Análisis y diseño de algoritmos

Generación de cadenas binarias

Francisco Javier Zaragoza Martínez

Departamento de Sistemas
Universidad Autónoma Metropolitana Azcapotzalco

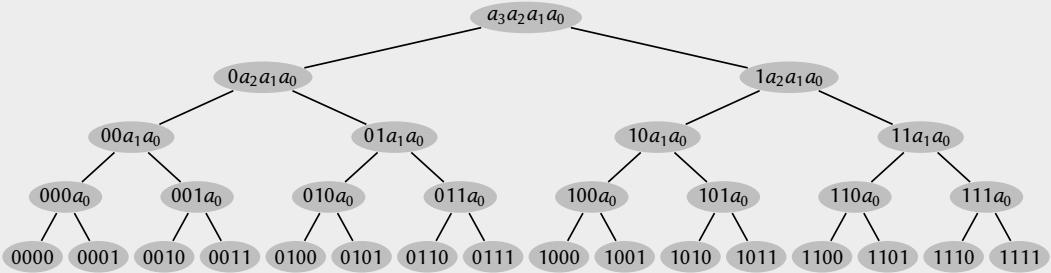
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Azcapotzalco

Posgrado en Optimización

10 de febrero de 2026

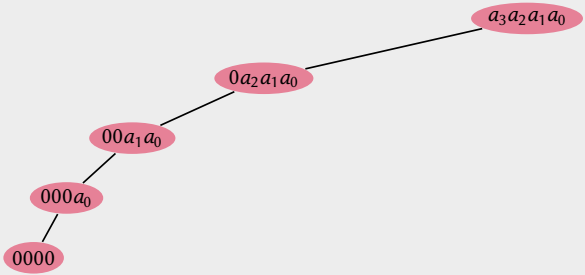
Generación de cadenas binarias en orden lexicográfico

Árbol de toma de decisiones



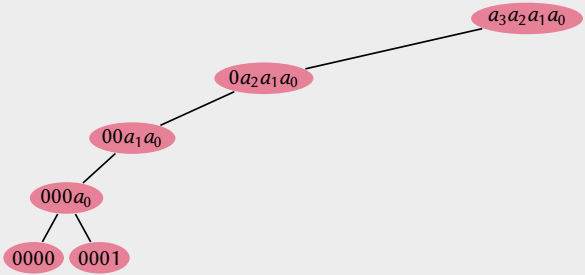
Generación de cadenas binarias en orden lexicográfico

Árbol de llamadas recursivas



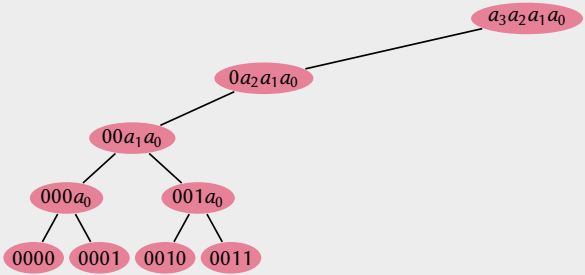
Generación de cadenas binarias en orden lexicográfico

Árbol de llamadas recursivas



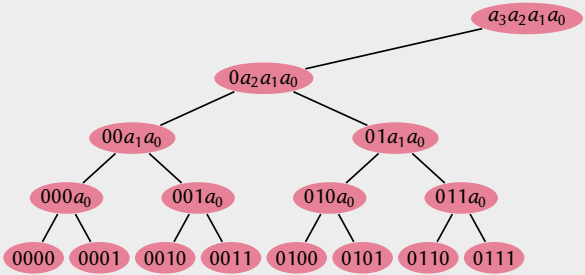
Generación de cadenas binarias en orden lexicográfico

Árbol de llamadas recursivas



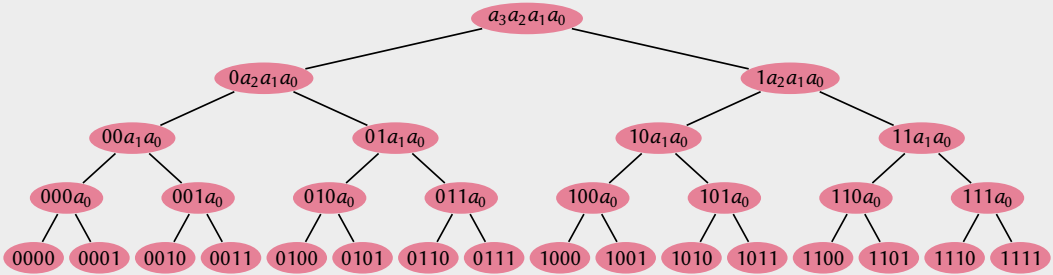
Generación de cadenas binarias en orden lexicográfico

Árbol de llamadas recursivas



Generación de cadenas binarias en orden lexicográfico

Árbol de llamadas recursivas



Generación de cadenas binarias en orden lexicográfico

Algoritmo recursivo

procedimiento LEXICOGRÁFICO(n, a) ▷ La cadena binaria es $a = (a_{n-1}, \ldots, a_0)$

  si $n > 0$ **entonces**

    $a_{n-1} \leftarrow 0$ ▷ Bit alto en 0 y llena recursivamente lo demás

    LEXICOGRÁFICO($n-1$, a)

    $a_{n-1} \leftarrow 1$ ▷ Bit alto en 1 y llena recursivamente lo demás

    LEXICOGRÁFICO($n-1$, a)

  si no

    PROCESA(a) ▷ Si $n = 0$ la cadena está completamente llena

Si $T(n)$ es la cantidad de veces que se escribe un bit, entonces $T(0) = 0$ y $T(n) = 2T(n-1) + 2$. De aquí se obtiene que $T(n) = 2(2^n - 1)$.

Generación de cadenas binarias en orden lexicográfico

Implementación en C

```
void blex(int n, int a[]) {
  if (n > 0) {
    a[n-1] = 0;
    blex(n-1, a);
    a[n-1] = 1;
    blex(n-1, a);
  } else procesa(a); // Lo que se quiera hacer
}
```

Generación de cadenas k-arias en orden lexicográfico

Algoritmo recursivo

procedimiento LEXICOGRÁFICO(n, k, a) ▷ La cadena k-aria es $a = (a_{n-1}, \ldots, a_0)$
  si $n > 0$ **entonces**
    para $i \leftarrow 1$ **a** k **haz**
      $a_{n-1} \leftarrow i$ ▷ Primera posición en i y llena recursivamente lo demás
      LEXICOGRÁFICO($n-1, k, a$)
  si no
    PROCESA(a) ▷ Si $n = 0$ la cadena está completamente llena

Si $T(n)$ es la cantidad de veces que se escribe una posición, entonces $T(0) = 0$ y $T(n) = kT(n-1) + k$. De aquí se obtiene que $T(n) = \frac{k}{k-1}(k^n - 1)$.

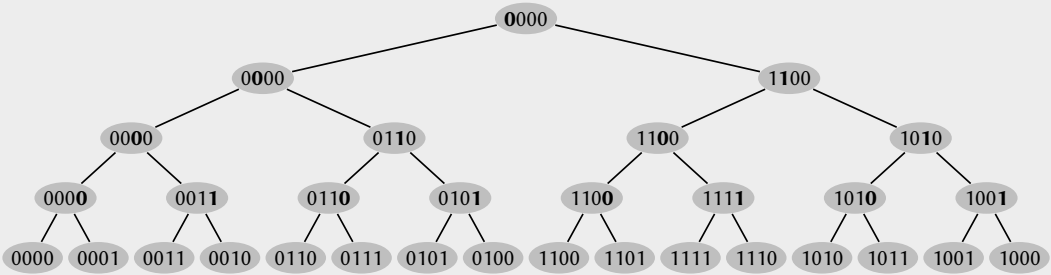
Generación de cadenas k-arias en orden lexicográfico

Implementación en C

```
void klex(int n, int k, int a[]) {
  if (n > 0) {
    for (int i = 1; i <= k; i++) {
      a[n-1] = i;
      klex(n-1, k, a);
    }
  } else procesa(a); // Lo que se quiera hacer
}
```

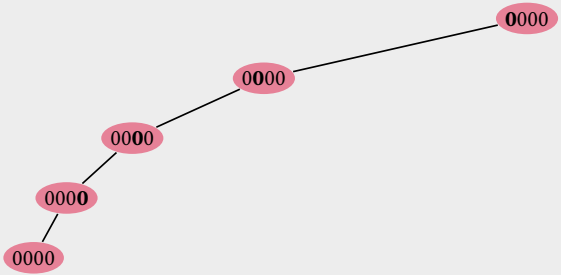
Generación de cadenas binarias en orden Gray reflejado

Árbol de toma de decisiones



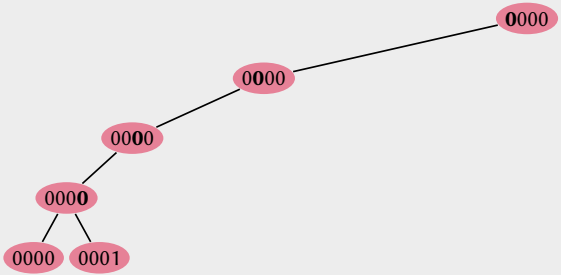
Generación de cadenas binarias en orden Gray reflejado

Árbol de llamadas recursivas



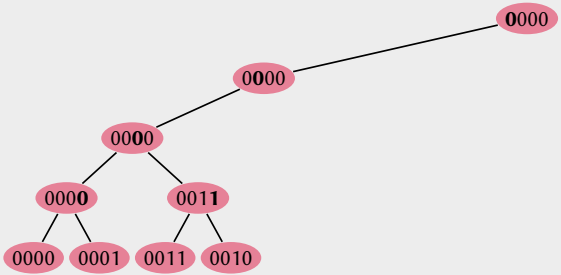
Generación de cadenas binarias en orden Gray reflejado

Árbol de llamadas recursivas



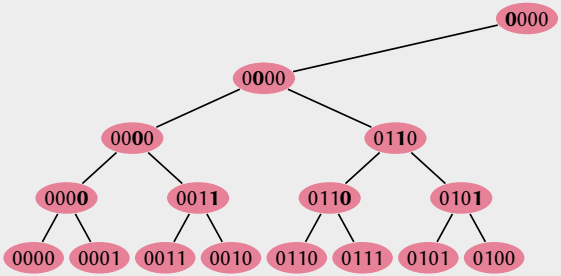
Generación de cadenas binarias en orden Gray reflejado

Árbol de llamadas recursivas



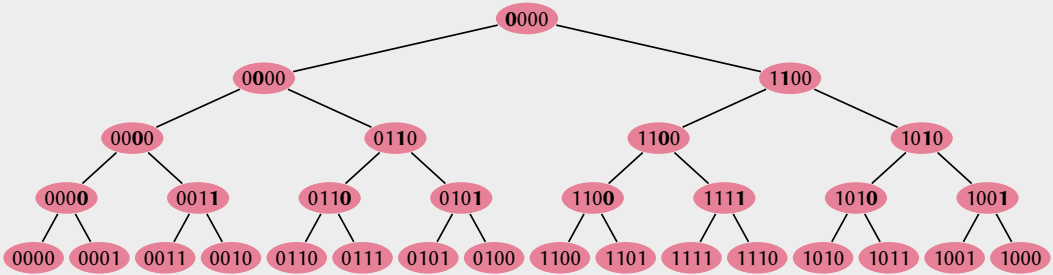
Generación de cadenas binarias en orden Gray reflejado

Árbol de llamadas recursivas



Generación de cadenas binarias en orden Gray reflejado

Árbol de llamadas recursivas



Requiere Al principio la cadena binaria a debe estar llena de 0

procedimiento REFLEJADO(n, a) ▷ La cadena binaria de n bits es $a = (a_{n-1}, \ldots, a_0)$

si $n > 0$ **entonces**

REFLEJADO$(n-1, a)$

$a_{n-1} \leftarrow 1 - a_{n-1}$ ▷ Complementa el bit alto y llena recursivamente lo demás

REFLEJADO$(n-1, a)$

si no

PROCESA(a) ▷ Si $n = 0$ la cadena está completamente llena

Si $T(n)$ es la cantidad de veces que se escribe un bit, entonces $T(0) = 0$ y $T(n) = 2T(n-1) + 1$. De aquí se obtiene que $T(n) = 2^n - 1$, lo cual es óptimo.

Generación de cadenas binarias en orden Gray reflejado

Implementación en C

```
void gray(int n, int a[]) {
  if (n > 0) {
    gray(n-1, a);
    a[n-1] = 1-a[n-1];
    gray(n-1, a);
  } else procesa(a); // Lo que se quiera hacer
}
```