Nombre y Apellido:..... Legajo N°: .....

Examen Final de Estructuras de Datos y Algoritmos

13/12/2010

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Ejercicio 4*	*NOTA*
/2.5	**/2.5**	**/2.5**	**/2.5**	

Ejercicio 1

Indicar para cada ítem si la afirmación se cumple siempre, a veces o nunca, justificando muy brevemente en cada caso:

a) La búsqueda en un árbol binario de búsqueda es más eficiente que en una lista.
b) En un grafo, para encontrar el menor camino de un nodo a otro se usa BFS
c) Un árbol B tiene mayor complejidad espacial que un árbol binario
d) Representar un grafo con listas de adyacencia ocupa menos espacio que con una matriz de adyacencia

Ejercicio 2

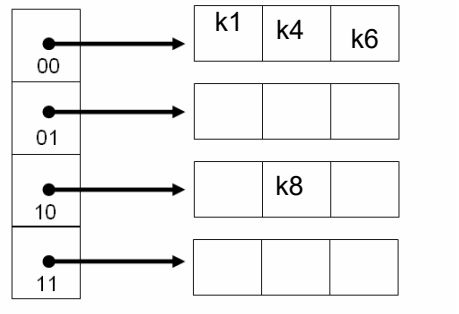
Indicar para cada ítem si la afirmación es verdadera o falsa. Justificar brevemente

a) Búsqueda binaria es un algoritmo de tipo "*Divide & Conquer"*
b) Dado un vector de N booleanos, para obtener todas las posibles combinaciones podemos utilizar un algoritmo de tipo "*backtracking"* de complejidad 2^N.
c) *Greedy* o Ávido es un tipo de algoritmo que arma todos los caminos posibles y se queda con el mejor
d) Un algoritmo basado en "Hill Climbing" tendrá complejidad a lo sumo polinómial.
e) Si los datos ya están ordenados, *BubbleSort* es más rápido que *QuickSort*

Ejercicio 3

Considerando la técnica ***Extendible Hashing*** con la siguiente tabla inicial, mostrar paso a paso el estado de la tabla al realizar las siguientes inserciones, en este orden: k1, k2, k3, ..., k8.

Considerar para el valor de hash los bits menos significativos.



Clave	Hash
k1	10000
k2	10110
k3	01110
k4	11000
k5	11100
k6	00000
k7	11110
k8	01010

100 k5

101

110 k2 k3 k7

111

Ejercicio 4

El siguiente método recibe un string y retorna una lista que contiene todas las palabras que son anagramas válidos de dicho string, donde el método *belongs* retorna true si la palabra es válida o no.

Por ejemplo, si el string fuera "maro"

podría retornar {"roma", "amor", "armo"}, pero no incluir "mar", "amo", ya que no están formadas con todas las letras del string.

a) Indicar la complejidad del algoritmo propuesto

b) Indicar si está correctamente implementado o no; en caso de contener errores corregirlo para que funcione correctamente.

```
public static List<String> anagram(String s) {
      List<String> l =  anagram("", s, null);
      return l;

}

private static List<String> anagram(String begin, String end, List<String> ans) {
      if (end.length() <= 1) {
            if ( belongs(begin + end)) {
                  ans.add(begin + end);
            }
            return ans;
      }
      for (int i = 0; i < end.length(); i++) {
            String newString = end.substring(0, i) + end.substring(i + 1);

            return anagram(begin + end.charAt(i), newString, ans);
      }
      return ans;
}
```