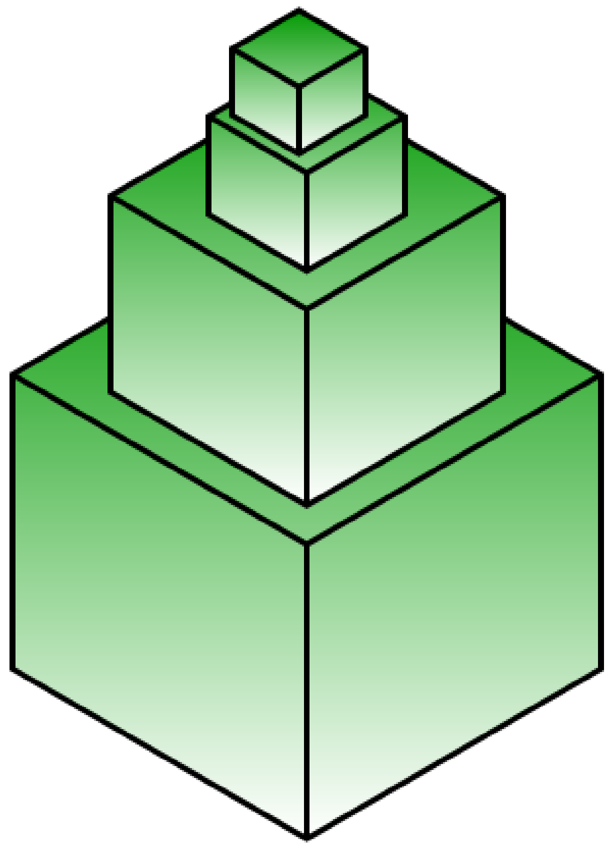
Sudo Placement 2
Quick Links for Dynamic Programming Algorithms
Recent Articles
Practice Problems
Quizzes
Videos
Basic Concepts
Tabulation vs Memoizatation
Overlapping Subproblems Property
Optimal Substructure Property
How to solve a Dynamic Programming Problem ?
Explore more...
Advanced Concepts
Bitmasking and Dynamic Programming | Set 1 (Count ways to assign unique cap to every person)
Bitmasking and Dynamic Programming | Set-2 (TSP)
Digit DP | Introduction
Explore more...
Basic Problems
Ugly Numbers & Fibonacci numbers
nth Catalan Number & Bell Numbers
Binomial Coefficient & Permutation Coefficient
Tiling Problem & Gold Mine Problem & Coin Change
Friends Pairing Problem & Subset Sum Problem
Subset Sum Problem in O(sum) space & Subset with sum divisible by m
Largest divisible pairs subset & Perfect Sum Problem
Compute nCr % p & Choice of Area
Cutting a Rod & Painting Fence Algorithm
Tiling with Dominoes
Golomb sequence
Moser-de Bruijn Sequence
Newman-Conway Sequence
maximum length Snake sequence
Print Fibonacci sequence using 2 variables
Longest Repeated Subsequence
Explore more...
Intermediate Problems
Lobb Number & Eulerian Number
Delannoy Number & Entringer Number
Rencontres Number & Jacobsthal and Jacobsthal-Lucas numbers
Super Ugly Number &Floyd Warshall Algorithm
Bellman–Ford Algorithm & 0-1 Knapsack Problem

Printing Items in 0/1 Knapsack & Unbounded Knapsack
Temple Offerings & Egg Dropping Puzzle
Dice Throw & Word Break Problem
Vertex Cover Problem & Tile Stacking Problem
Box Stacking Problem
Highway Billboard Problem & Largest Independent Set Problem
Print equal sum sets of array (Partition problem) | Set 1 & Set 2
Longest Bitonic Subsequence
Printing Longest Bitonic Subsequence
Print Longest Palindromic Subsequence
Longest palindrome subsequence with O(n) space
Explore more...
Hard Problems
Palindrome Partitioning & Word Wrap Problem
Mobile Numeric Keypad Problem & The painter's partition problem
Boolean Parenthesization Problem & Bridge and Torch problem
A Space Optimized DP solution for 0-1 Knapsack Problem
Matrix Chain Multiplication & Printing brackets in Matrix Chain Multiplication Problem
Number of palindromic paths in a matrix
Largest rectangular sub-matrix whose sum is 0
Largest rectangular sub-matrix having sum divisible by k
Maximum sum bitonic subarray
K maximum sums of overlapping contiguous sub-arrays
Maximum profit by buying and selling a share at most k times
Maximum points from top left of matrix to bottom right and return back
Check whether row or column swaps produce maximum size binary sub-matrix with all 1s
Minimum number of elements which are not part of Increasing or decreasing subsequence in array
Count ways to increase LCS length of two strings by one
Count of AP (Arithmetic Progression) Subsequences in an array
Explore more...

Box Stacking Problem | DP-22

You are given a set of n types of rectangular 3-D boxes, where the i^th box has height h(i), width w(i) and depth d(i) (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

Source: http://people.csail.mit.edu/bdean/6.046/dp/. The link also has video for explanation of solution.



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

The Box Stacking problem is a variation of LIS problem. We need to build a maximum height stack.

Following are the key points to note in the problem statement:
1) A box can be placed on top of another box only if both width and depth of the upper placed box are smaller than width and depth of the lower box respectively.
2) We can rotate boxes such that width is smaller than depth. For example, if there is a box with dimensions {1x2x3} where 1 is height, 2×3 is base, then there can be three possibilities, {1x2x3}, {2x1x3} and {3x1x2}
3) We can use multiple instances of boxes. What it means is, we can have two different rotations of a box as part of our maximum height stack.

Following is the **solution** based on DP solution of LIS problem.

1) Generate all 3 rotations of all boxes. The size of rotation array becomes 3 times the size of original array. For simplicity, we consider depth as always smaller than or equal to width.

2) Sort the above generated 3n boxes in decreasing order of base area.

3) After sorting the boxes, the problem is same as LIS with following optimal substructure property.
MSH(i) = Maximum possible Stack Height with box i at top of stack
MSH(i) = { Max (MSH(j)) + height(i) } where j < i and width(j) > width(i) and depth(j) > depth(i).
If there is no such j then MSH(i) = height(i)

4) To get overall maximum height, we return max(MSH(i)) where 0 < i < n

Following is the implementation of the above solution.

C++

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

/* Representation of a box */
struct Box
{
  // h --> height, w --> width, d --> depth
  int h, w, d;  // for simplicity of solution, always keep w <= d
};

// A utility function to get minimum of two intgers
int min (int x, int y)
{ return (x < y)? x : y; }

// A utility function to get maximum of two intgers
int max (int x, int y)
{ return (x > y)? x : y; }

/* Following function is needed for library function qsort(). We
   use qsort() to sort boxes in decreasing order of base area.
   Refer following link for help of qsort() and compare()
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare (const void *a, const void * b)
{
    return ( (*(Box *)b).d * (*(Box *)b).w ) -
           ( (*(Box *)a).d * (*(Box *)a).w );
}

/* Returns the height of the tallest stack that can be
   formed with give type of boxes */
int maxStackHeight( Box arr[], int n )
{
   /* Create an array of all rotations of given boxes
      For example, for a box {1, 2, 3}, we consider three
      instances{{1, 2, 3}, {2, 1, 3}, {3, 1, 2}} */
```

```
    Box rot[3*n];
    int index = 0;
    for (int i = 0; i < n; i++)
    {
       // Copy the original box
       rot[index].h = arr[i].h;
       rot[index].d = max(arr[i].d, arr[i].w);
       rot[index].w = min(arr[i].d, arr[i].w);
       index++;

       // First rotation of box
       rot[index].h = arr[i].w;
       rot[index].d = max(arr[i].h, arr[i].d);
       rot[index].w = min(arr[i].h, arr[i].d);
       index++;

       // Second rotation of box
       rot[index].h = arr[i].d;
       rot[index].d = max(arr[i].h, arr[i].w);
       rot[index].w = min(arr[i].h, arr[i].w);
       index++;
    }

    // Now the number of boxes is 3n
    n = 3*n;

    /* Sort the array 'rot[]' in non-increasing order
       of base area */
    qsort (rot, n, sizeof(rot[0]), compare);

    // Uncomment following two lines to print all rotations
    // for (int i = 0; i < n; i++ )
    //    printf("%d x %d x %d\n", rot[i].h, rot[i].w, rot[i].d);

    /* Initialize msh values for all indexes
       msh[i] --> Maximum possible Stack Height with box i on top */
    int msh[n];
    for (int i = 0; i < n; i++ )
       msh[i] = rot[i].h;

    /* Compute optimized msh values in bottom up manner */
    for (int i = 1; i < n; i++ )
       for (int j = 0; j < i; j++ )
          if ( rot[i].w < rot[j].w &&
               rot[i].d < rot[j].d &&
               msh[i] < msh[j] + rot[i].h
             )
          {
               msh[i] = msh[j] + rot[i].h;
          }


    /* Pick maximum of all msh values */
    int max = -1;
    for ( int i = 0; i < n; i++ )
       if ( max < msh[i] )
          max = msh[i];

    return max;
}

/* Driver program to test above function */
int main()
{
  Box arr[] = { {4, 6, 7}, {1, 2, 3}, {4, 5, 6}, {10, 12, 32} };
  int n = sizeof(arr)/sizeof(arr[0]);

  printf("The maximum possible height of stack is %d\n",
         maxStackHeight (arr, n) );

  return 0;
}
```

Run on IDE

Java

```
/* Dynamic Programming implementation
of Box Stacking problem in Java*/
import java.util.*;

public class GFG {

    /* Representation of a box */
    static class Box implements Comparable<Box>{

        // h --> height, w --> width,
        // d --> depth
        int h, w, d, area;

        // for simplicity of solution,
        // always keep w <= d

        /*Constructor to initialise object*/
        public Box(int h, int w, int d) {
            this.h = h;
            this.w = w;
            this.d = d;
        }

        /*To sort the box array on the basis
        of area in decreasing order of area */
        @Override
        public int compareTo(Box o) {
            return o.area-this.area;
        }
    }
```

```
    /* Returns the height of the tallest
    stack that can be formed with give
    type of boxes */
    static int maxStackHeight( Box arr[], int n){

        Box[] rot = new Box[n*3];

        /* New Array of boxes is created -
        considering all 3 possible rotations,
        with width always greater than equal
        to width */
        for(int i = 0;i < n;i++){
            Box box = arr[i];

            /* Orignal Box*/
            rot[3*i] = new Box(box.h, Math.max(box.w,box.d),
                                    Math.min(box.w,box.d));

            /* First rotation of box*/
            rot[3*i + 1] = new Box(box.w, Math.max(box.h,box.d),
                                       Math.min(box.h,box.d));

            /* Second rotation of box*/
            rot[3*i + 2] = new Box(box.d, Math.max(box.w,box.h),
                                       Math.min(box.w,box.h));
        }

        /* Calculating base area of
        each of the boxes.*/
        for(int i = 0; i < rot.length; i++)
            rot[i].area = rot[i].w * rot[i].d;

        /* Sorting the Boxes on the bases
        of Area in non Increasing order.*/
        Arrays.sort(rot);

        int count = 3 * n;

        /* Initialize msh values for all
        indexes
        msh[i] --> Maximum possible Stack Height
                   with box i on top */
        int[]msh = new int[count];
        for (int i = 0; i < count; i++ )
            msh[i] = rot[i].h;

        /* Computing optimized msh[]
        values in bottom up manner */
        for(int i = 0; i < count; i++){
            msh[i] = 0;
            Box box = rot[i];
            int val = 0;

            for(int j = 0; j < i; j++){
                Box prevBox = rot[j];
                if(box.w < prevBox.w && box.d < prevBox.d){
                    val = Math.max(val, msh[j]);
                }
            }
            msh[i] = val + box.h;
        }

        int max = -1;

        /* Pick maximum of all msh values */
        for(int i = 0; i < count; i++){
            max = Math.max(max, msh[i]);
        }

        return max;
    }

    /* Driver program to test above function */
    public static void main(String[] args) {

        Box[] arr = new Box[4];
        arr[0] = new Box(4, 6, 7);
        arr[1] = new Box(1, 2, 3);
        arr[2] = new Box(4, 5, 6);
        arr[3] = new Box(10, 12, 32);

        System.out.println("The maximum possible "+
                           "height of stack is " +
                           maxStackHeight(arr,4));
    }
}

// This code is contributed by Divyam
```

Run on IDE

Output:

```
The maximum possible height of stack is 60
```

In the above program, given input boxes are {4, 6, 7}, {1, 2, 3}, {4, 5, 6}, {10, 12, 32}. Following are all rotations of the boxes in decreasing order of base area.

```
   10 x 12 x 32
   12 x 10 x 32
   32 x 10 x 12
   4 x 6 x 7
```

```
4 x 5 x 6
6 x 4 x 7
5 x 4 x 6
7 x 4 x 6
6 x 4 x 5
1 x 2 x 3
2 x 1 x 3
3 x 1 x 2
```

The height 60 is obtained by boxes { {**3**, 1, 2}, {**1**, 2, 3}, {**6**, 4, 5}, {**4**, 5, 6}, {**4**, 6, 7}, {**32**, 10, 12}, {**10**, 12, 32}}

Time Complexity: O(n^2)
Auxiliary Space: O(n)

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

Practice Tags : Microsoft Amazon Codenation Dynamic Programming

Article Tags : Dynamic Programming Amazon Codenation Microsoft

Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Variations of LIS | DP-21
Program for Fibonacci numbers
Longest Increasing Subsequence | DP-3
Bellman–Ford Algorithm | DP-23
Minimum number of jumps to reach end
Paytm Interview experience for FTE (On-Campus)
Balanced expressions such that given positions have opening brackets | Set 2
Number of ways a convex polygon of n+2 sides can split into triangles by connecting vertices
Alternate Fibonacci Numbers

Find the largest area rectangular sub-matrix whose sum is equal to k

Logged in as **Dragan Nikolic**(Logout)

3.8 Average Difficulty : **3.8/5.0**
Based on **169** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Give Feedback Add your Notes Improve Article

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

Load Comments

Share this post!

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org