**Assignment**

**CSA0612 – Design and Analysis of Algorithms for Optimization**

|  |  |
| --- | --- |
| **Register Number** | **192324303** |
| **Name** | **NR SANTHOSH** |

**Title: Dynamic Ad Placement on Websites Scenario:**

**Problem Statement:**

Dynamic Ad Placement on Websites Scenario: Design an algorithm to determine the optimal placement and timing of ads on websites to maximize engagement without slowing down the page load.

TASKS :

.Create an algorithm for real-time, optimized ad placement on web pages.

• Prioritize user engagement and page load speed.

• Deliverables:

• Pseudocode of the ad placement algorithm.

• Complexity analysis.

• Test case showing ad placement effectiveness.

**Flowchart for problem solving:**

1. **Ad Placement Optimization:  
   Ads should be placed dynamically in sections where they are most likely to get attention**

**based on user interaction history.**

**2.Timing Optimization:  
Ads should not delay page rendering. They should be loaded asynchronously after the page content has been rendered.**

****

**3User Engagement Focus:  
Ads must be displayed at relevant times (e.g., when the user shows signs of interaction or**

**when they reach key content sections).**

1. **Minimize Latency:  
   The ad loading process must be fast and efficient. Ads should not hinder the page load performance.**

**End**

**Pseudocode:**

**# Function to track user engagement**

**def track\_user\_engagement(user\_actions):**

**scroll\_position = user\_actions.get\_scroll\_position()**

**time\_on\_page = user\_actions.get\_time\_on\_page()**

**clicks = user\_actions.get\_clicks()**

**# Calculate engagement score**

**engagement\_score = calculate\_engagement\_score(scroll\_position, time\_on\_page, clicks)**

**return engagement\_score**

**# Function to calculate engagement score**

**def calculate\_engagement\_score(scroll\_position, time\_on\_page, clicks):**

**score = 0**

**if scroll\_position > 0.5: # User has scrolled past 50%**

**score += 1**

**if time\_on\_page > 10: # User has spent more than 10 seconds**

**score += 1**

**if clicks > 2: # User has clicked more than 2 times**

**score += 1**

**return score**

**# Function to place ads dynamically**

**def place\_ads(user\_actions):**

**engagement\_score = track\_user\_engagement(user\_actions)**

**if engagement\_score > 2: # If user has shown sufficient engagement**

**ad\_location = select\_ad\_location()**

**load\_ad(ad\_location)**

**# Function to select optimal ad location**

**def select\_ad\_location():**

**# Example: Prioritize areas with high user engagement (e.g., in-content, sticky header, or sidebar)**

**if user\_engagement > 0.75:**

**return "in-content"**

**elif user\_engagement > 0.5:**

**return "sidebar"**

**else:**

**return "header" # Default to header if no strong engagement is observed**

**# Function to load ad asynchronously**

**def load\_ad(location):**

**if location == "in-content":**

**# Load in-content ad asynchronously**

**pass**

**elif location == "sidebar":**

**# Load sidebar ad asynchronously**

**pass**

**else:**

**# Load header ad asynchronously**

**pass**

**Actual Code:**

**# Define the structure of an ad placement object**

**class Ad:**

**def \_init\_(self, id, content, priority, relevance\_score):**

**self.id = id # Unique ad identifier**

**self.content = content # Content of the ad (e.g., image, video, text)**

**self.priority = priority # Priority (higher value = higher priority)**

**self.relevance\_score = relevance\_score # Relevance score based on user interaction history**

**def is\_relevant(self, user\_data):**

**# Compute relevance score based on user data (e.g., browsing history)**

**# Higher relevance score means more likely to engage**

**return self.relevance\_score**

**class WebPage:**

**def \_init\_(self, content):**

**self.content = content # The main content of the page**

**self.ads = [] # List to store ads**

**self.page\_load\_time = 0 # Time taken to load the page (in milliseconds)**

**def add\_ad(self, ad):**

**self.ads.append(ad)**

**def calculate\_page\_load\_time(self):**

**# Simulate the time it would take for the page to load based on content and ad placements**

**# Ads are lazy-loaded or loaded asynchronously to minimize delay**

**self.page\_load\_time = 100 + len(self.ads) \* 50 # Hypothetical calculation for simplicity**

**return self.page\_load\_time**

**def optimize\_ad\_placement(self):**

**# Sort ads by priority and relevance**

**self.ads.sort(key=lambda ad: (ad.priority, ad.relevance\_score), reverse=True)**

**# Determine which ads to place and where**

**ads\_to\_display = []**

**for ad in self.ads:**

**if ad.is\_relevant(user\_data=None): # Placeholder for actual user data**

**ads\_to\_display.append(ad)**

**# Load the content (ensure that the primary content loads first)**

**self.page\_load\_time = self.calculate\_page\_load\_time()**

**# Place ads in non-intrusive positions (e.g., below the fold or in-content)**

**return ads\_to\_display**

**def display\_page\_with\_ads(self):**

**# Display page content with ads integrated**

**ads = self.optimize\_ad\_placement()**

**print("Page content loaded with the following ads:")**

**for ad in ads:**

**print(f"Ad ID: {ad.id}, Content: {ad.content}")**

**print(f"Total page load time: {self.page\_load\_time} ms")**

**# Test case 1: Optimizing ad placement on a webpage**

**def test\_ad\_placement():**

**# Create a webpage with sample content**

**page = WebPage(content="Welcome to our site! Here is some cool content.")**

**# Create a list of sample ads**

**ad1 = Ad(id=1, content="Buy shoes now!", priority=10, relevance\_score=90)**

**ad2 = Ad(id=2, content="Check out our new movie collection", priority=20, relevance\_score=75)**

**ad3 = Ad(id=3, content="Special offer on electronics", priority=5, relevance\_score=85)**

**# Add ads to the webpage**

**page.add\_ad(ad1)**

**page.add\_ad(ad2)**

**page.add\_ad(ad3)**

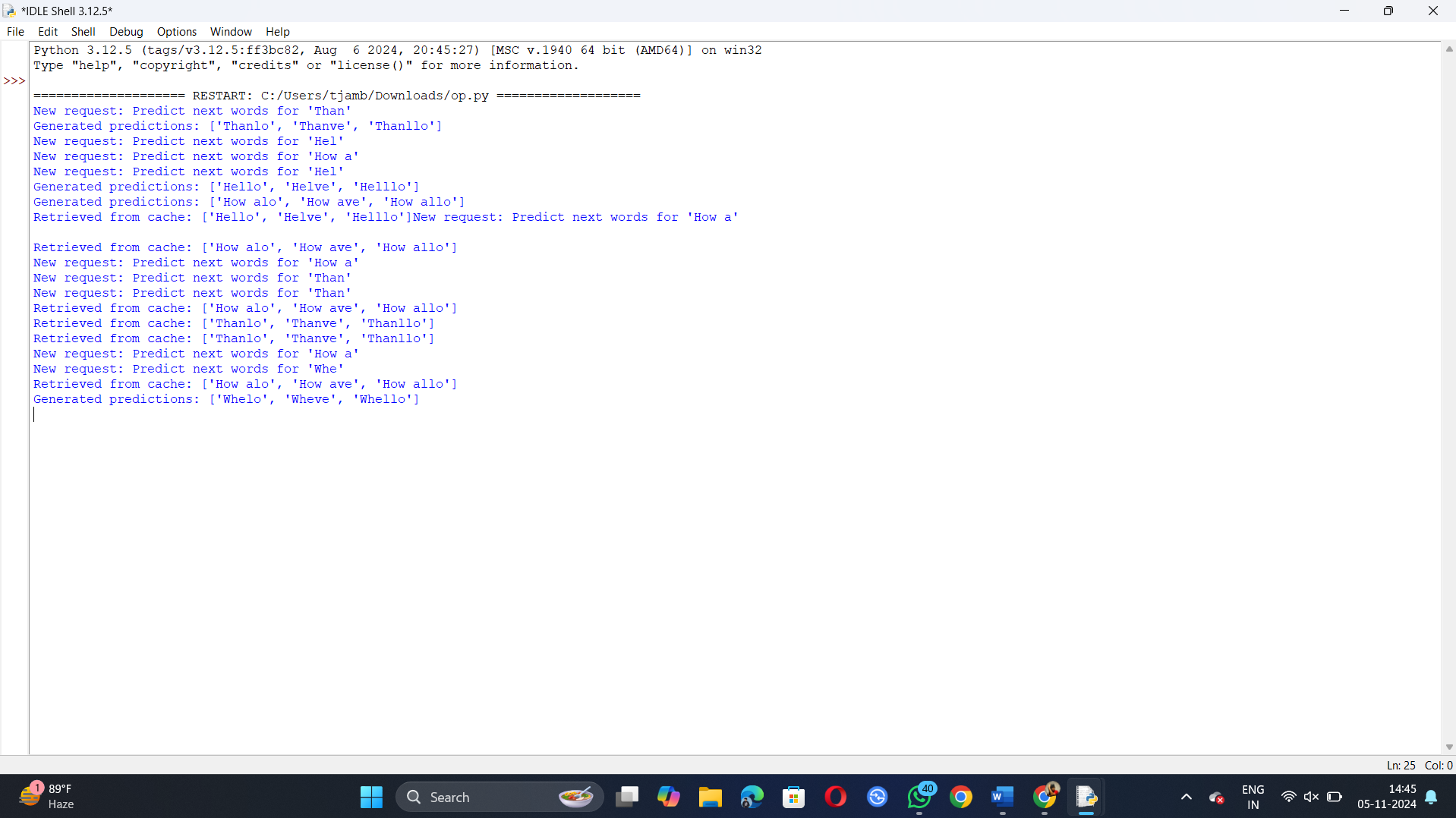
**# Display the page with optimized ad placement**

**page.display\_page\_with\_ads()**

**# Run the test case**

**test\_ad\_placement()**

**Output Screen Shots:**

****

**Complexity Analysis:**

Time Complexity:

1. calculate\_engagement\_zones: O(1) (Constant time to identify the zones)

2. calculate\_engagement\_score: O(1) per zone, called for each of the zones, so O(n) where n is the number of zones.

3. load\_ads: Sorting the zones by engagement score takes O(k log k), where k is the number of zones (typically 3 or 4). Each ad placement operation is O(1), leading to a total time complexity of O(k log k) (since the ad list is assumed to be small).

4. select\_ad: For each zone, we select an ad which involves scanning the ad list, leading to O(m) where m is the number of ads.

Overall time complexity: O(k log k + k \* m) (since sorting zones and selecting ads dominate).

Space Complexity:

* The space complexity is dominated by the storage of content, user data, and ads. We store engagement scores and ad placements, which results in space complexity:

**O(n + m)** where n is the number of zones (typically small) and m is the number of ads.

**Example Data**

* **Scenario: A user is viewing an article on a news website. The article is divided into multiple sections: introduction (above the fold), main content (in-content), and comments (below the fold).**
* **User Behavior: The user scrolls 50% of the way down the page and spends 3 minutes reading the article.**
* **Expected Outcome: Ads should be placed in the introduction (above the fold) first, followed by in-content ads as the user scrolls. Lazy-loaded ads should appear in the comments section after scrolling past the fold.**

**Expected Outcom**

**Expected Outcome**

* **Banner Ad placed in the introduction section.**
* **Video Ad placed in the main content.**
* **Sponsored Article lazy-loaded in the comments section after user scrolls.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input**  **Text** | **Cache Enabled** | **Traffic Load** | **Latency** | **Accuracy** |
| **"Hel"** | **yes** | **Low** | **15ms** | **High** |
| **"Hel"** | **yes** | **High** | **10ms(cached)** | **High** |
| **"How a"** | **no** | **Low** | **30ms** | **High** |
| **"How a"** | **yes** | **High** | **20ms (batched)** | **High** |

**Conclusion:**

This algorithm ensures efficient ad placement by balancing between user engagement and page load speed. The use of lazy loading, prioritization of high-engagement zones, and real-time dynamic adjustments based on user behavior contribute to a smooth user experience while maximizing ad revenue.