UNIVERSIDAD DE GRANADA
Departamento de Ciencias de la Computación e Inteligencia Artificial

**Estructuras de datos. Curso 2020-2021**
**Convocatoria extraordinaria de Febrero**
**Grado en Ingeniería Informática.**
**Doble Grado en Ingeniería Informática y Matemáticas**
**Doble Grado en Ingeniería Informática y ADE**

1. (1 punto) Se desea construir un **buscador** de productos dados por un código y el nombre del comercio en que se encuentran. Un producto puede estar en más de un comercio.
   • Dar una **representación** para el **TDA buscador** usando el tipo **map<int, set<string> >**
   • Implementar la función **insertar** que añade un producto dando el código junto con los comercios en que se puede encontrar.
   • Implementar una **función que obtenga** los comercios en que se encuentra un determinado producto.
   • Implementar la **clase iteradora** dentro de la clase buscador para poder iterar sobre todos los productos. Han de implementarse (aparte de las de la clase iteradora) las funciones **begin()** y **end()**.

2. (1.5 puntos) Implementar una función:
   **bool nullsum(list<int> &L, list<int> &L2);**

   que dada una lista L, devuelve true si hay un **rango de iteradores [p,q)**) tal que su suma de los elementos en el rango sea 0. En caso afirmativo debe retornar además el rango (uno de ellos, porque puede no ser único) a través de L2. En caso negativo L2 debe quedar vacía.

   Por ejemplo: si L=(-10, **14, -2, 7**, **-19**, -3, 2, 17, **-8, 8**) entonces debe retornar true y

   L2=(14, -2, 7, -19), ó bien L2=(-8, 8). En cambio:
   Si L=(1,3,-5) entonces debe retornar false y L2=().

3. (1 punto) Dado un bintree<int> T, implementar una función
   **void prom_nivel(bintree<int> &T, list<float> &P);**

   que genere una lista de reales P, donde el primer elemento de la lista sea el promedio de los nodos del árbol de nivel 0, el segundo sea el promedio de los de nivel 1, el tercero el promedio de los de nivel 2, y así sucesivamente. Es decir, que si el árbol tiene profundidad N, la lista tendrá N+1 elementos de tipo float.

4. (1.5 puntos) Implementar una función
   **bool parejasinset(vector< set<int> > &sw, int n);**
   que devuelva true si cada par de enteros (j; k) con 0 <= j < k; k < n está contenido en al menos uno de los conjuntos en sw[].
   P.ej si:
   sw[0] = {0; 1; 2; 3; 4};
   sw[1] = {0; 1; 5; 6; 7};
   sw[2] = {2; 3; 4; 5; 6; 7}
   parejasinset(sw,8) debe devolver **true** porque todas las parejas (j,k) 0<= j < k < 8 están en alguno de los conjuntos de sw

Por otra parte si tenemos:

sw[0] = {0; 2; 3; 4};
sw[1] = {0; 1; 5; 7};
sw[2] = {2; 3; 5; 6; 7}

entonces los pares (0; 6), (1; 2), (1; 3), (1; 4), (1; 6), (4; 5), (4; 6) y (4; 7) no están en ningún conjunto y parejasinset(sw,8) debe devolver **false.**

5. (1 punto) Determinar paso a paso las estructuras resultantes tras:

1-a **Insertar** las claves {31, 39, 43, 64, 33, 85, 50, 88, 36, 37} en una **Tabla Hash cerrada** de tamaño 13. A continuación **borrar** el 36 y el 64 y finalmente insertar el valor 74. Resolver las colisiones usando **rehashing doble**.
1-b **Insertar** las claves {24, 62, 75, 63, 85, 33, 50, 88, 36, 37, 46} en una **tabla hash abierta de tamaño 13,** resolviendo las colisiones usando árboles **AVL** en lugar de listas.

**Tiempo: 2.30 horas**