Atividade Revisão - 2ªVA - S13

1. Conceitos Básicos e Estrutura de Listas Encadeadas

a) Explique o conceito de lista simplesmente encadeada. Quais são as características principais desse tipo de lista e quais vantagens ela oferece em relação a uma lista sequencial (como um vetor)?

R:Uma lista simplesmente encadeada é uma estrutura de dados que consiste em uma sequência de elementos, onde cada elemento (ou nó) contém um valor e um ponteiro para o próximo nó na sequência. Essa estrutura permite a manipulação dinâmica de dados, onde os elementos podem ser adicionados ou removidos sem a necessidade de realocar ou reorganizar a lista inteira.

As características principais são (Nó; Cabeça; Fim da lista e Tamanho Dinâmico)

Vantagens que ela oferece:

O tamanho dinâmico, que ao contrário de um vetor, que tem um tamanho fixo, uma lista encadeada pode crescer ou encolher conforme necessário, economizando memória.

Sem necessidade de realocação, com listas encadeadas, não é necessário realocar memória quando novos elementos são adicionados. Em vetores, pode ser necessário criar um novo vetor maior e copiar os elementos para ele.

Uso eficiente da memória, onde a memória é alocada conforme necessário, o que pode ser mais eficiente em termos de uso de espaço, especialmente se a lista variar muito em tamanho.

b) Em uma lista duplamente encadeada, qual a diferença na estrutura do nó em comparação com uma lista simplesmente encadeada? Descreva o papel dos ponteiros adicionais que essa estrutura possui.

R: Em uma lista duplamente encadeada, cada nó contém três componentes: um valor, um ponteiro para o próximo nó e um ponteiro para o nó anterior. Isso difere da lista simplesmente encadeada, que possui apenas um ponteiro para o próximo nó. Os ponteiros adicionais na lista duplamente encadeada permitem navegação bidirecional, facilitando o acesso a elementos vizinhos e tornando as operações de inserção e remoção mais eficientes, já que o nó anterior pode facilmente ajustar seu ponteiro ao remover um nó. Essa estrutura oferece maior flexibilidade na manipulação dos dados.
Navegação Bidirecional: O ponteiro que aponta para o nó anterior permite percorrer a lista tanto para frente quanto para trás. Isso facilita operações que exigem acesso a elementos vizinhos ou a iteração reversa pela lista.

Facilidade de Inserção e Remoção: Ao remover um nó, o ponteiro para o nó anterior permite que esse nó ajuste facilmente seu ponteiro para ignorar o nó removido. Isso torna a remoção mais eficiente, pois não é necessário percorrer a lista a partir do início para encontrar o nó anterior.
Acesso a Elementos Anteriores: Com o ponteiro para o nó anterior, é possível acessar rapidamente o elemento anterior, o que pode ser útil em diversas operações, como inserções ou atualizações de dados.

c) O que diferencia uma lista circular de uma lista não circular? Cite um exemplo prático de aplicação onde uma lista circular pode ser mais vantajosa do que uma lista não circular.

R: Lista Circular: Na lista circular, o último nó aponta de volta para o primeiro nó, formando um ciclo. Isso permite que a lista seja percorrida continuamente, sem um fim definido.
Lista Não Circular: Na lista não circular, o último nó aponta para nulo, indicando o fim da lista. A travessia da lista termina quando se chega a esse nulo.

2. Operações em Listas Encadeadas

a) Considere uma lista simplesmente encadeada. Descreva o processo de inserção de um novo nó no início da lista e quais operações devem ser realizadas nos ponteiros para garantir que o novo nó seja o primeiro.

R: Inserção no início de uma lista encadeada é uma operação fundamental que consiste em adicionar um novo elemento (nó) no começo da lista. Essa operação é relativamente simples e eficiente, pois envolve apenas a manipulação de alguns ponteiros.

Criar o Novo Nó: Primeiro, aloque memória para o novo nó e defina seu valor.

Ajustar o Ponteiro do Novo Nó: O próximo ponteiro do novo nó deve ser configurado para apontar para o nó que atualmente é o primeiro da lista. Se a lista estiver vazia, o próximo ponteiro ficará como nulo.

Atualizar o Cabeçalho: Finalmente, o cabeçalho da lista deve ser atualizado para que passe a apontar para o novo nó, tornando-o o primeiro nó da lista.

b) Em uma lista duplamente encadeada, como é realizada a operação de remoção de um nó específico (considerando que o endereço do

nó a ser removido é conhecido)? Descreva o impacto dessa operação nos ponteiros de nós adjacentes.

R: Para remover um nó específico em uma lista duplamente encadeada, primeiro ajustam-se os ponteiros dos nós adjacentes. O ponteiro seguinte do nó anterior ao que está sendo removido deve passar a apontar para o nó seguinte, enquanto o ponteiro anterior do nó seguinte deve apontar para o nó anterior. Se o nó a ser removido for o primeiro da lista, o cabeçalho também precisa ser atualizado. Após esses ajustes, o nó pode ser liberado da memória, mantendo a integridade da lista.

c) Uma lista circular simplesmente encadeada possui um nó "cabeça" apontando para o início da lista. Descreva como seria a implementação da função para inserir um novo nó no final dessa lista.

R: Para inserir um novo nó no final de uma lista circular simplesmente encadeada, primeiro, você deve criar o novo nó e, em seguida, verificar se a lista está vazia. Se a lista estiver vazia, o novo nó deve apontar para ele mesmo, tornando-se o único elemento. Caso contrário, você deve percorrer a lista até o nó que aponta para o cabeçalho, ajustando o ponteiro seguinte desse nó para o novo nó e, finalmente, o ponteiro seguinte do novo nó deve apontar para o cabeçalho. Essa operação garante que o novo nó seja corretamente integrado à estrutura circular da lista.

3. Comparação entre Tipos de Listas
   a) Compare as vantagens e desvantagens de uma lista simplesmente encadeada em relação a uma lista duplamente encadeada. Em quais cenários você escolheria uma em detrimento da outra?

R: As listas simplesmente encadeadas são mais simples e consomem menos memória, já que cada nó possui apenas um ponteiro, tornando-as mais eficientes em termos de espaço. No entanto, elas oferecem acesso unidirecional, o que dificulta operações como a remoção de um nó anterior. Por outro lado, listas duplamente encadeadas permitem navegação em ambas as direções, facilitando inserções e remoções, mas consomem mais memória devido ao ponteiro extra. Nos cenários de escolha, escolheria listas simplesmente encadeadas para aplicações que requerem menos memória e operações simples, enquanto as duplamente encadeadas seriam preferíveis em cenários que demandam acesso bidirecional ou operações frequentes de inserção e remoção.

   b) Liste as principais diferenças entre listas circulares e listas lineares (não circulares). Dê exemplos de casos onde cada tipo seria aplicável.

1. **Estrutura**:
    - **Listas Circulares**: O último nó aponta de volta para o primeiro nó, formando um ciclo. Isso permite que você percorra a lista continuamente.
    - **Listas Lineares**: O último nó aponta para `null`, indicando o fim da lista. A navegação é unidirecional e limitada ao comprimento da lista.
2. **Navegação**:
    - **Listas Circulares**: Você pode começar em qualquer nó e percorrer a lista indefinidamente, o que é útil para certas aplicações.
    - **Listas Lineares**: A navegação deve sempre começar do início, e não há como voltar ao início sem reiniciar a leitura.
3. **Inserção e Remoção**:
    - **Listas Circulares**: Inserir ou remover nós pode ser mais eficiente, pois não há necessidade de verificar o fim da lista.
    - **Listas Lineares**: Pode ser mais simples em casos de manipulação, já que o final da lista é facilmente reconhecido.

Exemplos de Aplicações

- **Listas Circulares**: São úteis em aplicações como jogos (para gerenciar turnos) e em sistemas de buffer, onde a continuidade é importante
- **Listas Lineares**: São mais adequadas para situações em que a ordem e a estrutura sequencial são importantes, como em filas de atendimento ou em listas de tarefas, onde a navegação sequencial e a terminologia clara de "início" e "fim" são desejáveis.

c) Em uma lista duplamente encadeada circular, o que aconteceria se um ponteiro para o nó “cabeça” fosse perdido? Como esse problema poderia ser evitado ou corrigido?

R: Se um ponteiro para o nó "cabeça" de uma lista duplamente encadeada circular for perdido, o acesso à lista se tornaria impossível, pois não haveria uma referência para iniciar a navegação. Para evitar esse problema, uma abordagem é manter um ponteiro adicional que sempre referencia o nó "cabeça", ou implementar um mecanismo de verificação que salve o estado da lista em uma estrutura externa. Se o ponteiro for perdido, a lista pode ser recuperada a partir desse backup ou referência, permitindo que a estrutura permaneça acessível.

4.

a) **R:** O código implementa uma lista encadeada simples, onde cada nó contém um valor inteiro e um ponteiro para o próximo nó. A função inserir_inicio permite inserir um novo nó no início da lista, atualizando o ponteiro da cabeça para apontar para o novo nó. Essa estrutura é eficiente para operações de inserção, que ocorrem em tempo constante, caracterizando a natureza dinâmica da lista encadeada.

.

b)

R:
```
void remover_final(No** cabeca) {
  if (*cabeca == NULL) {
        return;
  }

  No* atual = *cabeca;

  while (atual->proximo != NULL) {
    atual = atual->proximo;
  }

  if (atual->anterior == NULL) {
    free(atual);
    *cabeca = NULL;
  } else {
        atual->anterior->proximo = NULL;
    free(atual);    }
}
```

c) Para verificar se todos os nós em uma lista circular duplamente encadeada foram acessados corretamente, podemos usar um contador e um nó de referência. Começamos a partir da cabeça da lista e percorremos os nós, incrementando o contador até retornarmos ao nó inicial. Se, ao final do percurso, o contador igualar ao número total de nós conhecidos (ou se encontrarmos um nó que já foi visitado), pode concluir que todos os nós foram acessados corretamente, evitando um loop infinito.