Ejercicios Tema 2

Algoritmia para problemas difíciles

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura – Universidad de Zaragoza

25 de octubre de 2018

Ejercicios sobre algoritmos probabilistas

Ejercicio 1 Sea Max-3SAT el siguiente problema de optimización:

Entrada: Un circuito booleano en CNF C con una única salida y con exactamente 3 literales por cláusula, además las cláusulas no tienen variables repetidas.

Salida: Una asignación de las variables que hace ciertas el máximo número posible de cláusulas.

Dar un algoritmo probabilista eficiente que calcule una asignación que se acerque a la óptima y razonar para cada C cuántas cláusulas satisface en media la asignación calculada. **Notación:** Un *literal* es una variable booleana afirmada o negada (p. ej. x, $\neg y$, etc). Una *cláusula* es una disyunción de literales (p. ej. $(x \vee \neg y)$). Un *circuito en CNF* es una conjunción de cláusulas (p. ej. $(x \vee \neg y \vee z) \wedge (\neg z \vee y) \wedge (\neg x)$). Una asignación de las variables satisface una cláusula cuando la hace cierta, una asignación de las variables satisface un circuito cuando hace ciertas todas sus cláusulas.

Ejercicio 2 Sean A y B dos algoritmos probabilistas eficientes para un mismo problema decisional. El algoritmo A es p-correcto y además su respuesta está garantizada cuando la respuesta correcta es *verdadero*; el algoritmo B es q-correcto y además su respuesta está garantizada cuando la respuesta correcta es *falso*. Mostrar la manera de combinar A y B en un algoritmo probabilista eficiente para resolver el mismo problema con menor probabilidad de error. ¿Cuánto tiempo tarda el nuevo algoritmo? ¿Qué probabilidad de error tiene?

Ejercicio 3 (Vale hasta el 125 %) *3-Coloreado* es un problema decisional (decidir si un grafo G se puede colorear con 3 colores), pero podemos considerar el siguiente problema de optimización asociado a *3-Coloreado*:
Supongamos que nos dan un grafo G y queremos colorear cada nodo con uno de tres posibles colores, aunque no podamos dar colores distintos a vértices adyacentes. En lugar de eso, decimos que la arista (u, v) se satisface si los colores asignados a u y v son diferentes.
Considerar un 3-coloreado que maximiza el número de aristas satisfechas y llamemos c a este número. Dar un algoritmo en tiempo polinómico que calcule un 3-coloreado que satisfaga al menos $2/3 \cdot c$ aristas. Si se desea puede ser un algoritmo probabilista, en ese caso el número *medio* de aristas satisfechas debe ser al menos $2/3 \cdot c$.

Ejercicio 4 Implementar y experimentar un posible test de primalidad basado en el pequeño teorema de Fermat y los falsos testigos de primalidad. ¿Cuántos números clasifica mal con alta probabilidad? ¿Cuáles? **Nota:** Con este ejercicio hay que entregar el código realizado y la lista de pruebas ejecutadas, además de una explicación completa de los resultados.

Ejercicio 5 Sea $T(1..n)$ un vector de n elementos. Se dice que x es el dato mayoritario en T si $|\{i|T(i) = x\}| > n/2$. Se dice que el vector T es mayoritario si contiene un elemento mayoritario.
Diseñar un algoritmo de Monte Carlo para determinar si un vector es mayoritario basado en la selección aleatoria de un elemento y la comprobación de si ese elemento es mayoritario o no. Argumentar sobre la

probabilidad de error del algoritmo. Basado en el algoritmo anterior, diseñar otro con tiempo de ejecución $O(n \log(1/e))$, donde n es el número de elementos del vector y e es la probabilidad de error.
Nota: Este ejercicio es sólo interesante como ilustración de los algoritmos de Monte Carlo pues se conoce un algoritmo determinista lineal (puntuación extra por incluirlo).

Ejercicio 6 Implementar el algoritmo de Las Vegas para el problema de las n reinas que coloca las k primeras reinas aleatoriamente antes de intentar completar la solución mediante búsqueda con retroceso. Experimentar con el problema de las 39 reinas para distintos valores de k. Encontrar una solución para el problema de las 100 reinas y otra para el de las 1000 reinas. **Nota:** Con este ejercicio hay que entregar el código realizado y la lista de pruebas ejecutadas, además de una explicación completa de los resultados.

Ejercicio 7 (Vale hasta el 125 %) Supón que estás diseñando estrategias para vender mercancía en un sitio web de subastas. A diferencia de otros sitios de subasta, este usa "subasta en un paso", en el que cada oferta debe ser inmediatamente (e irrevocablemente) aceptada o rechazada. Específicamente, el sitio trabaja como sigue

- Primero el vendedor pone un artículo a la venta.
- Entonces los compradores aparecen secuencialmente.
- Cuando aparece el comprador i hace una única oferta $b_i > 0$.
- El vendedor debe decidir inmediatamente si acepta la oferta o no. Si el vendedor acepta la oferta, el artículo es vendido y todos los compradores futuros son ignorados. Si el vendedor rechaza la oferta, el comprador i se marcha y la oferta es retirada; sólo entonces el vendedor ve otros compradores.

Supongamos que un artículo se pone a la venta, y hay n compradores, cada uno con una oferta distinta. Supongamos además que los compradores aparecen en orden aleatorio y que el vendedor conoce el número n de compradores. Queremos diseñar un algoritmo con el que el vendedor tenga una probabilidad razonable de aceptar la oferta más alta. Un algoritmo supone una regla por la que el vendedor decide si aceptar cada oferta presentada, basándose sólo en el valor n y la secuencia de ofertas vistas hasta el momento.
Por ejemplo, el vendedor podría aceptar siempre la primera oferta. Esto supone que el vendedor acepta la mayor de las n ofertas con probabilidad sólo $1/n$, ya que requiere que la oferta más alta sea la primera presentada.
Dar un algoritmo con el cual el vendedor acepte la oferta más alta con probabilidad al menos $1/4$ para cualquier valor de n. (Por simplicidad, se puede suponer que n es par.) Probar que el algoritmo satisface esta condición.

Ejercicio 8 (Difícil, vale hasta el 175 %) Una lista compacta de longitud n se implementa con dos vectores, $val(1..n)$ y $ptr(1..n)$, y un entero, cabeza. El primer elemento de la lista está en $val(cabeza)$, el siguiente en $val(ptr(cabeza))$, etcétera. En general, si $val(i)$ no es el último elemento de la lista, $ptr(i)$ guarda el índice en val del siguiente elemento de la lista. Si $val(i)$ es el último elemento de la lista, entonces $ptr(i) = 0$.
Considerar una lista compacta cuyos elementos están en orden creciente, es decir, $val(i) < val(ptr(i))$ para todo $i = 1, 2, \ldots, n$ tal que $ptr(i) \neq 0$. Sea x un elemento. El problema es localizar x en la lista, es decir, dar su posición en val. El método de búsqueda dicotómica no es aplicable porque no hay forma directa de localizar el punto intermedio de la lista. Diseñar un algoritmo probabilista más rápido que la búsqueda secuencial para resolver el problema, este algoritmo debe estar basado en hacer un número $s(n)$ de elecciones probabilistas y quedarse con la mejor de ellas. Plantear la ecuación en recurrencias del tiempo medio para cada entrada según $s(n)$ (no es necesario resolverla). Probar experimentalmente distintos valores de $s(n)$. **Nota:** Con este ejercicio hay que entregar el código realizado y la lista de pruebas ejecutadas, además de una explicación completa de los resultados.

Ejercicio 9 Dado un conjunto no ordenado S y un natural k, queremos seleccionar el k-ésimo elemento de S en orden de menor a mayor. Implementar un algoritmo probabilista que resuelva el problema utilizando

pivotes aleatorios de forma similar a la ordenación vista en clase. ¿Qué k da el peor coste medio? Plantear la ecuación en recurrencias del tiempo medio para cada entrada (no es necesario resolverla). Estimar experimentalmente el tiempo medio para cada entrada. **Nota:** Con este ejercicio hay que entregar el código realizado y la lista de pruebas ejecutadas, además de una explicación completa de los resultados.

Ejercicio 10 (difícil, vale hasta el 150 %) Sea G un grafo no dirigido con n vértices y m aristas. Para un conjunto de vértices X llamamos $G[X]$ al subgrafo *inducido* por X, es decir, con vértices X y aristas las aristas de G con los dos extremos en X.
Nos dan un $k \leq n$ y queremos encontrar un conjunto de k vértices con muchas aristas, en concreto, se pide un algoritmo en tiempo polinómico que calcule, dado $k \leq n$, un conjunto X de vértices tal que $G[X]$ tenga al menos $\frac{m \cdot k(k-1)}{n(n-1)}$ aristas.
Puedes dar un algoritmo determinista o uno probabilista con tiempo medio polinómico y que sólo produce respuestas correctas.

Ejercicio 11 Diseñar un algoritmo probabilista numérico para el cálculo aproximado de π basado en el lanzamiento aleatorio de dardos contra una diana circular inscrita en un cuadrado. Experimentar con ese algoritmo. **Nota:** Con este ejercicio hay que entregar el código realizado y la lista de pruebas ejecutadas, además de una explicación completa de los resultados.

En caso de entregar alguno de estos ejercicios, la fecha límite es el jueves 8 de noviembre, preferiblemente por correo electrónico.

Antes de realizar cualquiera de estos ejercicios el alumno debe enviar un correo a `elvira@unizar.es` indicando qué ejercicio desea realizar (preferiblemente indicar varias opciones en orden de prioridad).

Cualquier fuente utilizada en la resolución de estos ejercicios debe ser indicada claramente en la solución.