ID: 372027433

Name: Abdullah Salem basalamah

Array List

|  |  |
| --- | --- |
| Data Structure name and description | Data Structure name is Array List. It stores data in a linear fashion. It uses the array as its building block. It’s similar to the array but with this data Structure you can assign the size of the list to be either dynamic –run time -- or static – compilation time-- . |
| Usage and applications | This is the array is the simplest data structure. we construct an array than we insert elements in the array. In case we needed to delete item, we shift the whole array. We can insert at a given position, last or at first. The same goes for deletion. We track the list by a tracker to help us in insertion or deletion.  Applications: we can use it to store different variables entered from the user and then make arithmetic operations on it. |
| Various methods of implementation with advantages and limitations of each method | \* Insert: we either insert a value to last place or first place or place of our choice.  \* delete: in deletion we can delete specific element or the last element in the array.  \* isEmpty: this method checks if the array is empty of not. |
| Algorithms available for searching for an element and their complexity (big O) | If we’re given a position then the time complexity is O(1).  If we’re given a number and apply a search for it then the complexity is O(n). |
| Example codes (you can list one or two methods of the data structure class (such as insert or delete) | **public** **void** insert(**int** value, **int** loc) {  **if**(loc > tracker || isEmpty()) {  System.***out***.println("Enter appropriate location!!");  **return** ;  }    **int** shift = tracker +1 ;    **while**(shift > loc) {  storage[shift] = storage[shift-1];  shift-- ;  }  storage[loc]= value ;  tracker++;  } |

LinkedList

|  |  |
| --- | --- |
| Data Structure name and description | Data Structure name is LinkedList. It allows you store as much elements as you want. One disadvantage is that you only can access from the first root. |
| Usage and applications | The linkedList is constructed of nodes. The node has data and a reference to next element. First element is head which is the start of this data structure and you can use it to access other elements.  Applications: same application used for array list can be used with linked List but if we’re concerned about the space we use linked list. |
| Various methods of implementation with advantages and limitations of each method | * Insert: we can do all kinds of insertion which they insert First, insert Last, insert In Position. * Delete: we can do all delete operation. We can delete head or last or at a given position |
| Algorithms available for searching for an element and their complexity (big O) | The time complexity is always O(n) because we’re always starts from the head node and traverse to either the element we need. |
| Example codes (you can list one or two methods of the data structure class (such as insert or delete) | **public** **void** insertInth(Node afterNode, **int** data) {    Node temp = head ;  **while** (temp != afterNode) {  temp = temp.next;  }  Node newNode = **new** Node(data);  newNode.next = temp.next;  temp.next = newNode ;  } |

|  |  |
| --- | --- |
| Data Structure name and description | Data Structure name is Stack. This data structure fallows Last In First Out(LIFO). It either uses linkedList or array as its mean storage element and it has different methods for insertion and deletion. |
| Usage and applications | The data are always inserted from one end, the other end is closed if we can say. There’s a tracker for the data that it had been inserted in the stack. Using this tracker we can know it the stack is empty or not  Application: memory of the device, evaluate prefix, postfix and infix expressions. |
| Various methods of implementation with advantages and limitations of each method | * Push: this method used to insert an element at the top of the stack using the *top* tracker, you can insert only one element per push operation. * Pop: with this method you can delete a value from the stack, some people make a return for the element in the top. But this is only different implementation. * Peek: this method gives you the element at the top so it’s proper to use this element to return the top. |
| Algorithms available for searching for an element and their complexity (big O) | Search here is O(n) because we have to search in an array. |
| Example codes (you can list one or two methods of the data structure class (such as insert or delete) | **public** **void** push(**int** data) {  **if** (top == stack.length-1) {  System.***out***.println("The stack if full");  }    **else** {  top++ ;  stack[top] = data ;  }  }    **public** **int** pop() {  **int** data = 0;  **if** (isEmpty()) {  System.***out***.println("The stack is empty !!");  }    **else** {  top--;  data = stack[top];    }  **return** data;  } |

Stack

|  |  |
| --- | --- |
| Data Structure name and description | Data Structure name is Queue. This data structure uses method called First In First Out(FIFO). You can use either linkedlist or array implementation as the main storage blocks. There’s another form which is circular queue. |
| Usage and applications | In this data Structure you use two types of tracker: front, back to track the insertion of the data in the queue. You add elements to last and remove from first (which is the element that is processed)  Applications: when requests go to the server the first request come will be executed first. Shared resources like printer. |
| Various methods of implementation with advantages and limitations of each method | * Enqueue: this method allows you to enter one element in the queue, when you use it you increase the *back* tracker by one. * Dequeue: allows you to delete an element from the queue. It’s just as if you deleted it. You increase the *front* tracker so the you can’t get to the data. * Front: this method returns the element that the front tracker is pointing to. |
| Algorithms available for searching for an element and their complexity (big O) | Search here is O(n) because we have to search in an array. |
| Example codes (you can list one or two methods of the data structure class (such as insert or delete) | **public** **void** enqueue(**int** value) {  **if**(!(back < queArray.length-1) ) {  System.***out***.println("Queue is Full!");  }  **else** {  back++;  queArray[back] = value ;  count++ ;  }  }  **public** **void** dequeue() {  **if** (isEmpty()) {  System.***out***.println("No values in the queue! ");  }  **else** {  front++;  count--;  }  } |

Queue

Tree

|  |  |
| --- | --- |
| Data Structure name and description | Data Structure name is tree. This type of Data Structure is non-linear meaning there’s no sequential order for the data. trees are constructed from root and leaves nodes. There are different types of tree most known one is Binary Search Tree. |
| Usage and applications | Tree are constructed of nodes each node has left, right and data in case of binary search tree. Roots are always the start of the tree and then for each node you have one  Applications: you can use it to evaluate expressions. |
| Various methods of implementation with advantages and limitations of each method | * Add: you can use this method to enter new nodes to the tree. The newly added node is always a leaf node. * Max: this method returns the largest element in the tree by going to the right of the tree until you hit a null value when that happens, that means you get the largest element in the tree. * Min: this method returns the smallest element in the array. By going to the left of the tree using recursion. When you hit null value this means you reached the smallest element. * Display levelorder: this method prints the whole tree by levels. You first print the root value to go to next levet and print node from left to right. * Display preorder: this method prints the tree in this order   root > left > right.   * Display inorder: this method prints the tree in this order   Left > root > right .   * Display postorder: this method prints the tree in this order   Left > right > root.   * Delete: this method let you delete an element from the tree. Here there are 3 possibilities. First if the node is leaf. Second, if the node has one child. Third, if the node has 2 children. Each case has specific way of deleting the node so you don’t ruin the order of the tree. |
| Algorithms available for searching for an element and their complexity (big O) | The search in BST is O(log n). |
| Example codes (you can list one or two methods of the data structure class (such as insert or delete) | **public** **void** preOrder(Node current ) {  System.***out***.print(current.data + " ");  **if**(current.left != **null**)  preOrder(current.left);    **if**(current.right != **null**)  preOrder(current.right);  } |