Resolución de Problemas y Algoritmos

Clase 15:
Resolución de problemas utilizando recursión

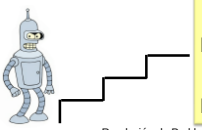
Dr. Alejandro J. García
http://cs.uns.edu.ar/~ajg

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

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.**

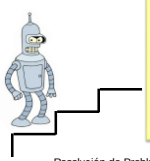


En la situación representada por el dibujo de la izquierda:
quedan-escalones retorna TRUE
Por lo tanto si ejecuto:
subir-escalón
La situación cambiará a la siguiente:

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.**

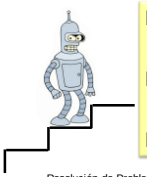


En esta nueva situación:
quedan-escalones retorna TRUE
Por lo tanto si ejecuto:
subir-escalón
La situación cambiará a la siguiente:

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:

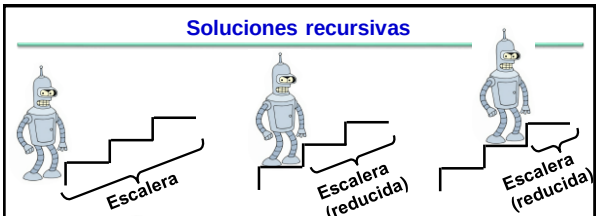
Soluciones recursivas

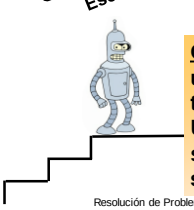
Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una de *percepción* y otra de *efecto*):

- **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!

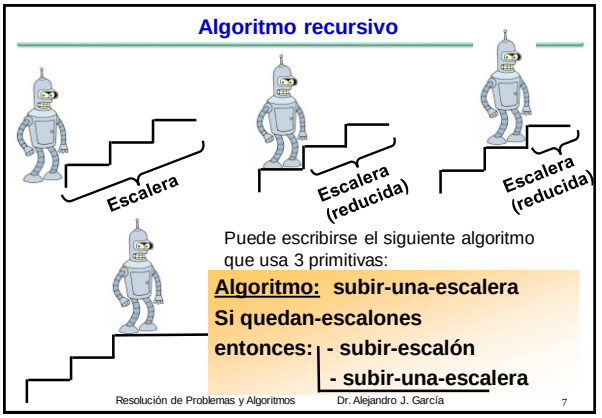
Soluciones recursivas





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"

Algoritmo recursivo



Puede escribirse el siguiente algoritmo que usa 3 primitivas:

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

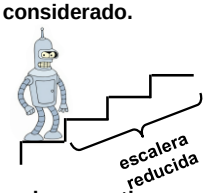
Conceptos: algoritmos recursivos

- **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.

Planteo: 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)

Otros algoritmos recursivos

Se quiere controlar un conjunto de solicitudes (de beca, o pase de comisión, etc) con las siguientes primitivas:

- **Procesar-solicitud: saca la solicitud del conjunto y realiza el control correspondiente**
- **quedan-solicitudes retorna verdadero o falso**

Algoritmo: controlar solicitudes
Si quedan-solicitudes
entonces: - procesar-solicitud
- controlar solicitudes

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-una porción
          - comer-ravioles
```

Para tomar un vaso de una bebida:

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


Soluciones recursivas en Pascal

- Los procedimientos y funciones de Pascal nos permitirán implementar soluciones recursivas.
- A continuación, para mostrar como hacer funciones y procedimientos recursivos, vamos a usar ejemplos más simples que programar robots o procesar una cola de impresión.
- Como verá en otras materias y en su vida profesional, "recursión" es una herramienta mucho más general y poderosa que la "iteración". Verá por ejemplo que hay lenguajes de programación que solo tienen recursión.

Problema propuesto: 2^N

Escribir una función recursiva en Pascal para computar la función 2^N (N no negativo).

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 ya 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 $2 * 2^{N-1}$


```
function dosAlaN (N:integer):integer;
var aux,nuevoN:integer;
 begin {función recursiva para 2 a la N}
  if (N = 0) then dosAlaN :=1 {caso base}
  else begin {caso general}
      nuevoN:=N-1;
      aux:= dosAlaN(nuevoN);
      dosAlaN:=2 * aux;
    end;
end;
```

Esta es una posible función en Pascal que respeta el planteo (no es la única).

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; {Programa de prueba para 2 a la N}
var  exp, resu: integer;
   function  dosAlaN (N:integer):integer;
   var aux,nuevoN:integer;
     begin {función recursiva para 2 a la N}
      if (N = 0) then dosAlaN :=1 {caso base}
      else begin  {caso general}
           nuevoN:=N-1;
           aux:= dosAlaN(nuevoN);
           dosAlaN:=2 * aux;
          end;
   end;
{El programa valida la entrada y llama a la función recursiva}
begin
writeln(' Ingrese un exponente >= 0');
repeat readln(exp) until exp>=0;
resu:=dosAlaN(exp); writeln(' 2 a la ',exp,' es ' , resu );
end.
```

Realice trazas para exp=0 y exp=3.

¿Cuántas veces se llama a la función dosAlaN con respecto al valor de exp?


```
function  dosAlaN (N:integer):integer;
{Otra versión de una función recursiva para
2 a la N (que no usa variables auxiliares)}
begin
  if (N = 0)   then dosAlaN :=1 {caso base}
  else dosAlaN := 2 * dosAlaN(N-1); {c. general}
end;
```

Esta es otra forma de implementar en Pascal la función recursiva que respeta el mismo planteo recursivo.

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 DosAlaNRecursivo2; {Prueba otra versión de 2 a la N}
var  exp, resu: integer;

function  dosAlaN (N:integer):integer;
{Otra versión de una función recursiva para
2 a la N (que no usa variables auxiliares)}
begin
  if (N = 0) then dosAlaN :=1 {caso base}
  else dosAlaN := 2 * dosAlaN(N-1); {general}
end;

{El programa valida la entrada y llama a la función recursiva}
begin
writeln(' Ingrese un exponente >= 0');
repeat readln(exp) until exp>=0;
resu:=dosAlaN(exp); writeln(' 2 a la ',exp,' es ' , resu );
end.
```

Realice trazas para exp=0 y exp=3.

¿Cuántas veces se llama a la función dosAlaN con respecto al valor de exp?

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.

Continuará …