

School of Computer Science and Artificial Intelligence

Lab Assignment # 11.1

Program	: B. Tech (CSE)
Specialization	: -
Course Title	: AI Assisted Coding
Course Code	: 23CS002PC304
Semester	: II
Academic Session	: 2025-2026
Name of Student	: G.Maniprasad
Enrollment No.	: 2403A51L48
Batch No.	: 51
Date	: 17/02/26

Submission Starts here**Screenshots:****Task Description #1 – Stack Implementation**

Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Sample code:

```
class Stack:  
    Pass
```

Prompt:

generate a Stack class with push, pop, peek, and is_empty
Methods and code.

Code:

```

1 class Stack:
2     def __init__(self):
3         """Initialize an empty stack."""
4         self.items = []
5
6     def push(self, item):
7         """Add an item to the top of the stack.
8
9         Args:
10             item: The item to be added to the stack.
11         """
12         self.items.append(item)
13
14     def pop(self):
15         """Remove and return the top item from the stack.
16
17         Returns:
18             The top item from the stack.
19
20         Raises:
21             IndexError: If the stack is empty.
22         """
23         if self.is_empty():
24             raise IndexError("Cannot pop from an empty stack")
25         return self.items.pop()
26
27     def peek(self):
28         """Return the top item from the stack without removing it.
29
30         Returns:
31             The top item from the stack.
32

```

```

33         Raises:
34             IndexError: If the stack is empty.
35         """
36         if self.is_empty():
37             raise IndexError("Cannot peek at an empty stack")
38         return self.items[-1]
39
40     def is_empty(self):
41         """Check if the stack is empty.
42
43         Returns:
44             True if the stack is empty, False otherwise.
45         """
46         return len(self.items) == 0
47
48     def size(self):
49         """Return the number of items in the stack.
50
51         Returns:
52             The number of items in the stack.
53         """
54         return len(self.items)
55
56     def __str__(self):
57         """Return a string representation of the stack.
58
59         Returns:
60             A string representation of the stack.
61         """
62         return f"Stack({self.items})"

```

```

63
64
65 # Example usage and testing
66 if __name__ == "__main__":
67     # Create a new stack
68     stack = Stack()
69
70     # Test is_empty on empty stack
71     print(f"Is stack empty? {stack.is_empty()}")
72     print(f"Stack size: {stack.size()}")
73     print()
74
75     # Push items onto the stack
76     print("Pushing items: 10, 20, 30")
77     stack.push(10)
78     stack.push(20)
79     stack.push(30)
80     print(f"Stack: {stack}")
81     print(f"Is stack empty? {stack.is_empty()}")
82     print(f"Stack size: {stack.size()}")
83     print()
84
85     # Peek at the top item
86     print(f"Peek at top: {stack.peak()}")
87     print(f"Stack after peek: {stack}")
88     print()
89
90     # Pop items from the stack
91     print("Popping items:")
92     print(f"Popped: {stack.pop()}")
93     print(f"Stack: {stack}")
94     print(f"Popped: {stack.pop()}")
95     print(f"Stack: {stack}")
96     print(f"Popped: {stack.pop()}")
97     print(f"Stack: {stack}")
98     print()
99
100     # Test is_empty after popping all items
101     print(f"Is stack empty? {stack.is_empty()}")
102     print(f"Stack size: {stack.size()}")

```

Output:

```

Is stack empty? True
Stack size: 0

Pushing items: 10, 20, 30
Stack: Stack([10, 20, 30])
Is stack empty? False
Stack size: 3

Peek at top: 30
Stack after peek: Stack([10, 20, 30])

Popping items:
Popped: 30
Stack: Stack([10, 20])
Popped: 20
Stack: Stack([10])
Popped: 10
Stack: Stack([])

Is stack empty? True
Stack size: 0
PS C:\2403A51L03\3-2\AI_A_C\Cursor AI>

```

Task Description #2 – Queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

class Queue:

Pass

Prompt:

Code:

```

1  class Queue:
2      def __init__(self):
3          """Initialize an empty queue."""
4          self.items = []
5
6      def enqueue(self, item):
7          """Add an item to the rear of the queue.
8
9          Args:
10             item: The item to add to the queue
11          """
12          self.items.append(item)
13
14      def dequeue(self):
15          """Remove and return the front item from the queue.
16
17          Returns:
18             The front item of the queue
19
20          Raises:
21             IndexError: If the queue is empty
22          """
23          if self.is_empty():
24              raise IndexError("Cannot dequeue from an empty queue")
25          return self.items.pop(0)
26
27      def peek(self):
28          """Return the front item of the queue without removing it.
29
30          Returns:
31             The front item of the queue
32
33          Raises:
34             IndexError: If the queue is empty
35          """
36          if self.is_empty():
37              raise IndexError("Cannot peek at an empty queue")
38          return self.items[0]
39
40      def size(self):
41          """Return the number of items in the queue.
42

```

```

43          Returns:
44             The number of items in the queue
45          """
46          return len(self.items)
47
48      def is_empty(self):
49          """Check if the queue is empty.
50
51          Returns:
52             True if the queue is empty, False otherwise
53          """
54          return len(self.items) == 0
55
56
57 # Example usage and testing
58 if __name__ == "__main__":
59     # Create a new queue
60     q = Queue()
61     # Click to chat, Ctrl+K to generate
62
63     # Test enqueue
64     print("Enqueueing items: 1, 2, 3, 4, 5")
65     q.enqueue(1)
66     q.enqueue(2)
67     q.enqueue(3)
68     q.enqueue(4)
69     q.enqueue(5)
70
71     # Test size
72     print(f"Queue size: {q.size()}")
73
74     # Test peek
75     print(f"Peek at front: {q.peek()}")
76
77     # Test dequeue
78     print("\nDequeueing items:")
79     while not q.is_empty():
80         print(f"Dequeued: {q.dequeue()}, Remaining size: {q.size()}")
81
82     # Test empty queue
83     print(f"\nQueue is empty: {q.is_empty()}")
84
85     # Test error handling
86     try:
87         q.dequeue()
88     except IndexError as e:
89         print(f"Error caught: {e}")
90
91     try:
92         q.peek()
93     except IndexError as e:
94         print(f"Error caught: {e}")

```

Output:

```
Enqueueing items: 1, 2, 3, 4, 5
Queue size: 5
Peek at front: 1

Dequeueing items:
Dequeued: 1, Remaining size: 4
Dequeued: 2, Remaining size: 3
Dequeued: 3, Remaining size: 2
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Enqueueing items: 1, 2, 3, 4, 5
Queue size: 5
Peek at front: 1

Dequeueing items:
Dequeued: 1, Remaining size: 4
Dequeued: 2, Remaining size: 3
Dequeued: 3, Remaining size: 2
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Peek at front: 1

Dequeueing items:
Dequeued: 1, Remaining size: 4
Dequeued: 2, Remaining size: 3
Dequeued: 3, Remaining size: 2
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Dequeueing items:
Dequeued: 1, Remaining size: 4
Dequeued: 2, Remaining size: 3
Dequeued: 3, Remaining size: 2
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Dequeueing items:
Dequeued: 1, Remaining size: 4
Dequeued: 2, Remaining size: 3
Dequeued: 3, Remaining size: 2
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Dequeueing items:
Dequeued: 2, Remaining size: 3
Dequeued: 3, Remaining size: 2
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Dequeueing items:
Dequeued: 4, Remaining size: 1
Dequeued: 5, Remaining size: 0

Dequeueing items:
Dequeued: 5, Remaining size: 0

Queue is empty: True
Error caught: Cannot dequeue from an empty queue
Error caught: Cannot peek at an empty queue
```

Task Description #3 – Linked List

Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

class Node:

Pass

Prompt:

generate a Singly Linked List with insert and display methods with code

Code:


```

1 class Node:
2     """Node class to represent a single node in the linked list"""
3     def __init__(self, data):
4         self.data = data # Data stored in the node
5         self.next = None # Reference to the next node
6
7
8 class SinglyLinkedList:
9     """Singly Linked List implementation with insert and display methods"""
10
11     def __init__(self):
12         self.head = None # Head pointer pointing to the first node
13
14     def insert(self, data):
15         """
16         Insert a new node at the end of the linked list
17
18         Args:
19             data: The data to be inserted into the linked list
20         """
21         new_node = Node(data)
22
23         # If the list is empty, make the new node the head
24         if self.head is None:
25             self.head = new_node
26         else:
27             # Traverse to the end of the list
28             current = self.head
29             while current.next is not None:
30                 current = current.next
31             # Insert the new node at the end
32             current.next = new_node
33
34     def insert_at_beginning(self, data):
35         """
36         Insert a new node at the beginning of the linked list
37
38         Args:
39             data: The data to be inserted into the linked list
40         """
41         new_node = Node(data)
42         new_node.next = self.head
43         self.head = new_node
44
45     def display(self):
46         """
47         Display all elements in the linked list
48         """
49         if self.head is None:
50             print("Linked list is empty")
51             return
52
53         current = self.head
54
55         elements = []
56         while current is not None:
57             elements.append(str(current.data))
58             current = current.next
59
60         # Display in format: data1 -> data2 -> data3 -> None
61         print(" -> ".join(elements) + " -> None")
62
63 # Example usage
64 if __name__ == "__main__":
65     # Create a new linked list
66     ll = SinglyLinkedList()
67
68     # Insert some elements
69     print("Inserting elements into the linked list...")
70     ll.insert(10)
71     ll.insert(20)
72     ll.insert(30)
73     ll.insert(40)
74
75     # Display the linked list
76     print("\nLinked List contents:")
77     ll.display()
78
79     # Insert at beginning
80     print("\nInserting 5 at the beginning...")
81     ll.insert_at_beginning(5)
82     ll.display()
83
84     # Create an empty list
85     print("\nCreating an empty linked list:")
86     empty_ll = SinglyLinkedList()
87     empty_ll.display()

```

Output:

```
Inserting elements into the linked list...
Inserting elements into the linked list...
```

```
Linked List contents:
```

```
Linked List contents:
```

```
Linked List contents:
```

```
10 -> 20 -> 30 -> 40 -> None
```

```
10 -> 20 -> 30 -> 40 -> None
```

```
Inserting 5 at the beginning...
```

```
Inserting 5 at the beginning...
```

```
5 -> 10 -> 20 -> 30 -> 40 -> None
```

```
5 -> 10 -> 20 -> 30 -> 40 -> None
```

```
Creating an empty linked list:
```

```
Linked List is empty
```

```
Linked List is empty
```

Task Description #4 – Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

Sample Input Code:

```
class BST:
    pass
```

Prompt:

create a BST with insert and in-order traversal methods and code

Code:


```

1 class Node:
2     def __init__(self, key: int):
3         self.key = key
4         self.left: "Node | None" = None
5         self.right: "Node | None" = None
6
7
8 class BST:
9     def __init__(self):
10         self.root: Node | None = None
11
12     def insert(self, key: int) -> None:
13         """Insert key into the BST (duplicates go to the right)."""
14         if self.root is None:
15             self.root = Node(key)
16             return
17
18         cur = self.root
19         while True:
20             if key < cur.key:
21                 if cur.left is None:
22                     cur.left = Node(key)
23                     return
24                 cur = cur.left
25             else:
26                 if cur.right is None:
27                     cur.right = Node(key)
28                     return
29                 cur = cur.right
30
31     def inorder(self) -> list[int]:
32         """Return keys in in-order (sorted) order."""
33         result: list[int] = []
34
35         def dfs(n: Node | None) -> None:
36             if n is None:
37                 return
38             dfs(n.left)
39             result.append(n.key)
40             dfs(n.right)
41
42         dfs(self.root)
43         return result
44
45
46 if __name__ == "__main__":
47     bst = BST()
48     for x in [7, 3, 9, 1, 5, 8, 10]:
49         bst.insert(x)
50     print("In-order:", bst.inorder())

```

Output:

```
In-order: [1, 3, 5, 7, 8, 9, 10]
```

Task Description #5 – Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code:

```
class HashTable:
```

```
    pass
```

Prompt:

implement a hash table with basic insert, search, and delete methods with code

Code:

```

1 class HashTable:
2     """
3     Hash table using separate chaining (list of buckets).
4
5     Methods:
6     - insert(key, value): add/update a key
7     - search(key): return value or None if not found
8     - delete(key): remove key, return True if removed else False
9     """
10
11     def __init__(self, capacity: int = 8) -> None:
12         if capacity < 1:
13             raise ValueError("capacity must be >= 1")
14         self._capacity = capacity
15         self._buckets = [[] for _ in range(self._capacity)] # List[List[tuple[key, value]]]
16         self._size = 0
17
18     def _index(self, key) -> int:
19         return hash(key) % self._capacity
20
21     def _rehash(self, new_capacity: int) -> None:
22         old_items = []
23         for bucket in self._buckets:
24             old_items.extend(bucket)
25
26         self._capacity = new_capacity
27         self._buckets = [[] for _ in range(self._capacity)]
28         self._size = 0
29
30         for k, v in old_items:
31             self.insert(k, v)
32
33     def insert(self, key, value) -> None:
34         # Resize when load factor gets too high (simple rule-of-thumb)
35         if (self._size + 1) / self._capacity > 0.75:
36             self._rehash(self._capacity * 2)
37
38         idx = self._index(key)
39         bucket = self._buckets[idx]
40
41         for i, (k, _) in enumerate[Any](bucket):
42             if k == key:
43                 bucket[i] = (key, value) # update existing
44                 return
45
46         bucket.append((key, value))
47         self._size += 1
48
49     def search(self, key):
50         idx = self._index(key)
51         bucket = self._buckets[idx]
52         for k, v in bucket:
53             if k == key:

```

```

54         return v
55     return None
56
57     def delete(self, key) -> bool:
58         idx = self._index(key)
59         bucket = self._buckets[idx]
60
61         for i, (k, _) in enumerate[Any](bucket):
62             if k == key:
63                 bucket.pop(i)
64                 self._size -= 1
65                 return True
66
67         return False
68
69     def __len__(self) -> int:
70         return self._size
71
72     def __contains__(self, key) -> bool:
73         return self.search(key) is not None
74
75     def __repr__(self) -> str:
76         return f"HashTable(size={self._size}, capacity={self._capacity})"
77
78
79 if __name__ == "__main__":
80     ht = HashTable()
81     ht.insert("name", "Alice")
82     ht.insert("age", 20)
83     ht.insert("age", 21) # update
84
85     print(ht) # HashTable(...)
86     print(ht.search("name")) # Alice
87     print(ht.search("age")) # 21
88     print(ht.search("x")) # None
89
90     print(ht.delete("age")) # True
91     print(ht.delete("age")) # False
92     print(len(ht)) # 1

```

Output:

```

HashTable(size=2, capacity=8)
Alice
21
None
HashTable(size=2, capacity=8)
Alice
21
None
21
None
True
False
1
True
False
1
False
1

```

Task Description #6 – Graph Representation

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

```
class Graph:  
    pass
```

Prompt:

implement a graph using an adjacency list with code

Code:

```

1 class Graph:
2     """
3     Graph implemented using an adjacency list.
4
5     - By default the graph is undirected.
6     - Set directed=True for a directed graph.
7     """
8
9     def __init__(self, directed: bool = False):
10         self.directed = directed
11         # adjacency list: vertex -> set of neighbor vertices
12         self.adj: dict[object, set[object]] = {}
13
14     def add_vertex(self, v: object) -> None:
15         """Add a vertex if it doesn't already exist."""
16         if v not in self.adj:
17             self.adj[v] = set[object]()
18
19     def add_edge(self, u: object, v: object) -> None:
20         """Add an edge u -> v (and v -> u if undirected)."""
21         self.add_vertex(u)
22         self.add_vertex(v)
23         self.adj[u].add(v)
24         if not self.directed:
25             self.adj[v].add(u)
26
27     def remove_edge(self, u: object, v: object) -> None:
28         """Remove an edge u -> v (and v -> u if undirected), if present."""
29         if u in self.adj:
30             self.adj[u].discard(v)
31         if not self.directed and v in self.adj:
32             self.adj[v].discard(u)
33
34     def remove_vertex(self, v: object) -> None:
35         """Remove a vertex and all edges incident to it."""
36         if v not in self.adj:
37             return
38
39         # Remove edges from neighbors to v
40         for n in list(self.adj[v]):
41             self.remove_edge(v, n)
42
43         # In directed graphs, also remove incoming edges to v
44         if self.directed:
45             for u in self.adj:
46                 self.adj[u].discard(v)
47
48         del self.adj[v]
49
50     def neighbors(self, v: object) -> list[object]:
51         """Return neighbors of v as a sorted list when possible."""
52         if v not in self.adj:
53             return []
54         try:
55             return sorted(self.adj[v])
56         except TypeError:
57             return list(self.adj[v])
58
59     def bfs(self, start: object) -> list[object]:
60         """Breadth-first traversal order starting from start."""
61         if start not in self.adj:
62             return []
63
64         visited = {start}
65         queue = [start]
66         order: list[object] = []
67
68         while queue:
69             v = queue.pop(0)
70             order.append(v)
71             for n in self.neighbors(v):
72                 if n not in visited:
73                     visited.add(n)
74                     queue.append(n)
75
76         return order
77
78     def dfs(self, start: object) -> list[object]:
79         """Depth-first traversal order starting from start."""
80         if start not in self.adj:
81             return []
82
83         visited: set[object] = set[object]()
84         order: list[object] = []
85
86         def _visit(v: object) -> None:
87             visited.add(v)
88             order.append(v)
89             for n in self.neighbors(v):
90                 if n not in visited:
91                     _visit(n)
92
93         _visit(start)
94         return order
95
96     def __str__(self) -> str:
97         lines = []
98         for v in self.adj:
99             lines.append(f"{v} -> {self.neighbors(v)}")
100         return "\n".join(lines)
101
102 if __name__ == "__main__":
103     g = Graph(directed=False) # change to True for a directed graph
104     g.add_edge("A", "B")
105     g.add_edge("A", "C")
106     g.add_edge("B", "D")
107     g.add_edge("C", "D")
108     g.add_edge("D", "E")
109
110     print("Adjacency list:")
111     print(g)
112     print()
113     print("BFS from A:", g.bfs("A"))
114     print("DFS from A:", g.dfs("A"))

```

Output:

```
Adjacency list:
A -> ['B', 'C']
B -> ['A', 'D']
Adjacency list:
A -> ['B', 'C']
B -> ['A', 'D']
A -> ['B', 'C']
B -> ['A', 'D']
B -> ['A', 'D']
C -> ['A', 'D']
D -> ['B', 'C', 'E']
E -> ['D']

BFS from A: ['A', 'B', 'C', 'D', 'E']
DFS from A: ['A', 'B', 'D', 'C', 'E']
D -> ['B', 'C', 'E']
E -> ['D']

BFS from A: ['A', 'B', 'C', 'D', 'E']
DFS from A: ['A', 'B', 'D', 'C', 'E']
E -> ['D']

BFS from A: ['A', 'B', 'C', 'D', 'E']
DFS from A: ['A', 'B', 'D', 'C', 'E']
BFS from A: ['A', 'B', 'C', 'D', 'E']
DFS from A: ['A', 'B', 'D', 'C', 'E']
DFS from A: ['A', 'B', 'D', 'C', 'E']
```

Task Description #7 – Priority Queue

Task: Use AI to implement a priority queue using Python's heapq module.

Sample Input Code:

```
class PriorityQueue:
    pass
```

Prompt:

implement a priority queue using Python's heapq module with code

Code:

```

1 import heapq
2 from itertools import count
3
4 class PriorityQueue:
5     """
6     Min-priority queue by default (smaller priority value = served first).
7     For max-priority behavior, push with -priority.
8     """
9     def __init__(self):
10         self._heap = []
11         self._seq = count(int)() # tie-breaker for equal priorities (FIFO)
12
13     def push(self, item, priority: int):
14         heapq.heappush(self._heap, (priority, next(self._seq), item))
15
16     def pop(self):
17         if not self._heap:
18             raise IndexError("pop from empty PriorityQueue")
19         priority, _, item = heapq.heappop(self._heap)
20         return item, priority
21
22     def peek(self):
23         if not self._heap:
24             raise IndexError("peek from empty PriorityQueue")
25         priority, _, item = self._heap[0]
26         return item, priority
27
28     def __len__(self):
29         return len(self._heap)
30
31     def empty(self):
32         return len(self._heap) == 0
33
34
35 if __name__ == "__main__":
36     pq = PriorityQueue()
37     pq.push("low", 5)
38     pq.push("urgent", 1)
39     pq.push("medium", 3)
40     pq.push("also urgent (arrives later)", 1)
41
42     while not pq.empty():
43         item, pr = pq.pop()
44         print(pr, item)
45
46     # Max-priority example (bigger number = served first):
47     maxpq = PriorityQueue()
48     for item, pr in [("A", 10), ("B", 2), ("C", 10)]:
49         maxpq.push(item, -pr) # negate priority
50
51     print("max first:", maxpq.pop()) # returns (item, neg_priority)

```

Output:

```

1 urgent
1 also urgent (arrives later)
3 medium
5 low
max first: ('A', -10)

```

Task Description #8 – Deque

Task: Use AI to implement a double-ended queue using `collections.deque`.

Sample Input Code:

```

class DequeDS:
    pass

```

Prompt:

implement a double-ended queue using `collections.deque` with code

Code:

```

1  from __future__ import annotations
2
3  from collections import deque
4  from typing import Deque, Generic, Iterator, Optional, TypeVar
5
6  T = TypeVar("T")
7
8
9  class DequeDS(Generic[T]):
10     """
11     Double-ended queue (deque) implemented using collections.deque.
12     Supports O(1) append/pop operations on both ends.
13     """
14
15     def __init__(self, items: Optional[Iterator[T]] = None) -> None:
16         self._dq: Deque[T] = deque([i for i in items or []])
17
18     # --- Add operations ---
19     def add_front(self, item: T) -> None:
20         """Insert item at the front (left)."""
21         self._dq.appendleft(item)
22
23     def add_rear(self, item: T) -> None:
24         """Insert item at the rear (right)."""
25         self._dq.append(item)
26
27     # --- Remove operations ---
28     def remove_front(self) -> T:
29         """Remove and return the front (left) item."""
30         if self.is_empty():
31             raise IndexError("remove_front from empty deque")
32         return self._dq.popleft()
33
34     def remove_rear(self) -> T:
35         """Remove and return the rear (right) item."""
36         if self.is_empty():
37             raise IndexError("remove_rear from empty deque")
38         return self._dq.pop()
39
40     # --- Peek operations ---
41     def peek_front(self) -> T:
42         """Return the front (left) item without removing it."""
43         if self.is_empty():
44             raise IndexError("peek_front from empty deque")
45         return self._dq[0]
46
47     def peek_rear(self) -> T:
48         """Return the rear (right) item without removing it."""
49         if self.is_empty():
50             raise IndexError("peek_rear from empty deque")
51         return self._dq[-1]

```



```

52
53     # --- Utility ---
54     def is_empty(self) -> bool:
55         return len(self._dq) == 0
56
57     def size(self) -> int:
58         return len(self._dq)
59
60     def clear(self) -> None:
61         self._dq.clear()
62
63     def __len__(self) -> int:
64         return len(self._dq)
65
66     def __iter__(self) -> Iterator[T]:
67         return iter(self._dq)
68
69     def __repr__(self) -> str:
70         return f"DequeDS({list[T](self._dq)!r})"
71
72
73 if __name__ == "__main__":
74     d = DequeDS[int]()
75     d.add_front(10)    # [10]
76     d.add_rear(20)     # [10, 20]
77     d.add_front(5)     # [5, 10, 20]
78     print("Deque:", d)
79     print("Front:", d.peak_front())
80     print("Rear:", d.peak_rear())
81     print("Remove front:", d.remove_front()) # 5
82     print("Remove rear:", d.remove_rear())  # 20
83     print("Deque now:", d)

```

Output:

```

Deque: DequeDS([5, 10, 20])
Front: 5
Rear: 20
Remove front: 5
Remove rear: 20
Deque now: DequeDS([10])

```

Task Description #9 Real-Time Application Challenge – Choose the Right Data Structure

Prompt:

Solve this clearly and concisely.

Design a Campus Resource Management System code with:

Student Attendance Tracking

Event Registration System

Library Book Borrowing

Bus Scheduling System

Cafeteria Order Queue

Choose the best data structure for each feature from:

Stack, Queue, Priority Queue, Linked List, BST, Graph, Hash Table, Deque

Output as a table:

Feature | Data Structure | 2–3 sentence justification

Code:

```

1  from __future__ import annotations
2
3  from dataclasses import dataclass
4  from collections import deque
5  import heapq
6  import itertools
7  from typing import Deque, Dict, List, Optional, Set, Tuple
8
9
10 # =====
11 # 1) Student Attendance Tracking (Hash Table)
12 # =====
13
14 class AttendanceTracker:
15     """
16     Data structure: Hash Table (Python dict)
17     __student_id__ -> {date: str -> present: bool}
18     """
19
20     def __init__(self) -> None:
21         self._records: Dict[str, Dict[str, bool]] = {}
22
23     def mark(self, student_id: str, date: str, present: bool) -> None:
24         self._records.setdefault(student_id, {})
25         self._records[student_id][date] = present
26
27     def is_present(self, student_id: str, date: str) -> Optional[bool]:
28         return self._records.get(student_id, {}).get(date)
29
30     def attendance_percent(self, student_id: str) -> float:
31         days = set(self._records.get(student_id, {}))
32         if not days:
33             return 0.0
34         present_count = sum(1 for v in days.values() if v)
35         return (present_count / len(days)) * 100.0
36
37 # =====
38 # 2) Event Registration System (Deque)
39 # =====
40
41 class EventRegistrationSystem:
42     """
43     Data structure: Queue (collections.deque)
44     FIFO registration requests + FIFO dequeue.
45     """
46
47     @dataclass(frozen=True)
48     class Event:
49         event_id: str
50         name: str
51         capacity: int
52
53     def __init__(self) -> None:
54         self._events: Dict[str, EventRegistrationSystem.Event] = {}
55         self._confirmed: Dict[str, Set[str]] = {}
56         self._requests: Dict[str, Deque[str]] = {}
57         self._waitlist: Dict[str, Deque[str]] = {}
58         self._event_id -> queue(self._student_id)
59
60     def create_event(self, event_id: str, name: str, capacity: int) -> None:
61         if capacity <= 0:
62             raise ValueError("Capacity must be > 0")
63         self._events[event_id] = self.Event(event_id, name, capacity)
64         self._confirmed.setdefault(event_id, set())
65         self._requests.setdefault(event_id, deque())
66         self._waitlist.setdefault(event_id, deque())
67
68     def request_registration(self, event_id: str, student_id: str) -> None:
69         self._confirm(event_id)
70         if student_id in self._confirmed[event_id]:
71             return
72         if student_id in self._requests[event_id] or student_id in self._waitlist[event_id]:
73             return
74         self._requests[event_id].append(student_id)
75
76     def process_next_request(self, event_id: str) -> Optional[str]:
77         """
78         Processes ONE pending request in FIFO order.
79         Returns the student_id that got confirmed (or None if no request).
80         """
81         self._process_event(event_id)
82         q = self._requests[event_id]
83         if not q:
84             return None
85         student_id = q.popleft()
86         if len(self._confirmed[event_id]) < self._events[event_id].capacity:
87             self._confirmed[event_id].add(student_id)
88             return student_id
89         self._waitlist[event_id].append(student_id)
90         return None
91
92     def cancel_registration(self, event_id: str, student_id: str) -> None:
93         self._process_event(event_id)
94         if student_id in self._confirmed[event_id]:
95             self._confirmed[event_id].remove(student_id)
96             self._process_from_waitlist(event_id)
97             return

```

```

100     return
101     self.remove_from_queue(self._requests[event_id], student_id)
102     self.remove_from_queue(self._waitlist[event_id], student_id)
103
104     def _confirm_seat(self, event_id: str) -> list[str]:
105         self._confirm_event(event_id)
106         return sorted(self._confirmed[event_id])
107
108     def _waitlist_list(self, event_id: str) -> list[str]:
109         self._waitlist_event(event_id)
110         return list(self._waitlist[event_id])
111
112     def _remove_from_waitlist(self, event_id: str) -> None:
113         if len(self._confirmed[event_id]) > self._events[event_id].capacity:
114             return
115         self._waitlist_event(event_id)
116         while self and len(self._confirmed[event_id]) < self._events[event_id].capacity:
117             self._confirmed[event_id].add(self._pop_left())
118
119     def _remove_from_queue(self, q: Deque[str], student_id: str) -> None:
120         # Since deque doesn't support fast middle removals, results for clarity.
121         if not q:
122             return
123         new_q = deque()
124         for x in q:
125             if x != student_id:
126                 new_q.append(x)
127         q.clear()
128         q.extend(new_q)
129
130     def _enqueue_event(self, event_id: str) -> None:
131         if event_id not in self._events:
132             raise ValueError(f"Unknown event_id: {event_id}")
133
134     # =====
135     # ID library book borrowing (BIB)
136     # =====
137
138     class Book:
139         isbn: str
140         title: str
141         total_copies: int
142         available_copies: int
143
144     class BookBorrower:
145         def __init__(self, user_id: str, book: Book) -> None:
146             self.user = user_id
147             self.book = book
148             self.left_optional_book_borrower = None
149             self.right_optional_book_borrower = None
150
151     class librarysystem:
152         """
153         Data structure: BIB (by ISBN) for catalog/inventory search and ordered traversal.
154         - Borrower indexes available copies (returning increments).
155         """
156
157         def __init__(self) -> None:
158             self._books: Optional[Dict[str, Book]] = None
159             self._users: Dict[str, list[int]] = {} # {student_id, isbn} -> count borrowed
160
161         def add_book(self, isbn: str, title: str, copies: int = 1) -> None:
162             if copies <= 0:
163                 raise ValueError("copies must be > 0")
164
165             existing = self._find(isbn)
166             if existing:
167                 existing.total_copies += copies
168                 existing.available_copies += copies
169                 return
170
171             book = Book(isbn=isbn, title=title, total_copies=copies, available_copies=copies)
172             self._books = self._insert_or_create(isbn, title, book)
173
174         def find(self, isbn: str) -> Optional[Book]:
175             book = self._find(isbn)
176             if book is None:
177                 return None
178             if book is None:
179                 return None
180             if book is None:
181                 return None
182             if book is None:
183                 return None
184             if book is None:
185                 return None
186             if book is None:
187                 return None
188             if book is None:
189                 return None
190             if book is None:
191                 return None
192             if book is None:
193                 return None
194             if book is None:
195                 return None
196             if book is None:
197                 return None
198             if book is None:
199                 return None
200             if book is None:
201                 return None
202             if book is None:
203                 return None
204             if book is None:
205                 return None
206             if book is None:
207                 return None
208             if book is None:
209                 return None
210             if book is None:
211                 return None
212             if book is None:
213                 return None
214             if book is None:
215                 return None
216             if book is None:
217                 return None
218             if book is None:
219                 return None
220             if book is None:
221                 return None
222             if book is None:
223                 return None
224             if book is None:
225                 return None
226             if book is None:
227                 return None
228             if book is None:
229                 return None
230             if book is None:
231                 return None
232             if book is None:
233                 return None
234             if book is None:
235                 return None
236             if book is None:
237                 return None
238             if book is None:
239                 return None
240             if book is None:
241                 return None
242             if book is None:
243                 return None
244             if book is None:
245                 return None
246             if book is None:
247                 return None
248             if book is None:
249                 return None
250             if book is None:
251                 return None
252             if book is None:
253                 return None
254             if book is None:
255                 return None
256             if book is None:
257                 return None
258             if book is None:
259                 return None
260             if book is None:
261                 return None
262             if book is None:
263                 return None
264             if book is None:
265                 return None
266             if book is None:
267                 return None
268             if book is None:
269                 return None
270             if book is None:
271                 return None
272             if book is None:
273                 return None
274             if book is None:
275                 return None
276             if book is None:
277                 return None
278             if book is None:
279                 return None
280             if book is None:
281                 return None
282             if book is None:
283                 return None
284             if book is None:
285                 return None
286             if book is None:
287                 return None
288             if book is None:
289                 return None
290             if book is None:
291                 return None
292             if book is None:
293                 return None
294             if book is None:
295                 return None
296             if book is None:
297                 return None
298             if book is None:
299                 return None
300             if book is None:
301                 return None
302             if book is None:
303                 return None
304             if book is None:
305                 return None
306             if book is None:
307                 return None
308             if book is None:
309                 return None
310             if book is None:
311                 return None
312             if book is None:
313                 return None
314             if book is None:
315                 return None
316             if book is None:
317                 return None
318             if book is None:
319                 return None
320             if book is None:
321                 return None
322             if book is None:
323                 return None
324             if book is None:
325                 return None
326             if book is None:
327                 return None
328             if book is None:
329                 return None
330             if book is None:
331                 return None
332             if book is None:
333                 return None
334             if book is None:
335                 return None
336             if book is None:
337                 return None
338             if book is None:
339                 return None
340             if book is None:
341                 return None
342             if book is None:
343                 return None
344             if book is None:
345                 return None
346             if book is None:
347                 return None
348             if book is None:
349                 return None
350             if book is None:
351                 return None
352             if book is None:
353                 return None
354             if book is None:
355                 return None
356             if book is None:
357                 return None
358             if book is None:
359                 return None
360             if book is None:
361                 return None
362             if book is None:
363                 return None
364             if book is None:
365                 return None
366             if book is None:
367                 return None
368             if book is None:
369                 return None
370             if book is None:
371                 return None
372             if book is None:
373                 return None
374             if book is None:
375                 return None
376             if book is None:
377                 return None
378             if book is None:
379                 return None
380             if book is None:
381                 return None
382             if book is None:
383                 return None
384             if book is None:
385                 return None
386             if book is None:
387                 return None
388             if book is None:
389                 return None
390             if book is None:
391                 return None
392             if book is None:
393                 return None
394             if book is None:
395                 return None
396             if book is None:
397                 return None
398             if book is None:
399                 return None
400             if book is None:
401                 return None
402             if book is None:
403                 return None
404             if book is None:
405                 return None
406             if book is None:
407                 return None
408             if book is None:
409                 return None
410             if book is None:
411                 return None
412             if book is None:
413                 return None
414             if book is None:
415                 return None
416             if book is None:
417                 return None
418             if book is None:
419                 return None
420             if book is None:
421                 return None
422             if book is None:
423                 return None
424             if book is None:
425                 return None
426             if book is None:
427                 return None
428             if book is None:
429                 return None
430             if book is None:
431                 return None
432             if book is None:
433                 return None
434             if book is None:
435                 return None
436             if book is None:
437                 return None
438             if book is None:
439                 return None
440             if book is None:
441                 return None
442             if book is None:
443                 return None
444             if book is None:
445                 return None
446             if book is None:
447                 return None
448             if book is None:
449                 return None
450             if book is None:
451                 return None
452             if book is None:
453                 return None
454             if book is None:
455                 return None
456             if book is None:
457                 return None
458             if book is None:
459                 return None
460             if book is None:
461                 return None
462             if book is None:
463                 return None
464             if book is None:
465                 return None
466             if book is None:
467                 return None
468             if book is None:
469                 return None
470             if book is None:
471                 return None
472             if book is None:
473                 return None
474             if book is None:
475                 return None
476             if book is None:
477                 return None
478             if book is None:
479                 return None
480             if book is None:
481                 return None
482             if book is None:
483                 return None
484             if book is None:
485                 return None
486             if book is None:
487                 return None
488             if book is None:
489                 return None
490             if book is None:
491                 return None
492             if book is None:
493                 return None
494             if book is None:
495                 return None
496             if book is None:
497                 return None
498             if book is None:
499                 return None
500             if book is None:
501                 return None
502             if book is None:
503                 return None
504             if book is None:
505                 return None
506             if book is None:
507                 return None
508             if book is None:
509                 return None
510             if book is None:
511                 return None
512             if book is None:
513                 return None
514             if book is None:
515                 return None
516             if book is None:
517                 return None
518             if book is None:
519                 return None
520             if book is None:
521                 return None
522             if book is None:
523                 return None
524             if book is None:
525                 return None
526             if book is None:
527                 return None
528             if book is None:
529                 return None
530             if book is None:
531                 return None
532             if book is None:
533                 return None
534             if book is None:
535                 return None
536             if book is None:
537                 return None
538             if book is None:
539                 return None
540             if book is None:
541                 return None
542             if book is None:
543                 return None
544             if book is None:
545                 return None
546             if book is None:
547                 return None
548             if book is None:
549                 return None
550             if book is None:
551                 return None
552             if book is None:
553                 return None
554             if book is None:
555                 return None
556             if book is None:
557                 return None
558             if book is None:
559                 return None
560             if book is None:
561                 return None
562             if book is None:
563                 return None
564             if book is None:
565                 return None
566             if book is None:
567                 return None
568             if book is None:
569                 return None
570             if book is None:
571                 return None
572             if book is None:
573                 return None
574             if book is None:
575                 return None
576             if book is None:
577                 return None
578             if book is None:
579                 return None
580             if book is None:
581                 return None
582             if book is None:
583                 return None
584             if book is None:
585                 return None
586             if book is None:
587                 return None
588             if book is None:
589                 return None
590             if book is None:
591                 return None
592             if book is None:
593                 return None
594             if book is None:
595                 return None
596             if book is None:
597                 return None
598             if book is None:
599                 return None
600             if book is None:
601                 return None
602             if book is None:
603                 return None
604             if book is None:
605                 return None
606             if book is None:
607                 return None
608             if book is None:
609                 return None
610             if book is None:
611                 return None
612             if book is None:
613                 return None
614             if book is None:
615                 return None
616             if book is None:
617                 return None
618             if book is None:
619                 return None
620             if book is None:
621                 return None
622             if book is None:
623                 return None
624             if book is None:
625                 return None
626             if book is None:
627                 return None
628             if book is None:
629                 return None
630             if book is None:
631                 return None
632             if book is None:
633                 return None
634             if book is None:
635                 return None
636             if book is None:
637                 return None
638             if book is None:
639                 return None
640             if book is None:
641                 return None
642             if book is None:
643                 return None
644             if book is None:
645                 return None
646             if book is None:
647                 return None
648             if book is None:
649                 return None
650             if book is None:
651                 return None
652             if book is None:
653                 return None
654             if book is None:
655                 return None
656             if book is None:
657                 return None
658             if book is None:
659                 return None
660             if book is None:
661                 return None
662             if book is None:
663                 return None
664             if book is None:
665                 return None
666             if book is None:
667                 return None
668             if book is None:
669                 return None
670             if book is None:
671                 return None
672             if book is None:
673                 return None
674             if book is None:
675                 return None
676             if book is None:
677                 return None
678             if book is None:
679                 return None
680             if book is None:
681                 return None
682             if book is None:
683                 return None
684             if book is None:
685                 return None
686             if book is None:
687                 return None
688             if book is None:
689                 return None
690             if book is None:
691                 return None
692             if book is None:
693                 return None
694             if book is None:
695                 return None
696             if book is None:
697                 return None
698             if book is None:
699                 return None
700             if book is None:
701                 return None
702             if book is None:
703                 return None
704             if book is None:
705                 return None
706             if book is None:
707                 return None
708             if book is None:
709                 return None
710             if book is None:
711                 return None
712             if book is None:
713                 return None
714             if book is None:
715                 return None
716             if book is None:
717                 return None
718             if book is None:
719                 return None
720             if book is None:
721                 return None
722             if book is None:
723                 return None
724             if book is None:
725                 return None
726             if book is None:
727                 return None
728             if book is None:
729                 return None
730             if book is None:
731                 return None
732             if book is None:
733                 return None
734             if book is None:
735                 return None
736             if book is None:
737                 return None
738             if book is None:
739                 return None
740             if book is None:
741                 return None
742             if book is None:
743                 return None
744             if book is None:
745                 return None
746             if book is None:
747                 return None
748             if book is None:
749                 return None
750             if book is None:
751                 return None
752             if book is None:
753                 return None
754             if book is None:
755                 return None
756             if book is None:
757                 return None
758             if book is None:
759                 return None
760             if book is None:
761                 return None
762             if book is None:
763                 return None
764             if book is None:
765                 return None
766             if book is None:
767                 return None
768             if book is None:
769                 return None
770             if book is None:
771                 return None
772             if book is None:
773                 return None
774             if book is None:
775                 return None
776             if book is None:
777                 return None
778             if book is None:
779                 return None
780             if book is None:
781                 return None
782             if book is None:
783                 return None
784             if book is None:
785                 return None
786             if book is None:
787                 return None
788             if book is None:
789                 return None
790             if book is None:
791                 return None
792             if book is None:
793                 return None
794             if book is None:
795                 return None
796             if book is None:
797                 return None
798             if book is None:
799                 return None
800             if book is None:
801                 return None
802             if book is None:
803                 return None
804             if book is None:
805                 return None
806             if book is None:
807                 return None
808             if book is None:
809                 return None
810             if book is None:
811                 return None
812             if book is None:
813                 return None
814             if book is None:
815                 return None
816             if book is None:
817                 return None
818             if book is None:
819                 return None
820             if book is None:
821                 return None
822             if book is None:
823                 return None
824             if book is None:
825                 return None
826             if book is None:
827                 return None
828             if book is None:
829                 return None
830             if book is None:
831                 return None
832             if book is None:
833                 return None
834             if book is None:
835                 return None
836             if book is None:
837                 return None
838             if book is None:
839                 return None
840             if book is None:
841                 return None
842             if book is None:
843                 return None
844             if book is None:
845                 return None
846             if book is None:
847                 return None
848             if book is None:
849                 return None
850             if book is None:
851                 return None
852             if book is None:
853                 return None
854             if book is None:
855                 return None
856             if book is None:
857                 return None
858             if book is None:
859                 return None
860             if book is None:
861                 return None
862             if book is None:
863                 return None
864             if book is None:
865                 return None
866             if book is None:
867                 return None
868             if book is None:
869                 return None
870             if book is None:
871                 return None
872             if book is None:
873                 return None
874             if book is None:
875                 return None
876             if book is None:
877                 return None
878             if book is None:
879                 return None
880             if book is None:
881                 return None
882             if book is None:
883                 return None
884             if book is None:
885                 return None
886             if book is None:
887                 return None
888             if book is None:
889                 return None
890             if book is None:
891                 return None
892             if book is None:
893                 return None
894             if book is None:
895                 return None
896             if book is None:
897                 return None
898             if book is None:
899                 return None
900             if book is None:
901                 return None
902             if book is None:
903                 return None
904             if book is None:
905                 return None
906             if book is None:
907                 return None
908             if book is None:
909                 return None
910             if book is None:
911                 return None
912             if book is None:
913                 return None
914             if book is None:
915                 return None
916             if book is None:
917                 return None
918             if book is None:
919                 return None
920             if book is None:
921                 return None
922             if book is None:
923                 return None
924             if book is None:
925                 return None
926             if book is None:
927                 return None
928             if book is None:
929                 return None
930             if book is None:
931                 return None
932             if book is None:
933                 return None
934             if book is None:
935                 return None
936             if book is None:
937                 return None
938             if book is None:
939                 return None
940             if book is None:
941                 return None
942             if book is None:
943                 return None
944             if book is None:
945                 return None
946             if book is None:
947                 return None
948             if book is None:
949                 return None
950             if book is None:
951                 return None
952             if book is None:
953                 return None
954             if book is None:
955                 return None
956             if book is None:
957                 return None
958             if book is None:
959                 return None
960             if book is None:
961                 return None
962             if book is None:
963                 return None
964             if book is None:
965                 return None
966             if book is None:
967                 return None
968             if book is None:
969                 return None
970             if book is None:
971                 return None
972             if book is None:
973                 return None
974             if book is None:
975                 return None
976             if book is None:
977                 return None
978             if book is None:
979                 return None
980             if book is None:
981                 return None
982             if book is None:
983                 return None
984             if book is None:
985                 return None
986             if book is None:
987                 return None
988             if book is None:
989                 return None
990             if book is None:
991                 return None
992             if book is None:
993                 return None
994             if book is None:
995                 return None
996             if book is None:
997                 return None
998             if book is None:
999                 return None
1000            if book is None:

```

```

1198 book = self.find(book)
1199 if not book:
1200     return False
1201 self._last_borrow += 1
1202 book.available_copies -= 1
1203 return True
1204
1205 def catalog_in_order(self) -> List[Book]:
1206     ret: List[Book] = []
1207     self._in_order(self._root, ret)
1208     return ret
1209
1210 def insert(self, node: Optional[BookNode], label: str, book: Book) -> BookNode:
1211     if node is None:
1212         return BookNode(label, book)
1213     if label < node.label:
1214         node.left = self.insert(node.left, label, book)
1215     else:
1216         node.right = self.insert(node.right, label, book)
1217     return node
1218
1219 def _in_order(self, node: Optional[BookNode], out: List[Book]) -> None:
1220     if node is None:
1221         return
1222     self._in_order(node.left, out)
1223     out.append(node.book)
1224     self._in_order(node.right, out)
1225
1226 # =====
1227 # A) Bus Scheduling System (Graph)
1228 # =====
1229
1230 class BusNetwork:
1231     """
1232     Data structure: Graph (adjacency list)
1233     - stop -> list of (neighbor_stop, travel_minutes)
1234     - shortest_path uses Dijkstra (non-negative weights).
1235     """
1236
1237     def __init__(self) -> None:
1238         self._adj: Dict[str, List[Tuple[str, int]]] = {}
1239
1240     def add_stop(self, stop: str) -> None:
1241         self._adj.setdefault(stop, [])
1242
1243     def add_edge(self, a: str, b: str, minutes: int, bidirectional: bool = True) -> None:
1244         if minutes < 0:
1245             raise ValueError("minutes must be non-negative")
1246         self._add_bidirectional(a, b, minutes)
1247         self._add_bidirectional(b, a, minutes)
1248         self._adj[a].append((b, minutes))
1249         self._adj[b].append((a, minutes))
1250
1251     def shortest_path(self, start: str, end: str) -> Tuple[str, List[str]]:
1252         if start not in self._adj or end not in self._adj:
1253             raise KeyError("start/end stop not found")
1254
1255         dist: Dict[str, int] = {start: 0}
1256         prev: Dict[str, Optional[str]] = {start: None}
1257         pq: List[Tuple[int, str]] = [(0, start)]
1258
1259         while pq:
1260             d, u = heapq.heappop(pq)
1261             if d != dist.get(u, 10**18):
1262                 continue
1263             if u == end:
1264                 break
1265             for v, w in self._adj[u]:
1266                 nd = d + w
1267                 if nd < dist.get(v, 10**18):
1268                     dist[v] = nd
1269                     prev[v] = u
1270                     heapq.heappush(pq, (nd, v))
1271
1272         if end not in dist:
1273             return (10**18, [])
1274
1275         # Reconstruct path
1276         path: List[str] = []
1277         cur: Optional[str] = end
1278         while cur is not None:
1279             path.append(cur)
1280             cur = prev.get(cur)
1281         path.reverse()
1282         return dist[end], path
1283
1284 # =====
1285 # B) Cafeteria Queue System (Priority Queue)
1286 # =====
1287
1288 class CafeteriaOrder:
1289     """
1290     order_id: int
1291     student_id: str
1292     item: str
1293     priority: int # Higher number => higher priority
1294     """
1295
1296 class CafeteriaOrderSystem:
1297     """
1298     Data structure: Priority Queue (heap)
1299     - Serve highest priority first; tie-break by arrival order.
1300     """
1301
1302     def __init__(self) -> None:
1303         self._heap: List[CafeteriaOrder] = []
1304         self._counter = itertools.count()
1305
1306     def place_order(self, student_id: str, item: str, priority: int = 0) -> CafeteriaOrder:
1307         order_id = next(self._counter)
1308         order = CafeteriaOrder(order_id, student_id, item, item, priority)
1309         # Insert in heap, insert priority in heap priority queue first
1310         heapq.heappush(self._heap, (-priority, order_id, order))
1311         return order
1312
1313     def serve_next(self) -> Optional[CafeteriaOrder]:
1314         if not self._heap:
1315             return None
1316         _, _, order = heapq.heappop(self._heap)
1317         return order
1318
1319     def pending_count(self) -> int:
1320         return len(self._heap)
1321
1322 # =====
1323 # Demo (optional)
1324 # =====
1325
1326 def main() -> None:
1327     # =====
1328     # 1) Attendance Tracker
1329     # =====
1330     att = AttendanceTracker()
1331     att.mark("S1", "2023-10-17", True)
1332     att.mark("S1", "2023-10-18", False)
1333     print("Attendance % S1", round(att.attendance_percent("S1"), 2))
1334
1335     # =====
1336     # 2) Events
1337     # =====
1338     events = EventRegistrationSystem()
1339     events.create_event("E100", "AI Workshop", capacity=2)
1340     for sid in ["S1", "S2", "S3"]:
1341         events.request_registration("E100", sid)
1342     events.process_next_request("E100")
1343     print("Confirmed E100: ", events.confirmed_list("E100"))
1344     print("Waitlist E100: ", events.waitlist_list("E100"))
1345     events.cancel_registration("E100", "S2")
1346     print("After Cancel S2, confirmed: ", events.confirmed_list("E100"))
1347
1348     # =====
1349     # 3) Library
1350     # =====
1351     lib = LibrarySystem()
1352     lib.add_book("9781492051301", "Effective Java", copies=2)
1353     lib.add_book("9781492051307", "Fluent Python", copies=1)
1354     print("Borrow Fluent Python: ", lib.borrow("S1", "9781492051307"))
1355     print("Borrow Fluent Python again: ", lib.borrow("S1", "9781492051307"))
1356     print("Cataling in order: ", [(b.label, b.available_copies) for b in lib.catalog_in_order()])
1357
1358     # =====
1359     # 4) Bus Network
1360     # =====
1361     buses = BusNetwork()
1362     buses.add_route("Hostel", "Gate", 5)
1363     buses.add_route("Gate", "Library", 4)
1364     buses.add_route("Hostel", "Cafeteria", 5)
1365     buses.add_route("Cafeteria", "Library", 4)
1366     minutes, path = buses.shortest_path("Hostel", "Library")
1367     print("Shortest bus path Hostel-Library: ", path, "minutes", minutes)
1368
1369     # =====
1370     # 5) Cafeteria (Priority Queue)
1371     # =====
1372     cafe = CafeteriaOrderSystem()
1373     cafe.place_order("S1", "Sandwich", priority=4)
1374     cafe.place_order("S1", "Coffee", priority=2) # Higher priority
1375     cafe.place_order("S1", "Burger", priority=1)
1376     print("Serve order: ", cafe.serve_next())
1377     print("Serve order: ", cafe.serve_next())
1378
1379 if __name__ == "__main__":
1380     main()

```

Output:

```
Attendance S1 %: 50.0
Confirmed E100: ['S1', 'S2']
Waitlist E100: ['S3']
After cancel S2, confirmed: ['S1', 'S3']
Borrow Fluent Python: True
Borrow Fluent Python again: False
Catalog in order: [('9780134685991', 2), ('9781492051367', 0)]
Shortest bus path Hostel->Library: ['Hostel', 'Cafeteria', 'Library'] minutes: 9
Attendance S1 %: 50.0
Confirmed E100: ['S1', 'S2']
Waitlist E100: ['S3']
After cancel S2, confirmed: ['S1', 'S3']
Borrow Fluent Python: True
Borrow Fluent Python again: False
Catalog in order: [('9780134685991', 2), ('9781492051367', 0)]
Shortest bus path Hostel->Library: ['Hostel', 'Cafeteria', 'Library'] minutes: 9
Waitlist E100: ['S3']
After cancel S2, confirmed: ['S1', 'S3']
Borrow Fluent Python: True
Borrow Fluent Python again: False
Catalog in order: [('9780134685991', 2), ('9781492051367', 0)]
Shortest bus path Hostel->Library: ['Hostel', 'Cafeteria', 'Library'] minutes: 9
Borrow Fluent Python again: False
Catalog in order: [('9780134685991', 2), ('9781492051367', 0)]
Shortest bus path Hostel->Library: ['Hostel', 'Cafeteria', 'Library'] minutes: 9
Serve order: CafeteriaOrder(order_id=2, student_id='S2', item='Coffee', priority=2)
Shortest bus path Hostel->Library: ['Hostel', 'Cafeteria', 'Library'] minutes: 9
Serve order: CafeteriaOrder(order_id=2, student_id='S2', item='Coffee', priority=2)
Serve order: CafeteriaOrder(order_id=2, student_id='S2', item='Coffee', priority=2)
Serve order: CafeteriaOrder(order_id=3, student_id='S3', item='Burger', priority=1)
Serve order: CafeteriaOrder(order_id=3, student_id='S3', item='Burger', priority=1)
```

Task Description #10: Smart E-Commerce Platform – Data Structure

Prompt:

Solve this clearly and concisely.

Design a Smart E-Commerce Platform with:

Shopping Cart Management – Add/remove products dynamically

Order Processing System – Process orders in placement order

Top-Selling Products Tracker – Rank products by sales count

Product Search Engine – Fast lookup using product ID

Delivery Route Planning – Connect warehouses and delivery locations

Choose the most appropriate data structure for each feature from:

Stack, Queue, Priority Queue, Linked List, BST, Graph, Hash Table, Deque

Output as a table:

Feature | Data Structure | 2–3 sentence justification

Code:

```

1 from collections import deque
2 import heapq
3 from typing import Dict, List, Tuple, Optional
4
5
6 # -----
7 # Product model
8 # -----
9 class Product:
10     def __init__(self, product_id: int, name: str, price: float):
11         self.id = product_id
12         self.name = name
13         self.price = price
14
15     def __repr__(self):
16         return f"Product(id={self.id}, name='{self.name}', price={self.price})"
17
18
19 # -----
20 # Product Search Engine (Hash Table)
21 # -----
22 class ProductSearchEngine:
23     def __init__(self):
24         # Hash Table: product_id -> Product
25         self.products: Dict[int, Product] = {}
26
27     def add_product(self, product: Product):
28         self.products[product.id] = product
29
30     def get_product(self, product_id: int) -> Optional[Product]:
31         return self.products.get(product_id)
32
33     def remove_product(self, product_id: int):
34         self.products.pop(product_id, None)
35
36
37 # -----
38 # Shopping Cart (linked list)
39 # -----
40 class CartNode:
41     def __init__(self, product: Product, quantity: int):
42         self.product = product
43         self.quantity = quantity
44         self.next: Optional[CartNode] = None
45
46
47 class ShoppingCart:
48     def __init__(self):
49         self.head: Optional[CartNode] = None
50
51     def add_product(self, product: Product, quantity: int = 1):
52         """
53         If product already exists in the list, increase quantity.
54         Otherwise, add new node at the front (O(1) insertion).
55         """
56         node = self.head
57         while node:
58             if node.product.id == product.id:
59                 node.quantity += quantity
60                 return
61             node = node.next
62
63         new_node = CartNode(product, quantity)
64         new_node.next = self.head
65         self.head = new_node
66
67     def remove_product(self, product_id: int, quantity: int = None):
68         """
69         Remove some or all quantity of a product.
70         If quantity is None or reaches 0, remove the node.
71         """
72         prev = None
73         node = self.head
74
75         while node:
76             if node.product.id == product_id:
77                 if quantity is None or node.quantity <= quantity:
78                     # delete the node
79                     if prev:
80                         prev.next = node.next
81                     else:
82                         self.head = node.next

```

```

113         else:
114             node.quantity -= quantity
115         return
116         prev = node
117         node = node.next
118
119     def list_items(self) -> List[Tuple[Product, int]]:
120         result = []
121         node = self.head
122         while node:
123             result.append((node.product, node.quantity))
124             node = node.next
125         return result
126
127     def total_price(self) -> float:
128         return sum(node.product.price * node.quantity
129                     for node in self.iter_nodes())
130
131     def iter_nodes(self):
132         node = self.head
133         while node:
134             yield node
135             node = node.next
136
137 # -----
138 # Order Processing System (Queue)
139 # -----
140 class Order:
141     _next_id = 1
142
143     def __init__(self, cart_snapshot: List[Tuple[Product, int]]):
144         self.id = Order._next_id
145         Order._next_id += 1
146         self.items = cart_snapshot # list of (Product, quantity)
147
148     def __repr__(self):
149         return f"Order(id={self.id}, items={[(p.id, q) for p, q in self.items]})"
150
151 class OrderProcessingSystem:
152     def __init__(self):
153         # Queue of orders (FIFO)
154         self.queue: deque[Order] = deque[Order]()
155
156     def place_order(self, cart: ShoppingCart) -> Order:
157         order = Order(cart.list_items())
158         self.queue.append(order)
159         return order
160
161     def process_next_order(self) -> Optional[Order]:
162         if not self.queue:
163             return None
164         return self.queue.popleft()
165
166     def pending_orders(self) -> int:
167         return len(self.queue)
168
169 # -----
170 # Top-Selling Products Tracker (Priority Queue / Max-Heap)
171 # -----
172 class TopSellingProductsTracker:
173     def __init__(self):
174         # product_id -> sales_count
175         self.sales: Dict[int, int] = {}
176         # priority queue entries: (sales_count, product_id)
177         self.heap: List[Tuple[int, int]] = []
178
179     def record_sale(self, product_id: int, quantity: int = 1):
180         self.sales[product_id] = self.sales.get(product_id, 0) + quantity
181         # Push new priority entry, lazy update (we'll verify against self.sales on pop)
182         heapq.heappush(self.heap, (-self.sales[product_id], product_id))
183
184     def top_k(self, k: int) -> List[Tuple[int, int]]:
185         """
186         Returns list of (product_id, sales_count) for top k products.
187         Uses lazy removal from the heap to keep it consistent.
188         """
189

```

```

190         result = []
191         seen = set[int]()
192
193         while self.heap and len(result) < k:
194             neg_sales, pid = heapq.heappop(self.heap)
195             current_sales = self.sales.get(pid, 0)
196
197             if current_sales == -neg_sales and pid not in seen:
198                 result.append((pid, current_sales))
199                 seen.add(pid)
200
201             # push back the elements we popped that are still valid
202             for pid in seen:
203                 heapq.heappush(self.heap, (-self.sales[pid], pid))
204
205         return result
206
207 # -----
208 # Delivery Route Planning (Graph + Dijkstra)
209 # -----
210 class DeliveryRoutePlanner:
211     def __init__(self):
212         # Graph as adjacency list: node -> list of (neighbor, distance)
213         self.graph: Dict[str, List[Tuple[str, float]]] = {}
214
215     def add_location(self, name: str):
216         if name not in self.graph:
217             self.graph[name] = []
218
219     def add_route(self, from_loc: str, to_loc: str, distance: float, bidirectional: bool = True):
220         self.add_location(from_loc)
221         self.add_location(to_loc)
222         self.graph[from_loc].append((to_loc, distance))
223         if bidirectional:
224             self.graph[to_loc].append((from_loc, distance))
225
226     def shortest_path(self, start: str, end: str) -> Tuple[float, List[str]]:
227         """
228         Dijkstra's algorithm returns (distance, path).
229         Distance is float('inf') if no path exists.
230         """
231         if start not in self.graph or end not in self.graph:
232             return float('inf'), []
233
234         # min-heap: (distance, node, path)
235         heap = [(0.0, start, [start])]
236         visited = set[str]()
237
238         while heap:
239             dist, node, path = heapq.heappop(heap)
240             if node in visited:
241                 continue
242             visited.add(node)
243
244             if node == end:
245                 return dist, path
246
247             for neighbor, weight in self.graph[node]:
248                 if neighbor not in visited:
249                     heapq.heappush(heap, (dist + weight, neighbor, path + [neighbor]))
250
251         return float('inf'), []
252
253 # -----
254 # Example usage
255 # -----
256 if __name__ == "__main__":
257     # Product search engine
258     search_engine = ProductSearchEngine()
259     p1 = Product(1, "Laptop", 1000.0)
260     p2 = Product(2, "Phone", 500.0)
261     p3 = Product(3, "Headphones", 100.0)
262     for p in (p1, p2, p3):
263         search_engine.add_product(p)

```

```

239 # Shopping cart
240 cart = ShoppingCart()
241 cart.add_product(search_engine.get_product(1), 1)
242 cart.add_product(search_engine.get_product(2), 2)
243 cart.add_product(search_engine.get_product(3), 3)
244 cart.remove_product(3, 1) # remove 1 headphone
245
246 print("Cart items:", cart.list_items())
247 print("Total price:", cart.total_price())
248
249 # Order processing
250 ops = OrderProcessingSystem()
251 order1 = ops.place_order(cart)
252 print("Placed order:", order1)
253 print("Pending orders:", ops.pending_orders())
254 processed = ops.process_next_order()
255 print("Processed order:", processed)
256 print("Pending orders:", ops.pending_orders())
257
258 # Top-selling products
259 tracker = TopSellingProductsTracker()
260 tracker.record_sale(10) # Laptop sold 10
261 tracker.record_sale(2, 5) # Phone sold 5
262 tracker.record_sale(3, 7) # Headphones sold 7
263 print("Top 2 products (id, sales):", tracker.top_k(2))
264
265 # Delivery route planner
266 planner = DeliveryRoutePlanner()
267 planner.add_route("WarehouseA", "City1", 10.0)
268 planner.add_route("WarehouseA", "City2", 20.0)
269 planner.add_route("City1", "City2", 5.0)
270 planner.add_route("City2", "City3", 7.0)
271
272 dist, path = planner.shortest_path("WarehouseA", "City3")
273 print("Shortest route WarehouseA -> City3:", path, "distance:", dist)

```

Output:

```

Cart items: [(Product(id=3, name='Headphones', price=100.0), 2), (Product(id=2, name='Phone', price=500.0), 2), (Product(id=1, name='Laptop', price=1000.0), 1)]
Total price: 2200.0
Placed order: Order(id=1, items=[(3, 2), (2, 2), (1, 1)])
Pending orders: 1
Processed order: Order(id=1, items=[(3, 2), (2, 2), (1, 1)])
Pending orders: 0
Top 2 products (id, sales): [(1, 10), (3, 7)]
Shortest route WarehouseA -> City3: ['WarehouseA', 'City1', 'City2', 'City3'] distance: 22.0
PS C:\2403A51L03\3-2\AI_A_C\cursor AI>

Total price: 2200.0
Placed order: Order(id=1, items=[(3, 2), (2, 2), (1, 1)])
Pending orders: 1
Processed order: Order(id=1, items=[(3, 2), (2, 2), (1, 1)])
Pending orders: 0
Top 2 products (id, sales): [(1, 10), (3, 7)]
Shortest route WarehouseA -> City3: ['WarehouseA', 'City1', 'City2', 'City3'] distance: 22.0
Pending orders: 0
Top 2 products (id, sales): [(1, 10), (3, 7)]
Shortest route WarehouseA -> City3: ['WarehouseA', 'City1', 'City2', 'City3'] distance: 22.0
Shortest route WarehouseA -> City3: ['WarehouseA', 'City1', 'City2', 'City3'] distance: 22.0

```