Guia de Algoritmos: Listas Encadeadas e Árvores Binárias

1. Inserções e suas variações

O ponto a ser observado em todas as questões de inserção (e qualquer questão que necessite de alocação de memória) é que a alocação pode falhar, resultando num ponteiro NULO. Assim, é preciso retornar FALSO para que o usuário saiba que houve uma falha.

1.1 Inserção no início

Inserções no início seguem uma lógica simples:

1. Faça o ponteiro prox do novo elemento apontar para onde l (início) antes apontava.
2. Faça l apontar para o novo elemento.

1.2 Inserção no final

Para inserir no final, devemos percorrer a lista até que o prox do nó atual ser igual a NULO, significando que chegamos ao fim da lista.

1. Percorrer a lista até o último.
2. Fazer o antigo último apontar para o novo nó.
3. O novo último deve apontar para NULO.

1.3 Inserção em lista ordenada

Inserções em listas ordenadas têm a condição de parada vinculada ao valor que irá compor a informação do nó. Se temos um valor x a ser inserido, devemos percorrer a lista até que a informação do próximo nó seja maior que x (ou encontrar o fim), significando que ali é o local onde deve ser inserido o novo nó para manter a ordem.

1.4 Inserção por posição

Em exercícios de inserção por posição, devemos avançar na lista p − 1 vezes para achar a posição certa onde deve ser inserido o novo nó. As variações desse exercício só diferem entre si na quantidade de ponteiros a serem arrumados, variando para lista simplesmente encadeada e duplamente encadeada.

---

2. Remoções e suas variações

Em todos os exercícios de remoção, a dificuldade reside em reposicionar os ponteiros que apontavam para o elemento que foi removido, podendo variar nos casos de lista dupla e simplesmente encadeada.

2.1 Remoção no início

Caso mais simples: devemos somente fazer o início da lista apontar para onde o primeiro elemento apontava (o segundo elemento) e utilizar a função desaloca no antigo primeiro elemento.

2.2 Remoção no final

Devemos percorrer a lista até estarmos no penúltimo termo (aquele cujo prox aponta para o último). Após isso, basta desalocarmos o último e fazer o nosso penúltimo apontar para NULO.

2.3 Remoção por posição

Para remover um nó em uma posição específica, devemos navegar até o nó anterior ao alvo (p-1). O nó a ser removido será o prox deste anterior. Ajustamos o ponteiro do anterior para pular o alvo e apontar para o seguinte, e então desalocamos o alvo.

2.4 Remoção por valor (Todas as ocorrências)

Diferente de remover apenas a primeira ocorrência, aqui precisamos percorrer a lista inteira. O ponto crítico é: se removermos um nó, não devemos avançar o ponteiro na mesma iteração, pois o nó que assumiu o lugar do removido (o novo vizinho) também precisa ser verificado. Também é necessário um loop específico para verificar se a cabeça da lista precisa ser removida repetidamente.

2.5 Remoção de duplicatas (Lista Ordenada)

Em listas ordenadas, elementos iguais são vizinhos. A lógica consiste em comparar o nó atual com seu próximo. Se forem iguais, removemos o próximo (mantendo o atual) e repetimos a comparação. Se forem diferentes, avançamos o ponteiro.

---

3. Alterações e Manipulações

3.1 Mover último para primeiro

Não envolve troca de valores, mas sim ajuste de ponteiros. É necessário encontrar o penúltimo e o último nó. O último passa a apontar para a antiga cabeça da lista (tornando-se o novo início). O penúltimo aponta para NULO (tornando-se o novo fim). O ponteiro da lista (L) é atualizado para o antigo último.

3.2 Junção por posição (Soma de vizinhos)

Consiste em somar o valor do nó atual com o do próximo e remover o próximo. É uma operação local: atualiza-se o valor do nó atual, segura-se o ponteiro do próximo em um auxiliar, faz-se o atual apontar para o próximo do próximo e desaloca-se o auxiliar.

---

4. Criação e Fusão de Listas

4.1 Intercalação (Merge Sort Logic)

Percorre-se duas listas (L1 e L2) simultaneamente. Compara-se os valores: o menor é inserido na nova lista e o ponteiro da lista de origem avança. Isso se repete até uma das listas acabar. Ao final, copia-se integralmente o restante da lista que sobrou para a nova lista.

4.2 Soma de Polinômios

Similar à intercalação, mas comparando expoentes (geralmente decrescentes).

- Se Exp1 > Exp2, copia-se o termo 1.
- Se Exp2 > Exp1, copia-se o termo 2.
- Se Exp1 = Exp2, soma-se os coeficientes. Regra importante: se a soma dos coeficientes for zero, o nó não é criado.

4.3 Tratamento de Erro (Rollback)

Em exercícios que criam novas listas, se a função aloca falhar (retornar NULO) no meio do processo, é obrigatório percorrer e desalocar todos os nós já criados até aquele momento antes de encerrar a função, para evitar vazamento de memória.

---

5. Árvores Binárias de Busca (ABB)

Estruturas hierárquicas onde, para qualquer nó, todos os elementos à esquerda são estritamente menores e todos à direita são estritamente maiores. Recursão é a ferramenta padrão para navegação.

5.1 Inserção em ABB

A inserção padrão sempre ocorre nas folhas (base da árvore). Navega-se recursivamente: se o valor a inserir for menor que o atual, vai para a esquerda; se for maior, vai para a direita. O caso base é encontrar um ponteiro NULO: é ali que o novo nó será alocado. Os filhos do novo nó nascem apontando para NULO.

5.2 Busca em ABB

Utiliza a mesma lógica de navegação da inserção. Se o valor buscado for igual ao nó atual, retorna sucesso. Se for menor, busca na subárvore esquerda. Se for maior, busca na direita. Se atingir NULO, o elemento não existe.

5.3 Remoção em ABB

A remoção é a operação mais complexa em árvores binárias, pois não podemos deixar "buracos" na estrutura. Ao encontrar o nó a ser removido, caímos em um de três cenários possíveis:

1. **Nó Folha (Sem filhos):** O caso mais simples. Basta liberar a memória do nó e fazer o ponteiro do pai apontar para NULO.
2. **Nó com 1 filho:** O nó a ser removido serve apenas de ligação. Fazemos o pai do nó removido apontar diretamente para o único filho do removido (o "neto" assume o lugar do "pai").
3. **Nó com 2 filhos:** Não removemos o nó fisicamente para não quebrar a árvore. A estratégia é **substituir o valor** (informação) do nó pelo seu **sucessor** (o menor valor da subárvore à direita) ou pelo seu **antecessor** (o maior valor da subárvore à esquerda). Após a troca de valores, chamamos recursivamente a remoção para o sucessor/antecessor (que, garantidamente, terá 0 ou 1 filho, caindo nos casos anteriores).

5.4 Inversão (Espelhamento) de Árvore

Inverter uma árvore binária significa transformar sua estrutura em um espelho (o que era esquerda vira direita e vice-versa).

- A lógica é percorrer a árvore (geralmente em pré-ordem ou pós-ordem) e, para cada nó visitado, realizar um **swap** (troca) entre os ponteiros `esq` e `dir`.
- Após trocar os ponteiros do nó atual, chama-se a recursão para os filhos. O resultado final é que todos os nós maiores que a raiz estarão à esquerda e os menores à direita