Dynamic Programming | Set 28 (Minimum insertions to form a palindrome)

Given a string, find the minimum number of characters to be inserted to convert it to palindrome.

Before we go further, let us understand with few examples:
ab: Number of insertions required is 1. **b**ab
aa: Number of insertions required is 0. aa
abcd: Number of insertions required is 3. **dcb**abcd
abcda: Number of insertions required is 2. a**dc**bcda which is same as number of insertions in the substring bcd(Why?).
abcde: Number of insertions required is 4. **edcb**abcde

Let the input string be *str[l……h]*. The problem can be broken down into three parts:
1. Find the minimum number of insertions in the substring str[l+1,…….h].
2. Find the minimum number of insertions in the substring str[l…….h-1].
3. Find the minimum number of insertions in the substring str[l+1……h-1].

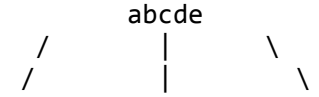
Recursive Solution
The minimum number of insertions in the string str[l…..h] can be given as:
minInsertions(str[l+1…..h-1]) if str[l] is equal to str[h]
min(minInsertions(str[l…..h-1]), minInsertions(str[l+1…..h])) + 1 otherwise

```
// A Naive recursive program to find minimum number insertions
// needed to make a string palindrome
#include <stdio.h>
#include <limits.h>
#include <string.h>

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

// Recursive function to find minimum number of insersions
int findMinInsertions(char str[], int l, int h)
{
    // Base Cases
    if (l > h) return INT_MAX;
    if (l == h) return 0;
    if (l == h - 1) return (str[l] == str[h])? 0 : 1;

    // Check if the first and last characters are same. On the basis of the
    // comparison result, decide which subrpoblem(s) to call
    return (str[l] == str[h])? findMinInsertions(str, l + 1, h - 1):
                               (min(findMinInsertions(str, l, h - 1),
                                    findMinInsertions(str, l + 1, h)) + 1);
}

// Driver program to test above functions
int main()
{
    char str[] = "geeks";
    printf("%d", findMinInsertions(str, 0, strlen(str)-1));
    return 0;
}
```

Output:

```
3
```

Dynamic Programming based Solution
If we observe the above approach carefully, we can find that it exhibits overlapping subproblems.
Suppose we want to find the minimum number of insertions in string “abcde”:



```
            bcde          abcd       bcd  <- case 3 is discarded as str[l] != str[h]
         /   |   \       /   |   \
        /    |    \     /    |    \
       cde   bcd  cd   bcd abc bc
     / | \  / | \ /|\ / | \
de cd d cd bc c…….
```

The substrings in bold show that the recursion to be terminated and the recursion tree cannot originate from there. Substring in the same color indicates overlapping subproblems.

How to reuse solutions of subproblems?
We can create a table to store results of subproblems so that they can be used directly if same subproblem is encountered again.

The below table represents the stored values for the string abcde.

```
a b c d e
----------
0 1 2 3 4
0 0 1 2 3
0 0 0 1 2
0 0 0 0 1
0 0 0 0 0
```

How to fill the table?
The table should be filled in diagonal fashion. For the string abcde, 0….4, the following should be order in which the table is filled:

```
Gap = 1:
(0, 1) (1, 2) (2, 3) (3, 4)

Gap = 2:
(0, 2) (1, 3) (2, 4)

Gap = 3:
(0, 3) (1, 4)

Gap = 4:
(0, 4)
```

```
// A Dynamic Programming based program to find minimum number
// insertions needed to make a string palindrome
#include <stdio.h>
#include <string.h>

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

// A DP function to find minimum number of insersions
int findMinInsertionsDP(char str[], int n)
{
    // Create a table of size n*n. table[i][j] will store
    // minumum number of insertions needed to convert str[i..j]
    // to a palindrome.
    int table[n][n], l, h, gap;
```

```
    // Initialize all table entries as 0
    memset(table, 0, sizeof(table));

    // Fill the table
    for (gap = 1; gap < n; ++gap)
        for (l = 0, h = gap; h < n; ++l, ++h)
            table[l][h] = (str[l] == str[h])? table[l+1][h-1] :
                          (min(table[l][h-1], table[l+1][h]) + 1);

    // Return minimum number of insertions for str[0..n-1]
    return table[0][n-1];
}

// Driver program to test above function.
int main()
{
    char str[] = "geeks";
    printf("%d", findMinInsertionsDP(str, strlen(str)));
    return 0;
}
```

Output:

```
3
```

Time complexity: O(N^2)
Auxiliary Space: O(N^2)

Another Dynamic Programming Solution (Variation of Longest Common Subsequence Problem)
The problem of finding minimum insertions can also be solved using Longest Common Subsequence (LCS) Problem. If we find out LCS of string and its reverse, we know how many maximum characters can form a palindrome. We need insert remaining characters. Following are the steps.
1) Find the length of LCS of input string and its reverse. Let the length be 'l'.
2) The minimum number insertions needed is length of input string minus 'l'.

```
// An LCS based program to find minimum number insertions needed to
// make a string palindrome
#include<stdio.h>
#include <string.h>

/* Utility function to get max of 2 integers */
int max(int a, int b)
{   return (a > b)? a : b; }

/* Returns length of LCS for X[0..m-1], Y[0..n-1].
   See http://goo.gl/bHQVP for details of this function */
int lcs( char *X, char *Y, int m, int n )
{
   int L[n+1][n+1];
   int i, j;

   /* Following steps build L[m+1][n+1] in bottom up fashion. Note
```

```
       that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
    for (i=0; i<=m; i++)
    {
      for (j=0; j<=n; j++)
      {
        if (i == 0 || j == 0)
          L[i][j] = 0;

        else if (X[i-1] == Y[j-1])
          L[i][j] = L[i-1][j-1] + 1;

        else
          L[i][j] = max(L[i-1][j], L[i][j-1]);
      }
    }

    /* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] */
    return L[m][n];
}

// LCS based function to find minimum number of insersions
int findMinInsertionsLCS(char str[], int n)
{
    // Creata another string to store reverse of 'str'
    char rev[n+1];
    strcpy(rev, str);
    strrev(rev);

    // The output is length of string minus length of lcs of
    // str and it reverse
    return (n - lcs(str, rev, n, n));
}

// Driver program to test above functions
int main()
{
     char str[] = "geeks";
     printf("%d", findMinInsertionsLCS(str, strlen(str)));
     return 0;
}
```

Output:

```
3
```

Time complexity of this method is also O(n^2) and this method also requires O(n^2) extra space.

This article is compiled by Aashish Barnwal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming

Writing code in comment? Please use **ideone.com** and share the link here.

lucy • 3 hours ago

@GeeksforGeeks i don't n know what is this long solution.
if we take an array of size [26] and store frequency of all character and count which have odd number of frequency if it is "x" then our answer is "x-1". we have also care if "x" is zero then answer will be "zero". please comment someone i am right or wrong..... ::)

shikhar jindal • a month ago

#include<stdio.h>
can anyone please provide some test cases for which this code goes wrong or is it right.

```
int main()
{
int t;
scanf("%d",&t);
while(t--)
{
char s[6101];
scanf("%s",s);
int n=strlen(s);
int i=0 ;int j=n-1 ;int k=0;
while(i<j) {="" if(s[i]="s[j])" {="" i++;="" j--;="" }="" else="" {="" if(k%2="0)" {="" j--;="" k++;="" }="" else="" {="" i++;="" k++;="" }="" }="" }="" printf("%d",k);="" }="" return="" 0;="" }="">
```

prashant jha • a month ago

http://ideone.com/psgfwD

• Reply • Share ›

Vito • 2 months ago

```
int min_insertions(char *str, int low, int high){

if(!str) {
return -1;
}

//Only one element
if(low == high) {
return 0;
}

//Two elements
if(high-low == 1) {
if(str[low] == str[high]){
return 0;
} else {
return 1;
}
}
```

see more

• Reply • Share ›

the_c0der • 4 months ago

in method 2 we should have : if(l+1 >=h-1) conditions also.. otherwise we will select dp[1][0].. and shouldn't the dp[][] initialized to INT_MAX rather than 0 ?

• Reply • Share ›

Shirley • 5 months ago

Can I just use a hashset, if the the hashset doesn't contain the character, add it, otherwise, remove it. return 0 or hs.size() - 1?

```
public class Solution {

public int minNumber (String s)

{

if (s == null || s.length() == 0)

return 0;
```

```
HashSet<character> hs = new HashSet<character>();

for (int i = 0; i < s.length(); i++)

{

if (!hs.contains(s.charAt(i)))
```

see more

• Reply • Share ›

Ashutosh Rathor • 6 months ago

what about this. time complexity O(n)

```
#include<stdio.h>

#include<string.h>

char s[26];

int main()

{

int i,si,li,cnt,n;

char c;

scanf("%d\n",&n); // string length

scanf("%s",&s);

si=0; li=n-1; cnt=0;
```

see more

• Reply • Share ›

Jon Snow • 7 months ago

@GeeksforGeeks
Below is link to space optimized code for second method which uses O(n) space. We need only current and previous row of matrix
to compute next value. So after each iteration we just swapped pointers
of both array. so current row become previous and previous row become
current.
http://ideone.com/RJnWq9

• Reply • Share ›

shashi • 8 months ago

a bit different solution -- non-DP

public String buildPalindrome(String data, int startIndex, int endIndex) {

if (startIndex == endIndex)

return data.substring(startIndex);

else if(startIndex > endIndex)return "";

if (data.charAt(startIndex) == data.charAt(endIndex)) {

return data.charAt(startIndex)

+ buildPalindrome(data, startIndex + 1, endIndex - 1)

+ data.charAt(endIndex);

} else {

StringBuffer remaining = new StringBuffer(data.substring(

startIndex, endIndex));

return remaining.toString() + data.charAt(endIndex)

+ remaining.reverse().toString();

}

}

• Reply • Share ›

dev21 • 9 months ago

DP metod of the above question
http://ideone.com/jqskXc

2 • Reply • Share ›

abhilash jaiswal • a year ago

i think in the inner for loop we have incresed the gap(i.e ++h) i think that is not correct.bcz as discussed we should fill table diagonally .

• Reply • Share ›

Ankit Mehta • a year ago

Recursion seems to be wrong. Check for this case : AANKKNA. In this first and last character are same, so you will say that optimal solution will be for string ANKKN, which is wrong. Even in the case of equality you will have to take min of i+1,j and i,j-1.

• Reply • Share ›

decade Ankit Mehta • 8 months ago

by observation AANKKNA will become palindrome in only 1 insertion at the last. similarly ANKKN will become palindrome in only 1 insertion at the last. Hence optimal solution of AANKKNA will lie in optimal solution of ANKKN .

• Reply • Share ›

ssk • 2 years ago

what the hell happened to my last comment???

• Reply • Share ›

ssk • 2 years ago

int min_ins(char str[], int len)
{
int i=0,j=len-1,count=0;
while(i<j) {="" if(str[i]="=" str[j])="" {="" i++;="" j--;="" }="" else="" {="" count++;="" i++;="" }="" }="" return="" count;="" }="" can="" anyone="" give="" an="" example="" for="" which="" this="" algo="" fails???="">

• Reply • Share ›

Natani Mayank • 2 years ago

is there any way to solve the problem using O(n) space?

1 • Reply • Share ›

H Kumar • 2 years ago

@Aashish Barnwal Can you please explain why do we need to fill the arrays in diagonal fashion?

• Reply • Share ›

Aashish H Kumar • 2 years ago

The algorithm finds the solution to smaller sub-problems first and then goes on solving larger sub-problems by building the table. Thus, all sub-strings of length 2 will be solved first(and results are stored in the table), then using the results stored to solve all sub-strings of length 3 and so on.
Please take a closer look at how the table gets filled. Do some paper work.

• Reply • Share ›

Sachchit • 2 years ago

@geeksforgeeks
why 1 is added in the min function 2nd argument ???

return (str[l] == str[h])? findMinInsertions(str, l + 1, h - 1):
(min(findMinInsertions(str, l, h - 1),findMinInsertions(str, l + 1, h)) + 1);

• Reply • Share ›

Wilber Torres ➚ Sachchit • 10 months ago

Because, In this case 1 char is added

• Reply • Share ›

GeeksforGeeks • 2 years ago

Please take a closer look. The approach will work fine. The length of LCS of AAAbcAAA and its reverse is 7.

• Reply • Share ›

Haoru HE • 2 years ago

I think the "Another Dynamic Programming Solution" is wrong, correct me if I am wrong.
Consider this string.
AAAbcAAA. You only need to insert a &#039b&#039 after &#039c&#039 to make it Pallindrome, but according to that algorithm, it says it need 8 - 3 = 5 characters.

• Reply • Share ›

NB ➚ Haoru HE • a year ago

its Subsequence , not substring we are looking at

• Reply • Share ›

You know me • 2 years ago

is the answer the difference of length of the string and
length of the longest palindrome in the string ??
Can anyone assure/prove this ??

• Reply • Share ›

ronny ➚ You know me • 2 years ago

Yes
the statement

return (n - lcs(str, rev, n, n));

in the Method 3 explains all.
n: length of the string
lcs(str, rev) : longest palindromic subsequence.

for more details on this refer : http://www.geeksforgeeks.org/d...

1 • Reply • Share ›

Devesh Batra • 2 years ago

(length of given string) - (longest pallindromic sequence in the given string).

• Reply • Share ›

Ravinth • 2 years ago

For Gap 3, the second value should be (1,4) not (0,4) as mentioned.
Gap = 3:
(0, 3) (0, 4)
should be
(0,3) (1,4)

• Reply • Share ›

GeeksforGeeks ➦ Ravinth • 2 years ago

@Ravinath: Thanks for pointing this out. We have corrected the example. Keep it up!

• Reply • Share ›

Prateek Sharma • 2 years ago

Python code with slight modification....

[sourcecode language="Python"]
import copy
def shiftList(listToBeShift,u,v):
for i in range(v-1,u-1,-1):
listToBeShift[i+1] = listToBeShift[i]

def makingStringPalindromeUsingMinimumInsertions(s):
lengthOfString = len(s)
listOfString = list(s)
anotherListOfString = copy.deepcopy(listOfString)
firstVariableOfList = anotherListOfString[0]
i = k = 0
j = l = lengthOfString -1
while i<j:
if ord(listOfString[i]) == ord(listOfString[j]):
i +=1

see more

• Reply • Share ›

Prateek Sharma • 2 years ago

Python code.....

[sourcecode language="Python"]
import copy
def shiftList(listToBeShift,u,v):
for i in range(v-1,u-1,-1):
listToBeShift[i+1] = listToBeShift[i]

```
def makingStringPalindromeUsingMinimumInsertions(s):
lengthOfString = len(s)
listOfString = list(s)
anotherListOfString = copy.deepcopy(listOfString)
firstVariableOfList = anotherListOfString[0]
i = k = 0
j = l = lengthOfString -1
while i<j:
if ord(listOfString[i]) == ord(listOfString[j]):
i +=1
```

see more

• Reply • Share ›

Programmer • 2 years ago

@geeksforgeeks

Algorithm:

1. Taking two variables i and j. i initalized to 0 and j initialized to (length of string)-1. Also declare a count variable and initialize it to 0. It will store the no of missing characters.

2. Run a loop until i and j are not equal.

3. At each iteration compare strings's ith and jth character

4. If they match increment i and decrement j. Then in next iteration loop will compare second and second last character(i+1 and j-1)

5. If they dont match just increment the count to include the missing character. Also decrement the j in while loop and then in next iteration it will compare the first and second last character (In this case i is not incremented).

6. One exception is when input is aa. Here both character match and is a valid palindrome. So a **if case is included after else in the while loop.**

see more

1 • Reply • Share ›

kartik ➚ Programmer • 2 years ago

In case of mismatch, it always decrements j. Does it work for input string "baa"?

• Reply • Share ›

programmer ➚ kartik • 2 years ago

try it yourself..here is the code

```
#include
#include
//palindrome minimum insertion function
int palprocess(char *str,int len);
int main (int argc, const char * argv[])
{

char str1[]="baa"; //put your input here

int str1len=strlen(str1);

//palindrome processing function
int result1=palprocess(str1,str1len);
printf("%d \n",result1);
return 0;
}
```

---

see more

• Reply • Share ›

kartik programmer • 2 years ago

I tried the code. It produced the output as 2, but correct output is 1. I hope you got it :)

• Reply • Share ›

programmer kartik • 2 years ago

Thanks for pointing it out...actually i just tried the above cases and it worked for them..didn't checked for baa and aab

so,i made a change.Actually i added one more loop. Then comparing both loops result and returning the minimum .now its giving 1 on **baa** and on **aab**

here is the code and its O(n)

```
 #include
#include
int min(int num1,int num2);
//palindrome minimum insertion function
int palprocess(char *str,int len);
int main (int argc, const char * argv[])
{


    char str1[]="baa";
```

see more

• Reply • Share ›

yash programmer • 5 months ago
its still not correct
for ABBC it gives the ans as 3 but the ans is 2 .
• Reply • Share ›

Yolo programmer • a year ago
I think this works! Even I had thought of the same algorithm.
• Reply • Share ›

programmer • 2 years ago
why to use dynamic programming nd recursive approach if this can be done easily as posted above by anon nd prateek sharma?

are they doing it rite?
• Reply • Share ›

GeeksforGeeks programmer • 2 years ago
@programmer: Thanks for your inputs. Could you please let us know the algorithm used in their code?
• Reply • Share ›

Prateek Sharma • 2 years ago
Python Code in o(n) time.....

[sourcecode language="Python"]
def shiftList(listOfString,u,v):
for i in range(v-1,u-1,-1):
listOfString[i+1] = listOfString[i]

def makingStringPalindromeUsingMinimumInsertions(s):
lengthOfString = len(s)
listOfString = list(s)
i =0
j = lengthOfString -1
while i!=j:
if ord(listOfString[i]) == ord(listOfString[j]):
i +=1
j -=1
else:
listOfString.append(None)

see more

• Reply • Share ›

anon • 2 years ago

O(n) solution

```
#include
#include
//palindrome minimum insertion function
int palprocess(char *str,int len);
int main (int argc, const char * argv[])
{

char str1[]="bcd";

int str1len=strlen(str1);

//palindrome processing function
int result1=palprocess(str1,str1len);
printf("%d \n",result1);
return 0;
}
```

see more

• Reply • Share ›

Suryaamsh ➚ anon • 2 years ago

This approach might not give 'least' number of insertions required.

Take an example:

abdfec...cefda

After processing 2 a's, you have seen 'b' != 'd', as per your algorithm, simply incrementing the number of insertions and decrementing j will produce huge result whereas the better step is to increment i(equivalent to adding 'b' before last 'a').

You can decide what to do using simple checks:

```
 if (a[i + 1] == a[j]) i++;
else if (a[i] == a[j-1]) j--;
else
  anything;
```

I don't think this correction produces optimum comparisons though. We are overlooking lot of scenarios here. One such scenario is drafted below

see more

• Reply • Share ›

programmer Suryaamsh • 2 years ago

try this code on **abdfec ... cefda...**
this runs loop twice nd returns minimum value...

```
#include
#include
int min(int num1,int num2);
//palindrome minimum insertion function
int palprocess(char *str,int len);
int main (int argc, const char * argv[])
{

char str1[]="baa";

int str1len=strlen(str1);

//palindrome processing function
int result1=palprocess(str1,str1len);
printf("%d \n",result1);
return 0;
```

see more

• Reply • Share ›

Amith Reddy programmer • a year ago

Above code do not work for bcdca. I am not sure whether it can be done in O(n). Let me know if it can be done.

• Reply • Share ›

Prateek Sharma • 2 years ago

Python Code.....

[sourcecode language="Python"]
/import copy
def charactersRequiredToMakeStringPalindrome(s):
strList = list(s)
copyList = copy.deepcopy(strList)
lengthOfString = len(s)
newList =[None]*(lengthOfString-1)
for i in range(lengthOfString-1):

```
for i in range(lengthOfString-1):
newList[i] = strList[-1]
strList.pop()
for i in copyList:
newList.append(i)
print "".join(newList)
def main():
string ='hgrthn'
charactersRequiredToMakeStringPalindrome(string)
if __name__ == '__main__':
main()
```

• Reply • Share ›

Felipe Viana Sousa • 2 years ago

Cool! I liked the recursive most but the others are very elegant.

• Reply • Share ›

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms

- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Follow @GeeksforGeeks

Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 2 hours ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 2 hours ago

- Andy Toh

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

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours 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