Listas circulares e suas aplicações e jogos digitais

Lista circular

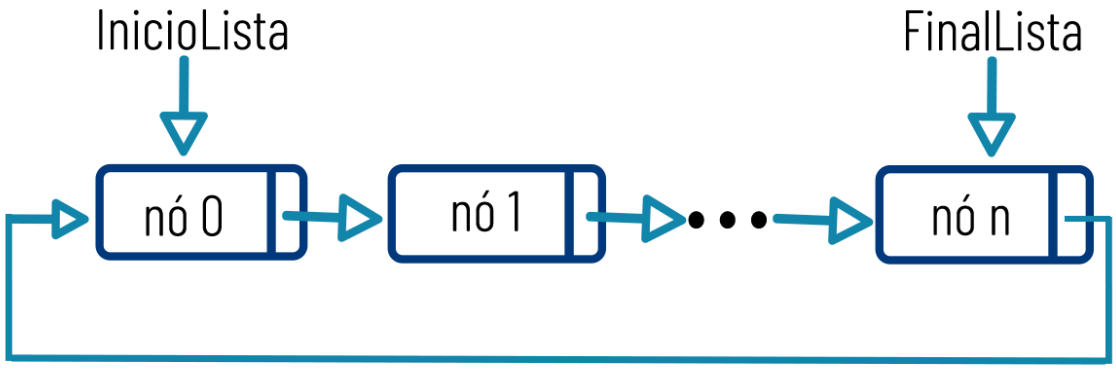
A lista circular é outro tipo de estrutura de dados que não tem fim ou começo. É possível continuar circulando pelos nós indefinidamente

Implementação da lista circular

A ideia da lista circular é permitir a representação de uma lista de dados cíclica, como um *loop* sem fim. Para isso, você precisa fazer com que o último nó seja seguido pelo primeiro, fechando o laço eterno.

No caso encadeado, isso pode ser facilmente resolvido apontando o ponteiro **próximo** do último nó para o primeiro nó, ao invés de apontar para nulo. Se sua lista for duplamente encadeada, você também precisa apontar o ponteiro **anterior** do primeiro nó para o último.

Entretanto, é importante continuar mantendo duas variáveis, uma apontando para o "início" da lista, e outra, para o "final" da fila. Essas duas variáveis são importantes para você fazer as operações de inserção e remoção da lista. Veja a estrutura:



Para o caso de alocação contígua, você simplesmente terá de considerar que o primeiro endereço (índice 0) é o seguinte ao último endereço (índice NúmeroNos - 1), e vice-versa.

Para fazer isso, você pode usar a matemática modular, com módulo **NumeroNos**, ao incrementar ou decrementar as variáveis **inicioLista** e **FinalLista**.

Operações em lista circular

As operações básicas em listas circulares são inserção, remoção e busca.

Inserção em lista circular

A inserção em lista circular funciona exatamente como na lista linear, com a única diferença que se você inserir após o “final” da lista, você deve atualizar a variável que aponta para o final da lista. No caso de ser encadeada, também precisará atualizar os ponteiros associados.

```
def insereCircular(novoNo):
        if inicioLista==None:                    #lista vazia
                inicioLista=novoNo
                finalLista=novoNo
                novoNo.proximo=novoNo
    else:
                finalLista.proximo=novoNo        #insere o nó
                finalLista=novoNo                #atualiza ponteiros
                finalLista.proximo=inicioLista
```

Se a lista for ordenada, você deve realizar a busca pelo local de inserção. Tome cuidado apenas quando a inserção for antes do primeiro nó, para atualizar o ponteiro de início da lista e apontar o último nó para o novo nó inicial.

Complexidade da inserção em lista circular

Se a inserção puder ser feita apenas ao final da lista, requer apenas algumas atribuições, tendo complexidade **O**(1), ou constante.

Caso você use a lista ordenada, a busca pela posição de inserção é **O**(**n**), e a colocação dos nós requer apenas passos constantes, logo, a inserção ordenada é **O**(**n**).

Remoção em lista circular

A remoção em lista circular funciona de forma semelhante à lista linear, precisando apenas atualizar os ponteiros de início e final da lista, caso um dos **extremos** seja removido.

```
def removeCircular(k):
        noAnterior = buscarAnterior(k)
                                                    #busca o nó anterior ao
que será removido
        if noAnterior==None:
        return None                          #erro: nó Alvo não encontrado
        if finalLista==inicioLista :              #lista com nó único
                lista=None                              # remove o nó
                return 0                          #lista agora está vazia, saída
normal
        if noAnterior==finalista:                  #remover nó inicial da lista
              finalLista.proximo=inicioLista.proximo
                 inicioLista=finalLista.proximo
        elif: noAnterior.proximo==finalista:      #remover nó final da lista
                 finalLista=noAnterior
                 noAnterior.proximo=inicioLista
        else:                                  # remoção de um nó que não está
nas pontas
              noAtual=noAnterior.proximo
         noAnterior.proximo=noAtual.proximo
        return 0                                                  # saída normal
```

Complexidade da remoção em lista circular

No caso da implementação encadeada, a busca pelo nó a ser removido é **O**(**n**), e a remoção do nó requer apenas passos constantes, logo, a remoção é **O**(**n**).

No caso de alocação contígua, temos a mesma situação da lista linear. Se a busca levar **x** etapas, todo o resto da lista precisará ser movido, resultando em (**n**−**x**) operações. Dessa forma, a remoção também é **O**(**n**) no caso contíguo.

Aplicações da lista circular em jogos digitais

Utilizando lista circular em jogos

A lista circular pode ser aplicada em situações que necessitem gerar elementos extraídos repetidamente de uma lista limitada, possivelmente seguindo um conjunto especificado de regras. Um exemplo desse tipo de uso são os jogos de cassino.

Exemplo

Você pode implementar uma lista circular de 38 nós que representa uma roleta, com um conjunto não ordenado de nós representando os 38 números da roleta em um arranjo não sequencial. Você, então, pode manter um ponteiro para um elemento da lista que representa a bolinha da roleta. Nesse caso, rodar a roleta seria apenas decidir um número de deslocamento, que seria quantos movimentos você percorre na lista, e o ponteiro apontará para o novo resultado.

Entretanto, o uso mais comum para uma lista circular em jogos digitais é representar um conjunto de jogadores, permitindo que a cada um seja dada sua vez de agir, de forma sequencial, voltando a vez para o primeiro, após o último agir.

A esse esquema, no qual cada um pode ter uma oportunidade de agir, damos o nome de Round Robin. E listas circulares são formas extremamente práticas de implementar sistemas Round Robin.

Aplicação de fila em alocação contígua em jogos digitais

Uma fila (na qual os nós só podem ser removidos do início e adicionados ao final) em alocação contígua, que ocupa um espaço limitado e fixo, é uma estrutura muito usada em jogos digitais on-line.

Como a velocidade de transmissão de dados pode ser menor que a velocidade de geração de dados, a estrutura citada, no caso, a fila, funciona como um buffer, armazenando os dados até eles serem transmitidos.

A grande vantagem é que o espaço de memória é fixo, e conforme os nós vão sendo consumidos da fila, o espaço vai sendo liberado para novos dados. Essa estrutura pode ser usada para manter os sistemas de conversa (chat) dos jogos, e também para armazenar os comandos introduzidos pelo jogador em seu dispositivo local, até que sejam consumidos pelo servidor.

A complexidade de inserção e remoção é a melhor possível: constante. O espaço de memória é fixo e não precisa ser reajustado e nem cresce com o tempo. Portanto, é uma estrutura ideal para esse tipo de aplicação.