Problem Solving

Monday, September 16, 2024 9:44 AM

Brute Force / Heuristic Approach - Trial and error

- All permutations and combinations to get desire result
- Straight Forward approach.
- For ex. Lock system on suitcase.
- Time consuming

What is Heuristics -

Guess and Check
Work Backwards
- Bottom to up approach.

Make a Systematic List
Orderly arrangement of data.

To find out the largest element.
No. of passes

2,8,3, 50,60,90,14,56

2 and 8 → 8.
8 and 3 → 8
8 and 50 → 50
50 and 60 → 60
60 and 90 → 90
90 and 14 → 90
90 and 56 → 90

Greedy Approach

- Optimization Problems.
- Make optimal choices at each step with hope of finding an optimum solution.

Traveling Salesman Problem
Discover the shortest route for visiting a group of cities

There are 5040 possible routes between these 8 cities. Choosing a route requires a strategy.

2 Opt Swap

Select two edges and reconnect them, producing a new route.

Local Search
Find a better soln by applying a single change to a known solution

Local minimum.

Soln not optimum

Simulated Annealing - Probabilistically accept worse solutions early in the search.

Dynamic Programming-Example
You have three jugs, which we will call A, B, and C. Jug A can hold exactly 8 cups of water, B can hold exactly 5 cups, and C can hold exactly 3 cups. A is filled to capacity with 8 cups of water. B and C are empty. We want you to find a way of dividing the contents of A equally between A and B so that both have 4 cups. You are allowed to pour water from jug to jug
has context menu

-

Initially:

- A = 8 cups (full)
- B = 0 cups (empty)
- C = 0 cups (empty)

Dynamic programming

Algorithms

- Sequential
- Selectional
- Iterational

Data Structures

- Linear

Array - Fixed, Contiguous memory, same data type.
Linked List - Can insert anywhere, Dynamic nature, Different data types of values you can store.

- Non Linear

Algorithm:

1. **Push Operation**:
    - Input: An element to be pushed onto the stack.
    - Action: Add the element to the top of the stack.
    - Output: The stack with the new element on top.
2. **Pop Operation**:
    - Input: None.
    - Action: Remove and return the top element from the stack.
    - Output: The popped element and the stack after the removal.
3. **Peek Operation**:
    - Input: None.
    - Action: Return the top element of the stack without removing it.
    - Output: The top element of the stack.
4. **isEmpty Operation**:
    - Input: None.
    - Action: Check if the stack is empty.

    - Output: Return true if the stack is empty, otherwise false.
5. **Size Operation**:
    - Input: None.
    - Action: Return the current size (number of elements) in the stack.
    - Output: The size of the stack.
6. **Display Operation**:
    - Input: None.
    - Action: Display all elements of the stack from the bottom to the top.
    - Output: The list of elements in the stack.

Bubble Sort -

Adjacent elements compare - largest element push at the end(bubble up).

- Create an array to store the marks of 10 students.
- Take input from the user for each student's marks.

- Compare the first and second elements:
    - If the first is larger, swap them.
- Move to the next pair of elements and repeat step 2.
- Continue comparing adjacent elements until the end of the array is reached.
- After the first pass, the largest element will be placed at the end.
- Ignore the last element in the next pass (as it is sorted) and repeat the process for the remaining elements.
- Repeat this process for a total of 9 passes (n-1 passes for n elements).
- After all passes, the array will be sorted in ascending order.

- Once the array is sorted, print the elements in ascending order.

Selection Sort -

- Create an array to store the marks of 10 students.
- Take input from the user for each student's marks.
- Start with an array of 10 marks.
- Assume the first element is the smallest.
- Compare this element with the remaining elements in the array.
    - If you find a smaller element, update the smallest element.
- After completing the comparisons, swap the smallest element found with the first element.
- Move to the second element and repeat the process for the remaining unsorted elements.
- Repeat the process for all positions until the second-last element (since the last element will already be sorted).
- After completing all the steps, the array will be sorted in ascending order.
- **Output the Sorted Marks**:
- Once the array is sorted, print the elements in ascending order.