

Algoritmos y estructuras de datos

Tipos de datos abstractos y concretos

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas



2 de agosto de 2024

Edith Stein

Todo lo **abstracto** es ultimadamente parte de lo **concreto**.

Joseph Albers

Para mí la **abstracción** es real, probablemente más real que la naturaleza.

Edsger W. Dijkstra

Una **abstracción** es algo que representa varias cosas reales igualmente bien.

Alexander Stepanov

Aún creo en la **abstracción**, pero ahora sé que uno termina con una **abstracción** y no inicia con ella. Aprendí que uno debe adaptar la **abstracción** a la realidad y no al revés.

Tipos de datos abstractos y concretos

Abstractos

Un **tipo de datos abstracto** es un modelo matemático que se define mediante las operaciones permitidas en ciertos datos, incluyendo las precondiciones, restricciones y efectos de dichas operaciones.

Concretos

Un **tipo de datos concreto** es una implementación de un tipo de datos abstracto en un programa de computadora.

Ejemplos de tipos de datos

Tipos de datos numéricos en C

Abstracto	Concreto	Restricción	Rango
entero	signed char	8 bits con signo	$\{-2^7, \dots, 2^7 - 1\}$
entero	short	16 bits con signo	$\{-2^{15}, \dots, 2^{15} - 1\}$
entero	int	32 bits con signo	$\{-2^{31}, \dots, 2^{31} - 1\}$
entero	long long	64 bits con signo	$\{-2^{63}, \dots, 2^{63} - 1\}$
natural	unsigned char	8 bits sin signo	$\{0, \dots, 2^8 - 1\}$
natural	unsigned short	16 bits sin signo	$\{0, \dots, 2^{16} - 1\}$
natural	unsigned int	32 bits sin signo	$\{0, \dots, 2^{32} - 1\}$
natural	unsigned long long	64 bits sin signo	$\{0, \dots, 2^{64} - 1\}$
real	float	precisión simple	$\pm 3.4 \times 10^{38}$
real	double	precisión doble	$\pm 1.7 \times 10^{308}$
real	long double	precisión extendida	$\pm 1.1 \times 10^{4932}$

Ejemplo de tipo de datos abstracto y concreto

Operaciones de enteros con signo

Abajo a y b son enteros, representados con `int` a , b .

entero	entero	<code>int</code>
suma	$a + b$	$a + b$
resta	$a - b$	$a - b$
producto	$a \cdot b$	$a * b$
cociente	a / b	a / b
residuo	$a \bmod b$	$a \% b$
igualdad	$a = b$	$a == b$
desigualdad	$a \neq b$	$a != b$
menor	$a < b$	$a < b$
mayor	$a > b$	$a > b$

Tipo de datos abstracto conjunto

Operaciones de conjuntos

Abajo a es un elemento, A y B son conjuntos.

pertenencia	$a \in A$
igualdad	$A = B$
inclusión	$A \subset B$
unión	$A \cup B$
intersección	$A \cap B$
diferencia	$A \setminus B$
simétrica	$A \triangle B$
cardinalidad	$ A $
complemento	$\overline{A}$

Adicionalmente queremos crear y destruir conjuntos, agregar o eliminar elementos de un conjunto, iterar sobre los elementos de un conjunto, etc.

Conjuntos como mapas de bits

Si queremos representar un conjunto en un programa debemos saber:

- 1 La cantidad de elementos (digamos `int n`).
- 2 El tipo de sus elementos (digamos `int` de 0 a `n-1`).
- 3 Dónde almacenar el conjunto (digamos `int a[n]`).
- 4 Cómo guardar los elementos (ceros y unos en `a[0..n-1]`).

Ejemplo

```
int n = 10; // hasta diez elementos en el conjunto
int a[10] = {0, 1, 0, 1, 1, 1, 0, 1, 0, 1};
int b[10] = {1, 0, 1, 1, 1, 0, 1, 0, 1, 0};
int c[10]; // conjunto sin inicializar
```

Conjuntos como mapas de bits

Construir y destruir un conjunto $A \leftarrow \emptyset$

Para construir un mapa de bits `a` vacío, inicializamos en 0 todas las entradas de `a`:

```
void creaMapa(int n, int a[]) {  
    for (int i = 0; i < n; i++)  
        a[i] = 0; // i no esta  
}
```

Para destruirlo no es necesario hacer nada, pues `a` desaparecerá automáticamente.

Conjuntos como mapas de bits

Agregar elementos $A \leftarrow A \cup x$

Para agregar un elemento x al mapa de bits a , bastaría hacer $a[x] = 1$:

```
void agregaMapa(int n, int a[], int x) {  
    a[x] = 1; // esto puede fallar  
}
```

Pero si hubiera riesgo de que x esté fuera de rango, revisamos antes:

```
void agregaMapa(int n, int a[], int x) {  
    if (0 <= x && x < n)  
        a[x] = 1;  
}
```

Conjuntos como mapas de bits

Eliminar elementos $A \leftarrow A \setminus x$

Para eliminar un elemento x del mapa de bits a , bastaría hacer $a[x] = 0$:

```
void eliminaMapa(int n, int a[], int x) {  
    a[x] = 0; // esto puede fallar  
}
```

Pero si hubiera riesgo de que x esté fuera de rango, revisamos antes:

```
void eliminaMapa(int n, int a[], int x) {  
    if (0 <= x && x < n)  
        a[x] = 0;  
}
```

Conjuntos como mapas de bits

Pertenencia de elementos $x \in A$

Para saber si x pertenece al mapa de bits a , bastaría regresar $a[x]$:

```
int enMapa(int n, int a[], int x) {  
    return a[x]; // esto puede fallar  
}
```

Pero si hubiera riesgo de que x esté fuera de rango, revisamos antes:

```
int enMapa(int n, int a[], int x) {  
    if (0 <= x && x < n)  
        return a[x];  
    else  
        return 0;  
}
```

Conjuntos como mapas de bits

Ejercicios

- 1 Reescribe `agregaMapa`, `eliminaMapa` y `enMapa` para que informen de alguna manera razonable si `x` está fuera de rango.
- 2 Reescribe `agregaMapa`, `eliminaMapa` y `enMapa` para que usen arreglos de `bool` (`#include <stdbool.h>`) en lugar de arreglos de `int`. ¿Qué se gana?
- 3 Reescribe `agregaMapa`, `eliminaMapa` y `enMapa` para que usen un bit por elemento. ¿Qué se gana? ¿Qué se pierde? **Pista:** Usa los operadores de bits de C (`~ & | ^ << >>`).

Conjuntos como mapas de bits

Igualdad de conjuntos $A = B$

Ejemplo (A y B no son iguales)

```
int n = 10;  
int a[10] = {0, 1, 0, 1, 1, 1, 0, 1, 0, 1};  
int b[10] = {1, 0, 1, 1, 1, 0, 1, 0, 1, 0};
```

Ejemplo (A y B sí son iguales)

```
int n = 10;  
int a[10] = {0, 1, 0, 1, 1, 1, 0, 1, 0, 1};  
int b[10] = {0, 1, 0, 1, 1, 1, 0, 1, 0, 1};
```

Conjuntos como mapas de bits

Igualdad de conjuntos $A = B$

Para que dos mapas de bits a y b sean iguales, todas sus entradas deben ser iguales:

```
int igualMapa(int n, int a[], int b[]) {  
    for (int i = 0; i < n; i++)  
        if (a[i] != b[i])  
            return 0; // i esta en uno de a y b  
    return 1;  
}
```

Conjuntos como mapas de bits

Inclusión de conjuntos $A \subseteq B$

Ejemplo (A no es subconjunto de B)

```
int n = 10;  
int a[10] = {0, 1, 0, 1, 1, 1, 0, 1, 0, 1};  
int b[10] = {1, 0, 1, 1, 1, 0, 1, 0, 1, 0};
```

Ejemplo (A sí es subconjunto de B)

```
int n = 10;  
int a[10] = {0, 0, 0, 1, 1, 0, 0, 0, 1, 0};  
int b[10] = {1, 0, 1, 1, 1, 0, 1, 0, 1, 0};
```

Conjuntos como mapas de bits

Inclusión de conjuntos $A \subset B$

Para que un mapa de bits **a** sea subconjunto de otro mapa de bits **b**, lo único que puede fallar es que un elemento **i** sí esté en **a** y no esté en **b**:

```
int subMapa(int n, int a[], int b[]) {  
    for (int i = 0; i < n; i++)  
        if (a[i] == 1 && b[i] == 0)  
            return 0; // i esta en a pero no en b  
    return 1;  
}
```

Conjuntos como mapas de bits

Unión de conjuntos $C \leftarrow A \cup B$

Esta función crea un mapa de bits `c` como la unión de los mapas de bits `a` y `b`:

```
void uneMapa(int n, int a[], int b[], int c[]) {  
    for (int i = 0; i < n; i++)  
        c[i] = a[i] || b[i];  
}
```

Ejemplo

```
int a[10] = {0, 1, 0, 1, 1, 1, 0, 1, 0, 1};  
int b[10] = {0, 1, 1, 1, 0, 1, 0, 1, 0, 1};  
int c[10] = {0, 1, 1, 1, 1, 1, 0, 1, 0, 1};
```

Conjuntos como mapas de bits

Ejercicios

- 1 Escribe una función `int cardMapa(int n, int a[])` que regrese la cardinalidad de un mapa de bits.
- 2 Escribe una función `void compMapa(int n, int a[])` que complemente los elementos de un mapa de bits.
- 3 Escribe funciones `void intersectaMapa(int n, int a[], int b[], int c[])`, `void diferenciaMapa(int n, int a[], int b[], int c[])` y `void difsimMapa(int n, int a[], int b[], int c[])` que calculen la intersección, la diferencia y la diferencia simétrica de dos mapas de bits.
- 4 Escribe funciones `int minMapa(int n, int a[])` e `int maxMapa(int n, int a[])` que regresen el menor y mayor elementos de un mapa de bits, respectivamente.
- 5 En un `multiconjunto` cada elemento puede aparecer una o más veces. ¿Cómo modificar lo que hicimos para implementar multiconjuntos?

Conjuntos como mapas de bits

Resumen de resultados

Operaciones sobre un conjunto A de hasta n elementos, representado como mapa de bits.

Operación	Símbolo	Función	Tiempo
crear	$\emptyset$	<code>creaMapa</code>	n
destruir			1
agregar	$A \cup x$	<code>agregaMapa</code>	1
eliminar	$A \setminus x$	<code>eliminaMapa</code>	1
pertenencia	$x \in A$	<code>enMapa</code>	1

Conjuntos como mapas de bits

Resumen de resultados

Operaciones sobre dos conjuntos A y B de hasta n elementos, representados como mapas de bits.

Operación	Símbolo	Función	Tiempo
igualdad	$A = B$	<code>igualMapa</code>	n
inclusión	$A \subset B$	<code>subMapa</code>	n
unión	$A \cup B$	<code>uneMapa</code>	n
intersección	$A \cap B$	<code>intersectaMapa</code>	n
diferencia	$A \setminus B$	<code>diferenciaMapa</code>	n
simétrica	$A \triangle B$	<code>difsimMapa</code>	n
cardinalidad	$ A $	<code>cardMapa</code>	n
complemento	$\overline{A}$	<code>compMapa</code>	n

Conjuntos como mapas de bits

Problemas y soluciones

Problemas

- 1 Todos los elementos deben ser del mismo tipo.
- 2 Todos los elementos deben ser enteros en el rango 0 a $n - 1$.
- 3 Debemos saber al principio la cantidad máxima de elementos.
- 4 No podemos cambiar esa cantidad en tiempo de ejecución.
- 5 Algunas funciones como `uneMapa` y `cardMapa` son *muy* lentas.
- 6 Las funciones requieren pasar al menos dos parámetros por conjunto.

Soluciones

En este curso resolveremos todos estos problemas (excepto el primero). [Regresaremos una y otra vez a implementar conjuntos de diversas formas.](#)