

Dynamic Programming | Set 17 (Palindrome Partitioning)

Given a string, a partitioning of the string is a *palindrome partitioning* if every substring of the partition is a palindrome. For example, "aba|b|bbabb|a|b|aba" is a palindrome partitioning of "ababbbabbababa". Determine the fewest cuts needed for palindrome partitioning of a given string. For example, minimum 3 cuts are needed for "ababbbabbababa". The three cuts are "a|babbbab|b|ababa". If a string is palindrome, then minimum 0 cuts are needed. If a string of length n containing all different characters, then minimum n-1 cuts are needed.

String :	ababbbabbababa
Palindrome partitioning:	a babbbab b ababa

Recommended: Please solve it on "PRACTICE" first, before moving on to the solution.

This problem is a variation of [Matrix Chain Multiplication](#) problem. If the string is palindrome, then we simply return 0. Else, like the Matrix Chain Multiplication problem, we try making cuts at all possible places, recursively calculate the cost for each cut and return the minimum value.

Let the given string be str and minPalPartion() be the function that returns the fewest cuts needed for palindrome partitioning. following is the optimal substructure property.

```
// i is the starting index and j is the ending index. i must be passed as 0 and j as n-1
minPalPartion(str, i, j) = 0 if i == j. // When string is of length 1.
minPalPartion(str, i, j) = 0 if str[i..j] is palindrome.

// If none of the above conditions is true, then minPalPartion(str, i, j) can be
// calculated recursively using the following formula.
minPalPartion(str, i, j) = Min { minPalPartion(str, i, k) + 1 +
```

```

        minPalPartion(str, k+1, j) }
    where k varies from i to j-1

```

Following is Dynamic Programming solution. It stores the solutions to subproblems in two arrays $P[][]$ and $C[][]$, and reuses the calculated values.

```

// Dynamic Programming Solution for Palindrome Partitioning Problem
#include <stdio.h>
#include <string.h>
#include <limits.h>

// A utility function to get minimum of two integers
int min (int a, int b) { return (a < b)? a : b; }

// Returns the minimum number of cuts needed to partition a string
// such that every part is a palindrome
int minPalPartion(char *str)
{
    // Get the length of the string
    int n = strlen(str);

    /* Create two arrays to build the solution in bottom up manner
       C[i][j] = Minimum number of cuts needed for palindrome partitioning
                of substring str[i..j]
       P[i][j] = true if substring str[i..j] is palindrome, else false
       Note that C[i][j] is 0 if P[i][j] is true */
    int C[n][n];
    bool P[n][n];

    int i, j, k, L; // different looping variables

    // Every substring of length 1 is a palindrome
    for (i=0; i<n; i++)
    {
        P[i][i] = true;
        C[i][i] = 0;
    }

    /* L is substring length. Build the solution in bottom up manner by
       considering all substrings of length starting from 2 to n.
       The loop structure is same as Matrix Chain Multiplication problem (
       See http://www.geeksforgeeks.org/archives/15553 )*/
    for (L=2; L<=n; L++)
    {
        // For substring of length L, set different possible starting indexes
        for (i=0; i<n-L+1; i++)
        {
            j = i+L-1; // Set ending index

            // If L is 2, then we just need to compare two characters. Else
            // need to check two corner characters and value of P[i+1][j-1]
            if (L == 2)
                P[i][j] = (str[i] == str[j]);
            else
                P[i][j] = (str[i] == str[j]) && P[i+1][j-1];

            // IF str[i..j] is palindrome, then C[i][j] is 0
            if (P[i][j] == true)
                C[i][j] = 0;
            else
            {
                // Make a cut at every possible location starting from i to j,
                // and get the minimum cost cut.
                C[i][j] = INT_MAX;
                for (k=i; k<=j-1; k++)
                    C[i][j] = min (C[i][j], C[i][k] + C[k+1][j]+1);
            }
        }
    }

    // Return the min cut value for complete string. i.e., str[0..n-1]
    return C[0][n-1];
}

```

```

}

// Driver program to test above function
int main()
{
    char str[] = "ababbbabbababa";
    printf("Min cuts needed for Palindrome Partitioning is %d",
        minPalPartion(str));
    return 0;
}

```

[Run on IDE](#)

Output:

Min cuts needed for Palindrome Partitioning is 3

Time Complexity: $O(n^3)$

An optimization to above approach

In above approach, we can calculating minimum cut while finding all palindromic substring. If we finding all palindromic substring 1st and then we calculate minimum cut, time complexity will reduce to $O(n^2)$.

Thanks for **Vivek** for suggesting this optimization.

```

// Dynamic Programming Solution for Palindrome Partitioning Problem
#include <stdio.h>
#include <string.h>
#include <limits.h>

// A utility function to get minimum of two integers
int min (int a, int b) { return (a < b)? a : b; }

// Returns the minimum number of cuts needed to partition a string
// such that every part is a palindrome
int minPalPartion(char *str)
{
    // Get the length of the string
    int n = strlen(str);

    /* Create two arrays to build the solution in bottom up manner
       C[i] = Minimum number of cuts needed for palindrome partitioning
             of substring str[0..i]
       P[i][j] = true if substring str[i..j] is palindrome, else false
       Note that C[i] is 0 if P[0][i] is true */
    int C[n];
    bool P[n][n];

    int i, j, k, L; // different looping variables

    // Every substring of length 1 is a palindrome
    for (i=0; i<n; i++)
    {
        P[i][i] = true;
    }

    /* L is substring length. Build the solution in bottom up manner by
       considering all substrings of length starting from 2 to n. */
    for (L=2; L<=n; L++)
    {
        // For substring of length L, set different possible starting indexes
        for (i=0; i<n-L+1; i++)
        {
            j = i+L-1; // Set ending index

            // If L is 2, then we just need to compare two characters. Else
            // need to check two corner characters and value of P[i+1][j-1]
            if (L == 2)
                P[i][j] = (str[i] == str[j]);

```

```
        else
            P[i][j] = (str[i] == str[j]) && P[i+1][j-1];
    }
}

for (i=0; i<n; i++)
{
    if (P[0][i] == true)
        C[i] = 0;
    else
    {
        C[i] = INT_MAX;
        for(j=0; j<i; j++)
        {
            if(P[j+1][i] == true && 1+C[j]<C[i])
                C[i]=1+C[j];
        }
    }
}

// Return the min cut value for complete string. i.e., str[0..n-1]
return C[n-1];
}

// Driver program to test above function
int main()
{
    char str[] = "ababbbabbababa";
    printf("Min cuts needed for Palindrome Partitioning is %d",
        minPalPartion(str));
    return 0;
}
```

[Run on IDE](#)

Output:

Min cuts needed for Palindrome Partitioning is 3

Time Complexity: $O(n^2)$

Asked in: **Amazon**



Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

[Dynamic Programming](#) [palindrome](#)

Recommended Posts:

[Dynamic Programming | Set 18 \(Partition problem\)](#)

[Dynamic Programming | Set 8 \(Matrix Chain Multiplication\)](#)

[Dynamic Programming | Set 19 \(Word Wrap Problem\)](#)

[Dynamic Programming | Set 12 \(Longest Palindromic Subsequence\)](#)

[Dynamic Programming | Set 16 \(Floyd Warshall Algorithm\)](#)

([Login](#) to Rate and Mark)

4.3 Average Difficulty : **4.3/5.0**
Based on **130** vote(s)

☐

Add to TODO List

☐

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

@geeksforgeeks, Some rights reserved

[Privacy Policy](#)

[Contact Us!](#)



[About Us!](#)



[Advertise with us!](#)



