Hoja 5: ejercicios variados

Algoritmia para problemas difíciles

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

22 de diciembre de 2022

Ejercicios extra

Nota: Si se realiza un ejercicio del Tema x se entenderá como perteneciente a la Hoja x (x=1,2,3), en caso de ya haber realizado uno de esa misma hoja sólo se tendrá en cuenta la mejor nota de los dos.

Ejercicio 1 (Tema 1) Un camino Hamiltoniano P es un camino que visita cada vértice exactamente una vez. Demostrar que el problema de comprobar si un grafo G contiene un camino Hamiltoniano es intratable (el camino no tiene porqué ser un ciclo). Podéis basaros únicamente en los problemas intratables que hemos visto en clase (ver transparencias del tema 1).
Encontrar un algoritmo en tiempo $O(n + m)$ (n es el número de vértices y m el número de aristas) que pruebe si un grafo dirigido acíclico G (DAG) contiene un camino Hamiltoniano. (Pista: pensar en ordenación topológica y búsqueda en profundidad.)

Ejercicio 2 (Tema 1) Demostrar que el siguiente problema es intratable:
Problema: Clique, independiente
Entrada: Un grafo G y un entero k.
Salida: ¿Contiene G un clique de k vértices y un conjunto independiente de k vértices?

Ejercicio 3 (Tema 1) Un camino simple P es un camino sin vértices repetidos. Demostrar que el problema definido a continuación es intratable: Dado un grafo G in un entero k, ¿existe un camino simple de k vértices en G?

Ejercicio 4 (Tema 3) **MOCHILA INVERSA** Tenemos n objetos, el objeto $i$ésimo tiene un peso p_i y un beneficio b_i. Esta vez el problema trata de llenar la mochila con objetos con beneficio total al menos un valor B dado (todos los pesos, los beneficios y B son reales positivos). El objetivo es **minimizar** el peso de un conjunto de objetos que tengan beneficio al menos B.

1. Diseñar un algoritmo eficiente que aproxime la solución óptima a este problema.
2. Justificar qué cota de tiempo cumple el algoritmo.
3. Justificar qué ratio de aproximación cumple dicho algoritmo.

Ambas cotas deben ser razonables (idealmente tiempo polinómico y aproximación constante o incluso mejor) pero no es necesario que sean ajustadas. **PISTAS:** (i) Se trata de usar programación dinámica asumiendo primero que los pesos son enteros, cada posición de la matriz depende únicamente de hasta 2 posiciones de la fila anterior. (ii) Se trata de cambiar de escala algunos datos, hay que pensar qué datos cambiar de escala y si son más convenientes los redondeos por exceso o por defecto.

Ejercicio 5 (Tema 3) En el problema de MAXCUT se nos da un grafo $G = (V, E)$ con un peso $w(e)$ en cada arista, y queremos separar los vértices en dos conjuntos S y $V - S$ de forma que el peso total de las aristas entre los dos conjuntos sea tan *grande* como sea posible.
Para cada $S \subseteq V$ llamamos $w(S)$ a la suma de todos los $w(e)$ en todas las aristas $e = (u, v)$ tales que

$|S \cap \{u, v\}| = 1$. Obviamente, MAXCUT trata de maximizar $w(S)$ sobre todos los subconjuntos de V. Consideremos el siguiente algoritmo de búsqueda local para MAX CUT:

```
1  comenzar con cualquier S ⊆ V
2  while hay un subconjunto S' ⊆ V tal que |S' − S| = 1 y w(S') > w(S)
3        do S = S'
```

1. Demostrar que se trata de una 2-aproximación para MAXCUT.
2. Pero, ¿es un algoritmo en tiempo polinómico?

Ejercicio 6 (Tema 3) Encontrar una implementación del algoritmo APROXSC visto en clase (página 46) con coste en tiempo $O(\sum_{i=1}^{m} |S_i|)$.

Ejercicio 7 (Tema 2) 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.

En caso de entregar alguno de estos ejercicios, la fecha límite es el domingo 8 de enero.

Antes de realizar cualquiera de estos ejercicios el alumno debe seleccionarlo en moodle.

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