INF 1010
Estruturas de Dados Avançadas

Conjuntos (*Sets*)

Uma interface geral

```
#ifndef SET_H
#define SET_H

#typedef struct _set Set;

void set_init(Set*set, int (*match)(const void *key1, const void *key2),
              void (*destroy)(void *data));

Set* set_destroy(Set* set);

int set_insert(Set *set, const void *data);
int set_remove(Set *set, void **data);

int set_union(Set *setu, const Set *set1, const Set *set2);
int set_intersection(Set *seti, const Set *set1, const Set *set2);
int set_difference(Set *setd, const Set *set1, const Set *set2);

int set_is_member(const Set *set, const void *data);
int set_is_subset(const Set *set1, const Set *set2);
int set_is_equal(const Set *set1, const Set *set2);

int set_size(Set* set);

#endif
```

Implementações

- Listas
- Árvores
- Hash
- etc…

Um caso particular muito eficiente… a seguir.

Conjuntos como vetor de bits

supondo conjuntos de até 8 elementos

C = {a,b,c,d,e,f,g,h}

cada elemento mapeado para um bit

b7	b6	b5	b4	b3	b2	b1	b0
h	g	f	e	d	c	b	a

cada conjunto pode ser representado por um byte, "ligando" os bits correspondentes a cada elemento

M = {a,d,e,f} ↔ 0011 1001

b7	b6	b5	b4	b3	b2	b1	b0
h	g	**f**	**e**	**d**	c	b	**a**
0	0	**1**	**1**	**1**	0	0	**1**

Conjuntos como vetor de bits
(até 4 elementos)

bits	conjunto	bits (em um byte)
0000	∅	0000 0000
0001	{ a }	0000 0001
0010	{ b }	0000 0010
0011	{a, b}	0000 0011
0100	{ c }	0000 0100
0101	{a, c}	0000 0101
0111	{a, b, c}	0000 0111
1000	{ d }	0000 1000
1001	{a, d}	0000 1001
1010	{b, d}	0000 1010
1011	{a, b, d}	0000 1011
1100	{c, d}	0000 1100
1101	{a, c, d}	0000 1101
1110	{b, c, d}	0000 1110
1111	{a, b, c, d}	0000 1111

Operadores sobre bits (bitwise operators)

& AND

&	0	1
0	0	0
1	0	1

| OR

|	0	1
0	0	1
1	1	1

^ XOR

^	0	1
0	0	1
1	1	0

~ complemento

~	0	1
	1	0

>> deslocamento bit a bit para a direita

<< deslocamento bit a bit para a esquerda

Pertence?

b7	b6	b5	b4	b3	b2	b1	b0
h	g	f	e	d	c	b	a

verificar se um elemento *e* pertence a um conjunto C =

verificar se o bit correspondente a *e* está ligado

representação(C) & máscara(*e*) ≠ 0 → *e* ∈ C

representação(C) & máscara(*e*) = 0 → *e* ∉ C

exemplo

d ∈ C?

representação de C = {a,d,e,f}: 0011 1001

máscara(d): & 0000 1000

0000 1000 ≠ 0 → d ∈ M

Como construir uma máscara ?

máscara para o bit i

1. inicia com 1
2. desloca os bits para a esquerda, i vezes

m = 1 << i;

máscara para d (bit 3)

m = 1 << 3;

m ← 0000 1000

b7	b6	b5	b4	**b3**	b2	b1	b0
h	g	f	e	**d**	c	b	a

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

Como inserir um elemento num conjunto?

ligando o bit correspondente

conjunto OR máscara(bit)

exemplo

C = {a,b}

C = C ∪ {d}

C = C | d;

b7	b6	b5	b4	**b3**	b2	b1	b0
h	g	f	e	**d**	c	b	a

C	0	0	0	0	0	0	**1**	**1**
d	0	0	0	0	**1**	0	0	0
C | d	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**

União de conjuntos

conjuntoA OR conjunto B

exemplo

A = {a,c,d,f}

B = {b,c,d}

C = A ∪ B

C = {a,b,c,d,f}

C = A | B;

	b7	b6	b5	b4	b3	b2	b1	b0
	h	g	f	e	d	c	b	a
A	0	0	1	0	1	1	0	1
B	0	0	0	0	1	1	1	0
A | B	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**

Interseção de conjuntos

conjuntoA AND conjunto B

exemplo

A = {a,c,d,f}

B = {b,c,d}

C = A ∩ B

C = {c,d}

C = A & B;

b7	b6	b5	b4	b3	b2	b1	b0
h	g	f	e	d	c	b	a
A	0	0	1	0	1	1	0	1
B	0	0	0	0	1	1	1	0
A & B	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

Complemento

C = ~A;

exemplo

A = {a,c,d,f}

~A = {b,e,g,h}

b7	b6	b5	b4	b3	b2	b1	b0
h	g	f	e	d	c	b	a

A	0	0	1	0	1	1	0	1
~A	1	1	0	1	0	0	1	0

Conjuntos de mais elementos

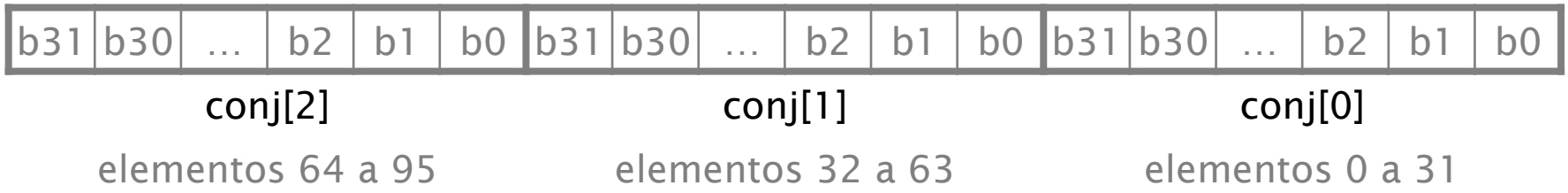
byte: até 8 elementos

int: até sizeof(int) elementos

mais elementos:

int conj[N]; /* N=ceiling(num_elementos / sizeof(int))*/

int conj[3]; /* 80 elementos; sizeof(int) = 32 */



Funções de acesso a um bit

```
int tam = sizeof(int)*8; /* número de bits */

int getbit(int* conj, int i) {
    return ( conj[i / tam] & ( 1 << (i % tam) ) != 0);
}

Void setbit(int* conj, int i) {
     conj[i/tam] |= 1<<(i%tam);
}
```

índice no vetor

máscara de bit

```
char x = getbit(34);
```

conj[2] elementos 64 a 95

conj[1] elementos 32 a 63

conj[0] elementos 0 a 31

Uma interface para conjuntos quaisquer

```
/* EDA: bitvet.h */
/* ADT: Sets where  pertinece is related with a bit in a Bit Vector */


#ifndef _bitvet
#define _bitvet

typedef struct _bitvector BitVector;

BitVector* bvInit(int mx);

int bvGet(BitVector* v,int i);

void bvSet(BitVector* v,int i);

void bvShow(BitVector* v, char* title);

BitVector* bvIntrsec(BitVector* a, BitVector* b);


#endif
```

```
struct _bitvector  {
       int max;
       int *vector;
};


BitVector* bvInit(int max) {
       int i,size = sizeof(int);
       int num=(max-1)/size+1;     /*  numero de ints para max bits */

        BitVector* bv=(BitVector*)malloc(sizeof(BitVector));
       bv->max=max;
       bv->vector=(int *)malloc(num*sizeof(int));
       for (i=0; i<num; i++)
              bv->vector[i]=0;
       return bv;
}
```

```
int bvGet(BitVector* bv,int i){
  int res=(bv->vector[i/sizeof(int)])&(1 <<i%sizeof(int));
       return (res!=0);
}


void bvSet(BitVector* bv,int i){
       bv->vector[i/sizeof(int)] |= 1 << i%sizeof(int);
}
```

```
BitVector* bvIntrsec(BitVector* a, BitVector* b){
  int i,minab=(a->max<b->max)?a->max:b->max;
  int num=(minab-1)/sizeof(int)+1; /* numero de ints para max bits */

  BitVector* bv=(BitVector*)malloc(sizeof(BitVector));

  bv->max=minab;

  bv->vector=(int*)malloc(num*sizeof(int));

  for (i=0; i<num; i++)
      bv->vector[i]=(a->vector[i])&(b->vector[i]);

  return bv;
}
```