CC40A: Diseño y Análisis de Algoritmos
Pauta Examen - Semestre Otoño 2000

Profesor: Ricardo Baeza Yates, Prof. Auxiliar: Felipe Saint-Jean

P1 (1.5 ptos.)

Considere el siguiente algoritmo:

```
Mistery( x, n ) {
    m = n/2;
    for( i=1; i<=m; i++ ) {
         temp[i] = (x[i]+x[i+1])/2;
         temp[m+i] = (x[i]-x[i+1])/2;
    }
    for( i=1; i<=n; i++ ) x[i] = temp[i];
    Mistery( x, n/2 );
}
```

- *¿Cuál es el valor final de x si llamamos a* `Mistery( [11,9,9,11,7,11,4,12], 8)`*? (0.5)*

  Mistery nunca termina pues la recursividad no está acotada. Sin embargo, cuando $n < 1$ Mistery no entra nunca más en ningún ciclo y entonces el valor final de x es `[0, 0, 0, 0, 1, 0, -1, 2]`.

- *¿Qué restricción hay para n para que* `Mistery` *funcione correctamente? (0.3)*

  Tenemos que $n \leq \min(|temp|, |x| - 1)$ para que el programa funcione.

- *¿Cuál es la complejidad de* `Mistery` *en función de n? (0.4)*

  Es no acotada pues Mistery no termina. Si hubiera una condición de término, por ejemplo cuando $n < 1$, tendríamos $T(n) = O(n) + T(n/2)$ lo que al final implicaría $T(n) = O(n)$.

- *¿Cómo modificaría* `Mistery` *para eliminar la llamada recursiva y* **no** *aumentar la complejidad temporal? (0.3)*

  Da lo mismo pues nunca termina, pero se puede convertir la llamada recursiva en un ciclo que contiene al código dado dividiendo en cada paso n por la mitad y sin condición de término.

P2 (1.5 ptos.)

Suponga que tiene un arreglo S con n números reales y un número real x. Determinar si existen dos elementos del arreglo que sumen x en los siguientes casos y con las siguientes complejidades:

- *S está desordenado ($O(n \log n)$) (0.8).*

  Ordenamos S usando Heapsort o Mergesort (peor caso) y luego buscamos para cada elemento del arreglo z usando búsqueda binaria si está $z - x$. Si está terminamos siendo z y el número encontrado la solución. Si al final no encontramos ninguno, no existe.

- *S está ordenado ($O(n)$) (0.7)*

  Usamos inducción. Tomamos el primer elemento y el último y los sumamos. Sea y la suma de S_1 y S_n. Si $y = x$ terminamos. Si $y > x$ entonces S_n no puede ser parte de la solución, porque como S está ordenado $S_i + S_n > x$ para todo i. Por lo tanto descartamos S_n. Si $y < x$, usando el mismo argumento descartamos S_1. Luego, a cada paso con un número constante de operaciones eliminamos un elemento y entonces el tiempo es $O(n)$.

P3 (1.5 ptos.)

Diseñe un algoritmo paralelo para ordenar n elementos usando comparaciones en un computador de memoria compartida con p procesadores (usando el modelo CREW) (1.0). Calcule el coeficiente de paralelización en función de p. Calcule el valor máximo de este coeficiente y para que p se alcanza, indicando si el grado de paralelización es el máximo posible (0.5).

Hay muchas soluciones, lo más fácil es paralelizar Mergesort que además es óptimo en el peor caso. Dividimos el arreglo en pedazos de tamaño n/p y cada procesador ordena su pedazo en tiempo $O(n\log(n/p)/p)$ usando Mergesort secuencial. Luego vamos haciendo las mezclas en paralelo de modo que usamos todos los procesadores en cada paso (si no el tiempo será $O(n\log n)$). Por ejemplo, como cada parte ya está ordenada, encontramos los puntos de corte en cada pedazo de modo que al mezclar cada procesador haga la misma cantidad de trabajo. Usando esto cada etapa de la mezcla se resuelve en tiempo $O(n/p)$ y hay $O(\log(p)$ etapas. Por lo tanto, el algoritmo en total tiene tiempo $(O(n/p(\log(n/p) + \log(p)))$.

El factor de paralelización es entonces orden de:

$$\frac{n\log(n)}{n/p(\log(n/p) + \log(p))} = \frac{p}{1 + \frac{\log p}{\log n}}$$

Notar que no podemos cancelar los dos términos $\log(p)$ porque estamos usando orden de y las constantes de cada término pueden ser distintas (supondremos que la segunda parte es más cara). Este factor es casi el máximo posible (que es p). En este algoritmo p puede ser hasta n, pero si p es muy cercano a n el denominador se aproxima a 2. El valor óptimo dependerá de las constantes de cada algoritmo, pero en la mayoría de los casos será cercano al p máximo.

P4 (1.5 ptos.)

Diseñe una heurística para colorear un mapa político del mundo tal que dos países fronterizos no tengan el mismo color y que use el menor número posible de colores (0.8). Suponga que su heurística usa c colores. Encuentre una cota inferior y superior para la constante de aproximación de su algoritmo respecto al número mínimo óptimo de colores (0.7).

Un mapa es un grafo planar y vimos en clases que un grafo planar se puede colorear en el peor caso con 4 colores. Si hay dos países fronterizos, necesitamos al menos 2 colores. Luego la constante de aproximación estará acotada por $c/4$ y $c/2$.

Hay muchas heurísticas. Por ejemplo, podemos usar un recorrido en profundidad de modo que cada vez que visitamos un nodo le asignamos un color de modo que no sea el mismo que los colores de los nodos adyacentes ya visitados. Si no existe ese color, agregamos un color nuevo y se lo ponemos a ese nodo. Terminamos cuando ya hemos recorrido todo el grafo (que podría no ser conexo).