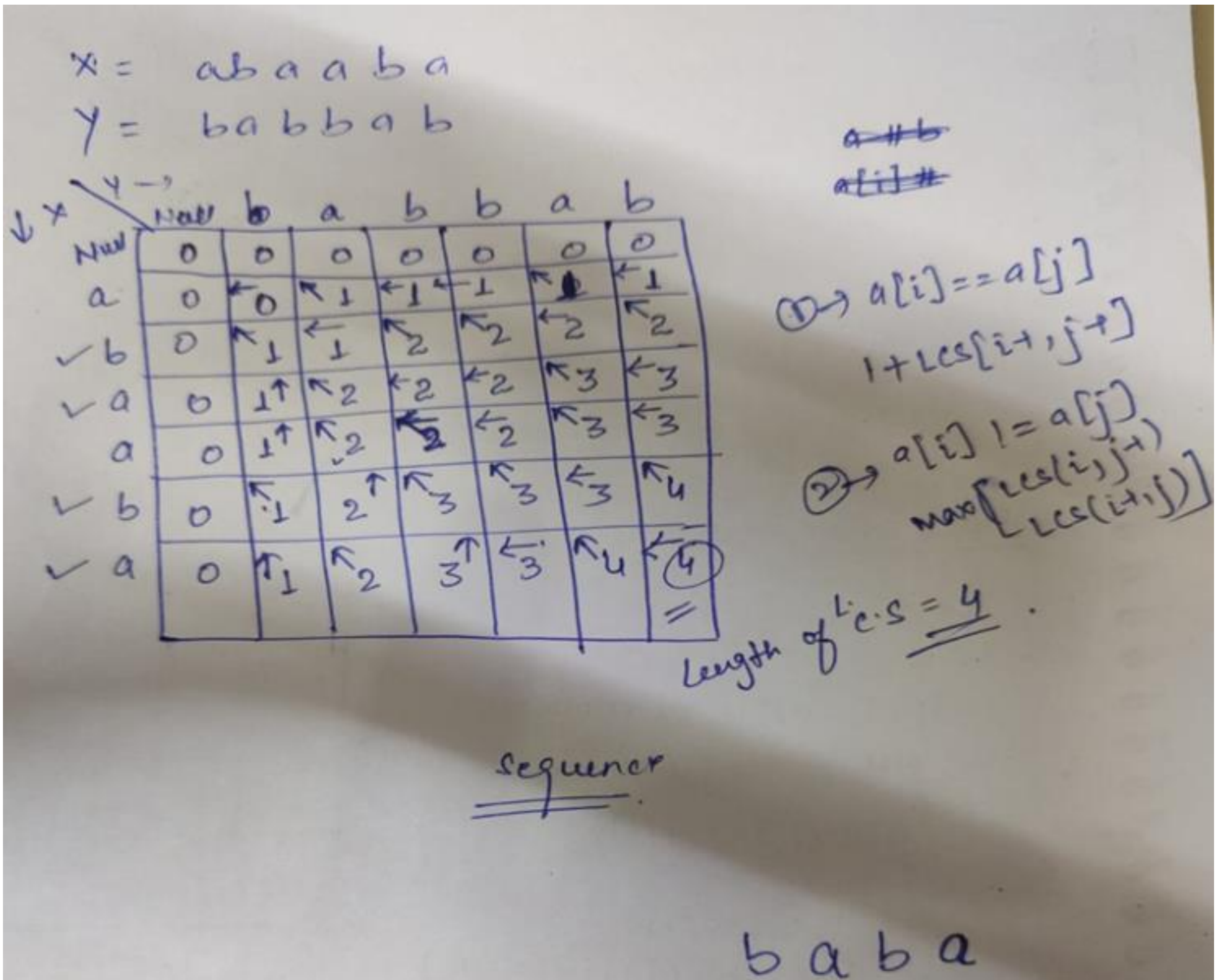
1. **[CO1] [Marks 4] Form a recurrence relation for the following code and solve it using Master's theorem:**

```
a. A(n){
        if (n<=1)
            return log n;
        else
            return √2 A(n/2);}
```

Solution:
$A(n) = \sqrt{2}A(n/2) + \log n$
$A(n) = \theta(\sqrt{n})$

2. **[CO3] [Marks 5] Given two strings str1="ABAABA" and str2="BABBAB". Find the maximum length of subsequence as well as the resultant subsequence which is present in both the strings. Strictly use memorization technique to solve the question.**



X = abaaba
Y = babbab

↓X \ Y→	Null	b	a	b	b	a	b
Null	0	0	0	0	0	0	0
a	0	←0	↖1	←1	←1	↖[illegible]	←1
✓ b	0	↖1	←1	↖2	↖2	←2	↖2
✓ a	0	1↑	↖2	←2	←2	↖3	←3
a	0	1↑	↖2	←2	←2	↖3	←3
✓ b	0	↖1	2↑	↖3	↖3	←3	↖4
✓ a	0	↑1	↖2	3↑	←3	↖4	←$\boxed{4}$

~~a ≠ b~~
~~a[i] ≠~~

① → a[i] == a[j]
1 + LCS[i+1, j+1]

② → a[i] != a[j]
max[LCS(i, j+1), LCS(i+1, j)]

Length of L.C.S = 4.

Sequence

baba

3. **[CO4] [Marks 6] There are a set of computers N here at JIIT Noida, and a set of computers M at JUET Guna. A connection of the computers from N to M is there such that the path NM starts at a node in N and ends at a node in M. Formally, you are a given a graph G = (V, E) with a designated subset N ⊆ V and M ⊆V, where N and M are disjoint. The computers in JIIT and JUET have variable transfer capacity. Further the data transfer rate on**

NM path is just one. Give an efficient algorithm to find a maximum-capacity of data transfer from N to M. Give the running time of your algorithm, prove that your algorithm is correct, and show that it finds the maximum data transfer capacity of such paths.

Answer Add a new source edge s, connected to all the nodes in N with infinite capacity. Add a new destination edge d, connected to all the nodes in M with infinite capacity. Set all other edge capacities to 1, and find the maximum flow, e.g., using the Edmonds Karp algorithm.

Let k be the maximum flow, which is also equal to the minimum cut. Using Edmonds-Karp, this takes time at most O(n2m). Next, find a flow decomposition of the graph. Let P1, P2, . . .,Pj be the paths constructed by the flows, with the nodes s and t removed from each path. Notice that each of these paths is an NM-path, since all the outgoing edges from s lead to N and all the incoming edges to t come from M. Also notice that constructing this decomposition takes at most O(nm) time: each iteration of the flow decomposition algorithm takes time O(m), and there are at most O(n) iterations, hence the total decomposition takes time O(nm).

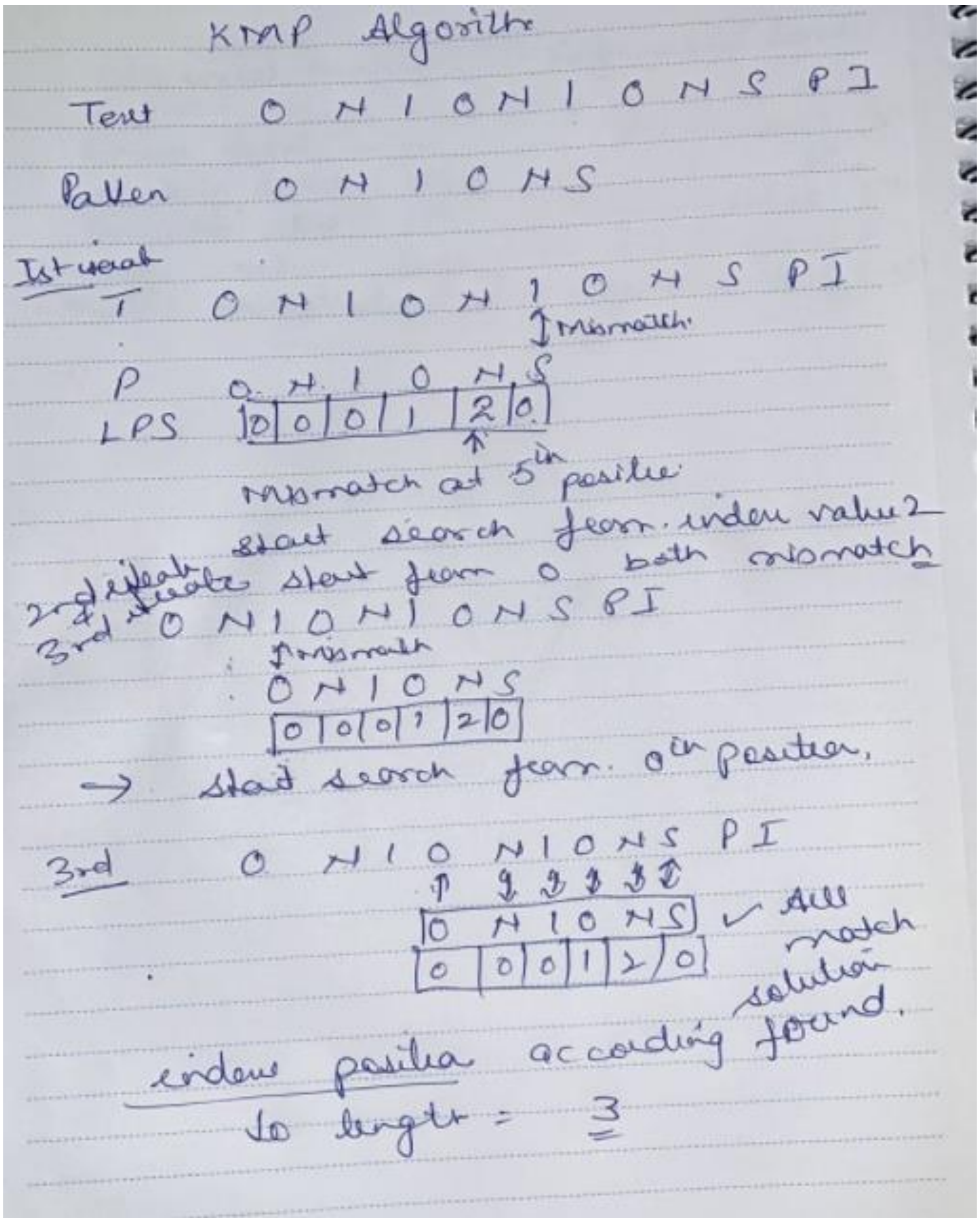
We now argue that these paths are edge disjoint. Assume that two paths Pi and p` share an edge. Recall that all flows constructed by the decomposition are integral, hence each of these paths has flow at least 1. Thus, the shared edge must have at least two units of flow, which exceeds its capacity (since all edges between N and M have capacity 1). Hence, we conclude that the paths must be edge disjoint. Finally, we argue that the number of paths is maximum. Since each path has value at most 1 (since it contains at least one edge of capacity 1), by the flow decomposition, there must be k paths (in order to reach a total of k units of flow). However, we know that the min-cut of the network is k, i.e., every path must cross one of the k edges in the min-cut between the source s and the target t. (Moreover, the cut cannot cross the infinite capacity edges.) Since each path is edge disjoint, we conclude that there are at most k edge disjoint NM-paths in the graph G. Thus, we have found the maximum number of edge disjoint paths in the graph.

(Note that an ideal solution would likely include a picture.)

4. **[CO3] [Marks 5] Apply the cost effective algorithm to find the give pattern from the text.**

TEXT: ONIONIONSPI

PATTERN: ONIONS



KMP Algorithm

Text O N I O N I O N S P I

Patten O N I O N S

1st iterat

T O N I O N I O N S P I

↑ mismatch

P O N I O N S

LPS

0	0	0	1	2	0

↑

mismatch at 5th position

start search from index value 2

2nd iterate start from 0 both mismatch

3rd O N I O N I O N S P I

mismatch

O N I O N S

0	0	0	1	2	0

→ start search from 0th position.

3rd O N I O N I O N S P I

O	N	I	O	N	S
0	0	0	1	2	0

✓ All match solution found.

index position according to length = 3

5. **[CO4] [Marks 3]** ***"Satisfiability, 0/1 knapsack, Longest Path these all have something in common i.e. if any one can solved in polynomial time then all others can be solved as well*"**. Derive the relation between satisfiability and 0/1 knapsack algorithm to validate the above statement.**

Answer Its theoretical question we need to check if student has reduced 0/1 knapsack to satisfiability problem. Use your wisdom to evaluate this question.

6. **[CO3] [Marks 5] A string 'AAAAAA' is stored using a suffix array. A pattern 'AAAAA' is required to be searched in the string. Write all the intermediate steps for finding the offset at which pattern is found. Floor function can be used while calculating the value of mid.**

str1 = AAA.AAA

l_1 = strlen(str1) = 7

suffix Array SA = {6, 5, 4, 3, 2, 1, 0}

str2 = AAAAA

l_2 = strlen(str2) = 6

Sub String matching

low = 0 high = l_1 = 7

mid = $\left\lfloor \frac{0+7}{2} \right\rfloor = \lfloor 3.5 \rfloor = 3$

k = strncmp(str2, S1 + SA[mid]))

strncmp(str2, S1 + 3) > 0

= strncmp(str2,

⇒ low = mid+1 = 4 high = ~~[illegible]~~ high = 7

mid = $\frac{7+4}{2}$ = 11/2 = 5

k = strncmp(str2, S1 + SA[5]))

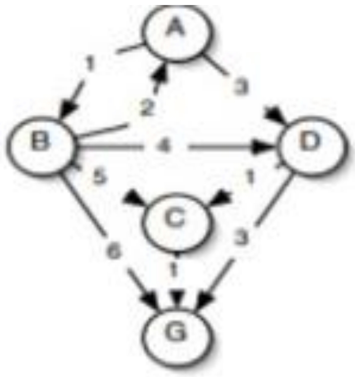
∴ strcmp(str2, S1 + 1) = 0

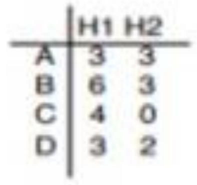
Pattern found at index = 1

or offset = 1

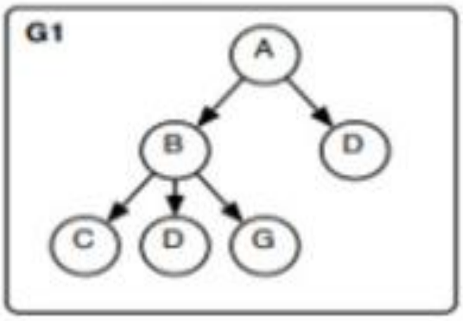
ie SA[5]

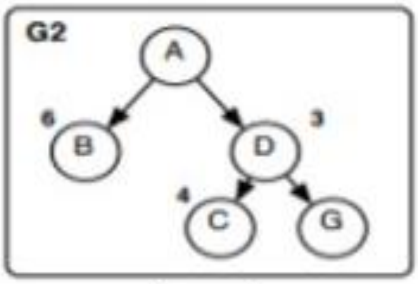
7. **[CO4] [Marks 7] Consider the graph shown in Figure. We can search it with a variety of different algorithms (Depth first search, Breadth first search, Uniform cost search, A* search, Best-first greedy search) resulting in different search trees. Each of the trees below was generated by searching this graph, but with a different algorithm. Name that algorithm for each tree (G1 to G7). Assume that children of a node are visited in alphabetical order. Each tree shows all the nodes that have been visited. Numbers next to nodes indicate the relevant "score" used by the algorithm for those nodes. In all cases a strict expanded list was used. Furthermore, if you choose an algorithm that uses a heuristic function, say whether we used H1 or H2.**

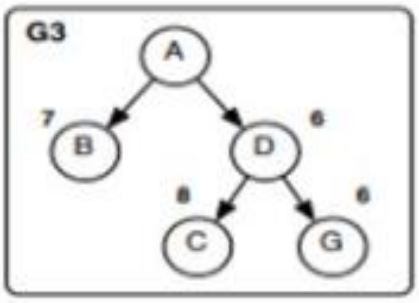


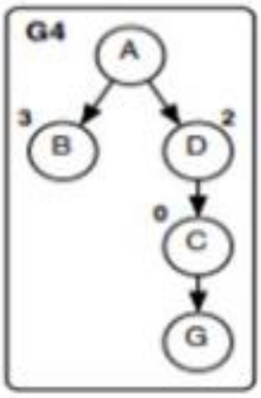


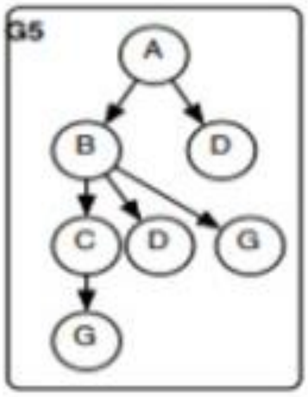
	H1	H2
A	3	3
B	6	3
C	4	0
D	3	2

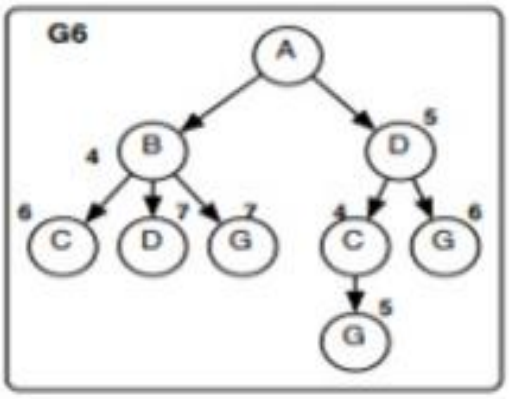


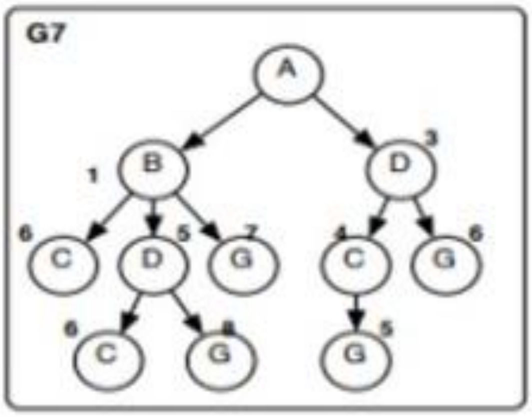












G1: 1. Algorithm: **Breadth First Search**

2. Heuristic (if any): **None**

3. Did it find least-cost path? If not, why? **No. Breadth first search is only guaranteed to find a path with the shortest number of links; it does not consider link cost at all.**

G2: 1. Algorithm: **Best First Search**

2. Heuristic (if any): **H1**

3. Did it find least-cost path? If not, why?

No. Best first search is not guaranteed to find an optimal path. It takes the first path to goal it finds.

G3: 1. Algorithm: **A***

2. Heuristic (if any): **H1**

3. Did it find least-cost path? If not, why? **No. A* is only guaranteed to find an optimal path when the heuristic is admissible (or consistent with a strict expanded list). H1 is neither: the heuristic value for C is not an underestimate of the optimal cost to goal.**

G4: 1. Algorithm: **Best First Search**

2. Heuristic (if any): **H2**

3. Did it find least-cost path? If not, why? **Yes. Though best first search is not guaranteed to find an optimal path, in this case it did.**

G5: 1. Algorithm: **Depth First Search**

2. Heuristic (if any): **None**

3. Did it find least-cost path? If not, why? **No. Depth first search is an any-path search; it does not consider link cost at all.**

G6: 1. Algorithm: **A***

2. Heuristic (if any): **H2**

3. Did it find least-cost path? If not, why? **Yes. A* is guaranteed to find an optimal path when the heuristic is admissible (or consistent with a strict expanded list). H2 is admissible but not consistent, since the link from D to C decreases the heuristic cost by 2, which is greater than the link cost of 1. Still, the optimal path was found.**

G7: 1. Algorithm: **Uniform Cost Search**

2. Heuristic (if any): **None**

3. Did it find least-cost path? If not, why? **Yes. Uniform Cost is guaranteed to find a shortest path.**