025089 – Projeto e Análise de Algoritmos

Aula 05

lizier@dc.ufscar.br

Hanoi – Aula anterior

$\underline{\textsc{Hanoi}(n, src, dst, tmp):}$

if $n > 0$

$\quad\quad \textsc{Hanoi}(n-1, src, tmp, dst)$

$\quad\quad$ move disk n from src to dst

$\quad\quad \textsc{Hanoi}(n-1, tmp, dst, src)$

Design de Algoritmos

- Padrões/Projeto/Design/Paradigma de algoritmos
  - Estratégias frequentemente utilizadas para resolver problemas
- Modelos:
  - **Força-bruta (busca-completa)**
  - Greedy (guloso ou ganancioso)
  - Divisão e conquista
  - Programação dinâmica
  - Transformação/Redução e conquista

Busca completa (ou força-bruta)

- Iterativa:
  - *Loops* aninhados
- Recursiva (*backtracking*):

ALGORITHM $Backtrack(X[1..i])$

//Gives a template of a generic backtracking algorithm
//Input: $X[1..i]$ specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions
if $X[1..i]$ is a solution **write** $X[1..i]$
else
    for each element $x \in S_{i+1}$ consistent with $X[1..i]$ and the constraints **do**
        $X[i+1] \leftarrow x$
        $Backtrack(X[1..i+1])$

Exemplo 1 - Iterativo

- Problema *3-sum*
  - Dado um conjunto de números, encontrar quantos subconjuntos de 3 números de soma zero existem

```
// int a[], int N

int search_3sum( ) {
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
                if (a[i] + a[j] + a[k] == 0)
                    cnt++;
    return cnt;
}
```

Exemplo 1 - Recursivo

- Problema *3-sum*
  - Dado um conjunto de números, encontrar quantos subconjuntos de 3 números de soma zero existem

```
// int a[], int N

int search_3sum( int sel, int ini, int sum ) {

    if (sel == 3 ) {
        if( sum == 0 )
            return 1;
        else
            return 0;
    } else {
        int ac = 0;
        for (int i = ini; i < N; i++)
            ac += search_3sum( a, N, sel+1, i+1, sum + a[i]);
        return ac;
    }
}
```

Busca completa (ou força-bruta)

- ***Backtracking***
  - Construir a árvore de estados (cada nó):
    - A raiz significa que nenhuma decisão foi tomada (estado inicial)
    - Os filhos de cada nó são as possibilidades de escolhas
    - Cada nó pode ser classificado como
      - Promissor: com real possibilidade de formar uma solução
      - Não-promissor: sem possibilidade de ser parte de uma solução
      - Solução: critério de parada da recursão

ALGORITHM *Backtrack*$(X[1..i])$

//Gives a template of a generic backtracking algorithm
//Input: $X[1..i]$ specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions
if $X[1..i]$ is a solution **write** $X[1..i]$
else
    for each element $x \in S_{i+1}$ consistent with $X[1..i]$ and the constraints **do**
        $X[i+1] \leftarrow x$
        Backtrack$(X[1..i+1])$

Busca completa (ou força-bruta)

- ***Backtracking***
- Estrutura:
  - Uma condição que verifica se uma solução foi encontrada;
  - Um laço que tenta todos os valores possíveis para uma única variável discreta;
  - Uma recursão que irá investigar se existe uma solução segundo os valores atribuídos às variáveis discretas até o momento.

ALGORITHM *Backtrack*$(X[1..i])$

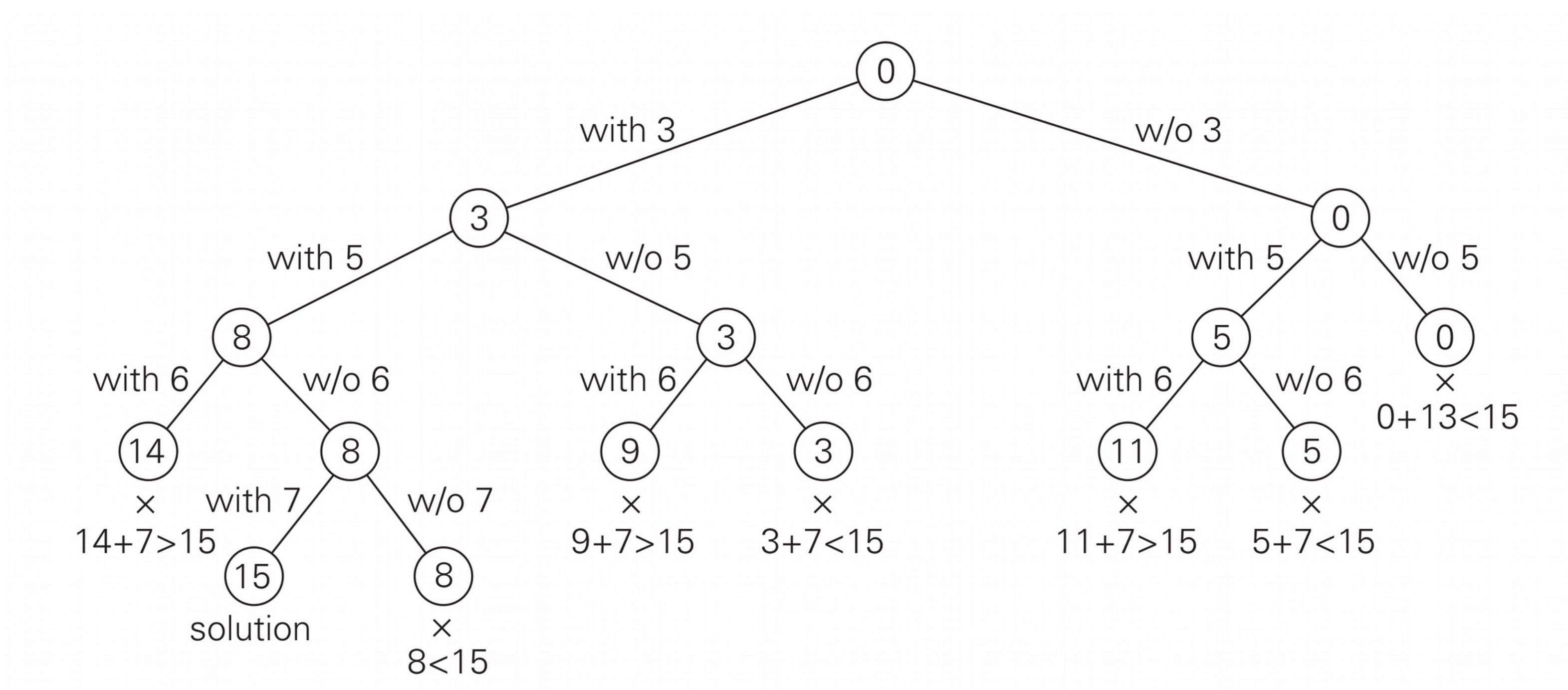
//Gives a template of a generic backtracking algorithm
//Input: $X[1..i]$ specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions
if $X[1..i]$ is a solution **write** $X[1..i]$
else
  for each element $x \in S_{i+1}$ consistent with $X[1..i]$ and the constraints **do**
    $X[i+1] \leftarrow x$
    Backtrack$(X[1..i+1])$

Exemplo 2

- Problema *Subset-Sum*
  - Dado um conjunto de ***N*** números positivos, encontrar quantos subconjuntos de soma ***k*** existem
  - Exemplo:
    - A={1, 2, 5, 6, 8}
    - K=9
    - Solução: {1, 2, 6} e {1, 8}

Exemplo 2

- *Problema Subset-Sum*
- Exemplo: A = { 3, 5, 6, 7}; d = 15
  - *State-Space-Tree*



Exemplo 2

- *Problema Subset-Sum*

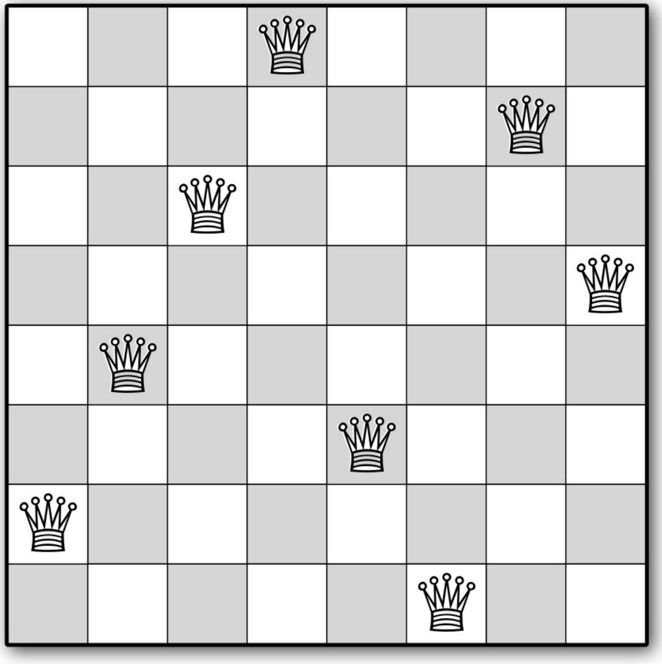
```
// int a[], int N, int s[], int K

void printSolution( int sel ) {
    cout << s[0];
    for(int i=1; i<sel; i++)
        cout << " " << s[i];
    cout << endl;
}

void subsetsum( int index, int sel, int sum ) {
    if( sum == K )
        printSolution( sel );
    else if( index == N )
        return;
    else {
        subsetsum( index+1, sel, sum);
        s[sel] = a[index];
        subsetsum( index+1, sel+1, sum+a[index]);
    }
}
```

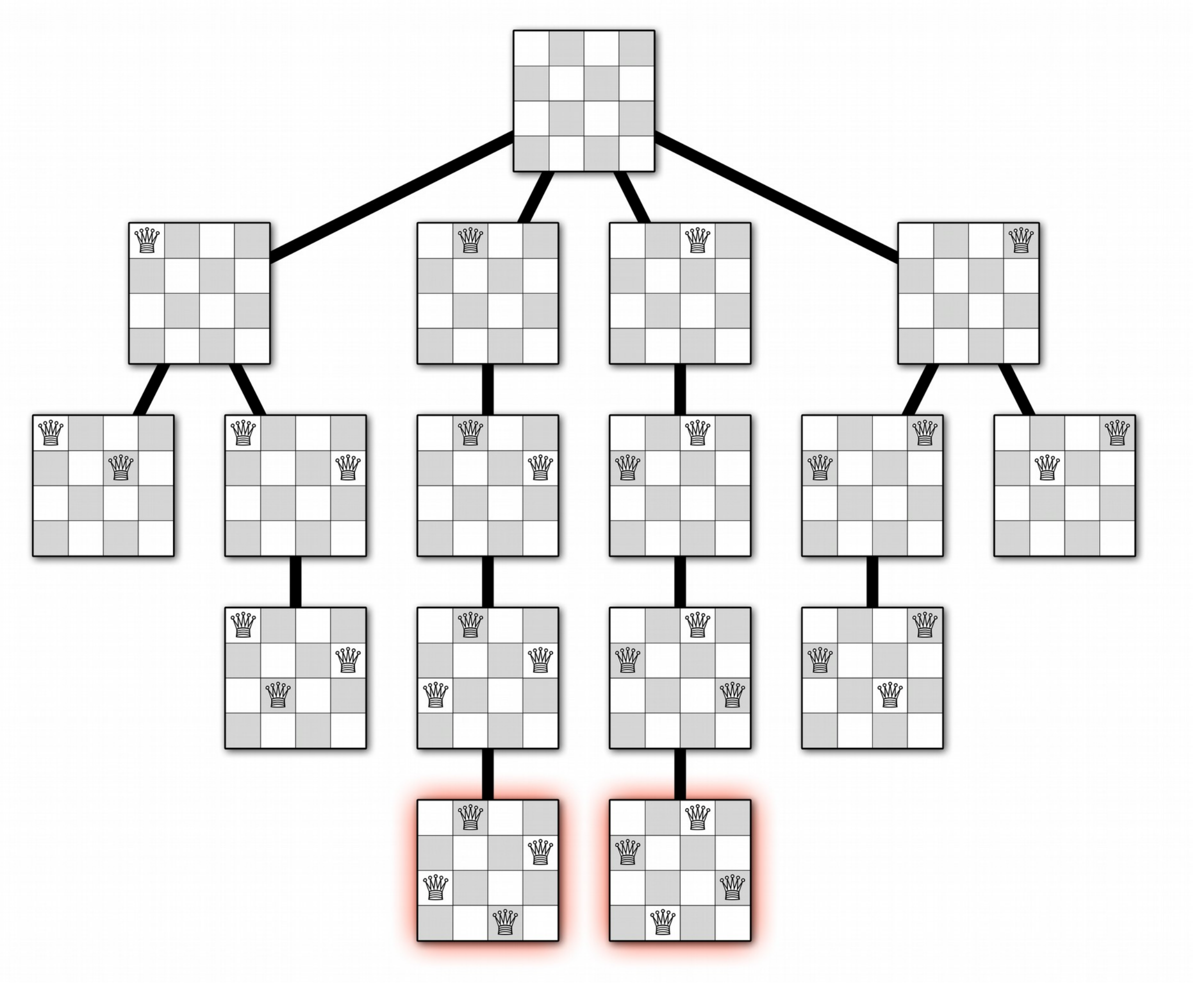
Exemplo 3

- Problema *n-queens*
  - Colocar *n* rainhas em um tabuleiro *nxn*, não permitindo que uma rainha possa atacar outra



Exemplo 3

- Problema *n-queens*



Exemplo 3

- Problema *n-queens*

```
RecursiveNQueens(Q[1..n], r):
    if r = n + 1
        print Q
    else
        for j ← 1 to n
            legal ← True
            for i ← 1 to r − 1
                if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
                    legal ← False
            if legal
                Q[r] ← j
                RecursiveNQueens(Q[1..n], r + 1)
```