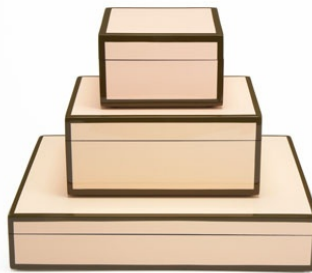


Dynamic Programming | Set 22 (Box Stacking Problem)

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. For example, if there is a box with dimensions $\{1 \times 2 \times 3\}$ where 1 is height, 2×3 is base, then there can be three possibilities, $\{1 \times 2 \times 3\}$, $\{2 \times 1 \times 3\}$ and $\{3 \times 1 \times 2\}$.
- 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

$MSH(i) = \{ \text{Max} (MSH(j)) + \text{height}(i) \}$ where $j < i$ and $\text{width}(j) > \text{width}(i)$ and $\text{depth}(j) > \text{depth}(i)$.

If there is no such j then $MSH(i) = \text{height}(i)$

4) To get overall maximum height, we return $\text{max}(MSH(i))$ where $0 < i < n$

Following is C++ implementation of the above solution.

```
/* 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 integers
int min (int x, int y)
{ return (x < y)? x : y; }

// A utility function to get maximum of two integers
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] = arr[i];
        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 decreasing order, using library
       function for quick sort */
    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](#)

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: $O(n)$

Asked in: Amazon, Codenation, Microsoft

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

Recommended Posts:

Partition a set into two subsets such that the difference of subset sums is minimum

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

3.7 Average Difficulty : **3.7/5.0**
Based on **79** 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

Contact Us!

About Us!

Advertise with us!

Privacy Policy



