_____
ICMC - Instituto de Ciências Matemáticas e da Computação
Disciplina: SCC0202 - Algoritmo e Estrutura de Dados I
Projeto 2

Aluno: Alec Campos Aoki (15436800)
Aluno: Jhonatan Barboza da Silva (15645049)

Descrição do projeto:
Escolhemos implementar os TADs utilizando uma lista sequencial ordenada, uma árvore binária de busca AVL e um TAD Conjunto, responsável por gerenciar as operações durante a execução.

Justificativa:
A escolha de implementar o TAD lista sequencial ordenada se deve à utilização da busca binária, que pode ser realizada em O(log(n)). Apesar de as operações de inserção e remoção apresentarem complexidade O(n), no geral, o desempenho é satisfatório em cenários onde a busca é predominante. Já a árvore binária de busca AVL foi escolhida por sua eficiência consistente, com complexidade O(log(n)) para as operações de busca, inserção e remoção, garantindo bom desempenho mesmo no pior caso, além do fato de ser uma estrutura encadeada (menor consumo de espaço).

Main:
A função main gerencia as operações e o fluxo do programa. Inicialmente, ela solicita ao usuário que escolha a estrutura de dados a ser utilizada (0 para lista e 1 para ABB AVL). Com base nessa escolha, dois conjuntos (conjA e conjB) são criados utilizando a estrutura selecionada.

Em seguida, o programa lê o tamanho dos conjuntos A e B. Caso ambos os tamanhos sejam zero, uma mensagem informando que os conjuntos estão vazios é exibida, e o programa é encerrado. Caso contrário, os elementos de cada conjunto são lidos e inseridos nas respectivas estruturas de dados por meio da função conjunto_inserir.

Posteriormente, o programa solicita ao usuário que selecione a operação a ser realizada entre os conjuntos. As operações disponíveis são: verificar a pertinência de um elemento a um conjunto (PERTENCE), realizar a união dos conjuntos (UNIAO) ou calcular a interseção dos conjuntos (INTERSEC). Com base na operação escolhida, a função correspondente é executada para obter e exibir o resultado.

Ao final do programa, os conjuntos criados são apagados da memória utilizando a função conjunto_apagar, e o programa é encerrado com sucesso.

TAD Lista Sequencial Ordenada
O objetivo do arquivo lista.c é implementar uma lista sequencial ordenada com suporte a busca binária. Ele contém as principais funções relacionadas à manipulação da lista, como inserção, remoção, busca binária e outras operações úteis. Abaixo segue uma breve descrição de cada função:

Funções:

1. lista_criar

* Descrição: Aloca memória para uma estrutura de lista e inicializa o tamanho como 0.
* Operações principais:
* Alocação de memória: O(1).
* Inicialização do tamanho: O(1).
* Complexidade Total: O(2).

2. lista_apagar

* Descrição: Libera a memória da lista e define o ponteiro como NULL.
* Operações principais:
* Liberação de memória: O(1).
* Atribuição de NULL: O(1).
* Complexidade Total: O(2).

3. lista_inserir

* Descrição: Insere um elemento na lista em ordem crescente, deslocando elementos para abrir espaço.
* Operações principais:
* Busca pela posição correta: O(n) no pior caso (percorre toda a lista).
* Verificação de duplicatas (chamada a lista_busca): O(1).
* Deslocamento de elementos para abrir espaço: O(n) no pior caso (inserção no início).
* Complexidade Total: O(2n + 1).

4. lista_remover

* Descrição: Remove um elemento da lista, deslocando elementos subsequentes para preencher o espaço.
* Operações principais:
* Busca binária para encontrar a posição do elemento: O(log n).
* Deslocamento de elementos para preencher o espaço: O(n) no pior caso (remoção do primeiro elemento).
* Complexidade Total: O(log(n) + n).

5. lista_imprimir

* Descrição: Imprime todos os elementos válidos da lista em ordem.
* Operações principais:
* Iteração pelos elementos da lista: O(n).
* Complexidade Total: O(n).

6. lista_busca

* Descrição: Busca um elemento na lista utilizando a função buscaBinariaLista.
* Operações principais:
* Busca binária: O(log(n)).
* Complexidade Total: O(log(n)).

6.1. buscaBinariaLista

* Descrição: Realiza busca binária recursiva para encontrar um elemento no vetor.
* Operações principais:
* Divisão recursiva do vetor em cada chamada: O(log(n)) no pior caso.
* Complexidade Total: O(log(n)).

7. lista_copiar
* Descrição: Cria uma cópia da lista original, inserindo cada elemento na nova lista.
* Operações principais:
* Criação de uma nova lista: O(1).
* Iteração para copiar elementos: O(n).
* Inserção de cada elemento (considerada O(1) neste contexto): O(n).
* Complexidade Total: O(2n + 1).

8. lista_consulta
* Descrição: Retorna o elemento armazenado em um índice específico da lista.
* Operações principais:
* Verificação de limites: O(1).
* Acesso ao elemento pelo índice: O(1).
* Complexidade Total: O(2).

TAD Árvore Binária de Busca AVL

O objetivo do arquivo AVL.c é implementar uma Árvore Binária de Busca AVL (uma árvore de busca balanceada). A característica principal dessa estrutura é o balanceamento, que garante que a diferença entre as alturas das subárvores esquerda e direita de qualquer nó nunca seja maior que 1. Caso essa condição seja violada após uma inserção ou remoção, a árvore realiza rotações para corrigir o balanceamento.

A implementação inclui definições auxiliares para criar, remover, balancear e acessar os nós, além das funções principais do TAD AVL.

Funções:

1. avl_criar

* Descrição: Aloca memória para uma estrutura AVL, inicializa a raiz como NULL e o tamanho como 0.
* Operações principais:
* Alocação de memória: O(1).
* Inicialização de ponteiros e variáveis: O(1).
* Complexidade Total: O(2).

2. avl_apagar

* Descrição: Apaga todos os nós da árvore recursivamente utilizando a função auxiliar no_apagar_recursivo.
* Operações principais:
* Chamada recursiva para percorrer todos os nós em pós-ordem: O(n), onde n é o número de nós.
* Liberação de memória de cada nó: O(1) por nó.
* Complexidade Total: O(2n).

3. avl_inserir

* Descrição: Insere um novo nó na árvore AVL e mantém o balanceamento.

* Operações principais:

* Criação de um novo nó: O(1).

* Chamada à função auxiliar avl_inserir_no para percorrer a árvore e realizar a inserção e rotações: O(h), onde h é a altura da árvore.

* Complexidade Total: O(h) como se trata de uma árvore balanceada: O(log(n) + 1).

4. avl_remover

* Descrição: Remove um nó da árvore AVL e mantém o balanceamento.

* Operações principais:

* Busca do nó a ser removido: O(h), onde h é a altura da árvore.

* Chamada à função auxiliar avl_remover_no para tratar os casos de remoção e realizar rotações: O(h).

* Complexidade Total: O(2h) como se trata de uma árvore balanceada: O(2 log(n)).

5. avl_get_altura

* Descrição: Calcula a altura da árvore utilizando a função auxiliar no_get_altura.

* Operações principais:

* Cálculo recursivo da altura para cada nó: O(h).

* Complexidade Total: O(h) como se trata de uma árvore balanceada: O(log(n)).

6. avl_get_tamanho

* Descrição: Retorna o tamanho da árvore armazenado na estrutura AVL.

* Operações principais:

* Acesso direto a uma variável: O(1).

* Complexidade Total: O(1).

7. avl_copiar

* Descrição: Cria uma cópia da árvore, duplicando todos os nós com a função auxiliar no_copiar_recursivo.

* Operações principais:

* Chamada recursiva para percorrer e copiar cada nó: O(n), onde n é o número de nós.

* Complexidade Total: O(n).

8. avl_imprimir

* Descrição: Imprime os elementos da árvore em ordem crescente (in-ordem) utilizando a função auxiliar avl_imprimir_arv.

* Operações principais:

* Percurso in-ordem para visitar todos os nós: O(n), onde n é o número de nós.

* Complexidade Total: O(n).

9. avl_busca

* Descrição: Busca uma chave na árvore AVL usando a função auxiliar busca_binaria_avl.

* Operações principais:

* Busca binária recursiva até encontrar o nó ou determinar sua ausência: O(h), onde h é a altura da árvore.
* Complexidade Total: O(h) como se trata de uma árvore balanceada: O(log(n)).

10. avl_get_chave_raiz
* Descrição: Retorna a chave armazenada na raiz da árvore ou erro se estiver vazia.
* Operações principais:
* Acesso direto ao ponteiro da raiz: O(1).
* Complexidade Total: O(1).

TAD Conjunto:

O código importa os arquivos lista.h, avl.h e conjunto.h e define a estrutura conjunto. Essa estrutura contém o campo TAD, que indica o tipo de estrutura de dados utilizada (lista ou árvore binária), o tamanho do conjunto e dois ponteiros: um para a lista (conjuntoLista) e outro para a árvore binária (conjuntoAVL). Apenas um desses ponteiros será utilizado, dependendo da estrutura escolhida.

A ideia principal das funções é gerenciar as chamadas para as funções específicas da lista ou da árvore binária, garantindo que as operações sejam executadas de acordo com a estrutura de dados escolhida.

Funções:

1. conjunto_criar
* Descrição: Aloca memória para o conjunto e inicializa o ponteiro correspondente à estrutura escolhida (lista ou árvore).
* Operações principais:
* Alocação de memória: O(1).
* Inicialização de ponteiros: O(1).
* Complexidade Total: O(1).

2. conjunto_apagar
* Descrição: Libera a memória ocupada pelo conjunto.
* Operações principais:
* Liberação do conjunto com chamada à função específica (detalhada posteriormente).
* Liberação da memória do próprio conjunto.
* Complexidade Total: O(2n), pior caso é o TAD AVL.

3. conjunto_inserir
* Descrição: Insere um elemento no conjunto e incrementa o tamanho.
* Operações principais:
* Verifica o tipo de estrutura e chama a função correspondente para inserção.
* Incrementa o tamanho do conjunto.
* Complexidade Total: O(2n + 1), pior caso é o TAD Lista.

4. conjunto_remover
* Descrição: Remove um elemento do conjunto e decrementa o tamanho.
* Operações principais:

* Verifica o tipo de estrutura e chama a função correspondente para remoção.
* Atualiza o tamanho do conjunto.
* Complexidade Total: O(log(n) + n), pior caso é o TAD Lista.

5. conjunto_imprimir
* Descrição: Imprime os elementos do conjunto.
* Operações principais:
* Verifica o tipo de estrutura e chama a função de impressão correspondente.
* Complexidade Total: O(n).

6. conjunto_pertence
* Descrição: Verifica se um elemento pertence ao conjunto.
* Operações principais:
* Chama a função de busca correspondente à estrutura escolhida.
* Retorna o resultado da busca.
* Complexidade Total: O(log(n)).

7. conjunto_uniao
* Descrição: Realiza a união de dois conjuntos.
* Operações principais:
* Copia o conjunto A.
* Chama as funções de inserção para adicionar os elementos do conjunto B.
* Complexidade Total: O(2nA + 1 + nB), pior caso é o TAD Lista.

8. conjunto_interseccao
* Descrição: Calcula a interseção de dois conjuntos.
* Operações principais:
* Cria um novo conjunto.
* Chama a função de busca para verificar a presença de elementos nos dois conjuntos.
* Insere os elementos encontrados na interseção.
* Complexidade Total: O(nA log(nA) + nB log(nB)), desconsiderando a complexidade das funções chamadas.

9. conjunto_copiar
* Descrição: Cria uma cópia de um conjunto.
* Operações principais:
* Cria um novo conjunto.
* Chama a função de cópia correspondente à estrutura utilizada.
* Complexidade Total: O(2n + 1), pior caso é o TAD Lista.