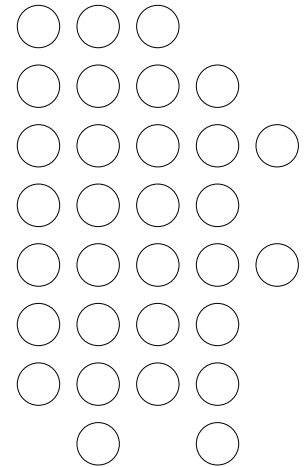


Técnicas de prueba y corrección de algoritmos

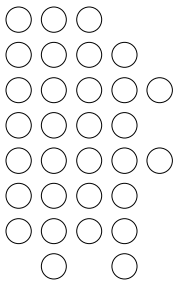


UNIVERSIDAD
DE LOS ANDES

Diseño y Análisis de Algoritmos
Cátedra de Programación
Carrera de Ingeniería de Sistemas
Prof. Isabel Besembel Carrera

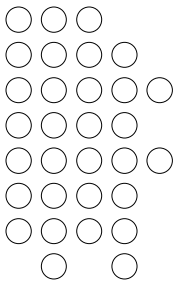


Algoritmos recursivos



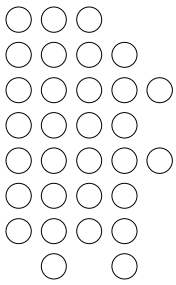
- Aparente circularidad del método: para resolver un problema, primero resuelva el problema!!!!
- Correcta interpretación del método: una instancia se resuelve mediante la solución de una o más instancias diferentes y más pequeñas
- Es un algoritmo recursivo si su ejecución provoca una o varias llamadas a él mismo.
- Recursividad simple: donde todas las llamadas recursivas se realizan dentro del mismo algoritmo

Algoritmos recursivos



- Recursividad cruzada: cuando las llamadas recursivas se realizan del algoritmo recursivo R al algoritmo recursivo V y viceversa
- Para probar que un algoritmo recursivo da la solución correcta a alguna instancia, se asume que da una solución correcta a las instancias más pequeñas y esto sugiere inmediatamente, que se debe usar inducción sobre el tamaño de las instancias para probar que el algoritmo es correcto.

Algoritmos recursivos

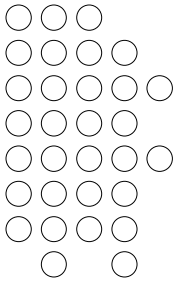


Ejemplo 1: calcular el factorial de un número n

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n * n - 1 & \text{si } n > 0 \end{cases}$$

Marzo/05		factorial(Entero: n):Entero
{ pre: $n \geq 0$ }		{ pos: $x = n!$ }
1	$x =$ Si ($n = 0$) entonces 1 sino $n * \text{factorial}(n-1)$ fsi	x: Entero. Valor del factorial de n
2	regrese f	
1	$n = 0 \Rightarrow x = 1$	
2	$n = 3 \Rightarrow x = 6$	

Algoritmos recursivos



➤ Prueba de corrección del ejemplo 1

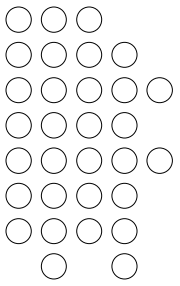
Teorema: Para todo entero $n \geq 0$, $\text{factorial}(n)$ regresa $n!$

Prueba: por inducción sobre n

Etapa base: $n = 0$. La condición $n = 0$ es cierta y el algoritmo regresa 1. Es correcto ya que $0! = 1$

Etapa inductiva: Hipótesis inductiva: $\text{factorial}(j)$ regresa $j!$, para todo j en el rango $0 \leq j \leq n-1$.

Algoritmos recursivos



Se debe demostrar que $\text{factorial}(n)$ regresa $n!$

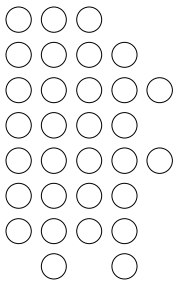
Si $n > 0$, la prueba de la condición $n = 0$ falla y

$x = n * \text{factorial}(n-1)$.

Por la hipótesis inductiva, $\text{factorial}(n-1)$ regresa $(n-1)!$, por lo cual $\text{factorial}(n)$ regresa $n \times (n-1)!$, lo que es igual a $n!$

- La prueba es posible solamente porque la llamada recursiva se hace sobre una instancia más pequeña, y por ello la hipótesis inductiva puede aplicarse

Algoritmos recursivos



➤ Prueba de corrección del ejemplo 2

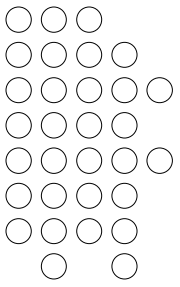
Teorema: Para todo $n \geq 0$, donde $n = b - a + 1$ es el número total de elementos en el vector $A[a..b]$,

$\text{busquedaBinaria}(A, a, b, x)$ regresa correctamente el valor de la condición $x \in A[a..b]$

Prueba: por inducción sobre n (el tamaño del arreglo)

Etapa base: $n = 0$. El vector está vacío, $a = b + 1$, la prueba $a > b$ es exitosa y el algoritmo regresa falso. Es correcto, ya que x no puede estar en el vector vacío

Algoritmos recursivos



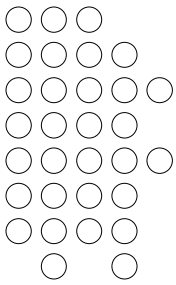
Etapa inductiva: $n > 0$. Hipótesis inductiva: $\forall j, 0 \leq j \leq n-1$, donde $j = b' - a' + 1$, `busquedaBinaria(A, a, b, x)` regresa correctamente la condición $x \in A[a..b]$. Por la asignación $m = (a+b)/2$ se concluye que $a \leq m \leq b$.

Si $x = A[m]$, claramente $x \in A[a..b]$ y el algoritmo regresa el valor correcto **verdadero**.

Si $x < A[m]$, como A está ordenado se concluye que $x \in A[a..b]$ si y solo si $x \in A[a..m-1]$. Por la hipótesis inductiva esta segunda condición la regresa `busquedaBinaria(A, a, m-1, x)`. Se aplica la hipótesis inductiva ya que $0 \leq (m-1) - a + 1 \leq n-1$.

El caso $x > A[m]$ es similar y se concluye que el algoritmo calcula correctamente todas las instancias de tamaño n

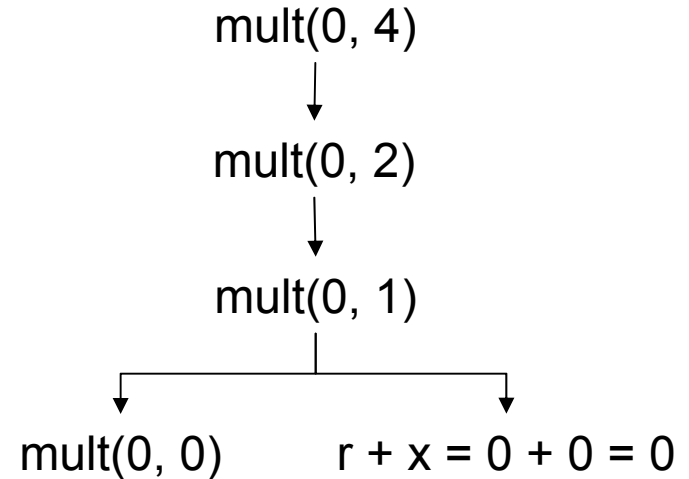
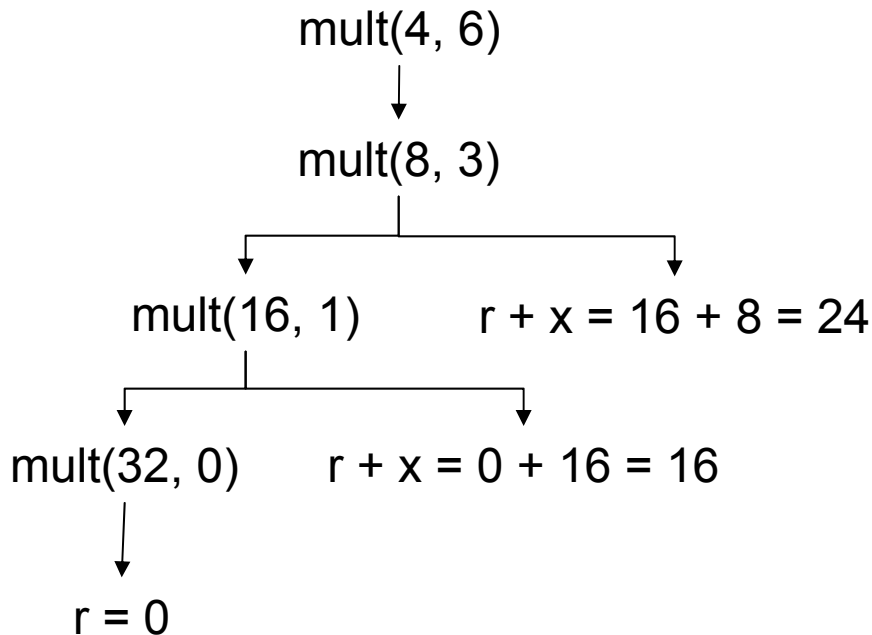
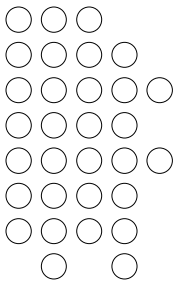
Algoritmos recursivos



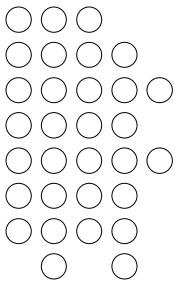
- Árboles de ejecución
 - ❖ Raíz: algoritmo recursivo etiquetado con la instancia inicial del problema
 - ❖ Nodos: etiquetados con las instancias de las llamadas realizadas

Marzo/05		
mult(Entero+: x, y):Entero		
{ pre: x, y ≥ 0 }		{ pos: r = xy ≥ 0 }
1	r = Si (y = 0) entonces 0 sino si (y es par) entonces mult(2x, y/2) sino mult(2x, y/2) r + x fsi	r: Entero+. Resultado de la multiplicación de x por y.
2	regrese r	
1	x = 0, y = 4 => r = 0	
2	x = 3, y = 0 => r = 0	
3	x = 4, y = 6 => r = 24	

Árboles de ejecución de algoritmos recursivos

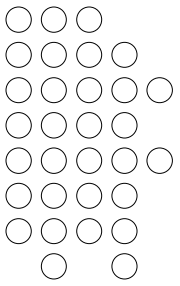


Eliminación de la recursión



- Todo algoritmo recursivo puede ser transformado en un algoritmo iterativo
- Los algoritmos recursivos se utilizan cuando los problemas admiten una descomposición recurrente natural, ya que el algoritmo calca dicha descomposición, siendo más clara y fácil la prueba de corrección
- Inconvenientes de los algoritmos recursivos:
 - ❖ Ciertos lenguajes de programación no soportan la recursividad (Ejm: el lenguaje de máquina)
 - ❖ Son más costosos en tiempo de ejecución y en espacio de memoria

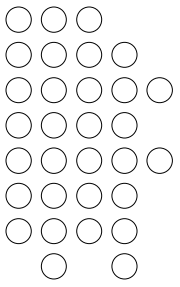
Eliminación de la recursión



Mismas etapas de un compilador para traducir el programa fuente a lenguaje de máquina

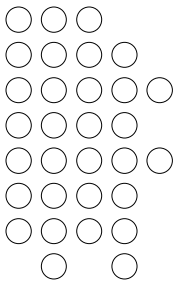
- Se realiza en dos etapas:
 - ❖ Eliminación de los parámetros y variables locales
 - ❖ Transformación
- Etapa 1: sustituir cada ocurrencia de un parámetro formal variable por el parámetro actual correspondiente a la llamada
 - ❖ Eliminación de los parámetros por valor
 - Usando una pila para preservar sus valores
 - Convirtiéndolos en variables globales

Eliminación de la recursión



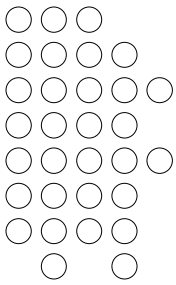
- ❖ Eliminación de las variables locales
- El compilador usa una pila para guardar los valores de cada parámetro por valor y cada variable local
- Etapa 2:
 - ❖ Una sola llamada recursiva: colocar un lazo con un contador o con el control de la pila
 - ❖ Varias llamadas recursivas: cada llamada se reemplaza por una bifurcación al inicio del algoritmo

Ejemplo



Marzo/05		hanoi(Entero: n, TipoEle: a, b, c)
	{ pre: $n \geq 0$ }	{ pos: $n \geq 0$ }
1	Si ($n = 1$) entonces mover(a,b) sino hanoi(a, c, b, n-1) mover(a,b) hanoi(c, b, a, n-1) fsi	mover() . Mueve un aro de a hacia b

Ejemplo



Marzo/05		hanoiIte(Entero: n, TipoEle: a, b, c)	
{ pre: $n \geq 0$ }		{ pos: $n \geq 0$ }	
1	$(n \neq 1) [\text{aux}, b, c, n = b, c, \text{aux}, n-1$ $\text{p.empilar}(1)]$	mover() . Mueve un aro de a hacia b aux: TipoEle . Variable auxiliar para efectuar el intercambio p: Pila . Pila auxiliar para la eliminación de la recursión empilar(), vaciaPila(), conPila(), desempilar() . Operaciones definidas para la clase Pila.	
2	$\text{mover}(a, b)$		
3	$(\neg \text{p.vaciaPila}() \wedge \text{p.conPila}() = 2) [\text{p.desempilar}()$ $\text{aux}, a, c, n = a, c, \text{aux}, n+1]$		
4	Si $(\neg \text{p.vaciaPila}())$ entonces $\text{p.desempilar}()$ $\text{aux}, b, c = b, c, \text{aux}$ $\text{mover}(a, b)$ $\text{aux}, a, c = a, c, \text{aux}$ $\text{p.empilar}(2)$ ir a 1 fsi		

Ejercicio: Eliminar el “ir a 1” del algoritmo anterior