| | | |
|---|---|---|
| Universidad de los Andes<br>Facultad de Ingeniería | **Ingeniería de Sistemas y Computación**<br>ISIS1105 – Diseño y análisis de algoritmos<br>Sección 1. Profesor: Jorge Duitama<br>Semestre: 2023-10 | Accredited ABET Engineering Accreditation Commission |

# Examen 1

Este examen es individual. Tiene una duración de 1 hora 20 min. Está prohibido usar el libro, apuntes, impresiones y cualquier tipo de dispositivo electrónico (celulares, agendas electrónicas, cámaras digitales, etc.).

Nombre:_____ Código: _____

1. [20%] Determinar la complejidad temporal T(n) del siguiente programa que determina si en un arreglo de números enteros a, de tamaño n=a.length, están alternados números positivos y negativos.

```
boolean isSignAlternating(int [] a) {
      return isSignAlternating(a,a.length-1);
}
boolean isSignAlternating(int [] a, int f) {
      if (f==0) return true;
      if (f==1) return true;
      return a[f]*a[f-1]<0 && a[f-1]*a[f-2]<0 && isSignAlternating(a,f-2);
}
```

Esta es la tabla de operaciones básicas llamadas en cada caso base (f=0 y f=1) y en el caso recursivo.

| Operación | Constante | Veces caso base f=0 | Veces caso base f=1 | Veces caso recursivo |
|---|---|---|---|---|
| Comparación (==,<) | $c_1$ | 1 | 2 | 4 |
| Resta (-) | $c_2$ | 0 | 0 | 4 |
| Multiplicación | $c_3$ | 0 | 0 | 2 |
| Conjunción | $c_4$ | 0 | 0 | 2 |

$$T(0) = c_1$$
$$T(1) = 2c_1$$
$$T(n) = 4c_1 + 4c_2 + 2c_3 + 2c_4 + T(n-2)$$
$$= k_1 + T(n-2)$$

Donde $k_1 = 4c_1 + 4c_2 + 2c_3 + 2c_4$

$T(n)$ es una ecuación de recurrencia lineal de orden 2.

Se resuelve como $T(n)=h(n)+p(n)$ donde $h(n)$ es solución a la ecuación de recurrencia homogenea correspondiente $h(n)-h(n-2)=0$ y $p(n)$ es una solución particular. El polinomio carácterístico relacionado con $h(n)$ es: $\lambda^2 - 1$ con raices $r=1$ y $r=-1$. Por lo tanto

$$h(n)=d_1(1)^n+d_2(-1)^n = d_1+d_2(-1)^n$$

donde $d_1$ y $d_2$ son constantes. Como la función constante soluciona $h(n)$, se busca una solución particular $p(n)=d_3n$. Reemplazando en la ecuación original:

$$T(n) - T(n-2) = k_1$$
$$d_3 n - d_3 (n-2)= k_1$$
$$d_3=k_1/2$$

La solución es entonces

$$T(n) = h(n) + p(n) = d_1 +d_2(-1)^n + nk_1/2$$

Como es de orden 2, las condiciones iniciales $T(0)$ y $T(1)$ son suficientes para averiguar $d_1$ y $d_2$:

$$c_1 = T(0) = d_1 + d_2 (-1)^0 + (0)k_1 /2 = d_1 + d_2$$
$$2c_1 = T(1) = d_1 + d_2 (-1)^1 + (1)k_1 /2 = d_1 - d_2 + k_1 /2$$

Sumando las dos ecuaciones

$$3c_1 = 2d_1 + k_1 /2$$
$$d_1 = 3c_1 /2 - k_1 /4$$

Reemplazando en la primera ecuación:

$$c_1 = 3c_1 /2 - k_1 /4 + d_2$$
$$d_2 = k_1 /4 - c_1 /2$$

Finalmente

$$T(n) = d_1 +d_2(-1)^n + nk_1/2$$
$$= 3c_1 /2 - k_1 /4 + (k_1 /4 - c_1 /2)(-1)^n + nk_1/2$$
$$= 3c_1 /2 - (4c_1 + 4c_2 + 2c_3 + 2c_4)/4 + ((4c_1 + 4c_2 + 2c_3 + 2c_4)/4 - c_1 /2)(-1)^n + n(4c_1 + 4c_2 + 2c_3 + 2c_4)/2$$
$$= (c_1 - 2c_2 - c_3 - c_4 + (c_1 + 2c_2 + c_3 + c_4)(-1)^n + n(2c_1 + 2c_2 + c_3 + c_4))/2$$

por lo tanto $T(n)$ es $O(n)$

2. [20%] Resolver la siguiente ecuación de recurrencia: $f(n)=4*f(n/2)+\log_2(n)+3$ con $f(1)=2$

$f(n)$ es una ecuación de recurrencia no lineal.

Se realiza cambio de dominio: $n = 2^m = g(m)$ y $\Phi(m) = T(g(m))$

$$f(n) - 4f(n/2) = \log_2(n)+3$$
$$f(2^m) - 4f(2^m/2) = \log_2(2^m)+3$$
$$f(2^m) - 4f(2^{m-1}) = m+3$$
$$f(g(m)) - 4f(g(m-1)) = m+3$$
$$\Phi(m) - 4\Phi(m-1) = m+3$$

Se resuelve como $\Phi(m)=h(m)+p(m)$ donde $h(m)$ es solución a la ecuación de recurrencia homogenea correspondiente $h(m)-4h(m-1)=0$ y $p(m)$ es una solución particular. El polinomio carácterístico relacionado con $h(m)$ es: $\lambda-4$ con raiz $r=4$. Por lo tanto $h(m)=d_1(4^m)$ donde $d_1$ es una constante. Se busca una solución particular $p(m)=d_2m+d_3$. Reemplazando en la ecuación original:

$$\Phi(m) - 4\Phi(m-1) = m+3$$
$$d_2 m + d_3 - 4(d_2 (m-1)+d_3)= m+3$$
$$d_2 m - 4d_2m - m = 3d_3 -4d_2 + 3$$
$$-m (3d_2 + 1) = 3d_3 -4d_2 + 3$$

Igualando a cero la parte izquierda $d_2 = -1/3$
Igualando a cero la parte derecha $3d_3 + 4/3 + 3 = 0 \Rightarrow d_3 = -13/9$

La solución es entonces:

$$\Phi(m) = h(m) + p(m) = d_1(4^m) + - m/3 - 13/9$$

Reemplazando $m = \log_2(n)$

$$f(n) = d_1 n^2 - \log_2(n)/3 - 13/9$$

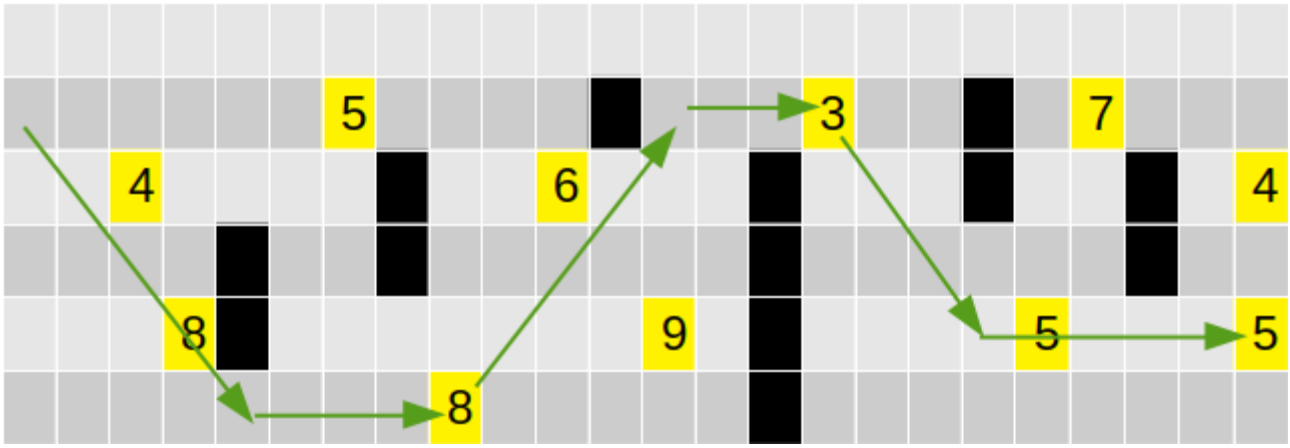
La condición inicial $f(1)$ es suficiente para hallar $d_1$:

$$2 = f(1) = d_1 (1)^2 - \log_2(1)/3 - 13/9 = d_1 - 13/9$$
$$d_1 = 31/9$$

Finalmente

$$f(n) = (31n^2 - 3\log_2(n) - 13)/9$$

3. Desarrollar un algoritmo de programación dinámica para resolver el siguiente problema. En un juego de video, un carro va por una via con n carriles. En cada metro de recorrido el carro puede permanecer en el mismo carril, moverse a alguno de los dos carriles adyacentes, a menos que esté en un borde y entonces solo tiene un carril para cambiarse. Durante el recorrido, en algunos carriles de la via puede haber obstáculos que harían estrellar al carro, o hay monedas de oro que se puede ganar el protagonista del juego. Dada una representación de la vía, incluyendo sus obstáculos y monedas de oro, determinar la máxima cantidad de monedas que puede obtener el jugador sin estrellarse.
Ejemplo: La siguiente figura muestra un ejemplo de una via con 6 carriles. Las posiciones amarillas indican posiciones con monedas de oro y las posiciones negras indican obstáculos. En verde se indica un camino que puede seguir el carro para maximizar la cantidad de monedas.




a. [10%] Describir entradas y salidas del problema

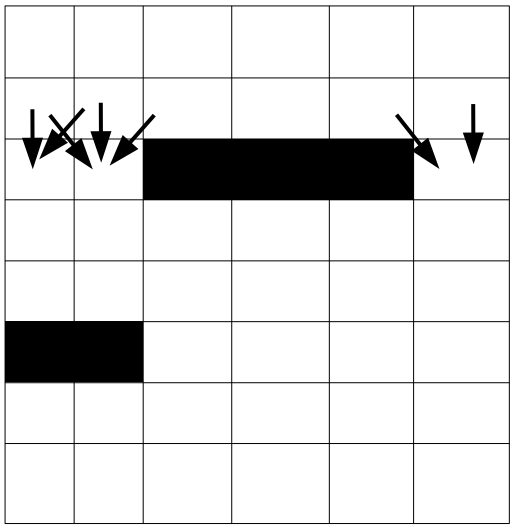
| E/S | Nombre | Tipo | Descripción |
|---|---|---|---|
| E | A | Matrix[0,N)x[0,M) of int | Matriz que representa la via con N metros y M carriles. Cada casilla contiene un -1 si hay obstáculo. En otro caso, contiene las monedas en la casilla correspondiente |
| S | r | nat | Maximo número de monedas que se pueden llevar con la restricción de que el carro siempre avanza un metro y se puede cambiar solamente a un carril adyacente |

b. [20%] Definir una ecuación de recurrencia `m(i,j)` que determine la máxima cantidad de monedas de oro que puede recolectar el jugador después de recorrer `i` metros y llegando al carril `j` sin estrellarse.

```
m(i,j) = -∞                                   si A[i,j]=-1
m(i,j) = A[i,j]                               si i=0 ∧ A[i,j]≥0
m(i,j) = A[i,j]+max(m(i-1,j),m(i-1,j+1))      si i>0 ∧ j=0 ∧ A[i,j]≥0
m(i,j) = A[i,j]+max(m(i-1,j),m(i-1,j-1))      si i>0 ∧ j=M-1 ∧ A[i,j]≥0
m(i,j) = A[i,j]+max(m(i-1,j-1),m(i-1,j),m(i-1,j+1))  si i>0 ∧ 0<j<M-1 ∧
A[i,j]≥0
```

c. [10%] Dibujar el grafo de necesidades relacionado con esta ecuación. Explicar qué representan los vértices y qué representan los ejes del grafo. Determinar si el grafo tiene ciclos.

El grafo tiene forma de matriz de NxM donde cada casilla es un vértice. Los ejes representan dependencias entre soluciones. De acuerdo con la ecuación, las casillas con obstáculos (marcadas con negro) no tienen dependencias. Se muestran ejemplos de los tres casos de dependencias



d. [20%] Desarrollar un algoritmo de programación dinámica para resolver el problema.

```
fun maxMoney (A: array[0,N)x[0,M) of int) ret r: int
var m: array [0,N)x[0,M) of int
var i,j,r: nat
i,r:=0,0
do i<N →
      j:=0
      do j< M →
            if A[i,j]=-1 → m[i,j] :=-∞
            [] A[i,j]≥0 →
                  if i=0 → m[i,j]:= A[i,j]
                  [] i>0 ∧ j=0 → m[i,j]:=A[i,j]+max(m[i-1,j],m[i-1,j+1])
                  [] i>0 ∧ j=M-1 → m[i,j]:=A[i,j]+max(m[i-1,j],m[i-1,j-1])
                  [] i>0 ∧ 0<j<M-1 → m[i,j]:=A[i,j]+max(m[i-1,j],
                                                m[i-1,j-1],m[i-1,j+1])
                  fi
            []
            fi
            if i=N-1 → r := max(r,m[i,j])
            [] i<N-1 → skip
            fi
            j:=j+1
      od
      i:=i+1
od
ret r
```

e. [10%] Determinar la complejidad temporal y espacial del algoritmo.

Suponiendo que en el peor de los casos no hay obstáculos y entendiendo que en el condicional anidado solo se ejecuta un caso en cada vuelta del ciclo, la siguiente tabla resume las operaciones que se ejecutan:

| **Operación** | **Constante** | **Veces** |
|---|---|---|
| Asignación (:=) | $c_1$ | $1+2N+NM+M$ |
| Comparación (=, <, >, ≥) | $c_2$ | $1+N+N(M+1)+12NM$ |
| Suma / Resta (+, -) | $c_3$ | $N+8NM$ |
| Max | $c_4$ | $(N-1)M+M$ |

$$T(N,M) = NM\ (c_1 + 13c_2 + 8c_3 + c_4\ ) + N\ (2c_1 + 2c_2 + c_3) + M(c_1) + c_1 + c_2$$

$T(N,M)$ es $O(NM)$

La complejidad espacial está determinada por el tamaño de la matriz m. Por lo tanto:

$S(N,M)$ es $O(N,M)$