Lesson:

Problems on Dynamic Programming-3

PW SKILLS

Pre Requisites:

- Dynamic Programming

List of concepts involved :

- Maximal Rectangle
- Traveling salesman
- Regex Matching
- House robber III

Q1. Maximal Rectangle

Given a rows x cols binary matrix of character type elements, filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Input1:
rows = 4
cols = 5
matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

Output1: 6

Input2:
rows = 1
cols = 1
matrix = [["0"]]
Output2: 0

Approach

- Initialize the maxArea variable to 0 to store the maximum area found so far.
- Create a vector heights of size cols to track the heights of each column.
- Iterate through each row of the matrix.
- Update the heights array for the current row:
- If the current cell is '1', increment the height of the column by 1.
- If the current cell is '0', reset the height of the column to 0.
- Calculate the maximum area using the histogram approach:Use a stack to maintain a non-decreasing sequence of column heights.
- Iterate through each column and compare the current height with the height of the top column in the stack

- If the current height is less than or equal to the height of the top column, pop the top column from the stack and calculate the area using the popped column as the height.
- Update the maximum area if necessary.
- Push the current column index to the stack.
- Update the maximum area if the current area is larger than the previous maximum area.
- Return the maximum area found.

Code
https://pastebin.com/TnX7hnpw

Q2. Traveling salesman Problem

Consider a Traveling Salesman Problem (TSP) with 4 cities, where the distances between the cities are given by an adjacency matrix. Using dynamic programming with bitmasking, what is the length of the shortest path that visits all 4 cities and returns to the starting city (city 0)?

Input1:
City 0 1 2 3
---
0 | 0 10 15 20
1 | 10 0 35 25
2 | 15 35 0 30
3 | 20 25 30 0

Output1:
80

Input2:
City 0 1 2 3 4
---
0 | 0 10 15 20 25
1 | 10 0 35 45 30
2 | 15 35 0 50 40
3 | 20 45 50 0 55
4 | 25 30 40 55 0

Output2:
110

Solution:
Code : https://pastebin.com/ZpigF1Ca

Output :

```
The cost of most efficient tour : 80
```

Approach:
- Let the given set of vertices be {1, 2, 3, 4,....n}. Let us consider 1 as the starting and ending point of output.
- For every other vertex I (other than 1), we find the minimum cost path with 1 as the starting point, I as the ending point, and all vertices appearing exactly once.
- Let the cost of this path cost (i), and the cost of the corresponding Cycle would cost (i) + dist(i, 1) where dist(i, 1) is the distance from I to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far.

- To calculate the cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems.
- Let us define a term C(S, i) as the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i.
- We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.
  - **If size of S is 2, then S must be {1, i},**
  - **C(S, i) = dist(1, i)**
  - **Else if size of S is greater than 2.**
  - **C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.**
- Below is the dynamic programming solution for the problem using top down recursive+memoized approach:-
  - For maintaining the subsets we can use the bitmasks to represent the remaining nodes in our subset.
  - Since bits are faster to operate and there are only a few nodes in the graph, bitmasks are better to use.
  - For example: –
    - 10100 represents node 2 and node 4 are left in set to be processed
    - 010010 represents nodes 1 and 4 are left in the subset.
- NOTE:- ignore the 0th bit since our graph is 1-based

Q3. Regex Matching

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

- **'.' Matches any single character.**
- **'*' Matches zero or more of the preceding element.**

The matching should cover the entire input string (not partial).

Input1: s = "aa", p = "a"
Output1: false
Explanation: "a" does not match the entire string "aa".

Input2: s = "abbbac", p = ".*a*"
Output2: true
Explanation: The first character is a '.' character. The '.' Character means that it should be with any one character. It is matched with 'A' character.
The second character is a '*' character. The '*' character means that it should be matched with either 0 or more characters, so it is matched with 3 'B' characters from the string.
The third character is a 'A'. The 'A' character with an 'A' from the string S.
The fourth character is a '*' character, and it is matched with an 'C' character from the string.
Therefore, all the characters of the regular expression are matched with a String S. It returns a true value.

Explanation:

- We define dp[i][j] to be true if s[0..i) matches p[0..j) and false otherwise. The state equations will be:

1. dp[i][j] = dp[i - 1][j - 1], if p[j - 1] != '*' && (s[i - 1] == p[j - 1] || p[j - 1] == '.');
2. dp[i][j] = dp[i][j - 2], if p[j - 1] == '*' and the pattern repeats for 0 time;
3. dp[i][j] = dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'), if p[j - 1] == '*' and the pattern repeats for at least 1 time.

Code:
https://pastebin.com/BfeMxw5f

Q4. House Robber III

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called root. Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that all houses in this place form a binary tree. It will automatically contact the police if two directly-linked houses were broken into on the same night. Given the root of the binary tree, return the maximum amount of money the thief can rob without alerting the police.

Input1: root = [3,2,3,null,3,null,1]
Output1: 7
Explanation: Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.
Explanation:-

1. Recursive Function Definition (`rec`):
- The function `rec` is defined to take a `TreeNode*` (representing the current node of the binary tree) as input and returns a pair of integers.
- The pair of integers represents:
- The maximum amount of money that can be robbed from the subtree rooted at the current node, considering the current node.
- The maximum amount of money that can be robbed from the subtree rooted at the current node, without considering the current node.

2. Base Case Check:
- Inside the `rec` function, the first thing checked is whether the current node is `NULL` (which indicates an empty house). If it is, the function returns a pair of zeros since there's no money to steal from an empty house.

3. Recursive Calls and Calculation:
- If the current node is not `NULL`, the function proceeds with recursive calls.
- Two recursive calls are made: one for the left child of the current node and one for the right child of the current node. These calls recursively calculate the pair of values for the respective subtrees.

4. Calculating for the Current Node:
- After obtaining the pairs of values from the recursive calls, the function calculates two values for the current node:
- `ans_c`: This represents the maximum amount of money that can be robbed from the current subtree when considering robbing the current house. It's calculated by adding the value of the current node to the sums of the maximum robbery gains from the subtrees of the left and right children, but without including their values.
- `ans_n`: This represents the maximum amount of money that can be robbed from the current subtree when skipping the current house. It's calculated by summing up the maximum robbery gains from the subtrees of the left and right children, considering or not considering their values.

5. Returning the Results:
- The function returns a pair of values:
- The maximum of `ans_c` and `ans_n` (which indicates whether it's better to rob the current house or not) as the first element of the pair.
- The value of `ans_n` as the second element of the pair.

6. Main Function (`rob`):
- The `rob` function takes the root of the binary tree as input.

- Inside this function, the `rec` function is called on the root to get the pair of values representing the maximum robbery gains when considering and not considering the root.
- The maximum value between the two values in the pair obtained from the `rec` function is returned as the final answer – the maximum amount of money that can be robbed without alerting the police.

Code:
https://pastebin.com/KqHYfsPV

Upcoming Lectures:

- Graphs