Programação 1: Conjuntos Dinâmicos e Aplicações

Prof. André Grégio

Conjuntos Dinâmicos

De acordo com Cormen et al.[1], conjuntos [de dados] manipulados por algoritmos podem aumentar, diminuir, ou mudar ao longo do tempo, sendo portanto chamados de **conjuntos dinâmicos**. Os conjuntos dinâmicos possuem nomes especiais conforme as operações que podem ser realizadas sobre eles (por ex., deve-se poder inserir e remover elementos de um "dicionário", bem como verificar se um elemento pertence a ele ou não).
Em geral, cada elemento de um conjunto dinâmico é um objeto estruturado cujos atributos podem ser acessados por meio de apontadores a ele. Para alguns objetos, esse atributo é a chave—um dicionário pode ser visto como um conjunto de chaves com valores associados (ou dados satélite). Se o objeto for um "nó" composto por por atributos como "valor" e "próximo nó", o primeiro é o dado satélite e o segundo é um apontador para outro objeto do tipo nó. Certos conjuntos dinâmicos assumem que seus dados estão **ordenados**, o que permite operações específicas para se definir elementos específicos: maior, menor, próximo, etc.
As operações que podem ser feitas sobre conjuntos dinâmicos podem ser resumidas em consultas—operações "passivas" que retornam alguma informação sobre o conjunto—e alterações—operações que "interferem" no conjunto. Uma lista não definitiva de operações sobre conjuntos dinâmicos é apresentada abaixo:
Consultas

- `BUSCA(S,k)`: dado um conjunto S e um valor de chave k, retorna um apontador x para um elemento k do conjunto ($x.chave \rightarrow k$) ou NULO.
- `MÍNIMO(S)`: dado um conjunto ordenado S, retorna um apontador para o elemento do conjunto com a menor chave.
- `MÁXIMO(S)`: dado um conjunto ordenado S, retorna um apontador para o elemento do conjunto com a maior chave.
- `SUCESSOR(S,x)`: dado um elemento x em que a chave pertence a um conjunto ordenado S, retorna um apontador para o próximo maior elemento do conjunto ou NULO.
- `PREDECESSOR(S,x)`: dado um elemento x em que a chave pertence a um conjunto ordenado S, retorna um apontador para o próximo menor elemento do conjunto ou NULO.

Alterações

- `INSERÇÃO(S,x)`: Adiciona um elemento apontado por x ao conjunto S, aumentando seu tamanho.
- `REMOÇÃO(S,x)`: Retira o elemento apontado por x do conjunto S, caso este exista, diminuindo seu tamanho.

EXERCÍCIO: Implemente o TAD Conjunto em C.

Aplicação de Conjuntos Dinâmicos

Suponha um conjunto composto de elementos que precisam ser interligados, isto é, *arranjados de modo linear*. Uma forma simples de alcançar o objetivo proposto é por meio de uma lista para, por exemplo, organizar os itens que devem ser comprados em um supermercado—uma **lista de compras**!
Na Seção anterior, foi discutido o conceito de conjuntos dinâmicos, ou seja, conjuntos de elementos que podem crescer ou diminuir com o passar do tempo e sobre os quais há algumas operações pré-definidas. Listas são estruturas que permitem organizar os elementos de um conjunto dinâmico, as quais podem crescer ou diminuir (pode-se inserir ou remover elementos dela) e que tem operações definidas para realizar tais modificações.

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*, 3^{rd} ed. 2009. MIT Press.

Portanto, uma lista é uma estrutura adequada para se aplicar em casos em que se precisa alocar dinamicamente a memória, ou seja, nos quais não se conhece a demanda final por objetos ou elementos e seus tipos. A lista de alunos em uma disciplina, quando esta se inicia, é um exemplo de conjuntos dinâmicos: é oferecido um número fixo de vagas (ex.: 30), mas permite-se abrir espaço para mais alunos se matricularem, o que faz com que a lista cresça; nas primeiras semanas, alguns alunos trancam ou desistem, uns mudam para a turma e pode haver matrículas atrasadas. Logo, a referida lista de alunos cresce (aloca-se memória para novos elementos) ou diminui (libera-se elementos) conforme as semanas passam.
Além disso, uma lista pode ser particionada em duas ou mais listas (ex.: produtos que necessitam ser refrigerados e produtos que não precisam; bebidas, enlatados, alimentos embalados e produtos perecíveis; produtos de limpeza/higiene, frutas/verduras, industrializados; entre outras), assim como listas distintas podem ser concatenadas em uma única lista (ex.: uma turma da disciplina foi cancelada e os alunos serão realocados em outra turma).
Uma lista cujos elementos estão organizados linearmente (linear) é um conjunto composto por uma sequência de "zero" ou mais elementos $a_1, a_2, ..., a_N$ de um tipo de dados específico, na qual a_1 é o primeiro elemento, a_N o último e N é o tamanho do conjunto. Se a lista contiver "zero" elementos, ela é dita **vazia**.

O TAD Lista e suas Operações

Por ora, os objetos do tipo Lista serão representados por três atributos: o último elemento (de acordo com a posição na lista), o número máximo de elementos que podem ser armazenados e uma estrutura para armazenar e manipular os elementos a serem guardados na lista. As operações básicas são definidas a seguir. No entanto, de acordo com a aplicação pretendida, pode-se incrementar o conjunto de operações disponíveis.

Operações básicas sobre listas:

1. Inicializar - cria uma lista linear vazia;
2. Inserir - adiciona um elemento à lista;
3. Remover - retira um dado elemento da lista;
4. Vazia - verifica se a lista está vazia;
5. Imprimir - mostra todos os elementos presentes na lista.

As operações básicas definidas acima são essenciais para se manipular listas lineares simples. O TAD Lista é ilustrado na Figura 1 e as funções que implementam as referidas operações são detalhadas na próxima seção.

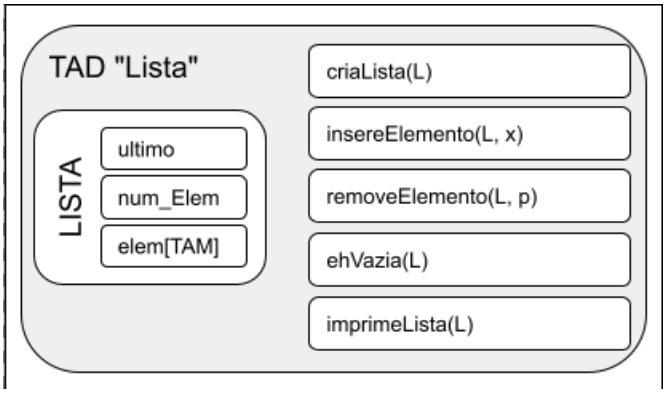

Figura 1: O TAD Lista, representado por sua estrutura, atributos e operações.

Listas Implementadas com Vetores

Como visto, uma lista é uma estrutura de dados cujos elementos são organizados de modo **linear**, isto é, sequencialmente. Ao se implementar uma lista com o auxílio de um vetor, a ordem linear é determinada pelos índices do vetor em questão. A Figura 2 mostra a implementação baseada no TAD Lista.

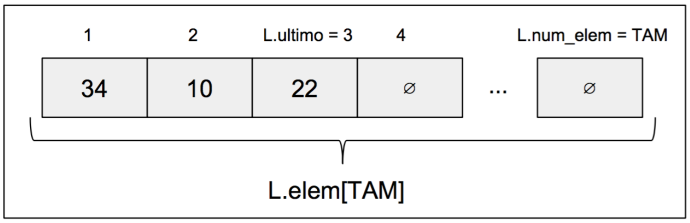

Figura 2: Exemplo de lista implementada com o auxílio de um vetor.

Funções de uma Lista Linear:

1. `criaLista(L)`. Cria uma lista vazia "L" e a retorna.
   - Dado um tamanho máximo (TAM), cria um objeto "L" do tipo Lista desse tamanho ou menor, de acordo com parâmetro passado pelo usuário:
     ```
     L.ultimo = 0;
     L.num_Elem = TAM;
     retorna L;
     ```
   - Pode-se "zerar" todas as posições da lista.
2. `insereElemento(L, x)`. Insere um elemento "x" após o último item da lista "L", caso ela não esteja cheia.
   - ```
     SE (L.ultimo >= 0 E L.ultimo < TAM) ENTÃO
     L.ultimo++;
     L[L.ultimo] = x;
     ```
   - ```
     SENÃO imprime "Lista cheia!";
     ```
3. `removeElemento(L, p)`. Retorna um elemento "x" na posição "p" da lista "L" e desloca todos os elementos a partir de "p+1" para as posições anteriores.
   - ```
     SE (NÃO ehVazia(L)) ENTÃO
     INTEIRO x = L.elem[p];
     PARA (INTEIRO i = p+1) ATÉ (i <= L.ultimo) FAÇA
     L.elem[i-1] = L.elem[i]; FIM_PARA
     L.ultimo-; FIM_SE
     RETORNA x;
     ```
   - Caso 2: `SE (p == L.ultimo)`?
   - Caso 3: tratar da posição inexistente.
4. `ehVazia(L)`. Verifica se a lista "L" está vazia e retorna *verdadeiro* se positivo, senão *falso*.
   - ```
     SE (L.ultimo == 0) ENTÃO
     retorna 'verdadeiro';
     SENÃO retorna 'falso';
     FIM_SE
     ```

- Como implementar a função `ehCheia(L)`?

5. `imprimeLista(L)`. Se a lista não for vazia, imprime todos os seus elementos na ordem em que aparecem pela posição.

   - ```
     SE (NÃO ehVazia(L)) ENTÃO
     PARA (i = 1) ATÉ (i <= L.ultimo) FAÇA
     imprime L[i];
     FIM_PARA
     FIM_SE
     ```

EXERCÍCIO: Implemente a Lista Linear usando vetor em C.
     ```