El TAD Multiconjunto

Estructuras de Datos
Facultad de Informática - UCM

En matemáticas, un *multiconjunto* es una colección de elementos, similar a un conjunto. Lo que diferencia un multiconjunto de un conjunto es que en el primero puede haber elementos duplicados. Podemos representar los multiconjuntos enumerando sus elementos entre dobles llaves (por ejemplo, $\{\{1, 6, 6, 9, 10, 10, 10\}\}$) para distinguirlos de los conjuntos, en los que las llaves son simples.

Las operaciones elementales sobre multiconjuntos (union, intersercción y diferencia) funcionan de manera similar a los conjuntos. Por ejemplo:

$$\begin{aligned} \{\{1,6,2,2\}\} \cup \{\{2,4\}\} &= \{\{1,6,4,2,2,2\}\} \\ \{\{5,5\}\} \cup \{\{1,5\}\} &= \{\{1,5,5,5\}\} \\ \{\{1,3,3,3,4\}\} \cap \{\{3,3,4,4\}\} &= \{\{3,3,4\}\} \\ \{\{10,7,7\}\} - \{\{9,9,7\}\} &= \{\{10,7\}\} \end{aligned}$$

En este ejercicio se pide implementar un TAD `MulticonjuntoChar` que represente multiconjuntos de letras mayúsculas.

```
class MulticonjuntoChar {
public:
  // Crea un multiconjunto vacío: {{ }}
  MulticonjuntoChar();

  // Determina el número de veces que se encuentra la 'letra'
  // dada en el conjunto 'this'.
  int multiplicidad(char letra) const;

  // Añade un elemento al multiconjunto
  void anyadir(char letra);

  // Elimina un elemento del multiconjunto (si existe).
  // Si no existe, no hace nada.
  void eliminar(char letra);

  // Modifica 'this' para que contenga this ∪ otro
  void unionM(const MulticonjuntoChar &otro);

  // Modifica 'this' para que contenga this ∩ otro
  void interseccion(const MulticonjuntoChar &otro);
```

```
  // Modifica 'this' para que contenga this - otro
  void diferencia(const MulticonjuntoChar &otro);
private:
  //...
};
```

1. Implementa el TAD `MulticonjuntoChar` con un array de 26 elementos que almacene mantenga la multiplicidad de cada letra.
2. Indica el coste de cada una de las operaciones.
3. Si quisiéramos implementar un TAD multiconjunto en el que los elementos sean números enteros (del rango máximo permitido por los `int`). ¿Podría utilizarse una representación similar a la que hemos utilizado para `MulticonjuntoChar`?

Solución

```
// Crea un multiconjunto vacío: {{ }}
// Coste: O(MAX_CHARS)
MulticonjuntoChar::MulticonjuntoChar() {
  for (int i = 0; i < MAX_CHARS; i++) {
    multiplicidades[i] = 0;
  }
}

// Determina el número de veces que se encuentra la ‘letra’
// dada en el conjunto ‘this’.
// Coste: O(1)
int MulticonjuntoChar::multiplicidad(char letra) const {
  assert ('A' <= letra && letra <= 'Z');
  return multiplicidades[letra - 'A'];
}

// Añade un elemento al multiconjunto
// Coste: O(1)
void MulticonjuntoChar::anyadir(char letra) {
  assert ('A' <= letra && letra <= 'Z');
  multiplicidades[letra - 'A']++;
}

// Elimina un elemento del multiconjunto (si existe).
// Si no existe, no hace nada.
// Coste: O(1)
void MulticonjuntoChar::eliminar(char letra) {
  assert ('A' <= letra && letra <= 'Z');
  if (multiplicidades[letra - 'A'] > 0) {
    multiplicidades[letra - 'A']--;
  }
}

// Modifica ‘this’ para que contenga this ∪ otro
// Coste: O(MAX_CHARS)
void MulticonjuntoChar::unionM(const MulticonjuntoChar &otro) {
  // Si el elemento X aparece N veces en ‘this‘ y aparece M veces en ‘otro‘,
  // entonces aparece N + M veces en la unión de ambos
  for (int i = 0; i < MAX_CHARS; i++) {
    multiplicidades[i] += otro.multiplicidades[i];
```

```
  }
}

// Modifica 'this' para que contenga this ∩ otro
// Coste: O(MAX_CHARS)
void MulticonjuntoChar::interseccion(const MulticonjuntoChar &otro) {
  // Si el elemento X aparece N veces en 'this' y aparece M veces en 'otro',
  // entonces aparece min(N, M) veces en la intersección de ambos
  for (int i = 0; i < MAX_CHARS; i++) {
    multiplicidades[i] = std::min(multiplicidades[i], otro.multiplicidades[i])
    ;
  }
}

// Modifica 'this' para que contenga this - otro
// Coste: O(MAX_CHARS)
void MulticonjuntoChar::diferencia(const MulticonjuntoChar &otro) {
  // Si el elemento X aparece N veces en 'this' y aparece M veces en 'otro',
  // entonces aparece N - M veces en la diferencia, pero si la diferencia es
  // negativa, entonces se queda en 0.
  for (int i = 0; i < MAX_CHARS; i++) {
    multiplicidades[i] = std::max(0, multiplicidades[i] - otro.multiplicidades
    [i]);
  }
}
```

La representación escogida es suficiente para representar conjuntos de letras, ya que hay un número limitado de ellas. Si quisiéramos implementar un TAD Multiconjunto de números enteros comprendidos en un determinado rango (por ejemplo $[0..N]$, con N relativamente pequeño), podríamos tener un array de $N + 1$ valores booleanos, y escribir una implementación similar a la vista en este ejercicio. Sin embargo, si quisiéramos almacenar un multiconjunto de enteros en un rango más amplio (por ejemplo, todos los valores permitidos por el tipo `int`), ya no podemos utilizar esta misma idea, porque el array de booleanos sería demasiado grande.