CC40A: Diseño y Análisis de Algoritmos
Pauta Control 2 - Semestre Otoño 2001

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

P1 (2 ptos.)

Considere el siguiente algoritmo:

```
Misterio( A, n ) {   // A es un arreglo de n enteros
   i = random(1..n); x = A[i]; k = 0;
   for( i=1; i<=n; i++ ) if( A[i] == x ) k++;
   return( k > n/2 );
}
```

- ¿Qué hace este algoritmo? ¿Este algoritmo es de tipo Las Vegas o Montecarlo?

  Este algoritmo retorna verdadero si encuentra un elemento que es mayoritario en el arreglo (es decir, por ejemplo, el ganador de una elección, recordar la pregunta 2 del control 1). Como siempre da una respuesta, aunque sea incorrecta, es de tipo Montecarlo. (0.7)

- ¿La respuesta es siempre correcta? Si no lo es, indique una cota superior de la probabilidad de una respuesta incorrecta. ¿Este algoritmo es 1-sesgado o 2-sesgado?

  No, ya que es de tipo Montecarlo. Si el algoritmo entrega un verdadero, es correcto, pues la solución es única (no pueden haber dos elementos que sean mayoritarios). Si la respuesta es falso, puede estar correcto, pero también puede haber escogido un elemento no mayoritario. Luego, cuando se equivoca la probabilidad es menor a $1/2$, porque si hay un elemento mayoritario, escoger uno no mayoritario es menor a $1/2$. Dado que un caso la respuesta siempre es correcta, es un algoritmo 1-sesgado. (0.7)

- Usando este algoritmo, escriba un algoritmo que siempre de una respuesta correcta con alta probabilidad $(1-\epsilon)$ cuando termina.

  Si repetimos k veces Misterio, la probabilidad de equivocarse será menor a $1/2^k$. Luego, $\epsilon < 1/2^k$, es decir $k > \log_2(1/\epsilon)$. Luego, basta con llamar a Misterio ese número de veces, pues cada intento es independiente del otro, deteniendose si retorna verdadero y sólo al final si todos los intentos dan falso, se retorna falso. (0.6)

P2 (2 ptos.)

a) Diseñe un algoritmo paralelo para p procesadores en el modelo CREW para calcular el máximo de n elementos almacenados en un arreglo. ¿Existe algún valor de p que minimice el tiempo paralelo de su algoritmo? Indique el factor de paralelización del algoritmo.

   Usamos la idea vista en clase. Dividimos el arreglo en p partes. Cada procesador obtiene el máximo de su pedazo en tiempo n/p. Al final tenemos p candidatos al máximo. Podemos hacer un torneo de tenis donde $p/2$ procesadores ven los primeros $p/2$ pares, luego $p/4$ los ganadores, etc. hasta tener sólo un ganador, el máximo. Esto toma tiempo $\log_2 p$ que es el número de rondas que hay que jugar. Luego el tiempo total es $n/p + \log_2 p$. El factor de paralelización es entonces $n/(n/p+\log_2 p)$ que es cercano a p lo que indica que es un problema altamente paralelizable. Este algoritmo tiene un óptimo para $p = n\ln(2)$, dando un tiempo de $\log_2 n + O(1)$, bastante mejor que el caso secuencial (aunque usa muchos procesadores así que no es muy eficiente en ese sentido).

b) Una empresa compra un nuevo computador paralelo de memoria distribuida donde los procesadores y los canales de comunicación son r veces más rápidos. Sin embargo, el número de procesadores y la topología de la red de interconexión son diferentes, proveyendo el sistema operativo una simulación de la topología antigua con carga c, congestión ℓ y dilatamiento d. Si un algoritmo tenía en el peor caso tiempo $Tp + Tc$ (procesamiento más comunicación) ¿Que tiempo en el peor caso tendrá en el nuevo computador?

   El tiempo de procesamiento será r veces más rápido, pero también c veces más lento en el peor caso. El tiempo de comunicación en el peor caso será $\ell\ d$ más lento y también r veces más rápido. Por lo tanto el nuevo tiempo es $(cT_p + \ell\ dT_c)/r$.

P3 (2 ptos.)

El superstring más corto de un conjunto de strings (SCS), es aquel string de largo mínimo que contiene a todos los otros como substrings. Diseñe un algoritmo eficiente que obtenga una solución subóptima para este problema cuando tenemos k strings de igual largo ℓ e indique la complejidad temporal y una cota superior al factor de aproximabilidad del mismo.

Una posible forma de hacerlo es maximizar los prefijos que son sufijos de otros strings, despues de eliminar strings repetidos. Para esto podemos usar un trie donde almacenamos todos los sufijos de los strings anotando en cada nodo si alguno de ellos coincide con uno o más de los strings completos. Luego vamos generando el SCS aproximado usando los nodos con mayor traslape posible uniendo los pedazos hasta que hayamos cubierto todos los strings. El tiempo de este algoritmo será a lo más cuadrático en $k\ell$. (1.0)

El factor de aproximación siempre se puede acotar por el caso trivial de pegar los k strings y en el peor caso la solución óptima tiene largo ℓ, con lo que el factor de aproximación es a lo más k. (0.5)

Una variante de este problema es el superstring más corto que contiene a todos los strings como secuencias (es decir, las letras aparecen en el mismo orden pero no necesariamente contiguas). ¿Es este problema más o menos difícil de aproximar?

Debiera ser más fácil de aproximar porque tiene un grado de libertad más que el problema anterior (las letras pueden estar separadas). (0.5)