Universidade Federal do Rio de Janeiro

Tópicos de Engenharia de Dados B: Estrutura de Dados

Luiza Costa Pacheco - DRE: 119169147
Eric Silva Kraus - DRE: 115104789
Felipe Gomes Táparo - DRE: 119959237

Hierarquia de Estrutura de Dados Lineares

Rio de Janeiro, 04 de Novembro de 2023

Sumário

Classe Array

A classe array foi implementada no arquivo array.py. Como foi utilizado a linguagem Python para tal trecho da tarefa, e, tendo em vista que a funcionalidade array não é nativa da linguagem, seu efeito foi simulado com a utilização de listas, para cumprir as tarefas pré-determinadas.

Foi criada uma classe "Array", com as propriedades self.name e self.lista e então criadas as funções descritas a seguir, que são chamadas externamente por uma rotina escrita externamente à classe, no final do arquivo. Inicialmente, dentro de um loop, pergunta-se ao usuário, se deseja criar um novo array. No caso de uma resposta negativa, encerra-se a rotina e o programa; no caso de uma positiva, solicita-se que o usuário nomeie o array a ser criado, para fatores de formatação e de tornar a interface mais amigável ao usuário (um projeto adicional, no entanto fora do escopo inicial, seria permitir ao usuário acessar os diversos arrays criados, a partir de seus nomes). Nomeado o array, dá-se início à criação do mesmo, com a função "CriaArray".

Criação do Array

Na função "CriaArray", é solicitado ao usuário o *input* de um item a ser adicionado ao array e então o mesmo é indagado se deseja adicionar mais um item. Este procedimento se repete, em um loop, até que o usuário opte por não inserir mais nenhum item. Ao final do procedimento, o usuário é apresentado a um *print* do array criado.

Vale ressaltar que foram aplicadas medidas de exceção, caso a resposta do usuário a respeito de introduzir ou não mais itens não fosse dentro do esperado.

Demais Operações

Após a formação do array, são proporcionadas ao usuário 4 escolhas: gerenciar os itens internos ao array, checar seu comprimento, inspecionar o array como um todo ou seus itens em particular, e terminar a rotina.

Gerenciar os Itens do Array

A primeira opção e também maior função desta classe, "ManageItems" começa apresentando ao usuário 4 opções: inserir um item, substituir um item, remover um item, e a tradicional opção de retornar ao menu anterior, implementada também em todas opções desta função.

Decidindo por inserir um item, o usuário pode optar por inseri-lo no começo do array, no final, em determinada posição (movendo os itens subsequentes uma posição à frente), antes de um item específico, e depois de um item específico. Em todos os casos, durante cada processo, é solicitado ao usuário que forneça o item a ser adicionado.

- Inserir um item no começo, utiliza a função padrão *insert*, realizando: **self.lista.insert(0,"item inserido")**.
- Inserir um item ao fim, utiliza a função *append* (que por padrão, sempre adiciona itens ao fim da lista), com: **self.lista.append("item inserido")**.
- Inserir um item em determinada posição, também utiliza a função *insert*, no entanto, desta vez perguntando previamente ao usuário o índice da posição a ser inserida. Conforme proposto, caso o índice exceda o índice do último item da lista por mais de "1", são introduzidas strings vazias à lista até que a mesma tenha comprimento suficiente para que o item possa ser adicionado na posição desejada.
- Inserir antes ou depois de um item específico, solicita ao usuário que informe o item a ser usado como referência, e retorna uma mensagem de erro, caso o mesmo não faça parte da lista, solicitando o fornecimento de um novo item, até que este possa ser encontrado na lista. Posteriormente, é obtido o índice de determinado item, por meio da operação **self.lista.index("item de referência")**, e então é adicionado o item desejado, novamente por meio da função *insert*, adicionando-se "-1" ou "+1" ao índice do item referenciado.

Decidindo por substituir um item, é perguntado ao usuário se deseja fazer isto por meio do índice do item a ser substituído, ou por meio de seu nome. Novamente, em ambos os casos, é solicitado ao usuário que informe o novo item.

- Ao optar pelo método do nome, é solicitado ao usuário que forneça o nome do item a ser substituído. Caso o nome fornecido não conste na lista, é retornado ao usuário "*"Referenced item not present in "+self.name*", e solicitado um novo nome. Caso o nome fornecido faça parte da lista, é realizado: **self.lista[self.lista.index("item de referência")]="novo item"**.
- Ao optar pelo método do índice, é solicitado ao usuário que forneça o índice do item a ser substituído. Caso o índice fornecido seja maior que o comprimento da lista, é retornado ao usuário "*IndexError: list index out of range*", e solicitado um novo índice. Caso o índice fornecido seja aceitável, é realizado: **self.lista[int("índice fornecido")]="novo item"**.

Decidindo por remover um item, o usuário tem a opção de remover o primeiro item, o último item, um item determinado por seu índice, e um item determinado por seu nome.

- O primeiro item é removido por meio de **self.lista.pop(0)**.
- O último item é removido por meio de **self.lista.pop(len(self.lista)-1)**.
- Remover um item por seu índice é feito com **self.lista.pop(int("índice fornecido"))**, e, caso o índice fornecido não seja menor que o comprimento da lista, é retornado ao usuário o erro "*IndexError: list index out of range*", solicitando um novo índice.
- Remover um item por seu nome é feito com **self.lista.remove("item de referência")** e, caso o nome fornecido não conste na lista, é retornado ao usuário o erro "*"Referenced item not present in "+self.name*", solicitando um novo nome.

Checar o Comprimento do Array

A função "LenArray" é uma função simples que utiliza a função *len* para retornar ao usuário, com uma formatação amigável à leitura, o valor do comprimento do array que foi formado.

Inspecionar o Array

A inspeção do array e de seus itens é realizada pela função "CheckArray", que oferece duas opções ao usuário: inspecionar o array como um todo, ou seus itens discriminadamente.

- A primeira opção simplesmente retorna um print do array como se encontra no momento.
- Inspecionar os itens individualmente permite que o usuário opte por inspecionar o primeiro item, o último, um especificado por seu índice, ou receber o índice de um item, dado o seu nome.
    - O primeiro item é retornado por **print(self.lista[0]).**
    - O último item é retornado por **print(self.lista[len(self.lista)-1])**
    - O item especificado por seu índice é retornado por **print(self.lista[int("indice")])**, caso o índice em questão seja menor ou igual ao comprimento do array menos um. Caso contrário, é retornado o erro *"IndexError: list index out of range"*, e solicitado um novo índice.
    - O índice do item especificado por seu nome é retornado por **print("The item's index is: "+str(self.lista.index("nome fornecido")))**, caso o nome encontre-se no array. Caso contrário, é retornado o erro "*"Referenced item not present in "+self.name*", e solicitado um novo nome.

Classe Pilha (stack)

A classe pilha foi implementada no arquivo “pilha.h”, utilizando argumento template. Essa classe tem funcionamento equivalente ao trabalho anterior. A classe pilha utiliza da classe lista encadeada simples como estrutura de dados.

Os métodos da classe pilha são:

- push(T)

Adiciona um elemento ao início da pilha, caso não esteja cheia. Se a pilha estiver cheia, uma exceção é gerada.

- T pop()

Retira e retorna o primeiro elemento da pilha.

- void swap()

Troca os dois primeiros elementos de lugar.

- T& peak()

Retorna o primeiro elemento da pilha sem retirar.

- bool empty()

Indica se a pilha está vazia.

- bool full()

Indica se a pilha está cheia.

- unsigned size()

Indica quantos elementos estão na pilha

- unsigned capacity()

Indica o tamanho máximo da pilha

- Pilha(unsigned)

Construtor recebendo tamanho máximo como argumento.

- Pilha(Pilha<T>&)

Construtor de cópia.

A classe pilha foi testada no problema torre de hanoi, um exemplo do output do programa está presente no repositório.

Classe Lista Encadeada Simples

A lista encadeada simples foi implementada no arquivo “lista.h”. A classe foi baseada no modelo disponibilizado pelo professor. O arquivo “lista.h” implementa também uma classe utilitária “C_no_lst_encadeada_simples”, utilizada pela classe principal “C_lista”. A classe foi feita com a utilização de argumentos templates.

Os métodos da classe lista são:

- void insere_no_inicio(T)

Adiciona um elemento ao início da fila.

- void insere_no_final(T)

Adiciona um elemento ao final da fila.

- T pop()

Retira o primeiro elemento da lista e retorna.

- T at(unsigned)

Retorna o elemento da posição especificada.

- void percore_lista()

Exibe todos os elementos da lista no terminal.

- cp(C_lista<T>&)

Faz uma cópia de outra lista.

A classe Pilha foi implementada utilizando essa classe para testes.

Classe Lista Encadeada Dupla

Desenvolvimento

Classe Node

Representa cada elemento de uma lista encadeada dupla, ou seja, um nó da lista.

Cada nó possui uma referência do elemento que vem antes e depois deste, representado por ponteiros. A lista é inicializada com ponteiros sem referência (apontando para o nada).

É permitido que a classe Doubly Linked List altere valores privados dos nós.

Classe Doubly Linked List

Representa uma lista completa, contendo diversos nós. Mas na realidade, só necessita de uma referência do primeiro e último nó da lista, e conta com esses nós para encontrar os demais nós.

Possui o atributo length para facilitar, sendo alterado a cada adição ou remoção de nós, mas é possível validá-lo percorrendo a lista.

- Inicializador: DoublyLinkedList(initializer_list<int> items)

Recebe uma lista de itens e envia cada item para a função “push” em ordem.

- Inserção: void push (int data)

Recebe um item e o adiciona na lista, fazendo com que o ponteiro “head”, referente ao primeiro item da lista, aponte para ele. Atualiza os ponteiros do antigo “head”, se houver.

- Inserção: void push_back (int data);

Igual ao anterior, mas em vez de “head”, realiza as alterações com o “tail”, a referência para o último nó.

- Remoção: int pop ();

Remove o atual “head”, o segundo nó passa a ser “head”.

- Remoção: int pop_back ();

Remove o atual “tail”, o segundo nó passa a ser “tail”.

- Percorrer: void traverseList (Directions direction);

Utiliza uma referência de direção para percorrer e imprimir na tela a lista, sendo ela de cima pra baixo ou de baixo pra cima (primeiro ao último nó ou do último ao primeiro nó)

- Pegar informação: int getLength();

Retorna o valor da variável length, o comprimento da lista.

- Procurar: Node *searchFromIndex(int index);

Recebe um valor inteiro positivo menor que o tamanho da lista. Retorna o n ésimo nó, sendo n o valor recebido: List[n], fazendo referência à vetores.

- Trocar: void swapNodes(int node1, int node2);

Utilizando a função de busca de nós, realiza a troca de 2 nós qualquer, caso existam.

- Ordenação: void bubbleSort();

Utilizando as funções de troca e procura de nós, realiza a ordenação da lista, do menor para o maior.

A função percorre a lista a partir do head, comparando 2 nós consecutivos e trocando-os de ordem, caso haja necessidade. A lista é percorrida diversas vezes, até que seja percorrida sem realizar nenhuma troca, podendo presumir que a lista está ordenada.

Classe Priority Queue

Representa uma lista de prioridades, uma lista que está sempre ordenada do maior para o menor. Ela foi feita utilizando uma lista encadeada dupla.

- Inicializador: PriorityQueue(initializer_list<int> items);

Recebe uma lista de itens e os envia como inicializador de uma lista encadeada. Logo em seguida, com a lista criada, ordena-os com bubble sort.

- Inserção: void insertItem(int data);

Insere um item no início da lista. Se o item for maior que o antigo head, o conteúdo dos nós troca de lugar. O ciclo se repete, conferindo se o dado inserido é maior que o dado seguinte. Quando for menor ou igual, não há mais necessidade de troca, podendo considerar que a lista está ordenada e o dado foi inserido no local correto.

- Percorrer: void traverseList();

Percorre a lista usando a própria função da lista encadeada do primeiro até o último item.

Função Principal Main

Possui um menu para teste das classes: um específico para lista encadeada e outro para a fila de prioridades. O menu da fila de prioridades pode ser acessado saindo do menu da lista encadeada.

Resultados Obtidos

As listas foram criadas com sucesso. Classes poderiam estar melhor padronizadas. get e set node não são necessárias, já que nós e a lista são classes amigas.

Classe Lista Circular Encadeada Simples

Desenvolvimento

Classe Node

Representa cada elemento de uma lista circular, ou seja, um nó da lista.

Cada nó possui uma referência do elemento que vem depois desse, representado por ponteiro. A lista é inicializada com o ponteiro sem referência (apontando para o nada).

É permitido que a classe Circular List altere valores privados dos nós.

Classe Circular List

Representa uma lista circular completa de tamanho fixo, contendo diversos nós. Mas na realidade, só necessita de uma referência do primeiro nó da lista, e conta com esse nó para encontrar os demais nós.

Possui o atributo size para facilitar, mas é possível validá-lo percorrendo a lista.

- Inicializador: CircularList(int size);

Recebe somente o tamanho da lista e cria essa mesma quantidade de nós. O primeiro nó é denominado “head”. Cada nó criado é ligado ao nó anterior, até chegar ao último nó, que é ligado ao head.

- Inserção: void insertData (int data, int index);

Insere um dado em um local dado da lista. É necessário percorrer a lista até chegar ao determinado local dela.

- Percorrer: void traverseList ();

Percorre e imprime na tela a lista, do primeiro ao último nó.

- Renovar: Node *removeData (int index);

Remove o dado do nó de determinada posição e retorna uma cópia do que era antes da remoção.

- Procurar: Node *checkData (int index);

Retorna uma referência ao nó na posição indicada.

- Trocar: void swap(int index);

Utilizando a função de busca de nós, realiza a troca de 2 nós consecutivos, caso existam.

- Ordenação: void bubbleSort();

Utilizando as funções de troca e procura de nós, realiza a ordenação da lista, do menor para o maior.

- Pegar informação: int getLength();

Retorna o valor da variável length, o comprimento da lista.

Classe Polygon

Utiliza duas listas circulares, representando as coordenadas em x e em y dos pontos que formam um polígono.

- Inicializador: Polygon (int size);

A partir da quantidade de pontos do polígono, cria as listas circulares com esse tamanho.

- Inserir: void insertDataX(int data, int index);

Envia um dado e uma posição para que a lista circular referente aos pontos em x a adicione.

- Inserir: void insertDataY(int data, int index);

Envia um dado e uma posição para que a lista circular referente aos pontos em y a adicione.

- Percorrer: void printPoints();

Imprime na tela o conjunto de pontos em x e em y.

Função Principal Main

Possui um menu para teste das classes: um específico para lista circular e outro para o polígono. O menu do polígono pode ser acessado saindo do menu da lista circular.

Resultados Obtidos

Tive problemas de interpretação com a lista circular, mas acredito que esteja funcionando como pedido.

Já no polígono, não consegui fazer o exercício pedido de identificar se duas linhas se cruzam e organizá-las. Acredito que a forma escolhida de estrutura de dados seja possível realizar a tarefa, mas não acho que seja ideal.

Classe Fila Simples (queue)