
            imprimir_vetor(vetor, n);
            return 1;
        } else
            return 0;
    } else {
        for (i = 1; i <= 8; i++) {
            vetor[m] = i;
            // ver se existe solução com prefixo
            if (existe_sol(vetor, m + 1, n))
                return 1;
        }
```

Enumerando e testando

Novo objetivo: ver se existe solução com as primeiras damas já dispostas

```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        if (eh_disposicao_valida(vetor, n) {
            imprimir_vetor(vetor, n);
            return 1;
        } else
            return 0;
    } else {
        for (i = 1; i <= 8; i++) {
            vetor[m] = i;
            // ver se existe solução com prefixo
            if (existe_sol(vetor, m + 1, n))
                return 1;
        }
        return 0; // não encontrou nenhuma solução com prefixo
    }
}
```

Enumerando e testando

Novo objetivo: ver se existe solução com as primeiras damas já dispostas

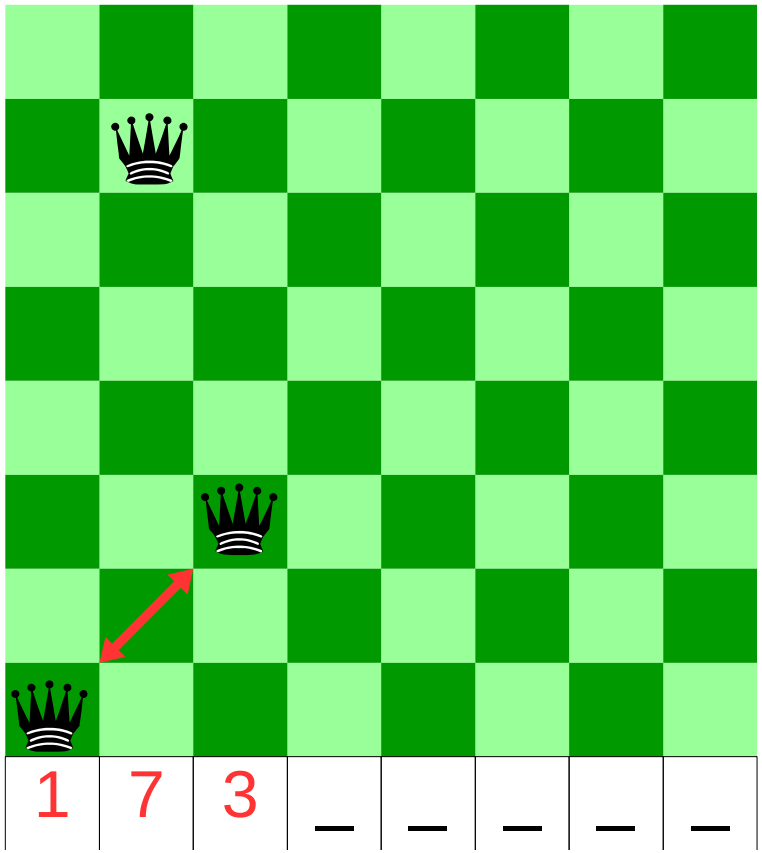
```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        if (eh_disposicao_valida(vetor, n) {
            imprimir_vetor(vetor, n);
            return 1;
        } else
            return 0;
    } else {
        for (i = 1; i <= 8; i++) {
            vetor[m] = i;
            // ver se existe solução com prefixo
            if (existe_sol(vetor, m + 1, n))
                return 1;
        }
        return 0; // não encontrou nenhuma solução com prefixo
    }
}
```

Exercício: implementar `eh_disposicao_valida`

Melhorando um pouco

Ideias

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



Melhorando

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

Melhorando

```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        ...
    }
```

Melhorando

```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        ...
    } else {
        if (!eh_prefixo_viavel(vetor, m)) {
            return 0;
        }
```

Melhorando

```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        ...
    } else {
        if (!eh_prefixo_viavel(vetor, m)) {
            return 0;
        }
        for (i = 1; i <= 8; i++) {
            ....
        }
        return 0; // não encontrou nenhuma solução com prefixo
    }
}
```

Melhorando

```
int existe_sol(int vetor[], int m, int n) {
    int i;
    if (n == m) {
        ...
    } else {
        if (!eh_prefixo_viavel(vetor, m)) {
            return 0;
        }
        for (i = 1; i <= 8; i++) {
            ....
        }
        return 0; // não encontrou nenhuma solução com prefixo
    }
}
```

Saímos antes de testar combinações desnecessárias!

Testando prefixo

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

Testando prefixo

```
int eh_prefixo_viavel(int vetor[], int m) {
    int i, lin;
    lin = vetor[m-1]; // posição do último no prefixo
    for (i = 0; i < m - 1; i++) {
        // se está na mesma linha
        if (vetor[i] == lin)
            return 0;
        // se está na mesma diagonal
        if ((m - 1) - i == abs(lin - vetor[i]))
            return 0;
    }
    return 1;
}
```

Testando prefixo

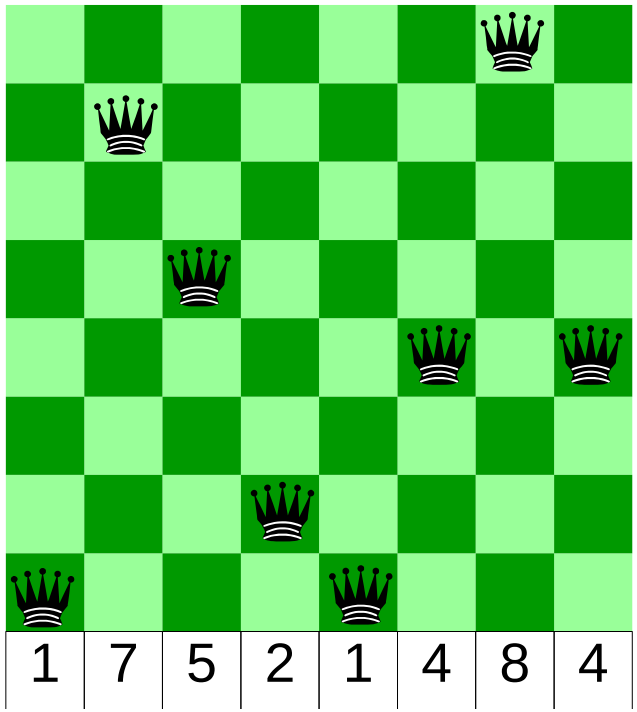
```
int eh_prefixo_viavel(int vetor[], int m) {
    int i, lin;
    lin = vetor[m-1]; // posição do último no prefixo
    for (i = 0; i < m - 1; i++) {
        // se está na mesma linha
        if (vetor[i] == lin)
            return 0;
        // se está na mesma diagonal
        if ((m - 1) - i == abs(lin - vetor[i]))
            return 0;
    }
    return 1;
}
```

Responda: Por que só precisamos comparar o último elemento do prefixo com os anteriores?

Modificando a estratégia

Como diminuir as combinações testadas?

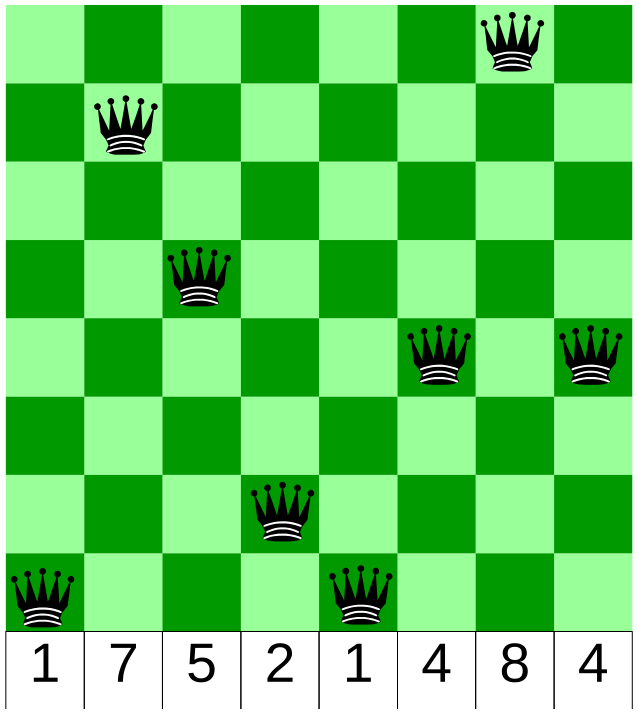
- cada coluna só deve ter uma dama:



Modificando a estratégia

Como diminuir as combinações testadas?

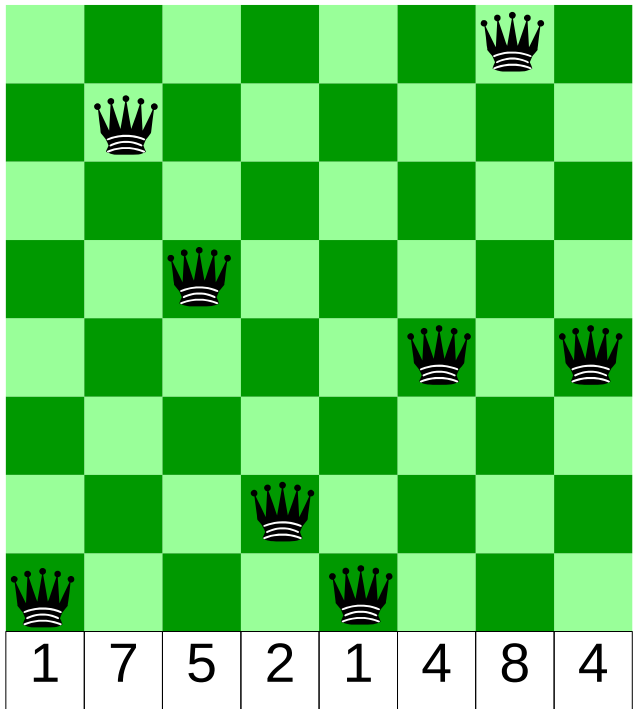
- cada coluna só deve ter uma dama: ✓



Modificando a estratégia

Como diminuir as combinações testadas?

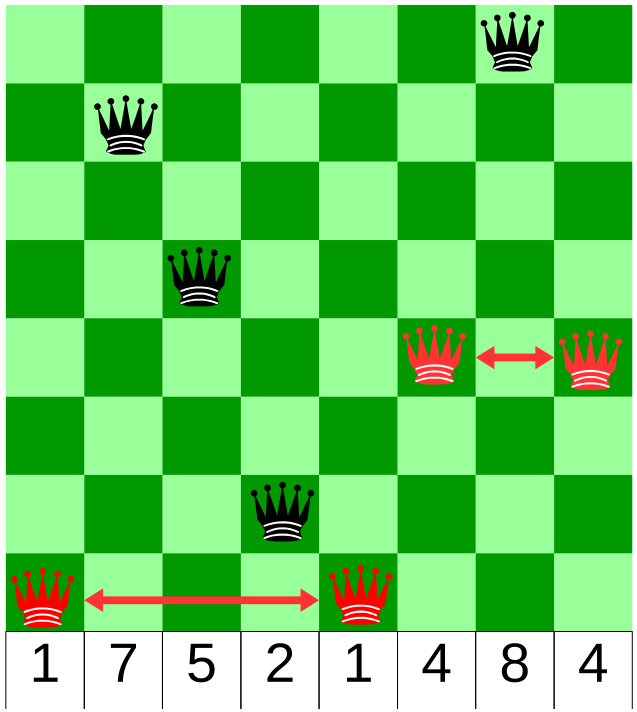
- cada coluna só deve ter uma dama: ✓
- cada linha só deve ter uma dama:



Modificando a estratégia

Como diminuir as combinações testadas?

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

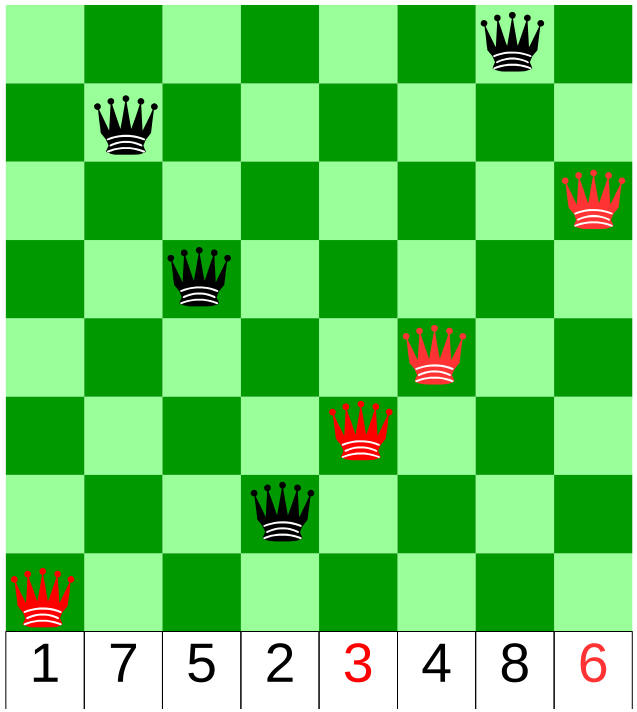


Modificando a estratégia

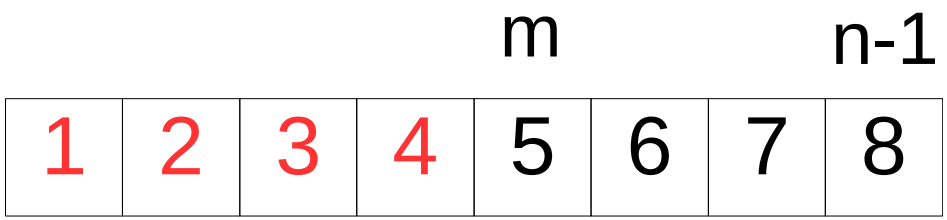
Como diminuir as combinações testadas?

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

Observação: Uma configuração deve ser uma permutação



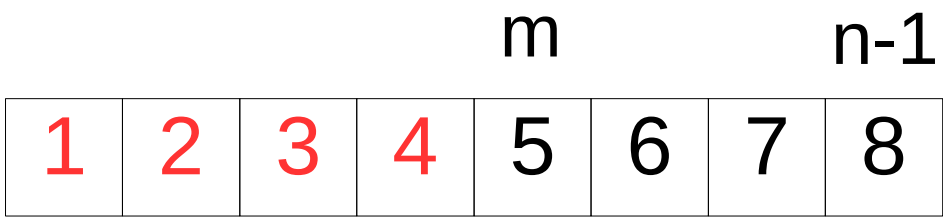
Gerando permutações



Vamos escrever uma função que receba uma permutação:

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

Gerando permutações

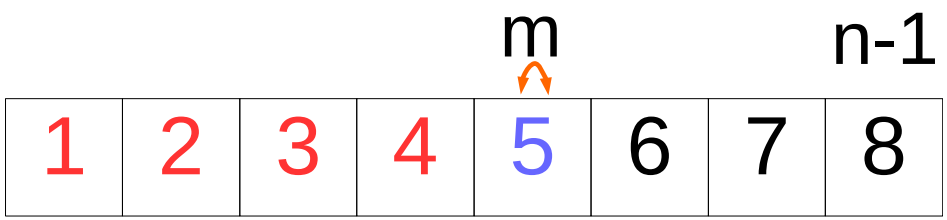


Vamos escrever uma função que receba uma permutação:

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

Objetivo: imprimir todas as permutações com prefixo dado.

Gerando permutações



Vamos escrever uma função que receba uma permutação:

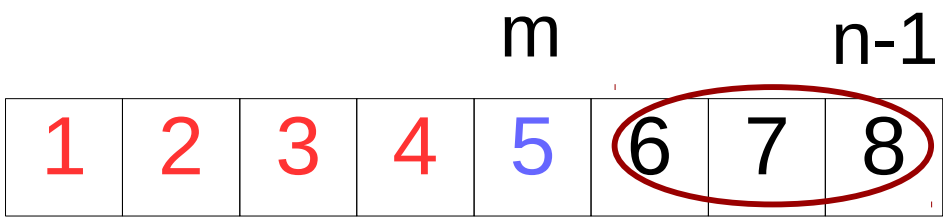
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

1. fixa primeiro elemento

Gerando permutações



Vamos escrever uma função que receba uma permutação:

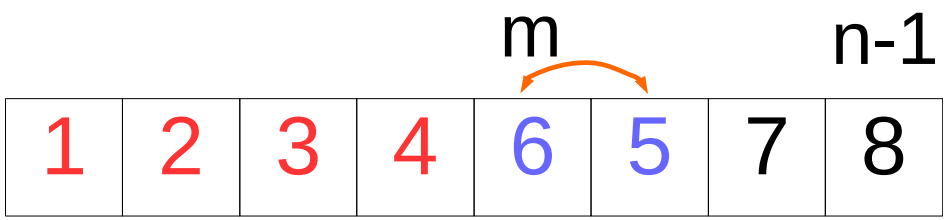
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

1. fixa primeiro elemento e permuta outros

Gerando permutações



Vamos escrever uma função que receba uma permutação:

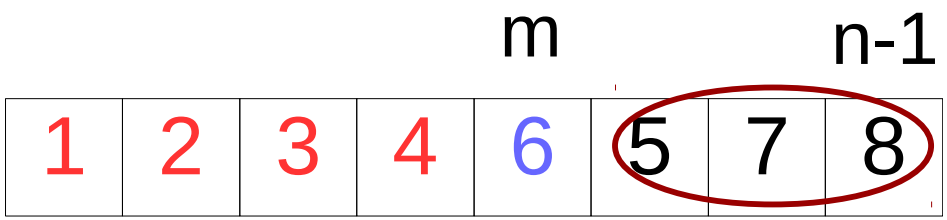
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

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

Gerando permutações



Vamos escrever uma função que receba uma permutação:

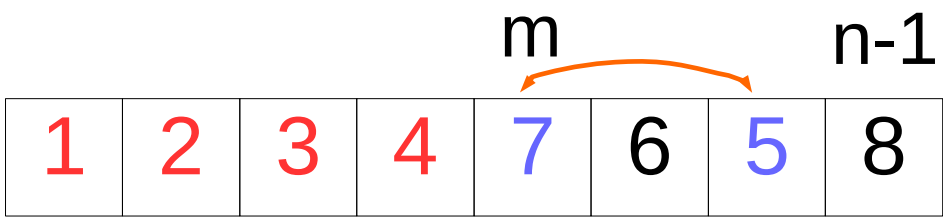
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

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:

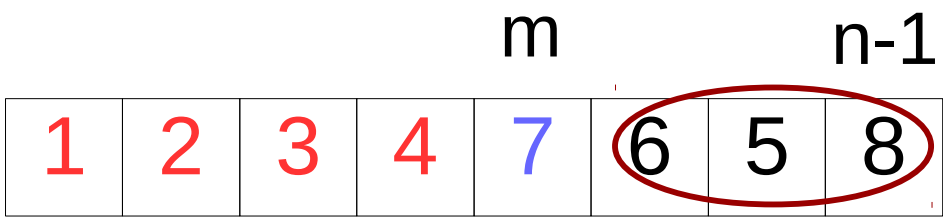
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

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:

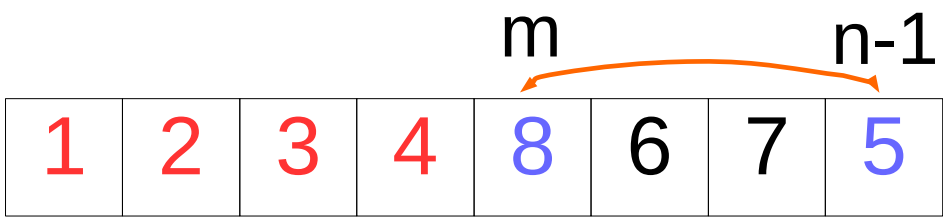
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

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:

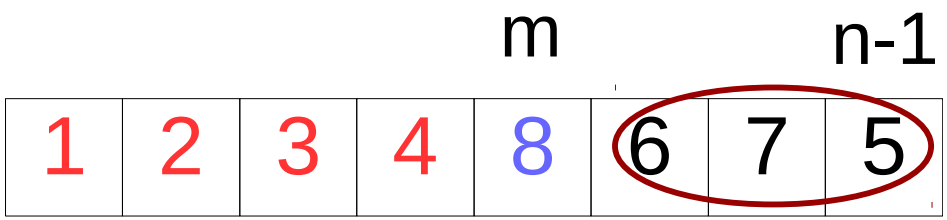
1. com valores fixos até uma posição $m - 1$
2. com posições abertas de m até n

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

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:

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

Objetivo: imprimir todas as permutações com prefixo dado.

Mesma ideia

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

Permutações:

Enumerando vetores de tamanho n

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

Programando

Permutações:

Enumerando vetores de tamanho n

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

Programando

Permutações:

Enumerando vetores de tamanho n

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

Programando

Permutações:

Enumerando vetores de tamanho n

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

Programando

Permutações:

Enumerando vetores de tamanho n

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

Programando

Permutações:

Enumerando vetores de tamanho n

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

Programando

Permutações:

Enumerando vetores de tamanho n

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

Exercício: Implementar para problema das damas usando permutação.

Retrocesso

Retrocesso

Retrocesso ou *Back-tracking* é um algoritmo genérico, com as seguintes propriedades:

- a(s) solução(ões) são construídas incrementalmente
- uma solução parcial é descartada tão logo ela se mostre inviável

Recursão mútua

Definição

Recursão mútua é a chamada recursiva indireta a uma função.

Recursão mútua

Definição

Recursão mútua é a chamada recursiva indireta a uma função.

Exemplo

```
int eh_par(int n);
int eh_impar(int n);

int eh_par(int n) {
    if (n == 0)
        return 1;
    else
        return eh_impar(n-1);
}
int eh_impar(int n) {
    if (n == 0)
        return 0;
    else
        return eh_par(n-1);
}
```

Um exemplo útil

Verificação sintática

Queremos verificar se uma expressão está bem formada de acordo com as seguinte "formas":

- expressão $\rightarrow$ termo $+$ ... $+$ termo (um ou mais termos)
- termo $\rightarrow$ fator $*$ ... $*$ fator (um ou mais fatores)
- fator $\rightarrow$ variável $|$ (expressão)
- variável $\rightarrow a \mid b \mid \ldots \mid z$ (uma letra)

Um exemplo útil

Verificação sintática

Queremos verificar se uma expressão está bem formada de acordo com as seguinte "formas":

- expressão $\rightarrow$ termo $+$ ... $+$ termo (um ou mais termos)
- termo $\rightarrow$ fator $*$ ... $*$ fator (um ou mais fatores)
- fator $\rightarrow$ variável | (expressão)
- variável $\rightarrow$ a | b | ... |z (uma letra)

Queremos criar uma função `int eh_valida(char *str)` como a seguir:

```
eh_valida("(a+b)*c+d")      // retorna 1

eh_valida("+a-c*d+")        // retorna 0

eh_valida("a+b+c@")         // retorna 0
```

Verificando expressão em C

```
int expressao(char *str, int *pos);  // tenta ler expressão
int termo(char *str, int *pos);      // tenta ler termo
int fator(char *str, int *pos);      // tenta ler fator
```

Verificando expressão em C

```
int expressao(char *str, int *pos);  // tenta ler expressão
int termo(char *str, int *pos);      // tenta ler termo
int fator(char *str, int *pos);      // tenta ler fator

int eh_valida(char *str) {
    int pos = 0;
```

Verificando expressão em C

```
int expressao(char *str, int *pos);  // tenta ler expressão
int termo(char *str, int *pos);      // tenta ler termo
int fator(char *str, int *pos);      // tenta ler fator

int eh_valida(char *str) {
    int pos = 0;

    // se conseguiu ler uma expressão
    if (expressao(str, &pos)) {
        // verifica se leu toda string
        if (str[pos] == '\0')
            return 1;
    }
```

Verificando expressão em C

```
int expressao(char *str, int *pos);  // tenta ler expressão
int termo(char *str, int *pos);      // tenta ler termo
int fator(char *str, int *pos);      // tenta ler fator

int eh_valida(char *str) {
    int pos = 0;

    // se conseguiu ler uma expressão
    if (expressao(str, &pos)) {
        // verifica se leu toda string
        if (str[pos] == '\0')
            return 1;
    }
    return 0;
}
```

Verificando expressão em C: expressão

Expressão

```
int expressao(char *str, int *pos) {
    for (;;) {
```

Verificando expressão em C: expressão

Expressão

```
int expressao(char *str, int *pos) {
    for (;;) {
        // tenta ler um termo
        if (!termo(str, pos))
            return 0;
```

Verificando expressão em C: expressão

Expressão

```
int expressao(char *str, int *pos) {
    for (;;) {
        // tenta ler um termo
        if (!termo(str, pos))
            return 0;

        // verifica se há mais termos
        if (str[*pos] == '+')
```

Verificando expressão em C: expressão

Expressão

```
int expressao(char *str, int *pos) {
    for (;;) {
        // tenta ler um termo
        if (!termo(str, pos))
            return 0;

        // verifica se há mais termos
        if (str[*pos] == '+')
            (*pos)++; // se sim, pula + e continua
```

Verificando expressão em C: expressão

Expressão

```
int expressao(char *str, int *pos) {
    for (;;) {
        // tenta ler um termo
        if (!termo(str, pos))
            return 0;

        // verifica se há mais termos
        if (str[*pos] == '+')
            (*pos)++; // se sim, pula + e continua
        else
            break;    // se não há mais termos, sai
    }
```

Verificando expressão em C: expressão

Expressão

```
int expressao(char *str, int *pos) {
    for (;;) {
        // tenta ler um termo
        if (!termo(str, pos))
            return 0;

        // verifica se há mais termos
        if (str[*pos] == '+')
            (*pos)++; // se sim, pula + e continua
        else
            break;    // se não há mais termos, sai
    }
    return 1;
}
```

Verificando expressão em C: termo

Termo

```
int termo(char *str, int *pos) {
    for (;;) {
        // tenta ler um fator
        if (!fator(str, pos))
            return 0;

        // verifica se há mais fatores
        if (str[*pos] == '*')
            (*pos)++; // se sim, pula * e continua
        else
            break;    // se não há mais fatores, sai
    }
    return 1;
}
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
    // primeiro, se próximo caractere é variável
    if (str[*pos] >= 'a' && str[*pos] <= 'z') {
        (*pos)++; // pula letra
        return 1;
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
    // primeiro, se próximo caractere é variável
    if (str[*pos] >= 'a' && str[*pos] <= 'z') {
        (*pos)++; // pula letra
        return 1;
    // senão, procura expressão entre parênteses
    } else {
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
    // primeiro, se próximo caractere é variável
    if (str[*pos] >= 'a' && str[*pos] <= 'z') {
        (*pos)++; // pula letra
        return 1;
    // senão, procura expressão entre parênteses
    } else {
        if (str[*pos] != '(') return 0;
        (*pos)++;
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
    // primeiro, se próximo caractere é variável
    if (str[*pos] >= 'a' && str[*pos] <= 'z') {
        (*pos)++; // pula letra
        return 1;
    // senão, procura expressão entre parênteses
    } else {
        if (str[*pos] != '(') return 0;
        (*pos)++;

        if (!expressao(str, pos)) return 0;
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
    // primeiro, se próximo caractere é variável
    if (str[*pos] >= 'a' && str[*pos] <= 'z') {
        (*pos)++; // pula letra
        return 1;
    // senão, procura expressão entre parênteses
    } else {
        if (str[*pos] != '(') return 0;
        (*pos)++;

        if (!expressao(str, pos)) return 0;

        if (str[*pos] != ')') return 0;
        (*pos)++;
```

Verificando expressão em C: fator

Fator

```
int fator(char *str, int *pos) {
    // primeiro, se próximo caractere é variável
    if (str[*pos] >= 'a' && str[*pos] <= 'z') {
        (*pos)++; // pula letra
        return 1;
    // senão, procura expressão entre parênteses
    } else {
        if (str[*pos] != '(') return 0;
        (*pos)++;

        if (!expressao(str, pos)) return 0;

        if (str[*pos] != ')') return 0;
        (*pos)++;
        return 1;
    }
}
```

Exercícios

1. Embora muitos algoritmos *backtracking* sejam implementados recursivamente, sabemos que esses algoritmos podem ser implementados usando uma pilha. Escreva ou esboce um programa para resolver o problema das n damas usando uma pilha Se preferir, comece com o exemplo na próxima página.
2. Modifique o programa acima, para que ele imprima todas as soluções.
3. Como você faria para adicionar a operação de exponenciação na expressão vista em sala? Como faria para calcular o valor da expressão (assumindo que as variáveis tenham um valor definido)?

Exercícios - exemplo

```
void damas(int n) {
    int *v; // pilha (guarda o vetor das
    int m;  // num de damas fixadas
    v = malloc(sizeof(int)*n);

    // inicializa primeira dama
    m = 0;
    v[m] = 0;

    while (nao fixou todas) {
        // aqui, v[m] é a última posicao testada para m

        // procura uma posicao diferente para dama m entre v[m]+1 e n

        // se encontrou posicaoviável, empilha (fixa) posicao
        // e inicializa proxima dama

        // se nao encontrou, desempilha (desafixa) posicao
    }
    // imprime tabuleiro
}
```