Maestría en Ciencias con Especialidad en Computación y Matemáticas Industriales
Proceso de admisión 2024
Centro de Investigación en Matemáticas (CIMAT), A.C.
Examen de programación
(Tiempo: 3 horas)

Nombre: _____

Fecha: _____

Instrucciones:

- Escriba lo más claro posible para que al digitalizar las respuestas sean legibles.
- Resuelva cada problema en hojas independientes.
- Escriba su nombre en cada hoja.
- No se debería de pasar más de 30 minutos en cada problema.
- Escriba el código lo más claramente posible y especifique el lenguaje/pseudo-código utilizado.
- Si requiere funciones estándar como determinar el máximo o mínimo de dos valores (max(a, b), min(a,b)), ordenar (sort), determinar el tamaño de una cadena de caracteres (strlen), obtener el valor absoluto (abs), raíz cuadrada (sqrt) o logaritmos (log) puede utilizarlas sin implementarlas. Será necesario que implemente cualquier función diferente a las mencionadas anteriormente.

Problema 1 [1.5 puntos]

Considere la matriz $\mathbf{A}$ de dimensión $m \times n$ y la matriz $\mathbf{B}$ de dimensión $p \times q$. El producto de Kronecker $\mathbf{A} \otimes \mathbf{B}$ es una matriz de dimensión $pm \times nq$ con la siguiente estructura:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \dots & a_{1mn}\mathbf{B} \end{bmatrix}$$

Desarrolle una función que reciba cualquier par de matrices $\mathbf{A}$ y $\mathbf{B}$ y regrese el producto de Kronecker $\mathbf{A} \otimes \mathbf{B}$.

Ejemplo: Para $\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ y $\mathbf{B} = \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix}$, la función regresará:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} 1\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 2\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \\ 3\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 4\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 5 & 0 & 10 \\ 6 & 7 & 12 & 14 \\ 0 & 15 & 0 & 20 \\ 18 & 21 & 24 & 28 \end{bmatrix}.$$

Solution:

```
for (i=0;i<m;i++) {
     for (j=0;j<n;j++) {
         for (k=0;k<p;k++) {
             for (l=0;l<q;l++) {
                 K[i*m+k][j*n+l] = A[i][j]*B[k][l];
             }
         }
     }
}
```

Problema 2 [1.5 puntos]

La función de Ackermann tiene aplicaciones interesantes en teoría de la computación. Esta función recibe como argumentos dos números enteros $n \geq 0$ y $m \geq 0$ y regresa como salida un número natural. Esta función se define como:

$$A(m,n) = \left\{ \begin{array}{ll} n+1, & \text{if } m = 0 \\ A(m-1,1), & \text{if } m > 0 \text{ y } n = 0 \\ A(m-1, A(m,n-1)), & \text{if } m > 0 \text{ y } n > 0 \end{array} \right\}$$

Proporcione un Pseudocódigo o implemente un código, que reciba como parámetros de entrada dos enteros $n \geq 0$, $m \geq 0$ y regrese como sálida el valor que retorna la función de Ackermann $A(m,n)$.

Ejemplo: Para

1. $A(2,2) = 7$
2. $A(3,1) = 13$
3. $A(2,3) = 9$

Solution:

```
int Ack(int m,int n){
   int output=1;
   if (m==0){
      return output=n+1;
   } else if (n==0){
      return Ack(m-1,1);
   }
```

```
     else {
        return Ack(m-1,Ack(m,n-1));
     }
}
```

Problema 3 [1.5 puntos]

Consideremos la función `count` siguiente:

```
unsigned int count(unsigned int N) {
     unsigned int i,j;
     unsigned int count = 0;
     for (i=0;i<N;i++) {
          for (j=0;j<N;j++) {
               if ((i%2==0 && j%2==1)|| (i%2==1 && j%2==0)){
                    count++;
               }
          }
     }
     return count;
}
```

Dar una expresión de lo que regresa esta función en términos de N.

Nota: El símbolo % es la operación módulo que regresa el residuo de la división entera, por ejemplo, $5\,\%2 = 1$, con 5 el dividendo, 2 el divisor y 1 el residuo. La operación a && b corresponde al AND lógico entre a y b. La operación a || b corresponde al OR lógico entre a y b.

Solution: El doble ciclo anidado se puede interpretar como el recorrido de una matriz de $N \times N$. En esa matriz, empezando por la primer fila, se suma una casilla sí y otra no, empezando desde la segunda casilla. En la segunda fila, se suma una casilla y otra no, empezando desde la primer casilla, y así sucesivamente. Al final, para cuando N es par, se cuentan sólo la mitad de las casillas de la matriz por lo que `count` $= \frac{N^2}{2}$. Cuando N es impar, la cuenta corresponde a una matriz de $(N+1) \times (N+1)$ donde se quitan los elementos que se hubieran sumado en su última fila y columna, esto es, `count` $= \frac{(N+1)\times(N+1)}{2} - (N+1) = \frac{N^2}{2} - \frac{1}{2}$. Dado lo anterior, el resultado es `count` $= \lfloor \frac{N^2}{2} \rfloor$.

Problema 4 [1.5 puntos]

Proporcione un código o pseudocódigo que reciba como entrada una lista simplemente ligada L, de forma tal que al recorrer la lista L solo una vez, retorne como salida una lista simplemente ligada L' que consista en los elementos de L en orden inverso.

Ejemplo: para $L = 1 \to 2 \to \ldots \to 9 \to$ NULL el algoritmo debe retornar $L' = 9 \to 8 \to \ldots \to 1 \to$ NULL.

Solution: La idea es posicionarse al inicio de la lista L. Recorrer la lista L con apoyo de tres apuntadores, el apuntador *actual* que irá recorriendo la lista desde el inicio hasta el final de la lista. El apuntador *sig* que apuntará al siguiente elemento de donde se encuentra *actual*, y *prev*

que apuntará al elemento previo del que se encuentra *actual*. En cada iteración se invierten el apuntador de *actual.sig* a la posición en la cual se encuentra el apuntador *prev*. El apuntador *sig* es un auxiliar que evita que se pierda el apuntador del elemento *actual*.

```
void invierte()
   {
       // Inicializa tres apuntadores
       Node* actual = head;
       Node *prev = NULL;
       Node *sig = NULL;

       while (actual != NULL) {
           // Actualiza el valor del apuntador sig
           sig = actual->sig;
           // Invierte el apuntador del nodo actual
           actual->sig = prev;
           // Mueve los apuntadores.
           prev = actual;
           actual = sig;
       }
       head = prev;
   }
```

Problema 5 [2 puntos]

Dado un arreglo de N números enteros, encontrar el subarreglo no vacío cuyo producto de sus elementos sea mínimo, esto sin enumerar todos los subarreglos posibles. Regresar el producto mínimo y el subarreglo de elementos correspondiente.

Ejemplo: Dado el arreglo $[-4, -6, -2, 3, 0, 6]$.
El subarreglo $[-4, -6, -2, 3, 6]$ tiene el producto mínimo con valor -864. Todos los otros subarreglos tendrán un producto mayor a -864.

Solution: Si es realizado mediante la generación de todos los subconjuntos, su orden será $O(2^N)$. Una posible solución es utilizar un conjunto de reglas. Si todos los valores son cero y positivos, el resultado es cero. Si todos lo valores son positivos, el resultado es el valor mínimo del arreglo. Si existe una cantidad impar de valores negativos, el resultado es el producto de todos los valores no cero. Si existe una cantidad par de valores negativos, el resultado es el producto de todos los valores diferentes de cero dividido entre el número negativo mayor.

Problema 6 [2 puntos]

Dado un número S y un arreglo de N números enteros en el rango $[1, 10^9]$, queremos encontrar un subarreglo (continuo) del menor tamaño posible, cuya suma sea mayor a S.

(1 punto) a) Desarrolle una función que reciba un arreglo y el valor de S, y devuelva el tamaño del menor arreglo que cumple que su suma es mayor a S. En caso de que no exista un subarreglo cuya suma sea mayor a S, debe devolver -1. Considere que la función debe terminar en menos de 1 segundo de cómputo y que el arreglo que se le va a pasar, es como máximo de tamaño $N = 100$. Haga una estimación de cuántas operaciones se realizan en función del valor de N.

(1 punto) b) Desarrolle una función con la misma funcionalidad para tamaño igual a $100,000$, es decir, que para dicho tamaño de arreglo termine en menos de 1 segundo. Igualmente haga una estimación de cuántas operaciones se realizan en función del valor de N. Nota: si no se le ocurre cómo extender su código para ese tamaño, pero se le ocurren algunas ideas para hacer algún código más eficiente que el desarrollado en el código a, explique dichas ideas.

Ejemplos:

- Para $[8, 20, 15, 17, 12]$ con $S = 34$, la función debe devolver el número 2, pues por ejemplo el subarreglo $[20, 15]$ suma 35, y no hay subarreglos de tamaño 1, cuya suma sea mayor a 35.
- Para $[8, 20, 15, 17, 12]$ con $S = 200$, la función debe devolver el número -1, pues incluso si consideramos el arreglo completo, la suma es menor a 200.
- Para $[8, 20, 15, 17, 12]$ con $S = 70$, la función debe devolver el número 5, pues el único subarreglo cuya suma es mayor a 70 es el conformado por el arreglo completo.

Solution: Para el apartado a) se puede hacer una búsqueda completa, es decir, iterar sobre todos los posibles subarreglos ($O(N^2)$) y para cada uno calcular la suma, dando una complejidad de $O(N^3)$. Hay diversas formas de mejorar, si por ejemplo evitamos las sumas de cada subarreglo mediante suma de prefijos, se desarrolla un algoritmo que es $O(N^2)$. Se puede bajar a $O(N)$ utilizando ventanas deslizantes, o a $O(NlogN)$ utilizando búsqueda binaria.