Dynamic Programming | Set 3 (Longest Increasing Subsequence)

We have discussed Overlapping Subproblems and Optimal Substructure properties in Set 1 and Set 2 respectively.

Let us discuss Longest Increasing Subsequence (LIS) problem as an example problem that can be solved using Dynamic Programming.
The longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, length of LIS for { 10, 22, 9, 33, 21, 50, 41, 60, 80 } is 6 and LIS is {10, 22, 33, 50, 60, 80}.

Optimal Substructure:
Let arr[0..n-1] be the input array and L(i) be the length of the LIS till index i such that arr[i] is part of LIS and arr[i] is the last element in LIS, then L(i) can be recursively written as.

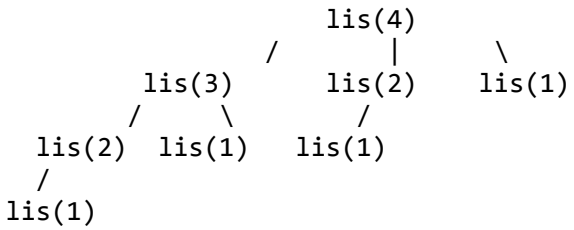
L(i) = { 1 + Max (L(j)) } where j < i and arr[j] < arr[i] and if there is no such j then L(i) = 1
To get LIS of a given array, we need to return max(L(i)) where 0 < i < n
So the LIS problem has optimal substructure property as the main problem can be solved using solutions to subproblems.

Overlapping Subproblems:
Following is simple recursive implementation of the LIS problem. The implementation simply follows the recursive structure mentioned above. The value of lis ending with every element is returned using max_ending_here. The overall lis is returned using pointer to a variable max.

```
/* A Naive recursive implementation of LIS problem */
#include<stdio.h>
#include<stdlib.h>

/* To make use of recursive calls, this function must return two things:
   1) Length of LIS ending with element arr[n-1]. We use max_ending_here
      for this purpose
   2) Overall maximum as the LIS may end with an element before arr[n-1]
      max_ref is used this purpose.
The value of LIS of full array of size n is stored in *max_ref which is our f
*/
int _lis( int arr[], int n, int *max_ref)
{
    /* Base case */
    if(n == 1)
        return 1;

    int res, max_ending_here = 1; // length of LIS ending with arr[n-1]

    /* Recursively get all LIS ending with arr[0], arr[1] ... ar[n-2]. If
       arr[i-1] is smaller than arr[n-1], and max ending with arr[n-1] needs
       to be updated, then update it */
    for(int i = 1; i < n; i++)
    {
        res = _lis(arr, i, max_ref);
        if (arr[i-1] < arr[n-1] && res + 1 > max_ending_here)
            max_ending_here = res + 1;
    }

    // Compare max_ending_here with the overall max. And update the
    // overall max if needed
    if (*max_ref < max_ending_here)
       *max_ref = max_ending_here;

    // Return length of LIS ending with arr[n-1]
    return max_ending_here;
}

// The wrapper function for _lis()
int lis(int arr[], int n)
{
    // The max variable holds the result
    int max = 1;
```

```
    // The function _lis() stores its result in max
    _lis( arr, n, &max );

    // returns max
    return max;
}

/* Driver program to test above function */
int main()
{
    int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Length of LIS is %d\n",  lis( arr, n ));
    getchar();
    return 0;
}
```

Considering the above implementation, following is recursion tree for an array of size 4. lis(n) gives us the length of LIS for arr[].



We can see that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation. Following is a tabluated implementation for the LIS problem.

```
/* Dynamic Programming implementation of LIS problem */
#include<stdio.h>
#include<stdlib.h>

/* lis() returns the length of the longest increasing subsequence in
    arr[] of size n */
int lis( int arr[], int n )
{
   int *lis, i, j, max = 0;
   lis = (int*) malloc ( sizeof( int ) * n );

   /* Initialize LIS values for all indexes */
   for ( i = 0; i < n; i++ )
      lis[i] = 1;

   /* Compute optimized LIS values in bottom up manner */
   for ( i = 1; i < n; i++ )
      for ( j = 0; j < i; j++ )
         if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)
```

```
            lis[i] = lis[j] + 1;

    /* Pick maximum of all LIS values */
    for ( i = 0; i < n; i++ )
       if ( max < lis[i] )
          max = lis[i];

    /* Free memory to avoid memory leak */
    free( lis );

    return max;
}

/* Driver program to test above function */
int main()
{
  int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
  int n = sizeof(arr)/sizeof(arr[0]);
  printf("Length of LIS is %d\n", lis( arr, n ) );

  getchar();
  return 0;
}
```

Note that the time complexity of the above Dynamic Programmig (DP) solution is O(n^2) and there is a O(nLogn) solution for the LIS problem (see this). We have not discussed the nLogn solution here as the purpose of this post is to explain Dynamic Programmig with a simple example.

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

Join the discussion…

Ashok Jangir • 17 days ago

I have a big concern: when you guys use pointers I would like to know how to use the same concept in Java, as there is nothing like pointers. :(
I couldn't understand the recursion because of the pointer.

• Reply • Share ›

Stallion • 20 days ago

I tried to solve it using Java something like this (Without Recursion) -

```
public class LongeestIncreaseSequence {

public static void main(String args[]){

int array [] = {10, 22, 9, 33, 21, 50, 41, 60, 80};

System.out.println( findLongestSequence(array));

}

static int findLongestSequence(int array[])

{

int max;

int arrayLength = array.length;

int sequenceLength = 1;
```

see more

• Reply • Share ›

MingWei Jie • a month ago

Typo. last sentence of the paragraph below recursion tree. "Following is a tabluated implementation for the LIS problem." tabluated => tabulated

• Reply • Share ›

jaideep • a month ago

```
#include <stdio.h>
int lis(int arr[], int n)
{
int max=arr[0];
int i,j,count=1;
```

```
for(i=1;i<=n;i++)
{
if(arr[i]>max)
{
max=arr[i];
count++ ;
}
}
return count;
}
int main()
{
int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60,80,90,1 };
int n = sizeof(arr)/sizeof(arr[0]);
printf("\n Length of LIS is %d\n", lis( arr, n ));
getchar();
return 0;

}
```

• Reply • Share ›

prashankmehta jaideep • 21 days ago

List: {19,1,2,3,4}
Expected length: 4
Your result: 1

• Reply • Share ›

Nikunj Mundhra prashankmehta • 19 days ago

it is coming four only according to your list values.
thank you.

• Reply • Share ›

Girish • a month ago

If the objective was to output the actual subsequence, how would one proceed?

• Reply • Share ›

sk Girish • a month ago

http://ideone.com/rHlKbn

• Reply • Share ›

Mission Peace • 2 months ago

https://www.youtube.com/watch?... Watch this video for LIS solution.

• Reply • Share ›

sunil • 2 months ago

```
int _lis( int arr[], int n, int *max_ref)
{
/* Base case */
if(n == 1)
return 1;

int res, max_ending_here = 1; // length of LIS ending with arr[n-1]

/* Recursively get all LIS ending with arr[0], arr[1] ... ar[n-2]. If
arr[i-1] is smaller than arr[n-1], and max ending with arr[n-1] needs
to be updated, then update it */
for(int i = 1; i < n; i++)
{
res = _lis(arr, i, max_ref);
if (arr[i-1] < arr[n-1] && res + 1 > max_ending_here)
max_ending_here = res + 1;
}

// Compare max_ending_here with the overall max. And update the
// overall max if needed
if (*max_ref < max_ending_here)
*max_ref = max_ending_here;

// Return length of LIS ending with arr[n-1]
return max_ending_here;
}
```

this is giving wrong output

▲ | ▼ • Reply • Share ›

thevagabond85 • 2 months ago

slight modification for method 2 :

```
int LIS(int A[], int n){
int LS[n]; // L[i] : Length of Increasing Subsequence whose last element is A[i]
int i, j;
int res =1;
LS[0] =1;
int maxim;

for(i=1; i<n; i++)="" {="" maxim="INT_MIN;" for(j="i-1;" j="">=0; j--) {
if( A[i]>A[j] ){
if(LS[j]>maxim){
maxim = LS[j];
```

```
}

}
}

LS[i] = 1 + maxim
// maintains the result , avoid seperate loop to find max
if(LS[i]>res){
res = LS[i]
}
}
return res;

}
```

• Reply • Share ›

Nobody • 3 months ago

I'm pretty sure that LIS can be solved in O(n^2) time by Brute Force approach.
Here's my code;

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
int n;
cin>>n;int c=1;vector<int>v;int k;
int a[n];
for(int i=0;i<n;i++) cin=""">>a[i];
for(int i=0;i<n;i++) {="" k="a[i];" for(int="" j="i+1;j&amp;lt;n;j++)" {="" if(a[j]="">k)
{
c++;
k=a[j];

}
}
v.push_back(c);//storing the lengths of all LIS's in a vector
c=1;
}
cout<<(*max_element(v.begin(),v.end())); //and then calculating the largest incresing
subsequence
}
```

Please help me if I'm wrong.

• Reply • Share ›

Anushkha Singh • 3 months ago

Refer following link for further understanding it better.
Longest Increasing subsequence

2 ^ | v • Reply • Share ›

NoobInHell • 3 months ago

In Recursive..

```
for(int i = 1; i < n; i++)
{
res = _lis(arr, i, max_ref);
if (arr[i-1] < arr[n-1] && res + 1 > max_ending_here)
max_ending_here = res + 1;
}
```

can be changed it to

```
for(int i = 1; i < n; i++)
{
if (arr[i-1] < arr[n-1] )){
res = _lis(arr, i, max_ref);
if(res + 1 > max_ending_here)
{
max_ending_here = res + 1;
}
}
}
```

^ | v • Reply • Share ›

Abhishek • 3 months ago

Get the detailed explanation here :

Longest Increasing subsequence

2 ^ | v • Reply • Share ›

coder12489 • 3 months ago

Posing a solution in ruby, correct me if wrong

```
class LongestIncreasingSubsequence
def initialize(args)
@arr = args
end

def find_pre(dp_arr, arr, element)
index = 0
```

```
max = -1
dp_arr.each do |a|
if max < a && arr[index] < element
max = a
end
index += 1
end
return ((max != -1) ? max : 0)
end
```

---

see more

• Reply • Share ›

venkatesham • 3 months ago

is this works for all the cases ???
it is an o(n) time

```
#include<iostream>

using namespace std;

int lis(int *a, int n,int priv)

{

if(n==0)

return 0;

else

{

if(a[n-1]<priv) {="" return="" 1+lis(a,n-1,a[n-1]);="" }="" else="" return="" lis(a,n-1,priv);="" }="" }="" int="" main()="" {="" int="" a[]="{" 88,66,77,1,2,3,4,5,60="" };="" int="" n="sizeof(a)/sizeof(int);" cout<<lis(a,n,81);="" }="">
```

• Reply • Share ›

jayasurya_j • 3 months ago

Take a loot at my solution (Recursion with memoization)

http://ideone.com/N9GHoe

• Reply • Share ›

Arseniy • 3 months ago

Perhaps I missed something, but lis(arr, n) returns LIS, ends with arr[n-1]. So I suppose, we must search maximum in lis(arr, k), where 1 =< k <= n

• Reply • Share ›

Bhuwnesh Dhakar • 3 months ago

This solution has O(n) Time complexity. Check it out
http://ideone.com/LKc4DP

• Reply • Share ›

Siya ➚ Bhuwnesh Dhakar • 3 months ago

It is giving wrong answer on your test case also.Right answer is 6 and sequence is 1,2,3,4,5,8. You are finding continuous increasing.

• Reply • Share ›

Bhuwnesh Dhakar ➚ Siya • 3 months ago

Oh.. I got the problem..Thankx for correcting this

• Reply • Share ›

prashant • 3 months ago

https://ideone.com/mlyYAb

• Reply • Share ›

Guest • 4 months ago

Solution in Java using an auxiliary array for caching. I think this is an O(n) solution. Correct me if I am wrong?

https://ideone.com/mdX8tp

• Reply • Share ›

Guest • 4 months ago

Solution in Java using an auxiliary array for caching. I think this is an O(n) solution. Correct me if I am wrong?

https://ideone.com/mdX8tp

• Reply • Share ›

tima ➚ Guest • 3 months ago

Wrong output for { 10, 11,1,2,3,4,5,3,8 }
Answer should be 5 {1,2,3,4,5}, your solution gives 6. Lot of cases wrong.

• Reply • Share ›

randomguy202 ➚ tima • 3 months ago

For input { 10, 11,1,2,3,4,5,3,8 } I think answer should be 6 {1,2,3,4,5,8}. The sub-sequence need not be continuous.

See example given in the article itself (quote from article): "For example, length

of LIS for { 10, 22, 9, 33, 21, 50, 41, 60, 80 } is 6 and LIS is {10, 22, 33, 50, 60, 80}."

• Reply • Share ›

Neelam • 4 months ago

Hi,

I tried to solve above problem using Binary Search Tree. Though it solves this problem. It is not correct. Can anybody please point out the problem in this approach. My code is:

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;

public class Solution {

class Node

{
```

see more

• Reply • Share ›

swjobs • 4 months ago

Hi, I have coded this Very simple JAVA Implementation-

https://ideone.com/Q87CuP

Is there something wrong in this code? If anybody could reply?

• Reply • Share ›

Free Coder • 4 months ago

Implementation in JAVA

```
HashMap<integer, arraylist<integer="">> result;

public List<integer> findLIS(int[] arr) {
result = new HashMap<integer, arraylist<integer="">>();

ArrayList<integer> lis = new ArrayList<integer>();
```

```
lis.add(arr[arr.length-1]);
result.put(arr.length-1, lis);

//have not handled the corner case when arr.length == 2
for(int i = arr.length-2; i >= 0 ;i--){
lis = new ArrayList<integer>();
// Construct LIS when arr[i] is included
lis.add(arr[i]);

// Find next element greater than arr[i]
```

see more

• Reply • Share ›

Baballa • 4 months ago

I like the explanations but the solutions posted are weird ugly looking. Maybe I'm noob but solution seems hideous.

2 • Reply • Share ›

Karan • 4 months ago

In the condition: arr[i] > arr[j] && lis[i] < lis[j] + 1, what is the signifance of lis[i] < lis[j] + 1? wouldn't it always be true?

• Reply • Share ›

Anurag Singh ➚ Karan • 4 months ago

lis array is initialized by 1.
After that we compare values and compute actual LIS. While this, if we find that earlier lis is less than what we see currently, we update the lis to current bigger value.
Put debug messages in code and see how lis is changing in every iteration OR run the code yourself on a paper to understand it properly.

• Reply • Share ›

Shruti • 4 months ago

Simple Explanation of LIS with Code in DP.
http://www.gohired.in/2014/12/...

1 • Reply • Share ›

Pritpal Singla • 5 months ago

Hi This one is giving also in o(n) and dont need to use extra array for this.
please let me know if i am wrong and or my code will fail for any test case please let me know

```
#include <stdio.h>

void get(int a[],int n)
```

{

int longC=0,longI=0,currentC=1,currentI=0;

int i;

for(i=1;i<n;i++) {="" if(a[i]="">a[i-1])

{

currentC++;

}

see more

• Reply • Share ›

jayandranath jaya • 5 months ago

hi this one giving O(n) please if anything is wrong

//Longest increasing subsequence

#include<iostream>

using namespace std;

void LIS(int ar[], int n)

{

int k=1;

int p =ar[0];

for(int i=1; i<n; i++)="" {="" if(p<ar[i])="" {="" p="ar[i];" ar[k]="p;" k++;="" }="" }="" int="" m="k;" for(int="" j="0;" j<m;="" j++)="" cout<<ar[j]<<"="" ";="" }="" int="" main()="" {="" int="" ar[]="{" 10,="" 22,="" 9,="" 33,="" 21,="" 50,="" 41,="" 60,="" 80="" };="" int="" n="sizeof(ar)/sizeof(ar[0]);" lis(ar,="" n);="" return="" 0;="" }="">

1 • Reply • Share ›

kar • 5 months ago

Isn't max = lis[n-1]. what is the need to traverse the array to find max?

• Reply • Share ›

Jongi ➚ kar • 4 months ago

No max is not ls[n-1]. It can be any value from ls[0] to ls[n-1], because we are looking for 'increasing' subsequence.

Like 2341
Here lis[0]=1
lis[1]=2
lis[2]=3
lis[3]=1 (Note,1 because an increasing sequence cannot be formed by including any previous numbers, all are greater than it)
So for now for n=4,
max is not lis[3], but lis[2] :-)

• Reply • Share ›

kar • 5 months ago

Can someone explain me why this is not O(n)?

```
int lis(int *a, int l){
if(l == 0) return 1;
else if(a[l] > a[l - 1]) return 1 + lis(a, l - 1);
else return lis(a, l - 1);
}
int main(){
int a[] = { 10, 22, 9, 33, 21, 50, 41, 60, 80 };
cout << lis(a, sizeof(a)/sizeof(a[0]) - 1);
}
```

1 • Reply • Share ›

Hello_world ➚ kar • 5 months ago

your algo will check only two consecutive number not whole array. Whenever you see one if prev element in array is less than current element you are increasing the counter.

• Reply • Share ›

oblix ➚ kar • 5 months ago

That algorithm is incorrect. You are just lucky that your input has lis=6 and your algorithm returns 6 as well. Add numbers 1 to 8 at the end of your sequence:
{10, 22, 9, 33, 21, 50, 41, 60, 80, 1, 2, 3, 4, 5, 6, 7, 8 }
Your program will return lis = 13 , but it is 8 (1..8) . You increment lis whenever a[i-1] < a[i].

2 • Reply • Share ›

Guest ➚ oblix • 4 months ago

Suppose we have arr[] = {7, 5 ,9 ,8}
now we have arr[n-1]=arr[3] = 8, which is greater than arr[0]
and again we have arr[1] = 5 which is smaller than arr[n-1].
the increment will be on both cases hence the LIS till iteration will be 2 but 7>5 which is not increasing. Can anyone explain.

I got it it is first checking with n as i = 1,2,3..n-1;
in _lis(arr, i, max_ref);
first i = 1 then so on.
this takes care of above problem.

HHH • 6 months ago

public static int findLIS(int[] input) {
int[] LS = new int[input.length]; LS[0] = 1;
for(int i=1; i<input.length; i++){="" int="" max="0;" for(int="" j="0;" j<i;="" j++){="" if(input[j]
<input[i])="" max="max">LS[j] ? max: LS[j];
}
LS[i] = 1 + max;
}
return maximum(LS);
}

private static int maximum(int[] input) {
int max = -10000;
for(int i=0; i<input.length; i++){="" max="max">input[i]?max:input[i];
}
return max;
}

algo1 • 6 months ago

Variant of this ,length of Longest ZigZag Sequence appeared in Amazon Campus Placement test

amine • 6 months ago

// bottom-up approach in O(n) time and O(n) space

// please let me know if this solution does not work, thanks

int longestIncreasingSubsequence(int A[], int n)

{

std::vector<int> lis(n);

lis[0] = 1;

int curr_max = A[0];

for(int i=1; i<n; ++i="")="" {="" if(="" a[i]="">= curr_max)

```
{

lis[i] = lis[i-1]+1;

curr_max = A[i];
```

see more

• Reply • Share ›

jdabck • 7 months ago

Please show how to print lis in c prog .....

• Reply • Share ›

jdabck • 7 months ago

Please show how to print lis .....

• Reply • Share ›

helper jdabck • 7 months ago

coo00OOl request...specially for you my dear friend.....
http://ideone.com/oz6nTr

4 • Reply • Share ›

rahul helper • 7 months ago

thanks for sharing .

1 • Reply • Share ›

santhoshvai • 8 months ago

My implementation in python that returns both the longest increasing subsequence and its length

http://ideone.com/ZiBqw3

• Reply • Share ›

Load more comments

- 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 · 1 hour ago

- Piyush

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

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