Assignment-3

Section-A

Ques1. Define the terms 'in-degree' and 'out degree' in context of directed graphs. How are these measures used in practical applications?

Ans) For a directed graph: $G=(V,E)$:

→ The in-degree of vertex v (denoted $deg^-(v)$) is the number of edges coming into v (i.e., edges of form (u,v)).

→ The out-degree of vertex v (denoted $deg^+(v)$) is the no. of edges going out of v (i.e., edges of form (v,w)).

- Practical Use:-

→ Web graphs: In-degree = no. of incoming links, out-degree for link Analysis.

→ Network flow / routing: degrees are used to detect sources & balance f

→ Social Networks: in-degree = no. of followers, out-degree = no. of people one follo

Ques2. If a graph has 10 vertices and a total of 15 edges, what is the minimum number of edges that must be removed to obtain a spanning tree?

Ans) A spanning tree on n-vertices must have exactly n-1 edges. Here, n=10, so it will need 9 edges. Graph currently has 15 edges you must remove atleast 15-9=6 edges.

∴ Minimum edges must be removed = 6 edges.

Ques3. Given text length n=100 and pattern length m=5, how many character comparisons in worst case for brute-force string matching to find all occurrences?

Ans) Worst case brute-force compares each alignment fully or

nearly fully. Number of alignments = n-m+1 = 100+5+1 = 96.
In the worst case each alignment takes up to m=5 comparisons, so total worst-case comparisons:-

$96 \times 5 = 480$

∴ 480 character comparisons (worst case).

Ques 4. Hash value of pattern 'ABCD' using simple hash where, A→1, B→2, C→3, D→4.

Ans Depends on hash function definition. Two common choices:

- Sum hash (very simple): 1+2+3+4 = 10.
- Positional (Polynomial) hash eg base b (common in Rabin-Karp): h = 1*b^3 + 2*b^2 + 3*b + 4 (Choose b like 101 or 256).

⇒ 10.

Ques 5. What is the primary objective of Knuth Morris-Pratt (KMP) algorithm, and how does it achieve this?

Ans Objective: To perform exact string matching in linear time O(n+m) by avoiding re-examination of text characters after mismatches.

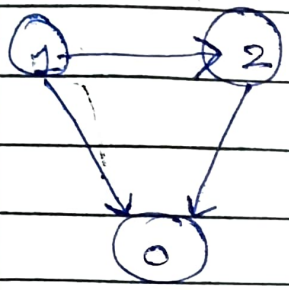
→ How it achieve this:- It avoids redundant character comparisons by precomputing a longest proper prefix that is also a suffix (LPS) array for pattern. This array is used to determine the optimal shift of the pattern after a mismatch, avoiding back-tracking in the text.

Section B

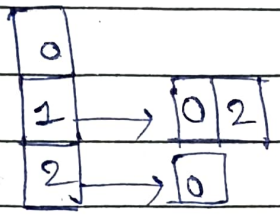
Ques Describe the concept of an adjacency list of graph representation, when it is used, advantages, disadvantages and an example scenario, it's particularly benefits?

Ans: An adjacency list represent a graph by storing, for each vertex, a list (or linked list/array/vector) of vertex adjacent to it (i.e, its neighbours) for directed graphs the list contains outgoing neighbours.

⇒ Consider, an directed and unweighted graph with 3 vertices and 3 edges.



Directed graph



Adjacency list.

U	V
1	0
1	2
2	0

Edges

Graph Representation

⟶ When it is used?

- These are most beneficial for sparse graphs, which are graphs with relatively few edges compared the number of vertices. A graph is considered sparse when the number of edges E is significantly less than maximum possible number of edges, which is $O(V^2)$.

Aspect	Advantages	Disadvantages
Space	Memory efficient. Use less memory for sparse graph, with a space complexity of O (V+E) as it only stores existing edges.	Slower edge lookup checking for existence of edge b/w two specific vertices can be slow requiring linear scan of vertex adjacency list.

Graph Type	Flexible — Works for both directed & undirected graphs as well as weighted graph.	Not ideal for dense graphs (graphs with many edges, as adjacency matrix can be more space efficient and faster edge lookup)
Traversal	Efficient Traversal + finding all neighbours of a vertex is very fast, making it ideal for algorithm like BFS & DFS.	More complex implementation, Slightly more complex than adjacency matrix, though matrix libraries and DS make it simpler.

→ Example Scenario:- Social Network

A social network like Facebook or LinkedIn are are excellent example where an adjacency list is beneficial

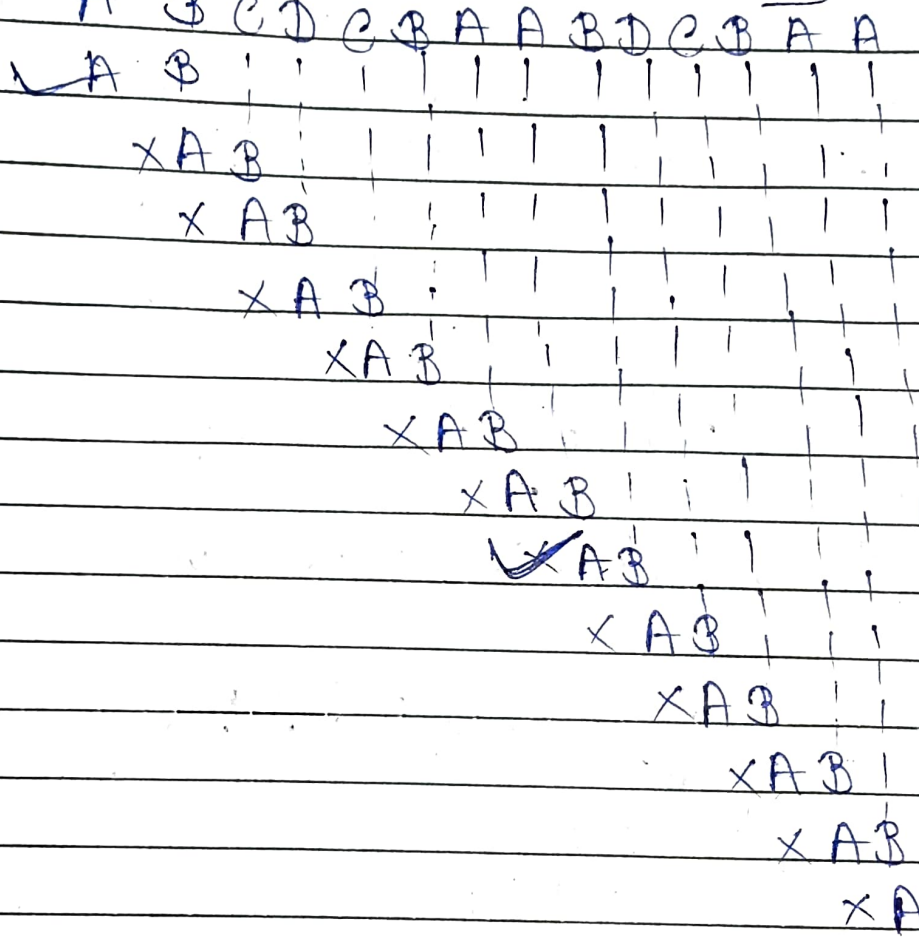
- Scenario — A social network has millions of users (vertices) but each user is connected to a small number of other users (edges) making graph very sparse.
- Benefits:
  - Memory efficiency.
  - Efficient operations.
  - Friend suggestions.

Ques. Given a text string "ABCDCBAABDCBAA" and a pattern 'AB', find and list all occurances of pattern using naive string machine Algorithm. Provide position of occurances within the text.

Ans: To find all occurances of the pattern 'AB', we will compare the pattern with each character/substring of text. Search will be performed using 0-based indexing

- Text (T) = ABCDCBAABDCBAA (Length = 14)
- Pattern (P) = AB (Length = 2)

Step-by-step naive string matching:-
We check every position from 1 to (n − m + 1) = 13



Here, Pattern "AB" occur at position : 1 and 9 (1-based index)

Ques 3. Explain the concept of Rabin-Karp String matching Algo. and its primary use. Given a text "ABCDABCDABCD" and a pattern "ABC", calculate the hash value of pattern and determine the position of pattern occurances using Rabin-Karp Algo.

Ans Concept:-
The Rabin-Karp Algo is a string matching method that uses hashing to efficiently find all occurences of a pattern in text.

→ Instead of comparing entire pattern each time, it compute a hash values for pattern and compare it with hash value at text substring of same length. If hash match, it then verifies the substring to confirm match.

⇒ Primary Use :- Efficient for multiple pattern searching and large texts, particularly in plagiarism detection, digital forensics & search engine.

⇒ Given text = ABCDABCD ABCD

Pattern = ABC

Assign Value A=1, B=2, C=3, D=4.

Pattern Hash P(H) = 1+2+3 = 6

A B C D A B C D A B C D

Pos-1 → (6) A B C

X (4) B C D

X (8) C D A

X (7) D A B

Position-5 → (6) A B C

X (9) B C D

(8) C D A

(7) D A B

Position-9 → (6) A B C

X (9) B C D

→ Here, Pattern "ABC" found at position 1, 5 & 9.

Section-C

Ques 1. (a). Explain the key principle of the finite automata string matching algo, include its main component and discuss strength & limitations of finite automata for string matching compared to other algorithm.

(b). Using the provided finite automation, find all occurance of the pattern "ABAB" in the text "ABABABABABABA" or "BABBA" and show the state transition. Describe how finite automation process text to detect pattern.

Ans (a). Core idea:- Build a deterministic finite Automation (DFA) that reads the text one character at a time and tracks how many characters of the pattern have been matched so far.

→ When the DFA reaches accepting state an occurence is found.

⇒ Main Component

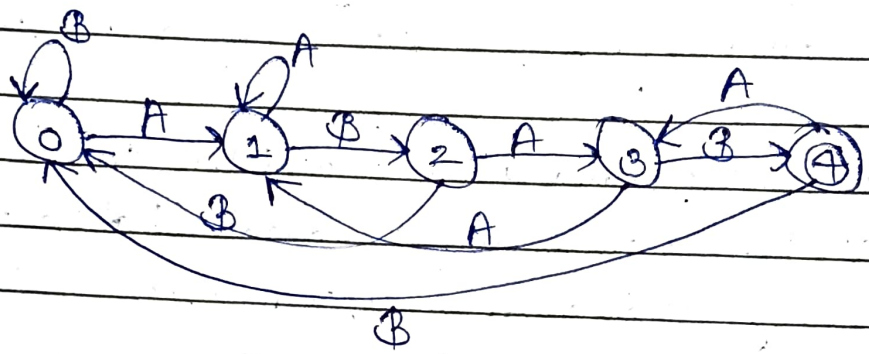
Q: Set of State.
Σ: Set of input.
q: ~~Internal state~~ Initial State.
F: Final State.
δ: Transition State.

⇒ Strength:
- Deterministic O(n) matching time.
- No back-tracking at match time.
- Good when you will search same pattern many times.

⇒ Limitation:

Pre-processing cost & Space.
Less space - Efficient than KMP for long alphabets
In practice, for large Σ or color memory is limited
KMP or rolling-hash methods are often preferred.

(b). Pattern = "ABAB"
Text = "ABABABABBABABABA"



	A	B
0	1	0
1	1	2
2	3	0
3	1	4
4	3	0

→ So, the pattern "ABAB" occur at ending position 4, 6, 10, 12 in given text.

⇒ Detected Occurances :→
Occurances ending at
index 4 → start = 1
index 6 → start = 3
index 8 → start = 5
index 10 → start = 7
index 12 → start = 9

Ques. (a). Explain the key principles of the KMP String matching Algorithm, including its pre-processing phase and pattern matching phase. Provide a step-by-step explanation of how Algorithm works & its time complexity.

(b). Perform the preprocessing step for the pattern "ABAB" and calculate partial match table. The use the calculate table to apply the KMP Algorithm and find all occurances of pattern "ABAB" in the text "ABABABBABABABA". Show the position and state transition for each occurance.

Ans (a). Core Idea:- KMP avoids re-checking character of the text that were already matched against pattern by pre-computing where to resume the pattern after a mismatch.

⇒ Pre-processing phase.
— Input pattern.
— Compute lps[i] = length of longest proper prefix of P[0 - - - i].
— Use two indices: length of previous longest prefix-suffix and i (current position).

⇒ Pattern-matching phase:
— Scan text T[0 - - - (n-1)] with index i and maintain j = number of position of pattern character matched so far.
— For each T[i]:
- If P[j] = T[i] → increase both i & j.
- If j == m → full match found, ending at i-1, report i-j. Then set j = lps[j-1], to look for next.

- On mismatch → set $j = lps[j-1]$ and retry if $j > 0$ else mismatch → increment i.

⇒ Time Complexity:-

Preprocessing: $O(m)$

Matching: $O(n)$

Total: $O(n+m)$ in Worst case.

Space: $O(m)$ for LPS table.

⇒ Given Pattern (P) = "ABAB"

Text (T) = "ABABABABABABA"

Step:1. Pre-processing Step — Construct LPS table:-

Index	Pattern char	Substring	Longest Prefix Suffix	LPS[i]
0	A	A	—	0
1	B	AB	None	0
2	A	ABA	A	1
3	B	ABAB	AB	2

∴ final LPS table:- LPS = [0, 0, 1, 2]

Step:-2 Apply KMP on the text.

We'll use two pointers:-

i → text index, j → pattern index.

Initial: $i=0, j=0$.

⇒ Step by step matching process.

Step	Text[i]	Pattern[j]	Match?	Action/Msg	i	Remark
1	A	A	✓	j=1	i=1	–
2	B	B	✓	j=2	i=2	–
3	A	A	✓	j=3	i=3	–
4	B	B	✓ full match	Report position= i-m+1=0	j=LPS[3]=2	i=4
5	A	A	✓	j=3	i=5	–
6	B	B	✓ full match	Report position= i-m+1=2	j=2	i=6
7	A	A	✓	j=3	i=7	–
8	B	B	✓ full match	Position=i-m+1 =4 Report	j=2	i=9
9	B	A	✗	j=LPS[1]=0	i stays 8	–
10	A	A	✗	i=9	–	–
11	B	A	✓	j=1	i=10	–
12	A	B	✓	j=2	i=11	–
13	B	A	✓	j=3	i=12	–
14	A	B	✓ full match	Position=i-m+1 =9	j=2	i=13
15	B	A	✓	j=3	i=11	–
16	A	B	✓ full match	Position=i-m+1 =11	j=2	i=15

Overall time Complexity: $O(n+m)$ = linear time.

Ques 3. Explain key principles of BFS and DFS in graph traversal. Discuss fundamental difference b/w these two Algo and provide a real world scenario where choosing one Algo over other is more advantageous.

(b). Explain key priciples of Boyer-Moore string matching Algo including its component? Advantages compared to naive string matching. Given text "ABCDABCDABCD" and pattern "ABC". Calculate no. of char comparison made by Boyer-Moore when finding the pattern.

Ans:- Key Principles:

1). BFS (Breadth First Search)

→ BFS explores a graph level by level starting from a source vertex.

→ It visits all the neighbours of a vertex before moving to next level of vertices.

→ It uses a queue (FIFO) data structure to manage order of vertex exploration.

→ It guarantees finding shortest path in unweighted graphs.

2). DFS (Depth First Search).

→ DFS explores as deep as possible along each branch before backtracking.

→ It uses a stack (LIFO) structure — either implemented explicitly or through recursion.

→ DFS is useful for tasks like cycle detection, topological sorting and connected component identification.

BFS	DFS
• Level-by-level	• Depth-wise.
• Queue.	• Stack / Recursion.
• Finds shortest Path.	• May not find shortest path.
• Higher.	• Lower.
• Shortest path, networking, social distance.	• Maze solving, dependency analysis, Game Trees.

Real World Scenario:

→ In social Networking, BFS is used to find shortest path/connection between two users.

→ In Web crawlers, DFS is preferred.

(b). Boyer–Moore String Matching Algo:-

→ Boyer–Moore Algo improve on naive string matching by comparing pattern with the text from right to left and skipping large portion of text when mismatches occur.

→ It uses two heuristics.

1). Bad Character Rule:-

→ When a mismatch occur, the pattern is shifted so that mismatched character in the text aligns with its rightmost occurence in the pattern.

2). Good Suffix Rule:- Uses knowledge of matched suffix to decide how far to shift the pattern to avoid redundant checks.

→ Advantages:-

- Skips unnecessary comparisons.
- Very fast in practice.
- Efficient for long text and large alphabets.

→ Example:-

Text: ABCDABCDABCD

Pattern: ABC (length $n=3$ & text length $n=12$)

→ For pattern "ABC" Last Occurrence Index.

Character:	Index
A	0
B	1
C	2

→ Step by Step matching:-

1) Initial Alignment (text [1-3])

C vs C

B vs B

A vs A → Pattern found at position 1.

2) Next Alignment (text [4-6])

C vs B → mismatch

Shift to 1.

3) Next Alignment (text [5-7])

C vs C

B vs B

A vs A → Pattern found at position 5.

4) Next (text [6-8])

B vs C

Shift to 1.

5) Next (text [9-11])

C vs C

B vs B

A vs A → Match found.

⇒ Character Comparison

1 - 3 → 3

4 - 6 → 1

5 - 7 → 3

8 - 10 → 1

9 - 11 → 3

11 → Total Comparisons.