

Esudo de TAD

Samuel de Souza Gomes

¹Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
Rua 36, 115 - Loanda, João Monlevade - MG, 35931-008

samuel.gomes2@aluno.ufop.edu.br

Abstract. *This report aims to study Abstract Data Types (TADs) and basic programming concepts. With this, perform tests through sets obtained by vectors.*

Resumo. *Este relatório tem como objetivo o estudo de Tipos Abstratos de Dados (TADs) e conceitos básicos de programação. Com isso, realizar testes através de conjuntos obtidos por vetores.*

1. Introdução

O problema a ser resolvido gira em torno de manipulação de conjuntos. Para isso, foi criada uma estrutura de tipo de abstrato e funções que realizam tarefas específicas para cada um deles. No entanto, todas essas funções são declaradas num arquivo header e executadas num arquivo .c que, por fim, chamadas na função principal Main.

2. Implementação

O programa é composto por 14 funções que realizam tarefas como: inicializar um conjunto, gerá-lo aleatoriamente, imprimi-lo, comparar com outro, entre outros. As mesmas serão detalhas a seguir:

2.1. Arquivo Header

Como dito antes, foi criada um TAD para armazenar os elementos e tamanhos dos conjuntos. Tal estrutura foi gerada no arquivo header, onde também foram inicializadas as funções utilizadas pelo programa.

2.2. Função Inicializar

A função inicializar é do tipo void e tem como objetivo simplesmente "alocar" um tamanho para o conjunto e inicializá-lo, no qual recebe uma variável do tipo TConj por passagem de referência.

2.3. Função Inserir Elemento

Essa função vai inserir elementos no conjunto de tipo TConj e "realocar" seu tamanho caso o tamanho atual seja insuficiente. Seu novo tamanho é aumentado de 10 em 10. Por fim, retorna 1 caso consiga realizar essa operação e 0 caso não consiga.

2.4. Função Excluir Elementos

A função excluir recebe por parâmetro um conjunto e um elemento aleatório. Um loop verifica se esse elemento pertence ao conjunto, caso encontre-o, o elemento então é excluído e retorna 1. Caso não encontre, a função simplesmente retorna 0.

2.5. Função "Setar" Elemento

Essa função vai receber como parâmetro um conjunto, um elemento e uma posição. Com isso, a função irá verificar se a posição passada existe no conjunto, caso exista essa posição vai receber o elemento passado e retornará 1. Caso não exista, a função simplesmente retorna 0;

2.6. Função Recuperar Elemento

A função recuperar elemento irá receber um conjunto, uma posição e um elemento por referência. Primeiramente, ela irá verificar se a posição passada existe no conjunto. Caso exista, a variável passada por referência recebe o valor que está na posição passada por parâmetro e retorna 1. Caso não exista, apenas retorna 0.

2.7. Função Testar Elemento

Essa função irá receber como parâmetro um conjunto e um elemento. Em seguida irá percorrer todo o vetor com objetivo de encontrar o elemento passado. Caso encontre-o a função retorna sua posição, senão retorna -1;

2.8. Função Gerar Conjunto

A função gerar conjunto irá receber por parâmetro um número inteiro, no qual será o tamanho do conjunto gerado. Por conseguinte, a função irá criar uma variável do tipo TConj e "alocar" o tamanho passado à ela. Após declarar, a função inicializa o conjunto criado com números aleatórios. Por fim, retorna o conjunto criado.

2.9. Gerar Conjunto à Partir de um Número

Essa função vai receber como parâmetro um número inteiro, que, por meio de cálculos de divisão e resto da divisão vai gerar um conjunto. Os cálculos são feitos em cima de cada algarismo do número, o que torna possível adicionar cada um separadamente no vetor de tipo TConj. Por fim, a função retorna o conjunto.

2.10. Gerar Número à Partir de um Conjunto

Essa função é exatamente o inverso da anterior. A mesma irá receber como parâmetro um conjunto do tipo TConj. Em seguida, irá fazer com que uma variável inteira receba o somatório da expressão programada. Finalmente, o número será gerado e retornado.

2.11. Comparar Conjuntos

A Função Comparar irá basicamente receber dois conjuntos por parâmetro e compará-los. Primeiramente, é feita uma verificação se os tamanhos dos vetores são iguais. Se forem diferentes já retorna 0, caso contrário, é feita uma comparações com cada posição dos conjuntos para realmente saber se todos os elementos são iguais.

2.12. União, Intersecção e Subtração

Tais funções vão de fato adotarem os conceitos matemáticos de conjuntos. Em todas são feitas comparações e a geração de um novo conjunto que será retornado. Portanto, na união o novo conjunto recebe todos os elementos que são distintos entre os vetores passados por parâmetro. Já na intersecção, o novo conjunto recebe todos os elementos que são iguais. Já na subtração, o vetor gerado recebe todos os elementos que pertencem ao primeiro conjunto, mas não pertencem ao segundo.

3. Estudo de Complexidade

Como não se trata de listas simples ou duplamente encadeadas, árvores e/ou busca binária, quase todas as funções vão ter tempo de complexidade $O(n)$ ou $O(n^2)$. Isso devido ao fato de que para a busca, inserção e deleção é necessário percorrer todo o conjunto para realizar as devidas comparações, com exceção das funções de inserir e recuperar elemento.

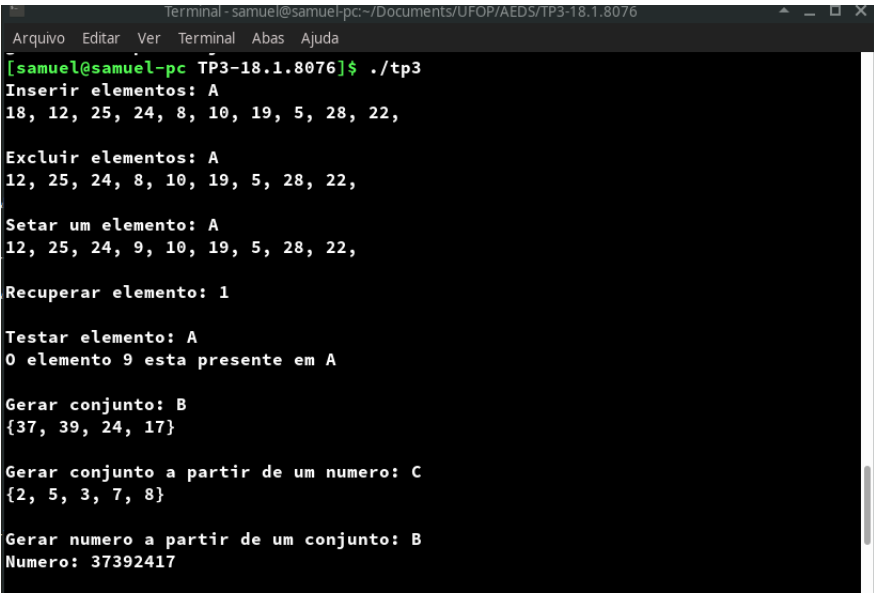
Estas funções tem complexidades diferentes, pois na inserção a função já insere o elemento na última posição. Já na função de recuperar elemento, a mesma recebe a posição do elemento a ser buscado por parâmetro. O que facilita e diminui bastante o tempo de execução, pois se precisasse procurar o elemento no vetor, a complexidade seria maior.

4. Teste Realizado

Após a realização dos testes podemos acompanhar os seguintes resultados.

Na primeira imagem é demonstrado que o conjunto A foi inicializado e a função de inserir elementos armazena valores aleatórios no vetor. Em seguida, o mesmo conjunto é passado para a função de excluir elementos também aleatórios. Esta função verifica se o elemento existe no conjunto e o exclui. Logo após, é inserido um elemento qualquer em determinada posição do conjunto A. Na função seguinte, é passado uma posição e uma variável por referência, a função irá verificar se a posição passada existe no vetor, caso exista, a função faz com que a variável receba o valor nessa posição do conjunto A. Já a função "Testar Elemento" realiza a tarefa de verificar se um determinado elemento existe no conjunto A

Dando sequência aos testes, a função gerar conjuntos cria um vetor B, que é retornado e impresso na tela. Um outro conjunto, C, é gerado à partir de um número passado para a função. A função seguinte é exatamente o inverso, ela gera um número à partir de um conjunto passado por parâmetro, que nesse caso foi o conjunto B.



```
Terminal - samuel@samuel-pc: ~/Documents/UFOP/AEDS/TP3-18.1.8076
Arquivo Editar Ver Terminal Abas Ajuda
[samuel@samuel-pc TP3-18.1.8076]$ ./tp3
Inserir elementos: A
18, 12, 25, 24, 8, 10, 19, 5, 28, 22,

Excluir elementos: A
12, 25, 24, 8, 10, 19, 5, 28, 22,

Setar um elemento: A
12, 25, 24, 9, 10, 19, 5, 28, 22,

Recuperar elemento: 1

Testar elemento: A
0 elemento 9 esta presente em A

Gerar conjunto: B
{37, 39, 24, 17}

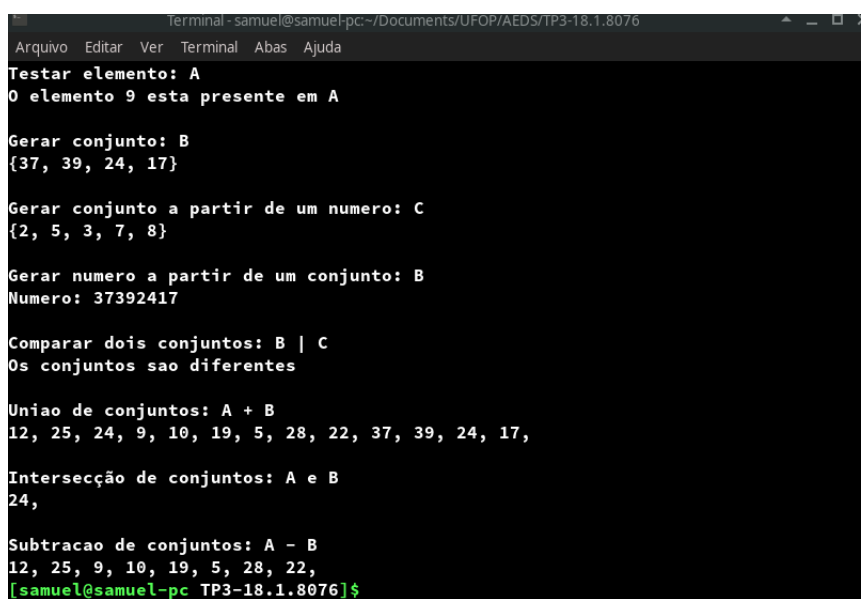
Gerar conjunto a partir de um numero: C
{2, 5, 3, 7, 8}

Gerar numero a partir de um conjunto: B
Numero: 37392417
```

Figure 1. Execução

Já na segunda imagem e segunda parte dos testes, são exibidos resultados das 4 últimas funções que vão envolver comparações entre os conjuntos gerados.

Destas, a primeira função (Comparar) vai simplesmente verificar se os conjuntos são exatamente iguais, ou seja, analisar se os elementos são os mesmos. A próxima (União) irá unir os dois conjuntos gerando um novo. A função "Intersecção" irá retornar um novo conjunto com os elementos iguais entre os vetores comprados. Por fim, a última função irá subtrair o segundo conjunto do primeiro, ou seja, retornará um novo conjunto com todos os elementos que pertencem ao primeiro, mas não o segundo.

A terminal window titled "Terminal - samuel@samuel-pc:~/Documents/UFOP/AEDS/TP3-18.1.8076" with a menu bar (Arquivo, Editar, Ver, Terminal, Abas, Ajuda). The terminal displays the following commands and outputs:

```
Testar elemento: A
0 elemento 9 esta presente em A

Gerar conjunto: B
{37, 39, 24, 17}

Gerar conjunto a partir de um numero: C
{2, 5, 3, 7, 8}

Gerar numero a partir de um conjunto: B
Numero: 37392417

Comparar dois conjuntos: B | C
Os conjuntos sao diferentes

Uniao de conjuntos: A + B
12, 25, 24, 9, 10, 19, 5, 28, 22, 37, 39, 24, 17,

Intersecção de conjuntos: A e B
24,

Subtracao de conjuntos: A - B
12, 25, 9, 10, 19, 5, 28, 22,
[samuel@samuel-pc TP3-18.1.8076]$
```

Figure 2. Execução

5. Conclusão

Com a realização do trabalho fica ainda mais evidente a importância, praticidade e eficiência dos TADs. Isso é provado à partir do momento em que se pode utilizar tal estrutura em todo o programa.

O que de fato é importante, pois ajuda na manutenção do código ao, principalmente ter a possibilidade de separá-lo em arquivos diferentes. É pratico porque permite a reutilização de código. E portanto, é eficiente pois como dito, pode-se utilizar em qualquer parte do código.

Além disso, à partir de uma única estrutura podemos criar várias instâncias distintas para representar algo do mesmo tipo de formas diferentes. Como por exemplo, o trabalho em questão, no qual foram desenvolvidos alguns conjuntos para testar o TAD através de funções que manipularam os elementos.

6. Referências

GEEKS FOR GEEKS. **Abstract Data Types**. Disponível em: <https://www.geeksforgeeks.org/abstract-data-types/>. Acesso em 21 jun. 2019