SCC 202 - Algoritmos e Estruturas de Dados I

TAD: Tipo Abstrato de Dados (3)
Implementação do TAD SET

12/8/2010

Exercício: TAD Conjuntos (SET)

- Um conjunto é uma coleção de membros (ou elementos); cada membro ou é um conjunto ou um elemento primitivo chamado de átomo.

- Todos os membros são diferentes: nenhum conjunto contém 2 cópias do mesmo elemento.

- Ex:

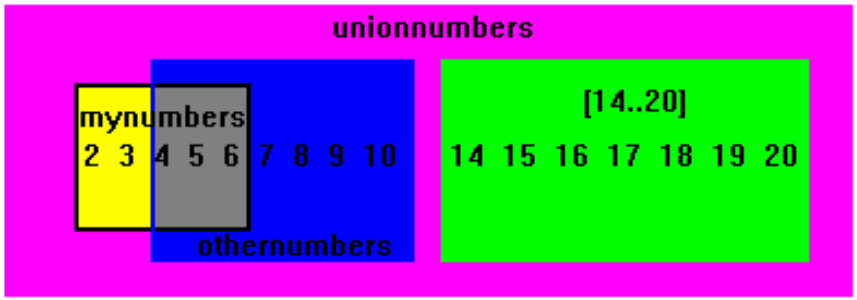
{1,4} ok

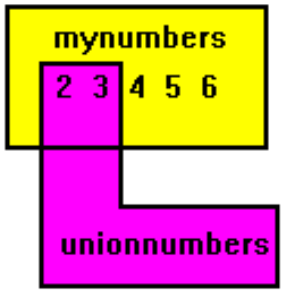
{1,4,1} não ok

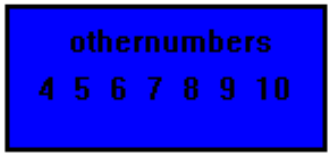
Operações básicas: união, intersecção e diferença

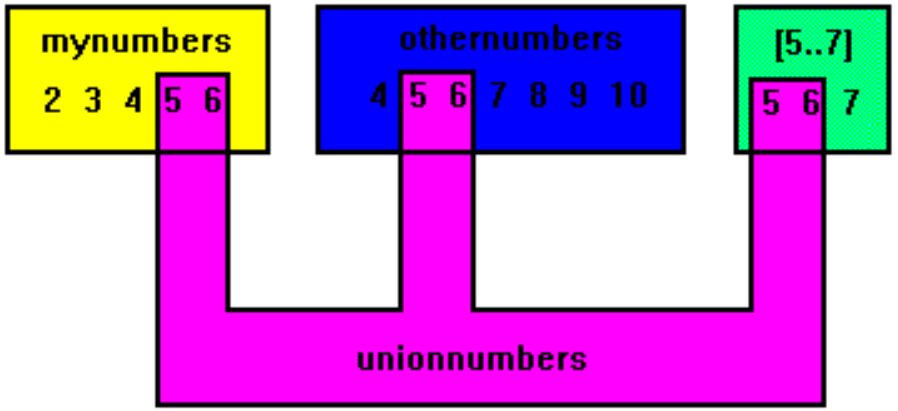
- Se A e B são conjuntos, então $A \cup B$ é o conjunto de elementos que são membros de A ou de B ou de ambos
- Se A e B são conjuntos, então $A \cap B$ é o conjunto de elementos que estão em A e em B
- Se A e B são conjuntos, então $A - B$ é o conjunto de elementos em A que não estão em B
- Exemplo: $A = \{a,b,c\}$ $B = \{b,d\}$
  $A \cup B = \{a,b,c,d\}$
  $A \cap B = \{b\}$
  $A - B = \{a,c\}$

unionnumbers: União, Diferença e Intersecção









Conjuntos em C

- Como implementar elegantemente um conjunto de inteiros de um intervalo predefinido em C?

- Vejam que este TAD é muito mais restrito do que um que manipule conjuntos em geral,

  pois todos os conjuntos criados poderão ter, no máximo, de 0..TAM elementos. Assim, a união de 2 conjuntos terá, no máximo, TAM elementos.

Conjuntos em C

- Pode-se representar de forma eficiente via um vetor de booleanos (também conhecido por bitarray):

```
# define N 100 //por exemplo, conjunto que tem números de 0 a 99
int conjunto[N]; //conjunto[i]=1 se i está no conjunto; 0, caso contrário
```

Operações?

Operações usuais

- União(A,B,C)
- Intersecção(A,B,C)
- Diferença(A,B,C)
- Membro(x,A)
- Cria_Conj_Vazio(A)
- Insere(x,A)
- Remover(x,A)
- Atribui(A,B)
- Min(A)
- Max(A)
- Igual(A,B)
- Destroi_conjunto(A)
- Tamanho(A)
- Imprime_conjunto(A) (fazer)

Definição das operações

- União(A,B,C): toma os argumentos A e B que são conjuntos e retorna A $\cup$ B à variavel C
- Intersecção(A,B,C): toma os argumentos A e B que são conjuntos e retorna A $\cap$ B à variavel C
- Diferença(A,B,C): toma os argumentos A e B que são conjuntos e retorna A - B à variavel C
- Membro(x,A): toma o conjunto A e o objeto x cujo tipo é o tipo do elemento de A e retorna um valor booleano – true se x $\in$ A e false caso contrário
- Cria_Conj_Vazio(A): faz o conjunto vazio ser o valor para a variável conjunto A. Primeira operação do conjunto.

Definição das operações

- Insere(x,A): toma o conjunto A e o objeto x cujo tipo é o tipo do elemento de A e faz x um membro de A. O novo valor de A = A $\cup$ {x}. Se x já é um membro de A, então a operação insere não muda A

- Remover(x,A): remove o objeto x, cujo tipo é o tipo do elemento de A, de A. O novo valor de A = A – {x}. Se x não pertence a A então a operação remove não altera A

- Destroi_Conj(A): remove o conjunto criado da memória.

- Imprime_conjunto(A): Imprime os elementos na forma de conjunto: { x, y, z }

Definição das operações

- Atribui(A,B): Seta o valor da variável conjunto A = ao valor da variável conjunto B
- Min(A): retorna o valor mínimo no conjunto A. Por exemplo: Min({2,3,1}) = 1 e Min({'a','b','c'}) = 'a'
- Max(A): Similar a Min(A) só que retorna o máximo do conjunto
- Igual(A,B): retorna true se e somente se os conjuntos A e B consistem dos mesmos elementos
- Tamanho(A): retorna o número de elementos do conjunto A

Exercício

- Em duplas, implementem em C o TAD conjunto de números inteiros com elementos do universo 0..TAM - 1
  - Use um arquivo .h

```
#define TAM $$$ // trabalha com elementos do universo 0..TAM -1

struct conjunto {
        int* v; //vetor booleano que armazenará o conjunto sendo que
                //o índice armazena o valor sendo true se o elemento está
                // no conjunto, false caso contrário
}
```

TADs em C: Exemplo

```
/* TAD: conjunto*/

/* Tipo e Tamanho Exportados */
#define TAM 10 // trabalha com elementos do universo 0..TAM -1
typedef struct conjunto Conjunto;


/* Funções Exportadas */


/* Função união – Une os elementos do conjunto A e B em um
   conjunto C. */
void uniao (Conjunto *A, Conjunto *B, Conjunto *C);
/* Função Intersecção – Armazena em C os mesmos elementos que
   estão no conjunto A e B*/
void interseccao (Conjunto *A, Conjunto *B, Conjunto *C);
/* Função Destroi_Conj – Libera a memória de um conjunto */
void Destroi_Conj (Conjunto *A);
```

Arquivo conjunto.h

TADs em C: Exemplo

```
/* Continuação... */

/* Função diferença – atribui ao conjunto C a diferença entre os
   conjuntos A e B  */
void diferença(Conjunto *A, Conjunto *B, Conjunto *C);
/* Função membro – verifica se o elemento elem está no Conjunto
   A. Retorna 1 se está e 0 caso não esteja.  */
int membro(int x, Conjunto *A);
/* Função Cria_Conj_Vazio – Cria um conjunto vazio e retorna o
   conjunto criado. A variável erro retorna 0 se o conjunto foi
   criado corretamente e 0 caso contrario. Deve ser usado como
   primeira operação sobre um conjunto. */
Conjunto *Cria_Conj_Vazio (int *erro);
/* Função insere – insere o elemento elem no conjunto A e
   retorna se a execução foi realizada com sucesso(1) ou
   não(0)*/
int insere(int x, Conjunto *A);
```

TADs em C: Exemplo

```
/* Continuação... */
/* Função remove (diferente da de PASCAL) - remove o
   elemento elem do Conjunto A, retorna 1 se o elemento
   foi retirado e 0 se o elemento não está no conjunto */
int remover(int x, Conjunto *A);
/* Função Atribui - faz a copia do conjunto A para o B*/
void atribui(Conjunto *A, Conjunto *B);
/* Função min - retorna o menor elemento do conjunto A -
   se o conjunto está vazio retorna TAM */
int min(Conjunto *A);
```

TADs em C: Exemplo

```
/* Continuação... */

/* Função max – retorna o maior elemento do conjunto A - se o
   conjunto está vazio retorna TAM */
int max(Conjunto *A);
/* Função igual – verifica se o conjunto A é igual a
Conjunto B. Retorna 1 se igual e 0 se diferente.   */
int igual(Conjunto *A, Conjunto *B);

/* Função tamanho – retorna o tamanho do conjunto A (0..TAM-1)
   */

int tamanho(Conjunto *A);

/* Função testa_vazio – verifica se o conjunto A é vazio.
   Retorna 1 se for vazio e 0 caso contrario*/

int testa_vazio(Conjunto *A);
```

TADs em C: Exemplo

```
#include <stdlib.h> /* malloc, free, exit */
#include <stdio.h> /* printf */
#include "conjunto.h"


struct conjunto {
        int* v;  //vetor booleano que armazenará o conjunto sendo que
         //o índice armazena o valor sendo true se o elemento está
         // no conjunto, false caso contrário
};

void Destroi_Conj (Conjunto *A){
   free(A->v);
   free(A);
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */

void uniao (Conjunto *A, Conjunto *B, Conjunto *C) {
   int i;// variável auxiliar para realização do loop

   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) || (B->v[i]==1) {
        C->v[i]=1;
        }
   }
}
void interseccao(Conjunto *A, Conjunto *B, Conjunto *C) {
   int i; // variável auxiliar para realização do loop

   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) && (B->v[i]==1) {
        C->v[i]=1;
        }
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */
/* faz o conjunto vazio ser o valor para a variável conjunto A. Deve ser
   usado como primeira operação sobre um conjunto. Retorna erro = 0 se
   sucesso e 1 se falhou.
*/
Conjunto* Cria_Conj_Vazio(int *erro) {
   int i; // variável auxiliar para realização do loop

   Conjunto* conj = (Conjunto*) malloc(sizeof(Conjunto));
   if (conj == NULL) {
        *erro= 1;
        exit(1);
   }
   conj->v = (int*)malloc(TAM*sizeof(int));
   for(i=0;i<TAM;i++){
     conj->v[i]=0;
   }
   *erro = 0;
   return conj;
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */
int insere(int x, Conjunto *A) {
   if (x>=TAM || x<0 ) {
        return 0;
   }
   A->v[x]=1;
   return 1;
}

int membro(int x, Conjunto *A) {
   if (x>=TAM || x<0 ||(A->v[x]==0)) {
     return 0;
   }
   return 1;
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */
int remover(int x, Conjunto *A) {
   if (x>=TAM || x<0 ||(A->v[x]==0)) {
        return 0;
   }
   A->v[x]=0;
   return 1;
}

void atribui(Conjunto *A, Conjunto *B) {
   int i; // variável auxiliar para realização do loop

   for(i = 0; i<TAM;i++){
     B->v[i]=A->v[i];
   }
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */

int min (Conjunto *A) {
    int i;
    for(i = 0; i<TAM;i++){
      if (A->v[i]==1) {
          return i;
      }
    }
    return TAM; // condição de conjunto vazio
}

int max (Conjunto *A) {
    int i;
    for(i = TAM - 1; i>-1;i--){
      if (A->v[i]==1) {
          return i;
      }
    }
    return TAM; // condição de conjunto vazio
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */

void diferenca(Conjunto *A, Conjunto *B, Conjunto *C){
   int i; // variável auxiliar para realização do loop

   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) && (B->v[i]==0) {
        C->v[i]=1;
        }
}

int igual(Conjunto *A, Conjunto *B){
   int i;
   for(i = 0; i<TAM;i++){
     if (A->v[i]!=B->v[i]) {
        return 0;
     }
   }
   return 1;
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */

int tamanho(Conjunto *A){
    int i,tam; // variável auxiliar para realização do loop e para a
    // verificação do tamanho do conjunto
    tam = 0;
    for(i = 0; i<TAM;i++){
      if (A->v[i]==1) {
        tam++;
      }
    }
    return tam;
}

int testa_vazio(Conjunto *A);{
    int i;
    for(i = 0; i<TAM;i++){
      if (A->v[i]==1) {
          return 0;
      }
    }
    return 1;
}
```

Arquivo conjunto.c

Possíveis melhorias

Uso de calloc em

```
Conjunto* Cria_Conj_Vazio(int *erro)
```

Para alocar o espaço do vetor v, pois ele aloca espaço como malloc e já atribui 0 para todos os bits dos bytes alocados.

Uso de Cria_Conj_Vazio

dentro das rotinas de união, interseção e diferença, para criar o conjunto C com todos os elementos zerados e assim garantir que tais operações irão funcionar corretamente. O TAD proposto "pede" a criação de um conjunto antes de seu uso, mas na solução acima isto estaria realmente acontecendo.