Mohammad Hassaan Ejaz

22i - 2434
SE - B

Question 1:

Approach and Data Structures Used:

1. Heap Data Structure:

A **d-ary heap** is a tree-based data structure where each node can have up to `d` children..

2. Key Operations:

- **Insert Operation (`ins`)**:
    - After insertion at the leaf, the element "floats up" to maintain the heap property.
- **Extract Operation (`extract`)**:
    - Removes and returns root (either min or max).
    - The last element in the heap replaces the root, and then the heap is maintained by pushing the element down to satisfy heap order.
- **Heapify**:
    - Uses heapifyUp or heapifyDown methods to maitnai heap structure.

3. Implementation Details:

- The heap is implemented as an array where:
    - Root is first and has up to `d` children and similarly to each node.

    - The index of a node's parent is calculated by the formula `(i-1)/d` where `i` is the current node's index.

    - The children of a node at index `i` are located at indices `d*i + 1, d*i + 2, ..., d*i + d`.

- **Float Up**: when a new element is added,. The element goes up until heap property is satisfied (min or max)
- **Sink Down**: The element at the root is replaced by the last element in the heap, and the new root goes down till heap is maitnainerd.

- **Heap Type Change**: Supports both min-heap and max-heap based on input

---

Code Explanation:

- The `HeapX` class implements the d-ary heap. It is generic and works with any data type `T` (although integers are preferred).

- `add()` inserts an element into the heap, `popTop()` removes and returns the root, and `show()` prints the heap in a level-wise format.

- The `flipType()` method allows switching between min-heap and max-heap

Question 2:

Finite Automaton:

- **Wildcard (`?`)**: The `?` wildcard matches any single character. This enhancement allows patterns like `a?ple` to match both "apple" and "ample".

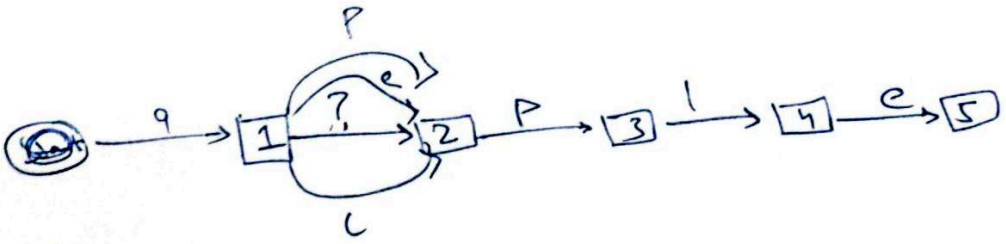
---

2. Transition Table and Diagram

QUESTION 2

Pattern : a?ple

	Input	Next	Prefix	Suffix
0	a	1	""	?ple
1	?	2	"a"	ple
2	p	3	"ap"	le
3	l	4	"app"	e
4	e	5	"a?ple"	""
5	""	Finish	"a?ple"	

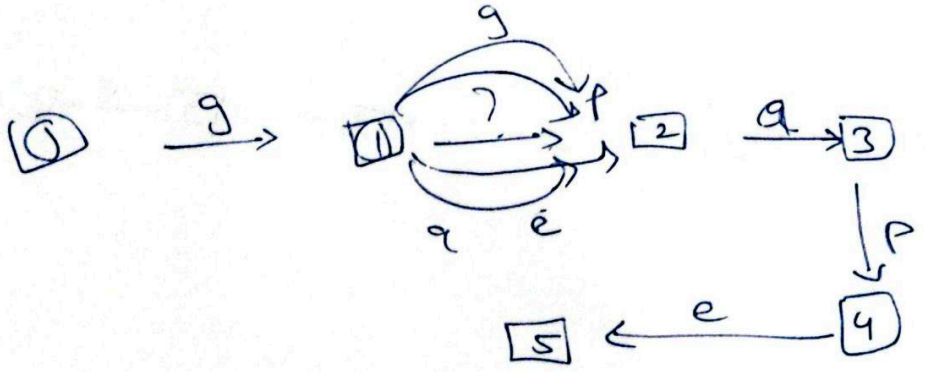


Q 2)

Hassan
22T-2434

Pattern g?ape

State	Input	Next State	Prefix	Suffix
0	g	1	" "	ape
1	?	2	g	ape
2	a	3	g?	pe
3	p	4	g?a	e
4	e	5	g?ap	" "
5	" "		g?ape	



---

4. Time Complexity Analysis

- **Preprocessing (Building Automaton)**: `O(m * n)` where `m` is pattern length and `n` is the number of characters (256 ASCII).
- **Search Time**: `O(L * m)` where `L` is the length of the text and `m` is the pattern length.

---

5. Code Explanation

- **FAEngine Class**: Builds the automaton's transition table and searches the text.
- **Construct Method**: Creates the table based on the pattern.
- **Scan Method**: Scans the text and finds matches using the automaton.
- **Lowercase Conversion**: Ensures case-insensitive matching.

Conclusion

The enhanced FA algorithm efficiently handles wildcard characters and matches multiple patterns across text with optimal time complexity.

Question 3:

Top K Gardens:

Steps

1. **Max-Heap** creation — this takes **O(n)** time.

2. **Pop top k** gardens from heap — each pop is **O(log n)**, so total is **O(k log n)**.
3. Return the registration IDs from popped gardens.

Time Complexity
O(n + k log n)

Problem B: Gardens Above a Score x (Max-Heap)

What's the Task?
Now the gardens are in a **Max-Heap**. We need to return all gardens with a score **strictly greater than x**.

How to Do It?

- Start at the top (index 0).
- If the current score > x:
    - Save its ID.
    - Recursively check the left and right children.
- If score ≤ x: ❌ Skip the whole subtree.

Pseudocode

```
PROCEDURE gardensAboveX

BEGIN

INPUT:

; heap is a structure array/list in which each has a current score
data member

       List[1…n] heap

       Number x, index

OUTPUT: number of gardens above a particular garden
```

```
    IF index >= n THEN
        return List [1 .. n] ; empty list
    ENDIF
    current INTEGER
    SET current = heap[index]
    IF current.score <= x THEN
        return List[1 .. n]  ; empty list i.e. skipping this branch
    ENDIF
      SET result = [current.registration_id]
      left Integer, right Integer
      SET left = 2 * index + 1
      SET right = 2 * index + 2

      SET result = result + gardensAboveX(heap, x, left) +
      gardensAboveX(heap, x, right)
      OUTPUT result
END
```

Time Complexity

$O(n_x)$ — where n_x is the number of gardens with score > x.

Quick Notes

Useful Tricks

- **Problem A:** Heap is fast for top-k.
- **Problem B:** Heap helps you skip bad branches quickly.

Edge Cases

- If `k > total gardens`: just return all.
- If all scores ≤ x: return empty.
- Empty list? Still works — just return nothing