|  |  |
| --- | --- |
| Semester | T.E. Semester V |
| Subject | Artificial Intelligence Lab |
| Subject Professor In-charge | Ms. Rasika Ransing |
| Laboratory | L 011 |

|  |  |
| --- | --- |
| Student Name | Harsh Rawte |
| Roll Number | 22101A0047 |

|  |  |
| --- | --- |
| Experiment  Number | 03 |
| Experiment Title | Program on uninformed search methods (BFS) |
| Objectives  (Skill Set /  Knowledge  Tested /  Imparted) | **To implement Breadth First search**   Understanding Breadth-First Search (BFS) Algorithm   Tree and Graph Representation   Problem-Solving and Analytical Thinking   Coding and Algorithm Implementation   Efficient Search Techniques   Real-World Applications of Algorithms |
| Theory  Code | **Breadth First search (BFS)**  In the context of the departmental store problem, we can model the store as a tree where each department is a node, and sub-departments or items are the children of these nodes. The root node is the main department store, with categories branching into sub-categories and eventually leading to individual items. The BFS algorithm helps in efficiently searching for an item in the store, exploring all departments and sub-departments in a breadth-first manner.  The BFS search begins at the main store and progressively explores each category, sub-category, and item until the desired item is found  import java.util.\*;  // Class representing the store  class Department {  String name; // Name of the department  List<Department> subDepartments; // List of sub-departments  public Department(String name) {  this.name = name;  this.subDepartments = new ArrayList<>();  }  public void addSubDepartment(Department subDepartment) {  this.subDepartments.add(subDepartment);  }  public String getName() {  return name;  }  public List<Department> getSubDepartments() {  return subDepartments;  }  // Method to print the department tree  public void printTree(String prefix) {  System.out.println(prefix + name);  for (Department sub : subDepartments) {  sub.printTree(prefix + " -> ");  }  }  }  public class DepartmentStore {  // Method to perform BFS and search for the item, returning the path  public static List<String> searchItemBFS(Department root, String item) {  if (root == null) {  return null;  }  Queue<Node> queue = new LinkedList<>();  queue.add(new Node(root, Arrays.asList(root.getName())));  Set<String> visited = new HashSet<>(); // Set to keep track of visited nodes  while (!queue.isEmpty()) {  Node currentNode = queue.poll();  Department currentDept = currentNode.department;  List<String> path = currentNode.path;  // Modified print statement for BFS visualization  System.out.println("Currently inspecting: " + currentDept.getName() + " | Route so far: " + String.join(" -> ", path));  // Check if the current department has the item  if (currentDept.getName().equalsIgnoreCase(item)) {  return path;  }  // Add all sub-departments to the queue with updated path  for (Department subDepartment : currentDept.getSubDepartments()) {  if (!visited.contains(subDepartment.getName())) { // Avoid revisiting nodes  visited.add(subDepartment.getName());  List<String> newPath = new ArrayList<>(path);  newPath.add(subDepartment.getName());  queue.add(new Node(subDepartment, newPath));  }  }  }  return null;  }  // Helper class to keep track of the department and the path  private static class Node {  Department department;  List<String> path;  Node(Department department, List<String> path) {  this.department = department;  this.path = path;  }  }  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  // Input store name  System.out.println("Enter the name of the departmental store:");  String storeName = scanner.nextLine();  // Input root department  Department root = new Department(storeName);  // Input categories  System.out.println("Enter the number of categories:");  int numCategories = getValidIntegerInput(scanner);  String[] categories = new String[numCategories];  Department[] categoryDepartments = new Department[numCategories];  System.out.println("Enter the names of all categories:");  String categoriesInput = scanner.nextLine();  String[] categoriesArray = categoriesInput.split("\\s\*,\\s\*"); // Split by comma and trim whitespace  if (categoriesArray.length != numCategories) {  System.out.println("The number of entered categories does not match the specified count.");  return;  }  for (int i = 0; i < numCategories; i++) {  categories[i] = categoriesArray[i];  categoryDepartments[i] = new Department(categories[i]);  root.addSubDepartment(categoryDepartments[i]);  }  // Input sub-categories and items  for (int i = 0; i < numCategories; i++) {  System.out.println("Enter the number of sub-categories in " + categories[i] + ":");  int numSubCategories = getValidIntegerInput(scanner);  String[] subCategories = new String[numSubCategories];  Department[] subCategoryDepartments = new Department[numSubCategories];  System.out.println("Enter the names of all sub-categories for " + categories[i] + ":");  String subCategoriesInput = scanner.nextLine();  String[] subCategoriesArray = subCategoriesInput.split("\\s\*,\\s\*"); // Split by comma and trim whitespace  if (subCategoriesArray.length != numSubCategories) {  System.out.println("The number of entered sub-categories does not match the specified count.");  return;  }  for (int j = 0; j < numSubCategories; j++) {  subCategories[j] = subCategoriesArray[j];  subCategoryDepartments[j] = new Department(subCategories[j]);  categoryDepartments[i].addSubDepartment(subCategoryDepartments[j]);  // Input items  System.out.println("Enter the number of items in " + subCategories[j] + ":");  int numItems = getValidIntegerInput(scanner);  String[] items = new String[numItems];  System.out.println("Enter the names of all items for " + subCategories[j] + ":");  String itemsInput = scanner.nextLine();  String[] itemsArray = itemsInput.split("\\s\*,\\s\*"); // Split by comma and trim whitespace  if (itemsArray.length != numItems) {  System.out.println("The number of entered items does not match the specified count.");  return;  }  for (int k = 0; k < numItems; k++) {  items[k] = itemsArray[k];  Department item = new Department(items[k]);  subCategoryDepartments[j].addSubDepartment(item);  }  }  }  // Keep the original print statement for store structure  System.out.println("\nDepartment Store Structure:");  root.printTree("");  // Input item to find  System.out.println("\nEnter the name of the item you are searching for:");  String searchItem = scanner.nextLine();  // Search for the item  List<String> path = searchItemBFS(root, searchItem);  // Output the result  if (path != null) {  System.out.println("Hooray! \"" + searchItem + "\" is located in: " + path.get(path.size() - 1));  System.out.println("Journey to the item: " + String.join(" -> ", path));  } else {  System.out.println("Oops! Couldn't locate \"" + searchItem + "\" in the departments.");  }  // Close the scanner  scanner.close();  }  // Helper method to get valid integer input from the user  private static int getValidIntegerInput(Scanner scanner) {  while (true) {  try {  int input = Integer.parseInt(scanner.nextLine());  if (input < 0) {  throw new NumberFormatException();  }  return input;  } catch (NumberFormatException e) {  System.out.println("Invalid input. Please enter a non-negative integer.");  }  }  }  } |
| Output | C:\Users\Lenovo\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\7B5BDFC3463973353C96415D45663CC4\WhatsApp Image 2024-08-15 at 16.16.27_a9f93127.jpg  C:\Users\Lenovo\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\AAE14BDA33AA45A1D45D45011529B806\WhatsApp Image 2024-08-15 at 16.16.28_76f3bb16.jpg |
| Conclusion | The departmental store problem is a practical application of BFS, where the store structure can be navigated efficiently to find any item. By using BFS, we ensure that the search proceeds level by level, covering all categories and sub-categories before moving deeper into the tree. This ensures that the search is thorough and avoids missing any items. BFS is particularly suitable for scenarios where the shortest path or first occurrence of an item is required, making it an ideal choice for our store layout search problem. |