SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

AN AUTONOMOUS INSTITUTION

Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (IoT AND CYBER SECURITY INCLUDING BLOCK CHAIN TECHNOLOGY)

23ITB201-DATA STRUCTURES & ALGORITHMS

Solved Puzzles Questions and Answers

Unit 1: List ADT, Stack ADT, Queue ADT, Sorting, Searching & Hashing

Puzzle 1: Stack Puzzle

Puzzle:
You have an array of integers [3, 1, 4, 1, 5, 9, 2, 6]. Using a single stack, sort the array in non-decreasing order. You can push and pop elements, but you cannot use any additional data structures except for the stack and the array itself.

Answer:
The puzzle can be solved by repeatedly pushing elements from the array to the stack and then popping them back in order to sort them:

1. Push the first element (3) onto the stack.
2. For each subsequent element, pop elements from the stack until you find the correct position for the current element, then push it.
3. Repeat until the array is sorted in non-decreasing order.

Puzzle 2: Queue Puzzle

Puzzle:
You have a circular queue that can hold a maximum of 5 integers. How can you insert 6 integers into the queue without causing an overflow?

Answer:
You cannot insert 6 integers into a circular queue that can hold only 5 integers without causing an overflow. This puzzle illustrates the importance of understanding the capacity limitations of data structures.

Puzzle 3: Linked List Puzzle

Puzzle:
You are given a singly linked list with an unknown length. How can you find the middle element of the list in a single traversal?

Answer:
Use the two-pointer technique:

1. Initialize two pointers, slow and fast, at the head of the list.
2. Move the slow pointer one step at a time and the fast pointer two steps at a time.
3. When the fast pointer reaches the end, the slow pointer will be at the middle of the list.

Puzzle 4: Sorting Puzzle

Puzzle:
You have an array of numbers where each number is within 1 of its correct position if the array were sorted. How can you sort the array efficiently?

Answer:
You can use **Insertion Sort** for this puzzle because it is efficient for nearly sorted arrays:

1. Traverse the array and use insertion sort to place each element in its correct position.
2. The time complexity will be nearly O(n) due to the small number of displacements required.

Puzzle 5: Hashing Puzzle

Puzzle:
You need to design a hash function for a phone directory where the hash table size is prime and smaller than the number of entries. How can you minimize collisions?

Answer:
Use a **modular arithmetic hash function** with a prime number as the table size to distribute entries uniformly:

1. Convert the phone numbers into integers.
2. Use the hash function h(x)=xmod mh(x) = x \mod mh(x)=xmodm, where mmm is the prime number representing the table size.
3. To further reduce collisions, implement double hashing or quadratic probing as collision resolution techniques.

Unit 2: Tree ADT, Binary Tree ADT, Tree Traversals, AVL Trees, Heaps

Puzzle 1: Binary Tree Puzzle

Puzzle:
You have a binary tree where every node has a value of either 0 or 1. Your task is to prune the tree so that all subtrees that do not contain the value 1 are removed. How would you do this?

Answer:
Use a recursive approach:

1. Traverse the tree from the bottom up.
2. If a subtree (left or right) of a node contains only 0, remove it by setting the corresponding child pointer to NULL.
3. Continue this process until the entire tree is pruned.

Puzzle 2: AVL Tree Puzzle

Puzzle:
You are given an AVL tree, and you need to insert a new node such that the tree remains balanced. What sequence of operations will you use to maintain the AVL tree property?

Answer:
After inserting the new node, check the balance factor of each node starting from the newly inserted node back to the root:

1. If a node becomes unbalanced (balance factor becomes greater than 1 or less than -1), perform the necessary rotations:
   - Single right rotation for left-left case.
   - Single left rotation for right-right case.
   - Left-right rotation for left-right case.
   - Right-left rotation for right-left case.
2. Recalculate the heights of the affected nodes.

Puzzle 3: Heap Puzzle

Puzzle:
Given a max-heap, how can you find the second largest element in the heap?

Answer:
In a max-heap, the second largest element will be one of the children of the root:

1. Compare the left and right children of the root.
2. The larger of the two is the second largest element in the heap.

Puzzle 4: Tree Traversal Puzzle

Puzzle:
You have a binary tree and a linked list. The task is to determine whether the linked list is a subtree of the binary tree, assuming both contain integer values.

Answer:
Perform a DFS traversal of the binary tree:

1. For each node in the tree, start matching it with the head of the linked list.
2. If the entire linked list matches a path in the tree, then the linked list is a subtree.
3. Otherwise, continue the search.

Puzzle 5: Knapsack Puzzle

Puzzle:
You are given a knapsack with a maximum weight capacity W and a list of items, each with a weight and a value. How can you determine which items to include in the knapsack to maximize the total value?

Answer:
Use the **0/1 Knapsack Dynamic Programming** approach:

1. Define a 2D DP table where the rows represent items and columns represent weight capacities.
2. Fill the table based on whether including an item yields a higher value than excluding it.
3. The final cell of the table will give you the maximum value, and you can trace back to find the items included.

Unit 3: Graph ADT, Graph Traversals, DAG, Topological Ordering, Dynamic Programming

Puzzle 1: Graph Traversal Puzzle

Puzzle:
Given an undirected graph, how would you determine if it contains any cycles?

Answer:
Use DFS:

1. Traverse the graph using DFS, keeping track of visited nodes.
2. If you revisit a node that is not the parent of the current node, then a cycle exists in the graph.

Puzzle 2: Topological Sorting Puzzle

Puzzle:
Given a directed acyclic graph (DAG) representing tasks with dependencies, how would you determine a valid order to perform all tasks?

Answer:
Perform **Topological Sorting**:

1. Use a DFS to visit each node.
2. Once a node and all its descendants are visited, add it to the topological order.
3. Continue until all nodes are visited, resulting in a valid task order.

Puzzle 3: Shortest Path Puzzle

Puzzle:
You have a weighted graph with some negative weights, but no negative cycles. How would you find the shortest path from a source node to all other nodes?

Answer:
Use the **Bellman-Ford Algorithm**:

1. Initialize the distance to the source node as 0 and to all other nodes as infinity.
2. Relax all edges up to (V-1) times, where V is the number of vertices.
3. The final distances represent the shortest paths from the source to all nodes.

Puzzle 4: Minimum Spanning Tree Puzzle

Puzzle:
Given a connected, undirected graph with weights, how would you find a spanning tree that connects all vertices with the minimum possible total edge weight?

Answer:
Use **Kruskal's or Prim's Algorithm**:

1. For Kruskal's, sort all edges by weight and add them to the spanning tree, ensuring no cycles are formed.
2. For Prim's, start with any vertex and grow the tree by adding the smallest edge connecting the tree to a new vertex.

Puzzle 5: P, NP, and NP-Complete Puzzle

Puzzle:
You are tasked with finding a Hamiltonian path (a path that visits every vertex exactly once) in a given graph. How can you determine if this problem is solvable in polynomial time?

Answer:
The Hamiltonian path problem is **NP-complete**, meaning it is unlikely to have a polynomial-time solution:

1. You can attempt to solve it using backtracking, checking each possible path.
2. Alternatively, you could use heuristics or approximation algorithms, but there is no known polynomial-time algorithm for this problem.

Unit 4: Dynamic Programming, Greedy Algorithms, Branch & Bound

Puzzle 1: Dynamic Programming Puzzle

Puzzle:
You are given a staircase with N steps, and you can either take 1 step or 2 steps at a time. How many distinct ways can you reach the top?

Answer:
Use dynamic programming:

1. Define DP[i] as the number of ways to reach step i.
2. DP[i] = DP[i-1] + DP[i-2].
3. Initialize DP[0] = 1 and DP[1] = 1, then compute DP[N] for the answer.

Puzzle 2: Greedy Algorithm Puzzle

Puzzle:
You are given a set of intervals, and you need to find the maximum number of non-overlapping intervals. How can you do this?

Answer:
Use a greedy approach:

1. Sort the intervals by their end times.
2. Select the first interval and then select the next interval that starts