- NAME: M. HASSAN MALIK
- SAP_ID: 59610
- SUBJECT: ANALYSIS OF ALGORITHM
- TOPIC: ASSIGNMENT-I
- LECTURER: Sir USMAN SHARIF
- (RIPHAH INTERNATIONAL UNIVERSITY, ISLAMABAD)

: QUESTION 1 :

Write down driving directions from going from your school to home with the precision required from an algorithm description. Also specify:

a) variables
b) Input statement
c) Calculation
d) Output statement.
e) Assignments ?

SOLUTION

Algorithm description:- This algorithm provides step by step driving directions with precise instructions, ensuring clarity and logical flow.

Algorithm : Find route : School to home

Input: Starting location (school), destination (home).

Output: A sequence of driving instructions leading to destination.

1- Start.
2- Set current_location = school
3. Set destination = Home
4- While current_location ≠ destination, do:
a) Identify next road segment.
b) Compute the distance to the next landmark
c) Display instruction: "Move forward for [distance] meters on [road_name]".

d) If an intersection is reached.
i) Determine correct turn direction (left, right, straight).
ii) Display instruction: "Turn [direction] at [intersection_name]."
e) If a roundabout is encountered:
i) Identify the correct exit.
ii) Display the instruction: "Take exit [exit number] at [roundabout name]".
f- Update current location to the new position.
5) Display message: "You have reached your destination"
6) Stop.

Example Execution (step by step)

Input:

- **Starting point:** School
- **Destination:** Home.

Roads & Intersections:

Road A → Intersection 1 → Road B → Traffic signal → Road C → Roundabout → Road D → home.

Output:

1) Move forward for 200 metres on road A.
2) Turn left at intersection 1 onto Road B.
3) Move forward for 500 metres on road B.
4) Turn right at traffic signal onto road C.
5) Move forward for 300 meters on Road C.
6) Take the second exit at Roundabout onto Road D.
7) Move forward for 600 metres on Road D.
8) You have reached your destination.

Algorithm Breakdown:

a) Variables:

- Current_location: Track user's position.
- Destination: Final location (home).
- road_name: store the current road segment name.
- Distance:- Distance to next action point.
- Turn_direction: Direction of turn (left, right, straight).
- Intersection_name: Stores the name of intersection.
- Exit number:- Specifies the roundabout exit.

b) Input statement:

- Read current_location (initially set to school).

- Read destination (set to Home)
- Fetch road data, including intersections, distances, and turn directions.

c) Calculation:

- Determine next road, distance, and required turns.
- Update current-location after each movement
- Check if current-location == destination.

d) Output statements

- Display driving instructions (e.g "Move forward for X metres", "Turn left").
- Confirm arrival at destination.

e) Assignments:

- Assign values to current-location, destination, road-name, distance, and turn-direction dynamically.
- Update current-location as the vehicle progresses.

QUESTION#02

Design an algorithm for computing $\sqrt{n}$ for any positive integer n. Besides assignment & comparison, your algorithm may only use the four basic arithmetical operations.

SOLUTION

Algorithm: Compute $\sqrt{n}$ using basic arithmetic

This algorithm only uses addition (+), subtraction (-), multiplication (*), and division (//)

Pseudocode:

Input: A positive number n
Output: Approximate square root of n.

1) Set x to n (starting guess)
2) Repeat until the guess does not change
a) Set new-x to (x+(n/x))/2 (improve guess)
b) If new-x is the same as x, return x.
c) Set x to new-x.
3) Return x.
4) End Algorithm.

Explanation:

Only allowed operations; i.e., (+), (-), (*), (//) are used.

No other operations (e.g: exponentiation, logarithms, square root functions).

Assignment ? Comparison only:

* x = new - x ensures iteration stops when value stops changing.
* NO unnecessary steps added.

Question # 03

Design algorithm to find common elements in 2 sorted lists.
2, 5, 5, 5 & 2, 2, 3, 5, 5, 7
output should 2, 5, 5.
What max no. of comp algo should made if lengths are m, n. Specify.
a) Variables b) Input statement
c) Calculation d) Output statement e) Assignment

SOLUTION

Algorithm :- common elements (list1, list2)
Input :- lists sorted list1, list2.
Output :- list of common elements.

1) set i to 0 (for list 1)
2) set j to 0 (for list 2)
3) create an empty list common_elements
4) When i < length of list1 and j < length of list 2:
a) if list1[i] is equal to list2[j]:
i) Add list1[i] to common_elements.
ii) Increase i and j by 1

b) If list 1[i] is smaller than list 2[j]:
   i) Increase i by 1
c) If list 1[i] is larger than list 2[j]:
   i) Increase j by 1.
5) Return common_elements.
6) End.

Maximum Number of comparisons:-

In worst case, the algorithm compares each elements in both lists at most once. This happens when we must check every element in both lists, which takes a maximum of m+n comparisons. where m and n are lengths of two lists.

Facts of algorithm:

a) Variables:

i - Point to track current elements of list 1.
j - Point to track current elements of list 2.
Common_elements: A list to store the common found in both lists.

b) Input Statement:-

- two sorted lists, list 1 and list 2 of lengths m and n.

c) Calculations:

The main operation is comparing the current elements in both lists (l1[i] & l2[j]). Based on comparison, the pointers are moved either forward in both lists (if elements are equal), or in one list (if elements are unequal).

d) Output statements:-

The algorithms return common - elements the list of common elements found in both list 1 and list 2.

e) Assignments:-

- i ← 0: Set initial position for list 1.
- j ← 0 : set initial position for list 2
- common - elements ← empty list: Initialize result list for storing common elements.
- Increment operations on i and j as we traverse the list.