Estruturas de Dados Avançadas (INF1010)

Mapas de Bits

Conjunto de Elementos

Um conjunto é um tipo de dados que representa um *universo* de elementos e operações sobre eles

- criar, destruir, inserir, remover, verificar presença de um elemento
- um conjunto é um caso particular de mapa, onde o valor é verdadeiro/falso

Algumas operações são específicas para este tipo

- união, interseção, diferença, ...

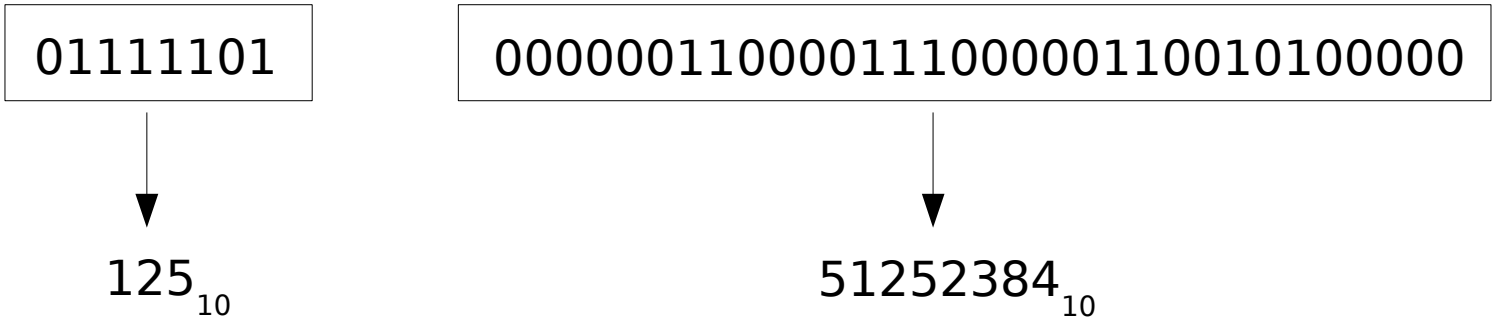
Representação da Informação

Computadores armazenam "sinais" de dois valores: 0 e 1

- binary digits ou "bits"

Agrupando sequências de bits podemos representar valores numéricos

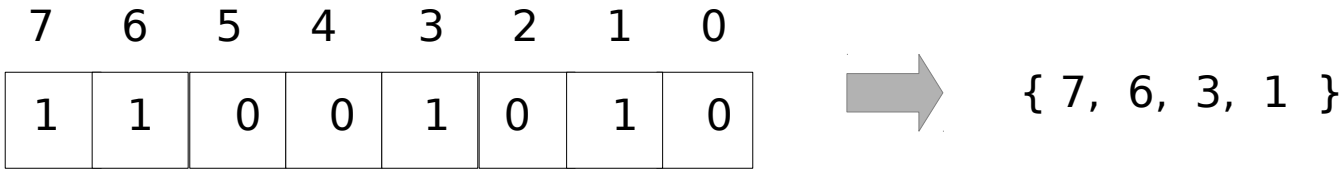
- representação em notação posicional (base 2)



Mapas de Bits

Universo de elementos de um conjunto é uma sequência finita de números naturais (inteiros não negativos)

- representação do conjunto pode ser feita com **mapas de bits**
- cada bit indica a presença ou ausência do elemento correspondente no conjunto



- manipulação da representação com operadores *bit a bit* de C

Operadores *bit a bit*

→ Operam sobre inteiros

→ Valor resultante é construído por operações sobre cada bit/par de bits

- operadores & | ~ ^
- baseados em álgebra booleana

→ Operadores de deslocamento: << >>

Operações Lógicas em C

Operadores **lógicos**: or (||), and (&&) e not (!)

- tratam qualquer argumento diferente de zero como **true** e igual a zero como **false**
- operação resulta em 1 (true) ou 0 (false)

Expressão	Resultado
`!0x41`	`0x00`
`!0x00`	`0x01`
`0xaa && 0x55`	`0x01`
`0xaa || 0x55`	`0x01`

Operadores *bit a bit* em C

Podem ser aplicados a qualquer dos tipos **inteiros**

Baseados em álgebra booleana (**bit a bit**)

A & B

&	0	1
0	0	0
1	0	1

A | B

|	0	1
0	0	1
1	1	1

~A

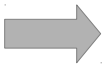
~	
0	1
1	0

A ^ B

^	0	1
0	0	1
1	1	0

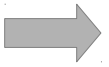
Exemplos

```
a | b
```



```
a   1111 1111 0000 0000
b   1010 0101 1010 0101
    1111 1111 1010 0101
```

```
a & b
```



```
a   1111 1111 0000 0000
b   1010 0101 1010 0101
    1010 0101 0000 0000
```

```
a ^ b
```

```
a   1111 1111 0000 0000
b   1010 0101 1010 0101
    0101 1010 1010 0101
```

```
~a
```

```
a   1111 1111 0000 0000
    0000 0000 1111 1111
```

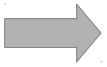
DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

Máscaras

```
a = ?
b = 1
```

```
a  ???? ???? ???? ????
b  0000 0000 0000 0001
```

```
a | b
```



```
a  ???? ???? ???? ????
b  0000 0000 0000 0001
   ???? ???? ???? ???1
```

```
a & b
```

```
a  ???? ???? ???? ????
b  0000 0000 0000 0001
   0000 0000 0000 000?
```

Deslocamento de bits

x << n

deslocamento (shift) para a esquerda de **n** bits

- bits à direita são preenchidos com 0

x >> n

deslocamento (shift) para a direita de **n** bits

- lógico: bits à esquerda são preenchidos com 0
- aritmético: bits à esquerda repetem o bit mais significativo

→ se o operando é ***unsigned***, o compilador usa o shift lógico

→ se o operando é ***signed***, é livre escolha do compilador

- mas em geral o compilador usa o shift aritmético

Exemplos

Operação	x = [01100011]	x = [10010101]
x << 4	0 0 1 1 0 0 0 0	0 1 0 1 0 0 0 0
x >> 4 (lógico)	0 0 0 0 0 1 1 0	0 0 0 0 1 0 0 1
x >> 4 (aritmético)	0 0 0 0 0 1 1 0	1 1 1 1 1 0 0 1

Construindo uma Máscara

	7	6	5	4	**3**	2	1	0
x	?	?	?	?	**b**	?	?	?

```
m = 1 << 3

res = x | m

res = x & ~m
```

máscara	binário	decimal
1	0000 0001	1
1<<1	0000 0010	2
1<<2	0000 0100	4
1<<3	0000 1000	8
1<<4	0001 0000	16
1<<5	0010 0000	32
1<<6	0100 0000	64
1<<7	1000 0000	128

Conjuntos de Inteiros

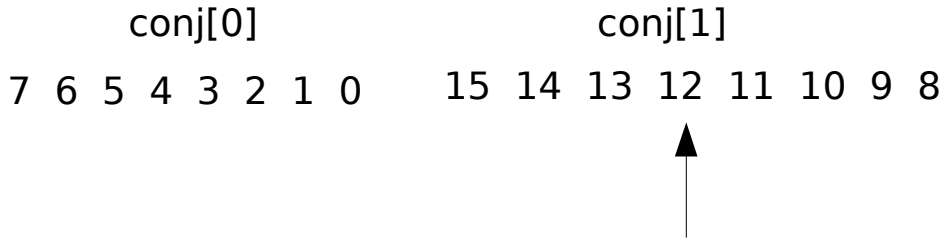
Numa plataforma de 64 bits:

- até 8 elementos: 8 bits → char
- até 32 elementos: 32 bits → int (sizeof(int) * 8 → 4 * 8)
- até 64 elementos: 64 bits → long (sizeof(long) * 8 → 8 * 8)
- mais elementos: n bits → char conj[N]

$$N = ((n - 1) / 8) + 1$$

elemento i → conj[i/8] bit i%8

elemento 12 -> conj[1] bit 4



Representação de conjuntos por mapas de bits

```
typedef struct set *Set;

/* cria um conjunto com n elementos */
Set *setCreate(void);
/* destroi (desaloca) o conjunto */
void setDestroy(Set *set);
/* mostra os elementos do conjunto */
void setShow(char* title, Set *set);
/* insere o elemento i no conjunto */
void setInsert(Set *set, int i);
/* remove o elemento i do conjunto */
void setRemove(Set *set, int i);
/* cria uma copia do conjunto */
Set *setCopy(Set *set);
/* cria a uniao de dois conjunto  */
Set *setUnion(Set *set1, Set *set2);
/* calcula a intersecao de dois conjuntos */
Set *setIntersection(Set *set1,  Set *set2);
/* calcula a diferenca de set1-set2 */
Set *setDifference(Set *set1, Set *set2);
/* verifica se o elemento i pertence ao conjunto */
int setIsMember(Set *set, int i);
/* verifica de set2 e' um sub conjunto de set1 */
int setIsSubset(Set *set1,  Set *set2);
/* verifica se dois conjuntos sao iguais */
int setIsEqual(Set *set1,  Set *set2);
/* informa a cardinalidade do conjunto */
int setNumberOfElements(Set *set);
/* cria o complemento de conjunto */
Set *setComplement(Set *set);
```