Algoritmos y estructuras de datos

Tipos de datos abstractos y concretos

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

4 de mayo de 2015

Tipos de datos

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.

Tipos de datos

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 simples

Tipos de datos numéricos en C.

Abstracto	Concreto	Restricción
carácter	`char`	ASCII
entero	`char`	8 bits
entero	`short`	16 bits con signo
entero	`int`	32 bits con signo
entero	`long long`	64 bits con signo
natural	`unsigned`	32 bits sin signo
real	`float`	precisión simple
real	`double`	precisión doble
real	`long double`	precisión extendida

Ejemplo de tipo de datos abstracto y concreto

Operaciones de enteros con signo

entero	`int`
suma	+
resta	-
producto	*
cociente	/
residuo	%
igualdad	==
desigualdad	!=
menor	<
mayor	>

Tipo de datos abstracto conjunto

Operaciones de conjuntos

Operaciones matemáticas de conjuntos:

pertenencia	$\in$
igualdad	$=$
inclusión	$\subset$
unión	$\cup$
intersección	$\cap$
diferencia	$\setminus$
simétrica	Δ
cardinalidad	$\lvert \cdot \rvert$
complemento	$\bar{\cdot}$

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

Tipo de datos concreto: Arreglos de bits

¿Cómo representar un conjunto?

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]`).

Tipo de datos concreto: Arreglos de bits

¿Cómo representar un conjunto?

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 de un conjunto

```
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 */
```

Construir y destruir un conjunto

Crear un conjunto vacío

```
void crea(int n, int a[])
{
   int i;

   for (i = 0; i < n; i++)
      a[i] = 0; /* i no esta */
}
```

Construir y destruir un conjunto

Crear un conjunto vacío

```
void crea(int n, int a[])
{
   int i;

   for (i = 0; i < n; i++)
      a[i] = 0; /* i no esta */
}
```

Destruir un conjunto

No es necesario hacer nada.

Agregar elementos

Agregar un elemento

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

Agregar elementos

Agregar un elemento

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

Agregar un elemento

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

Eliminar elementos

Eliminar un elemento

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

Eliminar elementos

Eliminar un elemento

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

Eliminar un elemento

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

Pertenencia de elementos

Pertenencia de un elemento

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

Pertenencia de elementos

Pertenencia de un elemento

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

Pertenencia de un elemento

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

Ejercicios

Ejercicios

1. Reescribe `agrega`, `elimina` y `pertenece` para que informen de alguna manera razonable si `x` está fuera de rango.
2. Reescribe `agrega`, `elimina` y `pertenece` para que usen arreglos de `char` en lugar de arreglos de `int`. ¿Qué se gana?
3. Reescribe `agrega`, `elimina` y `pertenece` para que usen un bit por elemento. ¿Qué se gana? ¿Qué se pierde? Pista: Deberás estudiar los operadores de bits de C (`~ & | ^ << >>`).

Igualdad

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};
```

Igualdad

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};
```

Igualdad

Igualdad $A = B$

```
int igual(int n, int a[], int b[])
{
  int i;

  for (i = 0; i < n; i++)
    if (a[i] != b[i])
      return 0;
  return 1;
}
```

Inclusión

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};
```

Inclusión

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};
```

Inclusión

Subconjunto $A \subset B$

```
int subconjunto(int n, int a[], int b[])
{
  int i;

  for (i = 0; i < n; i++)
    if (a[i] == 1 && b[i] == 0)
      return 0;
  return 1;
}
```

Unión

Unión destructiva $A \leftarrow A \cup B$

Esta función agrega el contenido del segundo conjunto al primero.

```
void une(int n, int a[], int b[])
{
  int i;

  for (i = 0; i < n; i++)
    a[i] = a[i] || b[i];
}
```

Unión

Unión destructiva $A \leftarrow A \cup B$

Esta función agrega el contenido del segundo conjunto al primero.

```
void une(int n, int a[], int b[])
{
  int i;

  for (i = 0; i < n; i++)
    a[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 a[10] = {0, 1, 1, 1, 1, 1, 0, 1, 0, 1};
```

Unión

Unión no destructiva $C \leftarrow A \cup B$

Esta función construye un tercer conjunto.

```
void une(int n, int a[], int b[], int c[])
{
  int i;

  for (i = 0; i < n; i++)
    c[i] = a[i] || b[i];
}
```

Unión

Unión no destructiva $C \leftarrow A \cup B$

Esta función construye un tercer conjunto.

```
void une(int n, int a[], int b[], int c[])
{
  int i;

  for (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};
```

Ejercicios

Ejercicios

1. Escribe `cardinalidad`, que diga cuántos elementos tiene un conjunto.
2. Escribe `complemento`, que invierta los elementos de un conjunto.
3. Escribe `intersecta`, `diferencia` y `simetrica` destructivas.
4. Escribe `intersecta`, `diferencia` y `simetrica` no destructivas, es decir, el resultado debe quedar en un tercer conjunto.

Mínimo y máximo

Escribe funciones `int min(int n, int a[])` e `int max(int n, int a[])` que regresen el menor y mayor elementos de un conjunto, respectivamente.

Multiconjuntos

En un multiconjunto cada elemento puede aparecer una o más veces. ¿Cómo modificar lo que vimos hoy para implementar multiconjuntos?

Conjuntos como arreglos de bits

Resumen de resultados

Operaciones sobre un conjunto A de hasta n elementos.

Operación	Símbolo	Tiempo
crear	$\emptyset$	n
destruir		1
agregar	$A \cup x$	1
eliminar	$A \setminus x$	1
pertenencia	$x \in A$	1

Conjuntos como arreglos de bits

Resumen de resultados

Operaciones sobre dos conjuntos A y B de hasta n elementos.

Operación	Símbolo	Tiempo
igualdad	$A = B$	n
inclusión	$A \subset B$	n
unión	$A \cup B$	n
intersección	$A \cap B$	n
diferencia	$A \setminus B$	n
simétrica	$A \triangle B$	n
cardinalidad	$\lvert A \rvert$	n
complemento	$\overline{A}$	n

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 `une` y `cardinalidad` son muy lentas.
6. Las funciones requieren pasar al menos dos parámetros por conjunto.

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 `une` y `cardinalidad` 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).