COMPSCI 130B Discussion

02/18/2021

Kha-Dinh (Jacob) Luong

Office hours: F 9-11 AM

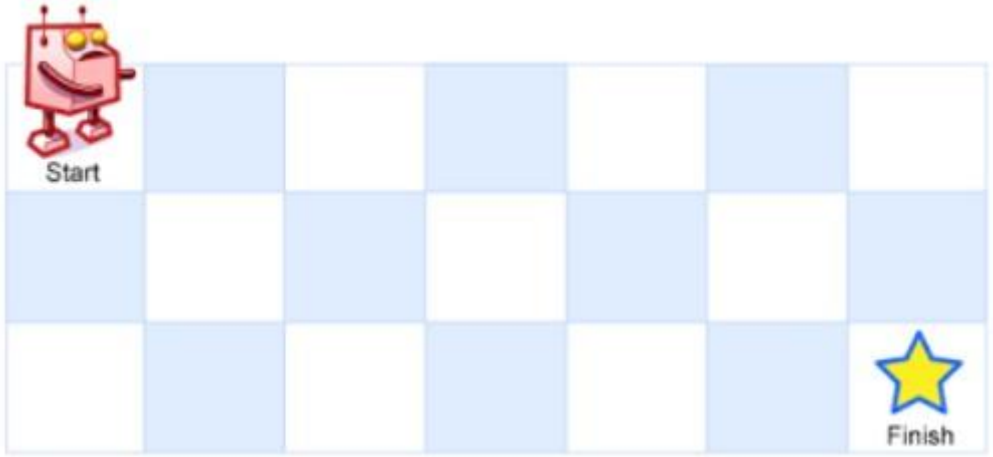
Objectives

- Practice Dynamic Programming.

Tips

- To find substructures and recurrence.
- Check and trivial cases and “versions” of the problem that can be solved right away. For example: OPT(1), OPT(0), OPT(2i), OPT(0,1), OPT(1,0),OPT(i,i).
  - This step may help you define OPT.
- Check all possible “paths” that may lead to the solution for OPT(i,j). For example: OPT(i-1,j), OPT(i,j-1), OPT(i-1,j-1), OPT(i-2,j-2).
  - This step may help you find the recurrence.

Unique Paths on Grid



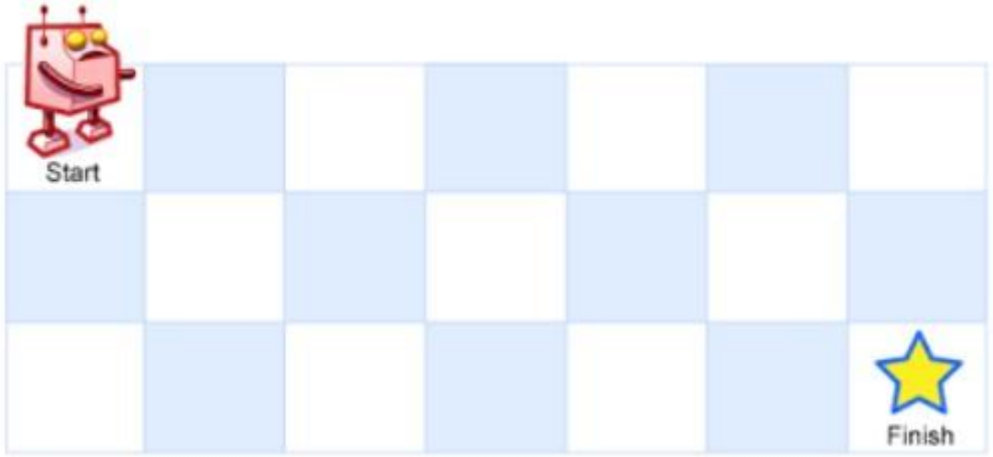
leetcode.com

- Given a m x n grid.
- Start: (0,0)
- End: (x,y)
- At any cell, only move Right or Down.
- Output: The number of unique paths from Start to End.

- What are the trivial cases? Substructure?

Unique Paths on Grid

- OPT(i,j): number of unique paths from Start to (i,j).
- OPT(0,0) = ?
- OPT(1,0) = OPT(0,1) = ?
- What are the possible paths that lead to OPT(i,j)? OPT(i,j) = ?



leetcode.com

Unique Paths on Grid

- OPT(i,j): number of unique paths from Start to (i,j).

- OPT(0,0) = 0
- OPT(1,0) = OPT(0,1) = 1

- OPT(i,j) = OPT(i,j-1) + OPT(i-1,j)

leetcode.com

Unique Paths on Grid

Top-down DP

```
func solution(i,j,dp) {
    if dp[i][j] > -1:
        return dp[i][j]
    else:
        dp[i][j] = solution(i-1,j,dp) + solution(i,j-1,dp)
        return dp[i][j]
}
 func main(m,n) {
     initialize a 2-D array of -1 called dp[m][n]
     dp[0][0] = 0
     dp[0][1] = 1
     dp[1][0] = 1
     return solution(m-1,n-1,dp)
}
```

Bottom-up DP

```
func main(m,n) {
    initialize a 2-D array of -1 called dp[m][n]
    dp[0][0] = 0
    dp[0][1] = 1
    dp[1][0] = 1
    for i in [1,m-1] {
        for j in [1,n-1] {
            dp[i,j] = dp[i-1,j] + dp[i,j-1]
        }
    }
    return DP[m-1][n-1]
}
```

House Robber

- Given a list of positive integers representing the amount of money in each house along a neighborhood. Determine the maximum amount of money a robber can steal from the neighborhood. The robber cannot steal from 2 consecutive houses or the alarm will go off.
- Example: arr = [2,7,9,3,1]
- Output: 12 = 2+9+1

- What are the trivial cases? Substructure?

House Robber

- OPT(i): maximum amount of money stolen from arr[0:i]

- OPT(0) = ?
- OPT(1) = ?

- OPT(i) = ?

House Robber

- OPT(i): maximum amount of money steal from arr[0:i]

- OPT(0) = arr[0]
- OPT(1) = max(arr[0],arr[1])

- OPT(i) = max(OPT(i-2) + arr[i], OPT(i-1))

House Robber

Top-down DP

```
func solution(i,arr,dp) {
    if dp[i] != null:
        return dp[i]
    else:
        dp[i] = max(solution(i-2,arr,dp) + arr[i], solution(i-1,arr,dp))
        return dp[i]
}
 func main(arr) {
     initialize a 1-D array of null called dp[size(arr)]
     dp[0] = arr[0]
     dp[1] = max(arr[0], arr[1])
     return solution(size(arr)-1,arr,dp)
}
```

Bottom-up DP

```
func main(arr) {
    initialize a 1-D array of null called dp[size(arr)]
    dp[0] = arr[0]
    dp[1] = max(arr[0],arr[1])
    for i in [2,size(arr)-1] {
        dp[i] = max(dp[i-2]+arr[i], dp[i-1])
    }
    return dp[size(arr)-1]
}
```

House Robber II

- Similar scenario but now the houses form a circle.

- Example: arr = [2,7,9,3,1]
- Output: 11 = 2+9

- Can we reuse the solution of House Robber?

House Robber II

Solution of House Robber

```
func main(arr) {
    initialize a 1-D array of null called dp[size(arr)]
    dp[0] = arr[0]
    dp[1] = max(arr[0],arr[1])
    for i in [2,size(arr)-1] {
        dp[i] = max(dp[i-2]+arr[i], dp[i-1])
    }
    return dp[size(arr)-1]
}
```

Solution of House Robber II

max(main(arr[1,n]), main(arr[0,n-1]))

Total Number of Palindromes

- Palindromes: a, aa, aba, abba, abcba, civic, ...
- Given a string, count all palindromic substring.
- Example:
  - abc -> 3 (a, b, c)
  - aaa -> 6 (a, a, a, aa, aa, aaa)

- Again, what are the trivial cases? Substructure?

Total Number of Palindromes

- Brute force is O(N^3):
  - Iterating all substrings is O(N^2)
  - Check if each substring is palindromic is O(N).
- We can use DP to speed up the checking part.
- Let OPT[i,j] -> bool stores whether the substring s[i,j] is palindromic.
- OPT[i,i] = True
- OPT[i,i+1] = s[i] == s[i+1]
- OPT[i,j] = OPT[i+1,j-1] and (s[i] == s[i+1])

Total Number of Palindromes

```
func main(string s) {
    initialize a 2-D array of FALSE called dp[n][n]
    for i in [0,n-1]:
        dp[i][i] = TRUE
    for i in [0,n-2]:
        dp[i][i+1] = (s[i] == s[i+1])
    for len in [3,n] {
        for i in [0,n-len-1] {
            for j in [i,i+len-1] {
                dp[i][j] = dp[i+1][j-1] && (s[i] == s[j])
            }
        }
    }
     return countTrue(dp)
}
```