Centro de
Investigación en
Computación

Recursion

Recursividad

Teoría y ejemplos

Course

Analysis and design of algorithms

Instructor

Acosta Bermejo Raúl

Lecture notes

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

1. Introduction
2. Definitions
3. Types
4. Backtracking
5. Bibliography

Introduction

Introducción

Recursion in computer language is a process in which a function calls itself directly or indirectly.

- Implementation

  When a procedure is called, its information is pushed onto a stack, and when the function terminates the information is popped out of the stack.

  This can be modeles as a state: parameters and local variables.

- Struct of function

  1. Break condition

     Setting boundary conditions: It is necessary to set boundary conditions to ensures that the recursive function comes to an end.

  2. Call, Invocation, ... Of the same procedure / method / function

     But this is not always as simple ...

Introduction

Introducción

Types of recursion

1. **Direct recursion**

   If a function calls itself, it's known as **direct recursion**. This results in a one-step recursive call: the function makes a recursive call inside its own function body.

```
void directRecursionFunction() {
    // some code...
    directRecursionFunction();
    // some code...
}
```

2. **Indirect recursion**

   If the function f1 calls another function f2 and f2 calls f1 then it is indirect recursion (or mutual recursion). This is a two-step recursive call: the function calls another function to make a recursive call.

```
void indirectRecursionFunctionf1()
{
    // some code...
    indirectRecursionFunctionf2();
    // some code...
}

void indirectRecursionFunctionf2 ()
{
    // some code...
    indirectRecursionFunctionf1 ();
    // some code...
}
```

Introduction

Introducción

Tail recursive

A recursive function is tail recursive when a recursive call is the last thing executed by the function.

```
void print(int n)
{
    if (n < 0)  return;
    cout << " " << n;

    // The last executed statement is recursive call
    print(n-1);
}
```

For the non-tail-recursive functions, the stack depth (maximum amount of stack space used at any time during compilation) is more.

Introduction

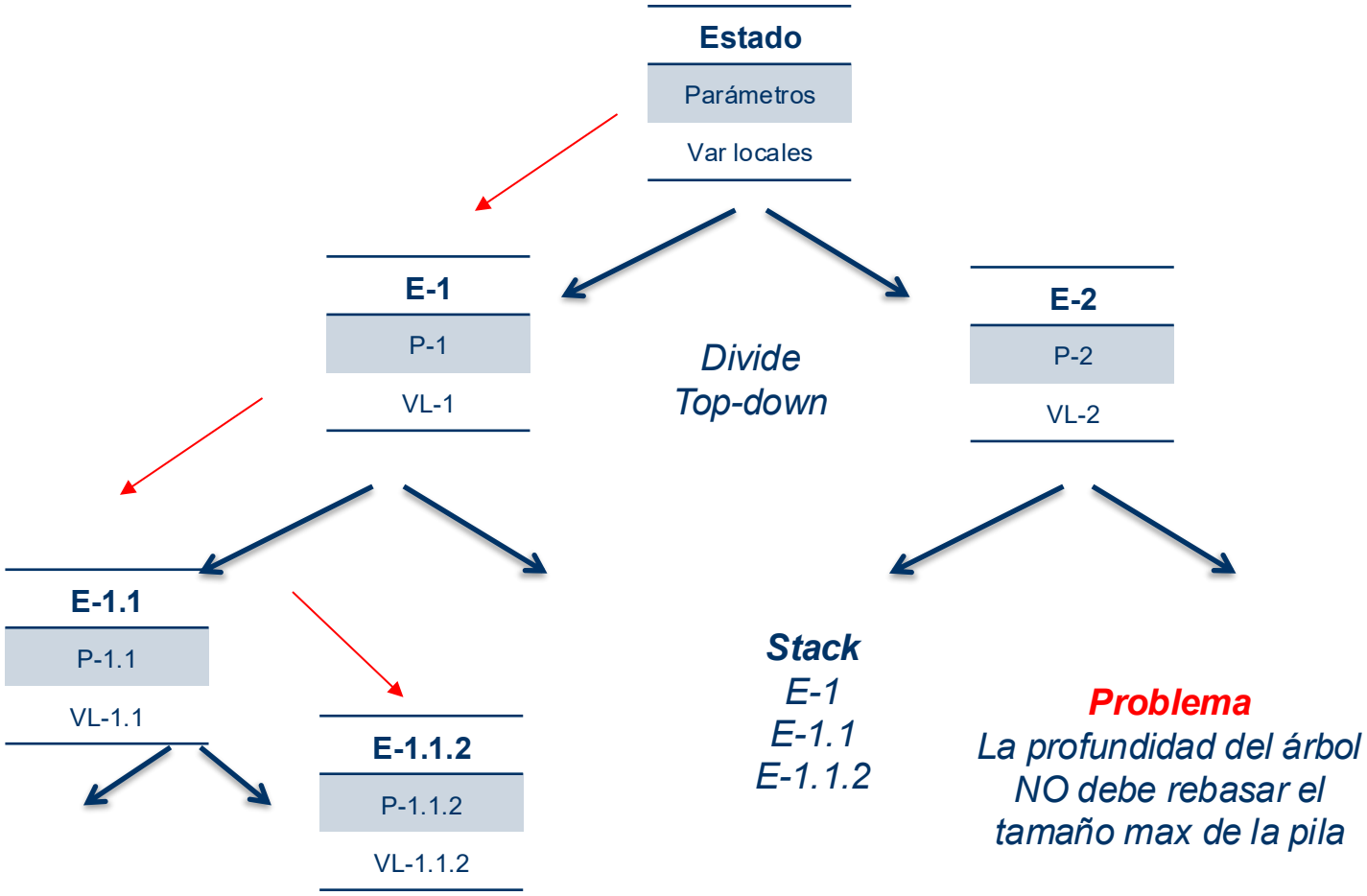
Introducción

Tail recursive

- In the case of tail recursion, it can be optimizes so that only one stack entry is used for all the recursive calls of the function.
- This means that even on large inputs, there can be no stack overflow. This is called Tail recursion optimization.
- Languages such as lisp and c/c++ have this sort of optimization.
- But, the Python interpreter doesn't perform tail recursion optimization. Due to this, the recursion limit of python is usually set to a small value (approx, 10^4).
  - The "sys" module in Python provides a function called **setrecursionlimit()** to modify the recursion limit in Python.

Introduction

Introducción



Centro de
Investigación en
Computación

Backtracking

Vuelta atrás / retroceso

Teoría y ejemplos

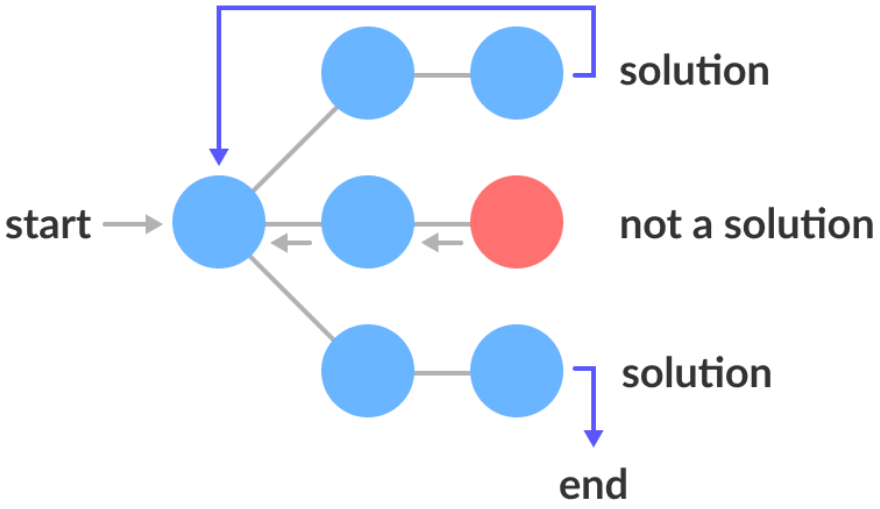
Instituto
Politécnico
Nacional

Backtracking

Introducción

Es una técnica de tipo Fuerza Bruta en la cual se explora todo el espacio de soluciones buscando la solución.

La exploración se puede representar como un grafo y en particular como un árbol:



Backtracking

Problemas

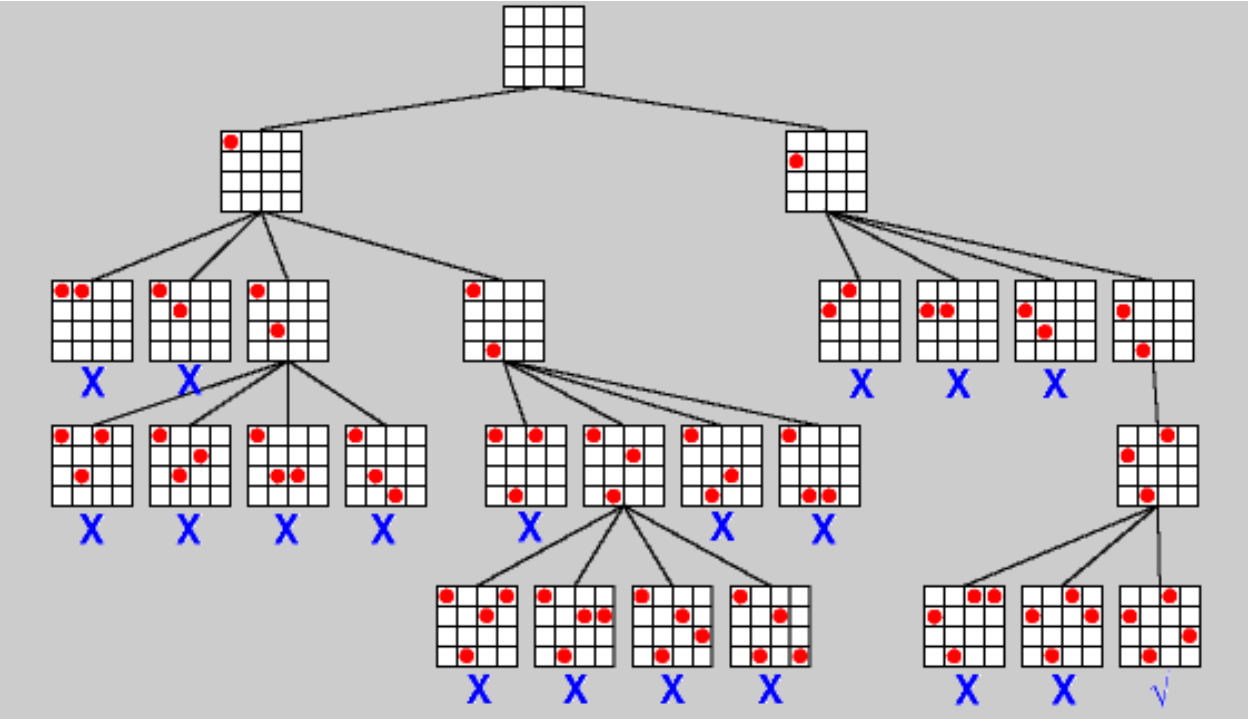
Problemas clásicos

1. Laberintos (maze)
2. Varios juegos como
    1. Ajedrez
3. Las ocho reinas
4. Los movimientos de un caballo.
5. Sudoku

Backtracking

Problemas

Tablero con piezas

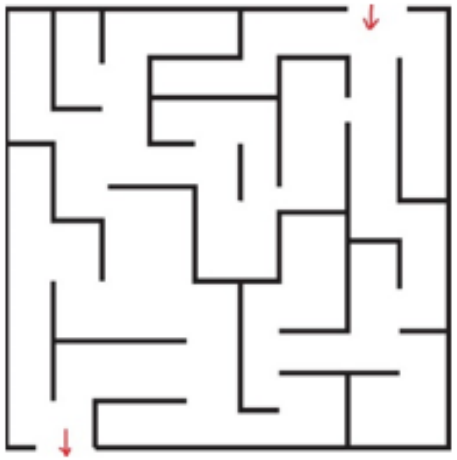


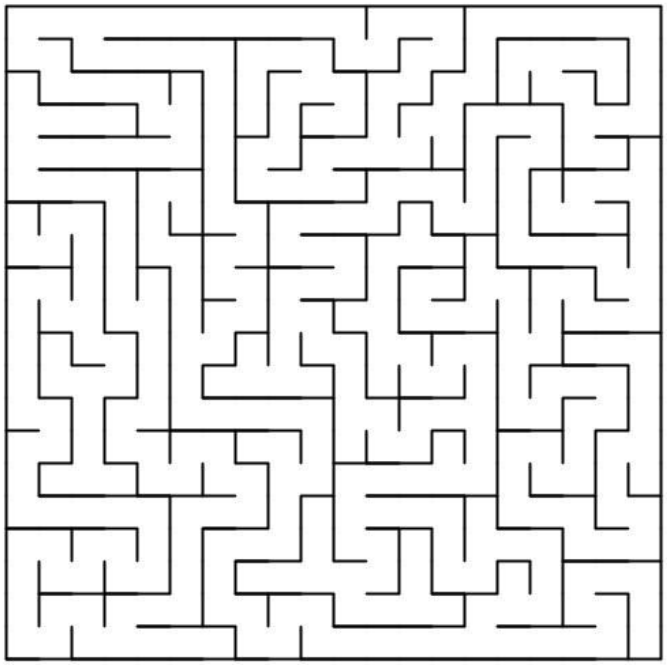
Backtracking

Problemas

Laberintos

1	0	0	1	1	1	0
1	0	0	1	0	1	0
1	1	1	1	0	1	0
0	1	0	0	0	1	0
0	1	0	1	1	1	0
0	1	0	1	0	0	0
1	1	0	1	1	1	1



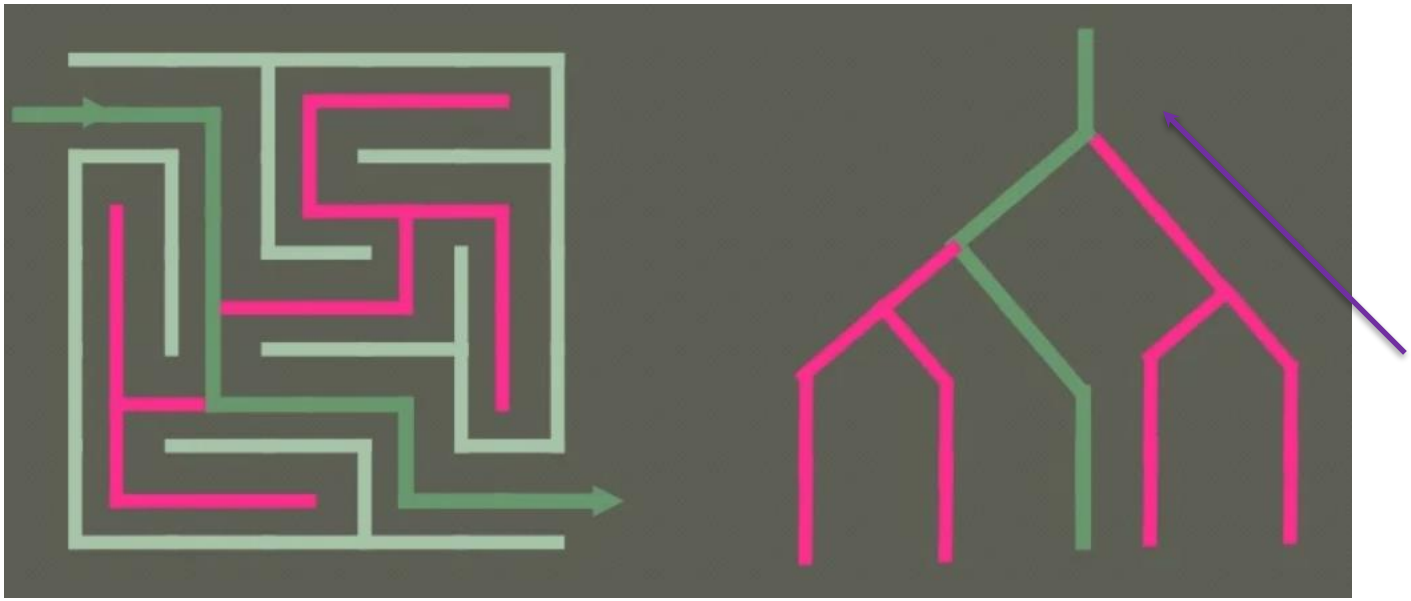


Backtracking

Problemas

Laberintos

Se retrocede
Si no hay éxito



Backtracking

Problemas

Pos
maze(matriz, coord_ini-L)
maze(matriz, coord_ini-R);
maze(matriz, coord_ini-U)
maze(matriz, coord_ini-D);

maze(matriz, (0,0), (10,10));

Implementación

Variantes:

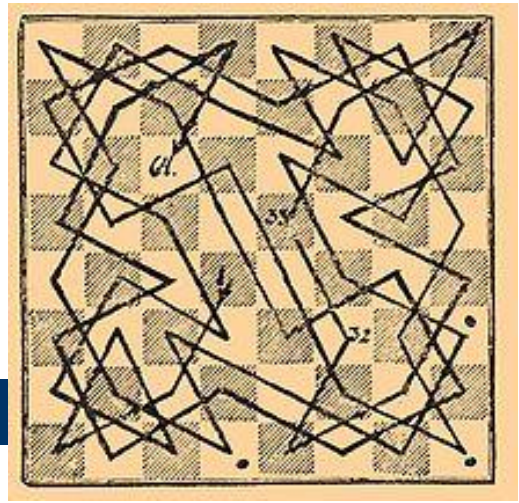
1. Si existe un camino o la ubicación de la salida.
2. Una secuencia de pasos (camino).
3. Todos los posibles caminos

O ya se sabe que se busca Valor

```
boolean maze(matriz, coord_ini, coord_fin){
    if( ya paso por la celda OR se sale tablero OR es muro OR…)
        return false
    if( matriz[coord_ini]==Valor OR coord_ini == coord_fin OR …)
        return true
     matriz[coord_ini]=marca
    if( maze(matriz, coord_ini-Pos) )
        return True
    else if( maze(matriz, coord_ini-Pos) )
}
```

Backtracking

Problemas



Descripción

Teniendo una cuadrícula de n x n casillas y un caballo de ajedrez colocado en una posición cualquiera (x, y), el caballo pase por todas las casillas y una sola vez.

El problema es una forma específica del problema más general conocido como la ruta Hamiltoniana en la teoría de grafos.

Ejercicio

- Encontrar una solución iterativa o recursiva.

Referencias

- https://es.wikipedia.org/wiki/Problema_del_caballo

Bibliography

Bibliografía

URL de varios temas:

- Geeks for Geeks
  - https://www.geeksforgeeks.org/tail-recursion/.
- Maze solving algorithm
  - https://en.wikipedia.org/wiki/Maze-solving_algorithm
- Path Finder algorithms
  - https://emmilco.github.io/path_finder/
- Maze generation algorithm
  - https://en.wikipedia.org/wiki/Maze_generation_algorithm

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652