DATA STRUCTURES AND ALGORITHMS

Hoang Cong Du - Bh00940

SE06302

Contents

01

What is DSA?

02

What is ADT?

03

Compare different between Stack and Queue

04

How many ways are there to implement Stack and Queue?

05

compare complexity between 2 sorting algorithms

06

Two network shortest path algorithms.

WHAT IS DSA?

Data structures are formats used to efficiently organize, store, and access data in computer memory. Each serves different purposes: arrays store fixed-size collections of similar items, while linked lists allow for dynamic addition and removal, useful for tasks such as to-do lists. Common data structures include arrays, linked lists, stacks, queues, trees, and graphs. Understanding these structures is essential to building efficient algorithms, as they play a key role in processing, retrieving, and managing data in almost any software system.

What is ADT?

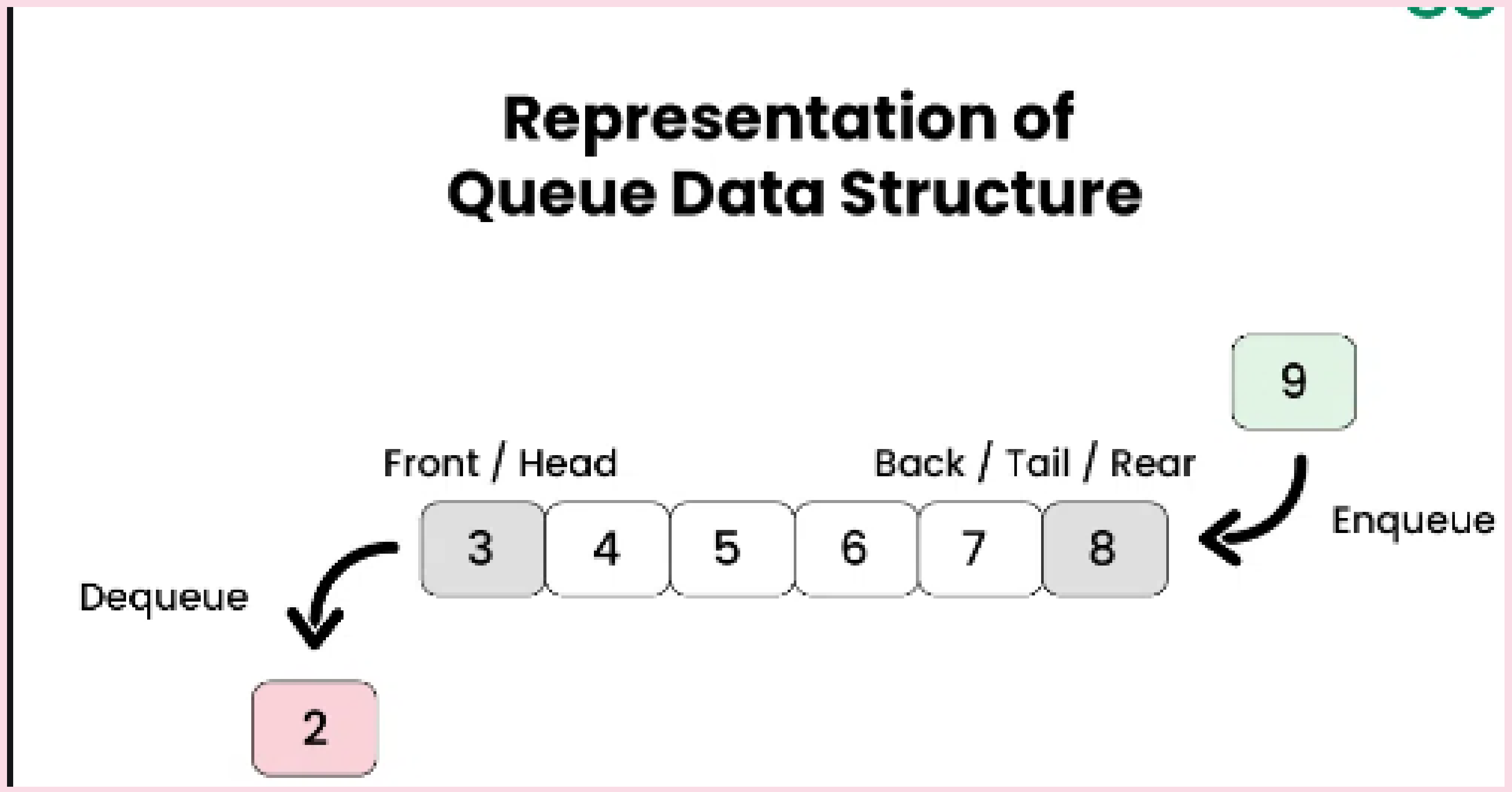
An abstract data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations. The definition of an ADT only states what operations will be performed but not how these operations will be implemented. It does not specify how the data will be organized in memory and what algorithms will be used to implement the operations. It is called "abstract" because it provides an implementation-independent view.

Quay lại Trang Chương trình

FIFO

A FIFO queue is a queue that operates on a first-in, first-out (FIFO) principle. First-in, first-out means that the request (like a customer in a store or a print job sent to a printer) is processed in the order in which it arrives. A first-come, first-served line is the most common type of queue that we join in our everyday lives and is generally accepted as the fairest way to operate a queue.
In queuing theory, the rule governing queue operation is known as queuing discipline. Besides first-in-first-out queues, other queuing disciplines include last-in-first-out, prioritized, and serve-in-random-order.
The online queues in Queue-it's virtual waiting room are based on the first-in, first-out queuing discipline.

Queue Data Structure



Compare different between Stack and Queue

Parameters	Stack	Queue
Working Principle	It follows the LIFO (Last In First Out) order to store the elements, which means the element that is inserted last will come out first.	It follows the FIFO (First In First Out) order to store the elements, which means the element that is inserted first will come out first.
Pointers	It has only one end, known as the top, at which both insertion and deletion take place.	It has two ends, known as the rear and front, which are used for insertion and deletion. The rear end is used to insert the elements, whereas the front end is used to delete the elements from the queue.
Operations	The insertion operation is known as push and the deletion operation is known as pop.	The insertion operation is known as enqueue and the deletion operation is known as dequeue.
Empty Condition	The condition for checking whether the stack is empty is top ==-1 as -1 refers to no element in the stack.	The condition for checking whether the queue is empty is front == -1

Compare different between Stack and Queue

Full Condition	The condition for checking if the stack is full is top==max-1 as max refers to the maximum number of elements that can be in the stack.	The condition for checking if the queue is full is rear==max-1 as max refers to the maximum number of elements that can be in the queue.
Variants	There are no other variants or types of the stack.	There are three types of queues known as circular, double-ended, and priority.
Implementation	It has a simple implementation compared to queues as no two pointers are involved.	It has a complex implementation compared to stacks as two pointers front and rear are involved.
Application	It is used to solve recursion-based problems.	It is used to solve sequential processing-based problems.
Data Representation	Often implemented with arrays or linked lists.	Can also be implemented with arrays or doubly linked lists.
Example	A real-life example of a stack can be the Undo/Redo operation in Word or Excel.	A real-life example of a queue can be an operating system process scheduling queues.
Visualization	Stack can be visualized as a vertical collection.	Queue can be visualized as a horizontal collection.

Implementing Stack

Array-based stack: Uses a fixed-size array to store elements. Operations like Push and Pop are simple, but the stack has a fixed capacity and resizing can be expensive.

Linked-list stack: Uses a linked list where each element points to the next element. The Push and Pop operations occur at the top, making it more flexible than an array-based stack because it does not have a fixed size.

Dynamic array-based stack (resizable array): Starts with an initial capacity but expands dynamically (like in Java's ArrayList). This combines the simplicity of an array with the flexibility, but resizing operations can add cost.

Implementing Queue

Array-based queue: Uses an array with two pointers for the front and back. Fixed array queues can suffer from the circular wraparound problem, which can be solved by using a circular queue (where the end connects back to the start).

Linked list-based queue: Each element points to the next element, with a head pointer for Dequeue and a tail pointer for Enqueue. This implementation does not require resizing and works well with dynamic memory.

Double-ended queue (Deque): Deque allows insertion and deletion at both ends. Although designed for double-ended operations, it can be used as a queue by restricting operations to one end.

Priority queue: A specialized queue in which each element has a priority and elements are removed from the queue based on priority rather than order. This is often implemented with a heap or binary tree and is useful in scheduling applications.

Sorting algorithms

Bubble Sort

Bubble Sort is a simple sorting algorithm that iteratively steps through the list to be sorted, comparing adjacent elements and swapping them if they are out of order. This process is repeated until the list is sorted. The algorithm is called “bubble sort” because, like bubbles rising to the surface, the largest unsorted elements gradually move to their correct positions at the end of the list.

Merge Sort

Merge Sort is a recursive divide-and-conquer sorting algorithm that divides a list into smaller sublists until each sublist contains only one element. It then merges the sublists together in sorted order. This algorithm is known for its efficiency and stability, making it suitable for large data sets.

Compare complexity between 2 sorting algorithms

Time Complexity

Bubble Sort:
Best Case: O(n)O(n)O(n) (when the array is already sorted; can terminate early if no swaps are made).
Average and Worst Case: O(n2)O(n^2)O(n2), as Bubble Sort compares and potentially swaps every element with every other element in each pass.
Merge Sort:
Best, Average, and Worst Case: O(nlogn)O(n \log n)O(nlogn). Merge Sort consistently splits the array and merges in O(logn)O(\log n)O(logn) passes, each of which involves an O(n)O(n)O(n) operation, resulting in O(nlogn)O(n \log n)O(nlogn).

Space Complexity

Bubble Sort: O(1)O(1)O(1). Bubble Sort performs sorting in-place, requiring no additional space apart from a few variables for tracking swaps.
Merge Sort: O(n)O(n)O(n). Merge Sort requires additional space to store temporary arrays during the merge phase, as it doesn’t sort in-place.

Two network shortest path algorithms.

Dijkstra algorithms

Dijkstra's algorithm is an algorithm for finding the shortest path from a source vertex to all other vertices in a non-negative weight graph. It is a very popular algorithm for computing shortest paths and is widely used in systems such as GPS, networks, and path optimization applications.

Prim-Jarnik Algorithm

The Prim-Jarnik algorithm (also known as Prim's algorithm) is a greedy algorithm for finding the Minimum Spanning Tree (MST) for a connected graph with non-negative weights. This algorithm constructs the minimum spanning tree by sequentially adding the shortest edges without creating cycles.

How Dijkstra's algorithm works

Initialization:

Set the distance from the source vertex to itself to 0 and all other vertices to infinity (∞).

Mark all vertices as unvisited.

Repeat until all vertices are visited:

Choose the unvisited vertex with the smallest distance from the source vertex and set it as the current vertex.

Mark the current vertex as visited.

Update the distances of the adjacent vertices of the current vertex. If the sum of the distances from the source vertex to the adjacent vertex (via the current vertex) is less than the previously known distance, we update this distance.

End:

After all vertices have been visited, the distance from the source vertex to each vertex in the graph will be the shortest distance.

Complexity of Dijkstra's Algorithm

Time:

With a priority queue, the complexity is O(ElogV) where

E is the number of edges and

V **is the number of vertices.**

Space:

$O(V)$**for the distance array and the vertex marker array.**

Limitations of Dijkstra's Algorithm

Only works correctly for graphs with non-negative weights.

Cannot be applied to graphs with negative weight edges.

Principle of Prim-Jarnik Algorithm

Initialization:
Choose any vertex as the starting vertex and add it to the spanning tree.
Mark all vertices as not yet added to the spanning tree.
Assign weights to the edges connecting the selected vertex and the vertices not yet added to the tree.
Repeat the following steps until all vertices are in the spanning tree:
Choose the edge with the smallest weight among the edges connecting the vertices already in the spanning tree and the vertices not yet added.
Add the vertex connected to the shortest edge to the spanning tree.
Update the edges and weights associated with the newly added vertex.
End:
When all vertices are in the spanning tree and there are no cycles, the algorithm will terminate, and the minimum spanning tree will be built.

Complexity of Prim-Jarnik Algorithm

Time:
Using priority queue, the algorithm has complexity of
O(ElogV), where
E is the number of edges and
V is the number of vertices.
Space:
O(V+E), to store the graph and priority queue.

Applications of Prim-Jarnik Algorithm

Optimizing networks (e.g. power grids, fiber optic networks).
Connecting nodes in the network without repeating routes, saving connection costs.

Thank You

For Listening!