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.

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 Matrx 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 localtion starting from i to
                // 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;
}
```

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 1^{st} 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;
}
```

Output:

```
Min cuts needed for Palindrome Partitioning is 3
```

Time Complexity: $O(n^2)$

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

Related Topics:

- Recursively print all sentences that can be formed from list of word lists
- Check if a given sequence of moves for a robot is circular or not
- Find the longest substring with k unique characters in a given string
- Function to find Number of customers who could not get a computer
- Find maximum depth of nested parenthesis in a string
- Find all distinct palindromic sub-strings of a given string
- Find if a given string can be represented from a substring by iterating the substring “n” times
- Suffix Tree Application 6 – Longest Palindromic Substring

Tags: Dynamic Programming

Recommend Share Sort by Newest

Join the discussion…

prashant jha • a month ago
http://uva.onlinejudge.org/ext...
my code :http://ideone.com/35vDnv
• Reply • Share ›

Goku • 4 months ago
Can anyone explain how the complexity for third method is O(n^2).I feel it's O(n^3) because for example str = "abcdef", the two inner loop will run O(n^2) times and loop for L will run O(n) times that makes it O(n^3).Can anyone please explain this?
• Reply • Share ›

Anurag Singh ➚ Goku • 4 months ago
Thanks for pointing this out. The closing bracket of 1st for loop was not in right place in optimized code. It is corrected now.
There is only one level of nesting i.e. one for loop is inside the other so complexity is O(n^2).
• Reply • Share ›

Xiaohui Liu • 5 months ago
Is there an obvious bug in the optimized approach? Should it be like this http://ideone.com/GDggYi ? Just one line of change.
• Reply • Share ›

geeksword ➚ Xiaohui Liu • 5 months ago
很明显笔误，否则还是O(n3)
• Reply • Share ›

Xiaohui Liu ➚ geeksword • 5 months ago
I was just surprised nobody pointed out this obvious typo.
1 • Reply • Share ›

Anurag Singh ➚ Xiaohui Liu • 4 months ago
Thanks. Loops were not nested correctly. The code is corrected now.
• Reply • Share ›

Rahul Patidar • 6 months ago
simple n O(n^2) solution.....

```
int minPalPartion(string s)

{ int n=s.length();

int i,j,b[n];

bool a[n][n];

for(i=0;i<n;i++) for(j="0;j&amp;lt;n;j++)" a[i][j]="false;" for(i="n-1;i">=0;i--){
b[i]=n-i-1;
for(j=i;j<n;j++) {="" if(s[i]="=s[j]&amp;amp;&amp;amp;(j-i&amp;lt;2||a[i+1][j-1]))){" a[i][j]="true;" f(j="=n-1){b[i]=0;}"
else="" if(b[i]="">b[j+1]+1)
b[i]=b[j+1]+1;
} } }

return b[0];
}
```

• Reply • Share ›

Anurag Singh ➚ Rahul Patidar • 6 months ago

If you really want others to look at code and provide feedback, please do not add code directly here. It's not readable. That's why at the end of every article, there is a message "**Writing code in comment? Please use** ideone.com **and share the link here.**"

• Reply • Share ›

Rahul Prajapati • 6 months ago

```
int minPalPartion(char *str)
{
// Get the length of the string
int n = strlen(str);
int C[n][n];

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

// Every substring of length 1 is a palindrome
for (i=0; i<n; i++)="" {="" c[i][i]="0;" }="" 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="" str[i..j]="" is="" palindrome,="" then="" c[i][j]="" is="" 0="" if="" ((l="=" 2)="" &&="" (str[i]="=" str[j]))="" c[i][j]="0;" else="" if((str[i]="=" str[j])="" &&="" !c[i+1][j-1])="" c[i][j]="0;" else="" {="" make="" a="" cut="" at="" every="" possible="" localtion="" 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];="" }="">
```

string.= i.e.,= str[0..n-1]= return= c[0][n-1],= }= >

• Reply • Share ›

A • 6 months ago

How do we actually find the cuts given our table C[i][j]

• Reply • Share ›

Guest • 8 months ago

solved in O(n^2)
http://ideone.com/5PdWjW

1 • Reply • Share ›

Sneha ➚ Guest • 6 months ago

your program is giving wrong output.
e.g. for input string "babbabab"
your output is 2
but actual output is 1 as we can cut the string bab | babab

• Reply • Share ›

Prasanth KP • 8 months ago

modified version of O(n^2) code submitted by sumit.. Thanks sumit for the code -
http://ideone.com/AIuj2P

1 • Reply • Share ›

Sorrowfull Blinger • 9 months ago

yo do we need 2 arrays P[] & C[] ....???

• Reply • Share ›

Prasanth KP ➚ Sorrowfull Blinger • 8 months ago

If use just a single array C[][],

then to determine if a particular string[i][j] is a palindrome, it would take linear time.. costly

to determine whether a particular string is palindrome in constant time, we'll be constructing another table P[][]

• Reply • Share ›

Guest • 9 months ago

--

• Reply • Share ›

Mohit Gandhi • 10 months ago

Somebody Please help, Is this code fine?

:

int FewestCut(char s[],int i,int j)

{

static int cut=0;

int max_i=-1,max_j=-1;

LongestPstring(s,i,j,&max_i,&max_j);

if(i==max_i&&j==max_j)

cut+=0;

else if(max_i==-1&&max_j==-1)

cut+=i-i;

see more

• Reply • Share ›

x • 10 months ago

int funrec(char *str,int l,int r,int n)

{

int ans[n][n];

for(int i=0;i<n;i++) {="" for(int="" j="0;j<n;j++)" {="" if(ispalindrom(str,i,j))="" ans[i][j]="0;" else="" ans[i][j]="INT_MAX;" }="" }="" if(ans[0][n-1]="=0)return" 0;="" for(int="" i="0;i<n;i++)" {="" for(int="" j="0;j<n;j++)" {="" int="" min="INT_MAX;int" t;="" if(ans[i][j]="=INT_MAX)" {="" for(int="" k="i;k<j;k++)" {="" if(ans[i][k]!="INT_MAX&ans[k+1][j]!=INT_MAX)" {="" t="1+ans[i][k]+ans[k+1][j];" if(t<min)min="t;" }="" }="" if(min!="INT_MAX)" ans[i][j]="min;" }="" }="" }="" return="" ans[l][r];="" }="" suggestion="" are="" welcome!!!="">

• Reply • Share ›

Bandi Sumanth • 10 months ago

C[i][j] = min (C[i][j], C[i][k] + C[k+1][j]+1);

C[k+1][j] is still not calculated....then why are you using it?

Isnt it a mistake..?

• Reply • Share ›

np ➚ Bandi Sumanth • 10 months ago

No its not a mistake. C[k+1][j] is already calculated if you look carefully value of k cant

be greater than j so from this code we can say that max value of k we are accessing is k+1 that is j so it will be c[j][j] which we already stored as 0 at the start itself.

Try to dry run the code you will notice we are filling value in a diagonal fashion so C[k+1][j] is already calculated

1 • Reply • Share ›

Bandi Sumanth • 10 months ago

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

How are you using P[i+1][j-1], which is still not calculated ?

Please somebody explain

• Reply • Share ›

Sorrowfull Blinger ➚ Bandi Sumanth • 9 months ago

that is basically to check the substring without last and first character ..and this would be already done because its size is 2 less than the current string.

1 • Reply • Share ›

Bandi Sumanth ➚ Sorrowfull Blinger • 8 months ago

thank you, I understood it...

• Reply • Share ›

Vivek • 10 months ago

O(n^2) solution

http://ideone.com/RXXXgk

3 • Reply • Share ›

Guest • 10 months ago

O(n^2)

for(i=0;i<n;i++) {="" if(ispalin[0][i])="" table[i]="0;" else="" {="" table[i]="INT_MAX;" for(j="0;j<i;j++)" {="" if(ispalin[j+1][i]="" &&="" 1+table[j]<table[i])="" table[i]="1+table[j];" }="" }="" }="" return="" table[n-1];="">

• Reply • Share ›

Vishal Shaw • 10 months ago

here is my code with auxillary space of o(n)... the idea is similar to choosing a center and expanding on both side to check for palindrome... time complexity is still o(n^3)
http://ideone.com/Pp2b2P

• Reply • Share ›

shine • a year ago

@GeeksforGeeks Can this problem be solved in O(n^2) time.

• Reply • Share ›

shine shine • a year ago

Please do reply..

• Reply • Share ›

AlienOnEarth shine • a year ago

No.. This problem is modified matrix chain multiplication. If that can be done in o(n^2) this problem can also be done in o(n^2)

• Reply • Share ›

Vishal Shaw AlienOnEarth • 10 months ago

check vivek soln it is o(n^2) soln...

• Reply • Share ›

Ankit Chaudhary • a year ago

I think it can be done in O(n^2).

```
bool palin[1000][1000];
void preprocess(char *str,int n){ // O(n^2)
for(int i=0;i<n;i++) palin[i][i]="true;" for(int="" l="2;L&amp;lt;n;L++){" for(int="" i="0;i+L&amp;lt;=n;i++){"
int="" j="i+L-1;" palin[i][j]="(str[i]==str[j])" &amp;&amp;="" palin[i+1][j-1];="" }="" }="" bool="" ispalin(int="
i,int="" j){return="" palin[i][j];}="" int="" partition(char="" *str,int="" low,int="" high){="
if(low="">=high || isPalin(low,high)) return 0;
if(dp[low][high]!=-1) return dp[low][high];
for(int i=low;i<=high;i++){
dp[low][high]=min(dp[low][high], partition(str,low,i) + partition(str,i+1,high)+1);
}
return dp[low][high];
}

int main(){
char str[1000];
scanf("%s",str);
int ans=partition(str);

}
```

1 • Reply • Share ›

NB • a year ago

What about this approach :

a) Find the longest common substring between the string and its reverse. This will give the longest palindrome present in the string.

b) Remove this palindrome from the original string and repeat the process

b) Remove this palindrome from the original string and repeat the process
c) base case is when string size =1 or when the entire string being looked at is a palindrome.
d) The the number of palindrome strings is x, then x -1 cuts are needed.

Eg :

String : abbacdcef Longest Pal : abba
String : cdcef Longest Pal : cdc
String : ef Longest Pal : e
String : f Longest Pal : f

No. of cuts = No of 'Longest. pal' -1 = 3

• Reply • Share ›

Mohit Bajaj NB • a year ago

very first step is wrong!!
example: 'abcdecba' reversed to 'abcedcba', LCS here is 'abc' or 'cba' and is definitely not a palindrome!

1 • Reply • Share ›

Susnata Roy • a year ago

have a look at the dp solution below... O(n^2) .. similar to idea of word break problem in gfg...

http://ideone.com/sYkYFo

Word break problem:

http://www.geeksforgeeks.org/d...

• Reply • Share ›

darshini • a year ago

how can we get the value of p[i+1][j-1] as we are currently calculating p[i][j].

• Reply • Share ›

s poonia • a year ago

this wont work for -ive numbers . try the following code. it following code. it will work for -ive and also take care if corner cases like 0 and -1 :

```
int numofone(int n){

int count=0;

while(n){

n=n&(n-1);

count++;
```

```
}

return count;

}

int mymethod(int x){
```

see more

1 • Reply • Share ›

s poonia s poonia • a year ago

wrong que... this is ans for http://www.geeksforgeeks.org/n...

• Reply • Share ›

Sumit Monga • 2 years ago

a solution in O(n^2) where we first check for a substring starting at i and ending at j is a palindrome or not . Then using this table we calculate the minimum no. of continuous palindromes upto the last character i.e. index (n-1). So, a 1D array is used so space complexity is also better .Here is the code:

```
#include<stdio.h>
#include<limits.h>
#include<string.h>

int no_of_pal_parts(char * str,int n)
{

int i,j,k,c;
bool is_pal[n][n];
memset(is_pal,0,sizeof(is_pal));
for(i = n-1; i >= 0; i--)
{
for(j = i; j < n; j++)
{
```

see more

5 • Reply • Share ›

trying • 2 years ago

a | babbbab | babab | a

I think this is the minimum cuts.

• Reply • Share ›

trying ↗ trying • 2 years ago

sorry i did not see the whole stuff.

• Reply • Share ›

equation • 2 years ago

i think there is any bug in the logic...
because for

```
char str[] = "ababbabbababa"
```

the output should be 1 as ababbabbaba|ba
but the code is giving 2.

@geeksforgeeks please correct me if m wrong.....

1 • Reply • Share ›

GeeksforGeeks ↗ equation • 2 years ago

The output 2 seems to be correct. Please note that ba is not a palindrome. If we modify the string to ababbabbaba**aa**, we get 1 because aa is a palindrome.

• Reply • Share ›

Pankaj Goyal • 2 years ago

can smone tell me what is the time complexity of recursive method? O(n*(2^n))?

• Reply • Share ›

abhishek08aug • 2 years ago

Intelligent :D

• Reply • Share ›

zyfo2 • 2 years ago

can be done in O(n^2) using dp. check leetcode for details
so basically, you don't need variable starting point. just start from 0 instead of n possible points.
cut n^3 to n^2

2 • Reply • Share ›

Amit • 2 years ago

```
 #include"stdio.h"
#include "malloc.h"
int min(int a, int b,int c)
{
        if(a<b)
        {
```

```
        {
                return a<c?a:c;
        }
        else
                return b<c?b:c;
}
int isPalindrome(char str[],int start,int end)
{
        while(start<end)
        {
                if(str[start] != str[end])
                        return 0;
                start++;
```

see more

• Reply • Share ›

Vishal Johri • 2 years ago

I thought of an alternate algo using Divide and Conquer..

Break string into 2 parts and merge them into 1 if after merging they are palindromes....

int merge(int lb1,int ub1,int lb2,int ub2) : merges only if after merging they are palindromes..if merging done=1 else =0..

void part(int lb,int ub) //send intially lb=0 ub=n-1...
{
static int merge_count=0;
mid=(lb+ub)/2;
part(lb,mid);
part(mid+1,ub);
merge_count=merge_count+merge(lb,mid,mid+1,ub);

}
merge_count counts number of mergings done..
Min number of cuts = merge_count (Think it carefully!)....

• Reply • Share ›

Vishal Johri ➚ Vishal Johri • 2 years ago

Sorry some changes...

1. Insert a special character '*' in between each character
initially to signify cuts...
At merging,remove this cut...else this cut(*) remains..

2. At the end,count the number of '*' cuts ==> x

Then, min number of cuts = x;

• Reply • Share ›

sush • 2 years ago

This implementation uses just one array for storing

```
 int minCuts(char *str)
{
        int n=strlen(str),i,j,l;
    int t[n][n];
        memset(t,-1,n*n*sizeof(int));
        for(i=0;i<n-1;++i)
        {
                t[i][i]=0;
                if(str[i]==str[i+1])
                t[i][i+1]=0;
                else
                t[i][i+1]=1;
        }
        t[i][i]=0;
        for(l=3;l<=n;++l)
        {
```

see more

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Privacy

Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago