Resolución de Problemas y Algoritmos

clase 17: resolución de problemas utilizando recursión

Dr. Alejandro J. García
e-mail: agarcia@cs.uns.edu.ar
http:// cs.uns.edu.ar /~ajg

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

Temas vistos

Los siguientes temas, vistos en clases anteriores, están todos relacionados y son muy importantes:

- **Diseño de la solución dividiendo el problema**
- **Funciones y procedimientos en Pascal**
- **Parámetros (por valor y por referencia)**
- **Entorno de referencia de los identificadores**
- **Visibilidad – Identificadores locales, globales, etc.**

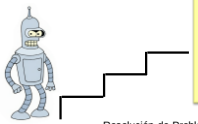
¿Alguna pregunta?

Esta importancia va más allá de RPA, en su vida profesional es muy probable que trabaje en un equipo y entonces...

Soluciones recursivas

Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una de *percepción* y otra de *efecto*):

- **quedan-escalones: función que retorna TRUE si hay escalones al frente, o FALSE en caso contrario.**
- **subir-escalón: hace al robot subir un escalón.**

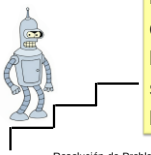


En esta situación:
quedan-escalones retorna TRUE
Por lo tanto si ejecuto:
subir-escalón
La situación cambiará a la siguiente:

Soluciones recursivas

Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una de *percepción* y otra de *efecto*):

- **quedan-escalones: función que retorna TRUE si hay escalones al frente, o FALSE en caso contrario.**
- **subir-escalón: hace al robot subir un escalón.**

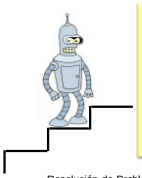


En esta nueva situación:
quedan-escalones retorna TRUE
Por lo tanto si ejecuto:
subir-escalón
La situación cambiará a la siguiente:

Soluciones recursivas

Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una de *percepción* y otra de *efecto*):

- **quedan-escalones: función que retorna TRUE si hay escalones al frente, o FALSE en caso contrario.**
- **subir-escalón: hace al robot subir un escalón.**

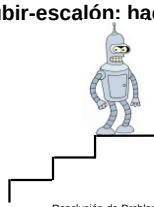


En esta nueva situación:
quedan-escalones retorna TRUE
Por lo tanto si ejecuto:
subir-escalón
La situación cambiará a la siguiente:

Soluciones recursivas

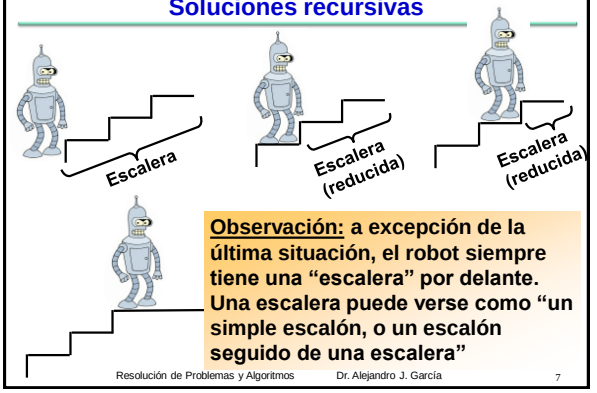
Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una de *percepción* y otra de *efecto*):

- **quedan-escalones: función que retorna TRUE si hay escalones al frente, o FALSE en caso contrario.**
- **subir-escalón: hace al robot subir un escalón.**



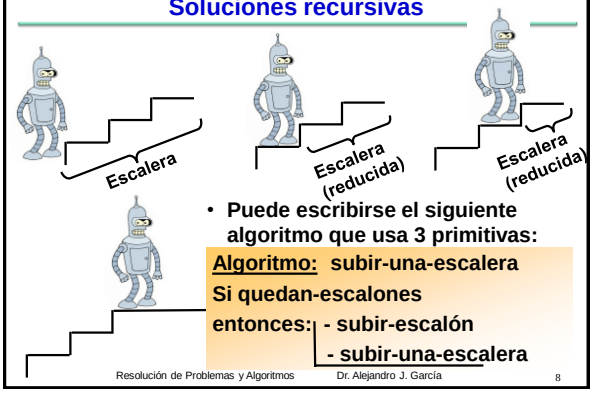
En esta nueva situación:
quedan-escalones retorna FALSE
Y el robot ¡ ya subió la escalera!

Soluciones recursivas



Observación: a excepción de la última situación, el robot siempre tiene una "escalera" por delante. Una escalera puede verse como "un simple escalón, o un escalón seguido de una escalera"

Soluciones recursivas



- Puede escribirse el siguiente algoritmo que usa 3 primitivas:

Algoritmo: subir-una-escalera
Si quedan-escalones
entonces: - subir-escalón
- subir-una-escalera

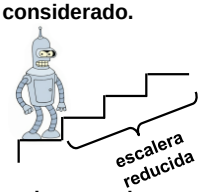
Algoritmos recursivos

- **Recursión** es la forma en la cual se especifica un proceso **basado en su propia definición**.
- La **solución** de un problema es **recursiva**, si el proceso que describe utiliza recursión.
- Un **algoritmo** es **recursivo** si se define en términos de sí mismo.
- Un algoritmo no debe entrar en un ciclo infinito, por ende, un algoritmo recursivo **será válido**, si:
  (a) **existe un caso base** que **no** se define en términos de si mismo, y
  (b) la **referencia a sí mismo es relativamente más sencilla (o reducida)** que el caso considerado.

Algoritmos recursivos

- Un algoritmo recursivo **será válido**, si:
  (a) **existe un caso base** que **no** se define en términos de si mismo, y
  (b) la **referencia a sí mismo es relativamente más sencilla o reducida** que el caso considerado.

Ejemplo: subir una escalera



(a) Caso base: no hay escalones
(b) Caso general:
- sube un escalón
- y luego queda por subir una escalera que tiene un escalón menos, y por lo tanto es más reducida.

Algoritmos recursivos

Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una de *percepción* y otra de *efecto*):

- quedan-escalones: retorna verdadero o falso
- subir-escalón: hace al robot subir un escalón

El siguiente algoritmo es **incorrecto** ¿Por qué?

Algoritmo 2 subir-escalera
-subir-escalón
-subir-escalera

MAL

Falla porque no hay indicado un **caso base** (a).

Algoritmos recursivos

El Algoritmo 3 también es **incorrecto** ¿Por qué?

Algoritmo 3 subir-la-escalera
Si quedan-escalones
entonces: - subir-la-escalera
- subir-escalón

MAL

En este caso falla (b) porque la **referencia a sí mismo NO es relativamente más sencilla o reducida (es igual)**

Soluciones recursivas

- Dado un problema P, la solución puede realizarse en forma recursiva cuando:
  - Hay un caso base, esto es una solución para un caso trivial del problema que no se define en términos de si misma.
  - En el caso general, la solución de cualquier instancia del problema P (excepto la trivial) requiere resolver instancias más simples o pequeñas del mismo.
- De esta forma, al utilizar una solución recursiva, el problema queda divido en dos subproblemas: (1) caso base y (2) caso general.

Otros algoritmos recursivos

- Generalmente los trabajos de impresión (jobs) son enviados a una cola de impresión (queue) antes de ser impresos. Se puede escribir un algoritmo que se encargue de imprimir los trabajos pendientes de la siguiente manera:

```
Algoritmo imprimir-trabajos
Si quedan-trabajos-en-cola
  entonces: - sacar-un-trabajo(T)
            - imprimir-un-trabajo(T)
            - imprimir-trabajos
```


Otros algoritmos recursivos

Para comer un plato de ravioles:

```
Algoritmo comer-ravioles
Si quedan-ravioles
entonces: - comer-un-raviol
          - comer-ravioles
```

Para tomar un vaso de una bebida:

```
Algoritmo tomar-bebida
Si queda-líquido
entonces: - tomar-un-sorbo
          - tomar-bebida
```


Problema propuesto: 2^N

- Para N no negativo, la función 2^N puede definirse recursivamente de la siguiente manera:

$$2^N \begin{cases} 2^0 = 1 \quad \text{(para N=0)} \\ 2^N = 2 * 2^{N-1} \quad \text{para N>0} \end{cases}$$

Observe que está dividido en 2 casos.

Planteo recursivo para 2^N

- Caso base o trivial: si N=0 entonces 2^N es 1
- Caso general o caso recursivo: Si N>0 entonces 2^N es $2 * 2^{N-1}$


```
program DosAlaNRecursivo;
var  num, resu: integer;
     FUNCTION  dosAlaN (N:integer):integer;
     var aux,nuevoN:integer;
       begin
        if (N = 0) then dosAlaN :=1
        else begin
            nuevoN:=N-1;
            aux:= dosAlaN(nuevoN);
            dosAlaN:=2 * aux;
           end;
       end;
begin
writeln(' Ingrese un número'); Readln(Num);
resu:=dosAlaN(Num); writeln(' 2 a la ', Num,' es ' , resu );
end.
```

Realice la traza para num=0 y num=3. ¿Cuántas veces se llama a la dosAlaN con respecto al valor de num?


```
program DosAlaNRecursivo;
var  num, resu: integer;
     FUNCTION  dosAlaN (N:integer):integer;
     {esta otra versión no usa variables auxiliares}
     begin
       if (N = 0) then dosAlaN :=1
                  else dosAlaN := 2 * dosAlaN(N-1);
     end;

begin
writeln(' Ingrese un número'); Readln(Num);
resu:=dosAlaN(Num);
writeln(' 2 a la ', Num,' es ' , resu );
end.
```

Realice la traza.

Problema propuesto

Escriba un programa que permita ingresar por teclado una secuencia de caracteres terminada en un punto (por ejemplo: "hola que tal.") y que la muestre por pantalla en orden inverso ("lat euq aloh"). Ejemplo:

Ingrese una cadena terminada en punto: un animal.
Invertida queda así: lamina nu

Planteo Recursivo para MostrarInvertida una secuencia S

caso base: si la secuencia S es solamente un "." mostrar el cartel "Invertida queda así:"

caso general: MostrarInvertida la secuencia S sin su primer elemento, y luego mostrar el primer elemento de S.


```
program Invertir_Recursivo;
    Procedure MostrarInvertida;
    {observe que no tiene parámetros}
    var caract: char;
    begin
      read(caract);
       IF caract = '.' THEN write(' Invertida queda así: ')
       ELSE begin
                MostrarInvertida;  {llamada recursiva}
                write(caract);
               end;
    end;

begin
writeln(' Ingrese una cadena terminada en punto: ');
MostrarInvertida;
end.
```