Tarea de programación dinámica

Dalia Camacho, Gabriela Vargas, Elizabeth Monroy

November 18, 2018

Partición

El problema de partición tiene la siguiente formula

$$Particion(m,n) = \begin{cases} 1, & si \quad m = 1 \quad o \quad n = 1 \\ Particion(m,m) & si \quad m < n \\ Particion(m,n-1) & si \quad m = n \\ Particion(m,n-1) + Particion(m-n,m) & si \quad m > n \end{cases} \tag{1}$$

El código de partición recursivo es el siguiente:

```
ParticionRec <- function(m,n){
  if(m==1 || n==1){
    return(1)
  }
  if(m<n){
    return(ParticionRec(m,m))
  }else if(m==n){
    return(1+ ParticionRec(m,n-1))
  }else{
    return(ParticionRec(m,n-1) + ParticionRec(m-n,n))
  }
}
```

Como ejemplo corremos el código partición para $m = 4$ y $n = 4$, lo hacemos para valores pequeños, ya que para valores grandes la pila se llena y el programa truena.

```
ParticionRec(4,4)
```

```
## [1] 5
```

El código recursivo se puede reescribir de forma iterativa. Para esto se define una matriz $M \in \mathbb{R}^{m \times n}$ si $n < m$, o bien una matriz $M \in \mathbb{R}^{m \times m}$ si $n \geq m$, dado que $Particion(m,n) = Particion(m,m)$ sin $n > m$. Dentro de la matriz M se van guardando los valores de la función *Partición* por lo que $M_{i,j} = Particion(i,j)$. M se inicializa con unos en la primera fila y en la primera columna. A partir de eso se define un ciclo sobre las filas empezando en la segunda fila y dentro de este un ciclo sobre las columnas, el cual también empieza en dos. Dentro de los ciclos se va actualizando la matriz M de acuerdo al algoritmo de partición.

```
ParticionIter <- function(m,n){
  if(m==1 || n==1){
    return(1)
  }
  if(m<n){
    n <- m
  }
  M <- matrix(1, nrow = m, ncol = n)
  for(i in 2:m){
    for (j in 2:n) {
```

```r
      if(i<j){
        M[i,j] <- M[i,i]
      }else if(i==j){
        M[i,j] <- 1 + M[i,j-1]
      }else{
        M[i,j] <- M[i,j-1] + M[i-j,j]
      }
    }
  }
  return(M[i,j])
}
```

Comprobamos que el programa iterativo regrese los mismos valores para $m = 4$ y $n = 4$.

```r
ParticionIter(4,4)
```

```
## [1] 5
```

Ahora lo probamos para $m = 100$, $n = 100$

```r
ParticionIter(100,100)
```

```
## [1] 190569292
```

Ackerman

El algoritmo de Ackerman esta dado de la siguiente manera:

$$A(m,n) = \begin{cases} n+1, & m = 0 \\ A(m-1,1), & m > 0 \quad y \quad n = 0, \\ A(m-1, A(m,n-1)) & m > 0 \quad y \quad n > 0 \end{cases} \tag{2}$$

El código recursivo está dadao por:

```r
AckermanRec <- function(m,n){
  if(m==0){
    return(n+1)
  }else if(m>0 & n==0){
    return(AckermanRec(m-1,1))
  }else if(m>0 & n>0){
    return(AckermanRec(m-1, AckermanRec(m,n-1)))
  }
}
```

Vemos el resultado para $m = 3$, $n = 3$.

```r
AckermanRec(3,3)
```

```
## [1] 61
```

Este algoritmo no lo pudimos reescribir en términos de programación dínamica o de forma iterativa, dada la forma en que crece el algoritmo no es posible predeterminar un arreglo del tamaño necesario para guardar los valores. Anteriormente vimos que en el caso de $m = 3$ y $n = 3$ Ackerman da como resultado 61, esto continúa aumentando conforme aumenta m. Seguimos el algoritmo para valores de $m = \in \{0, 1, 2, 3\}$ e intentamos encontrar una fórmula cerrada cada uno de esos casos. El proceso que seguimos fue parecido a un proceso iterativo. Calculamos primero la fórmula para $m = 0$ después para $m = 1$ y así sucesivamente. Además para cada m la n la recorrimos empezando en cero. De esta forma ya conocíamos la fórmula para $A(m-1,n)$ para cualquiern n y el valor de $A(m,n-1)$. Con esto era posible calcular el siguiente valor sin utilizar la cola. Para el caso con $m=0$ la solución se sigue de la definición del algoritmo.

$$A(0,n)=n+1.$$

Para $m=1$ $A(1,0)=A(0,1)=2$. $A(1,n)=A(0,A(1,n-1))=A(0,A(0,A(1,n-2)))$
$=A(0,A(0,A(0,A(0\ldots A(0,A(1,0))))))$
$=1+1+\cdots+1+2=n+2.$

Con lo que

$$A(1,n)=2.$$

Para $m=2$ $A(2,0)=A(1,1)=3$

$A(2,1)=A(1,A(2,0))=2+3=5$

$A(2,2)=A(1,A(2,1))=7$

$A(2,3)=A(1,A(2,2))=9$

A partir de esto se tiene que:

$$A(2,m)=2(n+1)+1$$

Finalmente lo hacemos para $m=3$ $A(3,0)=A(2,1)=5$

$A(3,1)=A(2,A(3,0))=13$

$A(3,2)=A(2,A(3,1))=A(2,13)=29$

$A(3,3)=A(2,A(3,2))=61$

$A(3,4)=A(2,A(3,3))=125$

Con esto podemos ver que $A(3,n)=8(2^n-1)+5$

Calculamos algunos valores para $m=4$

$A(4,0)=A(3,1)=13$

$A(4,1)=A(3,A(4,0))=A(3,13)=8(2^{(13)}-1)+5=65533$

$A(4,2)=A(3,A(4,1))=A(3,65533)=8(2^{(65533)}-1)+5$

Intentamos evaluar $A(4,2)$ con la fórmula anterior y nos da infinito:

```
8*(2^(65533)-1)+5
```

```
## [1] Inf
```

Por lo que nos detenemos en este punto, ya que el algoritmo se vuelve poco manejable.

Construimos una función de Ackerman para valores de $m \in 0,1,2,3$

$$A_restringido(m,n)=\begin{cases} n+1, & m=0 \\ n+2, & m=1 \\ 2(n+1)+1, & m=2 \\ 8(2^(n)-1)+5, & m=3 \end{cases} \tag{3}$$

```
AckermanRestringido <- function(m,n){
  if(m==0){
    return(n+1)
  }else if(m==1){
```

```
    return(n+2)
  }else if(m==2){
    return(2*(n+1)+1)
  }else if (m==3){
    return(8*(2^(n)-1)+5)
  }else{
    stop("Value of m>3, problem grows uncontrollably")
  }
}
```

Ahora lo probamos

```
AckermanRestringido(0,2)
```

```
## [1] 3
```

```
AckermanRestringido(0,189)
```

```
## [1] 190
```

```
AckermanRestringido(1,0)
```

```
## [1] 2
```

```
AckermanRestringido(1,189)
```

```
## [1] 191
```

```
AckermanRestringido(2,0)
```

```
## [1] 3
```

```
AckermanRestringido(2,189)
```

```
## [1] 381
```

```
AckermanRestringido(3,0)
```

```
## [1] 5
```

```
AckermanRestringido(3,189)
```

```
## [1] 6.277102e+57
```

Ruta óptima