[Java/C++/Python3] DP with Details Comments

lee215 ★ 47713 Last Edit: November 4, 2019 7:54 PM 2.5K VIEWS

30

Intuition

`A[i]` can be removed alone or it makes a pair.

Solution 1: Bottom up DP

Here's a link to the geeksforgeeks article, for the curious.

Java

from @g4g

```
public int minimumMoves(int[] A) {
    int N = A.length;
    //  declare dp array and initialize it with 0s
    int[][] dp = new int[N + 1][N + 1];
    for (int i = 0; i <= N; i++)
        for (int j = 0; j <= N; j++)
            dp[i][j] = 0;
    // loop for subarray length we are considering
    for (int len = 1; len <= N; len++) {
        // loop with two variables i and j, denoting starting and ending of subarray
        for (int i = 0, j = len - 1; j < N; i++, j++) {
            // If subarray length is 1, then 1 step will be needed
            if (len == 1)
                dp[i][j] = 1;
            else {
                // delete A[i] individually and assign result for subproblem (i+1,j)
                dp[i][j] = 1 + dp[i + 1][j];
                // if current and next element are same, choose min from current and subproblem (i+2,j)
                if (A[i] == A[i + 1])
                    dp[i][j] = Math.min(1 + dp[i + 2][j], dp[i][j]);
                // loop over all right elements and suppose Kth element is same as A[i] then
                // choose minimum from current and two subarray after ignoring ith and A[K]
                for (int K = i + 2; K <= j; K++)
                    if (A[i] == A[K])
                        dp[i][j] = Math.min(dp[i + 1][K - 1] + dp[K + 1][j], dp[i][j]);
            }
        }
    }
    return dp[0][N - 1];
}
```

C++

from @g4g

```
int minimumMoves(vector<int>& A) {
    int N = A.size();
    vector<vector<int>> dp(N + 1, vector<int>(N + 1))
    for (int len = 1; len <= N; len++) {
        for (int i = 0, j = len - 1; j < N; i++, j++) {
            if (len == 1)
                dp[i][j] = 1;
            else {
                dp[i][j] = 1 + dp[i + 1][j];
                if (A[i] == A[i + 1])
                    dp[i][j] = min(1 + dp[i + 2][j], dp[i][j]);
                for (int K = i + 2; K <= j; K++)
                    if (A[i] == A[K])
                        dp[i][j] = min(dp[i + 1][K - 1] + dp[K + 1][j], dp[i][j]);
```

```
}
```

Solution 2: Top down DP

Java

```
int[][] dp;
public int minimumMoves(int[] A) {
    int n = A.length;
    dp = new int[n][n];
    return dfs(0, n - 1, A);
}
int dfs(int i, int j, int[] A) {
    if (i > j) return 0;
    if (dp[i][j] > 0) return dp[i][j];
    int res = dfs(i, j - 1, A) + 1;
    if (j > 0 && A[j] == A[j - 1])
        res = Math.min(res, dfs(i, j - 2, A) + 1);
    for (int k = i; k < j - 1; ++k)
        if (A[k] == A[j])
            res = Math.min(res, dfs(i, k - 1, A) + dfs(k + 1, j - 1, A));
    dp[i][j] = res;
    return res;
}
```

C++

```
vector<vector<int>> dp;
int minimumMoves(vector<int>& A) {
    int n = A.size();
    dp = vector(n, vector<int>(n));
    return dfs(0, n - 1, A);
}
int dfs(int i, int j, vector<int>& A) {
    if (i > j) return 0;
    if (dp[i][j] > 0) return dp[i][j];
    int res = dfs(i, j - 1, A) + 1;
    if (j > 0 && A[j] == A[j - 1])
        res = min(res, dfs(i, j - 2, A) + 1);
    for (int k = i; k < j - 1; ++k)
        if (A[k] == A[j])
            res = min(res, dfs(i, k - 1, A) + dfs(k + 1, j - 1, A));
    dp[i][j] = res;
    return res;
}
```

Wrote it in 1-line but it's too long.

Python3

```
def minimumMoves(self, A):
    @lru_cache(None)
    def dp(i, j):
        if i > j: return 0
        return min(dp(i, k - 1) + (dp(k + 1, j - 1) if k < j - 1 else 1) for k in range(i, j + 1) if A[k] == A[j]
    return dp(0, len(A) - 1)
```

The whole version is that

```
from functools import lru_cache
class Solution(object):
    def minimumMoves(self, A):
        @lru_cache(None)
        def dp(i, j):
            for k in range(i, j - 1):
                if A[j] == A[k]:
                    res = min(res, dp(i, k - 1) + dp(k + 1, j - 1))
            return res
        return dp(0, len(A) - 1)
```

Comments: 5 Best Most Votes Newest to Oldest Oldest to Newest

Post

Hong_tao ★ 123 Last Edit: November 3, 2019 7:55 AM

```
for k in range(i, j - 1):
      if A[j] == A[k]:
            res = min(res, dp(i, k - 1) + dp(k + 1, j - 1) if k < j - 1 else 1)
return res
```

in this snippet, `if k < j -1 else 1` could be removed, i think

3 Show 5 replies Reply

mhelvens ★ 622 November 2, 2019 9:55 PM

Here's a link to the geeksforgeeks article, for the curious.

2 Show 1 reply Reply

jasonsun0311 ★ 15 Last Edit: December 11, 2019 2:03 AM

@lee215 Here is a slightly improved c++ implementation without global variable and "correct" scoping, based on your idea.
One could replace the Y-combinator with std::function<int(int, int)> to get the same recursive lambda. But this will induce type erasure and probably use heap space thereby yielding a slower execution.

At least two of the predicate in the else clause in the `f` could be shortened using the short if syntax but to me it reduces the readability of the code; hence I avoided it.

```
template <class F>
struct rec {
  F f;
  template <class... Args>
```

Read More

0 Reply

LVL99 ★ 148 Last Edit: November 2, 2019 11:30 PM

Java top-down memoization like your python solution

```
class Solution {
    int[][] dp;
    public int minimumMoves(int[] arr) {
        int n = arr.length;
        dp = new int[n+1][n+1];
        return solve(0,n-1,arr);
    }
    int solve(int i,int j,int[] arr){
        if(i>j) return 0;
```

Read More

0 Show 1 reply Reply

joshualian1989 ★ 421 Last Edit: November 2, 2019 10:43 PM

Since the array elements are integers from 1 to 20, there are a lot of duplicates

```
for k in range(i, j - 1):
    if A[j] == A[k]:
        res = min(res, dp(i, k - 1) + dp(k + 1, j - 1) if k < j - 1 else 1)
```

we can speed up this part by precomputing the occurrence indices for each element and do bisect, here is the code

```
from functools import lru_cache
from collections import defaultdict
```

Read More

0 Show 1 reply Reply