TDA *"Conjunto de enteros por extensión"*

Contrato

```
struct conjunto;
typedef struct conjunto conjunto_t;

conjunto_t *conjunto_crear();
void conjunto_destruir(conjunto_t *c);

bool conjunto_agregar(conjunto_t *c, int elem);
bool conjunto_remover(conjunto_t *c, int elem);

bool conjunto_pertenece(const conjunto_t *c, int elem);

conjunto_t *conjunto_inteseccion(const conjunto_t *a, const conjunto_t *b);
conjunto_t *conjunto_union(const conjunto_t *a, const conjunto_t *b);

// Recorre todos los elementos e de c aplicando la función f(e, data). Si
// f() devuelve false se corta la iteración.
void conjunto_recorrer(const conjunto_t *c, bool (*f)(int elem, void *data), void *data);
```

Diseño

```
struct conjunto {
    size_t n;
    int *e;
    size_t pedida;
};
```

Invariante de representación 1

`n` representa la cantidad de elementos agregados al conjunto, los elementos se guardan en `e[0..n-1]`. El vector dinámico `e` tiene pedida memoria para `pedida` enteros. (Puede aclararse: `pedida` $\neq 0$.)

(No debería ser necesario aclararlo: `n` $\leq$ `pedida`.)

Invariante de representación 2

Idem 1 pero no se admiten elementos repetidos y el vector `e` se mantiene ordenado.

Complejidad implementación 1

- `conjunto_agregar()`: $T(1)$, simplemente se inserta al final.
- `conjunto_remover()`: $T(n)$, se recorre para buscar las apariciones y se hace `swap()` con el último.
- `conjunto_pertenece()`: $T(n)$, hay que hacer una búsqueda lineal.

- `conjunto_inteseccion()`: $T(n^2)$, el algoritmo es:

  ```
  for(e in a)
      if(pertence(b, e))
          agregar(c, e)
  ```

  si asumimos que los tamaños de a y b son similares entonces buscar en b es $T(n)$ y se hace n veces (si no será $T(m \times o)$ con m tamaño de a y o tamaño de b).

- `conjunto_union()`: $T(n)$, como pueden repetirse elementos puede pegarse uno a continuación del otro.

  Ahora bien, si se exigiera que no hubieran repeticiones sería:

  ```
  for(e in a)
      agrego(c, e)
  for(e in b)
      if(! pertenece(a, e)
          agrego(c, e)
  ```

  Lo cual es $T(n^2)$ similarmente a intesección.

- `conjunto_recorrer()`: ¿Cómo se implementa recorrer para filtrar los elementos repetidos?

  Esto es un argumento fuerte para pensar que dejar que haya repetidos **es una mala idea**.

  Si no se permitieran repetidos el único orden que se modificaría es el de insertar que pasaría a ser $T(n)$ porque hay que buscar si el elemento está antes de insertarlo.

Complejidad implementación 2

- `conjunto_agregar`: $T(n)$, hay que buscar el elemento y luego insertarlo en su lugar desplazando los elementos que estén después de él.
- `conjunto_remover()`: $T(n)$, hay que buscarlo y luego desplazar.
- `conjunto_pertenece()`: $T(\log n)$, se hace por búsqueda binaria.
- `conjunto_inteseccion()`: $T(n)$, es una versión modificada de `merge()`. **Implementarla**.
- `conjunto_union()`: $T(n)$, es una versión modificada de `merge()`. **Implementarla**.
- `conjunto_recorrer()`: $T(n)$, hay que recorrer el vector, y los elementos salen en orden :).

Recorrer

Implementación

```
void conjunto_recorrer(const conjunto_t *c, bool (*f)(int elem, void *data), void *data) {
    for(size_t i = 0; i < c->n; i++)
        if(! f(c->e[i], data))
            return;
}
```

Uso

```
bool imprimir(int elem, void *d) {
    printf("%d\n", elem);
    return true;
}

bool guardar(int elem, void *d) {
    FILE *f = d; // Sabemos que d es un archivo.
    fprintf(f, "%d\n", elem);
    return true;
}

conjunto_t *conjunto;
// ...

// Imprimo:
conjunto_recorrer(conjunto, imprimir, NULL);

// Guardo en un archivo:
FILE *f = fopen("salida.txt", "wt");
conjunto_recorrer(conjunto, guardar, (void *)f);
fclose(f);
```