Instituto Tecnológico de Aeronáutica - ITA
Divisão de Ciência da Computação - IEC

CES-11 Algoritmos e Estruturas de Dados
Laboratório 01 – 2o Período de 2020
Denis Loubach | `dloubach@ita.br`

1 Armazenamento de conjuntos de números inteiros

Conjuntos de números inteiros podem ser armazenados em listas lineares.
Seja o seguinte conjunto universo:

$$U_S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\} \quad (1)$$

Pode haver conjuntos menores formados por elementos não repetidos deste conjunto.
Exemplo, $C = \{7, 16, 3, 9, 12\}$ e seu armazenamento numa lista linear:

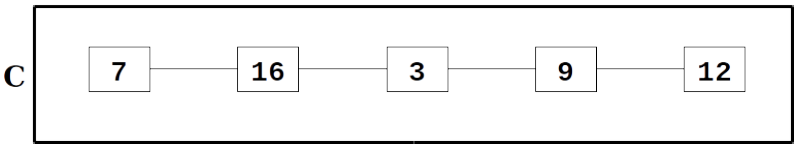

Figura 1: Lista linear

A entrada de um conjunto como dado em um programa envolve:

1. A digitação e leitura, ou leitura de um arquivo pré-definido;
2. A filtragem das duplicatas e dos elementos que não pertencem ao conjunto universo U_S; e
3. O armazenamento numa estrutura de dados.

1.1 O `TAD-lista-conjunto`

Definir a estrutura de dados (`set_t` e `position_t`) e escrever as seguintes funções operadoras do `TAD-lista-conjunto`, considerando a implementação com estrutura contígua.

- `void adt_initSet(set_t *)`
  Inicializa uma lista-conjunto vazia pronta para novas inserções.
- `position_t adt_first(set_t)`
  Retorna a posição do primeiro elemento da lista-conjunto.
- `position_t adt_after(position_t)`
  Retorna a posição do elemento da lista-conjunto seguinte a posição do elemento indicada no parâmetro da função.
- `position_t adt_last(set_t)`
  Retorna a posição do último elemento da lista-conjunto.
- `int adt_element(set_t, position_t)`
  Retorna o elemento da lista-conjunto da posição indicada no parâmetro da função.
- `position_t adt_position(set_t, int)`
  Retorna a posição na lista-conjunto do elemento indicado no parâmetro da função.
- `void adt_insertLast(set_t *, int)`
  Insere o elemento indicado no parâmetro da função no final da lista-conjunto.

Fora do escopo do TAD, deverão ser elaboradas *funções auxiliares* tais como:

- `set_t getFilteredSet(void)`
  Recuperação (pode ser de um arquivo pré-definido ou digitação) e armazenamento dos elementos de um conjunto, eliminando os elementos que não pertencerem ao conjunto U_S bem como os elementos repetidos. Por final, retorna o conjunto armazenado.

- `void printSet(set_t)`
  Exibe os elementos do conjunto.

- `void errorMessage(char *)`
  Exibe uma mensagem conforme o parâmetro da função.

Tais funções auxiliares devem ser *independentes da estrutura*, portanto deverão utilizar somente os operadores pré-definidos para o TAD para manipular a lista-conjunto.

1.2 União de dois conjuntos

Sejam os seguintes conjuntos C_1 e C_2.

$$C_1 = \{4, 5, 12, 10, 8\} \tag{2}$$
$$C_2 = \{3, 12, 10, 7, 4, 9\} \tag{3}$$

A união U é dada por:

$$U = Uniao(C_1, C_2) = \{4, 5, 12, 10, 8, 3, 7, 9\} \tag{4}$$

O cálculo do conjunto união U inclui:

1. Copiar C_1 em U; e
2. Inserir em U todo elemento de C_2 que não estiver presente em C_1.

Acrescentar a função `void adt_copySet(set_t *, set_t)` aos operadores do `TAD-lista-conjunto`.
Acrescentar a função `set_t unionSet(set_t, set_t)` às funções auxiliares independentes da estrutura.

1.3 Funções auxiliares adicionais

Incluir no programa as seguintes funções auxiliares que devem ser igualmente independentes da estrutura interna do `TAD-lista-conjunto`, assim como a função de união.

1. Interseção de dois conjuntos, exemplo:
   `set_t interSet(set_t, set_t)`

$$Intersecao(\{4, 5, 12, 10, 8\},\ \{3, 12, 10, 7, 4, 9\}) = \{4, 12, 10\} \tag{5}$$

2. Diferença de dois conjuntos, exemplo:
   `set_t diffSet(set_t, set_t)`

$$Diferenca(\{4, 5, 12, 10, 8\}, \{3, 12, 10, 7, 4, 9\}) = \{5, 8\} \tag{6}$$

3. Complemento de um conjunto em relação ao U_S, exemplo:
   `set_t compSet(set_t, set_t)`

$$Complemento(\{4, 5, 12, 10, 8, 3, 7, 9\}, U_S) = \{1, 2, 6, 11, 13, 14, 15, 16, 17, 18, 19, 20\} \tag{7}$$

1.4 Estrutura encadeada

Apresentar uma *segunda versão do programa* substituindo a estrutura contígua pela estrutura encadeada, alterando no programa exclusivamente a definição/declaração e o corpo das funções no escopo do TAD.

1. A estrutura de listas lineares deverá utilizar um ponteiro para o nó-líder e outro para o último nó;
2. Não é permitido alterar a função `main`; e
3. Não é permitido alterar os protótipos das outras funções indicadas anteriormente.

1.5 Execução esperada

O programa deve considerar, minimamente, a entrada e armazenamento de dois conjuntos, C_1 e C_2, conforme requisitos anteriores, além de exibir:

- O conteúdo dos conjuntos armazenados; e
- O resultado da união, interseção, diferença e complemento dos conjuntos armazenados.

As duas versões do programa, a primeira considerando estrutura contígua e a segunda estrutura encadeada, deverão executar esses requisitos.

Utilizar somente bibliotecas padrão da linguagem C, tais como `stdio.h`, `stdlib.h`.

2 Sugestão de abordagem

Dividir a implementação em arquivos fonte (`.c` e `.h`) diferentes para cada tipo de implementação no que se refere especificamente ao TAD, ou seja, para estrutura contígua utilizar `vectorAdtSet.{c, h}` e para estrutura encadeada utilizar `nodeAdtSet.{c, h}`.

Assim, para ter duas versões distintas do programa, será necessário apenas trocar no arquivo principal (que contém o `main`) o nome do arquivo `.h`, que por sua vez que contém todos os protótipos dos operadores e as definições da estrutura utilizada.

Exemplo:
`#include ‘‘vectorAdtSet.h’’` – para considerar a implementação com estrutura contígua, OU
`#include ‘‘nodeAdtSet.h’’` – para considerar a implementação com estrutura encadeada.

Um exemplo deste tipo de sugestão de abordagem pode ser encontrado em `https://github.com/dloubach/c-intro-examples/tree/master/conditional-compilation`.

3 Entregas

A entrega deverá ser realizada conforme enunciado no *Google Classroom*.

Referências

[1] F. C. Mokarzel, “Apostila de ces-11 estruturas de dados em slides.” ITA, 2011.

[2] F. C. Mokarzel, “Roteiro para aulas práticas em slides.” ITA, 2011.