TAD

Daniel de Souza Rodrigues - 18.2.8112 - Sistemas de Informação

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
Rua 36, Número 115 - Bairro Loanda, João Monlevade - CEP: 35931-008

daniel.sr@aluno.ufop.edu.br

Abstract. *In this paper we will present the implementation of an abstract data type (TAD) in C language. It is (TAD) presented a set of integers and will be made a brief analysis of functions implemented: Initialize; Insert; Delete; Assign (SET); Recover (GET); Test presence of element in (TAD); Generate a random (TAD); Create (TAD) from a number; Generate a number through a (TAD); Compare two (TAD); Print (TAD); Join two (TAD); Merge two (TAD); Subtract (TAD).*

Resumo. *Neste artigo será apresentado a implementação de um tipo abstrato de dado (TAD) em linguagem C, Está (TAD) é um conjunto de elementos inteiros e será feita uma breve analise sobre funções como: Inicializar; Inserir; Excluir; Atribuir (SET); Recuperar (GET); Testar presença de elemento no (TAD); Gerar uma (TAD) aleatória; Criar (TAD) a partir de um numero; Gerar um número através de um (TAD); Comparar dois (TAD); Imprimir (TAD); Unir dois (TAD); Fusionar dois (TAD); Subtrair (TAD);*

1. Introdução

Em computação, o tipo abstrato de dado (TAD) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados. Além disso, é uma metodologia de programação que tem como proposta reduzir a informação necessária para a criação/programação de um algoritmo através da abstração das variáveis envolvidas em uma única entidade fechada, com operações próprias à sua natureza.[Wikipédia 2018]

A (TAD) implementada ”TConj” é composta por um ponteiro ”elemento” do tipo “TElem”, ou seja ”TElem” também é um (TAD) onde cada elemento presente no conjunto é armazenado em sua variavel de inteiros ”item”, ”TConj” também possui uma variável “n” que armazena a quantidade de elementos contidos na “TElem” e uma variável auxiliar “last” que armazena o valor do elemento presente na última posição $(n - 1)$ de “TElem”.

```
typedef struct{
    int item;
}TElem;

typedef struct{
    TElem* elemento;
    int n;
    int last;
}TConj;
```

Figure 1. TAD implementado em C

2. Implementação

2.1. TConj

A implementação do (TAD) “TConj” foi baseado na ideia de estabelecer um conjunto de números inteiros utilizando um vetor para armazenar os elementos do conjunto.[Conjuntos 2018]

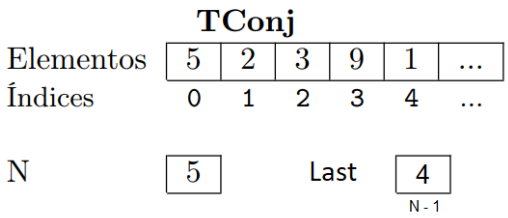

Figure 2. TConj

Para estabelecer o conjunto dos elementos foi utilizado um ponteiro, tornando assim a alocação de memória dinâmica, onde é possível criar funções que manipulam os elementos do conjunto de forma dinâmica, através da passagem por referência estabelecidas nos parâmetros de cada função.

2.2. Inicializar - $inicializar$(TConj*)

A função recebe com parâmetro um ponteiro para o “TConj”, ela parametriza o tamanho do “TConj” (n) em 10. Utiliza a função ($calloc$) para alocar dinamicamente 10 espaços de memoria e atribuiu a variável ($last$) o conteúdo presente no ultimo espaço de elemento (n-1).

2.3. Inserir elemento - $inserirElemento$(TConj*, int elemento)

A função recebe como parâmetro um ponteiro para o “TConj” e uma variável do tipo “int” com o valor a ser inserido, primeiramente a função utiliza um if para comparar todos os elementos do conjunto com o valor do elemento presente na variável ($elemento$) para saber se o valor já existe no conjunto, caso exista a função quebra o laço de repetição e retorna 0, caso o valor não esteja presente a função verifica se a um espaço vazio no conjunto, caso exista o valor contido na variável ($elemento$) é atribuído a posição vazia do conjunto, caso não exista posição vazia o espaço de memória do conjunto é realocado em seu tamanho atual (n + 1) utilizando “$realloc$” é o valor de retorno é 1;

2.4. Excluir elemento - $excluirElemento$(TConj*, int elemento)

A função recebe como parâmetro um ponteiro para o “TConj” e uma variável do tipo “int” com o valor a ser deletado, a função utiliza um laço com a comparação do valor recebido como parâmetro ($elemento$) e compara todos os elementos presentes no conjunto, caso o elemento não esteja presente no conjunto a função retorna 0, caso o elemento esteja um logo laço é executado para mover o elemento presente em sua próxima posição ($i + 1$) o laço é executado ate todos os elementos do conjunto serem reajustados, logo depois o conjunto é realocado dinamicamente com a função ”$realloc$” em (N – 1), sendo ”n” o tamanho do conjunto.

2.5. Fixar elemento (SET) - $setElemento$(TConj*, int pos, int elemento)

A função recebe como parâmetro um ponteiro para o "TConj" e duas variáveis do tipo "int" uma contendo o valor do elemento para o (SET) e outra com seu índice (posição), primeiramente a função compara o valor de índice passado como parâmetro, caso o tamanho conjunto seja menor que o valor passado como índice a função retorna 0, caso o valor seja (índice $>= 0$ & indice $<$ n) um laço é executado para verificar se o elemento já existe no conjunto, caso exista o retorno é 0, caso não exista a área apontada pelo conjunto na posição do indicie passado como parâmetro recebe o valor passado como parâmetro em ($elemento$).

2.6. Recuperar elemento (GET) - $getElemento$(TConj, int pos, int* pelem)

A função recebe como parâmetro um "TConj", uma variável "int" com o valor do indice a ser acessado e um ponteiro "int" que aponta um espaço de memória "pelem" que recebera o valor presente no conjunto na posição do índice passado como parâmetro, primeiramente o valor do índice passado como parâmetro é verificado caso seja (índice $>= 0$ & índice $<$ n) a área apontada por "pelem" recebe o valor presente no conjunto na posição do índice passado como parâmetro e retorna 1, caso não seja (índice $>= 0$ & índice $<$ n) a função retorna 0 e nenhum valor é atribuído a área apontada por "pelem".

2.7. Testar elemento - $testarElemento$(TConj, int elemento)

A função recebe como paramento um "TConj" e uma variável elemento do tipo "int" com o valor do elemento a ser testado, logo a função executa um laço para varrer o conjunto é comparar se o elemento está presente em uma das posições do conjunto, caso o elemento esteja presente a função retorna 1, caso não esteja presente retorna 0.

2.8. Gerar conjunto aleatório - $gerarConjunto$(int num)

A função recebe como parâmetro uma variável "num" do tipo "int", primeiramente verifica de o valor de "num" não é ($<= 0$) logo a função cria um "TConj" de nome 'aleatório' e atribui a seu tamanho o valor de "num", o "TElem" presente em 'aleatório' aloca dinamicamente um espaço de memorai correspondente ao valor de "num", logo depois um laço é executado para varrer o conjunto de (0 até num-1), para atribuir valores aleatórios a cada posição apontada por "elemento" utilizando a função rand() presente na biblioteca $< math.h >$, a variável "last" recebe o último elemento (num-1) presente no conjunto 'aleatório', depois de realizado todo o procedimento a função retorna o (TAD) 'aleatório'.

2.9. Gerar um conjunto a partir do número - $num2Conj$(int num)

A função recebe como parâmetro uma variável "num" do tipo "int", primeiramente "num" é comparado, caso (num $<= 0$) o programa executa "exit(1)", o algoritmo utiliza uma variável do tipo "'int" denominada 'i' a qual são atribuídos valores baseados em cada operação feita em seus loops internos, caso (num > 0) um ponteiro auxiliar é alocado dinamicamente com tamanho 2, logo um laço while é executado obedecendo a imposição (num < 0), a cada loop uma variável "int" auxiliar recebe o mod de num por 10 (num%10), logo num é dividido por 10 e o ponteiro auxiliar na posição do índice [i] recebe o valor da variável "auxiliar", logo uma nova comparação é feita, caso (num ¿ 0) o ponteiro auxiliar realoca dinamicamente memória baseado no seu (tamanho atual + 1), após este processo o ponteiro auxiliar é varrido para procurar elementos iguais, caso

exista elementos iguais a função executa exit(1), pois quebrara a regra de conjunto, onde não se pode ter elementos iguais, após todas as verificações um (TAD) 'num2' é criado e seu "TElem" é alocado com o valor do indice [I], logo outro loop é executado e todos os elementos do ponteiro auxiliar são transpostos para o conjunto de 'num2', a memória do ponteiro auxiliar é liberada e "last" de 'num2' recebe o elemento de [i-1], o algoritmo retorna o (TAD) 'num2'.

2.10. Gerar um número a partir do conjunto - $conj2Num$(TConj)

A função recebe como parâmetro um "TConj", primeiramente a função verifica se o tamanho do "TConj" é superior a 0, caso seja a função aloca um ponteiro auxiliar do tipo ''int" com o tamanho do índice de "TConj", um laço é executado para atribuir a cada posição [i] do ponteiro auxiliar o valor gerado por (elemento do conjunto[i] * potencia (10 sobre índice), o índice é decrementado em (-1) a cada interação do laço, logo uma variável 'soma' do tipo "int" é gerada, um novo laço é chamado para executar as atribuições dos valores presentes no ponteiro auxiliar (soma = soma + ponteiro auxiliar[0 até índice -1]), a memória alocada para o ponteiro auxiliar é liberada é a função retorna soma.

2.11. Comparar 2 conjuntos - $comparar Conj$(TConj A,TConj B)

A função recebe como parâmetro dois (TAD) "TConj", primeiramente a função compara o tamanho de cada "TConj" caso sejam diferentes a função retorna 0, logo é chamado o procedimento Insertion_Sort que recebe como parâmetro o endereço de memória de cada "TConj", o Insertion_Sort ordena os seus conjuntos, logo um laço é chamado para verificar se existe elementos em A diferentes de B, caso exista a função retorna 0, caso não existam a função retorna 1, e imprime que os conjuntos são iguais.

2.12. Imprimir os elementos do conjunto - $imprimir$(TConj)

O procedimento recebe um (TAD) "TConj" como parâmetro,o procedimento utiliza uma variável 'i' do tipo "int" a qual é atribuído (+1) a cada interação do laço, logo o procedimento verifica se (índice "TConj" ¡= 0), caso seja o procedimento executa exit(1), caso não seja o procedimento utiliza um printf("ARRAY:") e executa um laço, onde existe um printf("%d",elemento[i]) a cada loop é feita uma comparação , caso elemento[i] seja igual ao "last", o procedimento executa break.

2.13. Unir os conjuntos A e B, gerando um conjunto C - $uniao$(TConj A,TConj B)

A função recebe como parâmetro dois (TAD) A e B "TConj", logo ela aloca um ponteiro auxiliar com o tamanho (A+B), um laço é chamado para atribuir a cada posição do ponteiro auxiliar os valores do conjunto A, uma variável 'retorno' e utilizado para manter a posição ao ser atribuído os valores de B ao ponteiro auxiliar, o procedimento Insertion_SortP é executado recebendo como parâmetro o endereço de memória do ponteiro auxiliar, uma variavel 'tamanho_novo' é posicionada e a função rearranja recebe como parâmetro o (ponteiro auxiliar,tamanho, endereço de tamanho_novo) , logo ponteiro auxiliar é realocado com base no (tamanho - tamanho_novo), um novo (TAD) 'C' é criado e é alocado dinamicamente memória para seu "TElem", um loop é executado para atribuir os valores armazenados no ponteiro auxiliar ao "TElem" 'elemento' de C, gerando assim a união do conjunto, o espaço de memória alocado para ponteiro auxiliar é liberado e C é retornado;

2.14. Fusionar gerando C - $fusionar$(TConj A,TConj B)

A função recebe como parametro dois (TAD) A e B "TConj",ela utiliza uma variavel 'aux' do tipo "int" para comparações feitas no laço, logo ela aloca um ponteiro auxiliar com o tamanho de (A+B), a função Insertion_Sort é chamada recebendo como parametro o endereço de memória de cada "TConj", logo um laço encaixado é executado para comparar os elementos iguais presentes nos conjuntos A e B, a cada igualdade é atribuído (+1) a 'aux', o ponteiro auxiliar é realocado com o tamanho de 'aux', a função "rearranja" é chamada para remover os elementos iguais do ponteiro auxiliar, um novo (TAD) 'C' é criado e seu "TElem" é alocado com o (tamanho - tamanho_novo), um novo laço é executado para atribuir os elementos do ponteiro auxiliar ao conjunto "elementos" 'C', a memória alocada para o ponteiro auxiliar é liberada, a função retorna C.

2.15. Subtrair (A - B) gerando um conjunto C - $subtrair$(TConj A,TConj B)

A função recebe como parâmetro dois (TAD) A e B "TConj", ela utiliza duas variáveis 'aux' do tipo "int", e aloca 2 ponteiros auxiliares, cada ponteiro recebe o tamanho correspondente a A ou B, logo um laço encaixado é executado com a comparação (A[i]=B[j]), uma variável 'igual' é utilizada, caso permaneça ('igual' = 0) durante toda a verificação nenhuma outra interação é feita, caso igual seja atribuída o ponteiro auxiliar de A ou B recebe na posição[x] o valor do elemento A ou B [i], 'x' é inteirado em (+1) a cada atribuição, o mesmo processo é realizado para (B[I]=A[i]), logo os ponteiros são realocados com seu novo tamanho baseado no valor de 'aux', um novo ponteiro 'conjunt' é alocado e recebe os valores armazenados em ponteiro auxiliar A e B através de um laço, logo um novo (TAD) "C" é criado, o "TElem" de C é alocado com o tamanho de (aux+aux2), um novo laço é executado, os valores presentes em 'conjunt' são passados para o conjunto 'C', após toda a memória alocada nos ponteiros auxiliares e 'conjunt' é liberada, C é retornado.

2.16. IMPLEMENTAÇÕES EXTRAS

Foi implementando no arquivo "sort.c", algoritmos para auxiliar na ordenação e no rearranjo dos conjuntos e ponteiros auxiliares, todos os protótipos dos algoritmos estão em "TConj.h".[Wikipédia 2019b]

- $insertionSort$ procedimento no qual recebe um (TAD) "TConj" é ordena todos os elementos do (TAD) "TConj".
- $insertionSortP$ procedimento no qual recebe um "int" ponteiro 'pA' e seu tamanho, ordena todos os elementos presentes na area apontada por 'pA'.
- $rearanja$ procedimento no qual recebe como parâmetro uma área apontada por 'A', uma variável "int" 'tamanho' e uma área apontada 'tamanho_novo', o ($rearanja$) remove todos os elementos iguais presentes em 'A' e atribui a 'tamanho_novo' o número de rearranjos ocorridos em 'A'.

ALGORITMO	MELHOR CASO	PIOR CASO
insertionSort	O(n)	O(n^2)
insertionSortP	O(n)	O(n^2)
rearanja	O(n^2)	O(n^2)

3. Estudo de complexidade

As funções: Unir, Fusionar e Comparar utilizam o algoritmo de ordenação Insertion Sort que possui sua complexiade de tempo em Melhor Caso O(n), mas é possivel implementar todas essas funções utilizando algoritmos de melhor complexidade como o Heap Sort O($nlongn$)[Wikipédia 2019a], como todos os testes foram realizados em Conjuntos(TAD) com poucos elementos foi optado a utilização do Insertion Sort, por sua facil implementação é agilidade com ordenação de elementos em pequena escala.

Função / Procedimento	Melhor Caso	Pior Caso
Inicializar	O(1)	O(1)
Inserir	O(n)	O(n)
Excluir	O(n)	O(n^2)
Fixar(SET)	O(n)	O(n)
Recuperar(GET)	O(1)	O(1)
Testar Elemento	O(n)	O(n)
Gerar (TAD) Aleatorio	O(n)	O(n)
Gerar (TAD) com Número	O(n)	O(n^2)
Gerar Número com (TAD)	O(n)	O(n)
Comparar	O(n)	O(n^2)
Imprimir	O(n)	O(n)
Unir	O(n)	O(n^2)
Fusionar	O(n^2)	O(n^2)
Subtrair	O(n^2)	O(n^2)

Table 1. Tabela de complexidade das funções implementadas

4. Testes Realizados

- INICIALIZAR - OK - ESPAÇO DE MEMORIA ALOCADO
- INSERIR - Array: 37 9 27 23 15 28 26 1 4 24 10 29 14 21 30 18
- EXCLUIR - **Um fato curioso foi observado:**
  Definindo srand(171), nenhum elemento foi removido deletado teste realizado 5 vezes.
  Array: 37 9 27 23 15 28 26 1 4 24 10 29 14 21 30 18
  Definindo srand(time(NULL), foram removidos 2 elementos.
  Array: 27 23 15 28 26 1 4 24 10 29 14 21 30 18
- SET - Posição 2 Array: 37 9 999 23 15 28 26 1 4 24 10 29 14 21 30 18
- GET - Posição 3 – ELEMENTO 23
- TESTE DO ELEMENTO NO CONJUNTO (9) - Esta presente no conjunto! Posicao: 1
- CONJUNTO ALEATORIO: Array: 29 64 70 55 9 14 1 70 38 34
- TRANSFORMAR CONJ EM NUM INT: NUM: 756048413
- TRANSFORMAR NUM PARA CONJUNTO -
  Array: 7 5 6 0 4 8 4 1 3
- COMPARAR Os conjuntos sao diferentes
- UNIAO: Array: 1 4 9 10 14 15 18 21 23 24 26 28 29 30 34 37
- INTERSECAO - Array: 0 1
- SUBTRAIR - Array: 4 10 15 18 21 23 24 26 28 30 37 999 34 38 55 64 70

5. Conclusão

A implementação de (TAD) é uma boa pratica para programação, pelo fato de a alocação de memória ser pertinente, a possibilidade de abstração de dados é muito útil e eficaz quando precisamos criar um novo procedimento ou uma nova função na qual podemos ter um (TAD) que contém diversos dados distintos, um dos problemas que são enfrentados no TAD é que ele é uma estrutura metafórica gerada e não há nenhuma segurança que as operações e regras de operação desejadas para este tipo serão respeitadas.

”A abstração de informações através do TAD permitiu a melhor compreensão dos algoritmos e maior facilidade de programação, e por consequência aumentou a complexidade dos programas, tornando-se fundamental em qualquer projeto de software a modelagem prévia de seus dados.”[Wikipédia 2018]

References

Conjuntos (2018). Conjuntos definição e estudo.

Wikipédia (2018). Tipo abstrato de dado — wikipédia, a enciclopédia livre. [Online; accessed 19-outubro-2018].

Wikipédia (2019a). Heapsort — wikipédia, a enciclopédia livre. [Online; accessed 20-abril-2019].

Wikipédia (2019b). Insertion sort — wikipédia, a enciclopédia livre. [Online; accessed 17-junho-2019].